Neural Networks 78 (2016) 65–74

Contents lists available at ScienceDirect

Neural Networks

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

2016 Special Issue

A decentralized training algorithm for Echo State Networks in distributed big data applications

Simone Scardapane [a,*], Dianhui Wang [b], Massimo Panella [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 Information Technology, La Trobe University, Melbourne, VIC 3086, Australia

ARTICLE INFO

Article history:
Available online 18 August 2015

Keywords:
Recurrent neural network
Echo State Network
Distributed learning
Alternating Direction Method of Multipliers
Big data

ABSTRACT

The current big data deluge requires innovative solutions for performing efficient inference on large, heterogeneous amounts of information. Apart from the known challenges deriving from high volume and velocity, real-world big data applications may impose additional technological constraints, including the need for a fully decentralized training architecture. While several alternatives exist for training feed-forward neural networks in such a distributed setting, less attention has been devoted to the case of decentralized training of recurrent neural networks (RNNs). In this paper, we propose such an algorithm for a class of RNNs known as Echo State Networks. The algorithm is based on the well-known Alternating Direction Method of Multipliers optimization procedure. It is formulated only in terms of local exchanges between neighboring agents, without reliance on a coordinating node. Additionally, it does not require the communication of training patterns, which is a crucial component in realistic big data implementations. Experimental results on large scale artificial datasets show that it compares favorably with a fully centralized implementation, in terms of speed, efficiency and generalization accuracy.

1. Introduction

With 2.5 quintillion bytes of data generated every day, we are undoubtedly in an era of 'big data' (Wu, Zhu, Wu, & Ding, 2014). Amidst the challenges put forth to the machine learning community by this big data deluge, much effort has been devoted to efficiently analyze large amounts of data by exploiting parallel and concurrent infrastructures (Cevher, Becker, & Schmidt, 2014; Chu, Kim, Lin, & Yu, 2007), and to take advantage of its possibly structured nature (Bakir, 2007). In multiple real world applications, however, the main issue is given by the overall decentralized nature of the data. In what we refer to as 'data-distributed learning' (Scardapane, Wang, Panella, & Uncini, 2015a), training data is not available on a centralized location, but large amounts of it are distributed throughout a network of interconnected agents (e.g. computers in a peer-to-peer (P2P) network). Practically, a solution relying on a centralized controller may be technologically unsuitable, since it can introduce a single point of failure, and it is prone to communication bottlenecks. Additionally, training data may not be allowed to be exchanged throughout the nodes,[1] either for its size (as is typical in big data applications), or because particular privacy concerns are present (Verykios et al., 2004). Hence, the agents must agree on a single learned model (such as a specific neural network's topology and weights) by relying only on their data and on local communication between them. In the words of Wu et al. (2014), this can informally be understood as '*a number of blind men [...] trying to size up a giant elephant*', where the giant elephant refers to the big data, and the blind men are the agents in the network. Although this analogy referred in general to data mining with big data, it is a fitting metaphor for the data-distributed learning setting considered in this paper, which is graphically depicted in Fig. 1.

With respect to neural-like architectures, several decentralized training algorithms have been investigated in the last few years. This includes distributed strategies for training standard multilayer perceptrons with back-propagation (Georgopoulos & Hasler, 2014), support vector machines (Forero, Cano, & Giannakis,

* 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).

http://dx.doi.org/10.1016/j.neunet.2015.07.006
0893-6080/

[1] In the rest of the paper, we use the terms 'agent' and 'node' interchangeably, to refer to a single component of a network as the one in Fig. 1.

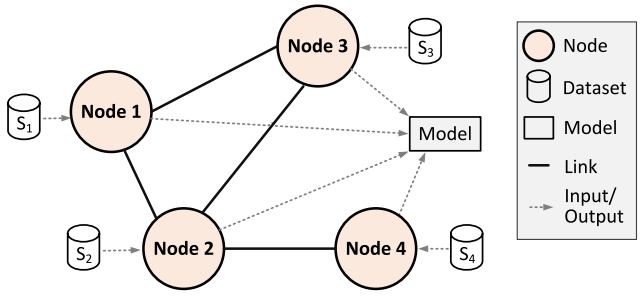


Fig. 1. Decentralized inference in a network of agents: training data is distributed throughout the nodes, all of which must converge to a single model. For readability, we assume undirected connections between agents.

2010; Navia-Vázquez, Gutierrez-Gonzalez, Parrado-Hernández, & Navarro-Abellan, 2006), deep networks (Dean et al., 2012) and random vector functional-link nets (Scardapane et al., 2015a). These are explored more in depth in Section 2.1. Additionally, decentralized inference has attracted interest from other research communities, including Wireless Sensor Networks (WSNs) (Predd, Kulkarni, & Poor, 2007), cellular neural networks (Luitel & Venayagamoorthy, 2012), distributed optimization (Boyd, Parikh, Chu, Peleato, & Eckstein, 2011; Cevher et al., 2014), distributed online learning (Zinkevich, Weimer, Li, & Smola, 2010), and others.

Much less attention has been paid to the topic of distributed learning with *recurrent* neural networks (RNNs). In fact, despite numerous recent advances (Hermans & Schrauwen, 2013; Martens & Sutskever, 2011; Monner & Reggia, 2012; Sak, Senior, & Beaufays, 2014; Sutskever, Vinyals, & Le, 2014), RNN training remains a daunting task even in the centralized case, mostly due to the well-known problems of the exploding and vanishing gradients (Pascanu, Mikolov, & Bengio, 2013). A decentralized training algorithm for RNNs, however, would be an invaluable tool in multiple large scale real world applications, including time-series prediction on WSNs (Predd et al., 2007), and multimedia classification over P2P networks (Scardapane, Wang, Panella, & Uncini, 2015b).

To summarize, in this paper we are concerned with big data scenarios defined by three characteristics: (i) time-varying (i.e., dynamic); (ii) distributed in nature, without the possibility of moving the data across the network; and (iii) large in volume. Clearly, the actual definition of this last point will depend on the specific application, e.g. large volumes in a WSN context have a completely different scale with respect to large volumes in a cluster of computers. Nonetheless, in this paper we investigate this problem by considering datasets that are 1–2 order of magnitudes greater with respect to previous works on related subjects.

To bridge the gap between distributed learning and RNNs, we propose here a data-distributed training algorithm for a class of RNNs known as Echo State Networks (ESNs) (Jaeger, 2002; Lukoševičius & Jaeger, 2009; Verstraeten, Schrauwen, d'Haene, & Stroobandt, 2007). The main idea of ESNs is to separate the recurrent part of the network (the so-called 'reservoir'), from the non-recurrent part (the 'readout'). The reservoir is typically fixed in advance, by randomly assigning its connections, and the learning problem is reduced to a standard linear regression over the weights of the readout. Due to this, ESNs do not required complex back-propagation algorithms over the recurrent portion of the network (Campolucci, Uncini, Piazza, & Rao, 1999; Pearlmutter, 1995), thus avoiding the problems of the exploding and vanishing gradients. ESNs have been applied successfully to a wide range of domains, including chaotic time-series prediction (Jaeger & Haas, 2004; Li, Han, & Wang, 2012), grammatical inference (Tong, Bickett, Christiansen, & Cottrell, 2007), stock price prediction (Lin, Yang, & Song, 2009), speech recognition (Skowronski & Harris, 2007) and acoustic modeling (Triefenbach, Jalalvand, Demuynck, & Martens, 2013), between others. While several researchers have investigated the possibility of spatially distributing the reservoir (Obst, 2014; Shutin & Kubin, 2008; Vandoorne, Dambre, Verstraeten, Schrauwen, & Bienstman, 2011), as depicted in Section 2.2, to the best of our knowledge, no algorithm has been proposed to train an ESN in the data-distributed case.

The algorithm that we propose is a straightforward application of the well-known Alternating Direction Method of Multipliers (ADMM) optimization technique, which is used to optimize a global loss function defined from all the local datasets. Communication between different agents is restricted to the computation of global averages starting from local vectors. This operation can be implemented efficiently in the decentralized setting with the use of the 'decentralized average consensus' (DAC) procedure (Barbarossa, Sardellitti, & Di Lorenzo, 2013; Olfati-Saber, Fax, & Murray, 2007). The resulting algorithm is formulated considering only local exchanges between neighboring nodes, thus making it applicable in networks where no centralized authority is present. Moreover, there is no need of exchanging training patterns which, as we stated previously, is a critical aspect in big data applications.

Our proposed algorithm is evaluated over several known benchmarks for ESNs applications, related to short-term and mid-term chaotic series prediction and non-linear system identification, where ESNs are known to achieve state-of-the-art results. As we stated before, to simulate a large scale application, we consider training datasets which are roughly 1–2 orders of magnitude larger than previous works. Experimental results show that the ADMM-based ESN trained in a decentralized fashion performs favorably with respect to a fully centralized approach, regarding speed, accuracy, and generalization capabilities.

The rest of the paper is organized as follows. In Section 2 we analyze related works on decentralized learning for feed-forward neural networks and support vector machines. Additionally, we detail previous research on spatially distributing the reservoir. In Section 3 we present the theoretical frameworks necessary to formulate our algorithm, i.e. the standard ESN framework with a least-square solution in Section 3.1, the DAC procedure in Section 3.2, and the ADMM optimization algorithm in Section 3.3. Them, we formalize the distributed training problem for ESNs in Section 4, and provide an ADMM-based algorithm for its solution. This is the main contribution of the present paper. Sections 5 and 6 detail multiple experiments on large scale artificial datasets. Finally, we conclude the paper in Section 7, by pointing out the current limitations of our approach, and possible future lines of research.

Notation

In the rest of the paper, vectors are denoted by boldface lowercase letters, e.g. $\mathbf{a}$, while matrices are denoted by boldface uppercase letters, e.g. $\mathbf{A}$. All vectors are assumed to be column vectors unless otherwise specified. The operator $\|\cdot\|_2$ is the standard L_2 norm on an Euclidean space. Finally, the notation $a[n]$ is used to denote dependence with respect to a time-instant, both for time-varying signals (in which case n refers to a time-instant) and for elements in an iterative procedure (in which case n is the iteration's index).

2. Related work

2.1. Data-distributed learning for feed-forward models

As stated in the previous section, several works over the last few years have explored the idea of fully decentralized training for feed-forward neural networks and support vector machines (SVMs). We present here a brief overview of a selection of these

works, which are related to the algorithm discussed in this paper.

In the context of SVMs, one of the earliest steps in this direction was the Distributed Semiparametric Support Vector Machine (DSSVM) presented in Navia-Vázquez et al. (2006). In the DSSVM, every local node selects a number of centroids from the training data. These centroids are then shared with the other nodes, and their corresponding weights are updated locally based on an iterated reweighted least squares (IRWLS) procedure. The DSSVM may be suboptimal, and it requires incremental passing of the support vectors (SVs), or centroids, between the nodes, which in turn requires the computation of a Hamiltonian cycle between them. An alternative Distributed Parallel SVM (DPSVM) is presented in Lu, Roychowdhury, and Vandenberghe (2008). Differently from the DSSVM, the DPSVM is guaranteed to reach the global optimal solution of the centralized SVM in a finite number of steps. Moreover, it considers general strongly connected networks, with only exchanges of SVs between neighboring nodes. Still, the need of exchanging the set of SVs, reduces the capability of the algorithm to scale to very large networks. A third approach is presented in Forero et al. (2010), where the problem is recast as multiple convex subproblems at every node, and solved with the use of the ADMM procedure (introduced in Section 3.3). ADMM is widely employed for deriving decentralized training algorithms in the case of convex optimization problems, such as the SVM and the algorithm presented in this paper. Other important applications include decentralized sparse linear regression (i.e. LASSO) (Mateos, Bazerque, & Giannakis, 2010) and its variations, such as group LASSO (Boyd et al., 2011).

A simpler approach, not deriving from the distributed optimization literature, is instead the 'consensus-based learning' (CBL) introduced in Georgopoulos and Hasler (2014). CBL allows to transform any iterative learning procedure (defined in the centralized case) into a general distributed algorithm. This is achieved by interleaving the local update steps at every node with global averaging steps over the network, with the DAC protocol (detailed in Section 3.2). In Georgopoulos and Hasler (2014) it is shown that, if the local update rule is contractive, the distributed algorithm converges to a unique classifier. The CBL idea is applied to the distributed training of a multilayer perceptron, with results comparable to that of a centralized classifier. In Scardapane et al. (2015a), the authors compare a DAC-based and an ADMM-based training algorithms for a class of neural networks known as Random Vector Functional-Links (RVFLs). The DAC-based algorithm, albeit heuristic, was found to be competitive with respect to the ADMM-based one in terms of generalization accuracy, with an extremely reduced training time. The connections between (Scardapane et al., 2015a) and the present paper are further explored in Section 4. An extension of the DAC-based algorithm for RVFLs, inspired from the CBL framework, is instead presented in Scardapane et al. (2015b), with an application to several music classification benchmarks.

A few additional works from the field of signal processing are also worth citing. Distributed learning using linear models and time-varying signals has been investigated extensively in the context of diffusion filtering (Cattivelli, Lopes, & Sayed, 2008; Di Lorenzo & Sayed, 2013). Clearly, linear models are not capable of efficiently capturing complex non-linear dynamics, which are common in WSNs and multimedia applications. Due to this, some algorithms have been proposed for distributed filtering using kernel-based methods, which are briefly reviewed in Predd et al. (2007); Predd, Kulkarni, and Poor (2009). However, these suffer from the same problems relative to SVMs, i.e., their models grow linearly with respect to the size of *all* the local datasets. See also Honeine, Richard, Bermudez, and Snoussi (2008) for further analyses on this point.

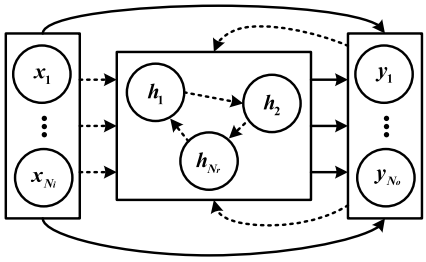


Fig. 2. Schematic depiction of a ESN. Random connections are shown with dashed lines, while trainable connections are shown with solid lines.

2.2. Spatially distributed echo state networks

Albeit no work has been done on data-distributed ESNs, several researchers have explored the possibility of spatially distributing the reservoir. The first work in this sense was the Echo State Wireless Sensor Network (ES-WSN) (Shutin & Kubin, 2008). In an ES-WSN, the WSN acts as a reservoir, where each sensor represents a neuron. As shown in the paper, distributed learning with ES-WSN is possible if the network is strongly connected. A slightly more general framework, with multiple neurons at every sensor and different inputs and output vectors, is the spatially organized distributed ESN (SODESN) (Obst, 2014). The SODESN was specifically introduced to solve the fault detection problem in WSNs.

A related line of research has investigated the possibility of implementing a fully parallel reservoir using specific hardware, with a centralized readout. Examples of this are reservoirs realized from coherent semiconductor optical amplifiers (Vandoorne et al., 2011) and passive silicon photonics chips (Vandoorne et al., 2014).

3. Preliminaries

In this section, we present the basic theoretical results needed to formulate our data-distributed ESN. We start by detailing ESN theory in Section 3.1. Then, we describe the DAC procedure in Section 3.2. Finally, we introduce the ADMM algorithm in Section 3.3.

3.1. Echo state networks

An ESN is a recurrent neural network which can be partitioned in three components, as shown in Fig. 2.

The N_i-dimensional input vector $\mathbf{x}[n] \in \mathbb{R}^{N_i}$ is fed to an N_r-dimensional reservoir, whose internal state $\mathbf{h}[n-1] \in \mathbb{R}^{N_r}$ is updated according to the state equation:

$$\mathbf{h}[n] = f_{\text{res}}(\mathbf{W}_i^r \mathbf{x}[n] + \mathbf{W}_r^r \mathbf{h}[n-1] + \mathbf{W}_o^r \mathbf{y}[n-1]), \tag{1}$$

where $\mathbf{W}_i^r \in \mathbb{R}^{N_r \times N_i}$, $\mathbf{W}_r^r \in \mathbb{R}^{N_r \times N_r}$ and $\mathbf{W}_o^r \in \mathbb{R}^{N_r \times N_o}$ are randomly generated matrices, $f_{\text{res}}(\cdot)$ is a suitably defined non-linear function, and $\mathbf{y}[n-1] \in \mathbb{R}^{N_o}$ is the previous N_o-dimensional output of the network. To increase stability, it is possible to add a small uniform noise term to the state update, before computing the non-linear transformation $f_{\text{res}}(\cdot)$ (Jaeger, 2002). Then, the current output is computed according to:

$$\mathbf{y}[n] = f_{\text{out}}(\mathbf{W}_i^o \mathbf{x}[n] + \mathbf{W}_r^o \mathbf{h}[n]), \tag{2}$$

where $\mathbf{W}_i^o \in \mathbb{R}^{N_o \times N_i}$, $\mathbf{W}_r^o \in \mathbb{R}^{N_o \times N_r}$ are adapted based on the training data, and $f_{\text{out}}(\cdot)$ is an invertible non-linear function. For simplicity, in the rest of the paper we will consider the case of one-dimensional output, i.e. $N_o = 1$, but everything we say extends straightforwardly to the case $N_o > 1$.

To be of use in any learning application, the reservoir must satisfy the so-called 'echo state property' (ESP) (Lukoševičius &

Jaeger, 2009). Informally, this means that the effect of a given input on the state of the reservoir must vanish in a finite number of time-instants. A widely used rule-of-thumb that works well in most situations is to rescale the matrix $\mathbf{W}_r^r$ to have $\rho(\mathbf{W}_r^r) < 1$, where $\rho(\cdot)$ denotes the spectral radius operator.[2] For simplicity, we adopt this heuristic strategy in this paper, but we refer the readers to Yildiz, Jaeger, and Kiebel (2012); Zhang, Miller, and Wang (2012) for recent theoretical results on this aspect.

To train the ESN, suppose we are provided with a sequence of Q desired input–outputs pairs $(\mathbf{x}[1], d[1]) \dots, (\mathbf{x}[Q], d[Q])$. The sequence of inputs is fed to the reservoir, giving a sequence of internal states $\mathbf{h}[1], \dots, \mathbf{h}[Q]$ (this is known as 'state harvesting' or 'state gathering'). During this phase, since the output of the ESN is not available for feedback, the desired output is used instead in Eq. (1) (so-called 'teacher forcing'). Define the hidden matrix $\mathbf{H}$ and output vector $\mathbf{d}$ as:

$$\mathbf{H} = \begin{bmatrix} \mathbf{x}^T[1] & \mathbf{h}^T[1] \\ \vdots & \\ \mathbf{x}^T[1] & \mathbf{h}^T[Q] \end{bmatrix} \tag{3}$$

$$\mathbf{d} = \begin{bmatrix} f_{\text{out}}^{-1}(d[1]) \\ \vdots \\ f_{\text{out}}^{-1}(d[Q]) \end{bmatrix}. \tag{4}$$

The optimal output weight vector is then given by solving the following regularized least-square problem:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^{N_i+N_r}} \frac{1}{2}\|\mathbf{H}\mathbf{w} - \mathbf{d}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2, \tag{5}$$

where $\mathbf{w} = \left[\mathbf{W}_i^o\, \mathbf{W}_r^o\right]^T$ and $\lambda \in \mathbb{R}^+$ is a positive scalar known as *regularization factor*.[3] Solution of problem in Eq. (5) can be obtained in closed form as:

$$\mathbf{w}^* = \left(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}\right)^{-1}\mathbf{H}^T\mathbf{d}. \tag{6}$$

More in general, we are provided with a training set S of multiple desired sequences. In this case, we can simply stack the resulting hidden matrices and output vectors, and solve Eq. (5). Additionally, we note that in practice we can remove the initial D elements (denoted as 'dropout' elements) from each sequence when solving the least-square problem, with D specified *a-priori*, due to their transient state. See Jaeger (2002) for a discussion of ESN training in the case of online learning.

3.2. Decentralized average consensus

Consider a network of L interconnected agents, as the one in Fig. 1, whose connectivity is known *a-priori* and is fixed. We can fully describe the connectivity between the nodes in the form of an $L \times L$ connectivity matrix $\mathbf{C}$, where $C_{ij} \neq 0$ if and only if nodes i and j are connected. In this paper, we assume that the network is connected (i.e., every node can be reached from another node with a finite number of steps), and undirected (i.e., $\mathbf{C}$ is symmetric). Additionally, suppose that every node has a measurement vector denoted by $\boldsymbol{\theta}_k[0]$, $k = 1 \dots L$.

DAC is an iterative network protocol to compute the global average with respect to the local measurement vectors, requiring only local communications between them (Barbarossa et al., 2013; Georgopoulos & Hasler, 2014; Olfati-Saber et al., 2007). Its simplicity makes it suitable for implementation even in the most basic networks, such as robot swarms (Georgopoulos & Hasler, 2014). At a generic iteration n, the local DAC update is given by:

$$\boldsymbol{\theta}_k[n] = \sum_{j=1}^{L} C_{kj}\boldsymbol{\theta}_j[n-1]. \tag{7}$$

If the weights of the connectivity matrix $\mathbf{C}$ are chosen appropriately, this recursive procedure converges to the global average given by Olfati-Saber et al. (2007):

$$\lim_{n\to+\infty} \boldsymbol{\theta}_k[n] = \frac{1}{L}\sum_{k=1}^{L}\boldsymbol{\theta}_k[0], \quad \forall k \in \{1, 2, \dots, L\}. \tag{8}$$

Practically, the procedure can be stopped after a certain predefined number of iterations is reached, or when the norm of the update is smaller than a certain user-defined threshold δ (see Scardapane et al., 2015a). In the case of undirected, connected networks, a simple way of ensuring convergence is given by choosing the so-called 'max-degree' weights (Olfati-Saber et al., 2007):

$$C_{kj} = \begin{cases} \dfrac{1}{d+1} & \text{if } k \text{ is connected to } j \\ 1 - \dfrac{d_k}{d+1} & \text{if } k = j \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

where d_k is the degree of node k, and d is the maximum degree of the network.[4] In practice, many variations on this standard procedure can be implemented to increase the convergence rate, such as the 'definite consensus' (Georgopoulos & Hasler, 2014), or the strategy introduced in Sardellitti, Giona, and Barbarossa (2010).

3.3. Alternating direction method of multipliers

The final theoretical tool needed in the following section is the ADMM optimization procedure (Boyd et al., 2011). ADMM solves optimization problems of the form:

$$\begin{aligned} &\underset{\mathbf{s}\in\mathbb{R}^d, \mathbf{z}\in\mathbb{R}^l}{\text{minimize}} \;\; f(\mathbf{s}) + g(\mathbf{z}) \\ &\text{subject to} \;\; \mathbf{A}\mathbf{s} + \mathbf{B}\mathbf{z} + \mathbf{c} = 0, \end{aligned} \tag{10}$$

where $\mathbf{A} \in \mathbb{R}^{p\times d}$, $\mathbf{B} \in \mathbb{R}^{p\times l}$ and $\mathbf{c} \in \mathbb{R}^p$. To solve this, consider the augmented Lagrangian given by:

$$\begin{aligned} \mathcal{L}_\rho(\mathbf{s}, \mathbf{z}, \mathbf{t}) &= f(\mathbf{x}) + g(\mathbf{z}) + \mathbf{t}^T(\mathbf{A}\mathbf{s} + \mathbf{B}\mathbf{z} + \mathbf{c}) \\ &\quad + \frac{\rho}{2}\|\mathbf{A}\mathbf{s} + \mathbf{B}\mathbf{z} + \mathbf{c}\|_2^2, \end{aligned} \tag{11}$$

where $\mathbf{t} \in \mathbb{R}^p$ is the vector of Lagrange multipliers, ρ is a scalar, and the last term is added to ensure differentiability and convergence (Boyd et al., 2011). The optimum of problem in Eq. (10) is obtained by iterating the following updates:

$$\mathbf{s}[n+1] = \arg\min_{\mathbf{s}} \left\{\mathcal{L}_\rho(\mathbf{s}, \mathbf{z}[n], \mathbf{t}[n])\right\} \tag{12}$$

$$\mathbf{z}[n+1] = \arg\min_{\mathbf{z}} \left\{\mathcal{L}_\rho(\mathbf{s}[n+1], \mathbf{z}, \mathbf{t}[n])\right\} \tag{13}$$

$$\mathbf{t}[n+1] = \mathbf{t}[n] + \rho\left(\mathbf{A}\mathbf{s}[n+1] + \mathbf{B}\mathbf{z}[n+1] + \mathbf{c}\right). \tag{14}$$

Some results on the asymptotic convergence of ADMM can be found in Boyd et al. (2011, Section 3.2). In particular, convergence

[2] The spectral radius of a generic matrix $\mathbf{A}$ is $\rho(\mathbf{A}) = \max_i \{|\lambda_i(\mathbf{A})|\}$, where $\lambda_i(\mathbf{A})$ is the ith eigenvector of $\mathbf{A}$.

[3] Since we consider one dimensional outputs, $\mathbf{W}_i^o$ and $\mathbf{W}_r^o$ are now row vectors, of dimensionality N_i and N_r respectively.

[4] The degree of a node is the cardinality of the set of its direct neighbors.

can be tracked by computing the so-called primal and dual residuals:

$$\mathbf{r}[n] = \mathbf{A}\mathbf{s}[n] + \mathbf{B}\mathbf{z}[n] + \mathbf{c}, \tag{15}$$

$$\mathbf{s}[n] = \rho \mathbf{A}^T \mathbf{B} \left(\mathbf{z}[n] - \mathbf{z}[n-1]\right). \tag{16}$$

ADMM can be stopped after a maximum number of iterations, or when both residuals are lower than two pre-specified thresholds:

$$\begin{cases} \|\mathbf{r}[n]\|_2 < \epsilon_{\text{primal}} \\ \|\mathbf{s}[n]\|_2 < \epsilon_{\text{dual}}. \end{cases}$$

Threshold can be chosen based on the combination of a relative and an absolute termination criteria (Boyd et al., 2011):

$$\epsilon_{\text{primal}} = \sqrt{p}\epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\left\{\|\mathbf{A}\mathbf{s}[n]\|_2, \|\mathbf{B}\mathbf{z}[n]\|_2, \|\mathbf{c}\|_2\right\} \tag{17}$$

$$\epsilon_{\text{dual}} = \sqrt{d}\epsilon_{\text{abs}} + \epsilon_{\text{rel}} \left\|\mathbf{A}^T \mathbf{t}[n]\right\|_2 \tag{18}$$

where ϵ_{abs} and ϵ_{rel} are user-specified tolerances, typically in the range 10^{-3}–10^{-5}.

4. Data-distributed ESN

In this section, we introduce our decentralized training algorithm for ESNs. Suppose that the training dataset S is distributed among a network of nodes, as the one described in Section 3.2. As a prototypical example, multiple sensors in a WSNs can collect different measurements of the same underlying time-series to be predicted. Additionally, suppose that the nodes have agreed on a particular choice of the fixed matrices $\mathbf{W}_i^r$, $\mathbf{W}_r^r$ and $\mathbf{W}_o^r$.[5] Denote by $\mathbf{H}_k$ and $\mathbf{d}_k$ the hidden matrices and output vectors, computed at the kth node according to Eqs. (3)–(4) with its local dataset. In this case, extending Eq. (5), the global optimization problem can be stated as:

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathbb{R}^{N_i+N_r}}{\arg\min} \frac{1}{2}\left(\sum_{k=1}^{L} \|\mathbf{H}_k\mathbf{w} - \mathbf{d}_k\|_2^2\right) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2. \tag{19}$$

In theory, distributed least-square problems as the one in Eq. (19) can be solved in a decentralized fashion with two sequential DAC steps, the first of which on the matrices $\mathbf{H}_k^T\mathbf{H}_k$, and the second on the vectors $\mathbf{H}_k^T\mathbf{d}_k$ (Xiao, Boyd, & Lall, 2005). In our case, however, this scheme is impractical to implement due to the typical large size of the reservoir, making the communication of the matrices $\mathbf{H}_k^T\mathbf{H}_k$ a probable network bottleneck.[6] More in general, the problem in Eq. (19) can be solved efficiently by distributed optimization routines, such as ADMM (Boyd et al., 2011). The idea is to reformulate the problem by introducing local variables $\mathbf{w}_k$ for every node, and forcing them to be equal at convergence:

$$\underset{\mathbf{z},\mathbf{w}_1,\ldots,\mathbf{w}_L\in\mathbb{R}^{N_i+N_r}}{\text{minimize}} \ \frac{1}{2}\left(\sum_{k=1}^{L} \|\mathbf{H}_k\mathbf{w}_k - \mathbf{d}_k\|_2^2\right) + \frac{\lambda}{2}\|\mathbf{z}\|_2^2 \tag{20}$$

$$\text{subject to} \qquad \mathbf{w}_k = \mathbf{z}, \quad k = 1\ldots L. \tag{21}$$

The augmented Lagrangian of problem in Eq. (21) is given by:

$$\mathcal{L}_\rho(\cdot) = \frac{1}{2}\left(\sum_{k=1}^{L} \|\mathbf{H}_k\mathbf{w}_k - \mathbf{d}_k\|_2^2\right) + \frac{\lambda}{2}\|\mathbf{z}\|_2^2 + \sum_{k=1}^{L} \mathbf{t}_k^T(\mathbf{w}_k - \mathbf{z}) + \frac{\gamma}{2}\sum_{k=1}^{L}\|\mathbf{w}_k - \mathbf{z}\|_2^2. \tag{22}$$

[5] There are several strategies to this end. In simple networks, this choice can be pre-implemented on every node. More generally, a single node can be chosen via a leader election protocol, assign the matrices, and broadcast them to the rest of the network.

[6] A similar issue is explored in Scardapane et al. (2015a, Remark 1).

Algorithm 1: Local training algorithm for ADMM-based ESN at kth node

Inputs: Training set S_k (local), size of reservoir N_r (global), regularization factors λ, γ (global), maximum number of iterations T (global)
Output: Optimal output weight vector $\mathbf{w}^*$

```
Assign matrices W_i^r, W_r^r and W_o^r, in agreement with the other agents in the network.
Gather the hidden matrix H_k and teacher signal d_k from S_k.
Initialize t_k[0] = 0, z[0] = 0.
for n from 0 to T do
    Compute w_k[n+1] according to Eq. (23).
    Compute averages ŵ and t̂ by means of the DAC procedure (see Section 3.2).
    Compute z[n+1] according to Eq. (24).
    Compute t_k[n+1] according to Eq. (25).
    Compute residuals according to Eqs. (27) and (28), and check termination criterion.
end for
return z[n]
```

Straightforward computations show that the updates for $\mathbf{w}_k[n+1]$ and $\mathbf{z}[n+1]$ can be computed in closed form as:

$$\mathbf{w}_k[n+1] = \left(\mathbf{H}_k^T\mathbf{H}_k + \gamma\mathbf{I}\right)^{-1}\left(\mathbf{H}_k^T\mathbf{d}_k - \mathbf{t}_k[n] + \gamma\mathbf{z}[n]\right), \tag{23}$$

$$\mathbf{z}[n+1] = \frac{\gamma\hat{\mathbf{w}} + \hat{\mathbf{t}}}{\lambda/L + \gamma}, \tag{24}$$

where we introduced the averages $\hat{\mathbf{w}} = \frac{1}{L}\sum_{k=1}^{L}\mathbf{w}_k[n+1]$ 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 Section 3.2. Eq. (14) instead simplifies to:

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

Eqs. (23) and (25) can be computed locally at every node. Hence, the overall algorithm can be implemented in a purely decentralized fashion, where communication is restricted to the use of the DAC protocol. In cases where, on a node, the number of training samples is lower than $N_r + N_i$, we can exploit the matrix inversion lemma to obtain a more convenient matrix inversion step (Mateos et al., 2010):

$$\left(\mathbf{H}_k^T\mathbf{H}_k + \gamma\mathbf{I}\right)^{-1} = \gamma^{-1}\left[\mathbf{I} - \mathbf{H}_k^T\left(\gamma\mathbf{I} + \mathbf{H}_k\mathbf{H}_k^T\right)\mathbf{H}_k\right]. \tag{26}$$

With respect to the training complexity, the matrix inversion and the term $\mathbf{H}_k^T\mathbf{d}_k$ in Eq. (23) can be precomputed at the beginning and stored into memory. Additionally, in this case the residuals in Eqs. (15)–(16) are given by:

$$\mathbf{r}_k[n] = \mathbf{w}_k[n] - \mathbf{z}[n], \tag{27}$$

$$\mathbf{s}[n] = -\gamma\left(\mathbf{z}[n] - \mathbf{z}[n-1]\right). \tag{28}$$

The pseudocode for the algorithm at a local node is provided in Algorithm 1.

Remark 1. ESNs admit a particular class of feed-forward neural networks, called Random Vector Functional-Links (RVFLs) (Igelnik & Pao, 1995; Pao & Takefuji, 1992), as a degenerate case. In particular, this corresponds to the case where $\mathbf{W}_r^r$ and $\mathbf{W}_o^r$ are equal to the zero matrix. An ADMM-based training algorithm for RVFLs was introduced in Scardapane et al. (2015a). Thus, the algorithm presented in this paper can be seen as a generalization of that work to the case of a dynamic reservoir.

Remark 2. A large number of techniques have been developed to increase the generalization capability of ESNs without increasing

its computational complexity (Lukoševičius & Jaeger, 2009). Provided that the optimization problem in Eq. (5) remains unchanged, and the topology of the ESN is not modified during the learning process, many of them can be applied straightforwardly to the distributed training case with the algorithm presented in this paper. Examples of techniques that can be used in this context include lateral inhibition (Xue, Yang, & Haykin, 2007), spiking neurons (Schliebs, Mohemmed, & Kasabov, 2011) and random projections (Butcher, Verstraeten, Schrauwen, Day, & Haycock, 2013). Conversely, techniques that cannot be straightforwardly applied include intrinsic plasticity (Steil, 2007) and reservoir's pruning (Scardapane, Nocco, Comminiello, Scarpiniti, & Uncini, 2014).

5. Experimental setup

In this section we describe our experimental setup. MATLAB code to repeat the experiments is available on the web under BSD license.[7] Simulations were performed on MATLAB R2013a, on a 64 bit operative system, using an Intel® Core™ i5-3330 CPU with 3 GHz and 16 GB of RAM.

5.1. Description of the datasets

We validate the proposed ADMM-based ESN on four standard artificial benchmarks applications, related to non-linear system identification and chaotic time-series prediction. These are tasks where ESNs are known to perform at least as good as the state of the art (Lukoševičius & Jaeger, 2009). Additionally, they are common in distributed scenarios. For comparisons between standard ESNs and other techniques, which go outside the scope of the present paper, we refer the reader to Lukoševičius and Jaeger (2009) and references therein. To simulate a large scale analysis, we consider datasets that are approximately 1–2 orders of magnitude larger than previous works. In particular, for every dataset we generate 50 sequences of 2000 elements each, starting from different initial conditions, summing up to 100.000 samples for every experiment. This is roughly the limit at which a centralized solution is amenable for comparison. Below we provide a brief description of the four datasets.

The NARMA-10 dataset (denoted by N10) is a non-linear system identification task, where the input $x[n]$ to the system is white noise in the interval $[0, 0.5]$, while the output $d[n]$ is computed from the recurrence equation (Jaeger, 2002):

$$d[n] = 0.1 + 0.3d[n-1] + 0.05d[n-1] \times \prod_{i=1}^{10} d[n-i] + 1.5x[n]x[n-9]. \tag{29}$$

The output is then squashed to the interval $[-1, +1]$ by the non-linear transformation:

$$d[n] = \tanh(d[n] - \hat{d}), \tag{30}$$

where $\hat{d}$ is the empirical mean computed from the overall output vector.

The second dataset is the extended polynomial (denoted by EXTPOLY) introduced in Butcher et al. (2013). The input is given by white noise in the interval $[-1 \ +1]$, while the output is computed as:

$$d[n] = \sum_{i=0}^{p} \sum_{j=0}^{p-i} a_{ij} x^i[n] x^j[n-l], \tag{31}$$

where $p, l \in \mathbb{R}$ are user-defined parameters controlling the memory and non-linearity of the polynomial, while the coefficients a_{ij} are randomly assigned from the same distribution as the input data. In our experiments, we use a mild level of memory and non-linearity by setting $p = l = 7$. The output is normalized using Eq. (30).

The third dataset is the prediction of the well-known Mackey–Glass chaotic time-series (denoted as MKG). This is defined in continuous time by the differential equation:

$$\dot{x}[n] = \beta x[n] + \frac{\alpha x[n-\tau]}{1 + x^{\gamma}[n-\tau]}. \tag{32}$$

We use the common assignment $\alpha = 0.2$, $\beta = -0.1$, $\gamma = 10$, giving rise to a chaotic behavior for $\tau > 16.8$. In particular, in our experiments we set $\tau = 30$. Time-series in Eq. (32) is integrated with a 4th order Runge–Kutta method using a time step of 0.1, and then sampled every 10 time-instants. The task is a 10-step ahead prediction task, i.e.:

$$d[n] = x[n + 10]. \tag{33}$$

The fourth dataset is another chaotic time-series prediction task, this time on the Lorenz attractor. This is a 3-dimensional time-series, defined in continuous time by the following set of differential equations:

$$\begin{cases} \dot{x}_1[n] = \sigma\,(x_2[n] - x_1[n]) \\ \dot{x}_2[n] = x_1[n]\,(\eta - x_3[n]) - x_2[n] \\ \dot{x}_3[n] = x_1[n]x_2[n] - \zeta x_3[n], \end{cases} \tag{34}$$

where the standard choice for chaotic behavior is $\sigma = 10$, $\eta = 28$ and $\zeta = 8/3$. The model in Eq. (34) is integrated using an ODE45 solver, and sampled every second. For this task, the input to the system is given by the vector $\left[x_1[n] \ x_2[n] \ x_3[n]\right]$, while the required output is a 1-step ahead prediction of the x_1 component, i.e.:

$$d[n] = x_1[n + 1]. \tag{35}$$

For all four datasets, we supplement the original input with an additional constant unitary input, as is standard practice in ESNs' implementations (Lukoševičius & Jaeger, 2009).

5.2. Description of the algorithms

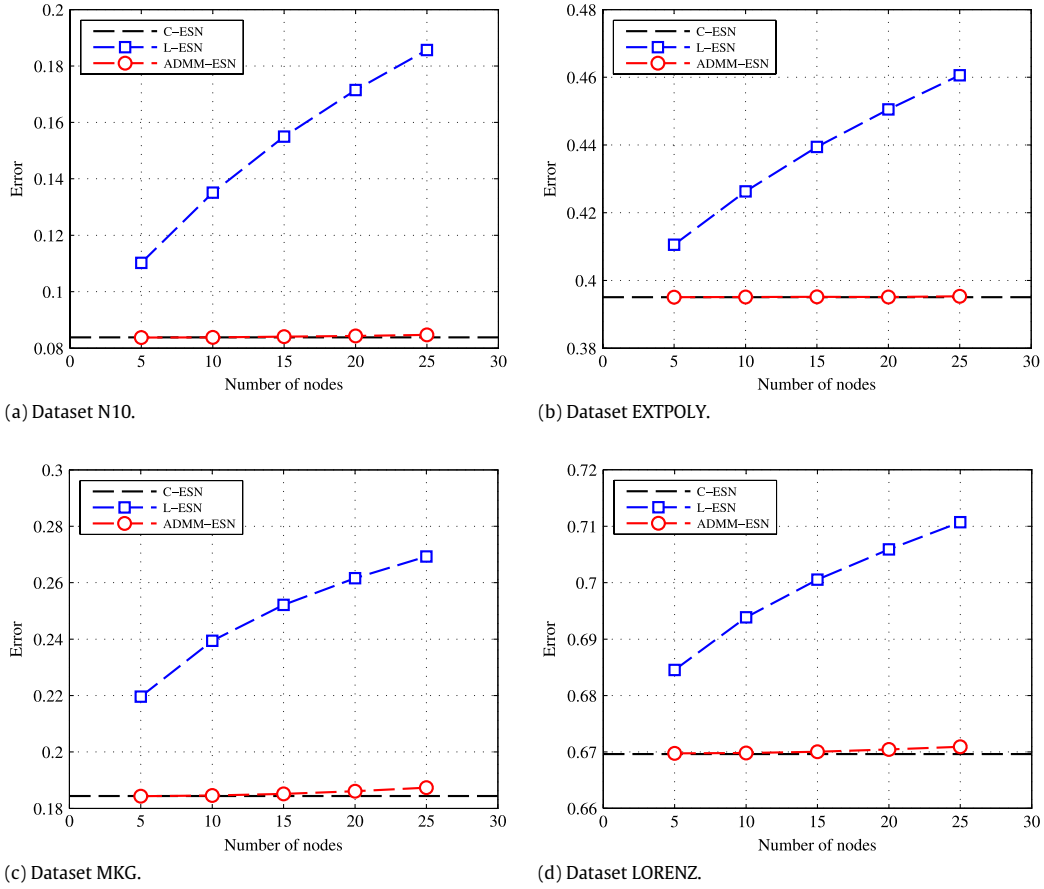
In our simulations we generate a network of agents, as the one of Fig. 1, using a random topology model for the connectivity matrix, where each pair of nodes can be connected with 25% probability. The only global requirement is that the overall network is connected. We experiment with a number of nodes going from 5 to 25, by steps of 5. To estimate the testing error, we perform a 3-fold cross-validation on the 50 original sequences. For every fold, the training sequences are evenly distributed across the nodes, and the following three algorithms are compared:

Centralized ESN (C-ESN): This simulates the case where training data is collected on a centralized location, and the net is trained by directly solving problem in Eq. (19).

Local ESN (L-ESN): In this case, each node trains a local ESN starting from its data, but no communication is performed. The testing error is then averaged throughout the L nodes.

ADMM-based ESN (ADMM-ESN): This is an ESN trained with the distributed protocol introduced in Section 4. We set $\rho = 0.01$, a maximum number of 400 iterations, and $\epsilon_{\text{abs}} = \epsilon_{\text{rel}} = 10^{-4}$.

[7] https://bitbucket.org/ispamm/distributed-esn [last visited on 2015-07-06].



(a) Dataset N10.

(b) Dataset EXTPOLY.

(c) Dataset MKG.

(d) Dataset LORENZ.

Fig. 3. Evolution of the testing error (defined as the NRMSE), for networks going from 5 agents to 25 agents. Performance of L-ESN is averaged across the nodes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

All algorithms share the same ESN architecture, which is detailed in the following section. The 3-fold cross-validation procedure is repeated 15 times by varying the ESN initialization and the data partitioning, and the errors for every iteration and every fold are collected. To compute the error, we run the trained ESN on the test sequences, and gather the predicted outputs $\tilde{y}_1, \ldots, \tilde{y}_K$, where K is the number of testing samples after removing the dropout elements from the test sequences. Then, we compute the Normalized Root Mean-Squared Error (NRMSE), defined as:

$$\text{NRMSE} = \sqrt{\frac{\sum_{i=1}^{K} \left[\tilde{y}_i - d_i\right]^2}{|K|\hat{\sigma}_d}}, \tag{36}$$

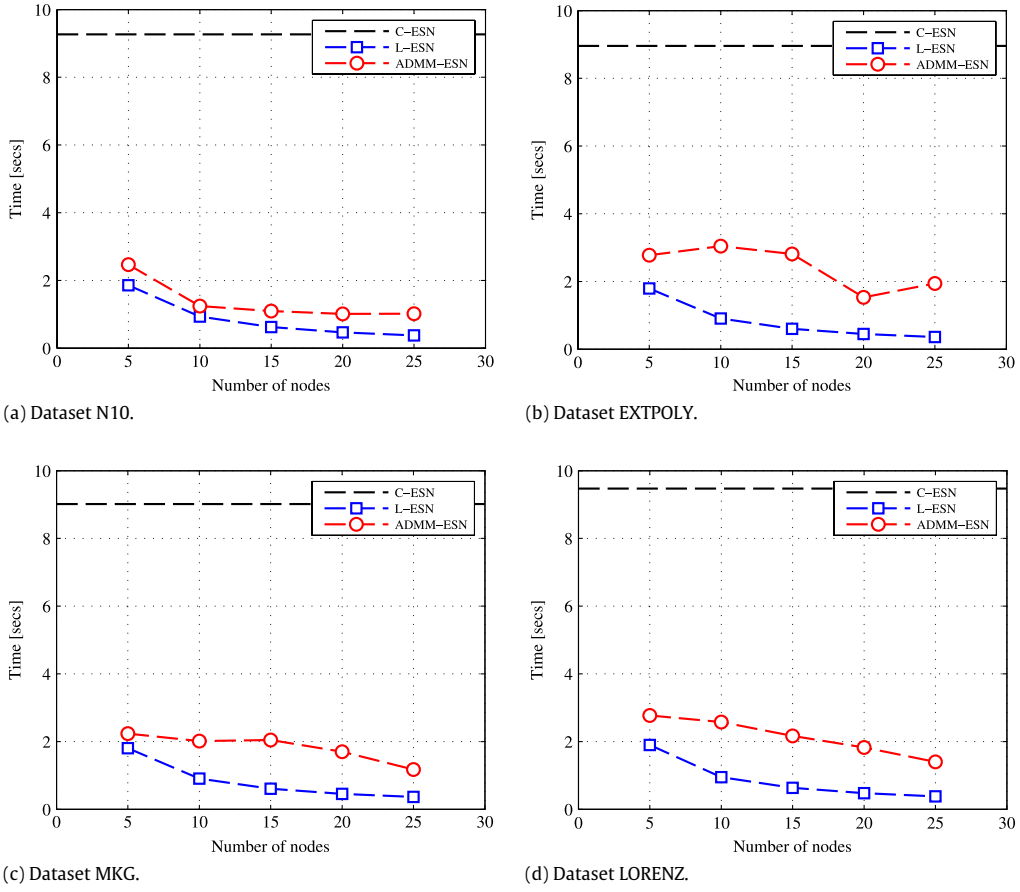
where $\hat{\sigma}_d$ is an empirical estimate of the variance of the true output samples $d_1, \ldots, d_K$. We note that, in the experiments, the network is artificially simulated on a single machine. A complete analysis of the time required to train ADMM-ESN would require knowledge of the overhead introduced by the realistic network channel, however, this goes outside the scope of the current paper.

5.3. ESN architecture

As stated previously, all algorithms share the same ESN architecture. In this section we provide a brief overview on the selection of its parameters. Firstly, we choose a default reservoir's size of $N_r = 300$, which was found to work well in all situations. Secondly, since the datasets are artificial and noiseless, we set a small regularization factor $\lambda = 10^{-3}$. Four other parameters are instead selected based on a grid search procedure. The validation error for the grid-search procedure is computed by performing a 3-fold cross-validation over 9 sequences, which are generated independently from the training and testing set. Each validation sequence has length 2000. In particular, we select the following parameters:

- The matrix $\mathbf{W}_i^r$, connecting the input to the reservoir, is initialized as a full matrix, with entries assigned from the uniform distribution $[-\alpha_\text{i}\ \alpha_\text{i}]$. The optimal parameter α_i is searched in the set $\{0.1, 0.3, \ldots, 0.9\}$.
- Similarly, the matrix $\mathbf{W}_o^r$, connecting the output to the reservoir, is initialized as a full matrix, with entries assigned from the uniform distribution $[-\alpha_\text{f}\ \alpha_\text{f}]$. The parameter α_f is searched in the set $\{0, 0.1, 0.3, \ldots, 0.9\}$. We allow $\alpha_\text{f} = 0$ for the case where no output feedback is needed.
- The internal reservoir matrix $\mathbf{W}_r^r$ is initialized from the uniform distribution $[-1\ +1]$. Then, on average 75% of its connections are set to 0, to encourage sparseness. Informally, a sparse internal matrix creates 'clusters' of heterogeneous features from its input (Jaeger, 2002; Scardapane et al., 2014). Finally, to preserve stability, the matrix is rescaled so as to have a desired spectral radius ρ, which is searched in the same interval as α_i (see the discussion on the spectral radius in Section 3.1).
- We use $\tanh(\cdot)$ non-linearities in the reservoir, while a scaled identity $f(s) = \alpha_\text{t} s$ as the output function. The parameter α_t is searched in the same interval as α_i.

Additionally, we insert uniform noise in the state update of the reservoir, sampled uniformly in the interval $\left[0, 10^{-3}\right]$, and we discard $D = 100$ initial elements from each sequence.



(a) Dataset N10.

(b) Dataset EXTPOLY.

(c) Dataset MKG.

(d) Dataset LORENZ.

Fig. 4. Evolution of the training time, for networks going from 5 agents to 25 agents. Time of L-ESN is averaged across the nodes.

Table 1
Optimal parameters found by the grid-search procedure. For a description of the parameters, see Section 5.2.

Dataset	ρ	α_i	α_t	α_f	N_r	λ
N10	0.9	0.5	0.1	0.3	300	2^{-3}
EXTPOLY	0.7	0.5	0.1	0		
MKG	0.9	0.3	0.5	0		
LORENZ	0.1	0.9	0.1	0		

Table 2
Final misclassification error and training time for C-ESN, provided as a reference, together with one standard deviation.

Dataset	NRMSE	Time (s)
N10	0.08 ± 0.01	9.26 ± 0.20
EXTPOLY	0.39 ± 0.01	8.96 ± 0.19
MKG	0.18 ± 0.03	9.02 ± 0.15
LORENZ	0.67 ± 0.01	9.47 ± 0.14

6. Experimental results

The final settings resulting from the grid-search procedure are listed in Table 1. It can be seen that, except for the LORENZ dataset, there is a tendency towards selecting large values of ρ. Output feedback is needed only for the N10 dataset, while it is found unnecessary in the other three datasets. The optimal input scaling α_f is ranging in the interval [0.5, 0.9], while the optimal teacher scaling α_t is small in the majority of cases.

The average NRMSE and training times (in seconds) for C-ESN are provided in Table 2 as a reference. Clearly, NRMSE and training time for C-ESN do not depend on the size of the agents' network, and they can be used as an upper baseline for the results of the distributed algorithms. Since we are considering the same amount of training data for each dataset, and the same reservoir's size, the training times in Table 2 are roughly similar, except for the LORENZ dataset, which has 4 inputs compared to the other three datasets (considering also the unitary input). As we stated earlier, performance of C-ESN are competitive with the state-of-the-art for all the four datasets. Moreover, we can see that it is extremely efficient to train, taking approximately 9 s in all cases.

To study the behavior of the decentralized procedures when training data is distributed, we plot the average error for the three algorithms, when varying the number of nodes in the network, in Fig. 3(a)–(d). The average NRMSE of C-ESN is shown as dashed black line, while the errors of L-ESN and ADMM-ESN are shown with blue squares and red circles respectively. Clearly, L-ESN is performing worse than C-ESN, due to its partial view on the training data. For small networks of 5 nodes, this gap may not be particularly pronounced. This goes from a 3% worse performance on the LORENZ dataset, up to a 37% decrease in performance for the N10 dataset (going from an NRMSE of 0.08 to an NRMSE of 0.11). The gap is instead substantial for large networks of up to 25 nodes. For example, the error of L-ESN is more than twice that of C-ESN for the N10 dataset, and its performance is 50% worse in the MKG dataset. Albeit these results are expected, they are evidence of the need for a decentralized training protocol for ESNs, able to take into account all the local datasets.

As is clear from Fig. 3, ADMM-ESN is able to perfectly track the performance of the centralized solution in all situations. A small gap in performance is present for the two predictions tasks when considering large networks. In particular, the performance of ADMM-ESN is roughly 1% worse than C-ESN for networks of 25 nodes in the datasets MKG and LORENZ. In theory, this gap can be reduced by considering additional iterations for the

ADMM procedure, although this would be impractical in real world applications.

Training time requested by the three algorithms is shown in Fig. 4(a)–(d). The training time for L-ESN and ADMM-ESN is averaged throughout the agents. Since the computational time of training an ESN is mostly related to the matrix inversion in Eq. (6), training time is monotonically decreasing in L-ESN with respect to the number of nodes in the network (the higher the number of agents, the lower the amount of data at every local node). Fig. 4 shows that the computational overhead requested by the ADMM procedure is limited. In the best case, the N10 dataset with 10 nodes, it required only 0.3 s more than L-ESN, as shown from Fig. 4(a). In the worst setting, the EXTPOLY dataset with 15 nodes, it required 2.2 s more, as shown from Fig. 4(b). In all settings, the time requested by ADMM-ESN is significantly lower compared to the training time of its centralized counterpart, showing it usefulness in large scale applications. Once again, it should be stressed that this analysis does not take into consideration the communication overhead between agents, which depends strictly on the actual network technology. With respect to this point, we note that the DAC procedure has been extensively analyzed over a large number of realistic networks (Barbarossa et al., 2013), showing competitive performance in most situations.

7. Conclusions

In this paper we have introduced a decentralized algorithm for training an ESN, in the case where data is distributed throughout a network of interconnected nodes. The proposed algorithm demonstrated good potential to resolve some benchmark problems where datasets are large and stored in a decentralized manner. It has multiple real-world applications in big data scenarios, particularly for large scale prediction over WSNs, or for decentralized classification of multimedia data. It is a direct application of the ADMM procedure, which we employ in our algorithm because of the following two reasons: (i) communication between nodes is restricted to local exchanges for the computation of an average vector, without reliance on a centralized controller; (ii) there is no need for the nodes to communicate training data between them, which is crucial in big data scenarios. Experimental results on multiple benchmarks, related to non-linear system identification and chaotic time-series prediction, demonstrated that it is able to efficiently track a purely centralized solution, while at the same time imposing a small computational overhead in terms of vector–matrix operations requested to the single node. Communication overhead, instead, is given by the iterative application of a DAC protocol. This represents a first step towards the development of data-distributed strategies for general RNNs, which would represent invaluable tools in real world applications. Future lines of research involve considering different optimization procedures with respect to ADMM, or more flexible DAC procedures. Additionally, although in this paper we have focused on batch learning, we envision to develop online strategies inspired to the CBL framework.

References

Bakir, G. (2007). *Predicting structured data*. MIT press.

Barbarossa, S., Sardellitti, S., & Di Lorenzo, P. (2013). Distributed detection and estimation in wireless sensor networks. In R. Chellapa, & S. Theodoridis (Eds.), *E-Reference signal processing* (pp. 329–408). Elsevier.

Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, *3*(1), 1–122.

Butcher, J. B., Verstraeten, D., Schrauwen, B., Day, C. R., & Haycock, P. W. (2013). Reservoir computing and extreme learning machines for non-linear time-series data analysis. *Neural Networks*, *38*, 76–89.

Campolucci, P., Uncini, A., Piazza, F., & Rao, B. D. (1999). On-line learning algorithms for locally recurrent neural networks. *IEEE Transactions on Neural Networks*, *10*(2), 253–271.

Cattivelli, F. S., Lopes, C. G., & Sayed, A. H. (2008). Diffusion recursive least-squares for distributed estimation over adaptive networks. *IEEE Transactions on Signal Processing*, *56*(5), 1865–1877.

Cevher, V., Becker, S., & Schmidt, M. (2014). Convex optimization for big data: scalable, randomized, and parallel algorithms for big data analytics. *IEEE Signal Processing Magazine*, *31*(5), 32–43.

Chu, C.-T., Kim, S. K., Lin, Y. A., & Yu, Y. Y. (2007). Map-reduce for machine learning on multicore. In *Advances in neural information processing systems* (pp. 281–288).

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., et al. (2012). Large scale distributed deep networks. In *Advances in neural information processing systems* (pp. 1223–1231).

Di Lorenzo, P., & Sayed, A. H. (2013). Sparse distributed learning based on diffusion adaptation. *IEEE Transactions on Signal Processing*, *61*(6), 1419–1433.

Forero, P. A., Cano, A., & Giannakis, G. B. (2010). Consensus-based distributed support vector machines. *The Journal of Machine Learning Research*, *11*, 1663–1707.

Georgopoulos, L., & Hasler, M. (2014). Distributed machine learning in networks by consensus. *Neurocomputing*, *124*, 2–12.

Hermans, M., & Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. In *Advances in neural information processing systems* (pp. 190–198).

Honeine, P., Richard, C., Bermudez, J. C. M., & Snoussi, H. (2008). Distributed prediction of time series data with kernels and adaptive filtering techniques in sensor networks. In *Proceedings of the 42nd Asilomar conference on signals, systems and computers* (pp. 246–250). IEEE.

Igelnik, B., & Pao, Y.-H. (1995). Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks*, *6*(6), 1320–1329.

Jaeger, H. (2002). Adaptive nonlinear system identification with echo state networks. In *Advances in neural information processing systems* (pp. 593–600).

Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science*, *304*(5667), 78–80.

Li, D., Han, M., & Wang, J. (2012). Chaotic time series prediction based on a novel robust echo state network. *IEEE Transactions on Neural Networks and Learning Systems*, *23*(5), 787–799.

Lin, X., Yang, Z., & Song, Y. (2009). Short-term stock price prediction based on echo state networks. *Expert Systems with Applications*, *36*(3), 7313–7317.

Lu, Y., Roychowdhury, V., & Vandenberghe, L. (2008). Distributed parallel support vector machines in strongly connected networks. *IEEE Transactions on Neural Networks*, *19*(7), 1167–1178.

Luitel, B., & Venayagamoorthy, G. K. (2012). Decentralized asynchronous learning in cellular neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *23*(11), 1755–1766.

Lukoševičius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*(3), 127–149.

Martens, J., & Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th international conference on machine learning, ICML* (pp. 1033–1040).

Mateos, G., Bazerque, J. A., & Giannakis, G. B. (2010). Distributed sparse linear regression. *IEEE Transactions on Signal Processing*, *58*(10), 5262–5276.

Monner, D., & Reggia, J. A. (2012). A generalized LSTM-like training algorithm for second-order recurrent neural networks. *Neural Networks*, *25*, 70–83.

Navia-Vázquez, A., Gutierrez-Gonzalez, D., Parrado-Hernández, E., & Navarro-Abellan, J. J. (2006). Distributed support vector machines. *IEEE Transactions on Neural Networks*, *17*(4), 1091–1097.

Obst, O. (2014). Distributed fault detection in sensor networks using a recurrent neural network. *Neural Processing Letters*, *40*(3), 261–273.

Olfati-Saber, R., Fax, J. A., & Murray, R. M. (2007). Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, *95*(1), 215–233.

Pao, Y.-H., & Takefuji, Y. (1992). Functional-link net computing: theory, system architecture, and functionalities. *Computer*, *25*(5), 76–79.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th international conference on machine learning*.

Pearlmutter, B. A. (1995). Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, *6*(5), 1212–1228.

Predd, J. B., Kulkarni, S. R., & Poor, H. V. (2007). Distributed learning in wireless sensor networks. *IEEE Signal Processing Magazine*, 56–69.

Predd, J. B., Kulkarni, S. R., & Poor, H. V. (2009). A collaborative training algorithm for distributed learning. *IEEE Transactions on Information Theory*, *55*(4), 1856–1871.

Sak, H., Senior, A., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proceedings of the annual conference of international speech communication association, INTERSPEECH*.

Sardellitti, S., Giona, M., & Barbarossa, S. (2010). Fast distributed average consensus algorithms based on advection–diffusion processes. *IEEE Transactions on Signal Processing*, *58*(2), 826–842.

Scardapane, S., Nocco, G., Comminiello, D., Scarpiniti, M., & Uncini, A. (2014). An effective criterion for pruning reservoir's connections in echo state networks. In *2014 International joint conference on neural networks (IJCNN)* (pp. 1205–1212). INNS/IEEE.

Scardapane, S., Wang, D., Panella, M., & Uncini, A. (2015a). Distributed learning with random vector functional-link networks. *Information Sciences*, *301*, 271–284.

Scardapane, S., Wang, D., Panella, M., & Uncini, A. (2015b). Distributed music classification using random vector functional-link nets. In *2015 International joint conference on neural networks (IJCNN)*. INNS/IEEE.

Schliebs, S., Mohemmed, A., & Kasabov, N. (2011). Are probabilistic spiking neural networks suitable for reservoir computing? In *2011 International joint conference on neural networks (IJCNN)* (pp. 3156–3163). INNS/IEEE.

Shutin, D., & Kubin, G. (2008). Echo state wireless sensor networks. In *2008 IEEE workshop on machine learning for signal processing* (pp. 151–156).

Skowronski, M. D., & Harris, J. G. (2007). Automatic speech recognition using a predictive echo state network classifier. *Neural Networks*, *20*(3), 414–423.

Steil, J. J. (2007). Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning. *Neural Networks*, *20*(3), 353–364.

Sutskever, I., Vinyals, O., & Le, Q. V. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).

Tong, M. H., Bickett, A. D., Christiansen, E. M., & Cottrell, G. W. (2007). Learning grammatical structure with echo state networks. *Neural Networks*, *20*(3), 424–432.

Triefenbach, F., Jalalvand, A., Demuynck, K., & Martens, J.-P. (2013). Acoustic modeling with hierarchical reservoirs. *IEEE Transactions on Audio, Speech and Language Processing*, *21*(11), 2439–2450.

Vandoorne, K., Dambre, J., Verstraeten, D., Schrauwen, B., & Bienstman, P. (2011). Parallel reservoir computing using optical amplifiers. *IEEE Transactions on Neural Networks*, *22*(9), 1469–1481.

Vandoorne, K., Mechet, P., Van Vaerenbergh, T., Fiers, M., Morthier, G., Verstraeten, D., et al. (2014). Experimental demonstration of reservoir computing on a silicon photonics chip. *Nature Communications*, *5*.

Verstraeten, D., Schrauwen, B., d'Haene, M., & Stroobandt, D. (2007). An experimental unification of reservoir computing methods. *Neural Networks*, *20*(3), 391–403.

Verykios, V. S., Bertino, E., Fovino, I. N., Provenza, L. P., Saygin, Y., & Theodoridis, Y. (2004). State-of-the-art in privacy preserving data mining. *ACM SIGMOD Record*, *33*(1), 50–57.

Wu, X., Zhu, X., Wu, G.-Q., & Ding, W. (2014). Data mining with big data. *The IEEE Transactions on Knowledge and Data Engineering*, *26*(1), 97–107.

Xiao, L., Boyd, S., & Lall, S. (2005). A scheme for robust distributed sensor fusion based on average consensus. In *Fourth international symposium on information processing in sensor networks, 2005* (pp. 63–70). IEEE.

Xue, Y., Yang, L., & Haykin, S. (2007). Decoupled echo state networks with lateral inhibition. *Neural Networks*, *20*(3), 365–376.

Yildiz, I. B., Jaeger, H., & Kiebel, S. J. (2012). Re-visiting the echo state property. *Neural Networks*, *35*, 1–9.

Zhang, B., Miller, D. J., & Wang, Y. (2012). Nonlinear system modeling with random matrices: echo state networks revisited. *IEEE Transactions on Neural Networks and Learning Systems*, *23*(1), 175–182.

Zinkevich, M., Weimer, M., Li, L., & Smola, A. J. (2010). Parallelized stochastic gradient descent. In *Advances in neural information processing systems* (pp. 2595–2603).