(c) 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works

Distributed Reservoir Computing with Sparse Readouts

Simone Scardapane, Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, Rome, Italy

Massimo Panella, Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, Rome, Italy

Danilo Comminiello, Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, Rome, Italy

Amir Hussain, School of Natural Sciences, University of Stirling, Stirling, UK

Aurelio Uncini, Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, Rome, Italy

Abstract

In a network of agents, a widespread problem is the need to estimate a common underlying function starting from locally distributed measurements. Real-world scenarios may not allow the presence of centralized fusion centers, requiring the development of distributed, message-passing implementations of the standard machine learning training algorithms. In this paper, we are concerned with the distributed training of a particular class of recurrent neural networks, namely echo state networks (ESNs). In the centralized case, ESNs have received considerable attention, due to the fact that they can be trained with standard linear regression routines. Based on this observation, in our previous work we have introduced a decentralized algorithm, framed in the distributed optimization field, in order to train an ESN. In this paper, we focus on an additional sparsity property of the output layer of ESNs, allowing for very efficient implementations of the resulting networks. In order to evaluate the proposed algorithm, we test it on two well-known prediction benchmarks, namely the Mackey-Glass chaotic time series and the 10th order nonlinear auto regressive moving average (NARMA) system.

I. Introduction

Supervised learning is the task of inferring a function starting from a finite set of labeled examples. Among many open research problems today, one of the most crucial is how to efficiently perform supervised inference when the

Corresponding Author: Simone Scardapane (Email: simone.scardapane@uniroma1.it).

data of the problem is not available on a centralized location, being distributed among a network of interconnected agents [1], [2]. Examples include peer-to-peer (P2P) networks [3], sensors [4], robotic swarms, and many others. This is more difficult than simply parallelizing the training computation over a cluster of processors (e.g. [5]), since many real-world unstructured networks do not possess any kind of centralized authority for collecting data, nor for coordinating the overall process [6]. Additionally, the data exchange might be limited due to bandwidth constraints or due to stringent privacy concerns over sensible portions of the data (e.g. medical records). As such, the agents in the network need to agree on a single model (such as a specific support vector machine) by only exchanging local messages among them, possibly with low knowledge of the structure of the network far from their spatial neighborhood [3].

In the specific context of distributed learning for artificial neural networks, large amount of work has been made in proposing decentralized algorithms for feedforward models. Among them, we can cite distributed protocols for support vector machines [7], functional-link networks [8], linear neurons [9], adaptive resonance theory (ART) networks [10], and many others. However, as we argued in [11], what is needed in many contexts is a distributed training algorithm for recurrent neural networks (RNNs). Thanks to the presence of recurrent connections, RNNs are able to efficiently capture the dynamics in the underlying process to be learned. Tasks that would benefit from such algorithms abound, including distributed multimedia classification [12], event detection with array of microphones [13], classification of texts in cluster environments [14] and prediction of highly nonlinear time-series in wireless sensor networks [4]. Still, it is known that training an RNN model is a challenging task even in a centralized context, which is far from being fully solved [15]–[18]. As such, there is a lack of available distributed protocols for RNN models satisfying all the requirements discussed above.

In [11], a first step towards this aim was made for a simple class of RNNs, known as echo state networks (ESNs), belonging to the wider family of reservoir computing methods [19]. In an ESN, the recurrent portion of the network (called reservoir) is fixed in advance, generally by extracting its parameters from a known probability distribution [19], [20]. Then, a feedforward output layer, called readout, is trained on top of the reservoir, using standard techniques from linear approximation theory, most notably ridge regression. Despite this simplification, ESNs have obtained remarkable results in many fields, including medical image segmentation [21], load prediction [22], language generation [23], and several others. Based on the strict separation between reservoir and readout, in [11] we proposed a distributed algorithm for training ESNs, by using the well-known optimization routine known as alternating direction method of multipliers (ADMM) [24]. The algorithm is able to perform as good as a centralized counterpart, and the only communication requested to the agents in the network is the computation of distributed averages. This is achieved by the use of decentralized average consensus (DAC) [25], [26], an efficient network protocol designed to this end. Additionally, the algorithm does not require the exchange of data points among the

agents, thus preserving privacy.

At the same time, standard ridge regression may not be the most suitable training algorithm for ESNs, as demonstrated by works exploring readouts trained via support vector based algorithms [27], elastic net penalties [22], and others. Specifically, in this paper we are concerned with training a sparse readout, i.e. a readout where the majority of the connections are set to zero. In the centralized case, this has been explored in depth in [28], where it is shown that a sparse readout can improve performance in specific scenarios. Moreover, having only a small number of connections can lead to extremely efficient implementations [29], [30], particularly on low-cost devices, and it has additional theoretical interests [31]. Thus, having the possibility of training sparse readouts for an ESN in a decentralized case can be a valuable tool. Since the readout is linear, sparsity can be enforced by including an additional L_1 regularization term to be minimized, resulting in the so-called LASSO algorithm [32], which can be solved efficiently in a wide variety of ways [33]. The LASSO estimator has also been investigated extensively in the distributed setting (albeit only for linear models), and a short review in this sense is provided in the following section.

Based on the above discussion, the aim of this paper is to extend the distributed ESN protocol presented in [11] with the inclusion of a decentralized LASSO training algorithm [34]. As we show in the subsequent sections, this allows us to obtain very sparse networks. Apart from achieving a higher accuracy in specific settings (as discussed in Section II), sparseness in the readout strongly reduces the number of parameters required for describing the ESN, allowing to reduce the in-network bandwidth required during the training phase, and the computational cost of performing a prediction. In order to test the validity of the proposal, we investigate two distributed (noisy) prediction problems, involving the Mackey-Glass chaotic time-series and a 10th order nonlinear auto regressive moving average (NARMA) system.

The rest of the paper is organized as follows. In Section II, we briefly investigate some works which are related to the topic of this paper, namely sparse linear models in distributed scenarios and in reservoir computing methods. Then, Section III introduces the basic ESN architecture which is used subsequently. In Section IV we present our distributed sparse algorithm, and we test it in Section V. Section VI concludes the paper, by providing a discussion on the current limitations and future directions of the present work.

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 column vectors. The operator $\|\cdot\|_p$ is the standard L_p 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).

II. Related works

In this section, we introduce some related works to the algorithm presented in this paper. Particularly, Section II-A describes works dealing with LASSO problems over a network of agents. Then, Section II-B provides an overview of ESN works dealing with sparse readouts.

A. Distributed LASSO over networks

Distributed training of a sparse linear method (i.e. LASSO) has been investigated extensively in the literature recently. Particularly, Mateos *et al.* [34] reformulate the problem of LASSO in a separable form, and then solve it by enforcing consensus constraints with the use of the ADMM procedure. They present three different versions, which differ in the amount of computational resources required by the single agent. Particularly, in the simplest case, it is shown that the local update step can be computed by an elementary thresholding operation. Mota *et al.* [35] solve in a similar way a closely related problem, denoted as basis pursuit. In [34, Section V], the authors discuss also a distributed cross-validation procedure for selecting an optimal regularization factor in a decentralized fashion.

An alternative formulation is presented in Chen and Sayed [36], where the L_1 norm is approximated with the twice-differentiable regularization term given by

$$\|\boldsymbol{\beta}\|_1 \approx \sum_{i=1}^{d} \sqrt{\beta_i^2 + \varepsilon^2}\,, \tag{1}$$

where d is the dimensionality of the vector $\boldsymbol{\beta}$, and ε is a sufficiently small number. The resulting problem is then solved with the use of diffusion adaptation (DA), a distributed optimization framework where local update steps are interleaved with averaging steps.

A third approach, based on the method of iterative thresholding, is instead presented by Ravazzi *et al.* [37], for both the LASSO problem and the optimally sparse ridge regression problem with an L_0 regularization term. Results are similar to [34], but the algorithm requires significantly less computations at every agent.

Much work has been done also in the case of online distributed LASSO problems, where data is arriving sequentially at each agent. Liu *et al.* [38] extend a distributed version of the standard least mean-square (LMS) filter, with the inclusion of L_0 and L_1 penalties, showing significant improvements with respect to the classical formulation when the underlying vector is sparse. A similar formulation is derived by Di Lorenzo and Sayed [9], with two important differences. Firstly, they consider two different stages of information exchange, allowing for a faster rate of convergence. Secondly, they consider an adaptive procedure for selecting an optimal regularization coefficient.

In the case of second order online algorithms, Liu *et al.* [39] present an algorithm framed on the principle of

maximum likelihood, with the use of expectation maximization and thresholding operators. An alternative, more demanding formulation, is presented by Barbarossa *et al.* [26, Section IV-A4], where the optimization problem is solved with the use of the ADMM procedure.

B. Sparse readouts for ESNs

In the ESN field, Dutoit *et al.* [28] were among the first to consider sparse readouts. They investigate different greedy methods to this end, including backward selection (where connections are removed one at a time based on an iterative procedure), random deletion, and others. Similar experiments are conducted by Kobialka and Kayani [40]. In both cases, improvements are found, in terms of both generalization accuracy and computational requirements.

The use of the LASSO algorithm, where sparsity is obtained by including an additional L_1 regularization term, is derived by Ceperic and Baric [41]. It is also possible to combine ridge regression with the LASSO algorithm, obtaining the so-called elastic net penalty [42]. This has been investigated independently by Ceperic and Baric [41], and Bianchi *et al.* [22].

Finally, a few additional works are worth being mentioned here. First, Butcher *et al.* [43] consider sparse readouts for a particular ESN architecture, where static projections are added to the reservoir. Second, Scardapane *et al.* [30] investigate the more general problem of pruning the internal connections of the reservoir, by computing the correlation among different states. Sparse readouts in the context of unreliable nanoscale networks are instead investigated in [44], [45].

III. Echo State Networks

In this section, we introduce the basic elements of the ESN framework. First, we detail the ESN architecture in Section III-A. Then, we show how to train ESNs in Section III-B, both with the ridge regression algorithm, and with the LASSO strategy.

A. Structure of an ESN

As we stated in Section I, ESN processing is subdivided into a fixed reservoir, followed by an adaptable readout. This is shown schematically in Fig. 1. More formally, let us denote as $\mathbf{x}[n]$ the N_i-dimensional input of the ESN at time n. After going through the N_r-dimensional reservoir, the internal state $\mathbf{h}[n-1] \in \mathbb{R}^{N_r}$ of the reservoir is updated according to the state equation:

$$\mathbf{h}[n] = f_{\text{res}}(\mathbf{W}_i^r \mathbf{x}[n] + \mathbf{W}_r^r \mathbf{h}[n-1] + \mathbf{w}_o^r y[n-1]) \,, \tag{2}$$

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 $\mathbf{w}_o^r \in \mathbb{R}^{N_r}$ are randomly generated matrices, $f_{\text{res}}(\cdot)$ is a suitably defined nonlinear function to be applied element-wise, and $y[n-1]$ is the (scalar) previous output of the network. The

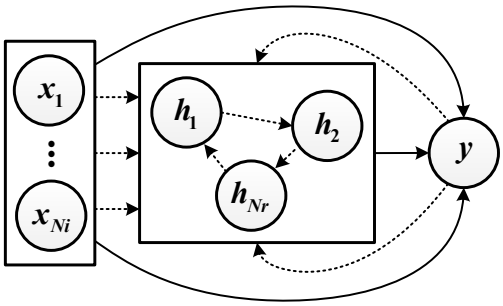


Figure 1. Schematic depiction of an ESN with a single output neuron. Random connections are shown with dashed lines, while trainable connections are shown with solid lines.

extension to a vector output is straightforward. Generally, the input is also supplemented by a unitary constant term, in order to provide an adaptable bias to the network. Additionally, during training it is possible to add an additional noise term in Eq. (2) to improve stability [20]. The new output of the ESN is then computed according to:

$$y[n] = f_{\text{out}}\left(\left(\mathbf{w}_i^o\right)^T \mathbf{x}[n] + \left(\mathbf{w}_r^o\right)^T \mathbf{h}[n]\right) , \tag{3}$$

where $\mathbf{w}_i^o \in \mathbb{R}^{N_i}$, $\mathbf{w}_r^o \in \mathbb{R}^{N_r}$ are adapted based on the training data, and $f_{\text{out}}(\cdot)$ is an invertible nonlinear function.

An important point in designing an ESN is to achieve the so-called echo state property (ESP), which ensures the stability of the reservoir's states [20]. This is commonly achieved by rescaling the internal matrix $\mathbf{W}_r^r$ so that its largest eigenvalue (in absolute terms) is below a certain threshold, which ensures a contractive behavior in the presence of zero input. In our experiments we follow this heuristic, as more advanced strategies tend to be either more complex [46], or too restrictive from a practical point of view [47].

B. Training the ESN

Training of an ESN is divided in three parts, due to the strict division between reservoir and readout. First, the fixed weights (shown with dashed lines in Fig. 1) are stochastically assigned at the beginning of the learning process. More details on this are provided in the experimental section. Next, suppose we are provided with a sequence of Q desired input-output pairs:

$$(\mathbf{x}[1], d[1]), \ldots, (\mathbf{x}[Q], d[Q]) .$$

In the 'warming' phase, this sequence is fed to the reservoir, giving a second sequence of internal states $\mathbf{h}[1], \ldots, \mathbf{h}[Q]$. During this phase, since the output of the ESN is not available for feedback, the desired output is used instead in

Eq. (2). Next, let us define the hidden matrix $\mathbf{H}$ and output vector $\mathbf{d}$ as:

$$\mathbf{H} = \begin{bmatrix} \mathbf{x}^T[1]\ \mathbf{h}^T[1] \\ \vdots \\ \mathbf{x}^T[Q]\ \mathbf{h}^T[Q] \end{bmatrix}, \tag{4}$$

$$\mathbf{d} = \begin{bmatrix} f_{\text{out}}^{-1}(d[1]) \\ \vdots \\ f_{\text{out}}^{-1}(d[Q]) \end{bmatrix}. \tag{5}$$

In the original implementation of ESNs, the optimal output weight vector is given by solving a ridge regression least-square problem as:

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathbb{R}^{N_i+N_r}}{\arg\min} \frac{1}{2}\left\|\mathbf{H}\mathbf{w}-\mathbf{d}\right\|_2^2 + \frac{\lambda}{2}\left\|\mathbf{w}\right\|_2^2, \tag{6}$$

where $\mathbf{w} = [\mathbf{w}_i^o\ \mathbf{w}_r^o]^T$ and $\lambda \in \mathbb{R}^+$ is a positive scalar known as *regularization factor*. The solution of the problem in Eq. (6) can be written in closed form by taking the gradient and setting it to zero, obtaining:

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

Practically, we can also remove the initial elements (denoted as 'wash-out' elements) from the sequence when solving the least-square problem, due to their transient state. However, we are interested in training a sparse readout, where most of the elements in $\mathbf{w}^*$ are set to zero. As shown in [28], this can lead to better regularization and, more importantly, it is more efficient to be implemented in low-cost hardware. To this end, we consider the standard LASSO problem originally introduced in [32]:

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathbb{R}^{N_i+N_r}}{\arg\min} \frac{1}{2}\left\|\mathbf{H}\mathbf{w}-\mathbf{d}\right\|_2^2 + \lambda\left\|\mathbf{w}\right\|_1, \tag{8}$$

where the L_1 norm $\|\mathbf{w}\|_1$ acts as a proxy of the L_0 norm, allowing for sparse results. The solution to problem (8) cannot be obtained in closed form anymore, however, many efficient methods are available to solve it [33].

IV. Distributed sparse ESNs

This section, which is the main innovative part of the present paper, introduces the sparse distributed algorithm for ESNs. In Section IV-A, following the ideas in [11], we detail our network's model and we formulate the distributed ESN training problem. Section IV-B details an ADMM-based algorithm for solving it, derived from the work in [34]. As is shown next, the communication among agents is restricted to a distributed computation of averages. How to achieve this with the DAC protocol is the topic of Section IV-C.

A. Formulation of the problem

For the rest of the paper, we suppose that the data of the problem, corresponding to the state matrix $\mathbf{H}$ and output vector $\mathbf{y}$, are not available on a centralized location. Instead, they are distributed row-wise over a network of L interconnected agents, each of which observes only a specific part of the available training samples. As an example, in a wireless sensor network (WSN), different sensors can make different observations of the underlying process that is common to all the sensors. For the purpose of this paper, we assume that the connectivity is known *a-priori* and is fixed. Given this, we can fully describe the connectivity between the agents in the form of an $L \times L$ connectivity matrix $\mathbf{C}$, where $C_{ij} \neq 0$ if and only if agents i and j are connected or $i = j$. We assume that the network is connected (i.e., every agent can be reached from any other agent with a finite number of steps), and undirected (i.e., $\mathbf{C}$ is symmetric).

For the purpose of applying the resulting algorithm to the widest possible class of problems, we also make two additional assumptions. First, agents can only communicate among their spatial neighbors, which ensures that the algorithm is applicable also in the case of largely unstructured networks, where no multi-hop communication protocols are available. Second, we are interested in training algorithms which do not require the exchange of training samples among the agents, an aspect that is crucial in big data and sensible applications [11].

Let us assume that all agents in the network have agreed on the choice of the fixed matrices $\mathbf{W}_i^r$, $\mathbf{W}_r^r$ and $\mathbf{w}_o^r$. This can be achieved in a variety of network-dependent modes, and we do not concern ourselves with this aspect here. In the simplest of cases, this choice can actually be pre-implemented on the agents, for example, by agreeing on a specific seed for the pseudo-random number generator. We denote with $\mathbf{H}_k$ and $\mathbf{d}_k$ the hidden matrices and output vectors, computed at the kth agent according to Eqs. (4)-(5) with its local dataset of observations. In this case, extending Eq. (8), the global optimization problem for an ESN with a sparse readout can be stated as:

$$\mathbf{w}^* = \underset{\mathbf{w} \in \mathbb{R}^{N_i+N_r}}{\arg\min} \frac{1}{2} \left(\sum_{k=1}^{L} \|\mathbf{H}_k \mathbf{w} - \mathbf{d}_k\|_2^2 \right) + \lambda \|\mathbf{w}\|_1 \ . \tag{9}$$

B. Solving the distributed optimization problem

The problem in Eq. (9) is a standard distributed LASSO problem, which can be solved with any technique among those described in Section II-A. Particularly, we choose the ADMM algorithm described in [24], [34] for a number of reasons. First, the ADMM is probably the widest employed algorithm for convex distributed optimization, due to its conceptual simplicity and efficiency [24]. Second, the original distributed ESN presented in [11] is also based on the ADMM algorithm, which ensures a fair comparison of the two implementations.

The ADMM algorithm can be derived in three successive steps. First, in order to decouple the problem at every agent, we introduce L local variables $\mathbf{w}_k$ for each agent, and we force them to be equal at convergence. This results

in the following constrained optimization problem:

$$\underset{\mathbf{z},\mathbf{w}_1,\ldots,\mathbf{w}_L\in\mathbb{R}^{N_i+N_r}}{\text{minimize}} \;\; \frac{1}{2}\left(\sum_{k=1}^{L}\|\mathbf{H}_k\mathbf{w}_k-\mathbf{d}_k\|_2^2\right)+\lambda\|\mathbf{z}\|_1$$
$$\text{subject to} \;\; \mathbf{w}_k=\mathbf{z},\; k=1,\ldots,L\,. \tag{10}$$

The ADMM requires the formulation of an augmented Lagrangian, which is composed of the standard Lagrangian function, supplemented by an additional L_2 term, which enforces the convergence:

$$\begin{aligned}\mathcal{L}_\gamma(\cdot)=&\frac{1}{2}\left(\sum_{k=1}^{L}\|\mathbf{H}_k\mathbf{w}_k-\mathbf{d}_k\|_2^2\right)+\lambda\|\mathbf{z}\|_1\\&+\sum_{k=1}^{L}\mathbf{t}_k^T(\mathbf{w}_k-\mathbf{z})+\frac{\gamma}{2}\sum_{k=1}^{L}\|\mathbf{w}_k-\mathbf{z}\|_2^2\,,\end{aligned} \tag{11}$$

where the additional term is weighted by a scalar $\gamma\in\mathbb{R}^+$, and $\mathbf{t}_k$ are the L sets of Lagrange multipliers. Based on this, the overall optimization problem is solved by using a three-step iterative procedure. At every iteration, we first optimize separately for $\mathbf{w}_k$ and $\mathbf{z}$, and then we update the Lagrangian multipliers using a steepest-descent approach:

$$\mathbf{w}_k[n+1]=\underset{\mathbf{w}_k\in\mathbb{R}^{N_i+N_r}}{\arg\min}\;\mathcal{L}\left(\mathbf{z}[n],\mathbf{w},\mathbf{t}[n]\right)\,, \tag{12}$$

$$\mathbf{z}[n+1]=\underset{\mathbf{z}\in\mathbb{R}^{N_i+N_r}}{\arg\min}\;\mathcal{L}\left(\mathbf{z},\mathbf{w}[n+1],\mathbf{t}[n]\right)\,, \tag{13}$$

$$\mathbf{t}_k[n+1]=\mathbf{t}_k[n]+\gamma\left(\mathbf{w}_k[n+1]-\mathbf{z}[n+1]\right)\,, \tag{14}$$

where with a slight abuse of notation we used $\mathbf{t}[n]$ (and similarly for the other variables) to denote the set of L variables $\mathbf{t}_1[n],\ldots,\mathbf{t}_L[n]$. This iterative procedure is guaranteed to reach the optimum of problem in Eq. (9) with linear asymptotic convergence. It is easy to show that, in this case, the updates can be computed in closed form as (see [24, Section 8.2.1]):

$$\mathbf{w}_k[n+1]=\left(\mathbf{H}_k^T\mathbf{H}_k+\gamma\mathbf{I}\right)^{-1}\left(\mathbf{H}_k^T\mathbf{d}_k+\gamma\left(\mathbf{z}[n]-\mathbf{t}_k[n]\right)\right)\,, \tag{15}$$

$$\mathbf{z}[n+1]=S_{\lambda/N\gamma}\left(\hat{\mathbf{w}}[n+1]+\hat{\mathbf{t}}[n]\right)\,, \tag{16}$$

$$\mathbf{t}_k[n+1]=\mathbf{t}_k[n]+\mathbf{w}_k[n+1]-\mathbf{z}[n+1]\,, \tag{17}$$

where we have introduced the averages:

$$\hat{\mathbf{w}}[n+1] = \frac{1}{L}\sum_{k=1}^{L}\mathbf{w}_k[n+1]\,, \tag{18}$$

$$\hat{\mathbf{t}}[n] = \frac{1}{L}\sum_{k=1}^{L}\mathbf{t}_k[n]\,. \tag{19}$$

The soft-thresholding operator $S_\alpha(\cdot)$ in Eq. (16) is defined for a generic vector $\mathbf{a}$ as:

$$S_\alpha(\mathbf{a}) = \left(1 - \frac{\alpha}{\|\mathbf{a}\|_2}\right)_+ \mathbf{a}\,,$$

where $(\cdot)_+$ is defined element-wise as $(\cdot)_+ = \max(0, \cdot)$. We can see that the first and third updates can be computed locally at every agent, while in the second update communication is restricted to an average computation. The next section shows how this can be implemented in general networks via the use of the DAC protocol. Before that, however, we make two remarks on Eq. (15). First, the matrix inversion does not depend on the specific iteration, and it can be cached for subsequent evaluations. More advanced speed-ups can also be obtained by caching the Cholesky decomposition [34]. Second, suppose that on an agent we have $N_k \ll N_i + N_r$, where N_k is the number of observations available at agent k. Then, we can exploit the matrix inversion lemma to obtain a more convenient matrix inversion step (see [34, Remark 3]):

$$\left(\mathbf{H}_k^T\mathbf{H}_k + \gamma\mathbf{I}\right)^{-1} = \gamma^{-1}\left[\mathbf{I} - \mathbf{H}_k^T\left(\gamma\mathbf{I} + \mathbf{H}_k\mathbf{H}_k^T\right)^{-1}\mathbf{H}_k\right]. \tag{20}$$

C. Distributed computation of the averages

From the previous section, it is possible to see that the only communication required by our training algorithm is the computation of the two averages (at every iteration) $\hat{\mathbf{w}}[n+1]$ and $\hat{\mathbf{t}}[n]$. Clearly, the specific implementation of this step is dependent on the actual details of the communication layer of the network. While we focus on the DAC protocol, which is typical in WSN networks, we stress that many other possibilities are available and can be used with our training algorithm, such as the use of random walks for computing approximate sums [48]. DAC is an interesting choice, however, since it requires communication only between neighbors in the network. Additionally, its asymptotic behavior has been investigated in-depth, with many variants proposed in the literature [3].

Each agent initializes its estimate of the global average as:

$$\mathbf{q}_k[n+1, 0] = \begin{bmatrix}\mathbf{w}_k[n+1] \\ \mathbf{t}_k[n]\end{bmatrix},$$

where we have stacked the two vectors in order to compute both averages in a single step. Additionally, we have introduced a second time index to denote the internal iteration for computing the average. In a DAC algorithm, this

value is iteratively refined at every agent as:

$$\mathbf{q}_k[n+1, j+1] = C_{kk}\mathbf{q}_k[n+1, j] + \sum_{l \in \mathcal{N}_k} C_{kl}\mathbf{q}_l[n+1, j]\,, \tag{21}$$

where C_{kl} denotes the (k, l)-th entry of the connectivity matrix $\mathbf{C}$, and $\mathcal{N}_k$ denotes the set of agents to which agent k is directly connected. Due to the way in which this matrix is constructed, every agent iteratively computes a weighted average of the values of its neighbors' estimates. Under suitable choices of the connectivity matrix (see [49]), the iteration defined by Eq. (21) converges to the global average at every agent, i.e.

$$\lim_{j \to +\infty} \mathbf{q}_k[n+1, j] = \frac{1}{L}\sum_{k=1}^{L} \mathbf{q}_k[n+1, 0]\,. \tag{22}$$

More specifically, for networks which are undirected and connected (as those considered in this paper), the connectivity matrix must respect:

$$\mathbf{C} \cdot \mathbf{1} = \mathbf{1}\,, \tag{23}$$

$$\rho\left(\mathbf{C} - \tfrac{\mathbf{1}\cdot\mathbf{1}^T}{L}\right) < 1\,, \tag{24}$$

where $\rho(\cdot)$ denotes the spectral radius operator. As an example, in this work we consider the so-called 'max-degree' weights given by:

$$C_{kj} = \begin{cases} \frac{1}{d+1} & \text{if } k \in \mathcal{N}_j \\ 1 - \frac{d_k}{d+1} & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}\,, \tag{25}$$

where d_k is the degree of agent k, and d is the degree of the network. The overall distributed sparse ESN algorithm is summarized in Algorithm I.

V. Experimental validation

A. Experimental setup

In order to test the validity of the proposed algorithm, we employ two standard time-series prediction tasks. The first one is the Mackey-Glass (MG) time-series, defined in continuous time by the differential equation:

$$\dot{x}(t) = \beta x(t) + \frac{\alpha x(t-\tau)}{1 + x^{\theta}(t-\tau)}\,. \tag{26}$$

We use the common assignment $\alpha = 0.2$, $\beta = -0.1$, $\theta = 10$, giving rise to a chaotic behavior for $\tau > 16.8$. In particular, in our experiments we set $\tau = 30$. Time-series in Eq. (26) is integrated with a 4th order Runge-Kutta method using a time step of $T = 0.1$, where $x[n] = x(nT)$, and then sampled every 10 time-instants. The task is

Algorithm I
LOCAL TRAINING ALGORITHM FOR SPARSE ADMM-BASED ESN AT kTH AGENT

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

1: Assign matrices $\mathbf{W}_i^r$, $\mathbf{W}_r^r$ and $\mathbf{w}_o^r$. These must be the same across all agents.
2: Gather the local hidden matrix $\mathbf{H}_k$ and teacher signal $\mathbf{d}_k$ from the local observations.
3: Initialize the Lagrange multipliers $\mathbf{t}_k[0] = \mathbf{0}$, and $\mathbf{z}[0] = \mathbf{0}$.
4: **for** n from 0 to T **do**
5: Compute $\mathbf{w}_k[n+1]$ according to Eq. (15).
6: Compute averages $\hat{\mathbf{w}}$ and $\hat{\mathbf{t}}$ according to the DAC protocol of Section IV-C.
7: Compute $\mathbf{z}[n+1]$ according to Eq. (16).
8: Update $\mathbf{t}_k[n+1]$ according to Eq. (17).
9: **end for**
10: **return** $\mathbf{z}[n]$

a noisy 10-step ahead prediction task, i.e.:

$$d[n] = x[n+10] + \mathcal{N}(0, \sigma^2)\,, \tag{27}$$

where $\mathcal{N}(0, \sigma^2)$ denotes Gaussian noise with mean 0 and variance $\sigma^2 = 0.01$.

The second task is the NARMA-10 dataset (denoted by N10), 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 in [20]:

$$d[n] = 0.1 + 0.3d[n-1] + 0.05d[n-1]\prod_{i=1}^{10} d[n-i] + 1.5x[n]x[n-9] + \mathcal{N}(0, \sigma^2)\,, \tag{28}$$

where similarly to before we include a noise term. The output is then squashed to the interval $[-1, +1]$ by the non-linear transformation:

$$d[n] = \tanh(d[n] - \hat{d})\,, \tag{29}$$

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

We generate one training sequence of 6,000 elements, and one test sequence of 4,000 elements. For the ESN architecture, we select $N_r = 500$ in order to have a slightly redundant reservoir, while the other design parameters are chosen in accordance to the grid-search procedure detailed in [11]. In particular, we make the following choices:

- 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 $[-\alpha_i,\ \alpha_i]$. We select $\alpha_i = 0.3$ for the MG dataset, and $\alpha_i = 0.5$ for the N10 dataset.
- The matrix $\mathbf{w}_o^r$, connecting the output to the reservoir, is initialized as a zero matrix, as feedback was not

found to provide improvement in performance in this case.

- The elements of the internal reservoir matrix $\mathbf{W}_r^r$ are initialized from the uniform distribution in $[-1,\ +1]$. Then, on average 50% of its connections are set to 0, to encourage sparseness. Finally, the matrix is rescaled to have a desired spectral radius $\rho = 0.9$ for both problems.
- We use the nonlinearity $\tanh(\cdot)$ in the reservoir, while a scaled identity $f(s) = \alpha_t s$ as the output function. The parameter α_t is set to 0.5 for MG and 0.1 for N10.

Moreover, we insert uniform noise in the state update of the reservoir, sampled uniformly in the interval $\left[0,\ 10^{-6}\right]$, and we discard $D = 600$ initial elements from the training sequence. After running the trained ESN on the test samples, we gather the predicted outputs $\tilde{y}_1, \ldots, \tilde{y}_K$, where K is the number of testing samples after removing the wash-out elements from the test sequence. From these, we compute the normalized root mean-squared error (NRMSE), defined as:

$$\text{NRMSE} = \sqrt{\frac{\sum_{i=1}^{K} \left[\tilde{y}_i - d_i\right]^2}{|K|\hat{\sigma}_d}}\,, \tag{30}$$

where $\hat{\sigma}_d$ is an empirical estimate of the variance of the true output samples $d_1, \ldots, d_K$. To average out empirical effects due to stochastic assignments, experiments are repeated 20 times and results are averaged over the runs. Simulations are performed in MATLAB, by adapting the code from [11].[1]

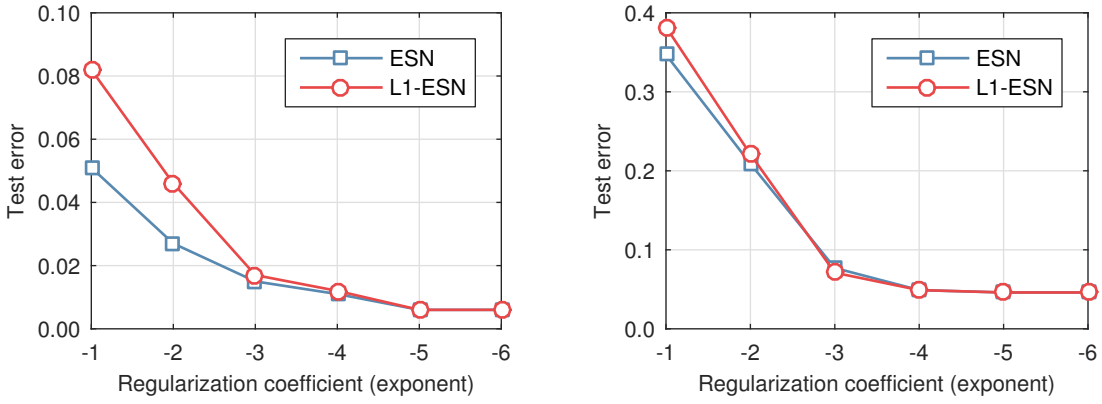
B. Comparisons in the centralized case

We begin our experimental evaluation by comparing the standard ESN and the ESN trained using the LASSO algorithm (denoted as L1-ESN) in the centralized case. This allows us to better investigate their behavior, and to choose an optimal regularization parameter λ. Particularly, we analyze test error, training time, and sparsity of the resulting L1-ESN when varying λ in 10^{-j}, with j going from 1 to 6. The LASSO problems are solved using a freely available implementation of the iterated ridge regression algorithm by Schmidt [50], [51].[2] The algorithm works by approximating the L_1 term in (8), and iteratively solving the resulting ridge regression problem. Results are presented in Fig. 2, where results for MG and N10 are shown in the left and right columns, respectively.

First of all, we can see clearly from Figs. 2a and 2b that the regularization effect of the two algorithms is similar, a result in line with the previous works [28]. Particularly, for large regularization factors, the estimates tend to provide an unsatisfactory test error, which, however, is relatively stable for sufficiently small coefficients. The tendency to select such a small factor is to be expected, due to the artificial nature of the datasets. A minimum in test error is reached for j around -5 for MG, and j around -4 for N10.

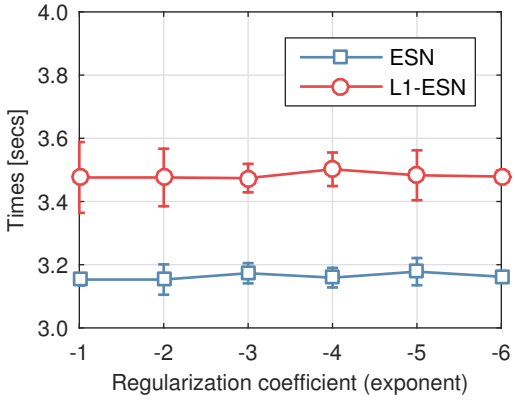
[1] https://bitbucket.org/ispamm/distributed-esn

[2] http://www.cs.ubc.ca/~schmidtm/Software/lasso.html

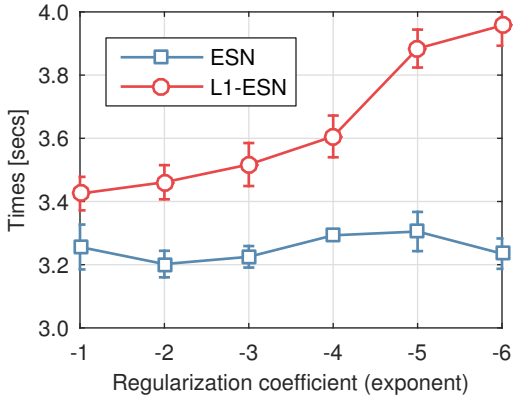


(a) Test error (MG)

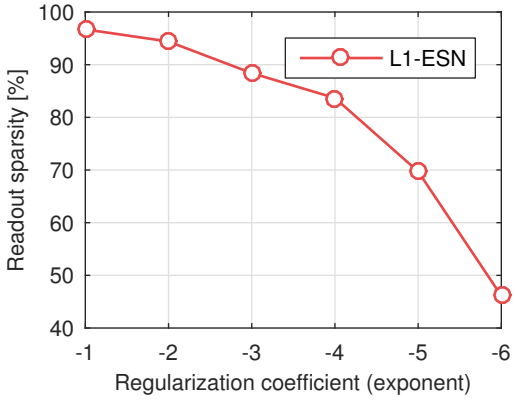
(b) Test error (N10)



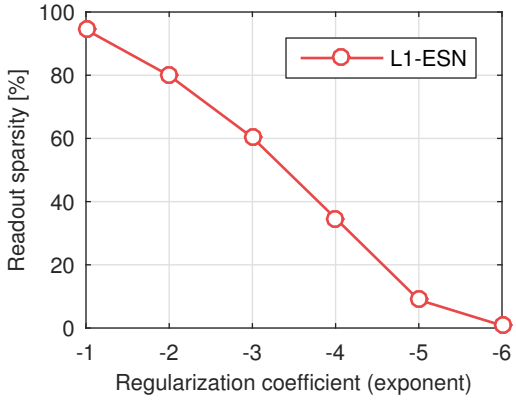
(c) Training time (MG)



(d) Training time (N10)



(e) Sparsity (MG)



(f) Sparsity (N10)

Figure 2. Evolution of (a-b) test error, (c-d) training time and (e-f) sparsity of the output vector when varying the regularization coefficient in 10^j, $j = -1, \ldots, -6$. Results for the MG dataset are shown on the left column, while results for the N10 dataset are shown in the right column.

Table I
THE RESULTS OF FIG. 2, SHOWN IN TABULAR FORM, TOGETHER WITH ONE STANDARD DEVIATION.

Dataset	Coefficient λ	Algorithm	Test error (NRMSE)	Training time [secs]	Sparsity [%]
Mackey-Glass	10^{-1}	ESN	0.051 ± 0.010	3.153 ± 0.019	0
		L1-ESN	0.082 ± 0.001	3.476 ± 0.112	0.967 ± 0.01
	10^{-2}	ESN	0.027 ± 0.006	3.153 ± 0.048	0
		L1-ESN	0.046 ± 0.001	3.476 ± 0.091	0.944 ± 0.06
	10^{-3}	ESN	0.015 ± 0.003	3.173 ± 0.032	0
		L1-ESN	0.017 ± 0.001	3.474 ± 0.045	0.884 ± 0.01
	10^{-4}	ESN	0.011 ± 0.001	3.159 ± 0.031	0
		L1-ESN	0.012 ± 0.001	3.502 ± 0.053	0.837 ± 0.04
	10^{-5}	ESN	0.006 ± 0.001	3.178 ± 0.043	0
		L1-ESN	0.006 ± 0.001	3.483 ± 0.079	0.697 ± 0.09
	10^{-6}	ESN	0.006 ± 0.001	3.162 ± 0.011	0
		L1-ESN	0.006 ± 0.001	3.976 ± 0.019	0.461 ± 0.03
NARMA 10	10^{-1}	ESN	0.347 ± 0.006	3.256 ± 0.071	0
		L1-ESN	0.382 ± 0.008	3.425 ± 0.053	0.944 ± 0.01
	10^{-2}	ESN	0.209 ± 0.004	3.202 ± 0.042	0
		L1-ESN	0.221 ± 0.007	3.461 ± 0.054	0.799 ± 0.04
	10^{-3}	ESN	0.077 ± 0.001	3.225 ± 0.034	0
		L1-ESN	0.071 ± 0.001	3.517 ± 0.068	0.603 ± 0.03
	10^{-4}	ESN	0.049 ± 0.001	3.293 ± 0.015	0
		L1-ESN	0.046 ± 0.001	3.606 ± 0.066	0.384 ± 0.02
	10^{-5}	ESN	0.046 ± 0.001	3.305 ± 0.062	0
		L1-ESN	0.046 ± 0.001	3.884 ± 0.060	0.089 ± 0.01
	10^{-6}	ESN	0.046 ± 0.001	3.235 ± 0.048	0
		L1-ESN	0.046 ± 0.001	3.957 ± 0.064	0.008 ± 0.01

With respect to the training time, ridge regression is relatively stable to the amount of regularization, as the matrix to be inverted tends to be already well conditioned. Training time of LASSO is regular for MG, while it slightly increases for larger values of j in the N10 case, as shown in Fig. 2d. In all cases, however, it is comparable to that of ridge regression, with a small increase of 0.5 seconds in average.

The most important aspect, however, is evidenced in Figs. 2e and 2f. Clearly, sparsity of the readout goes from almost 100% to 0% as the regularization factor decreases. At the point of best test accuracy, the resulting readout has an average sparsity of 70% for MG and 38% for N10. This, combined with the simultaneous possibility of pruning the resulting reservoir [28], [30], can lead to an extreme saving of computational resources requested at the single sensor during the prediction phase. In order to provide a simpler comparison of the results, we also display

them in tabular form in Table I.

C. Comparisons in the distributed case

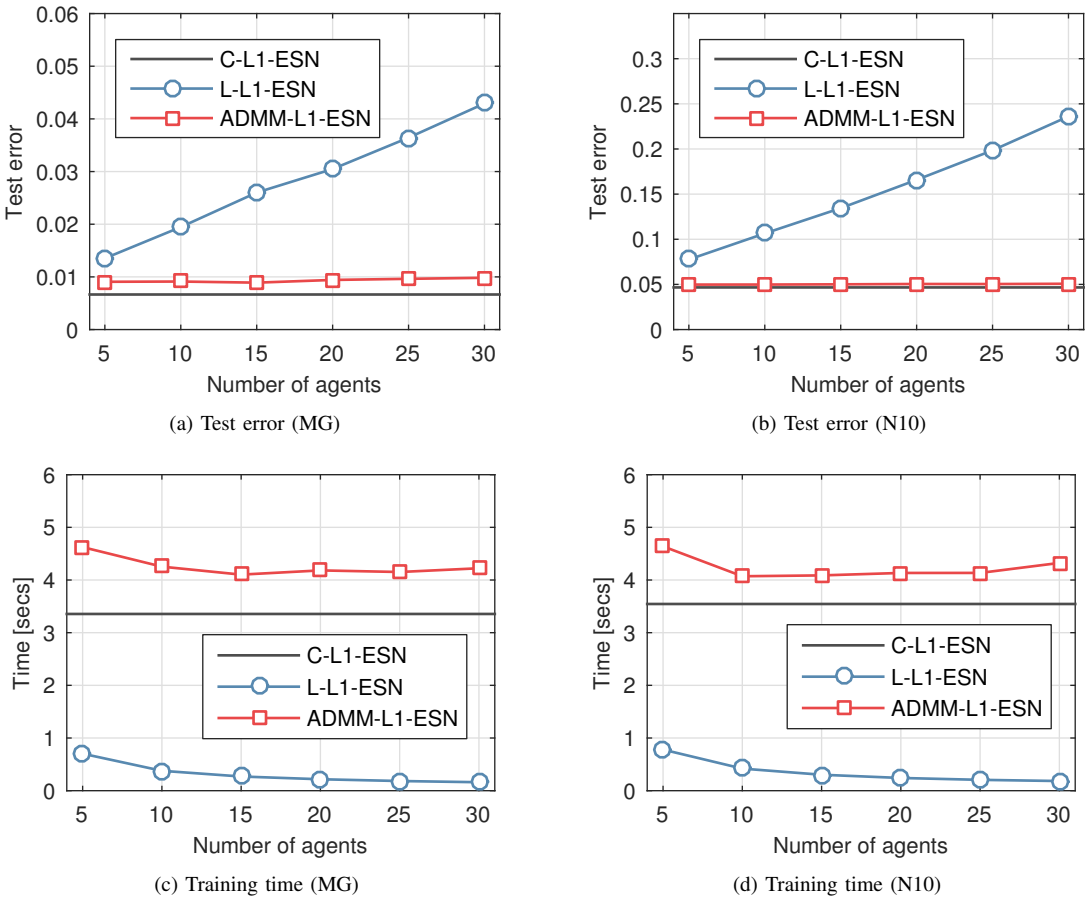
We now consider the implementation of the distributed L1-ESN over a network of agents. More in details, training observations are uniformly subdivided among the L agents in a predefined network, with L varying from 5 to 30 by steps of 5. For every run, the connectivity among the agents is generated randomly, such that each pair of agents has a 25% probability of being connected, with the only global requirement that the overall network is connected. The following three algorithms are compared:

1) **Centralized ESN** (C-L1-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. (9). This is equivalent to the ESN analyzed in the previous section, and following the results obtained there, we set $\lambda = 10^{-5}$ for MG, and $\lambda = 10^{-4}$ for N10.
2) **Local ESN** (L-L1-ESN): in this case, each agent trains an L1-ESN starting from its local measurements, but no communication is performed. The testing error is averaged throughout the L agents.
3) **ADMM-based ESN** (ADMM-L1-ESN): this is trained with the algorithm introduced in Section IV. In accordance with [11] we select $\gamma = 0.01$ and a maximum number of 400 iterations. For the DAC protocol, we set a maximum number of 300 iterations. DAC also stops whenever the updates (in norm) at every agent are smaller than a predefined threshold $\delta = 10^{-8}$:

$$\left\| \boldsymbol{q}_k[n+1;j] - \boldsymbol{q}_k[n+1;j-1] \right\|_2^2 < \delta, \; k \in \{1, 2, \ldots, L\} \; . \tag{31}$$

Results of this set of experiments are presented in Fig. 3. Similarly to before, results from the two datasets are presented in the left and right columns, respectively. From Figs. 3a and 3b we see that, although L-L1-ESN achieves degrading performance for bigger networks (due to the lower number of measurements per agent), ADMM-L1-ESN is able to effectively track the performance of the centralized counterpart, except for a small deviation in MG. Indeed, it is possible to reduce this gap by increasing the number of iterations; however, the performance gain is not balanced by the increase in computational cost.

With respect to the training time, it is possible to see from Figs. 3c and 3d that the training time is relatively steady for larger networks in ADMM-L1-ESN, showing its feasibility in the context of large sensor networks. Moreover, the computational cost requested by the distributed procedure is low and, in the worst case, it requires no more than 1 second with respect to the cost of a centralized counterpart. Clearly, in a practical situation, this must be supplemented by an analysis of the communication cost incurred over the channel, however, this aspect goes outside the current paper and it is well established in the literature [26]. Overall, we can see that our distributed



(a) Test error (MG)

(b) Test error (N10)

(c) Training time (MG)

(d) Training time (N10)

Figure 3. Evolution of (a-b) test error, (c-d) training time when varying the number of agents in the network from 5 to 30 by steps of 5.

protocol allows for an efficient implementation in terms of performance and training time, while at the same time guaranteeing a good level of sparsity of the resulting readout. This, in turn, is essential for many practical implementations where computational savings are necessary.

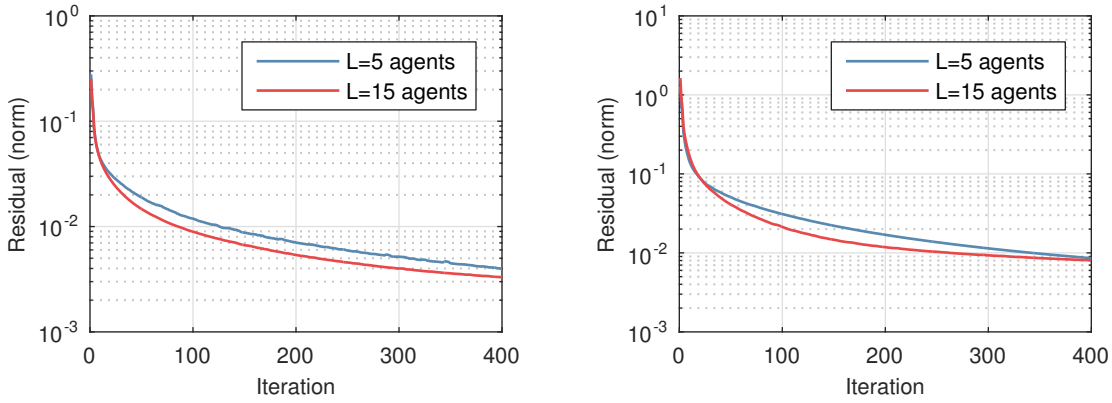
Some additional insights into the convergence behavior of ADMM-L1-ESN can also be obtained by analyzing the evolution of the so-called (primal) residual, given by [24]:

$$r[n+1] = \frac{1}{L}\sum_{k=1}^{L}\left\|\mathbf{w}_k[n+1] - \mathbf{z}[n+1]\right\|_2 . \tag{32}$$

As can be seen from Fig. 4 (shown with a logarithmic y-axis), this rapidly converges towards 0, ensuring that the algorithm is able to reach a stationary solution in a relatively small number of iterations.

VI. Conclusions

In this paper, we have proposed a distributed training protocol for ESNs, which is guaranteed to provide a sparse readout. This leads to efficient implementations in low-cost hardware and considerable computational savings. The



(a) Residual (MG) (b) Residual (N10)

Figure 4. Evolution of the (primal) residual for $L = 5$ and $L = 15$.

algorithm is based on a well-known distributed implementation of the LASSO problem, and it is formulated only in terms of local exchanges of information among neighboring agents. Overall, this represents one of the first fully distributed algorithms for recurrent neural network architectures. As such, it has a wide range of possible applications, including prediction over sensor networks, action modeling in a robotic swarm, distributed multimedia classification, and many others. Additionally, linking sparse modeling with RNNs has further theoretical interests.

Future works can consider the implementation of more efficient distributed LASSO procedures, such as those presented in Section II-A. More in general, it is possible to consider other distributed training criteria (such as training via a support vector algorithm) to be implemented in a distributed fashion. Finally, ESNs are known to perform worse for problems that require a long memory. In this case, it is necessary to devise distributed strategies for other classes of recurrent networks, such as LSTM architectures.

ACKNOWLEDGMENTS

A. Hussain was part-supported by the UK Engineering and Physical Science Research Council (EPSRC) grant no. EP/M026981/1, and the UK Royal Society of Edinburgh (RSE) and Natural Science Foundation of China (NNSFC) joint-project grant no. 61411130162. We also wish to thank the anonymous reviewers who helped improve the quality of our paper.

REFERENCES

[1] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[2] S. Scardapane, R. Fierimonte, P. Di Lorenzo, M. Panella, and A. Uncini, "Distributed semi-supervised support vector machines," *Neural Networks*, vol. 80, pp. 43–52, 2016.

[3] L. Georgopoulos and M. Hasler, "Distributed machine learning in networks by consensus," *Neurocomputing*, vol. 124, pp. 2–12, Jan. 2014.

[4] J. B. Predd, S. R. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Process. Mag.*, vol. 23, no. 4, pp. 56–69, 2007.

[5] M. A. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in Neural Inform. Process. Syst.*, 2010, pp. 2595–2603.

[6] E. Baccarelli, N. Cordeschi, A. Mei, M. Panella, M. Shojafar, and J. Stefa, "Energy-efficient dynamic traffic offloading and reconfiguration of networked datacenters for big data stream mobile computing: Review, challenges, and a case study," *IEEE Network*, vol. 30, no. 2, pp. 56–61, 2016.

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

[8] S. Scardapane, D. Wang, M. Panella, and A. Uncini, "Distributed learning for random vector functional-link networks," *Inform. Sciences*, vol. 301, pp. 271–284, 2015.

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

[10] A. Kulakov and D. Davcev, "Distributed data processing in wireless sensor networks based on artificial neural-networks algorithms," in *10th IEEE Symp. on Computers and Communications*. IEEE, 2005, pp. 353–358.

[11] S. Scardapane, D. Wang, and M. Panella, "A decentralized training algorithm for echo state networks in distributed big data applications," *Neural Networks*, vol. 78, pp. 65–74, 2016.

[12] S. Scardapane, R. Fierimonte, D. Wang, M. Panella, and A. Uncini, "Distributed music classification using random vector functional-link nets," in *Proc. Int. Joint Conf. on Neural Networks*. INNS/IEEE, 2015.

[13] B. Malhotra, I. Nikolaidis, and J. Harms, "Distributed classification of acoustic targets in wireless audio-sensor networks," *Comput. Networks*, vol. 52, no. 13, pp. 2582–2593, 2008.

[14] C. Silva, U. Lotrič, B. Ribeiro, and A. Dobnikar, "Distributed text classification with an ensemble kernel-based learning approach," *IEEE Trans. Syst. Man Cybern. C, Appl. Rev.*, vol. 40, no. 3, pp. 287–297, 2010.

[15] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *JMLR Workshop and Conf. Proc.*, vol. 28, no. 3, pp. 1310–1318, 2013.

[16] D. D. Monner and J. A. Reggia, "A generalized LSTM-like training algorithm for second-order recurrent neural networks," *Neural Networks*, vol. 25, pp. 70–83, 2012.

[17] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in *Advances in Neural Inform. Process. Syst.*, 2013, pp. 190–198.

[18] M. Wöllmer, F. Eyben, A. Graves, B. Schuller, and G. Rigoll, "Bidirectional LSTM networks for context-sensitive keyword detection in a cognitive virtual agent framework," *Cognitive Computation*, vol. 2, no. 3, pp. 180–190, 2010.

[19] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Review*, vol. 3, no. 3, pp. 127–149, 2009.

[20] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *Advances in Neural Inform. Process. Syst.*, 2002, pp. 593–600.

[21] B. Meftah, O. Lézoray, and A. Benyettou, "Novel approach using echo state networks for microscopic cellular image segmentation," *Cognitive Computation*, vol. 8, no. 2, pp. 237–245, 2015.

[22] F. M. Bianchi, S. Scardapane, A. Uncini, A. Rizzi, and A. Sadeghian, "Prediction of telephone calls load using echo state network with exogenous variables," *Neural Networks*, vol. 71, pp. 204–213, 2015.

[23] M. H. Tong, A. D. Bickett, E. M. Christiansen, and G. W. Cottrell, "Learning grammatical structure with echo state networks," *Neural Networks*, vol. 20, no. 3, pp. 424–432, 2007.

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

[25] L. Xiao, S. Boyd, and S. Lall, "A scheme for robust distributed sensor fusion based on average consensus," in *4th Int. Symp. on Inform. Process. in Sensor Networks*. IEEE, 2005, pp. 63–70.

[26] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Distributed detection and estimation in wireless sensor networks," in *Academic Press Library in Signal Processing, Vol. 2, Communications and Radar Signal Processing*, R. Chellappa and S. Theodoridis, Eds., 2014, pp. 329–408.

[27] Z. Shi and M. Han, "Support vector echo-state machine for chaotic time-series prediction," *IEEE Trans. Neural Netw.*, vol. 18, no. 2, pp. 359–372, 2007.

[28] X. Dutoit, B. Schrauwen, J. Van Campenhout, D. Stroobandt, H. Van Brussel, and M. Nuttin, "Pruning and regularization in reservoir computing," *Neurocomputing*, vol. 72, no. 7, pp. 1534–1546, 2009.

[29] M. Tanveer, "Robust and sparse linear programming twin support vector machines," *Cognitive Computation*, vol. 7, no. 1, pp. 137–149, 2015.

[30] S. Scardapane, G. Nocco, D. Comminiello, M. Scarpiniti, and A. Uncini, "An effective criterion for pruning reservoir's connections in echo state networks," in *Proc. Int. Joint Conf. on Neural Networks*. INNS/IEEE, 2014, pp. 1205–1212.

[31] S. Ganguli and H. Sompolinsky, "Short-term memory in neuronal networks through dynamical compressed sensing," in *Advances in Neural Inform. Process. Syst.*, 2010, pp. 667–675.

[32] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Statistical Soc.: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[33] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Optimization with sparsity-inducing penalties," *Found. and Trends in Machine Learning*, vol. 4, no. 1, pp. 1–106, 2012.

[34] G. Mateos, J. A. Bazerque, and G. B. Giannakis, "Distributed sparse linear regression," *IEEE Trans. Signal Process.*, vol. 58, no. 10, pp. 5262–5276, 2010.

[35] J. F. C. Mota, J. M. F. Xavier, P. M. Q. Aguiar, and M. Püschel, "Distributed basis pursuit," *IEEE Trans. Signal Process.*, vol. 60, no. 4, pp. 1942–1956, 2012.

[36] J. Chen and A. H. Sayed, "Diffusion adaptation strategies for distributed optimization and learning over networks," *IEEE Trans. Signal Process.*, vol. 60, no. 8, pp. 4289–4305, 2012.

[37] C. Ravazzi, S. M. Fosson, and E. Magli, "Distributed iterative thresholding for ℓ_0/ℓ_1-regularized linear inverse problems," *IEEE Trans. Inf. Theory*, vol. 61, no. 4, pp. 2081–2100, 2015.

[38] Y. Liu, C. Li, and Z. Zhang, "Diffusion sparse least-mean squares over networks," *IEEE Trans. Signal Process.*, vol. 60, no. 8, pp. 4480–4485, 2012.

[39] Z. Liu, Y. Liu, and C. Li, "Distributed sparse recursive least-squares over networks," *IEEE Trans. Signal Process.*, vol. 62, no. 6, pp. 1386–1395, 2014.

[40] H.-U. Kobialka and U. Kayani, "Echo state networks with sparse output connections," in *Artificial Neural Networks – ICANN 2010*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6352, pp. 356–361.

[41] V. Ceperic and A. Baric, "Reducing complexity of echo state networks with sparse linear regression algorithms," in *2014 UKSim-AMSS 16th Int. Conf. on Comput. Modelling and Simulation (UKSim'14)*, March 2014, pp. 26–31.

[42] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *J. Roy. Statistical Soc.: Series B Methodological*, vol. 67, no. 2, pp. 301–320, 2005.

[43] J. B. Butcher, C. R. Day, P. W. Haycock, D. Verstraeten, and B. Schrauwen, "Pruning reservoirs with random static projections," in *2010 IEEE Int. Workshop on Machine Learning for Signal Processing (MLSP'10)*, Aug 2010, pp. 250–255.

[44] A. Goudarzi, M. R. Lakin, and D. Stefanovic, "Reservoir computing approach to robust computation using unreliable nanoscale networks," in *Unconventional Computation and Natural Computation*. Springer, 2014, pp. 164–176.

[45] A. Goudarzi, M. R. Lakin, D. Stefanovic, and C. Teuscher, "A model for variation-and fault-tolerant digital logic using self-assembled nanowire architectures," in *Proc. of the 2014 IEEE/ACM Int. Symp. on Nanoscale Architectures*. ACM, 2014, pp. 116–121.

[46] M. C. Ozturk, D. Xu, and J. C. Príncipe, "Analysis and design of echo state networks," *Neural Computation*, vol. 19, no. 1, pp. 111–138, 2007.

[47] I. B. Yildiz, H. Jaeger, and S. J. Kiebel, "Re-visiting the echo state property," *Neural Networks*, vol. 35, pp. 1–9, 2012.

[48] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks: Algorithms and evaluation," *Performance Evaluation*, vol. 63, no. 3, pp. 241–263, 2006.

[49] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[50] J. Fan and R. Li, "Variable selection via nonconcave penalized likelihood and its oracle properties," *J. Amer. Statistical Assoc.*, vol. 96, no. 456, pp. 1348–1360, 2001.

[51] M. Schmidt, "Least squares optimization with L1-norm regularization," *Project Report CS542B, University of British Columbia*, 2005.