Procedia Computer Science

Volume 53, 2015, Pages 468–477

2015 INNS Conference on Big Data

Learning from Distributed Data Sources using Random Vector Functional-Link Networks

Simone Scardapane[1], Massimo Panella[1], Danilo Comminiello[1], and Aurelio Uncini[1]

Department of Information Engineering, Electronics and Telecommunications (DIET),
"Sapienza" University of Rome,
Via Eudossiana 18, 00184 Rome, Italy
simone.scardapane@uniroma1.it

Abstract

One of the main characteristics in many real-world big data scenarios is their distributed nature. In a machine learning context, distributed data, together with the requirements of preserving privacy and scaling up to large networks, brings the challenge of designing fully decentralized training protocols. In this paper, we explore the problem of distributed learning when the features of every pattern are available throughout multiple agents (as is happening, for example, in a distributed database scenario). We propose an algorithm for a particular class of neural networks, known as Random Vector Functional-Link (RVFL), which is based on the Alternating Direction Method of Multipliers optimization algorithm. The proposed algorithm allows to learn an RVFL network from multiple distributed data sources, while restricting communication to the unique operation of computing a distributed average. Our experimental simulations show that the algorithm is able to achieve a generalization accuracy comparable to a fully centralized solution, while at the same time being extremely efficient.

Keywords: Distributed learning, Random Vector Functional-Link, Multiple data sources, Alternating Direction Method of Multipliers

1 Introduction

In the machine learning community, big data is generally associated to the problem of processing large amounts of data, possibly arriving in a continuous fashion [1] and considering real computing constraints [2]. However, real-world big data applications may require to handle data sources that are *distributed* over a network of agents, such as computers in a peer-to-peer (P2P) network [3], automatic trading agents [4], or sensors in a Wireless Sensor Network (WSN) [5]. If the nodes have access to a fusion center (FC), they may trivially transmit all their data to the FC, resulting in a standard centralized learning problem. More in general, however, this

Selection and peer-review under responsibility of the Scientific Programme Committee of INNS-BigData2015

doi:10.1016/j.procs.2015.07.324

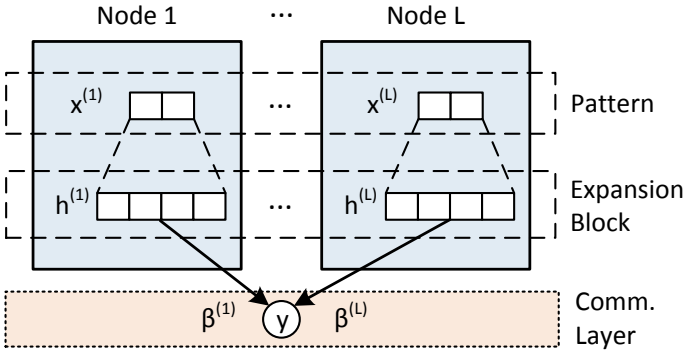


Figure 1: Schematic description of the proposed algorithm. Each node has access to a subset of the global pattern. This local feature vector is projected to a local expansion block, and the overall output is computed by a linear combination of the local expansions, through a suitable communication layer.

solution may not be technologically feasible [6]. Additionally, data may be too large to be efficiently transmitted, privacy concerns may be present, and/or communication may be allowed only over local neighborhoods of each node (as in *ad-hoc* WSNs [5]). Due to this, there is a growing need of fully decentralized strategies for solving the overall learning problem.

In a supervised scenario, data can be distributed in two orthogonal ways. In the case of 'horizontal partitioning' (HP) (also known as 'data-distributed' [7]) each agent in a network possesses a local set of patterns. For example, in distributed music classification over P2P networks [8], each peer has access to its own database of songs. Learning in an HP setting has obtained a large amount of attention recently, including distributed versions of support vector machines [9], neural networks [7], and boosting [10]. Conversely, in the 'vertical partitioning' (VP) scenario the features of each pattern are partitioned over the nodes. A prototypical example of this is found in the field of distributed databases [10], where several organizations possess only a partial view on the overall dataset (e.g., global health records distributed over multiple medical databases). In the centralized case, this is also known as the problem of learning from heterogeneous sources, and it is typically solved with the use of ensemble procedures [11]. However, as we show in our experimental results, in the VP setting naive ensembles over a network tend to achieve highly sub-optimal results, with respect to a fully centralized solution.

In a previous work [7], we presented two distributed protocols for training a specific class of neural networks, known as Random Vector Functional-Link (RVFL), in the HP setting. RVFL networks are composed of a fixed layer of non-linearities (known as the expansion block), whose parameters are randomly assigned at the beginning of the learning process, followed by a trainable linear layer [12]. Since the overall optimization problem can be formulated as a standard linear regression, in [7] it was shown that learning an RVFL network in an HP distributed scenario results in an extremely efficient algorithm, while at the same time keeping the universal approximation capabilities of the centralized case. The aim of this paper is to extend the previous work to the VP scenario. The proposed algorithm, which is summarized in Fig. 1, is made of two components. First, each node computes a local expansion block based on its subset of features. Secondly, the weights of the second layer are learned in a distributed fashion with the use of a standard decentralized optimization routine, known as Alternating Direction Method of Multipliers (ADMM) [13]. Communications between nodes are restricted

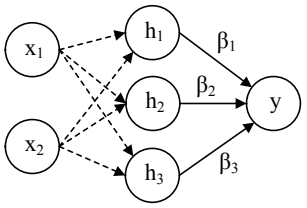


Figure 2: Schematic depiction of an RVFL architecture with two inputs, three hidden nodes, and one output. Fixed connections are shown as dashed lines, whilst trainable connections as fixed lines.

to the computation of averages, which is a standard primitive in any network technology. As an example, in this paper we employ the average consensus (AC) mechanism [14], which is routinely employed in WSNs. Our experimental results will show that this strategy, apart from being efficient, can perform comparably to a centralized solution, and strongly outperforms distributed ensemble techniques.

The rest of the paper is organized as follows. In Section 2 we details the basic mathematical formulation of RVFL networks. Then, in Section 3 we introduce our distributed training protocol in the VP scenario. Section 4 presents a wide set of experimental results, while Section 5 concludes the paper.

2 Random Vector Functional-Link Networks

Given a d-dimensional input vector $\mathbf{x} \in \mathbb{R}^d$, the output of a one-dimensional RVFL network is given by:

$$f(\mathbf{x}) = \sum_{m=1}^{B} \beta_m h_m(\mathbf{x}) = \boldsymbol{\beta}^T \mathbf{h}(\mathbf{x}) \,, \tag{1}$$

where each $h_m(\cdot)$ is a fixed non-linear function known as base or functional link [15]. This is shown schematically in Fig. 2. In an RVFL network, the internal parameters of the bases are generally assigned randomly, from a fixed probability distribution. Eq. (1) can be extended trivially to the case of a multidimensional output [7]. A classical example of base, which is employed in this work, is the sigmoidal squashing function:

$$h(\mathbf{x}) = \frac{1}{1 + \exp\left\{-\mathbf{a}^T\mathbf{x} + b\right\}} \,, \tag{2}$$

where the parameters $\mathbf{a}$ and b in Eq. (2) are assigned randomly at the beginning of the learning process. Eq. (1) can also be extended with the use of input-to-output weights, as in the original formulation [12], and was popularized recently under the name Extreme Learning Machine (ELM) [16]. Given a set of N samples of the desired function, called the training set, that is $S = \left\{\mathbf{x}^{(i)}, y^{(i)}\right\}$, $i = 1 \dots N$, we want to learn a set of weights $\boldsymbol{\beta}$ such that the error over unseen samples is lowest. Defining the hidden matrix $\mathbf{H} = \left[\mathbf{h}(\mathbf{x}^{(1)}) \dots \mathbf{h}(\mathbf{x}^{(N)})\right]^T$ and the output vector $\mathbf{y} = \left[y^{(1)} \dots y^{(N)}\right]^T$, we can formulate the learning problem as a standard regularized linear regression:

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta} \in \mathbb{R}^B}{\arg\min} \frac{1}{2} \left\|\mathbf{H}\boldsymbol{\beta} - \mathbf{y}\right\|_2^2 + \frac{\lambda}{2} \left\|\boldsymbol{\beta}\right\|_2^2 \,, \tag{3}$$

whose solution can be computed in closed form as:

$$\boldsymbol{\beta}^* = \left(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}\right)^{-1}\mathbf{H}^T\mathbf{Y}\,. \tag{4}$$

3 Distributed RVFL Networks

3.1 Training phase

In the distributed scenario, we suppose that the training data is not available on a single processor, but it is distributed throughout a network of $L \in \mathbb{N}$ interconnected nodes. The connectivity between the nodes can be fully described by a real-valued $L \times L$ matrix $\mathbf{C} \in \mathbb{R}^{L\times L}$, such that $C_{ij} > 0$ if nodes i and j are connected, 0 otherwise. In particular, in the VP scenario we suppose that the k-th agent has access to a subset $\mathbf{x}_k$ of features, such that:

$$\mathbf{x} = [\mathbf{x}_1|\ldots|\mathbf{x}_L]$$

The main problem for the distributed training of an RVFL network in this setting is that the computation of any functional link in Eq. (1) requires knowledge of the full sample. However, as we stated in Section 1, we would like to avoid exchange of data patterns, due to both size and privacy concerns. To this end, we approximate model (1) by considering local expansion blocks:

$$f(\mathbf{x}) = \sum_{k=1}^{L} \left(\boldsymbol{\beta}_k\right)^T \mathbf{h}_k(\mathbf{x}_k)\,. \tag{5}$$

In this way, each term $\mathbf{h}_k(\mathbf{x}_k)$ can be computed locally. Input vectors and expansion blocks may have different lengths at every node, depending on the application and on the local computational requirements. This is shown pictorially in Fig. 1. The overall optimization problem becomes:

$$\arg\min_{\boldsymbol{\beta}} \frac{1}{2}\left\|\sum_{k=1}^{L}\mathbf{H}_k\boldsymbol{\beta}_k - \mathbf{y}\right\|_2^2 + \frac{\lambda}{2}\sum_{k=1}^{L}\|\boldsymbol{\beta}_k\|_2^2\,, \tag{6}$$

where $\mathbf{H}_k$ denotes the hidden matrix computed at the k-th node, such that $\mathbf{H} = [\mathbf{H}_1|\ldots|\mathbf{H}_L]$. We solve this problem in a decentralized fashion using the ADMM optimization procedure [13]. To this end, we consider the equivalent optimization problem:

$$\begin{aligned} \underset{\boldsymbol{\beta}}{\text{minimize}} \quad & \frac{1}{2}\left\|\sum_{k=1}^{L}\mathbf{z}_i - \mathbf{y}\right\|_2^2 + \frac{\lambda}{2}\sum_{k=1}^{L}\|\boldsymbol{\beta}_k\|_2^2 \\ \text{subject to} \quad & \mathbf{H}_k\boldsymbol{\beta}_k - \mathbf{z}_k = \mathbf{0},\ k = 1\ldots L\,. \end{aligned} \tag{7}$$

where we introduced local variables $\mathbf{z}_k = \mathbf{H}_k\boldsymbol{\beta}_k$. The augmented Lagrangian of this problem is given by:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\beta}_k, \mathbf{z}_k, \mathbf{t}_k) = \frac{1}{2}\left\|\sum_{k=1}^{L}\mathbf{z}_k - \mathbf{y}\right\|_2^2 + \frac{\lambda}{2}\sum_{k=1}^{L}\|\boldsymbol{\beta}_k\|_2^2 + \\ + \sum_{k=1}^{L}\mathbf{t}_k^T\left(\mathbf{H}_k\boldsymbol{\beta}_k - \mathbf{z}_i\right) + \frac{\rho}{2}\sum_{k=1}^{L}\|\mathbf{H}_k\boldsymbol{\beta}_k - \mathbf{z}_i\|_2^2\,, \end{aligned} \tag{8}$$

where $\mathbf{t}_k$ are the Lagrange multipliers, $\rho \in \mathbb{R}^+$ is a regularization factor, and the last term is added to ensure convergence. The solution to problem (6) can computed by iterating the following updates [13]:

$$\boldsymbol{\beta}_k[n+1] = \arg\min_{\boldsymbol{\beta}_k} \mathcal{L}(\boldsymbol{\beta}_k, \mathbf{z}_k[n], \mathbf{t}_k[n]) , \tag{9}$$

$$\mathbf{z}_k[n+1] = \arg\min_{\mathbf{z}_k} \mathcal{L}(\boldsymbol{\beta}_k[n+1], \mathbf{z}_k, \mathbf{t}_k[n]) , \tag{10}$$

$$\mathbf{t}_k[n+1] = \mathbf{t}_k[n] + \rho \left(\mathbf{H}_k \boldsymbol{\beta}_k[n+1] - \mathbf{z}_k[n+1]\right) . \tag{11}$$

Following the derivation in [13, Section 8.3], and computing the gradient terms (which we omit for lack of space), the final updates can be expressed as:

$$\boldsymbol{\beta}_k[n+1] = \left(\frac{\lambda}{\rho}\mathbf{I} + \mathbf{H}_k^T \mathbf{H}_k\right)^{-1} \mathbf{H}_k^T \left(\mathbf{H}_k \boldsymbol{\beta}_k[n] + \overline{\mathbf{z}}[n] - \overline{\mathbf{H}\boldsymbol{\beta}}[n] - \mathbf{t}[n]\right) , \tag{12}$$

$$\overline{\mathbf{z}}[n+1] = \frac{1}{L+\rho} \left(\mathbf{y} + \overline{\mathbf{H}\boldsymbol{\beta}}[n+1] + \mathbf{t}[n]\right) , \tag{13}$$

$$\mathbf{t}[n+1] = \mathbf{t}[n] + \overline{\mathbf{H}\boldsymbol{\beta}}[n+1] - \overline{\mathbf{z}}[n+1] , \tag{14}$$

where we defined the averages $\overline{\mathbf{H}\boldsymbol{\beta}}[n] = \frac{1}{L}\sum_{k=1}^{L} \mathbf{H}_k \boldsymbol{\beta}_k[n]$, and $\overline{\mathbf{z}}[n] = \sum_{k=1}^{L} \mathbf{z}_k[n]$. Additionally, the variables $\mathbf{t}_k$ can be shown to be equal between every node [13], so we removed the subscript. Convergence of the algorithm can be tracked locally by computing the residual:

$$\mathbf{r}_k[n] = \mathbf{H}_k^T \boldsymbol{\beta}_k[n] - \mathbf{z}_k[n] . \tag{15}$$

It can be shown that, for the iterations defined by Eqs. (12)-(14), $\|\mathbf{r}_k[n]\|_2 \to 0$ as $n \to +\infty$, with the solution converging asymptotically to the solution of problem (6).

3.2 Distributed computation of the average

From the following section, we see that the only communication required by our training protocol is the computation of the average $\overline{\mathbf{H}\boldsymbol{\beta}}[n]$. In a real-world application, the specific implementation of this step will depend on the actual details of the communication layer, e.g., whether broadcast or point-to-point messaging is available; whether the topology of the network is fixed or time-varying, and so on. As an example of general purpose protocol to implement the step, we briefly discuss here the decentralized average consensus (DAG) [14], which is widely employed over WSNs. Still, we stress that this is only one of the many possible choices (e.g. the Push-Sum protocol in P2P networks [17]). DAG 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 [14].

Define $\mathbf{w}_k[n,0] = \mathbf{H}_k \boldsymbol{\beta}_k[n,0]$, where we introduced a second time index to denote the internal iteration for computing the average. In a DAG algorithm, this value is iteratively refined at every node as:

$$\mathbf{w}_k[n,j+1] = C_{kk}\mathbf{w}_k[n,j] + \sum_{l \in \mathcal{N}_k} C_{kl}\mathbf{w}_l[n,j] , \tag{16}$$

where C_{kl} denotes the (k,l)-th entry of the connectivity matrix $\mathbf{C}$. Practically, every node iteratively computes a weighted average of the value of its neighbors. Under suitable choices of the connectivity matrix (see [14]), the iteration defined by Eq. (16) converges to the global

Table 1: General description of the datasets. The values in the last two columns are taken from [7].

Dataset name	Features	Instances	Desired output	Classes	Optimal B	Optimal λ
Garageband	44	1856	Genre recognition	9 classes	200	2^{-3}
Sylva	216	14394	Forest Type	2 classes	450	2^{-5}
G50C	50	550	Gaussian of origin	2 classes	500	2^{3}

average at every node. As an example, in this work we consider the so-called 'max-degree' weights given by:

$$C_{kj} = \begin{cases} \frac{1}{d+1} & \text{if } k \in \mathcal{N}_j \\ 1 - \frac{d_k}{d+1} & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}, \tag{17}$$

where d_k is the degree of node k, and d is the degree of the network.

3.3 Prediction phase

After training, every node has access to its own local mapping $\mathbf{h}_k(\cdot)$, and to its subset of coefficients $\boldsymbol{\beta}_k$. Differently from the HP scenario, when the agents require a new prediction, the overall output defined by Eq. (5) has to be computed in a decentralized fashion. Once again, this part will depend on the actual communication layer available to the agents. As an example, it is possible to run the DAG protocol described in the previous section over the values $(\boldsymbol{\beta}_k)^T \mathbf{h}_k(\mathbf{x}_k)$, such that every node obtain a suitable approximation of $\frac{1}{L} f(\mathbf{x})$. For smaller networks, it is possible to compute an Hamiltonian cycle between the nodes [5]. Once the cycle is known to the agents, they can compute Eq. (5) by forward-propagating the partial sums up to the final node of the cycle, and then back-propagating the result. Clearly, many other choices are possible, depending on the network.

4 Experimental Results

4.1 Experimental setup

In this section, we present an experimental validation of the proposed algorithm on three classification tasks, which were already employed on previous works on RVFL networks [7, 18]. A schematic description of the three datasets is provided in Table 1. The last two columns in Table 1 detail the optimal RVFL parameters for the datasets in the centralized case, computed via an extensive grid search procedure detailed in [7, Table 1]. In our first set of experiments, we consider networks of 8 agents, whose connectivity is randomly generated such that every pair of nodes has a 60% probability of being connected, with the only global requirement that the overall network is connected. The input features are equally partitioned through the nodes, i.e., every node has access to roughly $d/8$ features, where d is the dimensionality of the dataset (fourth column in Table 1). We compare the following algorithms:

Centralized RVFL (C-RVFL): this corresponds to the case where a fusion center is available, collecting all local datasets and solving directly Eq. (3). Settings for this model are the optimal ones in Table 1.

Table 2: Misclassification error and training time for the four algorithms. Results are averaged over the 8 different nodes of the network. Standard deviation is provided between brackets.

Dataset	Algorithm	Misclassification error [%]	Training time [secs.]
Garageband	C-RVFL	41.32 (±1.24)	0.03 (±0.02)
	L-RVFL	82.79 (±3.82)	0.01 (±0.01)
	ENS-RVFL	61.01 (±1.97)	0.01 (±0.01)
	DIST-RVFL	41.34 (±1.34)	2.35 (±0.58)
Sylva	C-RVFL	1.18 (±0.13)	0.44 (±0.06)
	L-RVFL	49.80 (±36.35)	0.05 (±0.02)
	ENS-RVFL	6.04 (±0.12)	0.06 (±0.02)
	DIST-RVFL	1.22 (±0.15)	1.94 (±0.40)
G50C	C-RVFL	5.80 (±1.19)	0.05 (±0.02)
	L-RVFL	49.51 (±6.37)	0.01 (±0.01)
	ENS-RVFL	10.98 (±2.32)	0.01 (±0.01)
	DIST-RVFL	5.80 (±1.37)	0.38 (±0.16)

Local RVFL (L-RVFL): this is a naive implementation, where each node trains a local model with its own dataset, and no communication is performed. Accuracy of the models is then averaged throughout the L nodes. As a general settings, we employ the same regularization coefficient for every node as C-RVFL, and $B_k = \lceil B/8 \rceil$ expansions in every agent.

Ensemble RVFL (ENS-RVFL): this corresponds to the ensemble approach described in Section 1. As for L-RVFL, during the training phase every node trains a local model with its own dataset. In the testing phase, the nodes agree on a single class prediction by taking a majority vote over their local predictions. Parameters are the same as for L-RVFL.

Distributed RVFL (DIST-RVFL): this is trained using the distributed protocol of Section 3. Settings are the same as L-RVFL, while for the ADMM we set $\rho = 0.1$ and a maximum number of 200 iterations.

To compute the misclassification rate, we perform a 3-fold cross-validation on the overall dataset, and repeat the procedure 15 times. All algorithms have been implemented in MATLAB 2013a, using the Lynx toolbox.[1] Detailed instruction for repeating the simulations are available on the corresponding author's website.[2]

4.2 Results and discussion

Results of the experiments are presented in Table 2. It can be seen that, despite we approximate the global expansion block of C-RVFL using L distinct local expansions, this has minimal or no impact on the global solution. In fact, DIST-RVFL is able to achieve performance comparable to C-RVFL in all three datasets, while the ensemble approach is performing relatively poorly: it has a 20%, 5% and 5% increase in error respectively in each dataset. This shows that the

[1] `https://github.com/ispamm/Lynx-Toolbox`

[2] `http://ispac.diet.uniroma1.it/scardapane/software/lynx/heterogeneous-sources/`

relatively common approach of averaging over the local models may be highly sub-optimal in practical situations.

As a reference, in Table 2 we also provide the average training time spent at every node. However, we note that in our experiments the network was simulated in a serial architecture, removing all communication costs. Clearly, a practical analysis of this point would require knowledge of the communication layer, which goes beyond the scope of the current paper. Still, we can see from the fourth column of Table 2 that the proposed algorithm requires an acceptable computational time for performing the 200 iterations, since the matrix inversions in Eq. (12) can be pre-computed at the beginning of the training process. Additionally, we add that the training time of DIST-RVFL can be greatly reduced in practice by the implementation of an efficient stopping criterion [13, 7].

Finally, we show the evolution of the misclassification error for DIST-RVFL and ENS-RVFL when varying the size of the network from $L = 4$ to $L = 12$. Results of this experiment are given in Fig. 3 (a)-(c). Settings are kept fixed with respect to the previous experiment, while the features are equally partitioned as before (hence, for smaller networks each node has access to a larger subset of features). Performance of C-RVFL is given as a comparison with a dashed black line. As expected, we see that, although the behavior of ENS-RVFL strongly depends on the number of nodes in the network, DIST-RVFL is resilient to such change, always approximating very well the centralized performance. It is also interesting to note that the behavior of ENS-RVFL is not always monotonically increasing, as is shown in Fig. 3-(c), possibly due to its ensembling characteristics and to the artificial nature of the G50C dataset.

5 Conclusions

In this paper, we proposed a distributed training algorithm for RFVL networks, in the scenario when each agent in a network has access to a subset of features of the original pattern. The algorithm is based on the ADMM optimization procedure, and restricts communications to the distributed computation of averages. As an example of protocol for achieving this, we described the DAG routine, which lends itself easily to large, unstructured networks. Our simulation results show that the proposed algorithm achieves generalization accuracy comparable to that of a centralized solution, with a small overhead in training time. Future work will include the test of the algorithm on a realistic distributed scenario, such as a problem of distributed traffic prediction. Additionally, we plan to extend the basic framework to more general networks, including networks with time-changing topologies, faulty message passings, and asynchronous communications.

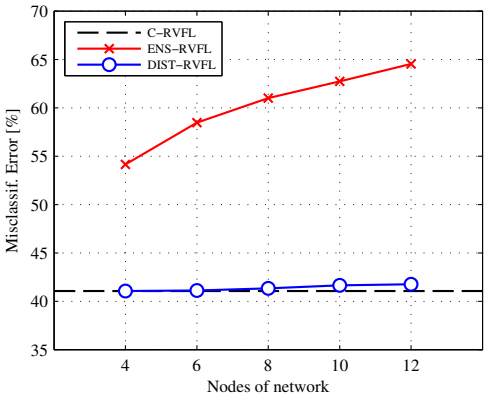
References

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

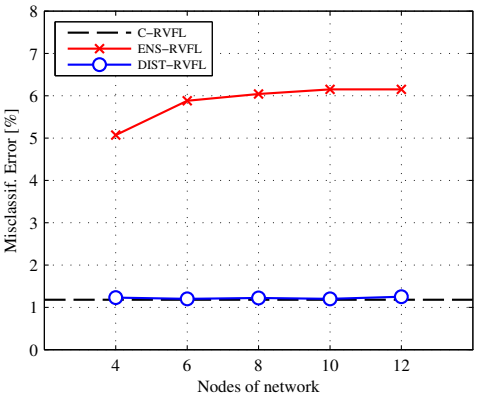
[2] E. Baccarelli, N. Cordeschi, A. Mei, M. Panella, M. Shojafar, J. Stefa, Energy-efficient Dynamic Traffic Offloading and Reconfiguration of Networked Datacenters for Big Data Stream Mobile Computing: Review, Challenges, and a Case Study, IEEE Network Magazine (to appear).

[3] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, H. Kargupta, Distributed data mining in peer-to-peer networks, IEEE Internet Computing 10 (4) (2006) 18–26.

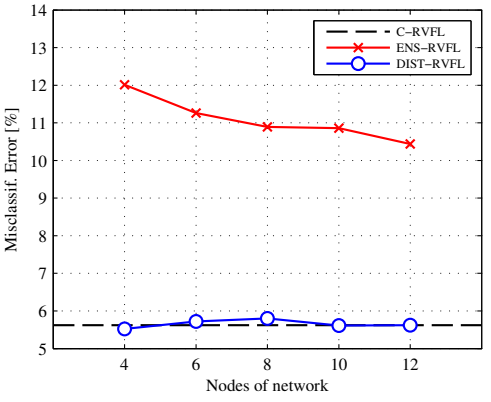
[4] M. Panella, F. Barcellona, R. D'Ecclesia, Forecasting Energy Commodity Prices Using Neural Networks, Advances in Decision Sciences 2012 (2012) 1–26.



(a) Dataset Garageband



(b) Dataset Sylva



(c) Dataset G50C

Figure 3: Evolution of the error for DIST-RVFL and ENS-RVFL when varying the size of the network from $L = 4$ to $L = 12$.

[5] J. B. Predd, S. R. Kulkarni, H. V. Poor, Distributed learning in wireless sensor networks, IEEE Signal Processing Magazine 23 (4) (2006) 56–69.

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

[7] S. Scardapane, D. Wang, M. Panella, A. Uncini, Distributed learning for Random Vector Functional-Link networks, Information Sciences 301 (2015) 271 – 284.

[8] S. Scardapane, R. Fierimonte, D. Wang, M. Panella, A. Uncini, Distributed Music Classification using Random Vector Functional-Link Nets, in: 2015 INNS/IEEE International Joint Conference on Neural Networks (IJCNN'15), 2015.

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

[10] A. Lazarevic, Z. Obradovic, The distributed boosting algorithm, in: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2001, pp. 311–316.

[11] W. Li, D. Wang, T. Chai, Flame image-based burning state recognition for sintering process of rotary kiln using heterogeneous features and fuzzy integral, IEEE Transactions on Industrial Informatics 8 (4) (2012) 780–790.

[12] Y.-H. Pao, G.-H. Park, D. J. Sobajic, Learning and generalization characteristics of the random vector functional-link net, Neurocomputing 6 (2) (1994) 163–180.

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

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

[15] D. Comminiello, M. Scarpiniti, L. Azpicueta-Ruiz, J. Arenas-Garcia, A. Uncini, Functional Link Adaptive Filters for Nonlinear Acoustic Echo Cancellation, IEEE Transactions on Audio, Speech, and Language Processing 21 (7) (2013) 1502–1512.

[16] S. Scardapane, D. Comminiello, M. Scarpiniti, A. Uncini, Online Sequential Extreme Learning Machine With Kernels, IEEE Transactions on Neural Networks and Learning Systems (under press)doi:10.1109/TNNLS.2014.2382094.

[17] C. Hensel, H. Dutta, GADGET SVM: a gossip-based sub-gradient svm solver, in: International Conference on Machine Learning (ICML), Numerical Mathematics in Machine Learning Workshop, 2009.

[18] S. Scardapane, D. Comminiello, M. Scarpiniti, A. Uncini, Music classification using extreme learning machines, in: 2013 8th International Symposium on Image and Signal Processing and Analysis (ISPA'13), 2013, pp. 377–381.