ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8.
Available from http://www.i6doc.com/en/.

Distributed learning algorithm for feedforward neural networks

Oscar Fontenla-Romero, Beatriz Pérez-Sánchez, Bertha Guijarro-Berdiñas
Diego Rego-Fernández

Department of Computer Science, University of A Coruña, Spain

Abstract. With the appearance of huge data sets new challenges have risen regarding the scalability and efficiency of Machine Learning algorithms, and both distributed computing and randomized algorithms have become effective ways to handle them. Taking advantage of these two approaches, a distributed learning algorithm for two-layer neural networks is proposed. Results demonstrate a similar accuracy when compared to an equivalent non-distributed approach whilst providing some advantages that make it especially well-suited for Big Data sets: over 50% savings in computational time; low communication and storage cost; no hyperparameters to be tuned; it allows online learning and it is privacy-preserving.

1 Introduction

Machine learning (ML) algorithms have achieved much success in the learning of small-scale problems. Recently, however, with the appearance of huge data sets, new challenges are raised regarding their scalability and efficiency. One of the most promising lines of research for data learning is distributed computing but only a few distributed learning algorithms have been proposed. Notable attempts can be found for Support Vector Machines (SVMs) [1, 2]. However, there have been very few studies conducted on the paradigm of neural networks. Randomized algorithms have become an effective way to handle large scale data sets. In this sense, Scardapane et al. [3] have recently proposed two learning algorithms for Random Vector Functional-Link (RVFL) networks. RVFLs are feedforward neural networks with a single hidden layer whose input weights are randomly chosen and fixed in advance before training process. Despite this simplification, they possess universal approximation capabilities providing a sufficiently large set of basis functions [4]. The key idea of the approach is to let all nodes train a local model using a subset of the training data and then obtain output weights of the master learned model. Two techniques for obtaining the common output weights can be applied: decentralized average consensus [5] and alternating direction method of multipliers [6]. These algorithms allow the nodes to obtain a single model, whose testing performance is similar to that obtained by a centralized model. Another variant for RVFL was proposed by Alhamdoosh [7] who developed a fast solution on building neural network ensembles. The hidden layer parameters of RVFL networks are initialized randomly and then the least square method with negative correlation learning scheme is employed to analytically calculate the output weights of these networks. Another emerging approach is the Extreme Learning Machine (ELM) which provides unified solutions to generalized feedforward networks. ELM theories [8, 9] show that hidden

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8.
Available from http://www.i6doc.com/en/.

neurons are important but can be randomly generated having both universal approximation and classification capabilities. Taking into account the benefits of distributed and randomized learning to deal with massive data applications, we propose a new distributed and learning algorithm for two-layer neural networks.

2 Background

For the sake of comprehension, this section contains some previous results [10] that will be used as foundation of this work. Specifically, we present a supervised algorithm that obtains the optimal weights of a feedforward neural network with a single layer (no hidden layers) and nonlinear output functions. In contrast to some other well-known algorithms it *backpropagates* the network's *desired output* signal instead of the *error* between the desired and the real outputs. In Figure 1 this process is depicted graphically. For each neuron j with nonlinear output function f_j, and for each input pattern $\mathbf{x}_s$, the corresponding desired output d_{js} is backpropagated using the inverse of the output function f_j^{-1}. Afterwards, the error is minimized between the internal network value z_{js} at the output of the summation and $f_j^{-1}(d_{js})$.

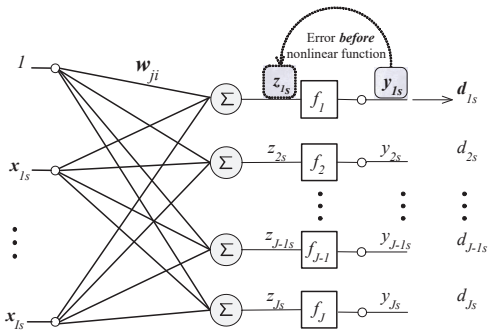


Fig. 1: Architecture of a single-layer feedforward neural network.

The details of this learning algorithm can be found in [10], in which a theorem is demonstrated stating that the minimization of the MSE between $\mathbf{d}$ and $\mathbf{y}$ at the output of the nonlinearity is equivalent, up to first Taylor order, to minimizing the MSE before the nonlinearity. Mathematically, it can be written as:

$$\min_W \; E[(\mathbf{d}-\mathbf{y})^T(\mathbf{d}-\mathbf{y})] \approx \min_W \; E[(f'(\bar{\mathbf{d}})\cdot\bar{\epsilon})^T(f'(\bar{\mathbf{d}})\cdot\bar{\epsilon})] \tag{1}$$

where $\bar{\mathbf{d}} = f^{-1}(\mathbf{d})$, $\bar{\epsilon} = \bar{\mathbf{d}} - \mathbf{z}$ and $(\cdot)$ denotes the element-wise Hadamard product of vectors. As a consequence, it is a restriction for the nonlinear output functions f_j to be inverse and derivative. In the following, we centre our attention on only one neuron ($J = 1$) in order to avoid a cumbersome derivation (to solve the full layer of neurons the process has to be applied identically for every neuron). Therefore, using this theorem, the weight vector $\mathbf{w} \in \mathbb{R}^{(I+1)}$ of a neural network

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8. Available from http://www.i6doc.com/en/.

has to be a stationary point of the right hand side of equation (1). Taking derivatives of this expression and equating to 0, a system of linear equations $\mathbf{A}\mathbf{w} = \mathbf{b}$ is obtained where $\mathbf{A}$ and $\mathbf{b}$ are defined as:

$$\mathbf{A} = \sum_{t=1}^{S} \mathbf{x}_t \mathbf{x}_t^T f'^2(\bar{d}_t) \quad ; \quad \mathbf{b} = \sum_{t=1}^{S} \bar{d}_t \mathbf{x}_t f'^2(\bar{d}_t) \tag{2}$$

Once $\mathbf{A}$ and $\mathbf{b}$ are known, the network's weights $\mathbf{w}$ can be easily calculated. This approach is applicable in both batch and incremental learning environments as in equation (2) $\mathbf{A}$ and $\mathbf{b}$ are a summation on S. Therefore, we can save $\mathbf{A}_t$ and $\mathbf{b}_t$ obtained at time t, and when new information is supplied up to time $t+p$ we can incrementally construct $\mathbf{A}_{t+p}$ and $\mathbf{b}_{t+p}$ using equation (2).

3 Distributed learning for Two-layer Neural Networks

In this work, we take advantage of the formulation cited in Section 2 to propose a distributed and fast training algorithm for two-layer feedforward neural networks. Our proposal will follow a typical scenario of the Map/Reduce paradigm consisting in learning separately from each of the distributed data sets (ideally, in parallel) and posterior model integration. Based on these premises, and for a scenario with N learning sites or *nodes*, Algorithm 1 is proposed. It is worth remembering that we centred our explanations on one output neural networks, for the sake of simplicity, and thus $\mathbf{w}_2$ corresponds to a vector. However, considering several outputs for multiclass problems is straightforward.

4 Experimental results

The method was tested on five public data sets whose characteristics are summarized in Table 1. The first two data sets were obtained from the Data Mining Institute of the University of Wisconsin[1] and the rest from the UCI Machine Learning Repository[2]. All the data sets are binary classification problems. These data sets were selected to consider cases of different sizes, from a small data set, such as the Brightdata, to large data sets like Higgs Bosons. For all the experiments, the sigmoid function was used as transfer function for the hidden layer while a linear function was used for the output layer. The weight matrix of the first layer ($\mathbf{W}_1$) was randomly established using the Nguyen-Widrow algorithm. The number of hidden neurons was varied between 8 and 1024, except for the Higgs Bosons that was varied between 16 and 4096 in order to obtain better results. Simulations were carried out using a Intel Xeon W3550 processor with 3.07GHz clock speed.

In the experiments, we compared the results obtained using 1 node (as a base case) and 4 nodes (as a distributed case). For the Higgs data set, due to its large size, 200 nodes were employed. In order to obtain more reliable results we applied

[1] http://research.cs.wisc.edu/dmi
[2] https://archive.ics.uci.edu/ml

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8.
Available from http://www.i6doc.com/en/.

Algorithm 1 Distributed learning algorithm

Input: N local data sets $(D_1, D_2, \ldots, D_N)$. Each data set D_n comprises both inputs $\mathbf{X}_n = \{\mathbf{x}_{n1}, \mathbf{x}_{n2}, \ldots, \mathbf{x}_{nS}\}$ and desired outputs $\mathbf{D}_n = \{\mathbf{d}_{n1}, \mathbf{d}_{n2}, \ldots, \mathbf{d}_{nS}\}$
Output: The optimal weight matrix/vector $\mathbf{W}_1, \mathbf{w}_2$ of the two-layer neural network *global* model.

A starting node, arbitrarily established at node n=1, randomly generates a matrix of weights $\mathbf{W}_1$ for the first layer of the neural network.

Broadcast this $\mathbf{W}_1$ matrix to every learning node to be used as the first layer of weights of their local neural networks.

At every learning node n=1, . . . , N

Train locally its neural network: propagate the inputs to the second layer using $\mathbf{W}_1$. For this second layer, compute $\mathbf{A}$ and $\mathbf{b}$ using Equation 2 needed to obtain $\mathbf{w}_2$.

Send matrix $\mathbf{A}$ and vector $\mathbf{b}$ to the starting node (n=1).

end.

At node 1, set the initial accumulative matrix $\mathbf{A}^* = \mathbf{A}_1$ and the initial accumulative vector $\mathbf{b}^* = \mathbf{b}_1$. Afterwards, from n=2, . . . ,N using the incremental properties of $\mathbf{A}$ and $\mathbf{b}$

Compute the accumulative matrix $\mathbf{A}^* = \mathbf{A}^* + \mathbf{A}_n$.

Compute the accumulative vector $\mathbf{b}^* = \mathbf{b}^* + \mathbf{b}_n$.

end

At this moment, matrix $\mathbf{A}^*$ and vector $\mathbf{b}^*$ at the starting node $n = 1$ contains the same information as if they were calculated using a central, non distributed, approach. Thus, at the starting node $n = 1$ compute the optimal weights $\mathbf{w_2}$ for the second layer of the neural network solving the system of linear equations with $\mathbf{A}^*$ and $\mathbf{b}^*$.

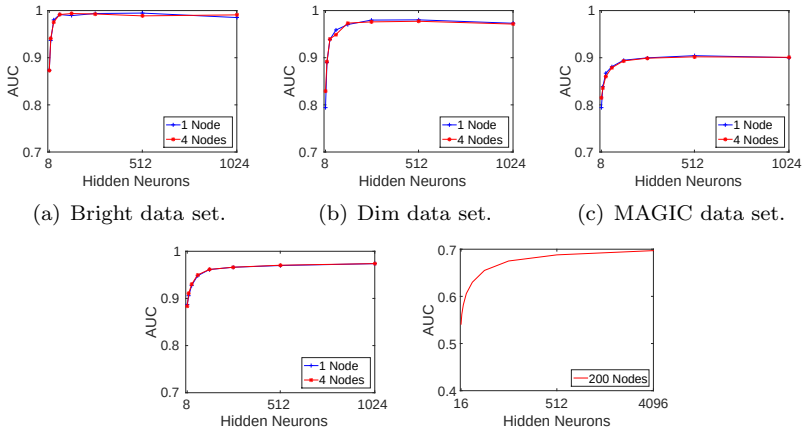
Broadcast the final global model $\mathbf{W}_1, \mathbf{w}_2$ to every node, if needed.

Data set name	Features	Instances
Bright data	14	2,462
Dim data	14	4,192
MAGIC Gamma Telescope	10	19,020
MiniBooNE particle identification	50	130,064
Higgs Bosons	28	11,000,000

Table 1: Description of the data sets

10-fold cross-validation. The Area Under de ROC Curve (AUC) has been used to determine an operational point, specifically, to establish the upper number of hidden neurons to use in the study. Figure 2 contains the results varying the number of hidden neurons. As can be seen there are no significant differences between results of 1 and 4 nodes due to the distribution of data. Figure 3 shows the mean CPU time required for the whole learning process again comparing 1 vs 4 nodes. As expected, the distributed approach brings a considerable savings in computational time. Specifically, focusing on the interesting operational points of the AUC, in all experiments a time reduction over 50% is achieved with 128 hidden neurons that grows over 60% when 256 or 512 hidden neurons are used.

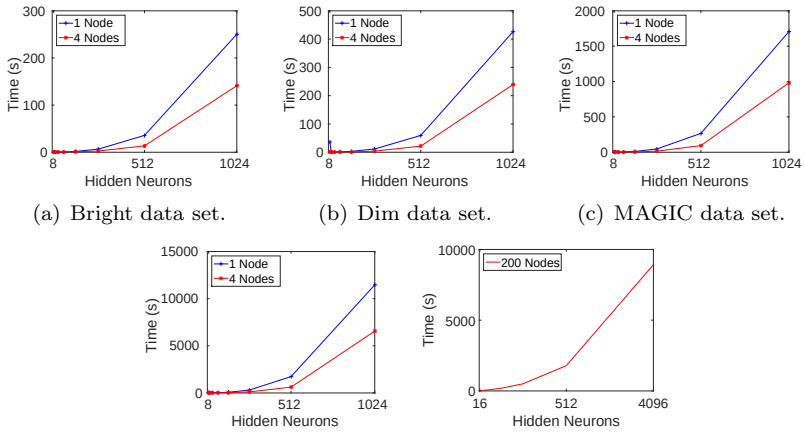
ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8.
Available from http://www.i6doc.com/en/.



(a) Bright data set. (b) Dim data set. (c) MAGIC data set.

(d) MiniBooNE data set. (e) Higgs data set.

Fig. 2: Mean test AUC vs the number of hidden neurons.



(a) Bright data set. (b) Dim data set. (c) MAGIC data set.

(d) MiniBooNE data set. (e) Higgs data set.

Fig. 3: Mean CPU time (in seconds) vs the number of hidden neurons.

5 Conclusions

The presented approach has some advantages that make it specially well-suited for learning from Big Data sets:

- It does not contain any hyperparameter to be tuned for a good performance, unlike most of the classic ML methods. The usual way for hyperparameter optimization has been grid search which is simply an exhaustive searching through a manually specified subset of the hyperparameter space. This is specially tedious or simply unaffordable for huge data sets.

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8. Available from http://www.i6doc.com/en/.

- Local learning is very fast, as $\mathbf{W_1}$ is obtained randomly and $\mathbf{w_2}$ is obtained by solving a system of linear equations.
- Due to the incremental capacity of the one-layer neural network learning algorithm, the local training step at each node can be further parallelized by dividing the dataset through multiple processors, cores, GPUs, etc. and finally obtaining $\mathbf{A}$ and $\mathbf{b}$ in accumulative way. This would speed up local learning.
- This incremental capacity also provides a solution when all training data is not available from the beginning but come in a sequential order. The model can be updated quickly with future data without the need to be completely retrained.
- During learning, only parameters $\mathbf{A}$ and $\mathbf{b}$ are shared through the network thus avoiding moving raw data among nodes. This is a remarkable property as: a) it is privacy-preserving, a critical aspect in many real applications, and b) it minimizes communication and storage costs.

6 Acknowledgements

This work has been supported in part by the Secretaría de Estado de Investigación of the Spanish Government (Grant TIN2015-65069-C2-1-R), and Xunta de Galicia (Grant GRC2014/035) with the European Union FEDER funds.

References

[1] A. Navia-Vazquez and E. Parrado-Hernandez. Distributed support vector machines. *IEEE T Neur Net Lear*, 17(4):1091–1097, 2006.

[2] P. A. Forero, A. Cano, and G. B. Giannakis. Consensus-based distributed support vector machines. *J Mach Learn Res*, 11:1663–1707, 2010.

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

[4] B. Igelnik and Y.-H. Pao. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE T Neur Net Lear*, 6(6):1320–1329, 1995.

[5] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. In *P IEEE*, volume 95, pages 215–233, 2007.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–22, 2011.

[7] M. Alhamdoosh and D. Wang. Fast decorrelated neural network ensembles with random weights. *Inform Sciences*, 264:104–117, 2014.

[8] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70:489–501, 2006.

[9] X. Bi, X. Zhao, G. Wang, P. Zhang, and C. Wang. Distributed extreme learning machine with kernels based on mapreduce. *Neurocomputing*, 149(A3):456–463, 2015.

[10] O. Fontenla-Romero, B. Guijarro-Berdiñas, B. Pérez-Sánchez, and A. Alonso-Betanzos. A new convex objective function for the supervised learning of single-layer neural networks. *Pattern Recogn*, 43(5):1984–1992, 2010.