25th Italian Workshop on Neural Networks (Vietri sul Mare)

A Comparison of Consensus Strategies for Distributed Learning of Random Vector Functional-Link Networks

Fierimonte R., Scardapane S., Panella M. and Uncini A.

Overview

Distributed learning

Distributed learning (DL) is the problem of inferring a function whenever the training data is distributed throughout a network of agents.

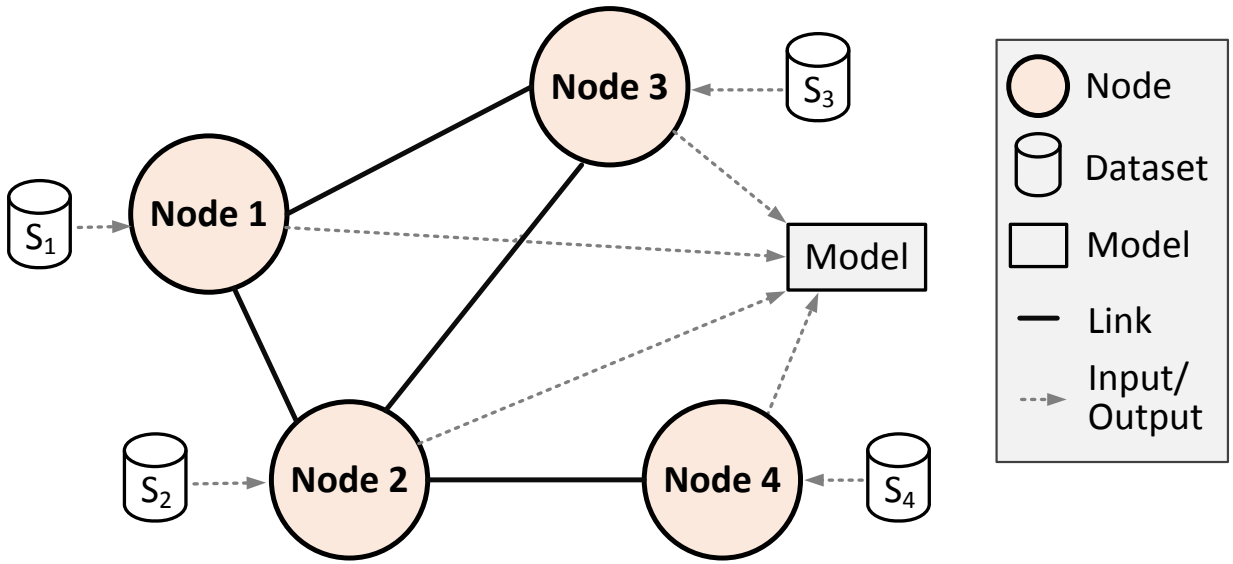


Figure : Schematic depiction of a DL setting.

Requirements for DL

In the most general setting, a DL algorithm should satisfy the following requirements:

Privacy The algorithm should not require the exchange of training examples between nodes.

Communication Communication must be restricted to the neighborhood of a node (*one-hop communication*).

Scalability It should scale well to large networks, without requiring a specific topology.

Online It can be applied to a sequential setting where data arrives continuously.

Learning by consensus

Learning by consensus (LBC) was proposed as a strategy for turning iterative learning algorithms into DL algorithms. It works in two steps:

1. Each agent updates its local classifier (e.g. with a gradient descent step).
2. The resulting local functions are averaged in a distributed fashion with the average consensus (AC) protocol.

The LBC framework can be used in the sequential setting without modifications. In this work we investigate the following question:

How does the setting of the AC procedure influences the training convergence?

Overview

Description of the AC protocol

Let $\boldsymbol{\beta}_i(t)$ be the vector of measurements associated with the i-th node of the network at instant t, and $N = |V|$, the task is to converge to:

$$\hat{\boldsymbol{\beta}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{\beta}_i(0) \,. \tag{1}$$

For discrete-time distributed systems the DAC protocol is defined by a set of linear updating equations in the form of:

$$\boldsymbol{\beta}_i(t+1) = \sum_{j \in \mathcal{N}_i} W_{ij} \boldsymbol{\beta}_j(t) \,. \tag{2}$$

where $W_{ij} = 0$ is nodes i and j are not connected, or more compactly:

$$\boldsymbol{\beta}(t+1) = \mathbf{W}\boldsymbol{\beta}(t) \,. \tag{3}$$

AC convergence

AC converges to the global average, irrespective of the initial states, provided that the network is undirected, connected, and that the connectivity matrix $\mathbf{W}$ respects the following properties:

$$\mathbf{W} \cdot \mathbf{1} = \mathbf{1}\,, \tag{4}$$

$$\rho\left(\mathbf{W} - \frac{\mathbf{1}\cdot\mathbf{1}^T}{N}\right) < 1\,, \tag{5}$$

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

$$\rho(\mathbf{M}) = \max_r \left\{ |\lambda_r(\mathbf{M})| \right\}, \tag{6}$$

where $\lambda_r(\mathbf{M})$ is the rth eigenvector of a generic matrix $\mathbf{M}$.

Strategy 1: Max-Degree

The first strategy that we consider is the max-degree weights matrix, which is a common choice in real-world applications, and is defined entry-wise by:

$$w_{ij} = \begin{cases} 1/(d+1) & i \neq j, \{i,j\} \in E \\ 1 - d_i/(d+1) & i = j \\ 0 & i \neq j, \{i,j\} \notin E \end{cases}, \tag{7}$$

where d_i is the degree of the i-th node, and d is the maximum degree of the network.

Strategy 2: Metropolis-Hastings

An even simpler choice is the Metropolis-Hastings weights matrix:

$$w_{ij} = \begin{cases} 1/(\max\{d_i, d_j\} + 1) & i \neq j, \{i,j\} \in E \\ 1 - \sum_{j \in \mathcal{N}_i} 1/(\max\{d_i, d_j\} + 1) & i = j \\ 0 & i \neq j, \{i,j\} \notin E \end{cases}, \tag{8}$$

where $\mathcal{N}_i$ is the set of nodes' indexes directly connected to node i.

Strategy 3: Minimum Asymptotic

The third matrix strategy considered here corresponds to the optimal strategy introduced in [XB04], wherein the weights matrix is constructed to minimize the asymptotic convergence factor $\rho(\mathbf{W}-\mathbf{1}\mathbf{1}^{\mathrm{T}}/N)$.

This is achieved by solving the constrained optimization problem:

$$\begin{aligned} &\text{minimize} && \rho(\mathbf{W}-\mathbf{1}\mathbf{1}^{\mathrm{T}}/N) \\ &\text{subject to} && \mathbf{W}\in\mathcal{W}, \quad \mathbf{1}^{\mathrm{T}}\mathbf{W}=\mathbf{1}^{\mathrm{T}}, \quad \mathbf{W}\mathbf{1}=\mathbf{1} \end{aligned} \tag{9}$$

This is non-convex, but it can be shown to be equivalent to a semidefinite programming (SDP) problem [XB04], solvable using efficient ad-hoc algorithms.

Strategy 4: Laplacian Heuristic

The fourth and last matrix considered in this work is an heuristic approach [XB04] based on constant edge weights matrix:

$$\mathbf{W} = \mathbf{I} - \alpha \mathbf{L}\,, \tag{10}$$

where $\alpha \in \mathbb{R}$ is a user-defined parameter, and $\mathbf{L}$ is the Laplacian matrix associated to the network [BSDL13]. The asymptotic convergence factor satisfies:

$$\begin{aligned} \rho(\mathbf{W} - \mathbf{1}\mathbf{1}^{\mathrm{T}}/N) &= \max\{\lambda_2(\mathbf{W}), -\lambda_n(\mathbf{W})\} \\ &= \max\{1 - \alpha\lambda_{n-1}(\mathbf{L}), \alpha\lambda_1(\mathbf{L}) - 1\}\,, \end{aligned} \tag{11}$$

where $\lambda_i(\mathbf{W})$ denotes the i-th eigenvalue associated to $\mathbf{W}$. The value of α that minimizes (11) is given by:

$$\alpha^* = \frac{2}{\lambda_1(\mathbf{L}) + \lambda_{N-1}(\mathbf{L})}\,. \tag{12}$$

Overview

RVFL Networks

An RVFL network is a model of the following form:

$$f_\omega(\mathbf{x}) = \sum_{m=1}^{B} \beta_m h_m(x; w_m) = \boldsymbol{\beta}^{\mathrm{T}} \mathbf{h}(\mathbf{x}; \mathbf{w}_1, \ldots, \mathbf{w}_m)\,, \tag{13}$$

where the internal parameters of the B basis functions $\{\mathbf{w}_i\}_{i=1\ldots B}$ are randomly generated from a fixed probability distribution before the learning process.

Sequential learning for RVFL networks

In a sequential scenario, at each time step we receive a training batch T_i, for a given number P of batches. Let us define $\mathbf{H}_i$ and $\mathbf{y}_i$ as the hidden matrix and output vector computed over the i-th batch, respectively. The optimal least-square solution can be computed recursively by means of the recursive least-square (RLS) algorithm [SCSU15]:

$$\mathbf{P}(n+1) = \mathbf{P}(n) - \mathbf{P}(n)\mathbf{H}_{n+1}^T \mathbf{M}_{n+1}^{-1} \mathbf{P}(n)\,, \tag{14}$$

$$\boldsymbol{\beta}(n+1) = \boldsymbol{\beta} + \mathbf{P}(n+1)\mathbf{H}_{n+1}^T \left[\mathbf{y}_{n+1} - \mathbf{H}_{n+1}\boldsymbol{\beta}(n)\right]\,, \tag{15}$$

where $\mathbf{P}(n)$ is an auxiliary state matrix, and we defined:

$$\mathbf{M}_{n+1} = \mathbf{I} + \mathbf{H}_{n+1}\mathbf{P}(n)\mathbf{H}_{n+1}^T\,. \tag{16}$$

Distributed learning for RVFL networks

When considering a distributed scenario, we suppose that training data is partitioned throughout a network of agents, such that at every time step each agent receives a new batch, and the local batches are mutually independent.

A distributed algorithm for RVFL for the case of a single batch is presented in [SWPU15], and later extended to the more general setting in [SFW+15]:

1. Locally update each RVFL network using the RLS approach.
2. Perform a global average step over the output weights using the AC protocol.

Overview

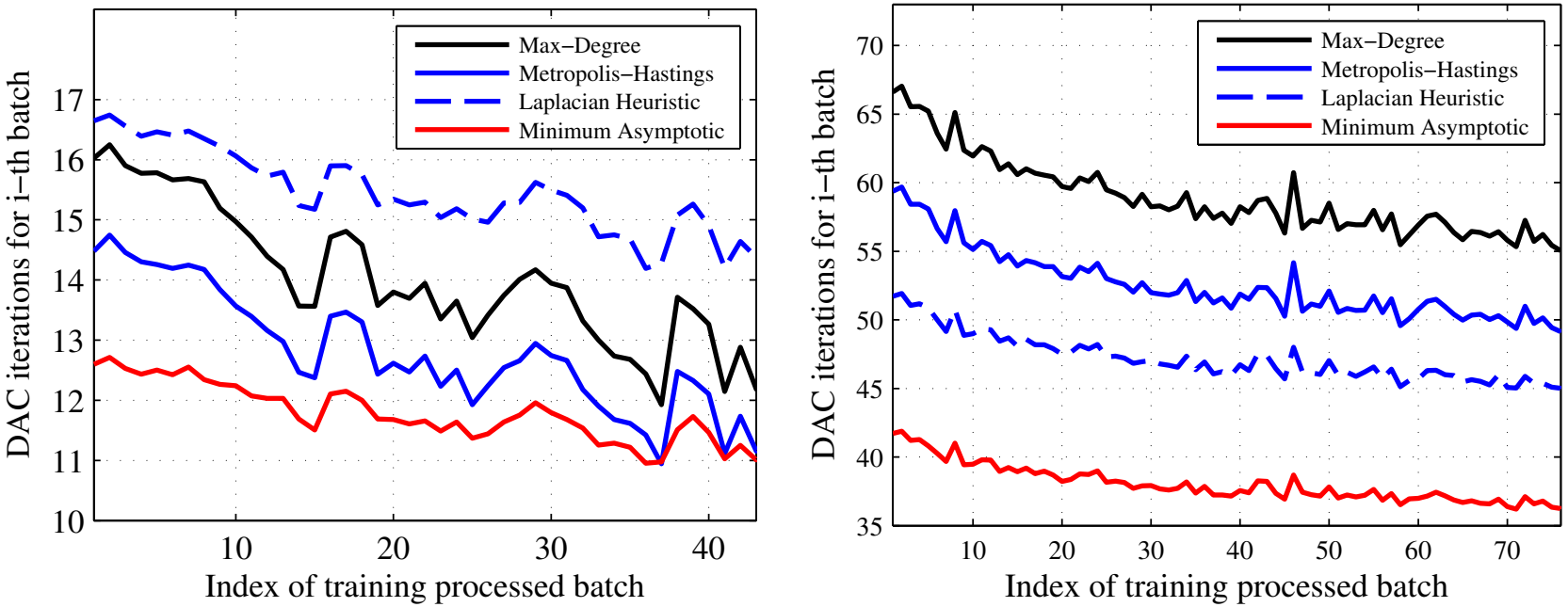
Datasets

Name	Features	Instances	Task
G50C	50	550	Gaussian of origin (classification)
CCPP	4	9568	Plant output (regression)

Table : Description of the datasets

We perform 25 rounds of simulation with 8 nodes in the network with random topology. At each round, datasets are subdivided in batches following the procedure detailed in [SFW+15]. Since in real applications the value of the average is not available to the nodes, in order to evaluate the number of iterations, we consider that all the nodes reached consensus when $\|\boldsymbol{\beta}_i(t) - \boldsymbol{\beta}_i(t-1)\|^2 \leq 10^{-6}$ for any value of i.

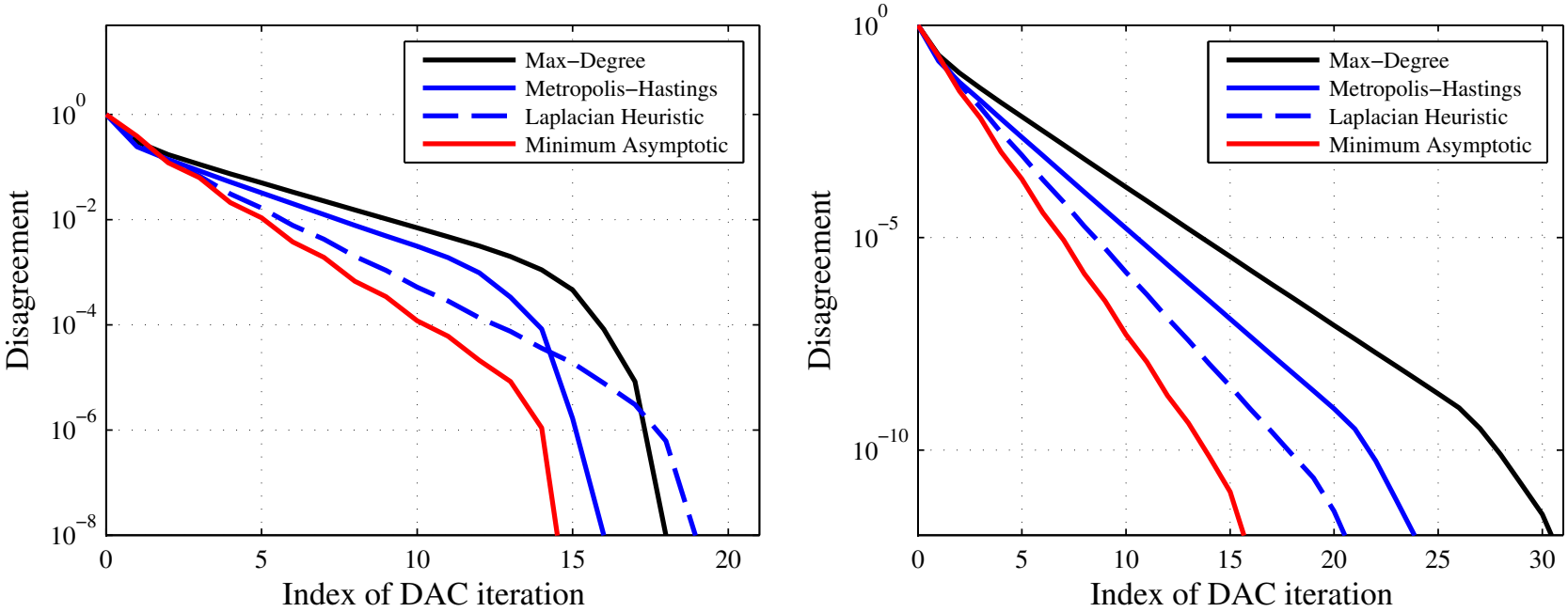
DAC iterations



(a) Dataset: G50C (b) Dataset: CCPP

Figure : Evolution of the DAC iterations required by the considered strategies to converge to the average, when processing successive amounts of training batches.

Disagreement



(a) Dataset: G50C (b) Dataset: CCPP

Figure : Evolution of the relative network disagreement for the considered strategies as the number of DAC iterations increases. The y-axis is shown with a logarithmic scale.

Overview

Concluding remarks

Main points:

- We have conducted an empirical comparison on the performance of different strategies for the AC protocol.
- We focused on the application of the DAC protocol to a recently proposed distributed training algorithm for RVFL networks.
- An appropriate choice of the weights matrix can lead to considerable improvements.

Future works:

- Nodes can be trained locally using different models than RVFL networks.
- Future works will extend the analysis to time-varying topologies [OSM04] and strategies involving communication constraints (e.g. on the time and energy required for data exchange).

References

S. Barbarossa, S. Sardellitti, and P. Di Lorenzo.
Distributed detection and estimation in wireless sensor networks.
In R. Chellapa and S. Theodoridis, editors, *E-Reference Signal Processing*, pages 329–408. Elsevier, 2013.

R. Olfati-Saber and R. M. Murray.
Consensus Problems in Networks of Agents With Switching Topology and Time-Delays.
IEEE Transactions on Automatic Control, 49(9):1520–1533, 2004.

S. Scardapane, D. Comminiello, M. Scarpiniti, and A. Uncini.
Online Sequential Extreme Learning Machine With Kernels.
IEEE Transactions on Neural Networks and Learning Systems (under press), 2015.

S. Scardapane, R. Fierimonte, D. Wang, M. Panella, and A. Uncini.
Distributed Music Classification using Random Vector Functional-Link Nets.
In *accepted for presentation at 2015 IEEE/INNS International Joint Conference on Neural Networks (IJCNN'15)*, 2015.

S. Scardapane, D. Wang, M. Panella, and A. Uncini.
Distributed learning for Random Vector Functional-Link networks.
Information Sciences, 301:271 – 284, 2015.

L. Xiao and S. Boyd.
Fast linear iterations for distributed averaging.
Systems & Control Letters, 53(1):65–78, 2004.