To be published as a chapter in '**Adaptive Learning Methods for Nonlinear System Modeling**', Elsevier Publishing, Eds. D. Comminiello and J.C. Principe (2018)

Adaptation and learning over networks for nonlinear system modeling

Simone Scardapane[a,*], Jie Chen[b], Cédric Richard[c]

[a] *Department of Information Engineering, Electronics and Telecommunications, Sapienza University of Rome, Via Eudossiana 18, 00184 Rome, Italy*
[b] *Northwestern Polytechnical University, Xi'an, School of Marine Science and Technology, 127 West Youyi Road, 710072, Xi'an (China)*
[c] *Université Côte d'Azur, Laboratoire Lagrange (UMR CNRS 7293), Parc Valrose, 06108, Nice Cedex 2 (France)*

Abstract

In this chapter, we analyze nonlinear filtering problems in distributed environments, e.g., sensor networks or peer-to-peer protocols. In these scenarios, the agents in the environment receive measurements in a streaming fashion, and they are required to estimate a common (nonlinear) model by alternating local computations and communications with their neighbors. We focus on the important distinction between single-task problems, where the underlying model is common to all agents, and multitask problems, where each agent might converge to a different model due to, e.g., spatial dependencies or other factors. Currently, most of the literature on distributed learning in the nonlinear case has focused on the single-task case, which may be a strong limitation in real-world scenarios. After introducing the problem and reviewing the existing approaches, we describe a simple kernel-based algorithm tailored for the multitask case. We evaluate the proposal on a simulated benchmark task, and we conclude by detailing currently open problems and lines of research.

Keywords: Nonlinear system identification, distributed systems, adaptive methods, reproducing kernel Hilbert spaces, diffusion algorithms

April 28, 2017

1. Introduction

Adaptive filters have been at the heart of digital signal processing over the last century, thanks to their capability of rapidly adapting to streams of incoming data. At the same time, classical filtering approaches have not been satisfactory to handle the challenges posed by large-scale, unstructured big data scenarios that are common today. This has fostered the recent development of techniques to deal with such problems, allowing signal processing to scale to truly massive datasets [1], and to work with less structured data types, such as graphs [2, 3]. In this chapter, we look at one peculiar aspect unifying several big data problems, namely, their *distributed* nature, where the data are naturally generated and aggregated at different locations with possibly poor or expensive network connectivity. Examples of problems in this category abound, including (but not limited to), wireless sensor networks (WSNs) [4], distributed databases, robotic swarms, fog computing platforms, among others. In all these scenarios, the agents in the network can be severely limited in their capabilities, in terms of either energy constraints (e.g., low-power devices in WSNs), connectivity, privacy, or other aspects. As a consequence, any solution devised for learning and inference over networks needs to be aware of these constraints, making this a challenging problem with wide applications.

Distributed learning can be cast as a decentralized optimization problem, which has a long history in the optimization field [5] and in artificial intelligence. In recent years, this problem has gained a renewed interest from the machine learning community, with the development of a number of learning protocols for a variety of models, including boosting [6], support vector machines [7, 8], kernel regression [4], and sparse linear models [9, 10], to cite a few. Several of these were also applied in signal processing problems, most notably in order to provide distributed inference capabilities in WSNs [4]. A large majority of them, however, is only applicable in *batch* situations, where each agent is allowed to manipulate its entire (local) dataset at each iteration. Distributed filtering algorithms, on the contrary, require the development of online solutions, where the data are received and processed, sequentially, by the agents.

In the filtering literature, a recent series of works was initiated by the

*Corresponding author. Phone: +39 06 44585495, Fax: +39 06 4873300.
Email address: simone.scardapane@uniroma1.it (Simone Scardapane)

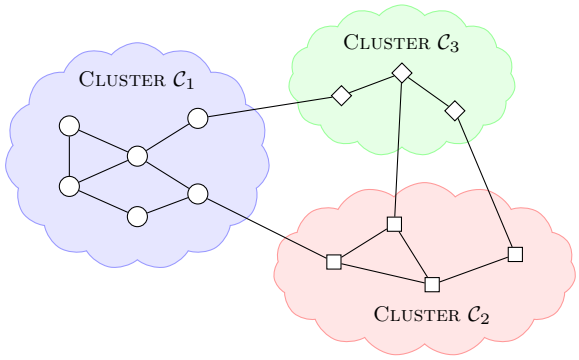


Figure 1: An example of network with 12 agents subdivided in three different clusters.

development of the so-called 'diffusion filtering' (DF) algorithms, starting from the diffusion LMS [11] and the diffusion RLS [12], up to their more general formulation in terms of generic convex cost functions [13, 14, 15], which are considered here. DF algorithms are characterized by interleaving local updates in parallel (mimicking classical filters), with communication steps, during which the agents exchange information on their current estimate with their neighbors. Following the development of the main theory, in the subsequent years, several authors have extended classical linear filters to the distributed case (e.g., sparse LMS [10] and group LMS [16]), while others have focused on nonlinear filters, as we review further on in the chapter. For the interested reader, we refer to the guest editorial in [17] and references therein for a recent overview of the literature.

All the works mentioned up to now assume that the agents are identifying a *common* model. This is a reasonable assumption in several contexts, particularly when the statistics of the data do not depend on the spatial locations of the agents, making it very common in distributed machine learning problems [7, 15]. In general, however, the agents could be interested in different identification problems, which are similar in some quantifiable sense. As an example, consider the setup illustrated in Fig. 1. If the agents are sensors deployed over an environment, trying to predict some quantity of interest (e.g., some pollutant concentration), the model might be different among groups (clusters) of agents, possibly due to the spatial conformation of the ground (e.g., sensors deployed over a mountain v.s. sensors deployed in a valley). Nonetheless, since they are all trying to predict the same quantity of interest, communication between agents belonging to different clusters can

be beneficial.

The first DF solution to this problem, which is termed *multitask* network, was analyzed in [18]. Following this, a range of algorithms was proposed in the linear case. Most notably, Chen *et al.* [19] and Zhao and Sayed [20] investigated the possibility of unsupervised learning of the clusters structure when it is not available *a priori*. Additional developments include the extension to asynchronous networks where, e.g., links may fail or agents might disconnect [21]; proximal updates for nondifferentiable regularization terms [22]; total least squares approaches [23]; and, finally, multitask learning over (linear) latent subspaces [24, 25]. Almost no work, however, has addressed the problem of learning in a multitask network with *nonlinear* models.

Based on the previous discussion, this chapter has three separate aims. First, we introduce the fundamental concept of DF in Section 2, which serves as a very general introduction to the topic. Next, we summarize recent works on nonlinear DF algorithms in Section 3, with an emphasis on three classes of solutions. In order to motivate research on multitask learning with nonlinear models, in Section 4 we propose a multitask kernel algorithm based on a functional formulation of DF. Finally, we validate the algorithm on an experimental benchmark in Section 5, before making some final remarks in Section 6.

2. Mathematical formulation of the problem

This section is intended to familiarize the reader with some basic theoretical elements underlying most distributed learning scenarios. We start by providing a setup for the problem in Section 2.1. Next, we describe a general class of algorithms based on diffusion protocols in Section 2.2. In Section 2.3, we show how these algorithms can be customized to address multitask scenarios. For conciseness, we only focus on a selection of key items, without providing a comprehensive treatment of these thematics. We refer the interested reader to [15, 14] for introductory references.

2.1. Problem setup

Let us consider a generic network of N agents (e.g., sensors in a WSN) as the one depicted in Fig. 1. We assume that time is slotted and, at every time instant n, each agent receives a new observation $(\mathbf{u}_{k,n}, d_k(n))$, where $\mathbf{u}_{k,n} \in \mathbb{R}^M$ is the model input vector at agent k (e.g., a buffer of the last M samples), and $d_k(n)$ the corresponding desired response. For simplicity, we

assume that $d_k(n)$ is a scalar. For the rest of the chapter, we shall use the subscript k to denote a quantity specific to one of the agents.

Following the standard supervised learning approach, the desired input/output relation can be modeled by choosing a function f in some hypothesis space $\mathcal{H}$. For simplicity, in this section we suppose that each function is parameterized by a vector of tunable parameters $\mathbf{w} \in \mathbb{R}^q$, e.g., a linear predictor.[1] With this setting, each agent is interested in finding a set of parameters $\mathbf{w}_k^*$ which minimizes some local cost function $J_k(\cdot)$ defined over the hypothesis space from streaming data. Specifically, for most estimation problems in practice, these local cost functions are defined as the expectation of some *error function* $L(\cdot, \cdot)$ with respect to the statistics of the local stream of data:

$$J_k(\mathbf{w}_k) = \mathbb{E}\Big\{ L\big(d_k, f_k(\mathbf{u}_k)\big) \Big\}. \tag{1}$$

where we use the shorthand $f_k(\mathbf{u}_k) = f(\mathbf{u}_k; \mathbf{w}_k)$. The global optimization problem to be solved at the network level is then given by the sum of the local cost functions:

$$\min_{\mathbf{w}_1, \ldots, \mathbf{w}_N \in \mathbb{R}^q} \Big\{ J_{\text{glob}}(\mathbf{w}_1, \ldots, \mathbf{w}_N) \Big\} = \sum_{k=1}^{N} J_k(\mathbf{w}_k)\,, \tag{2}$$

where $\mathbf{w}_k$ is the estimate at the kth agent. If we assume that no relation holds between the local cost functions, then (2) reduces to a set of N optimization problems that can be solved in parallel by every agent, independently of all the others. A more interesting formulation arises by assuming some form of relation among the cost functions (detailed below). In this case, the information gathered by one agent during its optimization process can potentially be used by the other agents to speedup their convergence, or even converge to a better solution using some shared information.

The difficulty arises from the fact that each agent has direct access to its local cost function, but it has no access to the local cost functions of the other agents for all the reasons mentioned in the introduction. Depending on the relation between cost functions, we can distinguish between three classes of distributed problems:

[1] Distributed kernel filters (Section 3.2) are an example of a non-parametric formulation, which is recast as a parametric problem thanks to the representer's theorem.

- **Single-task problems**: in this case, $\mathbf{w}_k^* = \mathbf{w}^*$, $k = 1, \ldots, N$, i.e., all the cost functions have the same minimizer which must be attained by all agents. This is the scenario which has drawn most attention in the literature, being particularly useful in distributed machine learning problems [7], where it is common to assume that the data of interest are generated by a single underlying distribution.

- **Multitask problems**: in this scenario, each local cost function has possibly a different minimizer $\mathbf{w}_k$.[2] In order to make the problem interesting, we assume that these minimizers are 'similar' (in some sense to be properly defined) among pairs of neighboring agents, so that communicating can increase their speed of convergence and possibly counter noisy environments.

- **Clustered multitask problems**: in this intermediate case, each agent belongs to one of T different groups (clusters), such that all agents belonging to the same cluster have the same minimizer, and *vice versa*, as shown in Fig. 1. Clearly, both single-task and multitask problems can be derived as extreme cases of this class of problems, by setting $T = 1$ and $T = N$.

Multitask problems can be further subdivided, depending on whether the similarity between tasks is known *a priori*, or whether it must be inferred from the data. Examples of the former case are the algorithm in [18], while examples of the latter case can be found in [19, 20]. Inferring knowledge about the groups might require the inclusion of some decentralized clustering procedure in the learning process, which is an interesting problem in its own right [29]. For simplicity, in this chapter we will focus on the multitask formulation, but we underline that almost all multitask algorithms can be generalized to handle the clustered multitask case, e.g., see [18].

[2] Some readers might recognize that in the machine learning literature, the term 'multitask learning' is employed in a slightly different meaning. It refers to the problem of solving several learning tasks defined on the same (or in similar) input domain(s) [26, 27]. While the setup and the objectives in the two cases do not perfectly overlap, we speculate that exploring the connections between them is of particular significance, particularly due to the increasing interest given by deep neural networks [28].

2.2. Diffusion-based algorithms

In order to describe a family of algorithms to solve the previous problem, we first need to define a model of communication between the different agents. Most of the literature focuses on the case where the communication links form a static, undirected, connected graph $\mathcal{G}$. At each time step, the kth agent is allowed to communicate with its set of direct neighbors $\mathcal{N}_k$, while it cannot send messages to agents to which it is not directly linked.[3] Nonetheless, the overall connectedness of the graph ensures that information can flow throughout the entire network. In this scenario, connectivity can be described by a symmetric, real-valued matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ such that $A_{kl} \neq 0$ only if agents k and l are connected, and:

$$\sum_{k=1}^{N} A_{kl} = 1\,, \;\; A_{kl} \geq 0 \;\; \text{for any } k, l = 1, \ldots, N\,. \tag{3}$$

These weights are used by the agents to scale and combine information received by their neighbors. The previous condition ensures that each row of the matrix defines a convex combination, so that the range of the information to be combined is always preserved (formally, the condition requires that $\mathbf{A}$ be left stochastic). There are several strategies allowing agents to build such matrices, as we show later. This formulation can also be extended in several ways, most notably with the use of asynchronous formulations [30] (thus avoiding the need for a common clock throughout the network), and mixing matrices that do not respect double stochasticity [31]. Nonetheless, since this chapter is only intended as an introduction, we will focus on the simpler case detailed before.

In the case of a single agent, (2) could be solved by a simple gradient descent algorithm. In order to counteract the lack of global information, the basic idea of diffusion algorithms is to interleave local optimization steps with communication steps, where each agent combines its own estimate with those of its neighbors. In the filtering literature, this strategy is generally

[3] The graph describes all *feasible* communication links. This is a relatively general formulation, since every multi-hop network can be described with an equivalent single-hop network by considering all possible paths as a direct link in the corresponding graph.

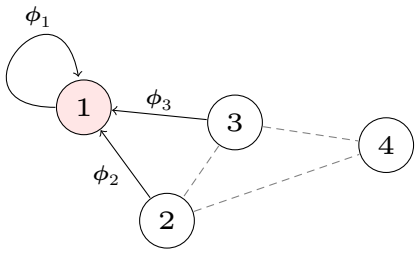


Figure 2: An example of a diffusion step relative to agent 1 in a simple network with 4 agents. Gray links are inactive.

denoted as adapt-then-combine (ATC):

$$\boldsymbol{\phi}_{k,n} = \mathbf{w}_{k,n-1} - \mu_k \nabla J_k(\mathbf{w}_{k,n-1}) \,, \tag{4}$$

$$\mathbf{w}_{k,n} = \sum_{l=1}^{N} A_{lk} \boldsymbol{\phi}_{l,n} \,, \tag{5}$$

where μ_k is a (possibly time-dependent) step size and, similarly to before, we use a double subscript (k, n) to denote the estimate of agent k at time n. Practically, the gradient term in (4) can be substituted with a noisy version, for example using an instantaneous approximation computed from the current data sample. An example of diffusion step is given in Fig. 2.

These algorithms are particularly suitable in single-task scenarios, where their convergence properties in the convex case have been analyzed extensively [14, 15]. Interestingly, they can still have good convergence properties in the multitask case, as shown in [19]. However, they provide a building block for all other formulations, as we show in the next subsection. Before that, we describe an example of diffusion algorithm for nonlinear learning with a distributed logistic regression model.

2.2.1. An example: logistic regression over networks

Consider a binary classification problem where $d_k(n) \in \{0, 1\}$, i.e., each output is a single bit representing whether the corresponding input belongs to a certain class or not. We approximate the underlying relation using a logistic predictor $f(\mathbf{u}) = \sigma(\mathbf{w}^T \mathbf{u})$, where $\sigma(\cdot)$ is the sigmoid function:

$$\sigma(s) = \frac{1}{1 + \exp\{-s\}} \,, \tag{6}$$

ensuring that the outputs of the model are properly scaled as valid probabilities. Each agent wishes to minimize the (regularized) expected cross-entropy over its stream of data:

$$J_k(\mathbf{w}) = \mathbb{E}\Big\{-d_k \cdot \log\big(f_k(\mathbf{u}_k)\big) - (1-d_k)\cdot \log\big(1-f_k(\mathbf{u}_k)\big)\Big\} + \frac{\lambda}{2N}\,\|\mathbf{w}_k\|_2^2\ , \quad (7)$$

where the factor $1/N$ in the regularization term ensures that the total penalization in (2), when summed over all agents, is equal to $\frac{\lambda}{2}$. By taking instantaneous approximations to the gradient, simple algebra manipulations show that the update steps in (4) are given by:

$$\boldsymbol{\phi}_{k,n} = \mathbf{w}_{k,n-1} + \mu_k\big(d_k(n) - f_{k,n-1}(\mathbf{u}_{k,n})\big)\mathbf{u}_{k,n} - \frac{1}{N}\mu_k\lambda\mathbf{w}_{k,n-1}\,. \quad (8)$$

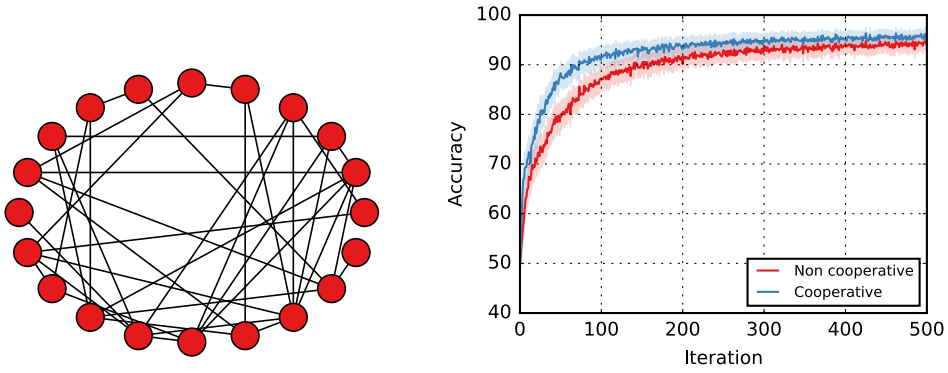
In order to show the speedup obtained by such procedure, in Fig. 3 we plot the average accuracy (see below) obtained with a network of 20 agents, whose connectivity is generated randomly, and where at every iteration each agent receives a randomly chosen example taken from the well-known Wisconsin Breast Cancer Database (WBCD).[4] The accuracy is defined as 1 if the agent makes a correct prediction (i.e., the sign of $f(\mathbf{u})$ agrees with d), 0 otherwise. It is computed before the adaptation step, and it is averaged with respect to the different agents and the different simulations.

2.3. Extension to multitask learning

The general ideas exposed in the previous section can be easily extended in order to be more efficient in the multitask scenario. Here, we detail a simple extension originally proposed in [18] to show one example of such extensions. We refer to the large number of works cited in the introduction for more recent proposals.

Suppose that, given two agents k and l, we have a way to quantify the similarity among their respective minimizers. In order to leverage this information, we can augment the original cost function in (2) with a regularization term forcing the similarity among minimizers, in terms of their Euclidean dis-

[4] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)



(a) Network connectivity (b) Accuracy

Figure 3: An example of distributed (online) logistic regression on the WBCD dataset. (a) The network of 20 agents used in this experiment. (b) Average accuracy for the diffusion algorithm, in blue, and for a network of N non-cooperating agents (which is equivalent to set $\mathbf{A} = \mathbf{I}$), in red. We use shaded contours to plot standard deviation.

tance:

$$J^{\text{glob}}(\mathbf{w}_1, \ldots, \mathbf{w}_N) = \sum_{k=1}^{N} J_k(\mathbf{w}_k) + \eta \sum_{k=1}^{N} \sum_{l \neq k, l \in \mathcal{N}_k} \rho_{kl} \left\| \mathbf{w}_k - \mathbf{w}_l \right\|_2^2 , \qquad (9)$$

where η is a regularization factor, and the nonnegative coefficients $\rho_{kl} \geq 0$ quantify our *a priori* knowledge about the similarities. We assume that, for each agent, the weights are positive and sum to one:

$$\sum_{l=1}^{N} \rho_{kl} = 1, \quad \text{and} \quad \rho_{kl} = 0 \text{ if } l \notin \mathcal{N}_k, \ \forall k \in \{1, \ldots, N\} . \qquad (10)$$

As a consequence of their definition, these regularization factors mirror the communication network of the agents. Due to this, taking a local optimization step with respect to the estimate of the kth agent immediately requires a diffusion step:

$$\mathbf{w}_{k,n} = \mathbf{w}_{k,n-1} - \mu_k \nabla J_k(\mathbf{w}_{k,n-1}) - \mu_k \eta \sum_{l \neq k, l \in \mathcal{N}_k} \frac{(\rho_{kl} + \rho_{lk})}{2} \left(\mathbf{w}_{k,n-1} - \mathbf{w}_{l,n-1} \right) , \qquad (11)$$

Since the estimates are already exchanged, the previous update step can be implemented without the need for additional combination steps like in the previous section. As an example, by considering a simple linear predictor $f_k(\mathbf{u}) = \mathbf{w}_k^T\mathbf{u}$ and instantaneous approximations to the gradient, we obtain the update rule for the multitask diffusion LMS presented in [18]:

$$\begin{aligned}\mathbf{w}_{k,n} = \mathbf{w}_{k,n-1} + \mu_k\big(d_k(n) - \mathbf{w}_{k,n-1}^T\mathbf{u}_{k,n}\big)\mathbf{u}_{k,n} - \\ \mu_k\eta \sum_{l\neq k, l\in\mathcal{N}_k} \frac{(\rho_{kl}+\rho_{lk})}{2}\left(\mathbf{w}_{k,n-1} - \mathbf{w}_{l,n-1}\right) .\end{aligned} \tag{12}$$

If we assume that the mixing weights are symmetric, $\frac{(\rho_{kl}+\rho_{lk})}{2}$ simplifies to ρ_{kl}. It is possible to obtain asymmetric regularization terms by considering a game-theoretical formulation of the optimization problem, see the discussion in [18].

3. Existing approaches to nonlinear distributed filtering

In this section we describe three approaches to extend the previous formulation to nonlinear models in an efficient way. We underline that all these algorithms have been devised mostly for the case of single-task networks. Further extensions to the multitask scenario are the topic of the next section.

3.1. Expansion over random bases

One immediate idea is to project the original input vector $\mathbf{u}$ to a high-dimensional space via some fixed function $\mathbf{h}(\mathbf{x}) : \mathbb{R}^M \to \mathbb{R}^B$ before using it with a linear predictor, where d is the dimensionality of the input vector $\mathbf{u}$, and B is a parameter which (in general) can be chosen by the user. In the distributed case, this requires only a small communication overhead in the beginning for the agents to agree on a specific projection function. Any distributed linear algorithm, such as the diffusion LMS or the diffusion RLS, can then be used.

Generally speaking, deterministic mappings (such as those mentioned in Chapters 2 and 3) are not efficient, because their size might grow exponentially with respect to M. A different idea is to use basis functions whose parameters are assigned stochastically, e.g. a parameterized sigmoid:

$$h_i(\mathbf{u}) = \frac{1}{1+\exp\left\{-\mathbf{a}_i^T\mathbf{u} - b_i\right\}} ,$$

where $\mathbf{a}_i$ and b_i might be extracted randomly from some uniform distribution (whose range is generally chosen in order to provide a good accuracy, see the discussion in [32]). Interestingly, it is possible to show that the resulting estimator (called a random vector functional-link network) is a universal approximator over compact functions provided that B is chosen large enough [33]. It is also possible to interpret it as a degenerate case of the echo state network described in Chapter 12, where connections between different nodes have been removed. The idea of using RVFL networks in a distributed context was proposed in [34] for the batch case, and in [35] for the online case with DF algorithms.

A different approach is to design a feature mapping $\mathbf{h}(\cdot)$ approximating a specific kernel $\mathcal{K}(\cdot,\cdot)$ function[5] chosen by the user:

$$\mathcal{K}(\mathbf{u}_1, \mathbf{u}_2) \approx \langle \mathbf{h}(\mathbf{u}_1), \mathbf{h}(\mathbf{u}_2) \rangle \,. \tag{13}$$

This idea was popularized by [36] for approximating shift-invariant kernels (e.g., the Gaussian kernel) in large-scale applications of kernel methods. In particular, it is possible to show that this class of kernels can be easily approximated with very simple stochastic mappings. [37] was the first to apply this idea explicitly to kernel filters, and similar algorithms were independently reintroduced in [38]. Since Chapter 8 is entirely devoted to this idea, we will not go further into it. We refer the interested reader to [32] for a recent overview on random feature methods.

3.2. Distributed kernel filters

An alternative line of research is devoted to distributed strategies for kernel filters, working directly on some reproducing kernel Hilbert space (RKHS), instead of approximating the kernel function as in the previous section. As we stated in the introduction, several distributed algorithms for kernel ridge regression were devised in the context of WSNs [4], followed by algorithms for the distributed optimization of SVMs [39, 7]. Any approach to dealing with kernels faces the challenge of working with a kernel-based model that depends explicitly on all the data in the training set. A naïve distributed implementation would thus require to exchange all the local datasets between the agents, which can become infeasible.

[5] We refer to Chapters 6-8 for introductory material on kernel filters.

In an online context, this is made worse by the growing nature of the kernel model [40]. This is a fundamental drawback underlying any kernel filter algorithm [41, 42, 43]. An initial investigation in developing a fully distributed version of the kernel LMS (KLMS) was made in [44], where the basic idea is to consider diffusion algorithms directly in a functional form. In particular, let us assume that the data received from the kth agent satisfies a model of the form:

$$d_k(n) = \psi_k^o(\mathbf{u}_{k,n}) + \nu_k(n)\,, \tag{14}$$

where ψ_k^o belongs to a RKHS $\mathcal{H}$, while $\nu_k(n)$ is a zero-mean white noise with variance σ_k^2. Restricting our attention to a generic single-task network, we have:

$$\psi_k^o = \psi^o \;\; \forall k \in \{1, \ldots, N\}\;,$$

Considering the classical squared error function, the gradient of the local cost functions can now be computed in terms of their Fréchet derivatives as:[6]

$$\nabla J_k(\psi_k) = -2\mathbb{E}\Big\{\big(d_k - \psi_k(\mathbf{u}_k)\big)\kappa(\cdot, \mathbf{u}_k)\Big\}\,, \tag{15}$$

where κ is the kernel function associated to the RKHS. Considering instantaneous approximations for the expectation as was done earlier, we arrive at a functional equivalent of the ATC diffusion framework:

$$\delta_{k,n} = \psi_{k,n-1} + \mu_k \left(d_k(n) - \psi_{k,n-1}(\mathbf{u}_{k,n})\right)\kappa(\cdot, \mathbf{u}_{k,n})\,, \tag{16}$$

$$\psi_{k,n} = \sum_{l=1}^{N} A_{lk}\delta_{l,n}\,, \tag{17}$$

Although this formulation is extremely general, one has still to solve the problem of the growing structure of the kernel functions. The idea pursued in [44] is to assume some shared dictionary $\mathcal{D}$ among nodes, whose selection is (at the moment) an open research question. Using this approximation, we can rewrite the desired function as:

$$\psi_{k,n} = \boldsymbol{\beta}_{k,n}^T \mathbf{k}_{k,n}\,, \tag{18}$$

[6] A functional derivative is needed because the dimentionality of $\mathcal{H}$ can be infinite. See [45] for an introduction to Fréchet derivatives in the context of kernel methods, and [46] for an introductory textbook on functional analysis.

where $\mathbf{k}_{k,n}$ is the vector of kernel values computed between the current input vector $\mathbf{u}_{k,n}$ and the shared dictionary $\mathcal{D}$. Each function $\psi_{k,n}$ is now parameterized by the set of linear coefficients $\boldsymbol{\beta}_{k,n}$. The previous algorithm can be rewritten as:[7]

$$\boldsymbol{\delta}_{k,n} = \boldsymbol{\beta}_{k,n-1} + \mu_k \left(d_k(n) - \boldsymbol{\beta}_{k,n-1}^T \mathbf{k}_{k,n})\right) \mathbf{k}_{k,n} \,, \tag{19}$$

$$\boldsymbol{\beta}_{k,n} = \sum_{l=1}^{N} A_{lk} \boldsymbol{\delta}_{l,n} \,. \tag{20}$$

The idea of preselecting a dictionary is not new in the kernel literature. In fact, one of the earliest algorithms for distributed SVMs [39] exploited a similar idea, which is termed semi-parametric SVM. In [47], a fixed dictionary is used to analyze the convergence behavior of the KLMS algorithm. A derivation of the functional diffusion KLMS algorithm when removing the fixed dictionary constraint is given in [48]. Another extension is presented in [49], where a set of consensus constraints is included in the problem to ensure convergence and speedup the algorithm.

3.3. Diffusion spline filters

Another possibility for nonlinear learning over networks is given by considering spline adaptive filters (SAFs).[8] The idea of using SAFs in a distributed environment was recently introduced in [50]. In the following we briefly describe the distributed algorithm. The interested readers can refer to [51, 52] for introductory material on the SAF model.

Let us assume that the data are generated according to a restricted Wiener model given by:

$$d_k(n) = f_k^o \left(\mathbf{w}_k^T \mathbf{u}_{k,n}\right) + \nu_k(n) \,. \tag{21}$$

where f_k^o is any smooth nonlinear function, and $\nu_k(n)$ is a noise term. A SAF mimics this architecture, where the nonlinear term is approximated via spline interpolation over a set of Q fixed control points that are adapted during learning. Hence, in a distributed scenario each agent has estimates of

[7] Note that we consider a simplified formulation with respect to [44], where two combination steps are used. Also, we use the same symbol $\boldsymbol{\delta}_k$ for the result of the adaptation step as in (16), but in boldface to underline that it is now a vector-valued quantity.

[8] SAFs are the topic of Chapter 5.

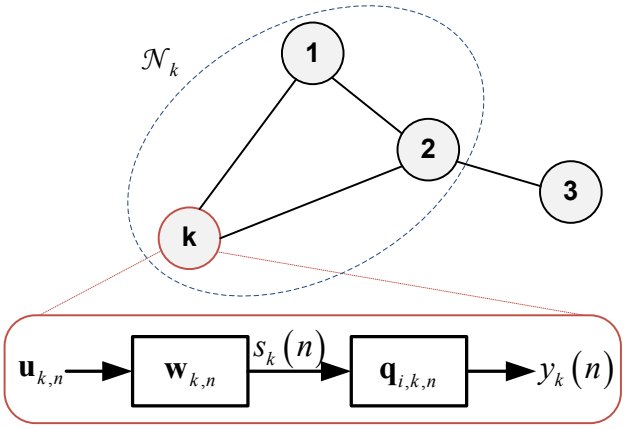


Figure 4: Illustration of SAF interpolation performed over a network of agents (adapted and reprinted with permission from [50]).

the local part of the filter, $\mathbf{w}_{k,n}$, and of the aforementioned control points, $\mathbf{q}_{k,n}$, as shown schematically in Fig. 4.

Consider a single-task scenario, where each agent tries to minimize the expected squared loss. Following [50], we consider a combine-then-adapt scheme (CTA), where the combination is performed before the adaptation. The two steps are applied to both sets of parameters $\mathbf{w}_{k,n}$ and $\mathbf{q}_{k,n}$ simultaneously. However, by exploiting the SAF structure we can avoid exchanging the full weight vector $\mathbf{q}_{k,n}$, as described in the following.

The combination step starts with the agents exchanging their current estimates of the linear weights, as:

$$\boldsymbol{\psi}_{k,n-1} = \sum_{l \in \mathcal{N}_k} A_{lk} \mathbf{w}_{l,n-1} \,. \tag{22}$$

The new weights are used to compute the output of the linear part of the filter, denoted as $s_k(n) = \boldsymbol{\psi}_{k,n-1}^T \mathbf{u}_{k,n}$. We use i to denote the index of the closest control point to $s_k(n)$ in our set of fixed control points. As described in Chapter 5, the final output of a Wiener SAF depends only on the ith control point and its P right neighbors, with P being the order of interpolation. Let us denote by $\mathbf{q}_{i,k,n-1}$ the set of such 'active' control points for agent k, which are called the 'span' of the filter (see Chapter 5 for more details). We use a third subscript to denote dependence with respect to the span. The second

combination step is performed only with respect to the current span:

$$\boldsymbol{\xi}_{k,n-1} = \sum_{l \in \mathcal{N}_k} A_{lk} \mathbf{q}_{i,l,n-1} \,. \tag{23}$$

In the case of cubic interpolation, each $\boldsymbol{\xi}_{k,n-1}$ has dimensionality 4, making its exchange extremely efficient, with only a fixed overhead with respect to a classical diffusion LMS. Practically, every agent sends its current span index i to its neighbors, and receives back the vectors $\mathbf{q}_{i,l,n-1}$. For simplicity, the mixing weights A_{lk} in the two diffusion steps are assumed identical.

Next, we proceed to the adaptation step. The complete SAF output given the new span is obtained as (again following the general rules of Chapter 5):

$$y_k(n) = \mathbf{u}^T \mathbf{B} \boldsymbol{\xi}_{k,n-1} \,. \tag{24}$$

where the vector $\mathbf{u}$ is constructed by taking powers up to a fixed order of the normalized value $\frac{s_k(n)}{\Delta x} - \left\lfloor \frac{s_k(n)}{\Delta x} \right\rfloor$, where Δx is the sampling precision of the spline. $\mathbf{B}$ is the spline matrix, e.g. the Catmull-Rom (CR) spline given by:

$$\mathbf{B} = \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \,. \tag{25}$$

Adaptation is made by performing two parallel gradient descent step:

$$\mathbf{w}_{k,n} = \boldsymbol{\psi}_{k,n-1} + \mu_k e_{k,n} \varphi'(s_k(n)) \mathbf{u}_{k,n} \,, \tag{26}$$

$$\mathbf{q}_{i,k,n} = \boldsymbol{\xi}_{k,n-1} + \mu_k e_{k,n} \mathbf{B}^T \mathbf{u} \,. \tag{27}$$

where $e_{k,n}$ is the instantaneous local error, and $\varphi'(s_k(n))$ is the spline derivative with respect to the linear weights. Note that the diffusion LMS can be obtained as a special case, where each node initializes its nonlinearity as the identity, and the step size of the nonlinear part is set to zero.

4. A distributed kernel filter for multitask problems

As we saw in the previous section, several ideas have been proposed to model nonlinear systems in a distributed fashion, but almost none is framed for the multi-task scenario. As a first step towards this line of research, in

this section we briefly combine some of the previous ideas to devise an efficient kernel-based diffusion algorithm for multi-task networks. In a nutshell, we combine the multi-task diffusion LMS presented in Section 2.3 with the functional diffusion KLMS of Section 3.2. To this end, consider again the data model in (14), where we assumed that all the minimizers are the same across the agents. More generally, we can consider the case where two functions ψ_k^o and ψ_l^o are assumed to be 'close' in the sense of the norm $\|\cdot\|_{\mathcal{H}}$ of the RKHS, whenever the corresponding agents are spatial neighbors:

$$\psi_k^o \sim \psi_l^o \;\text{ if } l \in \mathcal{N}_k \,, \tag{28}$$

where $\mathcal{N}_k$ denotes the set of neighbors of k, and $\sim$ denotes similarity. To recover the unknown functions, and leveraging over the basic idea described in Section 2.3, we aim at minimizing the following global cost function in a decentralized fashion:

$$J^{\text{glob}}(\psi_1,\ldots,\psi_N) = \sum_{k=1}^{N} \mathbb{E}\left\{\left|d_k(n) - \psi_k(\mathbf{u}_{k,n})\right|^2\right\} + \eta \sum_{k=1}^{N} \sum_{l\neq k, l\in\mathcal{N}_k} \rho_{kl} \left\|\psi_k - \psi_l\right\|_{\mathcal{H}}^2 \,, \tag{29}$$

where $\eta > 0$ is a regularization factor, and the nonnegative coefficients $\rho_{kl} \geq 0$ weight the similarity between different functions. Once again, we assume that, for each agent, the weights are positive and sum to one:

$$\sum_{l=1}^{N} \rho_{kl} = 1, \quad \text{and} \quad \rho_{kl} = 0 \text{ if } l \notin \mathcal{N}_k, \;\; \forall k \in \{1,\ldots,N\} \,. \tag{30}$$

Thus, each agent is interested in minimizing the local expected mean-squared error, under suitable proximity constraints on its function and the functions of its neighbors. The previous problem decomposes as a sum of local cost functions defined as:

$$J_k^{\text{loc}}(\psi_1,\ldots,\psi_N) = \mathbb{E}\left\{\left|d_k(n) - \psi_k(\mathbf{u}_{k,n})\right|^2\right\} + \eta \sum_{l\neq k, l\in\mathcal{N}_k} \rho_{kl} \left\|\psi_k - \psi_l\right\|_{\mathcal{H}}^2 \,. \tag{31}$$

Each local cost function is independent of the estimate of agents which are not in its immediate neighborhood. Taking the Fréchet derivative of (31)

gives us:

$$\nabla J_k^{\text{loc}}(\cdot) = -2\mathbb{E}\left\{\big(d_k(n) - \psi_k(\mathbf{u}_{k,n})\big)\kappa(\cdot, \mathbf{u}_{k,n})\right\} + 2\eta \sum_{l\neq k, l\in\mathcal{N}_k} \rho_{kl}\left(\psi_{k,n-1} - \psi_{l,n-1}\right) , \tag{32}$$

where $\kappa(\cdot,\cdot)$ is the reproducing kernel associated to $\mathcal{H}$. For simplicity, we assume that the mixing weights ρ_{kl} are symmetrical (see the discussion at the end of Section 2.3). Making an instantaneous approximation for the expectation gives us the following local update rule in functional form at time instant n:

$$\begin{aligned}\psi_{k,n} = \psi_{k,n-1} &+ \mu_k\left[d_k(n) - \psi_{k,n-1}(\mathbf{u}_{k,n})\right]\kappa(\cdot, \mathbf{u}_{k,n}) \\ &- \mu_k\eta \sum_{l\neq k, l\in\mathcal{N}_k} \rho_{kl}\left(\psi_{k,n-1} - \psi_{l,n-1}\right) ,\end{aligned} \tag{33}$$

where the factor 2 has been included in the step size μ_k. Considering $\mathcal{H}$ as the space of linear predictors over $\mathbf{u}_{k,n}$, then (33) reduces to the diffusion LMS for multitask networks presented earlier. In order to have a feasible implementation, once again we assume a shared dictionary $\mathcal{D}$ among agents. (33) reduces to:

$$\begin{aligned}\boldsymbol{\beta}_{k,n} = \boldsymbol{\beta}_{k,n-1} &+ \mu_k\left[d_k(n) - \boldsymbol{\beta}_{k,n-1}^T\mathbf{k}_{k,n}\right]\mathbf{k}_{k,n} \\ &- \mu_k\eta \sum_{l\neq k, l\in\mathcal{N}_k} \rho_{kl}\left(\boldsymbol{\beta}_{k,n-1} - \boldsymbol{\beta}_{l,n-1}\right) .\end{aligned} \tag{34}$$

5. Experimental evaluation

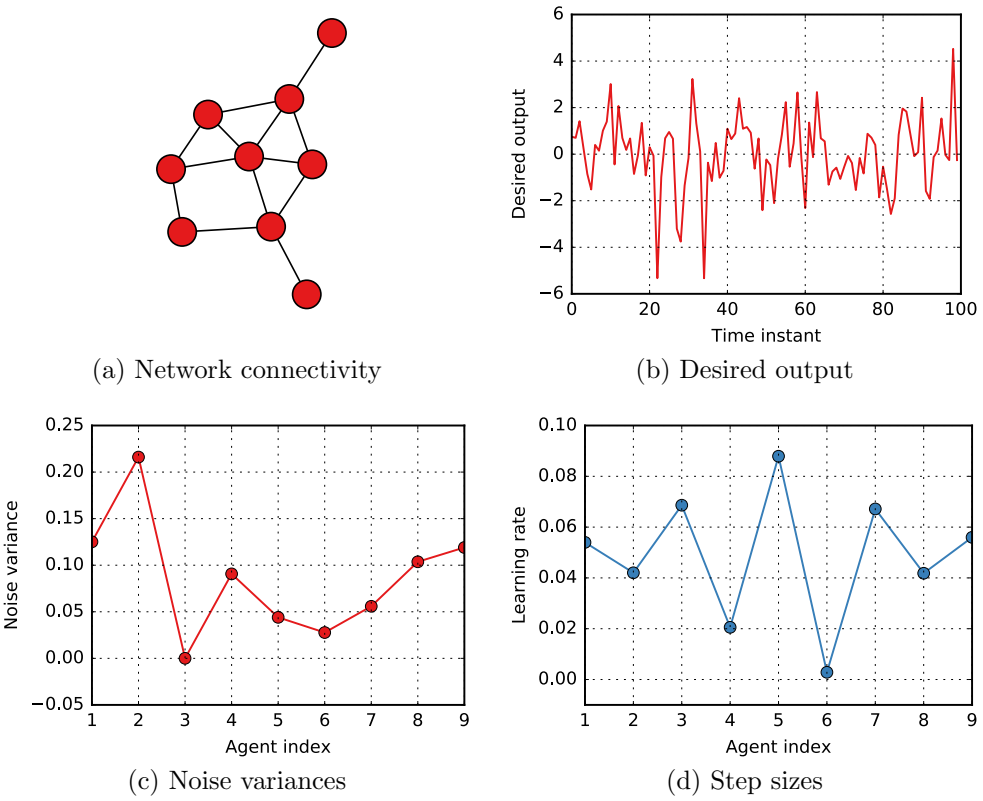
5.1. Experiment setup

In this section, we evaluate the proposed method on a simulated multitask nonlinear problem. The output at each agent is given by the following equation:

$$d_k(n) = f(\mathbf{u}_{k,n}) + \mathbf{w}_k^T\mathbf{u}_{k,n} + \nu_k(n) , \tag{35}$$

which is composed of a common nonlinear part $f(\cdot)$, a local linear part $\mathbf{w}_k^T\mathbf{u}_{k,n}$, and a local noise of variance σ_k^2. In particular, we considered a three dimensional input vector $\mathbf{u} = [u_1, u_2, u_3]^T$, with the following nonlinearity:

$$f(\mathbf{u}) = au_1^2 + bu_2u_3$$



(a) Network connectivity (b) Desired output

(c) Noise variances (d) Step sizes

Figure 5: General setup for the experimental section. (a) The network of 9 agents used in all experiments. (b) The first 100 samples of the desired output for the first agent. (c) Noise variance for each agent. (d) Learning rate for each agent. See the text for a description of how (b)-(d) were generated.

where a and b were generated from a normal distribution, similarly to the local coefficient vectors $\mathbf{w}_k$. Noise variances were generated uniformly for each agent in the interval $[0, 0.3]$. We added an additional level of diversity over the network by randomly assigning the learning rates to the agents from the uniform distribution over the interval $[0, 0.1]$. We considered a network of 9 agents, whose connectivity was randomly assigned such that each agent is connected in average with one fifth of the other agents, with the requirement that the overall graph is connected. The resulting network connectivity, an example of desired output, and a plot of the noise variances and learning rates, are all shown in Fig. 5.

We trained the network over a sequence of 1000 time instants, with white

Gaussian inputs with zero mean and unitary variance. The mixing matrix was chosen according to the max-degree heuristic:

$$A_{lk} = \begin{cases} \frac{1}{\deg_{\max}+1} & \text{if } l \text{ is connected to } k \\ 1 - \frac{\deg_k}{\deg_{\max}+1} & \text{if } k = l \\ 0 & \text{otherwise} \end{cases}, \tag{36}$$

where $\deg_k$ is the degree of node k, and $\deg_{\max}$ is the maximum degree of the network.[9] Each experiment was averaged over 500 different runs, by keeping fixed the assignments shown in Fig. 5.

5.2. Results and discussion

We compared the performance of a standard diffusion LMS (D-LMS), a multitask D-LMS as described in Section 2.3 (D-MT-LMS), the diffusion KLMS described in Section 3.2, and the proposed multitask D-KLMS introduced in Section 4 (D-MT-KLMS). For the kernel algorithms, we used a Gaussian kernel:

$$\mathcal{K}(\mathbf{u}_1, \mathbf{u}_2) = \exp\left\{-\gamma \left\|\mathbf{u}_1 - \mathbf{u}_2\right\|_2^2\right\},$$

where γ was chosen as the inverse of the dimensionality of $\mathbf{u}$, which was found to provide a good accuracy. For the multitask algorithms, the regularization coefficients were selected uniformly as:

$$\rho_{kl} = \begin{cases} \frac{1}{\deg_k} & \text{if } k \text{ is connected to } l \\ 0 & \text{otherwise} \end{cases}, \tag{37}$$

while the regularization factor was set to $\eta = 0.01$. For the kernel algorithms, we fixed *a priori* a dataset of size 100 with randomly extracted elements. The average MSE in dB across all runs is shown in Fig. 6.

As expected, the D-LMS was the poorest performing algorithm, due to the doubly incorrect assumptions that the agents share the same minimizer, and that the underlying function is linear. By relaxing one of the two assumptions, D-MT-LMS performed better, with an accuracy that is comparable to D-KLMS. Clearly, D-MT-KLMS was the best algorithm in this case,

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

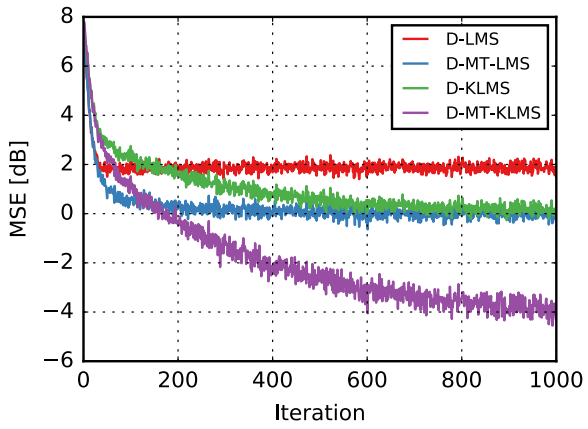


Figure 6: Average MSE for the algorithms under consideration, averaged both over agents and over 500 independent runs.

showing that it can be an effective solution for nonlinear multitask problems.

In Fig. 7 we show the MSE evolution for three representative agents. As expected, their performance is different depending on the selected learning rate and amount of noise, but the multitask algorithm is able to effectively combine the learning curves to obtain the average behavior as in the purple line of Fig. 6. Finally, in Fig. 8 we show the average MSE evolution when varying the size of the dictionary. Clearly, increasing the size improves the accuracy (up to a given upper bound), at the cost of a larger computational burden.

6. Discussion and open problems

Distributed inference is a fundamental tool according to today's technological trends. In the adaptive filtering community, many classical algorithms can be readily extended to the distributed scenario by exploiting diffusion principles, where local adaptation steps are interleaved with communication steps between neighbors. The resulting algorithms are both computationally efficient, and deployable over a large set of scenarios. In this chapter, we reviewed the basic tools of this field, and we briefly surveyed some of the nonlinear extensions that have been proposed.

An important distinction can be made between single-task problems, where all agents share the same minimizer, and multitask problems, where the minimizers can be different but it is known that they share some simi-

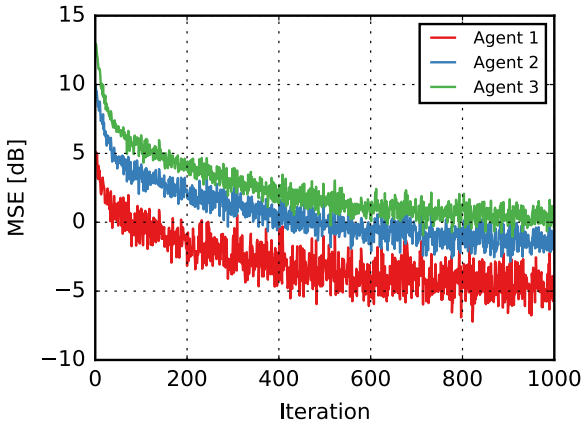


Figure 7: Local MSE averaged over 100 runs for 3 representative agents over the network.

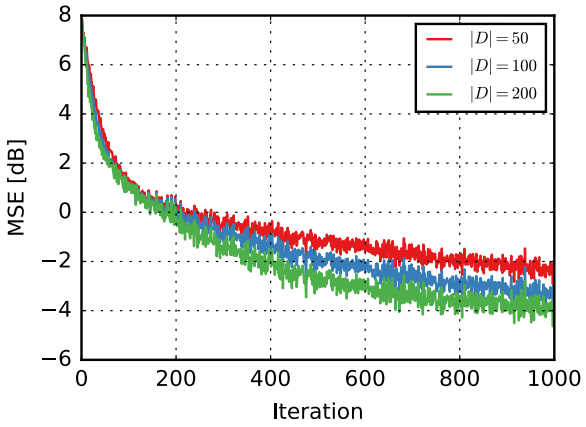


Figure 8: Average MSE for three different dictionary sizes.

larities. We underlined how very little work has been done on the nonlinear multitask case, and we proposed a simple kernel-based diffusion algorithm to this end. Many extensions over the basic setup of this chapter are possible, most notably a way to remove the assumption of a shared dictionary, an adaptive way to build the regularization coefficients, a theoretical analysis of the algorithm, or additional extensions towards asynchronous networks. Finally, we can consider mixing multitask networks with multi-objective algorithms [53], such that each agent is interested in minimizing multiple objectives simultaneously.

Acknowledgments

The work of Simone Scardapane was supported in part by Italian MIUR, "*Progetti di Ricerca di Rilevante Interesse Nazionale*", GAUChO project, under Grant 2015YPXH4W_004. The work of Jie Chen was supported in part by the National Natural Science Foundation of China (NSFC grant 61671382).

References

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

[2] A. Sandryhaila, J. M. F. Moura, Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure, IEEE Signal Processing Magazine 31 (5) (2014) 80–90.

[3] P. Di Lorenzo, S. Barbarossa, P. Banelli, S. Sardellitti, Adaptive least mean squares estimation of graph signals, IEEE Transactions on Signal and Information Processing over Networks 2 (4) (2016) 555–568.

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

[5] J. Tsitsiklis, D. Bertsekas, M. Athans, Distributed asynchronous deterministic and stochastic gradient optimization algorithms, IEEE Transactions on Automatic Control 31 (9) (1986) 803–812.

[6] A. Lazarevic, Z. Obradovic, The distributed boosting algorithm, in: Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 311–316.

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

[8] S. Scardapane, R. Fierimonte, P. Di Lorenzo, M. Panella, A. Uncini, Distributed semi-supervised support vector machines, Neural Networks 80 (2016) 43–52.

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

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

[11] C. G. Lopes, A. H. Sayed, Diffusion least-mean squares over adaptive networks: Formulation and performance analysis, IEEE Transactions on Signal Processing 56 (7) (2008) 3122–3136.

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

[13] J. Chen, A. H. Sayed, Diffusion adaptation strategies for distributed optimization and learning over networks, IEEE Transactions on Signal Processing 60 (8) (2012) 4289–4305.

[14] A. H. Sayed, Adaptive networks, Proceedings of the IEEE 102 (4) (2014) 460–497.

[15] A. H. Sayed, Adaptation, learning, and optimization over networks, Foundations and Trends in Machine Learning 7 (4-5) (2014) 311–801.

[16] J. Chen, S. K. Ting, C. Richard, A. H. Sayed, Group diffusion LMS, in: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2016, pp. 4925–4929.

[17] V. Matta, C. Richard, V. Saligrama, A. H. Sayed, Guest editorial inference and learning over networks, IEEE Transactions on Signal and Information Processing over Networks 2 (4) (2016) 423–425.

[18] J. Chen, C. Richard, A. H. Sayed, Multitask diffusion adaptation over networks, IEEE Transactions on Signal Processing 62 (16) (2014) 4129–4144.

[19] J. Chen, C. Richard, A. H. Sayed, Diffusion LMS over multitask networks, IEEE Transactions on Signal Processing 63 (11) (2015) 2733–2748.

[20] X. Zhao, A. H. Sayed, Distributed clustering and learning over networks, IEEE Transactions on Signal Processing 63 (13) (2015) 3285–3300.

[21] R. Nassif, C. Richard, A. Ferrari, A. H. Sayed, Multitask diffusion adaptation over asynchronous networks, IEEE Transactions on Signal Processing 64 (11) (2016) 2835–2850.

[22] R. Nassif, C. Richard, A. Ferrari, A. H. Sayed, Proximal multitask learning over networks with sparsity-inducing coregularization, IEEE Transactions on Signal Processing 64 (23) (2016) 6329–6344.

[23] C. Li, S. Huang, Y. Liu, Y. Liu, Distributed TLS over multitask networks with adaptive intertask cooperation, IEEE Transactions on Aerospace and Electronic Systems 52 (6) (2016) 3036–3052.

[24] J. Chen, C. Richard, A. O. Hero, A. H. Sayed, Diffusion LMS for multitask problems with overlapping hypothesis subspaces, in: 2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP), IEEE, 2014, pp. 1–6.

[25] J. Chen, C. Richard, A. Sayed, Multitask diffusion adaptation over networks with common latent representations, IEEE Journal of Selected Topics in Signal Processing.

[26] T. Evgeniou, M. Pontil, Regularized multi–task learning, in: Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2004, pp. 109–117.

[27] A. Argyriou, T. Evgeniou, M. Pontil, Multi-task feature learning, Advances in Neural Information Processing Systems 19 (2007) 41.

[28] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, R. Hadsell, Progressive neural networks, arXiv preprint arXiv:1606.04671.

[29] R. Jin, A. Goswami, G. Agrawal, Fast and exact out-of-core and distributed k-means clustering, Knowledge and Information Systems 10 (1) (2006) 17–40.

[30] X. Zhao, A. H. Sayed, Asynchronous adaptation and learning over networkspart i: Modeling and stability analysis, IEEE Transactions on Signal Processing 63 (4) (2015) 811–826.

[31] K. Yuan, B. Ying, X. Zhao, A. H. Sayed, Exact diffusion for distributed optimization and learning—part i: Algorithm development, arXiv preprint arXiv:1702.05122.

[32] S. Scardapane, D. Wang, Randomness in neural networks: An overview, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 7 (2).

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

[34] S. Scardapane, D. Wang, M. Panella, A. Uncini, Distributed learning for random vector functional-link networks, Information Sciences 301 (2015) 271–284.

[35] S. Huang, C. Li, Distributed extreme learning machine for nonlinear learning over network, Entropy 17 (2) (2015) 818–840.

[36] A. Rahimi, B. Recht, Random features for large-scale kernel machines., in: Advances in Neural Information Processing Systems, Vol. 3, 2007, pp. 1–5.

[37] A. Singh, N. Ahuja, P. Moulin, Online learning with kernels: Overcoming the growing sum problem, in: 2012 IEEE International Workshop on Machine Learning for Signal Processing (MLSP), IEEE, 2012, pp. 1–6.

[38] P. Bouboulis, S. Pougkakiotis, S. Theodoridis, Efficient KLMS and KRLS algorithms: A random Fourier feature perspective, in: 2016 IEEE Statistical Signal Processing Workshop (SSP), IEEE, 2016, pp. 1–5.

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

[40] P. Honeine, M. Essoloh, C. Richard, H. Snoussi, Distributed regression in sensor networks with a reduced-order kernel model, in: 2008 IEEE Global Telecommunications Conference (GLOBECOM), IEEE, 2008, pp. 1–5.

[41] C. Richard, J.-C. M. Bermudez, P. Honeine, Online prediction of time series data with kernels, IEEE Transactions on Signal Processing 57 (3) (2009) 1058–1067.

[42] W. D. Parreira, J.-C. M. Bermudez, C. Richard, J.-Y. Tourneret, Stochastic behavior analysis of the gaussian kernel least-mean-square algorithm, IEEE Transactions on Signal Processing 60 (5) (2012) 2208–2222.

[43] P. Honeine, C. Richard, J.-C. M. Bermudez, On-line nonlinear sparse approximation of functions, in: 2007 IEEE International Symposium on Information Theory (ISIT), IEEE, 2007, pp. 956–960.

[44] W. Gao, J. Chen, C. Richard, J. Huang, Diffusion adaptation over networks with kernel least-mean-square, in: 2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), IEEE, 2015, pp. 217–220.

[45] P. Bouboulis, S. Theodoridis, Extension of Wirtinger's calculus to reproducing kernel Hilbert spaces and the complex kernel LMS, IEEE Transactions on Signal Processing 59 (3) (2011) 964–978.

[46] A. V. Balakrishnan, Applied Functional Analysis, Springer Science & Business Media, 2012.

[47] J. Chen, W. Gao, C. Richard, J.-C. M. Bermudez, Convergence analysis of kernel LMS algorithm with pre-tuned dictionary, in: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2014, pp. 7243–7247.

[48] B.-S. Shin, H. Paul, A. Dekorsy, Distributed kernel least squares for nonlinear regression applied to sensor networks, in: 2016 24th European Signal Processing Conference (EUSIPCO), IEEE, 2016, pp. 1588–1592.

[49] S. Chouvardas, M. Draief, A diffusion kernel LMS algorithm for nonlinear adaptive networks, in: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2016, pp. 4164–4168.

[50] S. Scardapane, M. Scarpiniti, D. Comminiello, A. Uncini, Diffusion spline adaptive filtering, in: 2016 24th European Signal Processing Conference (EUSIPCO), IEEE, 2016, pp. 1498–1502.

[51] M. Scarpiniti, D. Comminiello, R. Parisi, A. Uncini, Nonlinear spline adaptive filtering, Signal Processing 93 (4) (2013) 772–783.

[52] M. Scarpiniti, D. Comminiello, G. Scarano, R. Parisi, A. Uncini, Steady-state performance of spline adaptive filters, IEEE Transactions on Signal Processing 64 (4) (2016) 816–828.

[53] J. Chen, A. H. Sayed, Distributed Pareto optimization via diffusion strategies, IEEE Journal of Selected Topics in Signal Processing 7 (2) (2013) 205–220.