Robust and Low Complexity Distributed Kernel Least Squares Learning in Sensor Networks

Fernando Pérez-Cruz, *Senior Member, IEEE*, and Sanjeev R. Kulkarni, *Fellow, IEEE*

***Abstract*—We present a novel mechanism for consensus building in sensor networks. The proposed algorithm has three main properties that make it suitable for sensor network learning. First, the proposed algorithm is based on robust nonparametric statistics and thereby needs little prior knowledge about the network and the function that needs to be estimated. Second, the algorithm uses only local information about the network and it communicates only with nearby sensors. Third, the algorithm is completely asynchronous and robust. It does not need to coordinate the sensors to estimate the underlying function and it is not affected if other sensors in the network stop working. Therefore, the proposed algorithm is an ideal candidate for sensor networks deployed in remote and inaccessible areas, which might need to change their objective once they have been set up.**

***Index Terms*—Consensus, distributed learning, kernel methods, sensor networks.**

I. Introduction

SENSORS networks are designed to infer in a decentralized manner aspects of the environment in which they have been deployed [2]. They are typically designed to solve a specific task under energy, communication, and computation constraints. Thereby, their communication, sensing, and inference algorithms need to be simple and reliable. This has triggered research in decentralized detection, estimation, and learning in network settings. There is substantial literature in decentralized detection for parametric models (e.g., see [3] and the references therein). But nonparametric methods are preferred if good parametric models are not know and/or if conditions change over time.

One promising approach for learning in sensor networks involves *message-passing algorithms* [4]. Message-passing algorithms were proposed for inference in graphical models, i.e. they compute the desired conditional marginal distribution, and they have been successfully applied in a number of applications [5], including channel coding [6]. On one hand, message-passing algorithms utilize only local information and local communication to produce global inference, making them ideal for sensor network inference. On the other hand, each message has to be particularized for each neighbor in the network, and if the network has many cycles with low girth they usually fail to return the correct inference [7]. Furthermore, each message conveys information about a full probability density function (pdf), which is a burdensome message to communicate to each neighbor, and they need prior information about the joint pdf with their neighboring sensors.

In this paper, we focus on nonparametric estimation, particularly on kernel methods for regression [8]. Kernel methods are universal classification and estimation algorithms. In [9], [10], a distributed kernel least square regression algorithm was proposed for inference in sensor networks. In this algorithm, each node sends the same massage to all its neighbors, reducing the communication burden from quadratic to linear in the number of neighbors, and its performance increases with the connectivity of the network. In this letter, we propose a variant of the algorithm in [10] (see Section III), that makes the communication burden independent of the number of neighbors and allows asynchronous updates. Thus, unlike the original algorithm, the network does not need to be synchronized and each sensor can train and transmit regardless of the status of other sensors.

II. Kernel Nonlinear Regression

We introduce the basic concepts of nonlinear regression, before presenting the distributed nonlinear regression estimator proposed in this paper. Given a training data set $\{\mathbf{x}_i, y_i\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, are drawn independently and identically distributed (i.i.d.) from $p(\mathbf{x}, y)$, we compute the functional relationship between the inputs, $\mathbf{x}$, and the output, y, that minimizes the mean loss between the output and our predictions, $f(\mathbf{x})$. Thereby, we minimize the risk functional:

$$R(f) = \int L\left(y, f(\mathbf{x})\right) p(\mathbf{x}, y) d\mathbf{x} dy \tag{1}$$

where $L(\cdot,\cdot)$ represents the loss that we incur when we predict $f(\mathbf{x})$ instead of y. If $p(\mathbf{x}, y)$ is unknown, one approach to directly estimate the function $f(\cdot)$ from the data is to use the empirical risk minimization principle [11]. Namely from a class of functions, we select $f(\cdot)$ that minimizes the empirical risk:

$$R_{emp}(f) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, f(\mathbf{x}_i)\right) \tag{2}$$

which is the sampled version of (1) over our training set. If the function class from which we choose $f(\cdot)$ has a finite VC (Vapnik–Chervonenkis) dimension, as the number of samples tends to infinity the minimum of (2) tends to the minimum of (1) over the class of functions [11].

Manuscript received November 30, 2009; revised January 07, 2010. First published January 22, 2010; current version published February 17, 2010. This work was supported in part by the U.S. Office of Naval Research under Grant N00014-07-1-0555, the U.S. Army Research Office under Grant W911NF-07-1-0185, the Spanish government (TEC2006-13514-C02-01/TCM, TEC2009-14504-C02-01, and CSD2008-00010). The work of F. Pérez-Cruz was supported by Marie Curie Fellowship 040883-AI-COM. This work was presented in part at ISIT 2009 [1]. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Hsiao-Chun Wu.

F. Pérez-Cruz is with Universidad Carlos III de Madrid, Madrid, Spain (e-mail: fernando@tsc.uc3m.es).

S. R. Kulkarni is with the Department of Electrical Engineering, Princeton University, Princeton NJ 08540 USA (e-mail: kulkarni@princeton.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LSP.2010.2040926

TABLE I
DKLS ALGORITHM

```
1) Initialization:
   a) Each sensor $i$ broadcasts their location $\mathbf{x}_i$ to neighboring sensors.
   b) Each sensor $i$ broadcasts their measurement $y_i$ to neighboring sensors.
   c) Each sensor $i$ initializes $z_k = y_k$, $\forall k \in N_i$.
   d) Each sensor $i$ initializes $f_{i,0} = \mathbf{0}$.
2) Training:
   For $t = 1, \dots, T$
      For $i = 1, \dots, N$
         $f_{i,t} = \operatorname{argmin}_f \sum_{k \in N_i} (f(\mathbf{x}_k) - z_k)^2 + \lambda_i \|f - f_{i,t-1}\|^2$
         Sensor $i$ broadcasts $f_{i,t}(\mathbf{x}_k)$ $\forall k \in N_i$.
         Every sensor $j \in N_i$ replaces $z_k$ by $f_{i,t}(\mathbf{x}_k)$ if $k \in N_j$.
      End
   End
```

Kernel methods were proposed in the 1990s to minimize (2) using a generalized linear regressor [8]. Kernel methods replace $f(\mathbf{x})$ by $\mathbf{w}^\top \phi(\mathbf{x})$, where $\phi(\mathbf{x})$ is a nonlinear transformation to a feature Hilbert space. They solve for $\mathbf{w}$ as follows:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, \mathbf{w}^\top \phi(\mathbf{x}_i)\right) + \lambda \|\mathbf{w}\|^2 \tag{3}$$

where we have added a term that penalizes large values of $\mathbf{w}$. This penalty term is a Tikhonov regularizer [12] that prevents the solution from overfitting the training data. The value of λ trades off the minimization of the empirical risk and the smoothness of the achieved solution.

If the loss function is convex in $\mathbf{w}^\top \phi(\mathbf{x})$, the Representer theorem [13] states that the optimal solution can be expressed as a linear combination of the training samples:

$$\mathbf{w}^* = \sum_{j=1}^{n} \alpha_j \phi(\mathbf{x}_j). \tag{4}$$

Now replacing (4) into (3), we obtain the following optimization problem:

$$\min_{\boldsymbol{\alpha}} \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, \sum_{j=1}^{n} \alpha_j k(\mathbf{x}_j, \mathbf{x}_i)\right) + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) \tag{5}$$

where we have replaced $\phi^\top(\mathbf{x}_j)\phi(\mathbf{x}_i)$ by its inner product or kernel $k(\mathbf{x}_j, \mathbf{x}_i)$ and $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_n]^\top$. When minimizing (5) instead of (3), one typically selects the kernel of the nonlinear transformation without explicitly defining or computing $\phi(\cdot)$, which greatly simplifies obtaining nonlinear regressors. Whether or not a given function $k(\cdot,\cdot)$ is a valid kernel (i.e., corresponds to an appropriate nonlinear transformation $\phi(\cdot)$) can be determined using Mercer's theorem [14].

III. DISTRIBUTED LEAST SQUARE REGRESSION

Predd *et al.* [10] proposed a distributed kernel least square (DKLS) regressor for consensus building in sensor networks with any topology and a single measurement. Their algorithm communicates sample estimates between neighboring nodes in the sensor network to yield a consensus about the field measurements. The DKLS algorithm builds a nonlinear regression model from the location of the nearby sensors and their measurements. The algorithm iterates throughout the sensors in the network until there is no further change in the function. Once consensus among the sensors has been reached, it is used to estimate the field.

This scheme is advantageous in several ways. First, there is no need to have a narrow parametric model for either the function to be estimated or the noise, because the DKLS regressor employs a robust kernel method for learning. Second, the prior knowledge needed about the network is quite limited. Each sensor only needs to know the location of its neighbors. Most of the methods mentioned in the introduction need further information about the network and its nodes, which make them less suitable when such information is unavailable and cannot be gathered. The DKLS algorithm solves a regularized least square functional for each sensor:

$$f_i^* = \arg\min_{f \in \mathcal{H}_K} \sum_{k \in N_i} (f(\mathbf{x}_k) - z_k)^2 + \lambda_i \|f - f_i\|^2_{\mathcal{H}_K} \tag{6}$$

where N_i is the set of neighbors[1] of the node i, $\mathcal{H}_K$ is the Reproducing Kernel Hilbert Space to which we restrict $f(\cdot)$ to be a member, $\mathbf{x}_k$ are the locations of the neighboring sensors and z_k is the latest prediction for the field measurement at $\mathbf{x}_k$, $f_i(\cdot)$ is the previous solution by the same sensor node. The regularizer λ_i ensures that the new estimate for the function at sensor i does not overfit the data. Although throughout this paper we consider only least square regression the proposed algorithm can be generalized to any other convex loss functions.

Once the algorithm obtains $f_i^*(\cdot)$, it broadcasts $f_i^*(\mathbf{x}_k)$ for $k \in N_i$ and they become the new z_k for the samples in N_i. For any sensor not in N_i the values of z_k remain unchanged for the next iteration. Once sensor i completes its training, we train a different node and repeat the procedure until convergence.

We show the outline of the DKLS algorithm in Table I and its convergence is analyzed in [10]. The number of iterations is typically fixed a priori, although some criteria can be imposed once $f_{i,t}$ does not change significantly from iteration to iteration.

Although this algorithm is beneficial in many ways, as explained earlier and in [10], it presents a few drawbacks. After each node completes its training, the node needs to communicate its predictions to all its neighbors. The communication burden grows with the size of N_i. For dense networks, communicating z_k to all neighbors can be a significant load. Also, as not all neighbors of i are neighbors among themselves, the nodes have to discard some of the received values and they need to identify which ones.

We also need a synchronization procedure to train the nodes, as we can only train one node at the time[2]. Otherwise a node might received two or more updates for z_k and it does not know which one it needs to keep. Finally, if a sensor stops working the learning procedure stops, as the next node does not receive new predictions and it does not start its training phase until it does.

We propose a simple modification to the update rule in the DKLS algorithm that solves most of these limitations and that

[1]The sensor i itself is contained in N_i.

[2]We can train several nodes at the same time, if the union of their set of neighbors is the empty set. But this further complicates the synchronization procedure.

TABLE II
MODIFIED DKLS ALGORITHM

```
1) Initialization:
    a) Each sensor $i$ broadcasts their location $\mathbf{x}_i$ to neighboring sensors.
    b) Each sensor $i$ broadcasts their measurement $y_i$ to neighboring sensors.
    c) Each sensor $i$ initializes $z_k = y_k, \forall k \in N_i$.
    d) Each sensor $i$ initializes $f_{i,0} = \mathbf{0}$.
2) Training:
    For $t = 1, \dots, T$
        For $i = 1, \dots, N$
            $f_{i,t} = \operatorname{argmin}_f \sum_{k \in N_i} (f(\mathbf{x}_k) - z_k)^2 + \lambda_i \|f - f_{i,t-1}\|^2$
            Sensor $i$ broadcasts $f_{i,t}(\mathbf{x}_i)$.
            Every sensor $j \in N_i$ replaces $z_i$ by $f_{i,t}(\mathbf{x}_i)$.
        End
    End
```

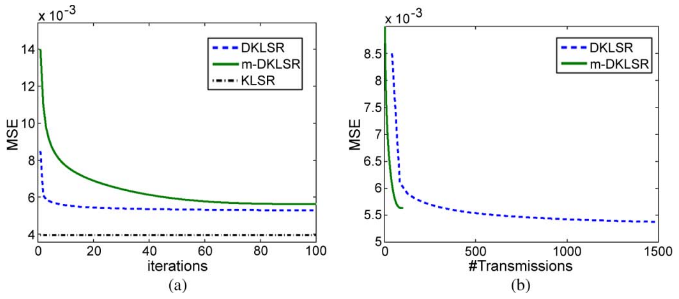


Fig. 1. In (a), we show MSE as a function of the number of iterations, and in (b) the MSE as a function of the number of transmissions.

presents some additional advantages of its own. Instead of transmitting $f_{i,t}(\mathbf{x}_k)$ for $k \in N_i$, we broadcast only $f_{i,t}(\mathbf{x}_i)$, the prediction for the current node. We show the algorithmic procedure for the modified DKLS (m-DKSL) algorithm in Table II, where we have placed in bold the two lines that contain the changes with respect to the algorithm in Table I, and whose computational complexity are identical.

First, we reduce the communication burden from N_i to 1 and this burden does not depend on the connectivity of the network. This reduction is significant, especially for highly connected networks for which the transmission burden grows quadratically with the number of sensors.

Since we broadcast only $f_i^*(\mathbf{x}_i)$, there is no need to synchronize the network. Each node decides when it wants to retrain and it transmits when it finishes. We only need a communication protocol to avoid collisions (e.g., Aloha). Each sensor can work with its own clock.

Finally, for our algorithm we do not need to have a predefined neighborhood for each node. For far away nodes, we might have a working connection in some instances and but not in others. This does not require changes to the learning procedure. The sensor has more information if it is available. Also, if a sensor stops transmitting, its neighbors can simply stop using that sensor's output and this does not stop the learning procedure from working.

IV. EXPERIMENTS

In this section, we empirically investigate the proposed modification for the DKLS algorithm and we compare it with the KLS solution, which is the best solution we can expect for the DKLS and m-DKLS regressors. We have carried out three different experiments. In all of them, we solve a 2-D problem in which the nodes are randomly deployed between -1 and 1 in both dimensions and the field measurements are give by

$$y_i = \exp\left(-\|\mathbf{x}_i\|^2\right) + n_i \tag{7}$$

where n_i is a zero-mean Gaussian variable with variance σ_n^2.

We have set the regularizer $\lambda_i = \kappa/|N_i|^2$, as proposed in [10], where $|\cdot|$ denotes the cardinality of a set. We have used a Gaussian nonlinear kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) \tag{8}$$

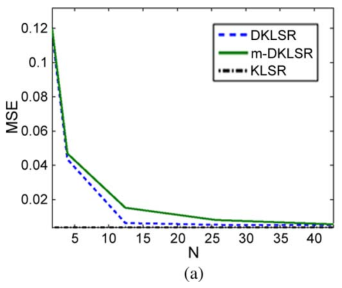
This is a universal smooth kernel that can be used in learning general nonlinear functions and that gives good results in many situations. For further details about kernels and kernel design refer to [8].

For all our experiments we have run 10 independent trials and we have chosen the hyperparameters by cross validation [15]. We have selected κ from a range between 10^{-4} and 10^4 and we have selected σ between 0.25 and 8 and we have set the noise standard deviation to $\sigma_n = 0.5$. In all of our experiments, we have run the DKLS and the m-DKLS algorithms for 100 iterations and we have defined a spherical neighborhood with the same radius for all the samples.

In Fig. 1(a), we have depicted the mean value of the mean square error (MSE) for 400 test samples (independent from the training set) for our ten independent trails as a function of the iterations for the DKLS and the m-DKLS algorithms. We have set the neighborhood radius to 0.4. We have drawn 400 training examples and the mean number of neighbors per node is 42.78.

The DKLS algorithm seems to converge faster than the m-DKLS algorithm, and it is slightly closer to the KLS solution when plotted as a function of the number of iterations. However, the difference in performance (in MSE and number of iterations) can be explained by the update rule proposed in Section III. In these examples, the m-DKLS algorithm transmits approximately 42 times less information between the nodes per iteration. To provide a fairer comparison, in Fig. 1(b) we show the MSE as a function of the number of transmissions for both algorithms. In it we can see that for any given MSE value the m-DKLS algorithm needs far fewer transmissions than the DKLS algorithm. Furthermore, the m-DKLS regressor can be trained in parallel and asynchronously, while the DKLS algorithm needs a synchronization procedure and can only train one node at the time. Therefore the m-DKLS algorithm can achieve its solution in a shorter time and with less communication between nodes than the DKLS regressor. The price we pay is a slightly increased MSE with respect to the results achieved by the DKLS solution.

To understand the MSE gap between the DKLS and the m-DKLS algorithms we have repeated the experiment with different numbers of neighbors. In Fig. 2(a), we plot the results for the experiment for 400 sensor nodes as a function of the mean number of neighbors and in Fig. 2(b) we plot the mean number of transmissions as a function of mean number of neighbors.



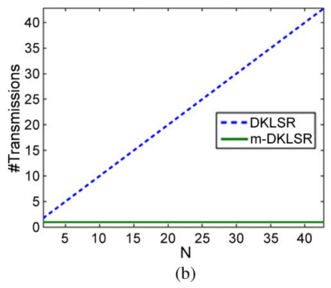
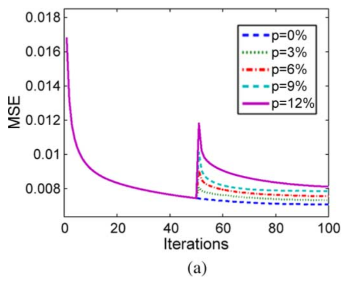


Fig. 2. MSE for the 100th iteration and mean number of transmissions per node, respectively, in (a) and (b) as a function of the number of neighbors.



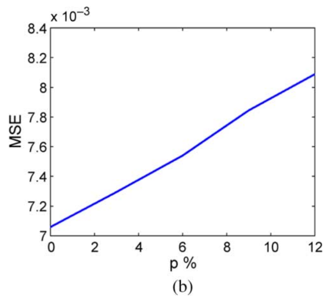


Fig. 3. In (a), we show the MSE as a function of the number of iterations for five different percentages (denoted by p) of nodes failing at iteration 50. In (b), we show the MSE after 100 iterations as a function of the percentage of node failures at iteration 50.

There is a range in which the DKLS algorithm performs slightly better in terms of MSE than our proposed modification. In this range, the updating rule that informs all the neighboring nodes about its predictions works considerably better. We believe that in this range the updating rule of the DKLS algorithm, which updates all z_k from the same predictions instead of the updates coming from different nodes, provides the subsequent nodes with less noisy predictions. In any case, to get an accurate prediction for both algorithms we need to increase the number of nodes or the neighborhood sufficiently so both algorithms perform equally well.

In Fig. 2(b), we can see that the number of mean transmissions per node grows linearly with the number of neighbors for the DKLS algorithm and it remains constant and equal to 1 for the m-DKLS algorithm. In these plots we can understand the main advantage of the m-DKLS algorithm. To get an accurate prediction, we need a large neighborhood and a dense network. In this case, the communication burden of the DKLS algorithm is quite high as the communication burden grows quadratically with the number of nodes in the network per iteration, while the communication burden of the m-DKLS algorithm only grows linearly with the number of nodes in the network.

Finally in the third experiment, we illustrate the robustness of the proposed m-DKLS algorithm. We have repeated the first experiment, but now, at iteration 50, $p\%$ of the sensors fail and the network has to adjust to this failure and continue learning without them. If this were to happen in the original DKLS algorithm, the algorithm would stop its learning procedure, once one of the failed nodes needs to retrain.

In Fig. 3(a), we show the MSE as a function of the number of transmission for $p = 0, 3, 6, 9$, and 12%. We can see that after the node failures, we have a significant increase in the reported MSE, but the networks recover from it and it is able to continue learning and report performances similar to the full network. Of course the results are not as good, because the network only posses $(1-p/100)N$ sensors, which limits the minimum reachable MSE. In Fig. 3(b), we show the degraded MSE at iteration 100, we can notice that when we lose 48 nodes, the performance degrades less than 15%. The proposed algorithm is able not only to learn in a completely decentralized manner, but it is able to adjust to sensor failures.

V. Conclusions

We have presented a modification to the DKLS algorithm from [10] that that achieves consensus with considerably less communication among the nodes without significantly compromising error performance. The DKLS algorithm is based on robust nonlinear estimation and the m-DKLS algorithm inherits the same characteristics. Moreover, while DKLS needs to synchronize the training in the nodes to avoid inconsistencies, an additional an ancillary advantage of the m-DKLS algorithm is that it can operate asynchronously. The consequence is that we can train the nodes independently at any time and they do not need to wait for other nodes to communicate their predictions. This advantage increases the tolerance to node failure for a network that uses m-DKLS.

References

[1] F. Perez-Cruz and S. R. Kulkarni, "Distributed least square for consensus building in sensor networks," in *Int. Symp. Information Theory*, Seoul, Korea, Jul. 2009.
[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–114, Aug. 2002.
[3] J. Tsitsiklis, "Decentralized detection," in *Advances in Statistical Signal Processing*, H. Poor and J. Thomas, Eds. Greenwich, CT: JAI, 1993, pp. 297–344.
[4] M. Çetin, L. Chen, J. W. Fisher, A. T. Ihler, R. L. Moses, M. J. Wainwright, and A. S. Willsky, "Distributed fusion in sensor networks," *IEEE Signal Process. Mag.*, vol. 56, pp. 42–55, Jul. 2006.
[5] B. J. Frey*, Graphical Models for Machine Learning and Digital Communication*. New York: MIT Press, 1998.
[6] D. J. C. MacKay*, Information Theory, Inference and Learning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2003.
[7] , M. I. Jordan, Ed.*, Learning in Graphical Models*. Cambridge, MA: MIT Press, 1999.
[8] B. Schölkopf and A. Smola*, Learning With Kernels*. Cambridge, MA: MIT Press, 2002.
[9] J. Predd, S. Kulkarni, and H. Poor, "Regression in sensor networks: Training distributively with alternating projections," in *SPIE Conf. Advanced Signal Processing Algorithms, Architectures, and Implementations XV*, San Diego, Jul.–Aug. 2005 [Online]. Available: http://www.tsc.uc3m.es/~fernando/Predd.pdf
[10] J. Predd, S. Kulkarni, and H. Poor, "A collaborative training algorithm for distributed learning," *IEEE Trans. Inform. Theory*, vol. 55, pp. 1856–1871, Apr. 2009.
[11] V. N. Vapnik*, Statistical Learning Theory*. New York: Wiley, 1998.
[12] A. N. Tikhonov and V. Y. Arsenin*, Solution of Ill-Posed Problems*. Washington, DC: Winston, 1977.
[13] G. S. Kimeldorf and G. Wahba, "Some results in tchebycheffian spline functions," *J. Math. Anal. Applicat.*, vol. 33, pp. 82–95, 1971.
[14] J. Mercer, "Functions of positive and negative type, and their connection with the theory of integral equations," *Philos. Trans. Roy. Soc. London*, vol. 209, 1909.
[15] G. Wahba and S. Wold, "A completely automatic french curve: Fitting spline functions by cross-validation," *Commun. Statist.*, ser. A4, no. 1, pp. 257–263, 1975.