Brief Papers

Online Training of LSTM Networks in Distributed Systems for Variable Length Data Sequences

Tolga Ergen and Suleyman S. Kozat, *Senior Member, IEEE*

***Abstract*— In this brief, we investigate online training of long short term memory (LSTM) architectures in a distributed network of nodes, where each node employs an LSTM-based structure for online regression. In particular, each node sequentially receives a variable length data sequence with its label and can only exchange information with its neighbors to train the LSTM architecture. We first provide a generic LSTM-based regression structure for each node. In order to train this structure, we put the LSTM equations in a nonlinear state-space form for each node and then introduce a highly effective and efficient distributed particle filtering (DPF)-based training algorithm. We also introduce a distributed extended Kalman filtering-based training algorithm for comparison. Here, our DPF-based training algorithm guarantees convergence to the performance of the optimal LSTM coefficients in the mean square error sense under certain conditions. We achieve this performance with communication and computational complexity in the order of the first-order gradient-based methods. Through both simulated and real-life examples, we illustrate significant performance improvements with respect to the state-of-the-art methods.**

***Index Terms*— Distributed learning, extended Kalman filtering (EKF), long short term memory (LSTM) networks, online learning, particle filtering.**

I. Introduction

Neural networks provide enhanced performance for a wide range of engineering applications, e.g., prediction [1] and human behavior modeling [2], thanks to their highly strong nonlinear modeling capabilities. Among neural networks, especially recurrent neural networks (RNNs) are used to model time series and temporal data due to their inherent memory storing the past information [3]. However, since simple RNNs lack control structures, the norm of gradient may grow or decay in a fast manner during training, i.e., the exploding and vanishing gradient issues [4]. Due to these problems, simple RNNs are insufficient to capture long- and short-term dependences [4]. To circumvent this issue, a novel RNN architecture with control structures, i.e., the long short term memory (LSTM) network, is introduced [5]. However, since LSTM networks have additional nonlinear control structures with several parameters, they may also suffer from training issues [5]. To circumvent these issues, we introduce highly effective and efficient training methods for the LSTM architecture.

To this end, in this brief, we consider online training of the parameters of an LSTM structure in a distributed network of nodes. Here, we have a network of nodes, where each node has a set of neighboring nodes and can only exchange information with these neighbors. In particular, each node sequentially receives a variable length data sequence with its label and trains the parameters of the LSTM network. Each node can also communicate with its neighbors to share information in order to enhance the training performance, since the goal is to train one set of LSTM coefficients using all the available data. As an example application, suppose that we have a database of labeled tweets and our aim is to train an emotion recognition engine based on an LSTM structure, where the training is performed in an online and distributed manner using several processing units. Words in each tweet are represented by word2vec vectors [6] and tweets are distributed to several processing units in an online manner.

The LSTM architectures are usually trained in a batch setting in the literature, where all data instances are present and processed together [3]. However, for applications involving big data, storage issues may arise due to keeping all the data in one place [7]. In addition, in certain frameworks, all data instances are not available beforehand, since instances are received in a sequential manner, which precludes batch training [7]. Hence, we consider online training, where we sequentially receive the data to train the LSTM architecture without storing the previous data instances. Note that even though we work in an online setting, we may still suffer from computational power and storage issues due to large amount of data [8]–[10]. As an example, in tweet emotion recognition applications, the systems are usually trained using an enormous amount of data to achieve sufficient performance, especially for agglutinative languages [6]. For such tasks, distributed architectures are used. In this basic distributed architectures, commonly named as a centralized approach [8], the whole data is distributed to different nodes and trained parameters are merged later at a central node [3]. However, this centralized approach requires high storage capacity and computational power at the central node [8]. In addition, centralized strategies have a potential risk of failure at the central node. To circumvent these issues, we distribute both the processing as well as the data to all the nodes and allow communication only between neighboring nodes; hence, we remove the need for a central node. In particular, each node sequentially receives a variable length data sequence with its label and exchanges information only with its neighboring nodes to train the common LSTM parameters.

For online training of the LSTM architecture in a distributed manner, one can employ one of the first-order gradient-based algorithms at each node due to their efficiency [3]. In the distributed implementation of the first-order gradient-based methods, each node exchanges either its estimate or its first-order gradient with its neighboring nodes in order to compute the final estimate, e.g., [11] directly shares the estimates at each node with its neighbors and then updates the linear combination of the estimates to get the final estimate. However, since these training methods only exploit the first-order gradient information, they suffer from poor performance and convergence issues. As an example, the stochastic gradient descent (SGD)-based algorithms usually have slower convergence compared with the second-order methods [3], [11]. On the other hand, the second-order gradient-based methods require much higher computational complexity and communication load while providing superior performance compared with the first-order methods [3]. Following the distributed implementation of the first-order methods, one can implement the second-order training methods in a distributed manner, where we share not only the estimates but also the Jacobian matrix, e.g., the distributed extended Kalman filtering

Manuscript received June 15, 2017; revised September 8, 2017; accepted November 1, 2017. Date of publication December 7, 2017; date of current version September 17, 2018. This work was supported by TUBITAK under Contract 115E917. *(Corresponding author: Tolga Ergen.)*

The authors are with the Department of Electrical and Electronics Engineering, Bilkent University, 06800 Ankara, Turkey (e-mail: ergen@ee.bilkent.edu.tr; kozat@ee.bilkent.edu.tr).

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/TNNLS.2017.2770179

(DEKF) algorithm [12], [13]. However, as in the first-order case, these sharing and combining the information at each node are ad hoc, which does not provide the optimal training performance [12]. In addition to the EKF algorithm, the Hessian-free (HF) [14] and quasi-Newton (QN) [15] algorithms are also employed as the second-order training methods for training RNNs. The HF algorithm avoids the direct Hessian computations, i.e., highly complex computations, so that it significantly reduces its computational complexity while enjoying high performance provided by a second-order method [14]. Similarly, the QN algorithm approximately computes Hessian to save computational resources [15]. However, both of these algorithms provide restricted performances in online tasks [3], [16]. In this brief, to provide improved performance with respect to the second-order methods while preserving both communication and computational complexity similar to the first-order methods, we introduce a highly effective distributed online training method based on the particle filtering algorithm [17]. We first propose an LSTM-based model for variable length data regression. We then put this model in a nonlinear state-space form to train the model in an online and optimal manner.

Our main contributions include the following.

1) We introduce distributed LSTM training methods in an online setting for variable length data sequences. Our distributed particle filtering (DPF)-based training algorithm guarantees convergence to the optimal centralized training performance in the mean square error (MSE) sense.
2) We achieve this performance with a computational complexity and a communication load in the order of the first-order gradient-based methods.
3) Through simulations involving real life and financial data, we illustrate significant performance improvements with respect to the state-of-the-art methods [16], [18].

This brief is organized as follows. In Section II, we first describe the variable length data regression problem in a network of nodes and then introduce an LSTM-based structure. Then, in Section III, we first put this structure in a nonlinear state-space form and then introduce our training algorithms. In Section IV, we illustrate the merits of our algorithms through simulations. We then finalize this brief with concluding remarks in Section V.

II. Model and Problem Description

Here,[1] we consider a network of K nodes. In this network, we declare two nodes that can exchange information as neighbors and denote the neighborhood of each node k as $\mathcal{N}_k$ that also includes node k, i.e., $k \in \mathcal{N}_k$. At each node k, we sequentially receive $\{d_{k,t}\}_{t\geq 1}$, $d_{k,t} \in \mathbb{R}$ and matrices, $\{\boldsymbol{X}_{k,t}\}_{t\geq 1}$, defined as $\boldsymbol{X}_{k,t} = [\boldsymbol{x}_{k,t}^{(1)}\ \boldsymbol{x}_{k,t}^{(2)} \ldots \boldsymbol{x}_{k,t}^{(m_t)}]$, where $\boldsymbol{x}_{k,t}^{(l)} \in \mathbb{R}^p$, $\forall l \in \{1, 2, \ldots, m_t\}$ and $m_t \in \mathbb{Z}^+$ is the number of columns in $\boldsymbol{X}_{k,t}$, which can change with respect to t. In our network, each node k aims to learn a certain relation between the desired value $d_{k,t}$ and matrix $\boldsymbol{X}_{k,t}$. After observing $\boldsymbol{X}_{k,t}$ and $d_{k,t}$, each node k first updates its belief about the relation and then exchanges an updated information with its neighbors. After receiving $\boldsymbol{X}_{k,t+1}$, each node k estimates the next signal $d_{k,t+1}$ as $\hat{d}_{k,t+1}$. Based on $d_{k,t+1}$, each node k suffers the loss $l(d_{k,t+1}, \hat{d}_{k,t+1})$ at time instance $t + 1$. This framework models a wide range of applications in the machine learning and signal processing literature, e.g., sentiment analysis [6]. As an example, in tweet emotion recognition application [6], each $\boldsymbol{X}_{k,t}$ corresponds to a tweet, i.e., the tth tweet at the node (processing unit) k. For the tth tweet at node k, one can construct $\boldsymbol{X}_{k,t}$ by finding word2vec representation of each word, i.e., $\boldsymbol{x}_{k,t}^{(l)}$ for the lth word. After receiving $d_{k,t}$, i.e., the desired emotion label for the tth tweet at node k, each node k first updates its belief about the relation between the tweet and its emotion label and then exchanges information, e.g., the trained system parameters, with its neighboring units to estimate the next label.

[1]All column vectors (or matrices) are denoted by boldface lower (or uppercase) case letters. For matrix $\boldsymbol{A}$ (or vector $\boldsymbol{a}$), $\boldsymbol{A}^T$ ($\boldsymbol{a}^T$) is its ordinary transpose. The time index is given as subscript, e.g., $\boldsymbol{a}_t$ is the vector at time t. Here, $\boldsymbol{1}$ (or $\boldsymbol{0}$) is a vector of all ones (or zeros) and $\boldsymbol{I}$ is the identity matrix, where the sizes of these notations are understood from the context.

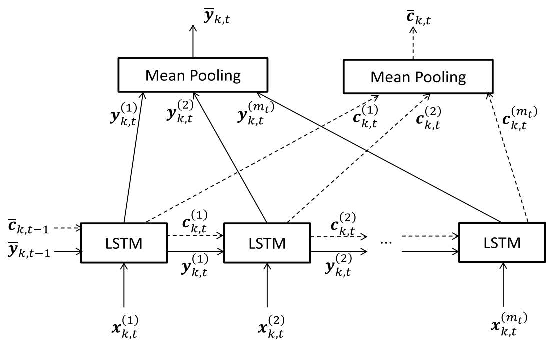
Mean Pooling
Mean Pooling
LSTM
LSTM
LSTM
$\bar{\boldsymbol{y}}_{k,t}$
$\bar{\boldsymbol{c}}_{k,t}$
$\boldsymbol{y}_{k,t}^{(1)}$
$\boldsymbol{y}_{k,t}^{(2)}$
$\boldsymbol{y}_{k,t}^{(m_t)}$
$\boldsymbol{c}_{k,t}^{(1)}$
$\boldsymbol{c}_{k,t}^{(2)}$
$\boldsymbol{c}_{k,t}^{(m_t)}$
$\bar{\boldsymbol{c}}_{k,t-1}$
$\bar{\boldsymbol{y}}_{k,t-1}$
$\boldsymbol{x}_{k,t}^{(1)}$
$\boldsymbol{x}_{k,t}^{(2)}$
$\boldsymbol{x}_{k,t}^{(m_t)}$

Fig. 1. Detailed schematic of each node k in our network.

In this brief, each node k generates an estimate $\hat{d}_{k,t}$ using the LSTM architecture. Although there exist different variants of LSTM, we use the most widely used variant [5], i.e., the LSTM architecture without peephole connections. The input $\boldsymbol{X}_{k,t}$ is first fed to the LSTM architecture as illustrated in Fig. 1, where the internal equations are given as [5]

$$\boldsymbol{i}_{k,t}^{(l)} = \sigma\big(\boldsymbol{W}_k^{(i)}\boldsymbol{x}_{k,t}^{(l)} + \boldsymbol{R}_k^{(i)}\boldsymbol{y}_{k,t}^{(l-1)} + \boldsymbol{b}_k^{(i)}\big) \tag{1}$$

$$\boldsymbol{f}_{k,t}^{(l)} = \sigma\big(\boldsymbol{W}_k^{(f)}\boldsymbol{x}_{k,t}^{(l)} + \boldsymbol{R}_k^{(f)}\boldsymbol{y}_{k,t}^{(l-1)} + \boldsymbol{b}_k^{(f)}\big) \tag{2}$$

$$\boldsymbol{c}_{k,t}^{(l)} = \boldsymbol{i}_{k,t}^{(l)} \odot g\big(\boldsymbol{W}_k^{(z)}\boldsymbol{x}_{k,t}^{(l)} + \boldsymbol{R}_k^{(z)}\boldsymbol{y}_{k,t}^{(l-1)} + \boldsymbol{b}_k^{(z)}\big) + \boldsymbol{f}_{k,t}^{(l)} \odot \boldsymbol{c}_{k,t}^{(l-1)} \tag{3}$$

$$\boldsymbol{o}_{k,t}^{(l)} = \sigma\big(\boldsymbol{W}_k^{(o)}\boldsymbol{x}_{k,t}^{(l)} + \boldsymbol{R}_k^{(o)}\boldsymbol{y}_{k,t}^{(l-1)} + \boldsymbol{b}_k^{(o)}\big) \tag{4}$$

$$\boldsymbol{y}_{k,t}^{(l)} = \boldsymbol{o}_{k,t}^{(l)} \odot h\big(\boldsymbol{c}_{k,t}^{(l)}\big) \tag{5}$$

where $\boldsymbol{x}_{k,t}^{(l)} \in \mathbb{R}^p$ is the input vector, $\boldsymbol{y}_{k,t}^{(l)} \in \mathbb{R}^n$ is the output vector, and $\boldsymbol{c}_{k,t}^{(l)} \in \mathbb{R}^n$ is the state vector for the l^{th} LSTM unit. Moreover, $\boldsymbol{o}_{k,t}^{(l)}$, $\boldsymbol{f}_{k,t}^{(l)}$, and $\boldsymbol{i}_{k,t}^{(l)}$ represent the output, forget, and input gates, respectively. $g(\cdot)$ and $h(\cdot)$ are set to the hyperbolic tangent function and apply vectors pointwise. Likewise, $\sigma(\cdot)$ is the pointwise sigmoid function. The operation $\odot$ represents the elementwise multiplication of two vectors of the same size. As the coefficient matrices and the weight vectors of the LSTM architecture, we have $\boldsymbol{W}_k^{(\cdot)}$, $\boldsymbol{R}_k^{(\cdot)}$, and $\boldsymbol{b}_k^{(\cdot)}$, where the sizes are chosen according to the input and output vectors. Given the outputs of LSTM for each column of $\boldsymbol{X}_{k,t}$ as shown in Fig. 1, we generate the estimate for each node k as follows:

$$\hat{d}_{k,t} = \boldsymbol{w}_{k,t}^T \bar{\boldsymbol{y}}_{k,t} \tag{6}$$

where $\boldsymbol{w}_{k,t} \in \mathbb{R}^n$ is a vector of the regression coefficients and $\bar{\boldsymbol{y}}_{k,t} \in \mathbb{R}^n$ is a vector obtained by taking average of the LSTM outputs for each column of $\boldsymbol{X}_{k,t}$, i.e., known as the mean pooling method, as described in Fig. 1.

Remark 1: In (6), we use the mean pooling method to generate $\bar{\boldsymbol{y}}_{k,t}$. One can also use the other pooling methods by changing the calculation of $\bar{\boldsymbol{y}}_{k,t}$ and then generate the estimate as in (6). As an example, for the max and last pooling methods, we use $\bar{\boldsymbol{y}}_{k,t} = \max_i \boldsymbol{y}_{k,t}^{(i)}$ and $\bar{\boldsymbol{y}}_{k,t} = \boldsymbol{y}_{k,t}^{(m_t)}$, respectively. All our derivations hold for these pooling methods and the other LSTM architectures. We provide the required updates for different LSTM architectures in Section III.

III. Online Distributed Training Algorithms

In this section, we first give the LSTM equations for each node in a nonlinear state-space form. Based on this form, we then introduce our distributed algorithms to train the LSTM parameters in an online manner.

Considering our model in Fig. 1 and the LSTM equations in (1)–(5), we have the following nonlinear state-spaces form for each node k :

$$\begin{aligned} \bar{\boldsymbol{c}}_{k,t} &= \Omega(\bar{\boldsymbol{c}}_{k,t-1}, \boldsymbol{X}_{k,t}, \bar{\boldsymbol{y}}_{k,t-1}) && (7)\\ \bar{\boldsymbol{y}}_{k,t} &= \Theta(\bar{\boldsymbol{c}}_{k,t}, \boldsymbol{X}_{k,t}, \bar{\boldsymbol{y}}_{k,t-1}) && (8)\\ \boldsymbol{\theta}_{k,t} &= \boldsymbol{\theta}_{k,t-1} && (9)\\ d_{k,t} &= \boldsymbol{w}_{k,t}^T \bar{\boldsymbol{y}}_{k,t} + \varepsilon_{k,t} && (10)\end{aligned}$$

where $\Omega(\cdot)$ and $\Theta(\cdot)$ represent the nonlinear mappings performed by the consecutive LSTM units and the mean pooling operation as illustrated in Fig. 1, and $\boldsymbol{\theta}_{k,t} \in \mathbb{R}^{n_\theta}$ is a parameter vector consisting of $\{\boldsymbol{w}_k, \boldsymbol{W}_k^{(z)}, \boldsymbol{R}_k^{(z)}, \boldsymbol{b}_k^{(z)}, \boldsymbol{W}_k^{(i)}, \boldsymbol{R}_k^{(i)}, \boldsymbol{b}_k^{(i)}, \boldsymbol{W}_k^{(f)}, \boldsymbol{R}_k^{(f)}, \boldsymbol{b}_k^{(f)}, \boldsymbol{W}_k^{(o)}, \boldsymbol{R}_k^{(o)}, \boldsymbol{b}_k^{(o)}\}$, where $n_\theta = 4n(n+p) + 5n$. Since the LSTM parameters are the states of the network to be estimated, we also include the static equation (9) as our state. Furthermore, $\varepsilon_{k,t}$ represents the error in observations and it is a zero-mean Gaussian random variable with variance $R_{k,t}$.

Remark 2: We can also apply the introduced algorithms to different implementations of the LSTM architecture [5]. For this purpose, we modify the function $\Omega(\cdot)$ and $\Theta(\cdot)$ in (7) and (8) according to the chosen LSTM architecture. We also alter $\boldsymbol{\theta}_{k,t}$ in (9) by adding or removing certain parameters according to the chosen LSTM architecture.

A. Online Training Using the DEKF Algorithm

In this section, we first derive our training method based on the EKF algorithm, where each node trains its LSTM parameters without any communication with its neighbors. We then introduce our training method based on the DEKF algorithm in order to train the LSTM architecture when we allow communication between the neighbors.

The EKF algorithm is based on the assumption that the state distribution, given the observations is Gaussian [13]. To meet this assumption, we introduce Gaussian noise to (7)–(9). By this, we have the following model for each node k :

$$\begin{bmatrix} \bar{\boldsymbol{c}}_{k,t} \\ \bar{\boldsymbol{y}}_{k,t} \\ \boldsymbol{\theta}_{k,t} \end{bmatrix} = \begin{bmatrix} \Omega(\bar{\boldsymbol{c}}_{k,t-1}, \boldsymbol{X}_{k,t}, \bar{\boldsymbol{y}}_{k,t-1}) \\ \Theta(\bar{\boldsymbol{c}}_{k,t}, \boldsymbol{X}_{k,t}, \bar{\boldsymbol{y}}_{k,t-1}) \\ \boldsymbol{\theta}_{k,t-1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{e}_{k,t} \\ \boldsymbol{\epsilon}_{k,t} \\ \boldsymbol{v}_{k,t} \end{bmatrix} \quad (11)$$

$$d_{k,t} = \boldsymbol{w}_{k,t}^T \bar{\boldsymbol{y}}_{k,t} + \varepsilon_{k,t} \quad (12)$$

where $[\boldsymbol{e}_{k,t}^T, \boldsymbol{\epsilon}_{k,t}^T, \boldsymbol{v}_{k,t}^T]^T$ is zero mean Gaussian process with covariance $\boldsymbol{Q}_{k,t}$. Here, each node k is able to observe only $d_{k,t}$ to estimate $\bar{\boldsymbol{c}}_{k,t}$, $\bar{\boldsymbol{y}}_{k,t}$, and $\boldsymbol{\theta}_{k,t}$. Hence, we group $\bar{\boldsymbol{c}}_{k,t}$, $\bar{\boldsymbol{y}}_{k,t}$, and $\boldsymbol{\theta}_{k,t}$ together into a vector as the hidden states to be estimated.

1) Online Training With the EKF Algorithm: In this section, we derive the online training method based on the EKF algorithm when we do not allow communication between the neighbors. Since the system in (11) and (12) is already in a nonlinear state-space form, we can directly apply the EKF algorithm [13] as follows.

Time Update:

$$\begin{aligned} \bar{\boldsymbol{c}}_{k,t|t-1} &= \Omega(\bar{\boldsymbol{c}}_{k,t-1|t-1}, \boldsymbol{X}_{k,t}, \bar{\boldsymbol{y}}_{k,t-1|t-1}) && (13)\\ \bar{\boldsymbol{y}}_{k,t|t-1} &= \Theta(\bar{\boldsymbol{c}}_{t|t-1}, \boldsymbol{X}_{k,t}, \bar{\boldsymbol{y}}_{k,t-1|t-1}) && (14)\\ \boldsymbol{\theta}_{k,t|t-1} &= \boldsymbol{\theta}_{k,t-1|t-1} && (15)\\ \boldsymbol{\Sigma}_{k,t|t-1} &= \boldsymbol{F}_{k,t-1}\boldsymbol{\Sigma}_{k,t-1|t-1}\boldsymbol{F}_{k,t-1}^T + \boldsymbol{Q}_{k,t-1}. && (16)\end{aligned}$$

Measurement Update:

$$R = \boldsymbol{H}_{k,t}^T \boldsymbol{\Sigma}_{k,t|t-1} \boldsymbol{H}_{k,t} + R_{k,t}$$

$$\begin{bmatrix} \bar{\boldsymbol{c}}_{k,t|t} \\ \bar{\boldsymbol{y}}_{k,t|t} \\ \boldsymbol{\theta}_{k,t|t} \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{c}}_{k,t|t-1} \\ \bar{\boldsymbol{y}}_{k,t|t-1} \\ \boldsymbol{\theta}_{k,t|t-1} \end{bmatrix} + \boldsymbol{\Sigma}_{k,t|t-1}\boldsymbol{H}_{k,t} R^{-1}(d_{k,t} - \hat{d}_{k,t})$$

$$\boldsymbol{\Sigma}_{k,t|t} = \boldsymbol{\Sigma}_{k,t|t-1} - \boldsymbol{\Sigma}_{k,t|t-1}\boldsymbol{H}_{k,t}R^{-1}\boldsymbol{H}_{k,t}^T\boldsymbol{\Sigma}_{k,t|t-1}$$

where $\boldsymbol{\Sigma} \in \mathbb{R}^{(2n+n_\theta)\times(2n+n_\theta)}$ is the error covariance matrix, $\boldsymbol{Q}_{k,t} \in \mathbb{R}^{(2n+n_\theta)\times(2n+n_\theta)}$ is the state noise covariance, and $R_{k,t} \in \mathbb{R}$ is the measurement noise variance. In addition, we assume that $R_{k,t}$ and $\boldsymbol{Q}_{k,t}$ are known terms. We compute $\boldsymbol{H}_{k,t}$ and $\boldsymbol{F}_{k,t}$ as follows:

$$\boldsymbol{H}_{k,t}^T = \left[\frac{\partial \hat{d}_{k,t}}{\partial \bar{\boldsymbol{c}}} \quad \frac{\partial \hat{d}_{k,t}}{\partial \bar{\boldsymbol{y}}} \quad \frac{\partial \hat{d}_{k,t}}{\partial \boldsymbol{\theta}}\right]\Bigg|_{\substack{\bar{\boldsymbol{c}}=\bar{\boldsymbol{c}}_{k,t|t-1}\\ \bar{\boldsymbol{y}}=\bar{\boldsymbol{y}}_{k,t|t-1}\\ \boldsymbol{\theta}=\boldsymbol{\theta}_{k,t|t-1}}} \quad (17)$$

and

$$\boldsymbol{F}_{k,t} = \begin{bmatrix} \frac{\partial\Omega(\bar{\boldsymbol{c}},\boldsymbol{X}_{k,t},\bar{\boldsymbol{y}})}{\partial\bar{\boldsymbol{c}}} & \frac{\partial\Omega(\bar{\boldsymbol{c}},\boldsymbol{X}_{k,t},\bar{\boldsymbol{y}})}{\partial\bar{\boldsymbol{y}}} & \frac{\partial\Omega(\bar{\boldsymbol{c}},\boldsymbol{X}_{k,t},\bar{\boldsymbol{y}})}{\partial\boldsymbol{\theta}} \\ \frac{\partial\Theta(\bar{\boldsymbol{c}},\boldsymbol{X}_{k,t},\bar{\boldsymbol{y}})}{\partial\bar{\boldsymbol{c}}} & \frac{\partial\Theta(\bar{\boldsymbol{c}},\boldsymbol{X}_{k,t},\bar{\boldsymbol{y}})}{\partial\bar{\boldsymbol{y}}} & \frac{\partial\Theta(\bar{\boldsymbol{c}},\boldsymbol{X}_{k,t},\bar{\boldsymbol{y}})}{\partial\boldsymbol{\theta}} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}\Bigg|_{\substack{\bar{\boldsymbol{c}}=\bar{\boldsymbol{c}}_{k,t|t}\\ \bar{\boldsymbol{y}}=\bar{\boldsymbol{y}}_{k,t|t}\\ \boldsymbol{\theta}=\boldsymbol{\theta}_{k,t|t}}} \quad (18)$$

where $\boldsymbol{F}_{k,t} \in \mathbb{R}^{(2n+n_\theta)\times(2n+n_\theta)}$ and $\boldsymbol{H}_{k,t} \in \mathbb{R}^{(2n+n_\theta)}$.

2) Online Training With the DEKF Algorithm: In this section, we introduce our online training method based on the DEKF algorithm for the network described by (11) and (12). In our network of K nodes, we denote the number of neighbors for node k as η_k, i.e., also called as the degree of node k [12]. With this structure, the time update equations in (13)–(16) still hold for each node k. However, since we have information exchange between the neighbors, the measurement update equations of each node k adopt the iterative scheme [12] as the following.

For the node k at time t

$$\begin{aligned} \boldsymbol{\phi}_{k,t} &\longleftarrow \left[\bar{\boldsymbol{c}}_{k,t|t-1}^T \quad \bar{\boldsymbol{y}}_{k,t|t-1}^T \quad \boldsymbol{\theta}_{k,t|t-1}^T\right]^T \\ \boldsymbol{\Phi}_{k,t} &\longleftarrow \boldsymbol{\Sigma}_{k,t|t-1}. \end{aligned}$$

For each $l \in \mathcal{N}_k$ repeat:

$$\begin{aligned} R &\longleftarrow \boldsymbol{H}_{l,t}^T\boldsymbol{\Phi}_{k,t}\boldsymbol{H}_{l,t} + R_{l,t} \\ \boldsymbol{\phi}_{k,t} &\longleftarrow \boldsymbol{\phi}_{k,t} + \boldsymbol{\Phi}_{k,t}\boldsymbol{H}_{l,t}R^{-1}\left(d_{l,t} - \boldsymbol{w}_{k,t|t-1}^T\bar{\boldsymbol{y}}_{k,t|t-1}\right) \\ \boldsymbol{\Phi}_{k,t} &\longleftarrow \boldsymbol{\Phi}_{k,t} - \boldsymbol{\Phi}_{k,t}\boldsymbol{H}_{l,t}R^{-1}\boldsymbol{H}_{l,t}^T\boldsymbol{\Phi}_{k,t}. \end{aligned}$$

Now, we update the state and covariance matrix estimate as

$$\begin{aligned} \boldsymbol{\Sigma}_{k,t|t} &= \boldsymbol{\Phi}_{k,t} \\ \left[\bar{\boldsymbol{c}}_{k,t|t}^T \ \bar{\boldsymbol{y}}_{k,t|t}^T \ \boldsymbol{\theta}_{k,t|t}^T\right]^T &= \sum_{l\in\mathcal{N}_k} c(k,l)\boldsymbol{\phi}_{l,t} \end{aligned}$$

where $c(k,l)$ is the weight between nodes k and l and we compute these weights using the Metropolis rule as follows:

$$c(k,l) = \begin{cases} 1/\max(\eta_k, \eta_l) & \text{if } l \in \mathcal{N}_k/k \\ 1 - \sum_{l\in\mathcal{N}_k/k} c(k,l) & \text{if } k = l \\ 0 & \text{if } l \notin \mathcal{N}_k. \end{cases} \quad (19)$$

With these steps, we can update all the nodes in our network as illustrated in Algorithm 1.

According to the procedure in Algorithm 1, the computational complexity of our training method results in $\mathcal{O}(\eta_k(n^8 + n^4\ p^4))$ computations at each node k due to matrix and vector multiplications on lines 8 and 19 as shown in Table I.

Algorithm 1 Training Based on the DEKF Algorithm

```
1: According to (17), compute $\boldsymbol{H}_{k,t}$, $\forall k \in \{1, 2, \ldots, K\}$
2: for $k = 1 : K$ do
3:     $\boldsymbol{\phi}_{k,t} \longleftarrow [\bar{\boldsymbol{c}}_{k,t|t-1}^T \ \bar{\boldsymbol{y}}_{k,t|t-1}^T \ \boldsymbol{\theta}_{k,t|t-1}^T]^T$
4:     $\boldsymbol{\Phi}_{k,t} \longleftarrow \boldsymbol{\Sigma}_{k,t|t-1}$
5:     for $l \in \mathcal{N}_k$ do
6:         $R \longleftarrow \boldsymbol{H}_{l,t}^T \boldsymbol{\Phi}_{k,t} \boldsymbol{H}_{l,t} + R_{l,t}$
7:         $\boldsymbol{\phi}_{k,t} \longleftarrow \boldsymbol{\phi}_{k,t} + \boldsymbol{\Phi}_{k,t} \boldsymbol{H}_{l,t} R^{-1}(d_{l,t} - \boldsymbol{w}_{k,t|t-1}^T \bar{\boldsymbol{y}}_{k,t|t-1})$
8:         $\boldsymbol{\Phi}_{k,t} \longleftarrow \boldsymbol{\Phi}_{k,t} - \boldsymbol{\Phi}_{k,t} \boldsymbol{H}_{l,t} R^{-1} \boldsymbol{H}_{l,t}^T \boldsymbol{\Phi}_{k,t}$
9:     end for
10: end for
11: for $k = 1 : K$ do
12:     Using (19), calculate $c(k, l)$, $\forall l \in \mathcal{N}_k$
13:     $[\bar{\boldsymbol{c}}_{k,t|t}^T \ \bar{\boldsymbol{y}}_{k,t|t}^T \ \boldsymbol{\theta}_{k,t|t}^T]^T \longleftarrow \sum_{l \in \mathcal{N}_k} c(k, l) \boldsymbol{\phi}_{l,t}$
14:     $\boldsymbol{\Sigma}_{k,t|t} \longleftarrow \boldsymbol{\Phi}_{k,t}$
15:     According to (18), compute $\boldsymbol{F}_{k,t}$
16:     $\bar{\boldsymbol{c}}_{k,t+1|t} \longleftarrow \Omega(\bar{\boldsymbol{c}}_{k,t|t}, \boldsymbol{X}_{k,t}, \bar{\boldsymbol{y}}_{k,t|t})$
17:     $\bar{\boldsymbol{y}}_{k,t+1|t} \longleftarrow \Theta(\bar{\boldsymbol{c}}_{k,t+1|t}, \boldsymbol{X}_{k,t}, \bar{\boldsymbol{y}}_{k,t|t})$
18:     $\boldsymbol{\theta}_{k,t+1|t} \longleftarrow \boldsymbol{\theta}_{k,t|t}$
19:     $\boldsymbol{\Sigma}_{k,t+1|t} \longleftarrow \boldsymbol{F}_{k,t} \boldsymbol{\Sigma}_{k,t|t} \boldsymbol{F}_{k,t}^T + \boldsymbol{Q}_{k,t}$
20: end for
```

TABLE I

COMPARISON OF THE COMPUTATIONAL COMPLEXITIES OF THE INTRODUCED TRAINING ALGORITHMS FOR EACH NODE k. IN THIS TABLE, WE ALSO CALCULATE THE COMPUTATIONAL COMPLEXITY OF THE SGD-BASED ALGORITHM BY DERIVING EXACT GRADIENT EQUATIONS; HOWEVER, WE OMIT THESE CALCULATIONS DUE TO PAGE LIMIT

Algorithm	Computational Complexity
SGD	$\mathcal{O}(n^4 + n^2p^2)$
DEKF	$\mathcal{O}(\eta_k(n^8 + n^4p^4))$
DPF	$\mathcal{O}(N(k)(n^2 + np))$

B. Online Training Using the DPF Algorithm

In this section, we first derive our training method based on the PF algorithm when we do not allow communication between the nodes. We then introduce our online training method based on the DPF algorithm when the nodes share information with their neighbors.

The PF algorithm only requires the independence of the noise samples in (11) and (12). Thus, we modify our system in (11) and (12) for node k as follows:

$$\boldsymbol{a}_{k,t} = \varphi(\boldsymbol{a}_{k,t-1}, \boldsymbol{X}_{k,t}) + \boldsymbol{\gamma}_{k,t} \tag{20}$$

$$d_{k,t} = \boldsymbol{w}_{k,t}^T \bar{\boldsymbol{y}}_{k,t} + \varepsilon_{k,t} \tag{21}$$

where $\boldsymbol{\gamma}_{k,t}$ and $\varepsilon_{k,t}$ are independent state and measurement noise samples, respectively, $\varphi(\cdot, \cdot)$ is the nonlinear mapping in (11), and $\boldsymbol{a}_{k,t} \triangleq [\bar{\boldsymbol{c}}_{k,t}^T \ \bar{\boldsymbol{y}}_{k,t}^T \ \boldsymbol{\theta}_{k,t}^T]^T$.

1) Online Training with the PF Algorithm: For the system in (20) and (21), our aim is to obtain $\mathbf{E}[\boldsymbol{a}_{k,t}|d_{k,1:t}]$, i.e., the optimal estimate for the hidden state in the MSE sense. To achieve this, we first obtain posterior distribution of the states, i.e., $p(\boldsymbol{a}_{k,t}|d_{k,1:t})$. Based on the posterior density function, we then calculate the conditional mean estimate. In order to obtain the posterior distribution, we apply the PF algorithm [19].

In this algorithm, we have the samples and the corresponding weights of $p(\boldsymbol{a}_{k,t}|d_{k,1:t})$, i.e., denoted as $\{\boldsymbol{a}_{k,t}^i, \omega_{k,t}^i\}_{i=1}^N$. Based on the samples, we obtain the posterior distribution as follows:

$$p(\boldsymbol{a}_{k,t}|d_{k,1:t}) \approx \sum_{i=1}^N \omega_{k,t}^i \delta(\boldsymbol{a}_{k,t} - \boldsymbol{a}_{k,t}^i). \tag{22}$$

Sampling from the desired distribution $p(\boldsymbol{a}_{k,t}|d_{k,1:t})$ is intractable in general so that we obtain the samples from $q(\boldsymbol{a}_{k,t}|d_{k,1:t})$, which is called as an importance function [19]. To calculate the weights in (22), we use the following formula:

$$\omega_{k,t}^i \propto \frac{p(\boldsymbol{a}_{k,t}^i|d_{k,1:t})}{q(\boldsymbol{a}_{k,t}^i|d_{k,1:t})}, \quad \text{where} \quad \sum_{i=1}^N \omega_{k,t}^i = 1. \tag{23}$$

We can factorize (23) such that we obtain the following recursive formula [19]:

$$\omega_{k,t}^i \propto \frac{p(d_{k,t}|\boldsymbol{a}_{k,t}^i) p(\boldsymbol{a}_{k,t}^i|\boldsymbol{a}_{k,t-1}^i)}{q(\boldsymbol{a}_{k,t}^i|\boldsymbol{a}_{k,t-1}^i, d_{k,t})} \omega_{k,t-1}^i. \tag{24}$$

In (24), we choose the importance function so that the variance of the weights is minimized. By this, we obtain particles that have nonnegligible weights and significantly contribute to (22) [19]. In this sense, since $p(\boldsymbol{a}_{k,t}^i|\boldsymbol{a}_{k,t-1}^i)$ provides a small variance for the weights [19], we choose it as our importance function. With this choice, we alter (24) as follows:

$$\omega_{k,t}^i \propto p(d_{k,t}|\boldsymbol{a}_{k,t}^i) \omega_{k,t-1}^i. \tag{25}$$

By (22) and (25), we obtain the state estimate as follows:

$$\begin{aligned}\mathbf{E}[\boldsymbol{a}_{k,t}|d_{k,1:t}] &= \int \boldsymbol{a}_{k,t} p(\boldsymbol{a}_{k,t}|d_{k,1:t}) d\boldsymbol{a}_{k,t} \\ &\approx \int \boldsymbol{a}_{k,t} \sum_{i=1}^N \omega_{k,t}^i \delta(\boldsymbol{a}_{k,t} - \boldsymbol{a}_{k,t}^i) d\boldsymbol{a}_{k,t} = \sum_{i=1}^N \omega_{k,t}^i \boldsymbol{a}_{k,t}^i.\end{aligned}$$

Although we choose the importance function to reduce the variance of the weights, the variance inevitably increases over time [19]. Hence, we apply the resampling algorithm introduced in [19] such that we eliminate the particles with small weights and prevent the variance from increasing.

2) Online Training With the DPF Algorithm: In this section, we introduce our online training method based on the DPF algorithm when the nodes share information with their neighbors. We employ the Markov Chain Distributed Particle Filter (MCDPF) algorithm [17] to train our distributed system. In the MCDPF algorithm, particles move around the network according to the network topology. In every step, each particle can randomly move to another node in the neighborhood of its current node. While randomly moving, the weight of each particle is updated using $p(d_{k,t}|\boldsymbol{a}_{k,t})$ at node k; hence, particles use the observations at different nodes.

Suppose we consider our network as a graph $G = (V, E)$, where the vertices V represent the nodes in our network and the edges E represent the connections between the nodes. In addition to this, we denote the number of visits to each node k in s steps by each particle i as $M^i(k, s)$. Here, each particle moves to one of its neighboring nodes with a certain probability, where the movement probabilities of each node to the other nodes are represented by the adjacency matrix, i.e., denoted as $\mathcal{A}$. In this framework, at each visit to each node k, each particle multiplies its weight with $p(d_{k,t}|\boldsymbol{a}_{k,t})^{\frac{2|E(G)|}{s\eta_k}}$ in a run of s steps [17], where $|E(G)|$ is the number of edges in G and η_k is the degree of node k. From (25), we have the following update for each particle i at node k after s steps:

$$w_{k,t}^i = w_{k,t-1}^i \prod_{j=1}^K p(d_{j,t}|\boldsymbol{a}_{k,t}^i)^{\frac{2|E(G)|}{s\eta_j} M^i(j,s)}. \tag{26}$$

We then calculate the posterior distribution at node k as

$$p(\boldsymbol{a}_{k,t}|O_{k,t}) \approx \sum_{i=1}^N w_{k,t}^i \delta(\boldsymbol{a}_{k,t} - \boldsymbol{a}_{k,t}^i) \tag{27}$$

Algorithm 2 Training Based on the DPF Algorithm

```
1: Sample $\{a^i_{j,t}\}_{i=1}^{N(j)}$ from $p(a_t|\{a^i_{j,t-1}\}_{i=1}^{N(j)})$, $\forall j$
2: Set $\{w^i_{j,t}\}_{i=1}^{N(j)} = 1$, $\forall j$
3: for $s$ steps do
4:     Move the particles according to $\mathcal{A}$
5:     for $j = 1 : K$ do
6:         $\{a^i_{j,t}\}_{i=1}^{N(j)} \leftarrow \bigcup_{l\in\mathcal{N}_j}\{a^i_{l,t}\}_{i\in\mathcal{I}_{l\to j}}$
7:         $\{w^i_{j,t}\}_{i=1}^{N(j)} \leftarrow \bigcup_{l\in\mathcal{N}_j}\{w^i_{l,t}\}_{i\in\mathcal{I}_{l\to j}}$
8:         $\{w^i_{j,t}\}_{i=1}^{N(j)} \leftarrow \{w^i_{j,t}\}_{i=1}^{N(j)} p(d_{j,t}|\{a^i_{j,t}\}_{i=1}^{N(j)})^{\frac{2|E(G)|}{s\eta_j}}$
9:     end for
10: end for
11: for j=1:K do
12:     Resample $\{a^i_{j,t}, w^i_{j,t}\}_{i=1}^{N(j)}$
13:     Compute the estimate for node $j$ using (28)
14: end for
```

where $O_{k,t}$ represents the observations seen by the particles at node k until t, and $\omega^i_{k,t}$ is obtained from (26). After we obtain (27), we calculate our estimate for $\boldsymbol{a}_{k,t}$ as follows:

$$\begin{aligned}\mathbf{E}[\boldsymbol{a}_{k,t}|O_{k,t}] &= \int \boldsymbol{a}_{k,t}\, p(\boldsymbol{a}_{k,t}|O_{k,t})d\boldsymbol{a}_{k,t}\\ &\approx \int \boldsymbol{a}_{k,t}\sum_{i=1}^{N}\omega^i_{k,t}\delta\big(\boldsymbol{a}_{k,t}-\boldsymbol{a}^i_{k,t}\big)d\boldsymbol{a}_{k,t}\\ &= \sum_{i=1}^{N}\omega^i_{k,t}\boldsymbol{a}^i_{k,t}. \end{aligned} \tag{28}$$

We can obtain the estimate for each node using the same procedure as illustrated in Algorithm 2. In Algorithm 2, $N(j)$ represents the number of particles at node j and $\mathcal{I}_{i\to j}$ represents the indices of the particles that move from node i to node j. Thus, we obtain a distributed training algorithm that guarantees convergence to the optimal centralized parameter estimation as illustrated in Theorem 1.

Theorem 1: For each node k, let $\boldsymbol{a}_{k,t}$ be the bounded state vector with a measurement density function that satisfies the following inequality:

$$0 < p_0 \le p(d_{k,t}|\boldsymbol{a}_{k,t}) \le ||p||_\infty < \infty \tag{29}$$

where p_0 is a constant and

$$||p||_\infty = \sup_{d_{k,t}} p(d_{k,t}|\boldsymbol{a}_{k,t}).$$

Then, we have the following convergence results in the MSE sense:

$$\sum_{i=1}^{N}\omega^i_{k,t}\boldsymbol{a}^i_{k,t} \to \mathbf{E}[\boldsymbol{a}_{k,t}|\{d_{j,1:t}\}_{j=1}^{K}] \text{ as } N\to\infty \text{ and } k\to\infty.$$

Proof of Theorem 1: Using (29), from [17], we obtain

$$\mathbf{E}\left[\left(\mathbf{E}\big[\pi(\boldsymbol{a}_t)|\{d_{j,1:t}\}_{j=1}^{K}\big] - \sum_{i=1}^{N}\omega^i_{k,t}\pi\left(\boldsymbol{a}^i_{k,t}\right)\right)^2\right] \le ||\pi||_\infty^2\left(C_t\sqrt{U(s,\upsilon)}+\sqrt{\frac{\varsigma_t}{N}}\right)^2 \tag{30}$$

where π is a bounded function, υ is the second largest eigenvalue modulus of $\mathcal{A}$, ς_t and C_t are time-dependent constants, and $U(s,\upsilon)$ is a function of s as described in [17] such that $U(s,\upsilon)$ goes to zero as s goes to infinity. Since the state vector $\boldsymbol{a}_{k,t}$ is bounded, we can choose $\pi(\boldsymbol{a}_{k,t}) = \boldsymbol{a}_{k,t}$. With this choice, evaluating (30) as N and s go to infinity yields the results. This concludes our proof. □

According to the update procedure illustrated in Algorithm 2, the computational complexity of our training method results in $\mathcal{O}(N(k)(n^2+np))$ computations at each node k due to matrix vector multiplications in (20) and (21) as shown in Table I.

IV. Simulations

We evaluate the performance of the introduced algorithms on different benchmark real data sets. We first consider the prediction performance on Hong Kong exchange rate data set [20]. We then evaluate the regression performance on emotion-labeled sentence data set [21]. For these experiments, to observe the effects of communication among nodes, we also consider the EKF- and PF-based algorithms without communication over a network of multiple nodes, where each node trains LSTM based on only its observations. Throughout this section, we denote the EKF- and PF-based algorithms without communication over a network of multiple nodes as "EKF" and "PF," respectively. Moreover, we denote the EKF- and PF-based algorithms with communication over a network of multiple nodes as "DEKF" and "DPF," respectively. We also consider the SGD-based algorithm without communication over a network of multiple nodes as a benchmark algorithm and denote it by "SGD."

We first consider the Hong Kong exchange rate data set [20]. For this data set, we have the amount of Hong Kong dollars that can buy one United States dollar on certain days. Our aim is to estimate future exchange rate by using the values in the previous two days. In online applications, one can demand a small steady state error or fast convergence rate based on the requirements of application [22]. In this experiment, we evaluate the convergence rates of the algorithms. For this purpose, we select the parameters such that the algorithms converge to the same steady-state error level. In this setup, we choose the parameters for each node k as follows. Since $\boldsymbol{X}_{k,t} \in \mathbb{R}^2$ is our input, we set the output dimension as $n = 2$. In addition to this, we consider a network of four nodes.

For the PF-based algorithms, we choose $N(k) = 80$ as the number of particles. In addition, we select $\boldsymbol{\gamma}_{k,t}$ and $\varepsilon_{k,t}$ as zero-mean Gaussian random variables with $\text{Cov}[\boldsymbol{\gamma}_{k,t}] = 0.0004\boldsymbol{I}$ and $\text{Var}[\varepsilon_{k,t}] = 0.01$, respectively. For the DPF-based algorithm, we choose $s = 3$ and $\mathcal{A} = [0\ (1/2)\ 0\ (1/2);\ (1/2)\ 0\ (1/2)\ 0;\ 0\ (1/2)\ 0\ (1/2);\ (1/2)\ 0\ (1/2)\ 0]$.

For the EKF-based algorithms, we select $\boldsymbol{\Sigma}_{k,0|0} = 0.0004\boldsymbol{I}$, $\boldsymbol{Q}_{k,t} = 0.0004\boldsymbol{I}$, and $R_{k,t} = 0.01$. Moreover, according to (19), the weights between nodes are calculated as $1/3$.

For the SGD-based algorithm, we set the learning rate as $\mu = 0.1$.

In Fig. 2(a), we illustrate the prediction performance of the algorithms. Due to the highly nonlinear structure of our model, the EKF and DEKF-based algorithms have slower convergence compared with the other algorithms. Moreover, due to only exploiting the first-order gradient information, the SGD-based algorithm has also slower convergence compared with the PF-based algorithms. Unlike the SGD- and EKF-based methods, the PF-based algorithms perform parameter estimation through a high-performance gradient-free density estimation technique; hence, they converge much faster to the final MSE level, i.e., defined as $(1/T)\sum_{t=1}^{T}(d_t-\hat{d}_t)^2$ for a sequence of length T. Among the PF-based methods, due to its distributed structure, the DPF-based algorithm has the fastest convergence rate.

In order to demonstrate the effects of the number of particles N and the number of Markov steps s, we perform another experiment using the Hong Kong exchange rate data set. In this experiment, we use the same setting with the previous case except $\text{Cov}[\boldsymbol{\gamma}_{k,t}] = 0.0001\boldsymbol{I}$, $\boldsymbol{\Sigma}_{k,0|0} = 0.0001\boldsymbol{I}$, and $\boldsymbol{Q}_{k,t} = 0.0001\boldsymbol{I}$. In Fig. 2(b), we observe that as s and N increase, the DPF-based algorithm obtains a faster convergence rate and a lower final MSE value. However, as s and N increase, the marginal performance improvement becomes

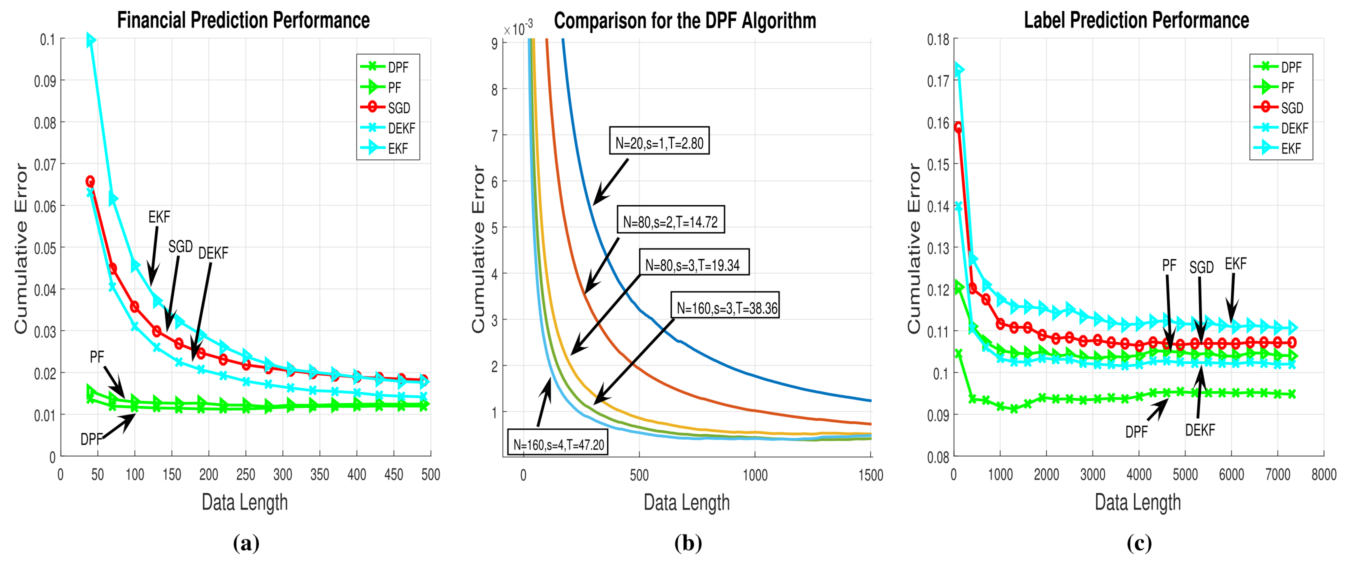

Fig. 2. Error performances (a) over the Hong Kong exchange rate data set, (b) for different N and s combinations of the DPF-based algorithm, and (c) over the sentence data set. In (b), we also provide computation times of the combinations (in seconds), i.e., denoted as T, where a computer with i5-6400 processor, 2.7-GHz CPU, and 16-GB RAM is used.

smaller with respect to the previous s and N values. Furthermore, the computation time of the algorithm increases with increasing s and N values. Thus, after a certain selection, a further increase does not worth the additional computational load. Therefore, we use $N(k) = 80$ and $s = 3$ in our previous simulation.

Other than the Hong Kong exchange rate data set, we consider the emotion-labeled sentence data set [21]. In this data set, we have the vector representation of each word in an emotion-labeled sentence. In this experiment, we evaluate the steady-state error performance of the algorithms. Thus, we choose the parameters such that the convergence rate of the algorithms is similar. To provide this setup, we select the parameters for each node k as follows. Since the number of words varies from sentence to sentence in this case, we have a variable length input regressor, i.e., defined as $\boldsymbol{X}_{k,t} \in \mathbb{R}^{2\times m_t}$, where m_t represents the number of words in a sentence. For the other parameters, we use the same setting with the Hong Kong exchange rate data set except $N(k) = 50$, $\text{Cov}[\boldsymbol{\gamma}_{k,t}] = (0.025)^2\boldsymbol{I}$, $\boldsymbol{\Sigma}_{k,0|0} = (0.025)^2\boldsymbol{I}$, $\boldsymbol{Q}_{k,t} = (0.025)^2\boldsymbol{I}$, and $\mu = 0.055$. In Fig. 2(c), we illustrate the label prediction performance of the algorithms. Again due to the highly nonlinear structure of our model, the EKF-based algorithm has the highest steady-state error value. In addition, the SGD-based algorithm also has a high final MSE value compared with the other algorithms. Furthermore, the DEKF-based algorithm achieves a lower final MSE value than the PF-based method thanks to its distributed structure. However, since the DPF-based method utilizes a powerful gradient-free density estimation method while effectively sharing information between the neighboring nodes, it achieves a much smaller steady-state error value.

We also perform another experiment that includes a larger data set, where we include two additional algorithms, namely the HF [14] and QN [15] algorithms to illustrate their performances. For this purpose, we use the temperature data set [23], and in this data set, we have temperature data that was collected from 2006 to 2013 by a weather station in Amherst, MA, USA. Our aim is to predict tomorrow's temperature value by examining the temperature of the previous four days. Since we aim to compare the convergence rates of the algorithms, we select the parameters such that all the algorithms converge to the same steady-state error level. Here, we use the same setting with the Hong Kong exchange rate data set except $N(k) = 80$, $\text{Cov}[\boldsymbol{\gamma}_{k,t}] = 0.0004\boldsymbol{I}$, $\boldsymbol{\Sigma}_{k,0|0} = 0.0004\boldsymbol{I}$, $\boldsymbol{Q}_{k,t} = 0.0004\boldsymbol{I}$, and $\mu = 0.001$. Furthermore, we set the learning rate of the QN algorithm

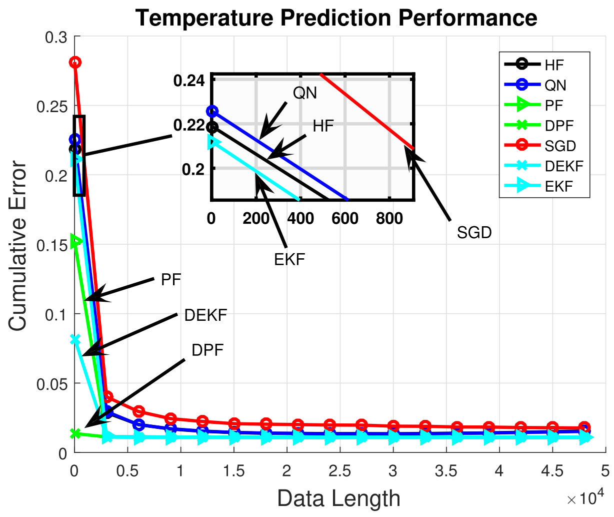

Fig. 3. Sequential prediction performance of the algorithms for the temperature data set.

as 0.0003, and for the other parameters of the HF and QN algorithms, we follow [14], [15]. Here, we denote the HF and QN algorithms as "HF" and "QN," respectively. In Fig. 3, we demonstrate the temperature prediction performances of the algorithms. Since the SGD-based algorithm only exploits the first-order gradient information, it has the slowest convergence rate compared with the others. The QN and HF algorithms perform similarly and achieve slightly slower convergence rate compared with the EKF-based algorithm. The PF-based algorithm achieves the fastest rate among the algorithms that lack communication thanks to its powerful gradient-free density estimation technique. However, overall the distributed algorithms achieve much faster rates due to their communication capability, among which the DPF-based algorithm converges to the final MSE level in a much faster manner.

In addition to the temperature data set, we also perform experiments on four different large data sets. For these data sets, we select the parameters such that all the algorithms have similar convergence rates. We first consider the CMU ARCTIC data set [24], where an English male speaker reads 1132 different utterances and it contains 4×10^5 samples. Our aim is to predict the next sample

TABLE II
TIME-ACCUMULATED ERRORS OF THE ALGORITHMS FOR THE SPEECH, ELEVATORS, PUMADYN, AND BANK DATA SETS

Algorithms / Datasets	PF	DPF	EKF	DEKF	SGD
Speech	0.01121	**0.01087**	0.01648	0.01621	0.02063
Elevators	5.5489×10^{-4}	$\mathbf{5.1489 \times 10^{-4}}$	6.5240×10^{-4}	5.9140×10^{-4}	6.6899×10^{-4}
Pumadyn	0.0011	**0.0008**	0.0012	0.0010	0.0015
Bank	0.0156	**0.0121**	0.01759	0.0158	0.0170

based on the previous four samples. For this experiments, we use the same setting with the temperature data set except $N(k) = 50$, $\text{Cov}[\boldsymbol{\gamma}_{k,t}] = 0.0001\boldsymbol{I}$, $\boldsymbol{\Sigma}_{k,0|0} = 0.0001\boldsymbol{I}$, $\boldsymbol{Q}_{k,t} = 0.0001\boldsymbol{I}$, and $\mu = 0.003$. We then perform another experiment on the bank data set [25], where we have feature vectors related to the queues in banks and we aim to estimate the fraction of the people that leaves the bank because of the full queues. This data set contains 8192 samples. For this experiment, we use the same setting with the speech data set except $n = 32$, $N(k) = 150$, $\text{Cov}[\boldsymbol{\gamma}_{k,t}] = 0.0016\boldsymbol{I}$, $\text{Var}[\varepsilon_{k,t}] = 0.25$, $\boldsymbol{\Sigma}_{k,0|0} = 0.0016\boldsymbol{I}$, $\boldsymbol{Q}_{k,t} = 0.0016\boldsymbol{I}$, $R_{k,t} = 0.25$, and $\mu = 0.07$. As the third data set, we use the elevator data set [26], where we have feature vectors related to an F16 aircraft and our aim is to predict a certain variable that explains the aircraft's actions. This data set contains 16000 samples and we use the same setting with the bank data set except $n = 18$, $N(k) = 100$, and $\mu = 0.7$. Finally, we perform an experiment on the pumadyn data set [26], where we have feature vectors related to the action of a robotic arm and we aim to predict its angular acceleration. Here, we have 8192 samples and we use the same setting with the bank data set except $N(k) = 170$ and $\mu = 0.4$. As shown in Table II, in all experiments, the DPF-based algorithm achieves much smaller time-accumulated error thanks to its distributed architecture and high-performance gradient-free density estimation technique.

V. CONCLUSION

We studied online training of the LSTM architecture in a distributed network of nodes for regression and introduced online distributed training algorithms for variable length data sequences. We first proposed a generic LSTM-based model for variable length data inputs. In order to train this model, we put the model equations in a nonlinear state-space form. Based on this form, we introduced DEKF-based and DPF-based online training algorithms. In this way, we obtain effective training algorithms for our LSTM-based model. Here, our DPF-based algorithm guarantees convergence to the optimal centralized parameter estimation in the MSE sense under certain conditions. We achieve this performance with communication and computational complexity in the order of the first-order methods [3]. Through simulations involving real life and financial data, we illustrate significant performance improvements with respect to the state-of-the-art methods [16], [18].

REFERENCES

[1] D. F. Specht, "A general regression neural network," *IEEE Trans. Neural Netw.*, vol. 2, no. 6, pp. 568–576, Nov. 1991.

[2] Y. Meng, Y. Jin, and J. Yin, "Modeling activity-dependent plasticity in BCM spiking neural networks with application to human behavior recognition," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 1952–1966, Dec. 2011.

[3] A. C. Tsoi, "Gradient based learning methods," in *Adaptive Processing of Sequences and Data Structures: International Summer School on Neural Networks*, C. Lee Giles and M. Gori, Eds. Berlin, Germany: Springer, Sep. 1998, pp. 27–62. [Online]. Available: https://doi.org/10.1007/BFb0053994, doi: 10.1007/BFb0053994.

[4] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[6] S. Vosoughi, P. Vijayaraghavan, and D. Roy, "Tweet2Vec: Learning tweet embeddings using character-level CNN-LSTM encoder-decoder," in *Proc. 39th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, Pisa, Italy, 2016, pp. 1041–1044. [Online]. Available: http://doi.acm.org/10.1145/2911451.2914762, doi: 10.1145/2911451.2914762.

[7] D. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Netw.*, vol. 16, no. 10, pp. 1429–1451, 2003.

[8] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[9] C. Yan *et al.*, "A highly parallel framework for hevc coding unit partitioning tree decision on many-core processors," *IEEE Signal Process. Lett.*, vol. 21, no. 5, pp. 573–576, May 2014.

[10] C. Yan *et al.*, "Efficient parallel framework for hevc motion estimation on many-core processors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 12, pp. 2077–2089, Dec. 2014.

[11] K. Yuan, Q. Ling, and W. Yin, "On the convergence of decentralized gradient descent," *SIAM J. Optim.*, vol. 26, no. 3, pp. 1835–1854, 2016.

[12] F. S. Cattivelli and A. H. Sayed, "Diffusion strategies for distributed Kalman filtering and smoothing," *IEEE Trans. Autom. Control*, vol. 55, no. 9, pp. 2069–2084, Sep. 2010.

[13] B. D. Anderson and J. B. Moore, *Optimal Filtering*. North Chelmsford, MA, USA: Courier Corporation, 2012.

[14] R. Kiros. (May 2013). "Training neural networks with stochastic Hessian-free optimization." [Online]. Available: http://arxiv.org/abs/1301.3641

[15] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A stochastic quasi-Newton method for large-scale optimization," *SIAM J. Optim.*, vol. 26, no. 2, pp. 1008–1031, 2016. [Online]. Available: https://doi.org/10.1137/140954362

[16] J. A. Pérez-Ortiz, F. A. Gers, D. Eck, and J. Schmidhuber, "Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets," *Neural Netw.*, vol. 16, no. 2, pp. 241–250, 2003.

[17] S. H. Lee and M. West, "Convergence of the Markov chain distributed particle filter (MCDPF)," *IEEE Trans. Signal Process.*, vol. 61, no. 4, pp. 801–812, Feb. 2013.

[18] A. W. Smith and D. Zipser, "Learning sequential structure with the real-time recurrent learning algorithm," *Int. J. Neural Syst.*, vol. 1, no. 02, pp. 125–131, 1989.

[19] P. M. Djuric *et al.*, "Particle filtering," *IEEE Signal Process. Mag.*, vol. 20, no. 5, pp. 19–38, Sep. 2003.

[20] E. W. Frees. *Regression Modeling With Actuarial and Financial Applications*. Accessed: May 25, 2017. [Online]. Available: http://instruction.bus.wisc.edu/jfrees/jfreesbooks/Regression%20Modeling/BookWebDec2010/data.html

[21] M. Lichman. (2013). "UCI machine learning repository." School Inf. Comput. Sci., Univ. California, Irvine, CA, USA. Tech. Rep. [Online]. Available: http://archive.ics.uci.edu/ml

[22] A. H. Sayed, *Fundamentals of Adaptive Filtering*. Hoboken, NJ, USA: Wiley, 2003.

[23] M. Liberatore. *UMass Trace Repository*. Accessed: May 25, 2017. [Online]. Available: http://traces.cs.umass.edu/index.php/Sensors/Sensors

[24] J. Kominek and A. W. Black. *CMU Arctic Database*. Accessed: May 25, 2017. [Online]. Available: http://www.festvox.org/cmuarctic/index.html

[25] L. Torgo. *Regression Data Sets*. Accessed: May 25, 2017. [Online]. Available: http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html

[26] J. Alcalá-Fdez *et al.*, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *J. Multiple-Valued Logic Soft Comput.*, vol. 17, nos. 2–3, pp. 255–287, 2011.