DISTRIBUTED LEARNING BASED ON RECURRENT NEURAL NETWORKS FOR BIG DATA APPLICATIONS

Tolga Ergen and Suleyman S. Kozat

Bilkent University
Department of Electrical and Electronics Engineering
Ankara 06800, Turkey

ABSTRACT

We study online training of Long Short Term Memory (LSTM) architectures in a network of nodes. For this framework, we first provide an LSTM based regression structure. To train this structure, we put the underlying LSTM equations in a nonlinear state space form at each node and then introduce a highly efficient and effective Distributed Particle Filtering (DPF) based online training algorithm. Here, our training method guarantees convergence to the optimal training performance in the Mean Square Error (MSE) sense. We achieve this performance with communication and computational complexity in the order of the first order gradient based methods. In our simulations, we demonstrate significant performance gains with respect to the conventional methods.

Index Terms— Distributed learning, particle filtering, big data, online learning, LSTM.

1. INTRODUCTION

Neural networks provide enhanced performance for a wide range of engineering applications [1, 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 [4]. Hence, simple RNNs are insufficient to capture time dependencies [4]. To circumvent this issue, a novel RNN architecture with control structures, i.e., the 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 this end, we consider online training of the parameters of an LSTM structure in a distributed network of nodes.

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 big data applications, storage issues may arise due to keeping all the data in one place [6]. Additionally, in certain frameworks, all data instances are not available beforehand since instances are received in a sequential manner, which precludes batch training [6]. Hence, we consider online training, where we sequentially receive the data for training. 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 so that distributed architectures are introduced [7]. In this basic distributed architectures, commonly named as centralized approach [7], 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 [7]. Additionally, centralized strategies have a risk of failure at the central node. To circumvent these issues, we distribute the processing to the nodes and allow communication only between neighbors, hence, we remove the need for a central node.

For training, one can employ one of the first order gradient based algorithms, e.g., the Stochastic Gradient Descent (SGD) algorithm, at each node due to their efficiency [3] and exchange estimates among neighboring nodes as in [8]. However, since these training methods only exploit the first order gradient information, they suffer from poor performance and convergence issues. On the other hand, the second order gradient based methods, e.g., the Extended Kalman Filtering (EKF) algorithm, require much higher computational complexity and communication load while providing superior performance [3, 9]. In this paper, 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 PF algorithm [10]. Thus, as the first time in the literature, we introduce an LSTM training method in an online setting for variable length data sequences. Our training method guarantees convergence to the optimal centralized training performance while requiring communication and computational complexity only in the order of the first order methods.

2. MODEL AND PROBLEM DESCRIPTION

Here, all column vectors (or matrices) are denoted by boldface lower (or uppercase) case letters. For a matrix $\boldsymbol{A}$ (or a vector $\boldsymbol{a}$), $\boldsymbol{A}^T$ ($\boldsymbol{a}^T$) is its ordinary transpose. The time index

This work is supported in part by Outstanding Researcher Programme Turkish Academy of Sciences and TUBITAK Contract No 115E917.

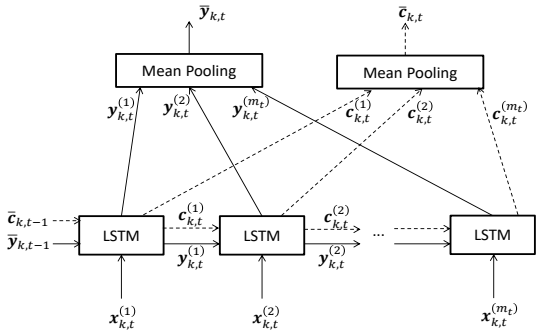

Fig. 1: Detailed schematic of each node k in our network.

is given as subscript, e.g., $\boldsymbol{a}_t$ is the vector at time t. $\boldsymbol{I}$ is the identity matrix, where the size is understood from the context.

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 the node k, i.e., $k \in \mathcal{N}_k$. At each node k, we sequentially receive the desired signal $\{d_{k,t}\}_{t\geq 1}$, $d_{k,t} \in \mathbb{R}$ and matrices, $\{\boldsymbol{X}_{k,t}\}_{t\geq 1}$, defined as $\boldsymbol{X}_{k,t} = [\boldsymbol{x}_{k,t}^{(1)}\, \boldsymbol{x}_{k,t}^{(2)} \dots \boldsymbol{x}_{k,t}^{(m_t)}]$, where $\boldsymbol{x}_{k,t}^{(l)} \in \mathbb{R}^p$, $\forall l \in \{1, 2, \dots, m_t\}$ 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 $\boldsymbol{X}_{k,t}$.

In this paper, each node k generates an estimate $\hat{d}_{k,t}$ for the desired value $d_{k,t}$ using the LSTM architecture [5]. 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]:

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

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

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

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

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

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 $\sigma(\cdot)$ are set to the hyperbolic tangent function and the sigmoid function respectively and apply vectors pointwise. 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 seen in Fig. 1, we generate the estimate for each node k as $\hat{d}_{k,t} = \boldsymbol{w}_{k,t}^T \bar{\boldsymbol{y}}_{k,t}$, 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.

3. ONLINE DISTRIBUTED TRAINING

In this section, we give the LSTM equations in a nonlinear state space form. Based on this form, 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.

Considering our model in Fig. 1 and the LSTM equations, we have the following state space form for each node k

$$\bar{\boldsymbol{c}}_{k,t} = \Omega(\bar{\boldsymbol{c}}_{k,t-1}, \boldsymbol{X}_{k,t}, \bar{\boldsymbol{y}}_{k,t-1}) \tag{6}$$

$$\bar{\boldsymbol{y}}_{k,t} = \Theta(\bar{\boldsymbol{c}}_{k,t}, \boldsymbol{X}_{k,t}, \bar{\boldsymbol{y}}_{k,t-1}) \tag{7}$$

$$\boldsymbol{\theta}_{k,t} = \boldsymbol{\theta}_{k,t-1} \tag{8}$$

$$d_{k,t} = \boldsymbol{w}_{k,t}^T \bar{\boldsymbol{y}}_{k,t} + \varepsilon_{k,t}, \tag{9}$$

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 (8) as our state. Furthermore, $\varepsilon_{k,t}$ represents the error in observations.

Based on the assumptions of the PF algorithm [11], we have the following compact form for the node k

$$\boldsymbol{a}_{k,t} = \varphi(\boldsymbol{a}_{k,t-1}, \boldsymbol{X}_{k,t}) + \boldsymbol{\gamma}_{k,t} \tag{10}$$

$$d_{k,t} = \boldsymbol{w}_{k,t}^T \bar{\boldsymbol{y}}_{k,t} + \varepsilon_{k,t}, \tag{11}$$

where $\boldsymbol{\gamma}_{k,t}$ and $\varepsilon_{k,t}$ are independent state and measurement noise samples, respectively, $\varphi(\cdot,\cdot)$ is the nonlinear mappings in (6), (7) and (8) and $\boldsymbol{a}_{k,t} \triangleq [\bar{\boldsymbol{c}}_{k,t}^T\, \bar{\boldsymbol{y}}_{k,t}^T\, \boldsymbol{\theta}_{k,t}^T]^T$.

3.1. Online Training with the PF Algorithm

Here, we aim 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 [11].

In this algorithm, we have the samples and the 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

$$p(\boldsymbol{a}_{k,t}|d_{k,1:t}) \approx \sum_{i=1}^{N} \omega_{k,t}^i \delta(\boldsymbol{a}_{k,t} - \boldsymbol{a}_{k,t}^i). \tag{12}$$

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 importance function [11]. To calculate the weights in (12), we use the following formula

$$w_{k,t}^i \propto \frac{p(\boldsymbol{a}_{k,t}^i|d_{k,1:t})}{q(\boldsymbol{a}_{k,t}^i|d_{k,1:t})}, \text{ where } \sum_{i=1}^{N} \omega_{k,t}^i = 1. \tag{13}$$

We can factorize (13) such that we obtain the following recursive formula [11]

$$\omega_{k,t}^i \propto \frac{p(d_{k,t}|\boldsymbol{a}_{k,t}^i)p(\boldsymbol{a}_{k,t}^i|\boldsymbol{a}_{k,t-1}^i)}{q(\boldsymbol{a}_{k,t}^i|\boldsymbol{a}_{k,t-1}^i, d_{k,t})}\omega_{k,t-1}^i. \quad (14)$$

In (14), 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 (12) [11]. In this sense, since $p(\boldsymbol{a}_{k,t}^i|\boldsymbol{a}_{k,t-1}^i)$ provides a small variance for the weights [11], we choose it as our importance function. With this choice, we alter (14) as follows

$$\omega_{k,t}^i \propto p(d_{k,t}|\boldsymbol{a}_{k,t}^i)\omega_{k,t-1}^i. \quad (15)$$

By (12) and (15), we obtain the state estimate as follows

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

Although we choose the importance function to reduce the variance of the weights, the variance inevitably increases over time [11]. Hence, we apply the resampling algorithm introduced in [11] such that we eliminate the particles with small weights and prevent the variance from increasing.

3.2. Online Training with the DPF Algorithm

In this subsection, 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 [10] to train our distributed system. In the MCDPF algorithm, particles move around the network according to the network topology, where each particle can randomly move to the neighboring nodes and update its weight while moving.

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 [10], where $|E(G)|$ is the number of edges in G and η_k is the degree of the node k. From (15), we have the following update for each particle i at the node k after s steps

$$w_{k,t}^i = w_{k,t-1}^i \prod_{j=1}^K p(d_{j,t}|\boldsymbol{a}_{k,t}^i)^{\frac{2|E(G)|}{s\eta_j}M^i(j,s)}. \quad (16)$$

We then calculate the posterior distribution at the node k as

$$p(\boldsymbol{a}_{k,t}|O_{k,t}) \approx \sum_{i=1}^N w_{k,t}^i \delta(\boldsymbol{a}_{k,t} - \boldsymbol{a}_{k,t}^i), \quad (17)$$

where $O_{k,t}$ represents the observations seen by the particles at the node k until t and $w_{k,t}^i$ is obtained from (16). After we obtain (17), we calculate our estimate for $\boldsymbol{a}_{k,t}$ as follows

$$\mathbf{E}[\boldsymbol{a}_{k,t}|O_{k,t}] = \int \boldsymbol{a}_{k,t}p(\boldsymbol{a}_{k,t}|O_{k,t})d\boldsymbol{a}_{k,t} \approx \sum_{i=1}^N \omega_{k,t}^i \boldsymbol{a}_{k,t}^i. \quad (18)$$

The whole procedure is illustrated in Algorithm 1, where $N(j)$ represents the number of particles at the node j and $\mathcal{I}_{i\to j}$ represents the indices of the particles that move from the node i to the node j.

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

Table 1: The complexities of the algorithms for each node k. Here, we omit the derivations of the SGD and EKF based algorithms due to page limit.

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

$$0 < p_0 \leq p(d_{k,t}|\boldsymbol{a}_{k,t}) \leq ||p||_\infty < \infty, \quad (19)$$

where p_0 is a constant and $||p||_\infty$ is the maximum value of $p(d_{k,t}|\boldsymbol{a}_{k,t})$. Then, we have the following convergence results in the MSE sense

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

Proof of Theorem 1. Using (19), from [10], we obtain

$$\mathbf{E}\Big[\big(\mathbf{E}[\pi(\boldsymbol{a}_t)|\{d_{j,1:t}\}_{j=1}^K] - \sum_{i=1}^N \omega_{k,t}^i \pi(\boldsymbol{a}_{k,t}^i)\big)^2\Big] \leq ||\pi||_\infty^2 \left(C_t\sqrt{U(s,\upsilon)} + \sqrt{\frac{\varsigma_t}{N}}\right)^2, \quad (20)$$

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 [10] 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 (20) as N and s go to infinity yields the results. This concludes our proof. □

4. SIMULATIONS

We evaluate the performance of the introduced algorithms on different benchmark real datasets. Throughout this section, we also consider the SGD and EKF based algorithms without communication over a network of multiple nodes as benchmark algorithms and denote them by "SGD" and "EKF", respectively.

We first consider the Hong Kong exchange rate dataset [12]. For this dataset, we have the amount of Hong Kong dollars that can buy one U.S. dollar on certain days. Our aim is to estimate future exchange rate by using the values in the previous two days. In this experiment, we evaluate the convergence

Algorithm 1 Training based on the DPF Algorithm

Sample $\{\boldsymbol{a}_{j,t}^i\}_{i=1}^{N(j)}$ from $p(\boldsymbol{a}_t|\{\boldsymbol{a}_{j,t-1}^i\}_{i=1}^{N(j)}), \forall j$
Set $\{w_{j,t}^i\}_{i=1}^{N(j)} = 1, \forall j$
for s steps **do**
  Move the particles according to $\mathcal{A}$
  for $j = 1 : K$ **do**
    $\{\boldsymbol{a}_{j,t}^i\}_{i=1}^{N(j)} \leftarrow \bigcup_{l\in\mathcal{N}_j}\{\boldsymbol{a}_{l,t}^i\}_{i\in\mathcal{I}_{l\to j}}$
    $\{w_{j,t}^i\}_{i=1}^{N(j)} \leftarrow \bigcup_{l\in\mathcal{N}_j}\{w_{l,t}^i\}_{i\in\mathcal{I}_{l\to j}}$
    $\{w_{j,t}^i\}_{i=1}^{N(j)} \leftarrow \{w_{j,t}^i\}_{i=1}^{N(j)} p(d_{j,t}|\{\boldsymbol{a}_{j,t}^i\}_{i=1}^{N(j)})^{\frac{2|E(G)|}{s\eta_j}}$
  end for
end for
for j=1:K **do**
  Resample $\{\boldsymbol{a}_{j,t}^i, w_{j,t}^i\}_{i=1}^{N(j)}$
  Compute the estimate for node j using (18)
end for

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. Additionally, 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 $\mathcal{A} = [0\ \frac{1}{2}\ 0\ \frac{1}{2}; \frac{1}{2}\ 0\ \frac{1}{2}\ 0; 0\ \frac{1}{2}\ 0\ \frac{1}{2}; \frac{1}{2}\ 0\ \frac{1}{2}\ 0]$. For the EKF based algorithm, we use the same noise statistics with the PF based algorithm. For the SGD based algorithm, we set the learning rate as $\mu = 0.1$. In Fig. 2, we illustrate the prediction performances. Due to the nonlinearity of our model, the EKF based algorithm has slower convergence compared to the other algorithms. Moreover, due to only exploiting the first order gradient information, the SGD based algorithm has also slower convergence compared to the PF based algorithms. Unlike the SGD and EKF based methods, the PF based algorithms have a high performance gradient free density estimation technique, hence, they converge much faster to the final MSE level. Overall, due to its distributed structure the DPF based algorithm has the fastest convergence rate.

We then consider the sentence dataset [13], where we have the vector representation of each word in a 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 are similar. In this case, we have a variable length input regressor $\boldsymbol{X}_{k,t} \in \mathbb{R}^{2\times m_t}$, where m_t represents the number of words in a sentence. For the other parameters, we use the same setting with the previous case except $N(k) = 50$, $\text{Cov}[\boldsymbol{\gamma}_{k,t}] = (0.025)^2\boldsymbol{I}$ and $\mu = 0.055$. In Fig. 3, we illustrate the label prediction performances. Again due to the highly nonlinear structure of our model, the EKF based algorithm has the highest steady state error value. Additionally, the SGD based algorithm also has a high final MSE value compared to the other algorithms. Again,

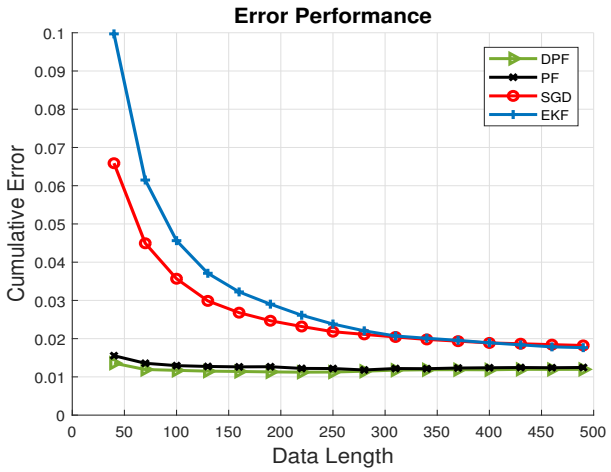

Fig. 2: Error performances over the Hong Kong exchange rate dataset.

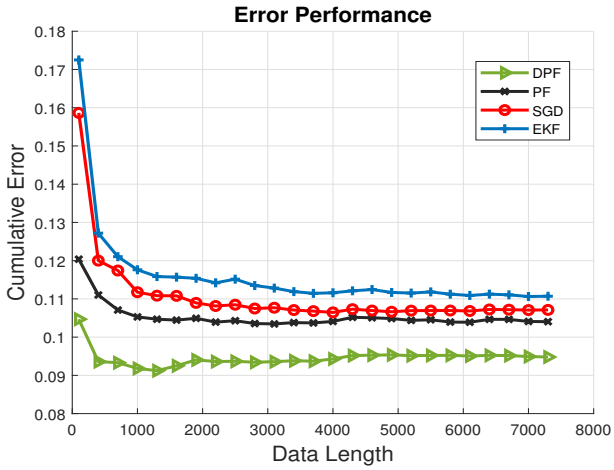

Fig. 3: Error performances over the sentence dataset.

the PF based methods achieve lower final MSE value among all the algorithms. However, since the DPF based method effectively shares information among neighboring nodes, it achieves smallest steady state error value.

5. CONCLUDING REMARKS

We study online training of the LSTM architecture in a network of nodes for regression and introduce online distributed training algorithms for variable length data sequences. We first propose an LSTM based model for variable length data inputs. To train this model, we put the model equations in a nonlinear state space form. We then introduce a distributed particle filtering based online training algorithm. Thus, we obtain an effective training algorithm for our LSTM based model. Our algorithm guarantees convergence to the optimal parameter estimation in the MSE sense. We achieve this performance with communication and computational complexity in the order of the first order methods [3]. In our simulations, we illustrate significant performance improvements with respect to the conventional methods [14, 15].

6. REFERENCES

[1] D. F. Specht, "A general regression neural network," *IEEE Transactions on Neural Networks*, 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 Transactions on Neural Networks*, 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*. Springer, 1998, pp. 27–62.

[4] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, 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] D. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Networks*, vol. 16, no. 10, pp. 1429 – 1451, 2003.

[7] X. Wu *et al.*, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, Jan 2014.

[8] K. Yuan *et al.*, "On the convergence of decentralized gradient descent," *SIAM Journal on Optimization*, vol. 26, no. 3, pp. 1835–1854, 2016.

[9] B. D. Anderson *et al.*, *Optimal filtering*. Courier Corporation, 2012.

[10] S. H. Lee and M. West, "Convergence of the markov chain distributed particle filter (MCDPF)," *IEEE Transactions on Signal Processing*, vol. 61, no. 4, pp. 801–812, Feb 2013.

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

[12] E. W. Frees, "Regression modelling with actuarial and financial applications." [Online]. Available: http://instruction.bus.wisc.edu/jfrees/jfreesbooks/Regression%20Modeling/BookWebDec2010/data.html

[13] M. Lichman, "UCI machine learning repository," 2013.

[14] J. A. Pérez-Ortiz *et al.*, "Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets," *Neural Networks*, vol. 16, no. 2, pp. 241–250, 2003.

[15] A. W. Smith and D. Zipser, "Learning sequential structure with the real-time recurrent learning algorithm," *International Journal of Neural Systems*, vol. 1, no. 02, pp. 125–131, 1989.