This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Efficient Online Learning Algorithms Based on LSTM Neural Networks

Tolga Ergen and Suleyman Serdar Kozat, *Senior Member, IEEE*

***Abstract*— We investigate online nonlinear regression and introduce novel regression structures based on the long short term memory (LSTM) networks. For the introduced structures, we also provide highly efficient and effective online training methods. To train these novel LSTM-based structures, we put the underlying architecture in a state space form and introduce highly efficient and effective particle filtering (PF)-based updates. We also provide stochastic gradient descent and extended Kalman filter-based updates. Our PF-based training method guarantees convergence to the optimal parameter estimation in the mean square error sense provided that we have a sufficient number of particles and satisfy certain technical conditions. More importantly, we achieve this performance with a computational complexity in the order of the first-order gradient-based methods by controlling the number of particles. Since our approach is generic, we also introduce a gated recurrent unit (GRU)-based approach by directly replacing the LSTM architecture with the GRU architecture, where we demonstrate the superiority of our LSTM-based approach in the sequential prediction task via different real life data sets. In addition, the experimental results illustrate significant performance improvements achieved by the introduced algorithms with respect to the conventional methods over several different benchmark real life data sets.**

***Index Terms*— Gated recurrent unit (GRU), Kalman filtering, long short term memory (LSTM), online learning, particle filtering (PF), regression, stochastic gradient descent (SGD).**

I. Introduction

A. Preliminaries

THE problem of estimating an unknown desired signal is one of the main subjects of interest in contemporary online learning literature, where we sequentially receive a data sequence related to a desired signal to predict the signal's next value [1]. This problem is known as online regression and it is extensively studied in the neural network [2], machine learning [1], and signal processing literatures [3], especially for prediction tasks [4]. In these studies, nonlinear approaches are generally employed because for certain applications, linear modeling is inadequate due to the constraints on linearity [3]. Here, in particular, we study the nonlinear regression in an online setting, where we sequentially observe a data sequence and its label to find a nonlinear relation between them to predict the future labels.

There exists a wide range of nonlinear modeling approaches in the machine learning and signal processing literatures for regression [1], [3]. However, most of these approaches usually suffer from high computational complexity and they may provide inadequate performance due to stability and overfitting issues [3]. Neural network-based regression algorithms are also introduced for nonlinear modeling since neural networks are capable of modeling highly nonlinear and complex structures [2], [4], [5]. However, they are also shown to be prone to overfitting problems and demonstrate less than adequate performance in certain applications [6], [7]. To remedy these issues and further enhance their performance, neural networks composed of multiple layers, i.e., known as deep neural networks (DNNs), are recently introduced [8]. In DNNs, a layered structure is employed so that each layer performs a feature extraction based on the previous layers [8]. With this mechanism, DNNs are able to model highly nonlinear and complex structures [9]. However, this layered structure poorly performs in capturing time dependencies in the data so that DNNs can provide only limited performance in modeling time series and processing temporal data [10]. As a remedy, basic recurrent neural networks (RNNs) are introduced since these networks have inherent memory that can store the past information [5]. However, basic RNNs lack control structures so that the long-term components cause either an exponential growth or decay in the norm of gradients during training, which are the well-known exploding and vanishing gradient problems, respectively [6], [11]. Hence, they are insufficient to capture long-term dependencies on the data, which significantly restricts their performance in real life tasks [12]. In order to resolve this issue, a novel RNN architecture with several control structures, i.e., long short term memory (LSTM) network [12], [13], is introduced. However, in the classical LSTM structures, we do not have the direct contribution of the regression vector to the output, i.e., the desired signal is regressed only using the state vector [4]. Hence, in this paper, we introduce LSTM-based online regression architectures, where we also incorporate the direct contribution of the regression vectors inspired from the well-known ARMA models [14].

After the neural network structure is fixed, there exists a wide range of different methods to train the corresponding parameters in an online manner. Especially the first-order gradient-based approaches are widely used due to their efficiency in training because of the well-known backpropagation

Manuscript received October 30, 2016; revised May 5, 2017 and August 15, 2017; accepted August 15, 2017. 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.2017.2741598

2162-237X © 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

recursion [4], [15]. However, these techniques provide poorer performance compared with the second-order gradient-based techniques [5], [16]. As an example, the real-time recurrent learning (RTRL) algorithm is highly efficient in calculating gradients [15], [16]. However, since the RTRL algorithm exploits only the first-order gradient information, it performs poorly on ill-conditioned problems [17]. On the other side, although the second-order gradient-based techniques provide much better performance, they are highly complex compared with the first-order methods [5], [16], [18]. As an example, the well-known extended Kalman filter (EKF) method also uses the second-order information to boost its performance, which requires to update the error covariance matrix of the parameter estimate and brings an additional complexity accordingly [19]. Furthermore, the second-order gradient-based methods provide limited training performance due to an abundance of saddle points in neural network-based applications [20]. To alleviate the training issues, we introduce particle filtering (PF) [21]-based online updates for the LSTM architecture. In particular, we first put the LSTM architecture in a nonlinear state space form and formulate the parameter learning problem in this setup. Based on this form, we introduce a PF-based estimation algorithm to effectively learn the parameters. Here, our training method guarantees convergence to the optimal parameter estimation performance in an online manner provided that we have sufficiently many particles and satisfy certain technical conditions. Furthermore, by controlling the amount of particles in our experiments, we demonstrate that we can significantly reduce the computational complexity while providing a superior performance compared with the conventional second-order methods. Here, our training approach is generic such that we also put the recently introduced gated recurrent unit (GRU) architecture [22] in a nonlinear state space form and then apply our algorithms to learn its parameters. Through extensive set of simulations, we illustrate significant performance improvements achieved by our algorithms compared with the conventional methods [18], [23].

B. Prior Art and Comparisons

Neural network-based learning methods are powerful in modeling highly nonlinear structures such that a single hidden layer neural network can adequately model any nonlinear structure [24]. In addition, these methods, especially complex RNNs-based methods, are capable of effectively processing temporal data and modeling time series [4], [12]. Complex RNNs, e.g., LSTM networks, provide this performance thanks to their memory to keep the past information and several control gates to regulate the information flow inside the network [12], [13]. However, for complex RNNs, adequate performance requires high computational complexity, i.e., training of a large number of parameters at every time instance [4]. Thus, to mitigate complexity, the LSTM network-based methods in [16] and [5] choose a low-complexity first-order gradient-based technique, i.e., stochastic gradient descent (SGD) [23], to train their parameters. Even though there exist certain applications of LSTM trained with the second-order techniques, e.g., EKF in [18] and a Hessian free technique in [25], they suffer from complexity issues and also poor performance due to an abundance of saddle points [20]. On the contrary, for basic RNNs, we have less parameters to train; however, these neural networks do not have control structures [12], [13]. Hence, the exploding and vanishing gradient problems occur due to long-term components [6], [11]. These problems prevent the basic RNNs from learning correlation between distant events [6]. To ameliorate performance, the basic RNN-based learning methods in [5] and [16] choose a high-complexity second-order gradient-based techniques to train their parameters. Hence, either low-complexity neural networks or low-complexity training methods are chosen to avoid unmanageable computational complexity increase. However, basic RNNs suffer from inadequately capturing long- and short-term dependencies compared with complex networks [12], [13]. On the other hand, the first-order gradient-based methods suffer from slower convergence and poorer performance compared with the second-order gradient-based techniques [5]. To circumvent these issues, in this paper, we derive online updates based on the PF algorithm [21] to train the LSTM architecture. Thus, we not only provide the second-order training without any *ad hoc* linearization but also accomplish this with a computational complexity in the order of the first-order methods (by carefully controlling the number of particles in modeling).

We emphasize that the conventional neural networks-based learning methods [5], [16], [18], [23] suffer from the well-known complexity–performance tradeoff. Due to this tradeoff, they usually are not chosen to address the nonlinear regression problem. There are certain neural network-based methods [5], [16] that particularly investigate the nonlinear regression; however, they only employ the basic RNN architecture for this purpose. In addition, in their regression approach, they provide the final estimate by setting the output of the basic RNN architecture as a scalar value so that the final estimate becomes linear combination of only the internal states. Instead, in this paper, we employ the LSTM architecture for the nonlinear regression and also introduce additional terms to incorporate the direct contribution of the regression vector to our final estimate. Therefore, we significantly improve the regression performance as illustrated in our simulations.

C. Contributions

Our main contributions are as follows.

1) As the first time in the literature, we introduce online learning algorithms based on the LSTM architecture for data regression, where we efficiently train the LSTM architecture in an online manner using our PF-based approach.
2) We propose novel LSTM-based regression structures to compute the final estimate, where we introduce an additional gate to the classical LSTM architecture to incorporate the direct contribution of the input regressor inspired from the ARMA models.
3) We put the LSTM equations in a nonlinear state space form and then derive online updates based on the state-of-the-art state estimation techniques [21], [26] for each

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

parameter. Here, our PF-based method achieves a substantial performance improvement in online parameter training with respect to the conventional second- and first-order methods [18], [23].

4) We achieve this substantial improvement with a computational complexity in the order of the first-order gradient-based methods [18], [23] by controlling the number of particles in our method. In our simulations, we also illustrate that by controlling the number of particles, we can achieve the same complexity with the first-order gradient-based methods while providing a far superior performance compared with the both first- and second-order methods.
5) Through extensive set of simulations involving real life and financial data, we illustrate performance improvements achieved by our algorithms with respect to the conventional methods [18], [23]. Furthermore, since our approach is generic, we also introduce GRU-based algorithms by directly applying our approach to the GRU architecture, i.e., also a complex RNN, in Section IV.

D. Organization of This Paper

The organization of this paper is as follows. We introduce the online regression problem and then describe our LSTM-based model in Section II. We then introduce different architectures to compute the final estimate for data regression in Section III-A. In Section III-B, we review the conventional training methods and extend these methods to the introduced architectures. We then introduce our PF-based training algorithm in Section III-C. In Section IV, we illustrate the merits of the proposed algorithms and training methods via extensive set of experiments involving real life and financial data, and we also introduce a GRU-based approach for online learning tasks. We then finalize our paper with concluding remarks in Section V.

II. Model and Problem Description

All vectors are column vectors and denoted by boldface lower case letters. Matrices are represented by boldface capital letters. For a vector $\boldsymbol{u}$ (or a matrix $\boldsymbol{U}$), $\boldsymbol{u}^T$ ($\boldsymbol{U}^T$) is the ordinary transpose. The time index is given as subscript, e.g., $\boldsymbol{u}_t$ is the vector at time t. The $\mathbf{1}$ is a vector of all ones, $\mathbf{0}$ is a vector or matrix of all zeros, $\boldsymbol{I}$ is the identity matrix, where the size is understood from the context. Given a vector $\boldsymbol{u}$, $\text{diag}(\boldsymbol{u})$ is the diagonal matrix constructed from the entries of $\boldsymbol{u}$.

We sequentially receive $\{d_t\}_{t\geq 1}$, $d_t \in \mathbb{R}$, and regression vectors, $\{\boldsymbol{x}_t\}_{t\geq 1}$, $\boldsymbol{x}_t \in \mathbb{R}^p$ such that our goal is to estimate d_t based on our current and past observations $\{\ldots, \boldsymbol{x}_{t-1}, \boldsymbol{x}_t\}$. Given our estimate $\hat{d}_t$, which can only be a function of $\{\ldots, \boldsymbol{x}_{t-1}, \boldsymbol{x}_t\}$ and $\{\ldots, d_{t-2}, d_{t-1}\}$, we suffer the loss $l(d_t, \hat{d}_t)$. This framework models a wide range of machine learning problems including financial analysis [27], tracking [28], and state estimation [19]. As an example, in one step ahead data prediction under the square error loss, where we sequentially receive data and predict the next sample, we receive $\boldsymbol{x}_t = [x_t, x_{t-1} \ldots, x_{t-p+1}]^T$ and then generate $\hat{d}_t$; after $d_t = x_{t+1}$ is observed, we suffer $l(d_t, \hat{d}_t) = (d_t - \hat{d}_t)^2$.

In this paper, to generate the sequential estimates $\hat{d}_t$, we use RNNs. The basic RNN structure is described by the following set of equations [16]:

$$\boldsymbol{h}_t = \kappa(\boldsymbol{W}^{(h)}\boldsymbol{x}_t + \boldsymbol{R}^{(h)}\boldsymbol{h}_{t-1}) \tag{1}$$

$$\boldsymbol{y}_t = u(\boldsymbol{R}^{(y)}\boldsymbol{h}_t) \tag{2}$$

where $\boldsymbol{h}_t \in \mathbb{R}^m$ is the state vector, $\boldsymbol{x}_t \in \mathbb{R}^p$ is the input, and $\boldsymbol{y}_t \in \mathbb{R}^m$ is the output. The functions $\kappa(\cdot)$ and $u(\cdot)$ apply to vectors pointwise and commonly set to $\tanh(\cdot)$. For the coefficient matrices, we have $\boldsymbol{W}^{(h)} \in \mathbb{R}^{m\times p}$, $\boldsymbol{R}^{(h)} \in \mathbb{R}^{m\times m}$, and $\boldsymbol{R}^{(y)} \in \mathbb{R}^{m\times m}$.

As a special case of RNNs, we use the LSTM neural network [12] with only one hidden layer. Although there exists a wide range of different implementations of the LSTM network, we use the most widely used extension, where the nonlinearities are set to the hyperbolic tangent function and the peephole connections are eliminated. This LSTM architecture is defined by the following set of equations [12]:

$$\boldsymbol{z}_t = h(\boldsymbol{W}^{(z)}\boldsymbol{x}_t + \boldsymbol{R}^{(z)}\boldsymbol{y}_{t-1} + \boldsymbol{b}^{(z)}) \tag{3}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{R}^{(i)}\boldsymbol{y}_{t-1} + \boldsymbol{b}^{(i)}) \tag{4}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{R}^{(f)}\boldsymbol{y}_{t-1} + \boldsymbol{b}^{(f)}) \tag{5}$$

$$\boldsymbol{c}_t = \boldsymbol{\Lambda}_t^{(i)}\boldsymbol{z}_t + \boldsymbol{\Lambda}_t^{(f)}\boldsymbol{c}_{t-1} \tag{6}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{R}^{(o)}\boldsymbol{y}_{t-1} + \boldsymbol{b}^{(o)}) \tag{7}$$

$$\boldsymbol{y}_t = \boldsymbol{\Lambda}_t^{(o)}h(\boldsymbol{c}_t) \tag{8}$$

where $\boldsymbol{\Lambda}_t^{(f)} = \text{diag}(\boldsymbol{f}_t)$, $\boldsymbol{\Lambda}_t^{(i)} = \text{diag}(\boldsymbol{i}_t)$, and $\boldsymbol{\Lambda}_t^{(o)} = \text{diag}(\boldsymbol{o}_t)$. Furthermore, $\boldsymbol{c}_t \in \mathbb{R}^m$ is the state vector, $\boldsymbol{x}_t \in \mathbb{R}^p$ is the input vector, and $\boldsymbol{y}_t \in \mathbb{R}^m$ is the output vector. Here, $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, and $\boldsymbol{o}_t$ are the input, forget, and output gates, respectively. The functions $g(\cdot)$ and $h(\cdot)$ apply to vectors pointwise and commonly set to $\tanh(\cdot)$. Similarly, the sigmoid function $\sigma(\cdot)$ applies pointwise to the vector elements. For the coefficient matrices and the weight vectors, we have $\boldsymbol{W}^{(z)} \in \mathbb{R}^{m\times p}$, $\boldsymbol{R}^{(z)} \in \mathbb{R}^{m\times m}$, $\boldsymbol{b}^{(z)} \in \mathbb{R}^m$, $\boldsymbol{W}^{(i)} \in \mathbb{R}^{m\times p}$, $\boldsymbol{R}^{(i)} \in \mathbb{R}^{m\times m}$, $\boldsymbol{b}^{(i)} \in \mathbb{R}^m$, $\boldsymbol{W}^{(f)} \in \mathbb{R}^{m\times p}$, $\boldsymbol{R}^{(f)} \in \mathbb{R}^{m\times m}$, $\boldsymbol{b}^{(f)} \in \mathbb{R}^m$, $\boldsymbol{W}^{(o)} \in \mathbb{R}^{m\times p}$, $\boldsymbol{R}^{(o)} \in \mathbb{R}^{m\times m}$, and $\boldsymbol{b}^{(o)} \in \mathbb{R}^m$. Given the output $\boldsymbol{y}_t$, we generate the final estimate as

$$\hat{d}_t = \boldsymbol{w}_t^T\boldsymbol{y}_t \tag{9}$$

where the final regression coefficients $\boldsymbol{w}_t$ will be trained in an online manner in the following. Our goal is to design the system parameters so that $\sum_{t=1}^n l(d_t, \hat{d}_t)$ or $\mathbf{E}[l(d_t, \hat{d}_t)]$ is minimized.

Remark 1: The basic LSTM network can be extended by including last s outputs in the recursion, e.g., $\{\boldsymbol{y}_{t-s}, \ldots, \boldsymbol{y}_{t-1}\}$; however, this case corresponds to an extended output definition, i.e., an extended super output vector consisting of all $\{\boldsymbol{y}_{t-s}, \ldots, \boldsymbol{y}_{t-1}\}$. We use only $\boldsymbol{y}_{t-1}$ for notational simplicity.

In the following section, we first introduce novel LSTM network-based regression architectures inspired from the ARMA models. Then, we review and extend the conventional methods [18], [23] to learn the parameters of LSTM in an online manner. Finally, we provide our novel PF-based training method.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

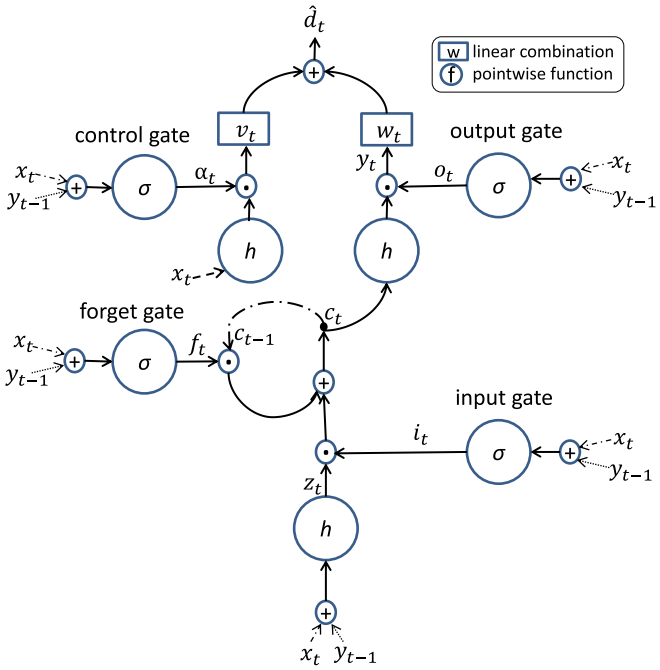

Fig. 1. Detailed schematic of the proposed architecture in (11) for the regression tasks. Note that for the summations before the gate and $h(\cdot)$ functions, we multiply $\boldsymbol{x}_t$ and $\boldsymbol{y}_{t-1}$ by $\boldsymbol{W}^{(.)}$ and $\boldsymbol{R}^{(.)}$, respectively, and also we add the weight vector $\boldsymbol{b}^{(.)}$ to these summations. We omit these operations for presentation simplicity.

III. Novel Learning Algorithms Based on LSTM Neural Networks

In this section, we first introduce our novel contributions for data regression. For these contributions, we also derive online updates based on the SGD, EKF, and PF algorithms.

A. Different Regression Architectures

We first consider the direct linear combination of the output $\boldsymbol{y}_t$ with the weight vector $\boldsymbol{w}_t$. In this case, given (8), we generate the final estimate as

$$\begin{aligned}\hat{d}_t^{(1)} &= \boldsymbol{w}_t^T \boldsymbol{y}_t \\ &= \boldsymbol{w}_t^T \boldsymbol{\Lambda}_t^{(o)} h(\boldsymbol{c}_t)\end{aligned} \tag{10}$$

where $\boldsymbol{w}_t \in \mathbb{R}^m$. In (10), the final estimate of the system does not directly depend on $\boldsymbol{x}_t$. However, in generic nonlinear regression tasks, the final estimate usually depends on the current regression vector also [29]. For this purpose, we introduce a linear term to incorporate the effects of the input vector, i.e., the regression vector, to the final estimate as shown in Fig. 1. Hence, we introduce the second regression architecture as

$$\hat{d}_t^{(2)} = \boldsymbol{w}_t^T \boldsymbol{\Lambda}_t^{(o)} h(\boldsymbol{c}_t) + \boldsymbol{v}_t^T \boldsymbol{\Lambda}_t^{(\alpha)} h(\boldsymbol{x}_t) \tag{11}$$

$\boldsymbol{v}_t \in \mathbb{R}^p$, in accordance with (10), where $\boldsymbol{\Lambda}_t^{(\alpha)} = \text{diag}(\boldsymbol{\alpha}_t)$ and

$$\boldsymbol{\alpha}_t = \sigma\left(\boldsymbol{W}^{(\alpha)} \boldsymbol{x}_t + \boldsymbol{R}^{(\alpha)} \boldsymbol{\Lambda}_{t-1}^{(o)} h(\boldsymbol{c}_{t-1}) + \boldsymbol{b}^{(\alpha)}\right).$$

Here, the final estimate directly depends on $\boldsymbol{x}_t$ and also the dependence is controlled by the control gate, i.e., $\boldsymbol{\alpha}_t$.

In (10) and (11), the effects of the input and state vectors are controlled by the control and output gates, respectively. Thus, these gates may restrict the exposure of the state and input contents in nonlinear regression problems. To expose the full content of the state and input vectors, we remove the control and output gates in (11) and introduce the third regression architecture as follows:

$$\hat{d}_t^{(3)} = \boldsymbol{w}_t^T h(\boldsymbol{c}_t) + \boldsymbol{v}_t^T h(\boldsymbol{x}_t). \tag{12}$$

Note that $\hat{d}_t^{(2)}$ is our most general architecture to compute the final estimate since the updates for $\hat{d}_t^{(1)}$ are a special case when $\boldsymbol{\Lambda}_t^{(\alpha)} = \boldsymbol{0}$ and the updates for $\hat{d}_t^{(3)}$ are a special case when $\boldsymbol{\Lambda}_t^{(o)} = \boldsymbol{I}$ and $\boldsymbol{\Lambda}_t^{(\alpha)} = \boldsymbol{I}$. In the following sections, we provide the full derivations for $\hat{d}_t^{(1)}$ for notational and presentation simplicity, and also provide the required updates to extend these basic derivations to $\hat{d}_t^{(2)}$ and $\hat{d}_t^{(3)}$.

TABLE I

Comparison of the Computational Complexities of the Proposed Online Training Methods. p Represents the Dimensionality of the Regressor Space, m Represents the Dimensionality of the Network's Output Space, and N Represents the Number of Particles for the PF Algorithm

Algorithm	Computational Complexity
SGD	$O\left(m^4 + m^2 p^2\right)$
EKF	$O\left(m^8 + m^4 p^4\right)$
PF	$O\left(N(m^2 + mp)\right)$

B. Conventional Online Training Algorithms

In this section, we introduce methods to learn the corresponding parameters of the introduced architectures in an online manner. We first derive the online updates based on the SGD algorithm [17], i.e., also known as the RTRL algorithm [23] in the neural network literature, where we derive the recursive gradient formulations to obtain the online updates for the LSTM architecture.

The SGD algorithm exploits only the first-order gradient information so that it usually converges slower compared with the second-order gradient-based techniques and performs poorly on ill-conditioned problems [17]. To mitigate these problems, we next consider the second-order gradient-based techniques, which have faster convergence rate and are more robust against ill-conditioned problems [5]. We first put the LSTM equations in a nonlinear state space form so that we can consider the EKF algorithm [19] to train the parameters in an online manner. However, the EKF algorithm requires the first-order Taylor series expansion to linearize the nonlinear network equations and this degrades its performance [5], [19]. In addition, Table I shows that the EKF algorithm has high computational complexity compared with the SGD algorithm.

In the following sections, we derive both the SGD- and EKF-based training methods and extend these derivations to the regression architectures in (10)–(12).

1) Online Learning With the SGD Algorithm: For each parameter set, we next derive the stochastic gradient updates, i.e., also known as the RTRL algorithm [23], to minimize the instantaneous loss, i.e., $l(d_t, \hat{d}_t) = (d_t - \hat{d}_t)^2$, and extend these

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

calculations to the introduced architectures. For the weight vector, we use

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \mu_t \nabla_{\boldsymbol{w}_t} l(d_t, \hat{d}_t) \\ &= \boldsymbol{w}_t + 2\mu_t (d_t - \hat{d}_t) \boldsymbol{\Lambda}_t^{(o)} h(\boldsymbol{c}_t)\end{aligned} \tag{13}$$

where for the learning rate μ_t, we have $\mu_t \to 0$ as $t \to \infty$ and $\sum_{k=1}^{t} \mu_k \to \infty$ as $t \to \infty$, e.g., $\mu_t = 1/t$. For the parameter $\boldsymbol{W}^{(z)}$, we have the following update:

$$\boldsymbol{W}^{(z)} = \boldsymbol{W}^{(z)} - \mu_t \nabla_{\boldsymbol{W}^{(z)}} l(d_t, \hat{d}_t).$$

For notational simplicity, we derive the updates for each entry of $\boldsymbol{W}^{(z)}$ separately. We denote the entry in the ith row and jth column of $\boldsymbol{W}^{(z)}$ by $w_{ij}^{(z)}$. We have the following update for each entry of $\boldsymbol{W}_{(z)}$:

$$w_{ij}^{(z)} = w_{ij}^{(z)} + 2\mu_t (d_t - \hat{d}_t) \boldsymbol{w}_t^T \frac{\partial \left(\boldsymbol{\Lambda}_t^{(o)} h(\boldsymbol{c}_t) \right)}{\partial w_{ij}^{(z)}}. \tag{14}$$

We write the partial derivative in (14) as

$$\frac{\partial \left(\boldsymbol{\Lambda}_t^{(o)} h(\boldsymbol{c}_t) \right)}{\partial w_{ij}^{(z)}} = \boldsymbol{\Lambda}_t^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}} \right)} h(\boldsymbol{c}_t) + \boldsymbol{\Lambda}_t^{(o)} \boldsymbol{\Lambda}_t^{(h'(c))} \frac{\partial \boldsymbol{c}_t}{\partial w_{ij}^{(z)}} \tag{15}$$

where $h'(\cdot)$ denotes the differential of $h(\cdot)$ with respect to its argument, $\boldsymbol{\Lambda}_t^{(h'(c))} = \operatorname{diag}(h'(\boldsymbol{c}_t))$, and

$$\boldsymbol{\Lambda}_t^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}} \right)} = \operatorname{diag}\left(\frac{\partial \boldsymbol{o}_t}{\partial w_{ij}^{(z)}} \right).$$

Now, we compute the partial derivatives of $\boldsymbol{o}_t$ and $\boldsymbol{c}_t$ with respect to $w_{ij}^{(z)}$. Taking derivative of (7) gives

$$\begin{aligned}\frac{\partial \boldsymbol{o}_t}{\partial w_{ij}^{(z)}} = \boldsymbol{\Lambda}_t^{(\sigma'(\zeta^{(o)}))} \Bigg[& \boldsymbol{R}^{(o)} \boldsymbol{\Lambda}_{t-1}^{(o)} \boldsymbol{\Lambda}_{t-1}^{(h'(c))} \boldsymbol{\psi}_{ij,t-1}^{(z)} \\ & + \boldsymbol{R}^{(o)} \boldsymbol{\Lambda}_{t-1}^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}} \right)} h(\boldsymbol{c}_{t-1}) \Bigg]\end{aligned} \tag{16}$$

where

$$\boldsymbol{\zeta}_t^{(o)} = \boldsymbol{W}^{(o)} \boldsymbol{x}_t + \boldsymbol{R}^{(o)} \boldsymbol{\Lambda}_{t-1}^{(o)} h(\boldsymbol{c}_{t-1}) + \boldsymbol{b}^{(o)} \tag{17}$$

and

$$\boldsymbol{\psi}_{ij,t-1}^{(z)} = \frac{\partial \boldsymbol{c}_{t-1}}{\partial w_{ij}^{(z)}}. \tag{18}$$

To get (15), we also compute the partial derivative of $\boldsymbol{c}_t$ with respect to $w_{ij}^{(z)}$. Using (18), we write the following recursive equation:

$$\boldsymbol{\psi}_{ij,t}^{(z)} = \boldsymbol{\Lambda}_t^{(z)} \frac{\partial \boldsymbol{i}_t}{\partial w_{ij}^{(z)}} + \boldsymbol{\Lambda}_t^{(i)} \frac{\partial \boldsymbol{z}_t}{\partial w_{ij}^{(z)}} + \boldsymbol{\Lambda}_{t-1}^{(c)} \frac{\partial \boldsymbol{f}_t}{\partial w_{ij}^{(z)}} + \boldsymbol{\Lambda}_t^{(f)} \boldsymbol{\psi}_{ij,t-1}^{(z)}. \tag{19}$$

To obtain (19), we compute the partial derivatives of (3)–(5) with respect to $w_{ij}^{(z)}$ as follows:

$$\begin{aligned}\frac{\partial \boldsymbol{i}_t}{\partial w_{ij}^{(z)}} = \boldsymbol{\Lambda}_t^{(\sigma'(\zeta^{(i)}))} \Bigg[& \boldsymbol{R}^{(i)} \boldsymbol{\Lambda}_{t-1}^{(o)} \boldsymbol{\Lambda}_{t-1}^{(h'(c))} \boldsymbol{\psi}_{ij,t-1}^{(z)} \\ & + \boldsymbol{R}^{(i)} \boldsymbol{\Lambda}_{t-1}^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}} \right)} h(\boldsymbol{c}_{t-1}) \Bigg]\end{aligned} \tag{20}$$

$$\begin{aligned}\frac{\partial \boldsymbol{f}_t}{\partial w_{ij}^{(z)}} = \boldsymbol{\Lambda}_t^{(\sigma'(\zeta^{(f)}))} \Bigg[& \boldsymbol{R}^{(f)} \boldsymbol{\Lambda}_{t-1}^{(o)} \boldsymbol{\Lambda}_{t-1}^{(h'(c))} \boldsymbol{\psi}_{ij,t-1}^{(z)} \\ & + \boldsymbol{R}^{(f)} \boldsymbol{\Lambda}_{t-1}^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}} \right)} h(\boldsymbol{c}_{t-1}) \Bigg]\end{aligned} \tag{21}$$

$$\begin{aligned}\frac{\partial \boldsymbol{z}_t}{\partial w_{ij}^{(z)}} = \boldsymbol{\Lambda}_t^{(h'(\zeta^{(z)}))} \Bigg[& \boldsymbol{\delta}_{ij} \boldsymbol{x}_t + \boldsymbol{R}^{(z)} \boldsymbol{\Lambda}_{t-1}^{(o)} \boldsymbol{\Lambda}_{t-1}^{(h'(c))} \boldsymbol{\psi}_{ij,t-1}^{(z)} \\ & + \boldsymbol{R}^{(z)} \boldsymbol{\Lambda}_{t-1}^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}} \right)} h(\boldsymbol{c}_{t-1}) \Bigg]\end{aligned} \tag{22}$$

where $\boldsymbol{\delta}_{ij}$ is an $m \times p$ matrix with all entries zeros, except a 1 in the ijth position. With these equations, we can compute (19) and then obtain (15) using (19) and (16). By this, we have all the required equations for the SGD update in (14).

Remark 2: Here, we derive the updates just for the entries of $\boldsymbol{W}^{(z)}$. When we take the partial derivative of $\hat{d}_t$ with respect to the entries of the other parameters, (14), (15), (18), and (19) still hold with a change of the derivative variable. For (16) and (20)–(22), we also have a change in the form and location of the $\boldsymbol{\delta}_{ij} \boldsymbol{x}_t$ term. In particular, as in (22), when we take the derivative of $\boldsymbol{W}^{(.)}$, $\boldsymbol{R}^{(.)}$, and $\boldsymbol{b}^{(.)}$ with respect to their entries, respectively, additional $\boldsymbol{\delta}_{ij} \boldsymbol{x}_t$, $\boldsymbol{\delta}_{ij} \boldsymbol{y}_{t-1}$, and $\boldsymbol{\delta}_{ij}$ terms appear in the derivative equation of the corresponding structure, i.e., one of (16) and (20)–(22). Here, the size of $\boldsymbol{\delta}_{ij}$ changes accordingly.

Remark 3: In case of $\hat{d}_t^{(2)}$, instead of (14), we have the following update:

$$\begin{aligned}w_{ij}^{(z)} = w_{ij}^{(z)} + 2\mu_t (d_t - \hat{d}_t) \Bigg[& \boldsymbol{w}_t^T \frac{\partial \left(\boldsymbol{\Lambda}_t^{(o)} h(\boldsymbol{c}_t) \right)}{\partial w_{ij}^{(z)}} \\ & + \boldsymbol{v}_t^T \boldsymbol{\Lambda}_t^{\left(\frac{\partial \alpha}{\partial w_{ij}^{(z)}} \right)} h(\boldsymbol{x}_t) \Bigg]\end{aligned} \tag{23}$$

where the introduced partial derivative term $\partial \boldsymbol{\alpha}_t / \partial w_{ij}^{(z)}$ is computed in the same manner with (16). Furthermore, we

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

have an additional update for $\boldsymbol{v}_t$ as follows:

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t + 2\mu_t(d_t - \hat{d}_t)\boldsymbol{\Lambda}_t^{(\alpha)} h(\boldsymbol{x}_t). \tag{24}$$

Then, we follow the derivations in (13), (15), (16), and (19)–(22). For $\hat{d}_t^{(3)}$, we just set $\boldsymbol{\Lambda}_t^{(o)} = \boldsymbol{I}$ and $\boldsymbol{\Lambda}_t^{(\alpha)} = \boldsymbol{I}$ and then all the derivations in (13), (15), (16), (19), and (20)–(24) follow as in $\hat{d}_t^{(2)}$.

According to the update equations in (15), (16), and (19), update of an entry of a parameter has a computational complexity $O(m^2 + mp)$ due to the matrix vector multiplications in (17). Since we have mp, m^2, and m entries for $\boldsymbol{W}^{(.)}$, $\boldsymbol{R}^{(.)}$, and $\boldsymbol{b}^{(.)}$, respectively, this results in $O(m^4 + m^2p^2)$ computational complexity to update the entries of all parameters as given in Table I.

2) Online Learning With the EKF Algorithm: We next provide the updates based on the EKF algorithm in order to train the parameters of the system described in (3)–(8) and (10). In the literature, there are certain EKF-based methods to train LSTM (see [18], [30]); however, these methods estimate only the parameters, i.e., $\boldsymbol{\theta}_t$. However, in our case, we also estimate the state and the output vector of LSTM, i.e., $\boldsymbol{c}_t$ and $\boldsymbol{y}_t$, respectively. In the following, we derive the updates for our approach and extend these to the introduced architectures.

The EKF algorithm assumes that the posterior density function of the states given the observations is Gaussian [19]. This assumption can be satisfied by introducing perturbations to the system equations via Gaussian noise [31]. Hence, we first write the LSTM system in a nonlinear state space form and then introduce Gaussian noise terms to be able to use the EKF updates. For convenience, we group the parameters $\{\boldsymbol{w}, \boldsymbol{W}^{(z)}, \boldsymbol{R}^{(z)}, \boldsymbol{b}^{(z)}, \boldsymbol{W}^{(i)}, \boldsymbol{R}^{(i)}, \boldsymbol{b}^{(i)}, \boldsymbol{W}^{(f)}, \boldsymbol{R}^{(f)}, \boldsymbol{b}^{(f)}, \boldsymbol{W}^{(o)}, \boldsymbol{R}^{(o)}, \boldsymbol{b}^{(o)}\}$ together into a vector $\boldsymbol{\theta}$, $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$, where $n_\theta = 4m(m+p) + 5m$. By this, we write the LSTM system as

$$\boldsymbol{y}_t = \tau(\boldsymbol{c}_t, \boldsymbol{x}_t, \boldsymbol{y}_{t-1}) + \boldsymbol{\epsilon}_t \tag{25}$$
$$\boldsymbol{c}_t = \Omega(\boldsymbol{c}_{t-1}, \boldsymbol{x}_t, \boldsymbol{y}_{t-1}) + \boldsymbol{v}_t \tag{26}$$
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \boldsymbol{e}_t \tag{27}$$
$$d_t = \boldsymbol{w}_t^T \boldsymbol{y}_t + \varepsilon_t \tag{28}$$

where $\tau(\cdot)$ and $\Omega(\cdot)$ are the nonlinear functions in (8) and (6), respectively, and $\boldsymbol{\epsilon}_t$, $\boldsymbol{e}_t$, $\boldsymbol{v}_t$, and ε_t are zero-mean Gaussian random variables. In addition, $[\boldsymbol{\epsilon}_t^T, \boldsymbol{v}_t^T, \boldsymbol{e}_t^T]^T$, and ε_t are with variances $\boldsymbol{Q}_t$ and R_t, respectively. Here, we assume that $\boldsymbol{Q}_t$ and R_t are known or can be estimated from the data as detailed later in this paper. We write (25)–(27) in a compact form as

$$\begin{bmatrix} \boldsymbol{y}_t \\ \boldsymbol{c}_t \\ \boldsymbol{\theta}_t \end{bmatrix} = \begin{bmatrix} \tau(\boldsymbol{c}_t, \boldsymbol{x}_t, \boldsymbol{y}_{t-1}) \\ \Omega(\boldsymbol{c}_{t-1}, \boldsymbol{x}_t, \boldsymbol{y}_{t-1}) \\ \boldsymbol{\theta}_{t-1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\epsilon}_t \\ \boldsymbol{v}_t \\ \boldsymbol{e}_t \end{bmatrix} \tag{29}$$
$$d_t = \boldsymbol{w}_t^T \boldsymbol{y}_t + \varepsilon_t. \tag{30}$$

In the system described in (29) and (30), we are able to observe only d_t and we can estimate $\boldsymbol{y}_t$, $\boldsymbol{c}_t$, and $\boldsymbol{\theta}_t$ based on the observed d_t values. Thus, we directly apply the EKF algorithm [19] to estimate $\boldsymbol{y}_t$, $\boldsymbol{c}_t$, and $\boldsymbol{\theta}_t$ as follows:

$$\begin{bmatrix} \boldsymbol{y}_{t|t} \\ \boldsymbol{c}_{t|t} \\ \boldsymbol{\theta}_{t|t} \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}_{t|t-1} \\ \boldsymbol{c}_{t|t-1} \\ \boldsymbol{\theta}_{t|t-1} \end{bmatrix} + \boldsymbol{L}_t\big(d_t - \boldsymbol{w}_{t|t-1}^T \boldsymbol{y}_{t|t-1}\big) \tag{31}$$
$$\boldsymbol{y}_{t|t-1} = \tau(\boldsymbol{c}_{t|t-1}, \boldsymbol{x}_t, \boldsymbol{y}_{t-1|t-1}) \tag{32}$$
$$\boldsymbol{c}_{t|t-1} = \Omega(\boldsymbol{c}_{t-1|t-1}, \boldsymbol{x}_t, \boldsymbol{y}_{t-1|t-1}) \tag{33}$$
$$\boldsymbol{\theta}_{t|t-1} = \boldsymbol{\theta}_{t-1|t-1} \tag{34}$$
$$\boldsymbol{L}_t = \boldsymbol{\Sigma}_{t|t-1}\boldsymbol{H}_t\big(\boldsymbol{H}_t^T \boldsymbol{\Sigma}_{t|t-1}\boldsymbol{H}_t + R_t\big)^{-1} \tag{35}$$
$$\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{L}_t\boldsymbol{H}_t^T\boldsymbol{\Sigma}_{t|t-1} \tag{36}$$
$$\boldsymbol{\Sigma}_{t|t-1} = \boldsymbol{F}_{t-1}\boldsymbol{\Sigma}_{t-1|t-1}\boldsymbol{F}_{t-1}^T + \boldsymbol{Q}_{t-1} \tag{37}$$

where $\boldsymbol{\Sigma} \in \mathbb{R}^{(2m+n_\theta)\times(2m+n_\theta)}$ is the error covariance matrix, $\boldsymbol{L}_t \in \mathbb{R}^{(2m+n_\theta)}$ is the Kalman gain, $\boldsymbol{Q}_t \in \mathbb{R}^{(2m+n_\theta)\times\ (2m+n_\theta)}$ is the process noise covariance, and $R_t \in \mathbb{R}$ is the measurement noise variance. We compute $\boldsymbol{H}_t$ and $\boldsymbol{F}_t$ as follows:

$$\boldsymbol{H}_t^T = \left[\frac{\partial \hat{d}_t}{\partial \boldsymbol{y}} \quad \frac{\partial \hat{d}_t}{\partial \boldsymbol{c}} \quad \frac{\partial \hat{d}_t}{\partial \boldsymbol{\theta}}\right]\Bigg|_{\substack{\boldsymbol{y}=\boldsymbol{y}_{t|t-1} \\ \boldsymbol{c}=\boldsymbol{c}_{t|t-1} \\ \boldsymbol{\theta}=\boldsymbol{\theta}_{t|t-1}}} \tag{38}$$

and

$$\boldsymbol{F}_t = \begin{bmatrix} \dfrac{\partial \tau(\boldsymbol{c}, \boldsymbol{x}_t, \boldsymbol{y})}{\partial \boldsymbol{y}} & \dfrac{\partial \tau(\boldsymbol{c}, \boldsymbol{x}_t, \boldsymbol{y})}{\partial \boldsymbol{c}} & \dfrac{\partial \tau(\boldsymbol{c}, \boldsymbol{x}_t, \boldsymbol{y})}{\partial \boldsymbol{\theta}} \\ \dfrac{\partial \Omega(\boldsymbol{c}, \boldsymbol{x}_t, \boldsymbol{y})}{\partial \boldsymbol{y}} & \dfrac{\partial \Omega(\boldsymbol{c}, \boldsymbol{x}_t, \boldsymbol{y})}{\partial \boldsymbol{c}} & \dfrac{\partial \Omega(\boldsymbol{c}, \boldsymbol{x}_t, \boldsymbol{y})}{\partial \boldsymbol{\theta}} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}\Bigg|_{\substack{\boldsymbol{y}=\boldsymbol{y}_{t|t} \\ \boldsymbol{c}=\boldsymbol{c}_{t|t} \\ \boldsymbol{\theta}=\boldsymbol{\theta}_{t|t}}}$$

where $\boldsymbol{F}_t \in \mathbb{R}^{(2m+n_\theta)\times\ (2m+n_\theta)}$ and $\boldsymbol{H}_t \in \mathbb{R}^{(2m+n_\theta)}$. For (35) and (37), we use $\boldsymbol{Q}_t$ and R_t; however, these may not be known in advance. To estimate R_t, we can use exponential smoothing as follows:

$$R_t = (1-\alpha)R_{t-1} + \alpha\lambda_t^2$$

where $0 < \alpha < 1$ is the smoothing constant and

$$\lambda_t = \big(d_t - \boldsymbol{w}_{t|t-1}^T \boldsymbol{y}_{t|t-1}\big). \tag{39}$$

For the estimation of $\boldsymbol{Q}_t$, we cannot use the exponential smoothing technique due to our inability to observe the states at each time instance. Although there exists a wide variety of techniques to estimate $\boldsymbol{Q}_t$, we use the algorithm in [32], which provides a highly effective estimate of $\boldsymbol{Q}_t$.

Remark 4: For the EKF derivations of $\hat{d}_t^{(2)}$, we change the observation model in (30), the update in (31), the Jacobian computation in (38), and the definition in (39) according to the definition of the architecture in (11). In addition, we also extend the parameter vector $\boldsymbol{\theta}_t$ by adding $\boldsymbol{v}_t$, $\boldsymbol{W}^{(\alpha)}$, $\boldsymbol{R}^{(\alpha)}$, and $\boldsymbol{b}^{(\alpha)}$. Hence, we have $\boldsymbol{\theta}_t \in \mathbb{R}^{n_\theta}$, where $n_\theta = (4m + p)(m+p)+5m+2p$. For the EKF derivations of $\hat{d}_t^{(3)}$, we change the observation model in (30), the update in (31), the Jacobian computation in (38), and the definition in (39) according to (12). Moreover, we modify $\boldsymbol{\theta}_t$ by removing $\boldsymbol{W}^{(\alpha)}$, $\boldsymbol{R}^{(\alpha)}$, $\boldsymbol{b}^{(\alpha)}$, $\boldsymbol{W}^{(o)}$, $\boldsymbol{R}^{(o)}$, and $\boldsymbol{b}^{(o)}$ from its definition for $\hat{d}_t^{(2)}$. Hence, we obtain $\boldsymbol{\theta}_t \in \mathbb{R}^{n_\theta}$, where $n_\theta = 3m(m+p) + 4m + p$.

According to the update equations in (31)–(33) and (35)–(37), the computational complexity of the updates based on the EKF algorithm results in $O(m^8 + m^4p^4)$ due to the matrix multiplications in (35)–(37).

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

C. Online Training Based on the PF Algorithm

Since the conventional training methods [18], [23] provide restricted performance as explained in the previous section, we introduce a novel PF-based method that provides superior performance compared with the second-order training methods. Furthermore, we achieve this performance with a computational complexity in the order of the first-order methods depending on the choice of N as shown in Table I. In the following, we derive the updates for our PF-based training method and extend these calculations to the introduced architectures.

The PF algorithm [21] requires no assumptions other than the independence of noise samples in (29) and (30). Hence, we modify the system in (29) and (30) as follows:

$$\boldsymbol{a}_t = \varphi(\boldsymbol{a}_{t-1}, \boldsymbol{x}_t) + \boldsymbol{\eta}_t \tag{40}$$

$$d_t = \boldsymbol{w}_t^T \boldsymbol{y}_t + \xi_t \tag{41}$$

where $\boldsymbol{\eta}_t$ and ξ_t are independent noise samples, $\varphi(\cdot,\cdot)$ is the nonlinear mapping in (29), and

$$\boldsymbol{a}_t = \begin{bmatrix} \boldsymbol{y}_t \\ \boldsymbol{c}_t \\ \boldsymbol{\theta}_t \end{bmatrix}.$$

For (40) and (41), we seek to obtain $\mathbf{E}[\boldsymbol{a}_t|d_{1:t}]$, i.e., the optimal state estimate in the mean square error (MSE) sense. For this purpose, we first find the posterior probability density function $p(\boldsymbol{a}_t|d_{1:t})$. We then calculate the conditional mean of the state vector based on the posterior density function. To obtain the density function, we employ the PF algorithm [21] as follows.

Let $\{\boldsymbol{a}_t^i, \omega_t^i\}_{i=1}^N$ denote the samples and the associated weights of the desired distribution, i.e., $p(\boldsymbol{a}_t|d_{1:t})$. Then, we obtain the desired distribution from its samples as follows:

$$p(\boldsymbol{a}_t|d_{1:t}) \approx \sum_{i=1}^{N} \omega_t^i \delta(\boldsymbol{a}_t - \boldsymbol{a}_t^i) \tag{42}$$

where $\delta(\cdot)$ represents the Dirac delta function. Since obtaining the samples from the desired distribution is intractable in most cases [21], an intermediate function is introduced to obtain the samples $\{\boldsymbol{a}_t^i\}_{i=1}^N$, which is called as importance function [21]. Hence, we first obtain the samples from the importance function and then estimate the desired density function based on these samples as follows. As an example, in order to calculate $\mathbf{E}_p[\boldsymbol{a}_t|d_{1:t}]$, we use the following trick:

$$\mathbf{E}_p[\boldsymbol{a}_t|d_{1:t}] = \mathbf{E}_q\left[\boldsymbol{a}_t \frac{p(\boldsymbol{a}_t|d_{1:t})}{q(\boldsymbol{a}_t|d_{1:t})}\middle| d_{1:t}\right]$$

where $\mathbf{E}_f$ represents an expectation operation with respect to a certain density function $f(\cdot)$. Hence, we observe that we can use $q(\cdot)$, i.e., called as importance function, when direct sampling from the desired distribution $p(\cdot)$ is intractable. Here, we use $q(\boldsymbol{a}_t|d_{1:t})$ as our importance function to obtain the samples and the corresponding weights are calculated as follows:

$$\omega_t^i \propto \frac{p(\boldsymbol{a}_t^i|d_{1:t})}{q(\boldsymbol{a}_t^i|d_{1:t})} \tag{43}$$

where the weights are normalized such that

$$\sum_{i=1}^{N} \omega_t^i = 1.$$

To simplify the weight calculation, we can factorize (43) to obtain a recursive formulation for the update of the weights as follows [26]:

$$\omega_t^i \propto \frac{p\left(d_t|\boldsymbol{a}_t^i\right) p\left(\boldsymbol{a}_t^i|\boldsymbol{a}_{t-1}^i\right)}{q\left(\boldsymbol{a}_t^i|\boldsymbol{a}_{t-1}^i, d_t\right)} \omega_{t-1}^i. \tag{44}$$

In (44), we aim to choose the importance function such that the variance of the weights is minimized. Thus, we can guarantee that all the particles have nonnegligible weights and contribute considerably to (42) [33]. In this sense, the optimal choice of the importance function is $p(\boldsymbol{a}_t|\boldsymbol{a}_{t-1}^i, d_t)$; however, this requires an integration that does not have an analytic form in most cases [34]. Thus, we choose $p(\boldsymbol{a}_t|\boldsymbol{a}_{t-1}^i)$ as the importance function, which provides a small variance for the weights but not zero as the optimal importance function does [21], [34]. This simplifies (44) as follows:

$$\omega_t^i \propto p\left(d_t|\boldsymbol{a}_t^i\right) \omega_{t-1}^i. \tag{45}$$

We can now get the desired distribution to compute the conditional mean of the augmented state vector $\boldsymbol{a}_t$ using (42) and (45). By this, we obtain the conditional mean for $\boldsymbol{a}_t$ as follows:

$$\begin{aligned} \mathbf{E}[\boldsymbol{a}_t|d_{1:t}] &= \int \boldsymbol{a}_t\, p(\boldsymbol{a}_t|d_{1:t}) d\boldsymbol{a}_t \\ &\approx \int \boldsymbol{a}_t \sum_{i=1}^{N} \omega_t^i \delta\left(\boldsymbol{a}_t - \boldsymbol{a}_t^i\right) d\boldsymbol{a}_t = \sum_{i=1}^{N} \omega_t^i \boldsymbol{a}_t^i. \end{aligned} \tag{46}$$

While applying the PF algorithm, the variance of the weights inevitably increases over time so that after a few time steps, all but one of the weights get values that are very close to zero [33]. Due to this reason, although particles with very small weights have almost no contribution to our estimate in (46), we have to update them using (40) and (45). Hence, most of our computational effort is used for the particles with negligible weights, which is known as the degeneracy problem [21]. To measure degeneracy, we use the effective sample size introduced in [35], which is calculated as follows:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} \left(\omega_t^i\right)^2}. \tag{47}$$

Note that a small N_{eff} value indicates that the variance of the weights is high, i.e., the degeneracy problem. If N_{eff} is smaller than a certain threshold [33], then we apply the resampling algorithm introduced in [26], which eliminates the particles with negligible weights and focuses on the particles with large weights to avoid degeneracy. By this, we obtain an online training method (see Algorithm 1 for the pseudocode) that converges to $\mathbf{E}[\boldsymbol{a}_t|d_{1:t}]$, where the convergence is guaranteed under certain conditions as follows.

Remark 5: For the PF derivations of $\hat{d}_t^{(2)}$, we change the observation model in (41) according to the definition in (11). We also modify $\boldsymbol{a}_t$ by adding $\boldsymbol{v}_t$, $\boldsymbol{W}^{(\alpha)}$, $\boldsymbol{R}^{(\alpha)}$, and $\boldsymbol{b}^{(\alpha)}$ to $\boldsymbol{\theta}_t$.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

For the PF derivations of $\hat{d}_t^{(3)}$, we modify (41) according to the definition in (12). Furthermore, we modify $\boldsymbol{\theta}_t$ by removing $\boldsymbol{W}^{(\alpha)}$, $\boldsymbol{R}^{(\alpha)}$, $\boldsymbol{b}^{(\alpha)}$, $\boldsymbol{W}^{(o)}$, $\boldsymbol{R}^{(o)}$, and $\boldsymbol{b}^{(o)}$ from its definition for $\hat{d}_t^{(2)}$.

Theorem 1: Let $\boldsymbol{a}_t$ be the state vector such that

$$\sup_{\boldsymbol{a}_t} |\boldsymbol{a}_t|^4 p(d_t|\boldsymbol{a}_t) < K_t \tag{48}$$

where K_t is a finite constant independent of N. Then we have the following convergence result:

$$\sum_{i=1}^{N} \omega_t^i \boldsymbol{a}_t^i \to \mathbf{E}[\boldsymbol{a}_t|d_{1:t}] \text{ as } N \to \infty.$$

Proof of Theorem 1. From [36], we have

$$\mathbf{E}\left[\left|\mathbf{E}[\pi(\boldsymbol{a}_t)|d_{1:t}] - \sum_{i=1}^{N} \omega_t^i \pi\left(\boldsymbol{a}_t^i\right)\right|^4\right] \leq C_t \frac{||\pi||_{t,4}^4}{N^2} \tag{49}$$

where

$$||\pi||_{t,4} \triangleq \max\{1, (\mathbf{E}[|\pi(\boldsymbol{a}_{t'})|^4|d_{1:t'}])^{\frac{1}{4}}, t' = 1, 2, \ldots, t\}$$

$\pi \in B_t^4$, i.e., a class of functions with certain properties described in [36], and C_t represents a finite constant independent of N. With (48), $\pi(\boldsymbol{a}_t) = \boldsymbol{a}_t$ satisfies the conditions of B_t^4. Therefore, applying $\pi(\boldsymbol{a}_t) = \boldsymbol{a}_t$ to (49) and then evaluating (49) as N goes to infinity conclude our proof. □

This theorem provides a convergence result under (48). The inequality in (48) implies that the conditional distribution of the observations, i.e., $p(d_t|\boldsymbol{a}_t)$, decays faster than $\boldsymbol{a}_t$ increases [36]. Since generic distributions usually decrease exponentially, e.g., Gaussian distribution, or they are nonzero only for bounded intervals, (48) is not a strict assumption for $\boldsymbol{a}_t$. Hence, we can conclude that Theorem 1 can be employed for most cases.

According to update equations in (40), (41), (45), and (46), each particle costs $O(m^2 + mp)$ due to the matrix vector multiplications in (40) and (41), and this results in $O(N(m^2 + mp))$ computational complexity to update all particles.

IV. Simulations

In this section, we illustrate the performances of our algorithms on different benchmark real data sets under various scenarios. We first consider the regression performance for real life data sets such as kinematic [37], elevators [38], bank [39], and pumadyn [38]. We then consider the regression performance for financial data sets, e.g., Alcoa stock price [40] and Hong Kong exchange rate data [41]. We then compare the performances of the algorithms based on two different neural networks, i.e., the LSTM and GRU networks [22]. Finally, we comparatively illustrate the merits of our LSTM-based regression architectures described in (10)–(12).

Throughout this section, "Architecture 1" represents the LSTM network with (10) as the final estimate equation, similarly "Architecture 2" represents the LSTM network with (11), and "Architecture 3" represents the LSTM network with (12).

Algorithm 1 Online Training Based on the PF Algorithm

1: **for** $i = 1 : N$ **do**
2: Draw $\boldsymbol{a}_t^i \sim p(\boldsymbol{a}_t|\boldsymbol{a}_{t-1}^i)$
3: Assign w_t^i according to (45)
4: **end for**
5: Calculate total weight: $S = \sum_{j=1}^{N} w_t^j$
6: **for** $i = 1 : N$ **do**
7: Normalize: $w_t^i = w_t^i / S$
8: **end for**
9: Calculate N_{eff} according to (47)
10: **if** $N_{eff} < N_T$ **then** $\%N_T$ is a threshold for N_{eff}
11: Apply the resampling algorithm in [26]
12: Obtain new pairs $\{\bar{\boldsymbol{a}}_t^i, \bar{\omega}_t^i\}_{i=1}^N$, where $\bar{w}_t^i = 1/N, \forall i$
13: **end if**
14: Using $\{\bar{\boldsymbol{a}}_t^i, \bar{\omega}_t^i\}_{i=1}^N$, compute the estimate according to (46)

A. Real Life Data Sets

In this section, we evaluate the performances of the algorithms for the real life data sets. We first evaluate the performances of the algorithms for the kinematic data set [37]. We then examine the effect of the number of particles on the convergence rate of the PF-based algorithm using the same data set. Furthermore, in order to illustrate the effects of model size while keeping the computation time same, we perform another experiment on the same data set for the PF-based algorithm. Finally, we consider three benchmark real data sets, i.e., elevators [38], bank [39], and pumadyn [38], to evaluate the regression performances of our algorithms.

We first consider the kinematic data set [37], i.e., a simulation of eight-link all-revolute robotic arm. Our aim is to predict the distance of the effector from a target. We first select a fixed architecture. For this purpose, we can choose any one of three architectures since the algorithm with the best performance is the same for all three architectures as detailed later in this section. Here, we choose Architecture 1. Furthermore, we choose the parameters such that all the introduced algorithms reach their maximum performance for fair comparison. To provide this fair setup, we have the following parameters. For this data set, the input vector is $\boldsymbol{x}_t \in \mathbb{R}^8$ and we set the output dimension of the neural network as $m = 8$. For the PF-based algorithm, the crucial parameter is the number of particles; we set this parameter as $N = 1500$. In addition, we choose $\boldsymbol{\eta}_t$ and ξ_t as zero-mean Gaussian random variables with the covariance $\text{Cov}[\boldsymbol{\eta}_t] = 0.01\boldsymbol{I}$ and the variance $\text{Var}[\xi_t] = 0.25$, respectively. For the EKF-based algorithm, we choose the initial error covariance as $\boldsymbol{\Sigma}_{0|0} = 0.01\boldsymbol{I}$. Moreover, we choose $\boldsymbol{Q}_t = 0.01\boldsymbol{I}$ and $R_t = 0.25$. For the SGD-based algorithm, we set the learning rate as $\mu = 0.03$. As seen in Fig. 2, the PF-based algorithm converges to a much smaller final MSE level, and hence significantly outperforms the other algorithms.

In order to illustrate the effect of the number of particles on the convergence rate, we perform a new experiment on the kinematic data set, where we use the same setup except the

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

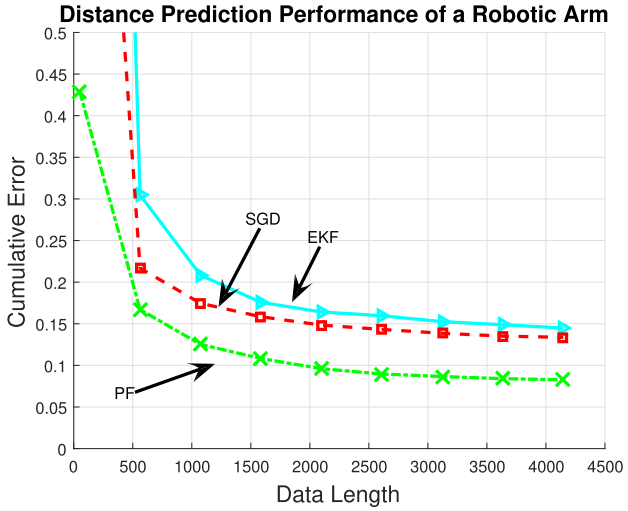

Fig. 2. Sequential prediction performances of the algorithms for the kinematic data set.

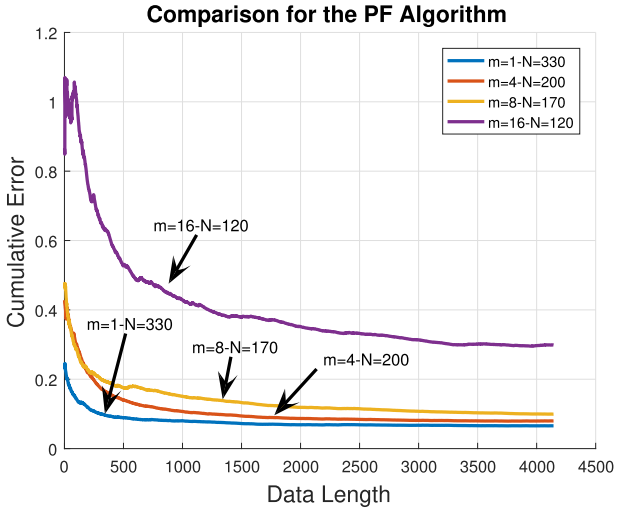

Fig. 4. Comparison of the PF-based algorithm with different N-m combinations for the kinematic data set. Note that all the combinations have the same computation time.

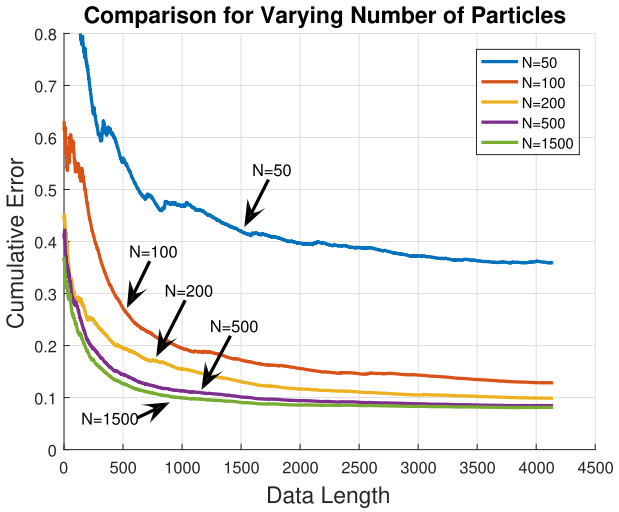

Fig. 3. Comparison of the PF-based algorithm with different number of particles for the kinematic data set.

number of particles. In Fig. 3, we observe that as the number of particles increases, the PF-based algorithm achieves a lower MSE value with a faster convergence rate. Furthermore, as N increases, the marginal performance improvement achieved becomes smaller compared with the previous N values. As an example, we observed that even though there is a significant improvement between $N = 50$ and $N = 100$ cases, there is a slight improvement between $N = 500$ and $N = 1500$ cases. Hence, if we further increase N, the marginal performance improvement may not worth the increase in the computational complexity for our case. Thus, we illustrate that $N = 1500$ is a reasonable choice for our simulations.

In addition to the simulation for the convergence rate, we perform another experiment on the same data set in order to observe the effects of model size while keeping the computation time the same. To provide this setup, we choose four different output dimensions, i.e., m, and the number of particles, i.e., N, combinations so that each combination consumes the same amount of the computation time. In Fig. 4, we observed that as the model size increases, the performance of the PF-based algorithm decreases. Since the PF-based algorithm approximates a density function based on the particles, as the number of particles decreases, we expect to obtain worse approximations for the density function. Hence, Fig. 4 matches with our interpretation for the PF-based algorithm.

Other than the kinematic data set, we also consider the elevators [38], bank [39], and pumadyn [38] data sets. For all of these data sets, we again select a fixed architecture, i.e., Architecture 1. In addition, we choose the performance maximizing parameters while forcing the PF-based algorithm to consume less training time than the other algorithms by controlling N. With this setup, we have the following parameter selection for each data set. The elevators data set is obtained from the procedure that is related to controlling an F16 aircraft and our aim is to predict the variable that expresses the actions of the aircraft. For this data set, we have $\boldsymbol{x}_t \in \mathbb{R}^{18}$ and we set the output dimension of the neural network as $m = 18$. For the other parameters, we use the same settings with the kinematic data set case except that we choose $N = 100$, $\boldsymbol{Q}_t = 0.0016\boldsymbol{I}$, $\text{Cov}[\boldsymbol{\eta}_t] = 0.0016\boldsymbol{I}$, and $\mu = 0.7$. Moreover, the pumadyn data set is obtained from the simulation of Unimation Puma 560 robotic arm and our goal is to predict the angular acceleration of the arm. We have $\boldsymbol{x}_t \in \mathbb{R}^{32}$ and we set the output dimension of the neural network as $m = 32$. In addition, we set the learning rate as $\mu = 0.4$ and the number of particles as $N = 170$. For the other parameters, we use the same settings with the elevators data set case. Finally, the bank data set is generated from a simulator that simulates the queues in banks and our aim is to predict the fraction of the customers that leave the bank due to full queues. In this case, we have $\boldsymbol{x}_t \in \mathbb{R}^{32}$ and we set the output dimension of the neural network as $m = 32$. Moreover, we set the learning rate as $\mu = 0.07$ and the number of particles as $N = 150$. For the other parameters, we use the same settings with the elevators data set case. As shown in Table II, the PF-based algorithm achieves a smaller time accumulated error

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE II

TIME ACCUMULATED ERRORS AND THE CORRESPONDING TRAINING TIMES (IN SECONDS) OF THE LSTM-BASED ALGORITHMS FOR THE ELEVATORS, PUMADYN, AND BANK DATA SETS. NOTE THAT HERE WE USE A COMPUTER WITH i5-6400 PROCESSOR, 2.7-GHz CPU, AND 16-GB RAM

Algorithms / Data Sets	PF		EKF		SGD	
	Error	Time	Error	Time	Error	Time
Elevators	**5.26×10^{-4}**	**5.5s**	6.61×10^{-4}	12.12s	6.84×10^{-4}	6.16s
Pumadyn	**0.0010**	**9.64s**	0.0013	20.17s	0.0015	10.97s
Bank	**0.016**	**2.48s**	0.018	5.50s	0.022	2.78s

value while consuming less training time compared with its competitors; therefore, it has superior performance compared with the other algorithms in these real life tasks.

B. Financial Data Sets

In this section, we evaluate the performances of the algorithms under two different financial scenarios. We first consider the Alcoa stock price data set [40], which contains the daily stock price values. Our goal is to predict the future prices by examining the past prices. As in the previous section, we first choose a fixed architecture. Since for all architectures, we obtain the best performance from the same algorithm as detailed later in this section, we can choose any architecture. Hence, we again choose Architecture 1. Moreover, we set the parameters such that all the introduced algorithms converge to the same steady-state error level. To provide this fair setup, we choose the parameters as follows. For the Alcoa stock price data set, we choose to examine the price of the previous five days, so that we have the input $\boldsymbol{x}_t \in \mathbb{R}^5$ and we set the output dimension of the neural network as $m = 5$. For the PF-based algorithm, we set the number of particles as $N = 2000$. In addition, we choose $\boldsymbol{\eta}_t$ and ξ_t as zero mean Gaussian random variables with $\text{Cov}[\boldsymbol{\eta}_t] = 0.0036\boldsymbol{I}$ and $\text{Var}[\xi_t] = 0.01$. For the EKF-based algorithm, we choose $\boldsymbol{\Sigma}_{0|0} = 0.0036\boldsymbol{I}$, $\boldsymbol{Q}_t = 0.0036\boldsymbol{I}$, and $R_t = 0.01$. For the SGD-based algorithm, we set the learning rate as $\mu = 0.1$. With these fair settings, Fig. 5 illustrates that the PF-based algorithm converges much faster.

Aside from the Alcoa stock price data set, we also consider the Hong Kong exchange rate data set [41], for which we have the amount of Hong Kong dollars that one is able to buy for US$1 on a daily basis. Our aim is to predict the future exchange rates by exploiting the data of the previous five days. We again choose Architecture 1 and then we select the parameter such that the convergence rates of the algorithms are the same. We use the same parameters with the Alcoa stock price data set case except $\boldsymbol{Q}_t = 0.0004\boldsymbol{I}$ and $\text{Cov}[\boldsymbol{\eta}_t] = 0.0004\boldsymbol{I}$. In this case, Fig. 6 shows that the PF-based algorithm converges to a much smaller steady-state error value.

C. LSTM and GRU Networks

In this section, we consider the regression performances of the algorithms based on two different RNNs, i.e., the LSTM and GRU networks. In the previous sections, we use the LSTM architecture. Since our approach is generic, we also apply our approach to the recently introduced GRU architecture, which is described by the following set of

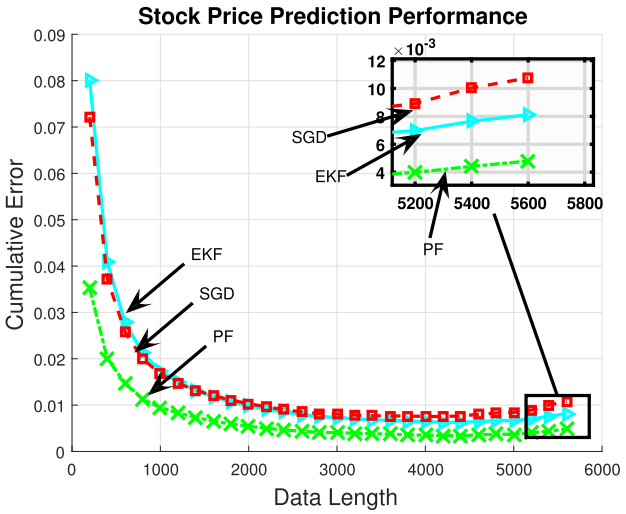

Fig. 5. Future price prediction performances of the algorithms for the Alcoa stock price data set.

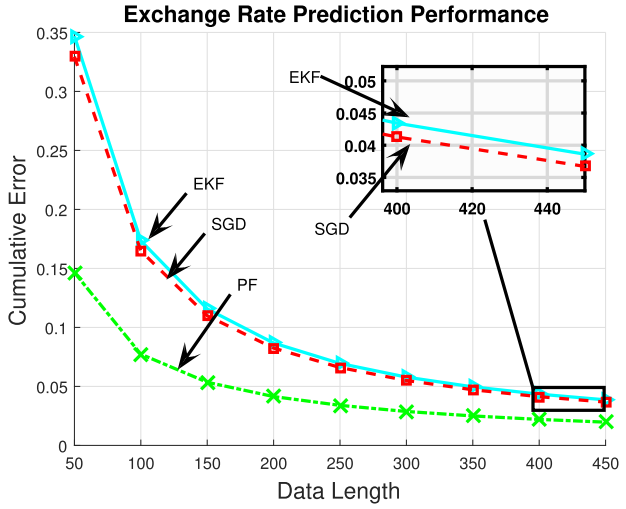

Fig. 6. Exchange rate prediction performances of the algorithms for the Hong Kong exchange rate data set.

equations [22]:

$$\tilde{\boldsymbol{z}}_t = \sigma(\boldsymbol{W}^{(\tilde{z})}\boldsymbol{x}_t + \boldsymbol{R}^{(\tilde{z})}\boldsymbol{y}_{t-1}) \tag{50}$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}^{(r)}\boldsymbol{x}_t + \boldsymbol{R}^{(r)}\boldsymbol{y}_{t-1}) \tag{51}$$

$$\tilde{\boldsymbol{y}}_t = g(\boldsymbol{W}^{(y)}\boldsymbol{x}_t + \boldsymbol{r}_t \odot (\boldsymbol{R}^{(y)}\boldsymbol{y}_{t-1})) \tag{52}$$

$$\boldsymbol{y}_t = \tilde{\boldsymbol{y}}_t \odot \tilde{\boldsymbol{z}}_t + \boldsymbol{y}_{t-1} \odot (\boldsymbol{1} - \tilde{\boldsymbol{z}}_t) \tag{53}$$

where $\boldsymbol{x}_t \in \mathbb{R}^p$ is the input vector and $\boldsymbol{y}_t \in \mathbb{R}^m$ is the output vector. The functions $g(.)$ and $\sigma(.)$ are set to the hyperbolic tangent and sigmoid functions, respectively. For the coefficient matrices, we have $\boldsymbol{W}^{(\tilde{z})} \in \mathbb{R}^{m\times p}$, $\boldsymbol{R}^{(\tilde{z})} \in \mathbb{R}^{m\times m}$, $\boldsymbol{W}^{(r)} \in \mathbb{R}^{m\times p}$, $\boldsymbol{R}^{(r)} \in \mathbb{R}^{m\times m}$, $\boldsymbol{W}^{(y)} \in \mathbb{R}^{m\times p}$, and $\boldsymbol{R}^{(y)} \in \mathbb{R}^{m\times m}$. Here, $\tilde{\boldsymbol{z}}_t$ and $\boldsymbol{r}_t$ are the update and reset gates, respectively. To obtain GRU-based algorithms, we directly replace the LSTM equations with the GRU equations and then apply our regression and training approaches. However, the GRU network lacks the output gate, which controls the amount of the incoming memory content. Furthermore, these networks differ in the location of the forget gates or the corresponding reset gates.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

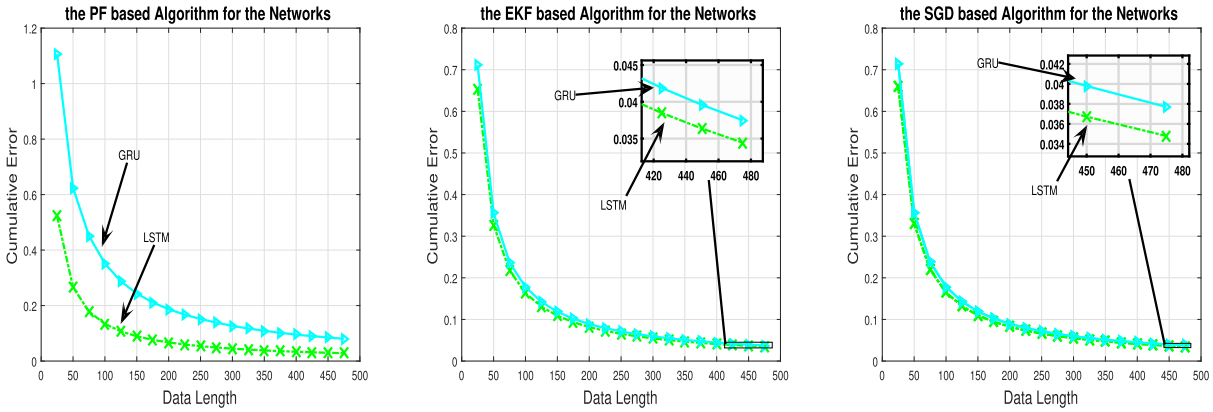

Fig. 7. Comparison of the LSTM and GRU architectures in terms of regression error performance for (a) PF-based algorithm, (b) EKF-based algorithm, and (c) SGD-based algorithm.

TABLE III

TIME ACCUMULATED ERRORS OF THE LSTM-BASED REGRESSION ALGORITHMS DESCRIBED IN (10)–(12) FOR EACH ALGORITHM

Algorithms \ Architectures	Architecture 1	Architecture 2	Architecture 3
PF	0.03590	**0.03489**	0.03600
EKF	0.03824	**0.03744**	0.03825
SGD	**0.03708**	0.03988	0.04090

Hence, they have significant differences. To compare them, we use the Hong Kong exchange rate data set as in the previous section. For a fair comparison, we again select a fixed architecture. Here, since we compare the performances of the networks rather than the algorithms, we arbitrarily choose one of the architectures. We select Architecture 1. Moreover, we choose the same parameters with the previous subsection so that convergence rates of the algorithms are the same. With this fair setup, Fig. 7(a)–(c) shows that the LSTM network-based approach achieves a smaller steady-state error; therefore, it is superior to the GRU architecture-based approach in the sequential prediction task in our experiments.

D. Different Regression Architectures

In this section, we compare the performances of different LSTM-based regression architectures. For this purpose, we use the Hong Kong exchange rate data set as in the previous section. For a fair comparison, we select the parameters such that the convergence rates of the algorithms are the same. We choose the same parameter with the previous subsection except $\mathbf{\Sigma}_{0|0} = 0.01\boldsymbol{I}$. Under this fair setup, Table III shows that for the PF- and EKF-based algorithms, Architecture 2 achieves a smaller time accumulated error thanks to the contribution of the regression vector with the control gate α_t. Due to the lack of the control and output gates, although Architecture 3 also has the direct contribution of the regression vector, it has a greater error value compared with its competitors. For the SGD-based algorithm, the direct contribution of the regression vector does not provide improvement on the error performance. Hence, Architecture 1 achieves a smaller time accumulated error. However, overall Architecture 2 trained with the PF-based algorithm achieves the smallest time accumulated error among our alternatives; hence, it significantly outperforms its competitors in these simulations.

V. CONCLUSION

We studied the nonlinear regression problem in an online setting and introduced novel LSTM-based online algorithms for data regression. We then introduced low-complexity and effective online training methods for these algorithms. We achieved these by first proposing novel regression algorithms to compute the final estimate, where we introduced an additional gate to the classical LSTM architecture. We then put the LSTM system in a state space form, and then based on this form, we derived online updates based on the SGD, EKF, and PF algorithms [17], [19], [26] to train the LSTM architecture. By this way, we obtain an effective online training method, which guarantees convergence to the optimal parameter estimation provided that we have a sufficient number of particles and satisfy certain technical conditions. We achieve this performance with a computational complexity in the order of the first-order gradient-based methods [5], [16] by controlling the number of particles. In Section IV, thanks to the generic structure of our approach, we also introduced a GRU architecture-based approach by directly replacing the LSTM equations with the GRU architecture and observed that our LSTM-based approach is superior to the GRU-based approach in the sequential prediction tasks studied in this paper. Furthermore, we demonstrate significant performance improvements achieved by the introduced algorithms with respect to the conventional methods [18], [23] over several different data sets (used in this paper).

REFERENCES

[1] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
[2] D. F. Specht, "A general regression neural network," *IEEE Trans. Neural Netw.*, vol. 2, no. 6, pp. 568–576, Nov. 1991.
[3] A. C. Singer, G. W. Wornell, and A. V. Oppenheim, "Nonlinear autoregressive modeling and estimation in the presence of noise," *Digit. Signal Process.*, vol. 4, no. 4, pp. 207–221, Oct. 1994.
[4] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space Odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published, doi: 10.1109/TNNLS.2016.2582924.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[5] A. C. Tsoi, "Gradient based learning methods," in *Adaptive Processing of Sequences and Data Structures*, C. L. 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.

[6] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," Ph.D. dissertation, Inst. Inform., Tech. Univ. Munich, München, Germany, 1991.

[7] N. D. Vanli, M. O. Sayin, I. Delibalta, and S. S. Kozat, "Sequential nonlinear learning for distributed multiagent systems via extreme learning machines," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 546–558, Mar. 2017.

[8] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608014002135

[9] U. Shaham, A. Cloninger, and R. R. Coifman, "Provable approximation properties for deep neural networks," *Appl. Comput. Harmon. Anal.*, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1063520316300033, doi: https://doi.org/10.1016/j.acha.2016.04.003

[10] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 190–198.

[11] 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.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[13] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, Oct. 2000. [Online]. Available: http://dx.doi.org/10.1162/089976600300015015

[14] J. Fan and Q. Yao, *ARMA Modeling and Forecasting*. New York, NY, USA: Springer, 2003, pp. 89–123. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-69395-8_3

[15] J. Mazumdar and R. G. Harley, "Recurrent neural networks trained with backpropagation through time algorithm to estimate nonlinear load harmonic currents," *IEEE Trans. Ind. Electron.*, vol. 55, no. 9, pp. 3484–3491, Sep. 2008.

[16] H. Jaeger, *Tutorial on Training Recurrent Neural Networks, Covering BPPT, RTRL, EKF and the Echo State Network Approach*. Sankt Augustin, Germany: GMD-Forschungszentrum Informationstechnik, 2002.

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

[18] 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, Mar. 2003.

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

[20] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, Cambridge, MA, USA, 2014, pp. 2933–2941. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969033.2969154

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

[22] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling." [Online]. Available: https://arxiv.org/abs/1412.3555

[23] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, 1989.

[24] B. C. Csáji, "Approximation with artificial neural networks," Faculty Sci., Eötvös Loránd Univ., Budapest, Hungary, Tech. Rep., 2001, vol. 24, p. 48.

[25] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proc. 28th Int. Conf. Mach. Learn. (ICML)*, 2011, pp. 1033–1040.

[26] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 174–188, Feb. 2002.

[27] Z. Li, Y. Li, F. Yu, and D. Ge, "Adaptively weighted support vector regression for financial time series prediction," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2014, pp. 3062–3065.

[28] I. Patras and E. Hancock, "Regression-based template tracking in presence of occlusions," in *Proc. 8th Int. Workshop Image Anal. Multimedia Interact. Services (WIAMIS)*, Jun. 2007, p. 15.

[29] D. M. Bates, D. G. Watts, *Nonlinear Regression Analysis and Its Applications*. New York, NY, USA: Wiley, 1988.

[30] F. A. Gers, J. A. Péerez-Ortiz, D. Eck, and J. Schmidhuber, "DEKF-LSTM," in *Proc. ESANN*, 2002, pp. 369–376.

[31] Y. C. Ho and R. Lee, "A Bayesian approach to problems in stochastic estimation and control," *IEEE Trans. Autom. Control*, vol. 9, no. 4, pp. 333–339, Oct. 1964.

[32] M. Enescu, M. Sirbu, and V. Koivunen, "Recursive estimation of noise statistics in Kalman filter based MIMO equalization," in *Proc. 27th General Assembly Int. Union Radio Sci. (URSI)*, Maastricht, The Netherlands, 2002, pp. 17–24.

[33] A. Kong, J. S. Liu, and W. H. Wong, "Sequential imputations and Bayesian missing data problems," *J. Amer. Statist. Assoc.*, vol. 89, no. 425, pp. 278–288, 1994.

[34] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statist. Comput.*, vol. 10, no. 3, pp. 197–208, Jul. 2000.

[35] N. Bergman, "Recursive bayesian estimation," Doctoral dissertation, Dept. Elect. Eng., Linköping Univ., Linköping, Sweden, 1999, vol. 579.

[36] X.-L. Hu, T. B. Schon, and L. Ljung, "A basic convergence result for particle filtering," *IEEE Trans. Signal Process.*, vol. 56, no. 4, pp. 1337–1348, Apr. 2008.

[37] C. E. Rasmussen *et al.*, *Delve Data Sets*. Accessed: Oct. 1, 2016. [Online]. Available: http://www.cs.toronto.edu/~delve/data/datasets.html

[38] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, and S. García, "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.

[39] L. Torgo. *Regression Data Sets*. Accessed: Oct. 1, 2016. [Online]. Available: http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html

[40] Alcoa Inc. *Common Stock*. Accessed: Oct. 1, 2016.[Online]. Available: http://finance.yahoo.com/quote/AA?ltr=1

[41] E. W. Frees. *Regression Modelling With Actuarial and Financial Applications*. Accessed: Oct. 1, 2016. [Online]. Available: http://instruction.bus.wisc.edu/jfrees/jfreesbooks/Regression%20Modeling/BookWebDec2010/data.html

Tolga Ergen received the B.S. degree in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2016. He is currently pursuing the M.S. degree with the Department of Electrical and Electronics Engineering, Bilkent University.

His current research interests include online learning, adaptive filtering, machine learning, optimization, and statistical signal processing.

Suleyman Serdar Kozat (A'10–M'11–SM'11) received the B.S. (Hons.) degree from Bilkent University, Ankara, Turkey, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Illinois at Urbana–Champaign, Urbana, IL, USA.

He joined the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA, as a Research Staff Member and later became a Project Leader with the Pervasive Speech Technologies Group, where he focused on problems related to statistical signal processing and machine learning. He was a Research Associate with the Cryptography and Anti-Piracy Group, Microsoft Research, Redmond, WA, USA. He is currently an Associate Professor with the Electrical and Electronics Engineering Department, Bilkent University. He has co-authored over 100 papers in refereed high impact journals and conference proceedings. He holds several patent inventions (used in several different Microsoft and IBM products) due to his research accomplishments with the IBM Thomas J. Watson Research Center and Microsoft Research. His current research interests include cyber security, anomaly detection, big data, data intelligence, adaptive filtering, and machine learning algorithms for signal processing.

Dr. Kozat received many international and national awards. He is the Elected President of the IEEE Signal Processing Society, Turkey Chapter.