Energy-Efficient LSTM Networks for Online Learning

Tolga Ergen, Ali H. Mirza, and Suleyman Serdar Kozat, *Senior Member, IEEE*

***Abstract*— We investigate variable-length data regression in an online setting and introduce an energy-efficient regression structure build on long short-term memory (LSTM) networks. For this structure, we also introduce highly effective online training algorithms. We first provide a generic LSTM-based regression structure for variable-length input sequences. To reduce the complexity of this structure, we then replace the regular multiplication operations with an energy-efficient operator, i.e., the ef-operator. To further reduce the complexity, we apply factorizations to the weight matrices in the LSTM network so that the total number of parameters to be trained is significantly reduced. We then introduce online training algorithms based on the stochastic gradient descent (SGD) and exponentiated gradient (EG) algorithms to learn the parameters of the introduced network. Thus, we obtain highly efficient and effective online learning algorithms based on the LSTM network. Thanks to our generic approach, we also provide and simulate an energy-efficient gated recurrent unit (GRU) network in our experiments. Through an extensive set of experiments, we illustrate significant performance gains and complexity reductions achieved by the introduced algorithms with respect to the conventional methods.**

***Index Terms*— ef-operator, exponentiated gradient (EG), gradient descent, long short-term memory (LSTM), matrix factorization.**

I. Introduction

A. Preliminaries

NEURAL networks are extensively studied in the literature thanks to their highly strong modeling capabilities [1]–[4]. Especially, recurrent neural networks (RNNs) are the main source of interest in these studies due to their inherent memory unit that can store time (or state) information, which boosts their capability to model time series data [5]. However, due to lacking control structures, basic RNNs might suffer from exponential growth or decay in the norm of the gradient of their parameters during training, which are also known as the exploding and vanishing gradient problems [6], [7]. Thus, basic RNNs are not usually able to capture long- and short-term dependences present in the data [6]. To address these issues, an advanced RNN architecture, i.e., the long short-term memory (LSTM) network, is introduced, which uses several information gates to regulate the information flow [8]. However, the LSTM networks have several additional nonlinear control structures (gates) and parameters, which result in complexity and training problems [8].

To this end, in this article, we investigate the efficient training of LSTM networks for data regression. In the literature, the LSTM networks are usually trained in a batch setting, where all data sequences are available and processed together for training [9], [10]. However, in big data applications, such approaches might cause storage problems due to the need to store all data sequences at one place [5], [10], [11]. Furthermore, in certain scenarios, we sequentially receive data instances, which prevents training in a batch setting [10]. Hence, we investigate efficient training of the LSTM network in an online setting, where we sequentially receive a data sequence with its label to train the parameters of the LSTM network and forget the data sequence after using it.

In the current literature, there exist several online training methods for the LSTM network [5], [11]–[14]. Among these methods, the first-order gradient-based algorithms are generally employed due to their computational efficiency in training the LSTM network [11], [12], [15], [16]. The first-order gradient-based training algorithms usually perform additive updates, i.e., each parameter is updated through an addition operation, e.g., the stochastic gradient descent (SGD) algorithm [5], [11], [12]. However, such algorithms suffer from slow convergence rate and poor performance, especially when only a few components of the input data are related to the desired label [17]. To circumvent these issues, the first-order training method with multiplicative updates, i.e., the exponentiated gradient (EG) algorithm, is introduced [17], [18]. However, since the EG algorithm employs multiplicative updates, it requires more computational resources compared with additive updates, which restricts its usage in real-life applications [17], [19], [20].

Recently, many applications require LSTMs to be implemented on embedded systems [21], [22]. However, since embedded devices have constraints, e.g., power, resource, and budget, it becomes either highly costly or impossible to employ LSTMs in real-life applications [21]–[23]. In particular, LSTMs have several parameters to train and require performing various arithmetic operations, which trigger its complexity issues. Among arithmetic operations, since the multiplication operation consumes more energy (or resources),

Manuscript received May 2, 2018; revised April 8, 2019; accepted August 12, 2019. Date of publication September 13, 2019; date of current version August 4, 2020. This work was supported in part by the Tubitak Project under Grant 117E153. *(Corresponding author: Tolga Ergen.)*

T. Ergen is with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: ergen@stanford.edu).

A. H. Mirza and S. S. Kozat are with the Department of Electrical and Electronics Engineering, Bilkent University, 06800 Ankara, Turkey (e-mail: mirza@ee.bilkent.edu.tr; kozat@ee.bilkent.edu.tr).

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

Digital Object Identifier 10.1109/TNNLS.2019.2935796

it is the decisive factor to determine the computational complexity of training LSTMs. To be more precise, [22] shows that a multiplication operation consumes more than four times the energy required by an addition operation. Thus, especially, matrix-vector multiplications in LSTMs prohibit their implementation in real-life applications such as embedded devices.

In addition to the complexity and energy issues, the performance of LSTMs might be degraded by multiplication operations. Particularly, multiplicative terms in LSTMs might cause the exploding and vanishing gradient problems [6], [7] so that the gradient-based training methods, e.g., SGD, can provide less than adequate training performance. Moreover, training algorithms that employ multiplicative updates, e.g., EG [17], [18], exacerbate such problems even further.

In order to address these issues, we introduce a novel energy-efficient LSTM network and the training methods based on the EG [17] and SGD [11] algorithms. Particularly, we first introduce an LSTM-based regression structure to process variable-length input sequences. We then introduce an energy-efficient LSTM network, which has a significantly smaller number of multiplication operations (only required for certain scaling operations) compared with the classical LSTM network. In order to further reduce the complexity, we also apply a matrix factorization method [24] to the LSTM parameters, such that the number of parameters that needs to be learned is significantly reduced. For this structure, we also introduce online training algorithms based on the EG and SGD algorithms. Thus, unlike the methods in the literature [11], [25], we not only enjoy high performance provided by the LSTM network but also achieve low computational complexity in training. Here, thanks to our generic approach, we also apply this approach to the gated recurrent unit (GRU) network [26] in our experiments. Through an extensive set of simulations, we illustrate significant performance gains and complexity reductions with respect to the conventional methods [11].

B. Prior Art and Comparisons

Various first-order gradient-based training algorithms have been introduced to train the RNN architectures in an online manner [11], [12], [25], [27]. These first-order gradient-based training algorithms usually employ additive updates in order not to exacerbate complexity issues, e.g., the SGD algorithm [11], [17]. However, the training algorithms with additive updates suffer from slow convergence and inadequate performance, specifically when the input data contains sparse information [17]. In addition, even the first-order algorithms with additive updates might suffer from high complexity while training certain complex RNN architectures, e.g., the LSTM network [5], [12]. To mitigate the complexity issues, [24] applies a matrix factorization method to the parameters of the LSTM network. Thus, they significantly reduce the total number of parameters to be learned. On the other hand, [20], [23], and [28] replace the regular multiplication operation in neural networks with an energy-efficient operator, i.e., the ef-operator. Unlike the regular multiplication operation, the ef-operator only requires sign multiplication and addition, and thus, it significantly reduces the complexity of neural networks [23], [28]. However, since both approaches employ additive updates, they provide restricted performance in certain tasks [17], [18].

To remedy the performance issues relevant to additive updates, the first-order gradient-based algorithms with multiplicative updates, e.g., the EG algorithm, are introduced [17]. Although such algorithms provide higher performance and faster convergence rate than the algorithms with additive updates, they are highly complex due to the multiplicative structure [19], [20]. As an example, Srinivasan *et al.* [25] derived the backpropagation algorithm for a simple neural network with one hidden layer using both the GD and EG algorithms. Their calculations clearly illustrate high computational complexity in the application of the EG algorithm. In order to achieve high performance provided by multiplicative updates while enjoying low computational complexity, in this article, we introduce an energy-efficient LSTM network, where we replace the regular multiplication operation with the ef-operator [19]. To further reduce the computational complexity, we also apply a matrix factorization method to the matrices in the classical LSTM architecture [24]. Thus, we significantly diminish the number of parameters to be trained in our LSTM network. We then train the introduced network with a training method based on the EG algorithm, as well as a training method based on the SGD algorithm.

C. Contributions

Our main contributions are as follows.

1) As the first time in the literature, we introduce an energy-efficient LSTM network, where we apply a matrix factorization method to reduce the computational complexity of our network. Here, we also replace each regular multiplication operation with an energy-efficient operator that only requires sign multiplication and addition to further reduce the computational complexity.
2) We introduce online training methods based on the EG and SGD algorithms to train our energy-efficient LSTM architecture, where we derive online updates for each parameter. Here, the energy-efficient LSTM network trained with our algorithms achieves substantial performance gains with respect to the classical LSTM architecture [8] trained with the conventional training methods [11].
3) We achieve these substantial performance gains with a computational complexity that is significantly less than the conventional methods in the literature [11].
4) Through an extensive set of simulations, we demonstrate significant performance improvements achieved by the introduced methods with respect to the conventional methods [11]. Moreover, since our approach is generic, we also introduce an energy-efficient GRU network in Section IV.

D. Organization of This Article

The organization of this article is as follows. We describe the variable-length online regression problem and provide our

LSTM-based structure in Section II. In Section III, we first introduce the basic energy-efficient RNN and LSTM networks using the ef-operator and then apply the matrix factorization method to these networks, where we also introduce our training methods based on the EG and SGD algorithms. In Section IV, we demonstrate the merits of the introduced energy-efficient networks and training algorithms through several experiments, where we also provide an energy-efficient GRU network. Finally, we present concluding remarks in Section V.

II. Model and Problem Description

In this article, all vectors are column vectors and denoted by boldface lowercase letters. Matrices are represented by boldface uppercase letters. For a matrix $\boldsymbol{U}$ (or a vector $\boldsymbol{u}$), $\boldsymbol{U}^T$ (or $\boldsymbol{u}^T$) is its ordinary transpose. The time index is given as subscript, e.g., $\boldsymbol{u}_t$ is the vector at time t. For a vector $\boldsymbol{u}$, $|\boldsymbol{u}|$ is the ℓ_1-norm. For a vector $\boldsymbol{u}_t$, $u_{t,i}$ is the ith element of that vector. Similarly, for a matrix $\boldsymbol{U}$, u_{ij} is the entry at the ith row and jth column of $\boldsymbol{U}$. Given a vector $\boldsymbol{u}$, $\boldsymbol{D}^{(u)}$ is the diagonal matrix with the entries of $\boldsymbol{u}$ at its diagonal.

We sequentially receive $\{d_t\}_{t\geq 1}$, $d_t \in \mathbb{R}$, and matrices $\{\boldsymbol{X}_t\}_{t\geq 1}$, i.e., defined as $\boldsymbol{X}_t = [\boldsymbol{x}_{t,1}, \boldsymbol{x}_{t,2} \ldots, \boldsymbol{x}_{t,n_t}]$, where $\boldsymbol{x}_{t,j} \in \mathbb{R}^p, \forall j \in \{1, 2, \ldots, n_t\}$, and $n_t \in \mathbb{Z}^+$ is the number of columns in $\boldsymbol{X}_t$ that may vary with respect to time t. Here, we aim to find a relation between the desired label d_t and the corresponding input vector sequence $\boldsymbol{X}_t$. To find this relation, after receiving each $\boldsymbol{X}_t$, we generate an estimate $\hat{d}_t$ based on the current and past observations. We then receive the desired value d_t and suffer the loss $L(\hat{d}_t, d_t)$ based on our estimate. This framework can be encountered in several machine learning and signal processing applications [29]. As an example, in sequential prediction under the square loss, at each time t, we receive a set of features, i.e., $\boldsymbol{X}_t$ in our case, related to the desired label d_t. We then generate the estimate through a function, i.e., $\hat{d}_t = \kappa(\boldsymbol{X}_t)$. After the desired label d_t is observed, we suffer the square loss $L(\hat{d}_t, d_t) = (d_t - \hat{d}_t)^2$.

In this article, we use RNNs to obtain $\hat{d}_t$. Since we have variable-length data sequences, we use a structure as shown in Fig. 1 to obtain fixed-length sequences. The basic RNN architecture is defined by the following equations [10]:

$$\boldsymbol{h}_{t,j} = f(\boldsymbol{W}\boldsymbol{x}_{t,j} + \boldsymbol{R}\boldsymbol{h}_{t,j-1}) \tag{1}$$

$$\boldsymbol{z}_{t,j} = g(\boldsymbol{U}\boldsymbol{h}_{t,j}) \tag{2}$$

where $\boldsymbol{x}_{t,j} \in \mathbb{R}^p$ is the input vector, $\boldsymbol{h}_{t,j} \in \mathbb{R}^m$ is the state vector, and $\boldsymbol{z}_{t,j} \in \mathbb{R}^m$ is the output vector for the jth RNN unit. Here, $f(\cdot)$ and $g(\cdot)$ usually set to the hyperbolic tangent function, and they apply to vectors pointwise. Moreover, $\boldsymbol{W}$, $\boldsymbol{R}$, and $\boldsymbol{U}$ are the parameters of the basic RNN architecture, where the sizes are chosen according to the size of the input and output vectors.

We use the LSTM network as a special variant of RNNs to obtain $\hat{d}_t$. Among different implementations of LSTM, we choose the most widely used one, i.e., the LSTM network without peephole connections [12]. Since we receive variable-length data sequence, we apply the LSTM network to each column of $\boldsymbol{X}_t$ as shown in Fig. 1, where the internal LSTM equations for the jth unit are as follows [8], [12]:

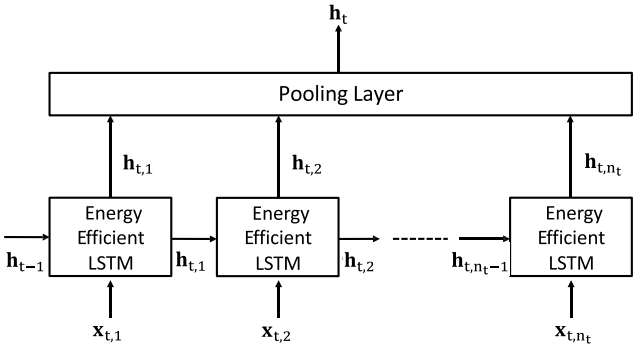

Fig. 1. Detailed schematic of energy-efficient LSTM-based architecture.

$$\tilde{\boldsymbol{c}}_{t,j} = g(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_{t,j} + \boldsymbol{R}^{(\tilde{c})}\boldsymbol{h}_{t,j-1} + \boldsymbol{b}^{(\tilde{c})}) \tag{3}$$

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

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

$$\boldsymbol{c}_{t,j} = \boldsymbol{D}^{(i)}_{t,j}\tilde{\boldsymbol{c}}_{t,j} + \boldsymbol{D}^{(f)}_{t,j}\boldsymbol{c}_{t,j-1} \tag{6}$$

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

$$\boldsymbol{h}_{t,j} = \boldsymbol{D}^{(o)}_{t,j}g(\boldsymbol{c}_{t,j}) \tag{8}$$

where $\boldsymbol{c}_{t,j} \in \mathbb{R}^m$ is the state vector, $\boldsymbol{x}_{t,j} \in \mathbb{R}^p$ is the input vector, $\boldsymbol{h}_{t,j} \in \mathbb{R}^m$ is the output vector. Here, $\boldsymbol{i}_{t,j}$, $\boldsymbol{f}_{t,j}$, and $\boldsymbol{o}_{t,j}$ are the input, forget, and output gates, respectively. The function $g(\cdot)$ applies to vectors pointwise and commonly set to $\tanh(\cdot)$. Similarly, the sigmoid function $\sigma(\cdot)$ applies to vectors pointwise. The sizes of the other matrices and vectors are determined according to the size of the input and output vectors. After the consecutive applications of the LSTM network to each column as shown in Fig. 1, we take the average of the outputs of the LSTM networks, i.e., the mean pooling method, in order to obtain a fixed-length representation, i.e., denoted as $\boldsymbol{h}_t \in \mathbb{R}^m$ at time t. Using the fixed-length vectors, we generate the final estimate as

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

where $\boldsymbol{w}_t \in \mathbb{R}^m$ represents the regression coefficients at time t. In this framework, we aim to train the system parameters, such that the total loss at time t, i.e., $\sum_{i=1}^{t} L(\hat{d}_i, d_i)$, is minimized.

For the pooling operation in Fig. 1, we use the mean pooling method to obtain the fixed-length output vectors as $\boldsymbol{h}_t = (1/n_t)\sum_{j=1}^{n_t}\boldsymbol{h}_{t,j}$. However, there are certain other pooling methods in the literature, and we can also employ them in our approach. As an example, we can apply the max and last pooling methods in our case by using $\boldsymbol{h}_t = \max_j \boldsymbol{h}_{t,j}$ and $\boldsymbol{h}_t = \boldsymbol{h}_{t,n_t}$, respectively. With such changes, our derivations can be extended to the other pooling methods.

III. Online Learning With Energy-Efficient RNN Architectures

In this section, we first apply the ef-operator to the basic RNN and LSTM architectures. We then introduce our energy-efficient RNN and LSTM architectures using the

matrix factorization method along with the ef-operator. Finally, we introduce online training algorithms based on the SGD and EG algorithms, where we provide the required updates for each parameter.

A. RNN With EF-Operator

In this section, we study a modified version of the basic RNN architecture, where we replace the regular multiplication operations with the ef-operator.

Let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^p$, and the ef-operator [19] on $\boldsymbol{a}$ and $\boldsymbol{b}$ is defined as

$$\boldsymbol{a} \diamond \boldsymbol{b} := \sum_{i=1}^{p} \text{sign}(a_i \times b_i)(|a_i| + |b_i|) \tag{10}$$

where the sign$(\cdot)$ function returns the sign of its input. Equation (10) can also be written as

$$\boldsymbol{a} \diamond \boldsymbol{b} := \sum_{i=1}^{p} \text{sign}(a_i) b_i + \text{sign}(b_i) a_i.$$

From the above-mentioned definition, it is obvious that the ef-operator only uses addition and sign multiplications, which are all energy-efficient operators.

In a similar manner, we define the ef-operator for matrix multiplications as follows:

$$(\boldsymbol{A} \diamond \boldsymbol{B})_{ij} = \boldsymbol{a}_i \diamond \boldsymbol{b}_j$$

where $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$ are the ith row of $\boldsymbol{A}$ and jth column of $\boldsymbol{B}$, respectively.

By applying the ef-operator, (1) can be written as

$$\boldsymbol{h}_{t,j} = f(\boldsymbol{a}_h \odot (\boldsymbol{W} \diamond \boldsymbol{x}_{t,j}) + \boldsymbol{b}_h \odot (\boldsymbol{R} \diamond \boldsymbol{h}_{t,j-1})) \tag{11}$$

where $\boldsymbol{a}_h \in \mathbb{R}^m$ and $\boldsymbol{b}_h \in \mathbb{R}^m$ are the scaling coefficients introduced in [23] and [28] and $\odot$ is the element-by-element multiplication of two vectors of the same size. Here, the scaling coefficients are the crucial factors to match the performance of the classical multiplicative networks. In particular, even though we eliminate several matrix-vector multiplications, which are one of the main pillars of RNNs, these coefficients keep the modeling capabilities of the network at the same level by introducing an additional multiplicative term before the nonlinear function is applied. In addition, note that the weight matrices, i.e., $\boldsymbol{W}$ and $\boldsymbol{R}$, in (1) and (11) are not necessarily the same, however, their function is the same, i.e., both are weight matrices that multiply the input vector. In (11), $\boldsymbol{W} \diamond \boldsymbol{x}_{t,j}$ and $\boldsymbol{R} \diamond \boldsymbol{h}_{t,j-1}$ are given as follows:

$$\boldsymbol{W} \diamond \boldsymbol{x}_{t.j} = [\boldsymbol{w}_1 \diamond \boldsymbol{x}_{t,j} \;\; \boldsymbol{w}_2 \diamond \boldsymbol{x}_{t,j} \;\; \dots \;\; \boldsymbol{w}_m \diamond \boldsymbol{x}_{t,j}]^T$$

where $\boldsymbol{w}_i$ represents the transpose of the ith row of $\boldsymbol{W}$ and $\boldsymbol{w}_i \diamond \boldsymbol{x}_{t,j}$ is given as

$$\boldsymbol{w}_i \diamond \boldsymbol{x}_{t,j} = \sum_{k=1}^{p} \text{sign}(x_{t,jk}) w_{ik} + \text{sign}(w_{ik}) x_{t,jk}$$

where $x_{t,jk}$ is the kth element of $\boldsymbol{x}_{t,j}$. Similarly

$$\boldsymbol{R} \diamond \boldsymbol{h}_{t,j-1} = [\boldsymbol{r}_1 \diamond \boldsymbol{h}_{t,j-1} \;\; \boldsymbol{r}_2 \diamond \boldsymbol{h}_{t,j-1} \dots \boldsymbol{r}_m \diamond \boldsymbol{h}_{t,j-1}]^T$$

where $\boldsymbol{r}_i$ represents the transpose of the ith row of $\boldsymbol{R}$ and $\boldsymbol{r}_i \diamond \boldsymbol{h}_{t,j}$ is given as

$$\boldsymbol{r}_i \diamond \boldsymbol{h}_{t,j} = \sum_{k=1}^{m} \text{sign}(h_{t,jk}) r_{ik} + \text{sign}(r_{ik}) h_{t,jk}$$

where $h_{t,jk}$ is the kth element of $\boldsymbol{h}_{t,j}$. Likewise, (2) is modified as follows:

$$\boldsymbol{z}_{t,j} = g(\boldsymbol{b}_z \odot (\boldsymbol{U} \diamond \boldsymbol{h}_{t,j})) \tag{12}$$

where $\boldsymbol{b}_z$ is the scaling coefficient.

B. LSTM With EF-Operator

In this section, we replace the regular multiplication operators in the classical LSTM architecture with the ef-operator, as illustrated in Fig. 2. Based on this modification, (3)–(8) are written as follows:

$$\tilde{\boldsymbol{c}}_{t,j} = g(\boldsymbol{a}_{\tilde{c}} \odot (\boldsymbol{W}^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j}) + \boldsymbol{b}_{\tilde{c}} \odot (\boldsymbol{R}^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j-1}) + \boldsymbol{b}^{(\tilde{c})}) \tag{13}$$

$$\boldsymbol{i}_{t,j} = \sigma(\boldsymbol{a}_i \odot (\boldsymbol{W}^{(i)} \diamond \boldsymbol{x}_{t,j}) + \boldsymbol{b}_i \odot (\boldsymbol{R}^{(i)} \diamond \boldsymbol{h}_{t,j-1}) + \boldsymbol{b}^{(i)}) \tag{14}$$

$$\boldsymbol{f}_{t,j} = \sigma(\boldsymbol{a}_f \odot (\boldsymbol{W}^{(f)} \diamond \boldsymbol{x}_{t,j}) + \boldsymbol{b}_f \odot (\boldsymbol{R}^{(f)} \diamond \boldsymbol{h}_{t,j-1}) + \boldsymbol{b}^{(f)}) \tag{15}$$

$$\boldsymbol{c}_{t,j} = \boldsymbol{i}_{t,j} \blacklozenge \tilde{\boldsymbol{c}}_{t,j} + \boldsymbol{f}_{t,j} \blacklozenge \boldsymbol{c}_{t,j-1} \tag{16}$$

$$\boldsymbol{o}_{t,j} = \sigma(\boldsymbol{a}_o \odot (\boldsymbol{W}^{(o)} \diamond \boldsymbol{x}_{t,j}) + \boldsymbol{b}_o \odot (\boldsymbol{R}^{(o)} \diamond \boldsymbol{h}_{t,j-1}) + \boldsymbol{b}^{(o)}) \tag{17}$$

$$\boldsymbol{h}_{t,j} = \boldsymbol{o}_{t,j} \blacklozenge g(\boldsymbol{c}_{t.j}) \tag{18}$$

where $\boldsymbol{a}_{(\cdot)}, \boldsymbol{b}_{(\cdot)} \in \mathbb{R}^m$ are the scaling coefficients and the $\blacklozenge$ operation is defined as follows:

$$\begin{aligned} \boldsymbol{a} \blacklozenge \boldsymbol{b} &:= [a_1 \diamond b_1 \quad a_2 \diamond b_2 \dots a_p \diamond b_p]^T \\ &:= \text{sign}(\boldsymbol{a}) \odot \boldsymbol{b} + \text{sign}(\boldsymbol{b}) \odot \boldsymbol{a}. \end{aligned}$$

In (13), $\boldsymbol{W}^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j}$ is written as

$$\boldsymbol{W}^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j} = \left[\boldsymbol{w}_1^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j} \quad \boldsymbol{w}_2^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j} \dots \boldsymbol{w}_m^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j}\right]^T \tag{19}$$

where $\boldsymbol{w}_i^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j}$ is given as

$$\boldsymbol{w}_i^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j} = \sum_{k=1}^{p} \text{sign}(x_{t,jk}) w_{ik}^{(\tilde{c})} + \text{sign}(w_{ik}^{(\tilde{c})}) x_{t,jk}$$

and

$$\boldsymbol{R}^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j-1} = \left[\boldsymbol{r}_1^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j-1} \quad \boldsymbol{r}_2^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j-1} \dots \boldsymbol{r}_m^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j-1}\right]^T \tag{20}$$

where $\boldsymbol{r}_i^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j}$ is given as

$$\boldsymbol{r}_i^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j} = \sum_{k=1}^{m} \text{sign}(h_{t,jk}) r_{ik}^{(\tilde{c})} + \text{sign}\left(r_{ik}^{(\tilde{c})}\right) h_{t,jk}$$

where $\boldsymbol{w}_i^{(\tilde{c})}$ and $\boldsymbol{r}_i^{(\tilde{c})}$ are the ith row of $\boldsymbol{W}^{(\tilde{c})}$ and $\boldsymbol{R}^{(\tilde{c})}$, respectively.

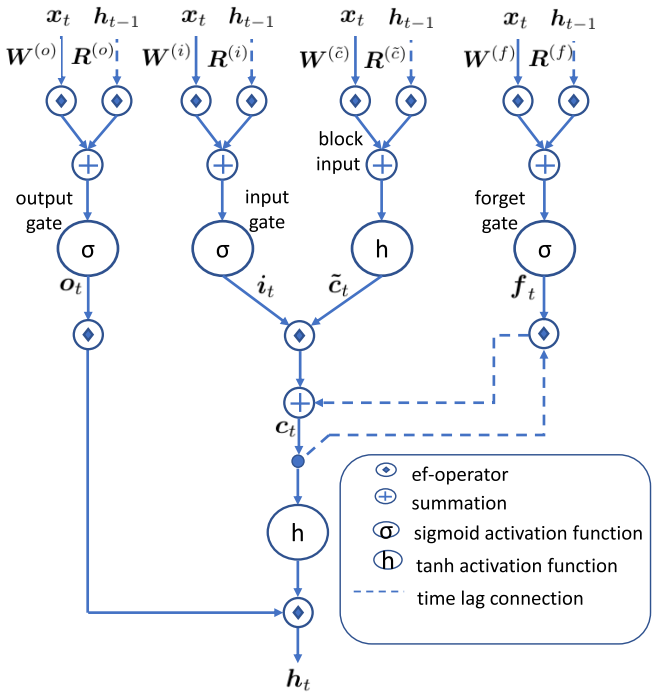

Fig. 2. Detailed schematic of energy-efficient LSTM block at time t. Note that the solid lines represent the direct connections, while the dotted lines represent the time-lagged connections. For the sake of simplicity, bias terms are not shown in the figure.

For the other multiplications, we change the parameters in either (19) or (20) according to the chosen coefficient matrix. Other than that, we follow the same procedures in (19) and (20).

Remark 1: Compared with the original LSTM network in (3)–(8), we convert $4m(m+p)+3m$ regular multiplication operations into sign multiplication and addition operations thanks to the ef-operator. However, due to the scaling factors introduced in (13)–(15) and (17), we have $8m$ additional regular multiplications. Overall, since for large m and p values $8m \ll 4m(m+p)+3m$, we significantly reduce the number of regular multiplications. Thus, we provide a substantial decrease in the computational complexity and energy consumption compared with the classical LSTM network.

C. Energy-Efficient RNN With Weight Matrix Factorization

In this section, we apply the matrix factorization method [24] to the weight matrices of the basic RNN architecture in order to reduce the number of parameters to be trained.

We first factorize the weight matrices in (11) as $\boldsymbol{W} \approx \boldsymbol{M}\boldsymbol{N}$ and $\boldsymbol{R} \approx \boldsymbol{P}\boldsymbol{Q}$, where $\boldsymbol{W} \in \mathbb{R}^{m\times p}$, $\boldsymbol{M} \in \mathbb{R}^{m\times d}$, $\boldsymbol{N} \in \mathbb{R}^{d\times p}$, $\boldsymbol{R} \in \mathbb{R}^{m\times m}$, $\boldsymbol{P} \in \mathbb{R}^{m\times f}$, and $\boldsymbol{Q} \in \mathbb{R}^{f\times m}$.

Remark 2: We factorize the RNN weight matrices into two smaller matrices. The rank of these two smaller matrices is selected, such that $d, f \ll \min(m, p)$. Thus, we significantly reduce the number of parameters that needs to be learned, e.g., $\boldsymbol{W}$ has mp entries, while $\boldsymbol{M}$ and $\boldsymbol{N}$ have $d(m+p) \ll mp$.

The energy-efficient RNN with weight matrix factorization can be written as

$$\boldsymbol{h}_{t,j} = f(\boldsymbol{a}_h \odot (\boldsymbol{M} \diamond \boldsymbol{N} \diamond \boldsymbol{x}_{t,j}) + \boldsymbol{b}_h \odot (\boldsymbol{P} \diamond \boldsymbol{Q} \diamond \boldsymbol{h}_{t,j-1})). \tag{21}$$

In (21), $\boldsymbol{M} \diamond \boldsymbol{N} \diamond \boldsymbol{x}_{t,j}$ is given as follows:

$$\boldsymbol{M} \diamond \boldsymbol{N} \diamond \boldsymbol{x}_{t,j} = [\boldsymbol{\mu}_1 \diamond \boldsymbol{x}_{t,j} \quad \boldsymbol{\mu}_2 \diamond \boldsymbol{x}_{t,j} \ldots \boldsymbol{\mu}_m \diamond \boldsymbol{x}_{t,j}]^T$$

where $\boldsymbol{\mu}_i \in \mathbb{R}^p$ is the ith row of $\boldsymbol{M} \diamond \boldsymbol{N}$ and $\boldsymbol{\mu}_i \diamond \boldsymbol{x}_{t,j}$ is given as

$$\boldsymbol{\mu}_i \diamond \boldsymbol{x}_{t,j} = \sum_{k=1}^{p} \text{sign}(x_{t,jk})\mu_{ik} + \text{sign}(\mu_{ik})x_{t,jk} \tag{22}$$

and

$$\boldsymbol{P} \diamond \boldsymbol{Q} \diamond \boldsymbol{h}_{t,j-1} = [\boldsymbol{v}_1 \diamond \boldsymbol{h}_{t,j-1} \quad \boldsymbol{v}_2 \diamond \boldsymbol{h}_{t,j-1} \ldots \boldsymbol{v}_m \diamond \boldsymbol{h}_{t,j-1}]^T$$

where $\boldsymbol{v}_i \in \mathbb{R}^m$ is the ith row of $\boldsymbol{P} \diamond \boldsymbol{Q}$ and $\boldsymbol{v}_i \diamond \boldsymbol{h}_{t,j}$ is given as

$$\boldsymbol{v}_i \diamond \boldsymbol{h}_{t,j} = \sum_{k=1}^{m} \text{sign}(h_{t,jk})v_{ik} + \text{sign}(v_{ik})h_{t,jk}. \tag{23}$$

In a similar manner, (12) is modified as follows:

$$\boldsymbol{z}_{t,j} = g(\boldsymbol{b}_z \odot (\boldsymbol{S} \diamond \boldsymbol{T} \diamond \boldsymbol{h}_{t,j})) \tag{24}$$

where we factorize the $\boldsymbol{U}$ matrix as $\boldsymbol{U} \approx \boldsymbol{S}\boldsymbol{T}$ so that the number of columns in $\boldsymbol{S}$ (or the number of rows in $\boldsymbol{T}$) is significantly smaller than the number of rows in $\boldsymbol{S}$ (or the number of columns in $\boldsymbol{T}$).

D. Energy-Efficient LSTM With Weight Matrix Factorization

In this section, we apply the matrix factorization method [24] to the weight matrices of the LSTM network to diminish the number of parameters in the network.

We factorize the LSTM neural network weight matrices into two sub-matrices of lower rank as $\boldsymbol{W}^{(\cdot)} \approx \boldsymbol{M}^{(\cdot)}\boldsymbol{N}^{(\cdot)}$ and $\boldsymbol{R}^{(\cdot)} \approx \boldsymbol{P}^{(\cdot)}\boldsymbol{Q}^{(\cdot)}$, where $\boldsymbol{W}^{(\cdot)} \in \mathbb{R}^{m\times p}$, $\boldsymbol{M}^{(\cdot)} \in \mathbb{R}^{m\times d}$, $\boldsymbol{N}^{(\cdot)} \in \mathbb{R}^{d\times p}$, $\boldsymbol{R}^{(\cdot)} \in \mathbb{R}^{m\times m}$, $\boldsymbol{P}^{(\cdot)} \in \mathbb{R}^{m\times f}$, and $\boldsymbol{Q}^{(\cdot)} \in \mathbb{R}^{f\times m}$, such that $d, f \ll \min(p, m)$. We then apply this factorization to the LSTM neural network in (13)–(18) by replacing the weight matrices with their factorized forms. The modifications for the jth LSTM unit in Fig. 1 are as follows.

Here, $\boldsymbol{M}^{(\tilde{c})} \diamond \boldsymbol{N}^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j}$ is given as follows:

$$\boldsymbol{M}^{(\tilde{c})} \diamond \boldsymbol{N}^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j} = \left[\boldsymbol{\mu}_1^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j} \quad \boldsymbol{\mu}_2^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j} \ldots \boldsymbol{\mu}_m^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j}\right]^T \tag{25}$$

where $\boldsymbol{\mu}_i^{(\tilde{c})} \in \mathbb{R}^p$ is the ith row of $\boldsymbol{M}^{(\tilde{c})} \diamond \boldsymbol{N}^{(\tilde{c})}$ and $\boldsymbol{\mu}_i^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j}$ is given as

$$\boldsymbol{\mu}_i^{(\tilde{c})} \diamond \boldsymbol{x}_{t,j} = \sum_{k=1}^{p} \text{sign}(x_{t,jk})\mu_{ik}^{(\tilde{c})} + \text{sign}\left(\mu_{ik}^{(\tilde{c})}\right)x_{t,jk}$$

and

$$\boldsymbol{P}^{(\tilde{c})} \diamond \boldsymbol{Q}^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j-1} = \left[\boldsymbol{v}_1^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j-1} \quad \boldsymbol{v}_2^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j-1} \ldots \boldsymbol{v}_m^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j-1}\right]^T \tag{26}$$

where $\boldsymbol{v}_i^{(\tilde{c})} \in \mathbb{R}^m$ is the ith row of $\boldsymbol{P}^{(\tilde{c})} \diamond \boldsymbol{Q}^{(\tilde{c})}$ and $\boldsymbol{v}_i^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j}$ is given as

$$\boldsymbol{v}_i^{(\tilde{c})} \diamond \boldsymbol{h}_{t,j} = \sum_{k=1}^{m} \text{sign}(h_{t,jk})v_{ik}^{(\tilde{c})} + \text{sign}(v_{ik}^{(\tilde{c})})h_{t,jk}.$$

For the other weight matrices, we replace the factorized form of the chosen weight matrix in (25) and (26). Then, we follow the same operations in (25) and (26).

Remark 3: We reduce the total number of LSTM network parameters by applying weight matrix factorization. In the original LSTM equations in (3)–(8), we have $4m(m+p)$ scalar parameters in the weight matrices, i.e., $\boldsymbol{W}^{(\cdot)}$ and $\boldsymbol{R}^{(\cdot)}$. However, in our energy-efficient LSTM network, we have $4d(m+p)+8mf$, which is significantly less than $4m(m+p)$, provided that $d,\ f \ll \min(m,\ p)$.

E. Online Training Algorithms

In this section, we derive the online updates to train the parameters of the introduced energy-efficient networks. We first derive the online updates based on the SGD algorithm. We then derive the online updates based on the EG algorithm.

We first employ the SGD algorithm [11] to obtain the online updates for each parameter. For $\boldsymbol{w}_t$, the SGD update is computed as follows:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \nabla_{\boldsymbol{w}_t} L \tag{27}$$

where $\nabla_{\boldsymbol{w}_t}$ represents the gradient of a certain function with respect to $\boldsymbol{w}_t$ and η_t is the learning rate. Here, L is the instantaneous loss, i.e., the squared error $(d_t - \hat{d}_t)^2$, and we denote it as L rather than $L(\hat{d}_t, d_t)$ for notational simplicity. Note that SGD updates, e.g., (27), are additive updates since the gradient information is being added at each time step.

On the other hand, for $\boldsymbol{w}_t$, the EG update [17] is computed as follows:

$$w_{t+1,i} = \frac{w_{t,i} r_{t,i}}{\sum_{j=1}^{m} w_{t,j} r_{t,j}} \tag{28}$$

where

$$r_{t,i} = \exp\big(-\eta_t\ L'_{w_{t,i}}\big)$$

and $w_{t,i}$ is the ith component of $\boldsymbol{w}_t$ and $L'_{w_{t,i}}$ is the partial derivative of the instantaneous loss function with respect to $w_{t,i}$. As seen in (28), EG updates are multiplicative updates since the gradient information is encapsulated in the exponent part and multiplied at each time step. In order to eliminate the multiplication and exponentiation in (28), we use the first-order Taylor series expansion along with the ef-operator as follows:

$$w_{t+1,i} = \frac{w_{t,i} \diamond \hat{r}_{t,i}}{\sum_{j=1}^{m} (w_{t,j} \diamond \hat{r}_{t,j})} \tag{29}$$

where

$$\hat{r}_{t,i} = 1 - \eta_t\ L'_{w_{t,i}}.$$

Note that since the division in (29) is the same for all possible i values, it is just a scaling factor in the implementation of the algorithm. Thus, this operation does not require high amount of energy and computational resources unlike the regular multiplication operation.

Remark 4: Since the weight vector $\boldsymbol{w}_t$ might contain negative components, we use the slightly modified version of the original EG algorithms, i.e., the $\text{EG}^{\pm}$ algorithm [17], which uses a weight vector $\boldsymbol{w}_t^+ - \boldsymbol{w}_t^-$. In this algorithm, the weight vector is updated as follows:

$$w^+_{t+1,i} = \frac{w^+_{t,i} \diamond \hat{r}^+_{t,i}}{\sum_{j=1}^{m} \left(w^+_{t,j} \diamond \hat{r}^+_{t,j} + w^-_{t,j} \diamond \hat{r}^-_{t,j}\right)}$$

$$w^-_{t+1,i} = \frac{w^-_{t,i} \diamond \hat{r}^-_{t,i}}{\sum_{j=1}^{m} \left(w^+_{t,j} \diamond \hat{r}^+_{t,j} + w^-_{t,j} \diamond \hat{r}^-_{t,j}\right)}$$

where

$$\hat{r}^+_{t,i} = 1 - \eta_t\ L'_{w^+_{t,i}}$$

$$\hat{r}^-_{t,i} = 1 + \eta_t\ L'_{w^-_{t,i}}.$$

In Sections III-E1 and III-E2, we derive the updates for the parameters of the proposed energy-efficient RNN and LSTM networks.

1) Online Training of Energy-Efficient RNN: We compute the first-order gradient of the loss function with respect to each parameter in order to perform SGD and EG updates.

In the basic RNN architecture, we have $z_t = \sum_{j=1}^{n_t} z_{t,j}/n_t$ as the output at time t. Although our structure in Fig. 1 is generic in the sense that it can process variable-length data sequences, here, we only derive the equations for $n_t = 1$ for notational and presentation simplicity. However, at the end of this section, we also provide the required extensions to obtain the equations for generic n_t values. With this modification, we have $\boldsymbol{z}_t = \boldsymbol{z}_{t,1}$, and thus, we generate the estimate as $\hat{d}_t = \boldsymbol{w}_t^T \boldsymbol{z}_{t,1}$.

Under the square loss, we compute the first-order derivative of the loss function with respect to b_{zi}, i.e., the ith element of $\boldsymbol{b}_z$, as follows:

$$\frac{\partial L}{\partial b_{zi}} = \frac{\partial L}{\partial \hat{d}_t} \frac{\partial \hat{d}_t}{\partial \boldsymbol{z}_{t,1}} \frac{\partial \boldsymbol{z}_{t,1}}{\partial b_{zi}} = -2(d_t - \hat{d}_t)\boldsymbol{w}_t^T \left[g'(\boldsymbol{\varphi}_t) \odot \left((\boldsymbol{u}_i \diamond \boldsymbol{h}_{t,1})\boldsymbol{e_i} + \boldsymbol{b}_z \odot \boldsymbol{\lambda}_t^{(Uh)}\right)\right] \tag{30}$$

where g' is the derivative of $g(\cdot)$ with respect to its argument, $\boldsymbol{e}_i$ is a vector of zeros except a 1 at the ith index, and

$$\boldsymbol{\varphi}_t = \boldsymbol{b}_z \odot (\boldsymbol{U} \diamond \boldsymbol{h}_{t,1}).$$

Moreover, for the ith element of $\boldsymbol{\lambda}_t^{(Uh)} = \partial(\boldsymbol{U} \diamond \boldsymbol{h}_{t,1})/\partial b_{zi}$, we use the following formula in (30):

$$\lambda_{t,i}^{(Uh)} = \sum_{j=1}^{m} \left(u_{ij} 2\delta(h_{t,1j}) \gamma_{t,ij}^{(b_z)} + \text{sign}(u_{ij}) \gamma_{t,ij}^{(b_z)}\right) \tag{31}$$

where $\delta(\cdot)$ is the dirac delta function, we compute the derivative of the sign function as $d(\text{sign}(x))/dx = 2\delta(x)$ [30] and

$$\gamma_{t,ij}^{(b_z)} = \frac{\partial h_{t,1j}}{\partial b_{zi}} = f_j'(\boldsymbol{\theta}_t) b_{hj} \lambda_{t-1,j}^{(Rh)} \tag{32}$$

where f_i' is the derivative of the ith element of $f(\cdot)$ with respect to its argument and

$$\boldsymbol{\theta}_t = \boldsymbol{a}_h \odot (\boldsymbol{W} \diamond \boldsymbol{x}_{t,1}) + \boldsymbol{b}_h \odot (\boldsymbol{R} \diamond \boldsymbol{h}_{t-1,1}).$$

Similarly, we have the following derivative for u_{ik}:

$$\frac{\partial L}{\partial u_{ik}} = \frac{\partial L}{\partial \hat{d}_t}\frac{\partial \hat{d}_t}{\partial z_{t,1}}\frac{\partial z_{t,1}}{\partial u_{ik}} = -2(d_t - \hat{d}_t)\boldsymbol{w}_t^T\big[g'(\boldsymbol{\varphi}_t) \odot \big(b_{zi}(\text{sign}(h_{t,1k}) + 2\delta(u_{ik})h_{t,1k})\boldsymbol{e_i} + \boldsymbol{b}_z \odot \boldsymbol{\alpha}_t^{(Uh)}\big)\big] \tag{33}$$

where

$$\alpha_{t,i}^{(Uh)} = \sum_{j=1}^{m}\big(u_{ij}2\delta(h_{t,1j})\gamma_{t,ij}^{(u_{ik})} + \text{sign}(u_{ij})\gamma_{t,ij}^{(u_{ik})}\big) \tag{34}$$

and

$$\gamma_{t,ij}^{(u_{ik})} = \frac{\partial h_{t,1j}}{\partial u_{ik}} = f_j'(\boldsymbol{\theta}_t)b_{hj}\alpha_{t-1,j}^{(Rh)}.$$

Remark 5: When we take the derivative of L with respect to the other parameters, the position of the term with $\boldsymbol{e}_i$ changes. As an example, for the derivative of L with respect to b_{hi}, the $\boldsymbol{r}_i \diamond \boldsymbol{h}_{t-1,1}$ term appears in (32) when $j = i$; otherwise, (32) does not change. If we write (32) in a vector form, the contribution of $\boldsymbol{r}_i \diamond h_{t-1,1}$ can be written as $(\boldsymbol{R} \diamond \boldsymbol{h}_{t-1,1})\boldsymbol{e}_i$. As seen in this case, for the other derivatives, the position and the form of the term with $\boldsymbol{e}_i$ slightly change; other than that, we follow the same procedure in (30)–(34).

Remark 6: For the other n_t values, i.e., $n_t \neq 1$, the recursion in (32) is performed through the outputs of the different RNN blocks at a certain time t, as shown in Fig. 1. Thus, rather than having $t-1$ in (32), we have multiple recursions based on another index at time t. Besides this slight change, all of our derivations hold for generic n_t values.

Remark 7: For the energy-efficient RNN architecture with weight matrix factorization, we take the derivative of the loss function with respect to the parameters of each factorized matrix. As an example, in (33), instead of only taking the derivative with respect to the parameters of $\boldsymbol{U}$, we compute the derivatives of the loss with respect to the entries of both $\boldsymbol{S}$ and $\boldsymbol{T}$, i.e., the factorized versions of $\boldsymbol{U}$. Other than such changes, we follow the same procedures in (30)–(34).

With the derived gradients, we can update each parameter of the basic RNN architecture as in (27) and (29).

2) Online Training of Energy-Efficient LSTM: Here, we derive the first-order gradient of the loss function with respect to each LSTM parameter to obtain the online updates based on the SGD and EG algorithms. We again derive the derivatives for the $n_t = 1$ case for notational and presentation simplicity. However, at the end of this section, we also provide the required extensions to obtain the equations for generic n_t values. With this modification, we have $\boldsymbol{h}_t = \boldsymbol{h}_{t,1}$, and hence, we generate the estimate $\hat{d}_t = \boldsymbol{w}_t^T\boldsymbol{h}_{t,1}$.

We first compute the derivative of L with respect to $w_{ij}^{(\tilde{c})}$, i.e., the element at the ith row and the jth column of $\boldsymbol{W}^{(\tilde{c})}$, as follows:

$$\frac{\partial L}{\partial w_{ij}^{(\tilde{c})}} = \frac{\partial L}{\partial \hat{d}_t}\frac{\partial \hat{d}_t}{\partial \boldsymbol{h}_{t,1}}\frac{\partial \boldsymbol{h}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} = -2(d_t - \hat{d}_t)\boldsymbol{w}_t^T\frac{\partial(\boldsymbol{o}_{t,1} \blacklozenge g(\boldsymbol{c}_{t,1}))}{\partial w_{ij}^{(\tilde{c})}}. \tag{35}$$

In (35), we calculate the partial derivative as

$$\begin{aligned}\frac{\partial(\boldsymbol{o}_{t,1} \blacklozenge g(\boldsymbol{c}_{t,1}))}{\partial w_{ij}^{(\tilde{c})}} &= \frac{\partial \boldsymbol{o}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \odot \text{sign}(g(\boldsymbol{c}_{t,1})) \\ &\quad + \boldsymbol{o}_{t,1} \odot 2\delta(g(\boldsymbol{c}_{t,1})) \odot g'(\boldsymbol{c}_{t,1}) \odot \frac{\partial \boldsymbol{c}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \\ &\quad + 2\delta(\boldsymbol{o}_{t,1}) \odot \frac{\partial \boldsymbol{o}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \odot g(\boldsymbol{c}_{t,1}) \\ &\quad + \text{sign}(\boldsymbol{o}_{t,1}) \odot g'(\boldsymbol{c}_{t,1}) \odot \frac{\partial \boldsymbol{c}_{t,1}}{\partial w_{ij}^{(\tilde{c})}}.\end{aligned} \tag{36}$$

For (36), we now compute the derivatives of $\boldsymbol{o}_{t,1}$ and $\boldsymbol{c}_{t,1}$ with respect to $w_{ij}^{(\tilde{c})}$. With $\boldsymbol{\lambda}_{t-1}^{(R^{(o)}h)} = \partial(\boldsymbol{R}^{(o)} \diamond \boldsymbol{h}_{t-1,1})/\partial w_{ij}^{(\tilde{c})}$ as in (31), the derivative of (17) is as follows:

$$\frac{\partial \boldsymbol{o}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} = \boldsymbol{D}_{t,1}^{(\sigma'(\zeta^{(o)}))}\big(\boldsymbol{b}_o \odot \boldsymbol{\lambda}_{t-1}^{(R^{(o)}h)}\big) \tag{37}$$

where

$$\boldsymbol{\zeta}_{t,1}^{(o)} = \boldsymbol{a}_o \odot (\boldsymbol{W}^{(o)} \diamond \boldsymbol{x}_{t,1}) + \boldsymbol{b}_o \odot (\boldsymbol{R}^{(o)} \diamond \boldsymbol{h}_{t-1,1}) + \boldsymbol{b}^{(o)}. \tag{38}$$

In order to calculate (36), we also compute the derivative of $\boldsymbol{c}_{t,1}$ with respect to $w_{ij}^{(\tilde{c})}$. For this derivative, we obtain the following recursive relation from (16):

$$\begin{aligned}\frac{\partial \boldsymbol{c}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} &= \text{sign}(\tilde{\boldsymbol{c}}_{t,1}) \odot \frac{\partial \boldsymbol{i}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} + 2\delta(\tilde{\boldsymbol{c}}_{t,1}) \odot \frac{\partial \tilde{\boldsymbol{c}}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \odot \boldsymbol{i}_{t,1} \\ &\quad + \text{sign}(\boldsymbol{i}_{t,1}) \odot \frac{\partial \tilde{\boldsymbol{c}}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} + 2\delta(\boldsymbol{i}_{t,1}) \odot \frac{\partial \boldsymbol{i}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \odot \tilde{\boldsymbol{c}}_{t,1} \\ &\quad + \text{sign}(\boldsymbol{c}_{t-1,1}) \odot \frac{\partial \boldsymbol{f}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} + 2\delta(\boldsymbol{c}_{t-1,1}) \odot \frac{\partial \boldsymbol{c}_{t-1,1}}{\partial w_{ij}^{(\tilde{c})}} \odot \boldsymbol{f}_{t,1} \\ &\quad + \text{sign}(\boldsymbol{f}_{t,1}) \odot \frac{\partial \boldsymbol{c}_{t-1,1}}{\partial w_{ij}^{(\tilde{c})}} + 2\delta(\boldsymbol{f}_{t,1}) \odot \frac{\partial \boldsymbol{f}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \odot \boldsymbol{c}_{t-1,1}.\end{aligned} \tag{39}$$

For (39), we compute the derivatives of (13)–(15) with respect to $w_{ij}^{(\tilde{c})}$ as follows:

$$\frac{\partial \boldsymbol{i}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} = \boldsymbol{D}_{t,1}^{(\sigma'(\zeta^{(i)}))}\big(\boldsymbol{b}_i \odot \boldsymbol{\lambda}_{t-1}^{(R^{(i)}h)}\big) \tag{40}$$

$$\frac{\partial \boldsymbol{f}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} = \boldsymbol{D}_{t,1}^{(\sigma'(\zeta^{(f)}))}\big(\boldsymbol{b}_f \odot \boldsymbol{\lambda}_{t-1}^{(R^{(f)}h)}\big) \tag{41}$$

$$\frac{\partial \tilde{\boldsymbol{c}}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} = \boldsymbol{D}_{t,1}^{(g'(\zeta^{(\tilde{c})}))}\big(\big(\text{sign}(x_{t,1j}) + 2\delta\big(w_{ij}^{(\tilde{c})}\big)x_{t,1j}\big)\boldsymbol{e_i} + \boldsymbol{b}_{\tilde{c}} \odot \boldsymbol{\lambda}_{t-1}^{(R^{(\tilde{c})}h)}\big). \tag{42}$$

Using (40)–(42), we compute (39). Then, we compute (36) using (39) and (37) in order to calculate (35). After obtaining (35), we update the parameter using the SGD- and EG-based algorithms as in (27) and (29).

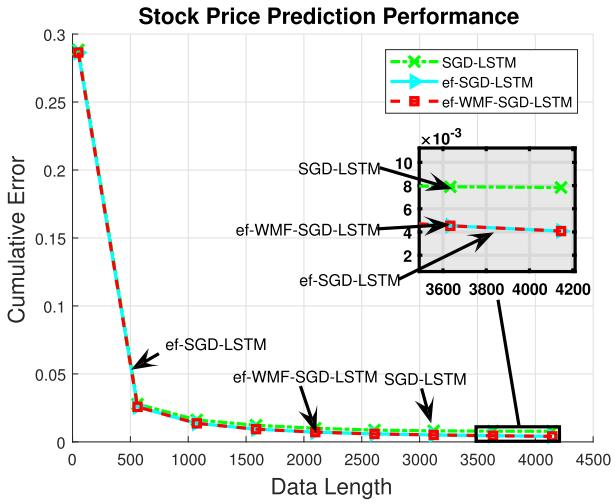

Fig. 3. Daily stock price prediction performances of the algorithms with the SGD updates on the Alcoa Corporation stock price data set.

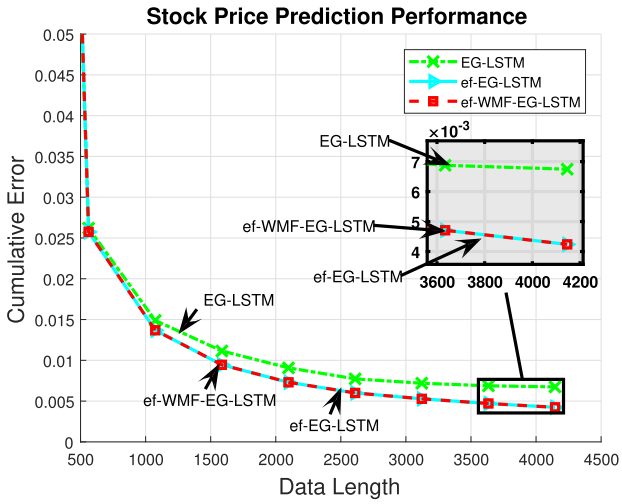

Fig. 4. Daily stock price prediction performances of the algorithms with the EG updates on the Alcoa Corporation stock price data set.

As in Remark 5, when we take the derivative with respect to the other parameters, only the location of the term with $\boldsymbol{e}_i$ changes. Similar to the RNN case, when $n_t \neq 1$, the recursion in (39) is performed through the outputs of the different LSTM blocks at a certain time t, as shown in Fig. 1. Moreover, for the factorized LSTM network, we compute the derivatives of the loss function with respect to each factorized matrix parameter as in Remark 7.

With the derived gradients, we can update each parameter of the energy-efficient LSTM architecture as in (27) and (29).

IV. Simulations

In this section, we illustrate the performances of our algorithms on various data sets under different scenarios. We first compare the regression performances of our algorithms on a financial data set, i.e., the Alcoa Corporation stock price rate data set [31]. We then evaluate the regression performances on the real-life data sets, i.e., the kinematic [32] and elevators [33] data sets. Since our approach is generic, we also compare the performances of our training algorithms on two different RNNs, i.e., the LSTM and GRU neural networks. We then compare the structural complexity of our energy-efficient algorithms with the conventional structures.

Throughout this section, "Model 1" represents the conventional LSTM network (LSTM). Similarly, "Model 2" represents the introduced LSTM network with the ef-operator (ef-LSTM), and "Model 3" represents the introduced LSTM network with the ef-operator and weight matrix factorization (ef-WMF-LSTM).

A. Financial Data Set

In this section, we compare the performances of our algorithms on a financial data set. We consider the Alcoa Corporation stock price data set [31], for which we have the daily stock price values. In this case, our aim is to predict future stock prices based on the past prices, where we examine the past five days for prediction. Here, we evaluate the regression

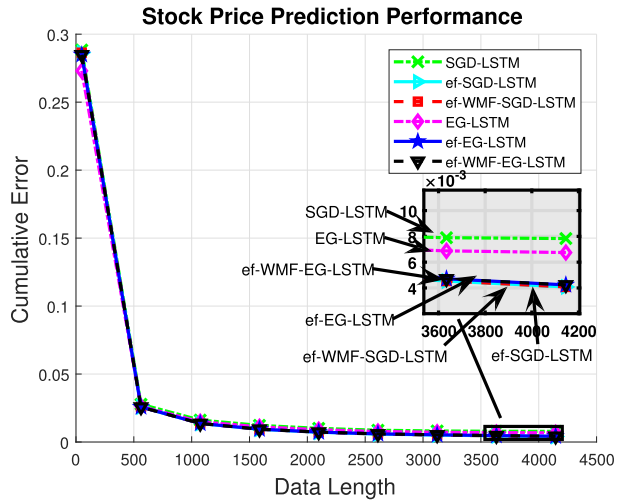

Fig. 5. Comparison of all the algorithms with the SGD and EG updates on the Alcoa Corporation stock price data set.

performance of Model 1 and consider this performance to be a benchmark for our proposed models, i.e., Model 2 and Model 3. To provide a fair setup, we select the same values for the common parameters of all the models. In addition, for Model 3, we set the rank of factorized LSTM weight matrices as 2 based on our observations in Section IV-C. For all these experiments, we perform 100 trials and plot the averaged curves. Moreover, we set the learning rate as $\eta = 0.1$, $\boldsymbol{X}_t \in \mathbb{R}^5$, and the output dimensionality as $m = 5$.

Since Model 2 and Model 3 have an additive structure, the gradient of each parameter becomes more robust to the vanishing and exploding gradient problems. Thus, in Fig. 3, Model 2 and Model 3 outperform Model 1 in terms of the error performance. Although both the models, i.e., Model 2 and Model 3, perform similarly, we consider Model 3 as superior due to having a smaller number of network parameters to be trained (see details in Section IV-E). Likewise, in Fig. 4, Model 2 and Model 3 have smaller error than Model 1. In Fig. 5, we evaluate the combined results of all the models with both

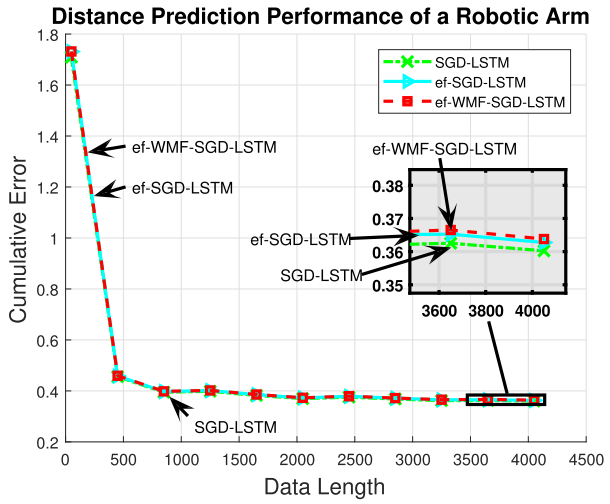

Fig. 6. Distance prediction performances of the algorithms with the SGD updates on the kinematic data set.

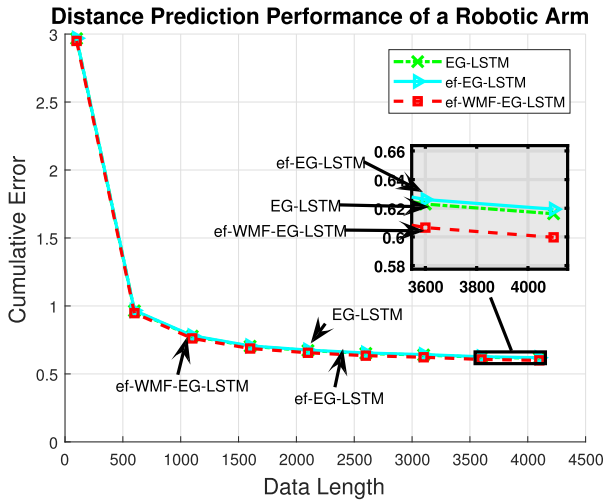

Fig. 7. Distance prediction performances of the algorithms with the EG updates on the kinematic data set.

the SGD and EG updates. Model 2 and Model 3 with the SGD updates provide slightly smaller steady-state errors compared with all other models. Overall, Model 3 with the SGD updates outperforms its competitors in terms of both error performance and complexity issues.

B. Real-Life Data Sets

In this section, we compare the performances of our algorithms using two real-life data sets, i.e., the kinematic [32] and elevators [33] data sets. We first evaluate the performances of the models on the kinematic data set [32], which contains the data related to a realistic simulation of the forward dynamics of an eight-link all-revolute robot arm. Our aim is to predict the distance of the end-effector from a target. In order to provide a fair experimental environment, we select the same common parameters for all the models. For this data set, the input vector is $\boldsymbol{X}_t \in \mathbb{R}^8$, $m = 8$, and the learning rate for all the models is $\eta = 0.1$. For Model 3, we select the rank of network matrices as 2. As shown in Fig. 6, all the models perform similarly. However, in terms of computational complexity and the total number of network parameters, Model 3 has the lowest complexity and the total number of parameters. In Fig. 7, Model 3 outperforms all other models thanks to having a smaller number of parameters and less complicated optimization problem for the parameters. In Fig. 8, we compare the models with both the SGD and EG updates. We observe that Model 1 with the SGD updates achieves a slightly smaller error compared with Model 2 and Model 3.

In addition to the kinematic data set, we also evaluate the performances on the elevators data set [33], which is obtained from the movements of an F16 aircraft, and we aim to predict the variable that expresses the movements of the aircraft. In this case, we have $\boldsymbol{X}_t \in \mathbb{R}^{18}$, $m = 18$, and select the learning rate as $\eta = 0.1$. The rank for Model 3 is 2. In Fig. 9, all the models using the EG updates outperform the models using the SGD updates, which arises from the

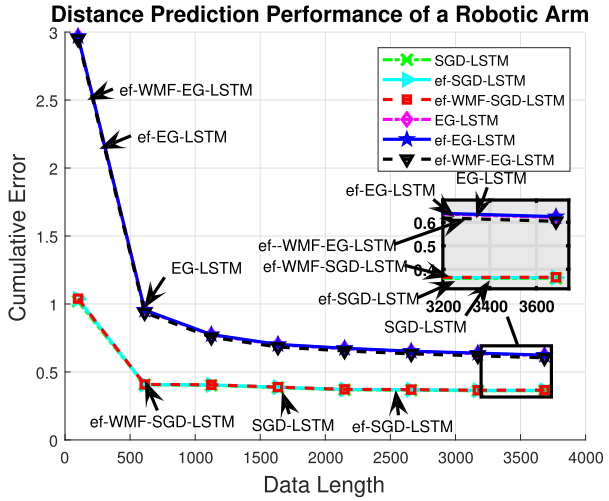

Fig. 8. Comparison of sequential prediction performances of the algorithms on the kinematic data set.

sparseness of $\boldsymbol{X}_t$ unlike the previous experiments. Among the models with the SGD updates, Model 3 has the smallest error, and for the EG updates, all the models provide comparable performances. Although all the models perform similarly, Model 3 is the ideal choice because of having less complexity and a smaller number of network parameters compared with the other models.

C. Rank Effect on WMF

In this section, we illustrate the effects of the rank on the performances of the introduced models. For this purpose, we use the Alcoa Corporation stock price data set. In Table I, we observe that as the rank decreases, the training time also decreases for the LSTM-based WMF models with both the SGD and EG updates, and the errors stay approximately the same. From this fact, we conclude that we can use lower rank weight matrices for our proposed models and still get the same performance in less amount of time. Thus, with our approach,

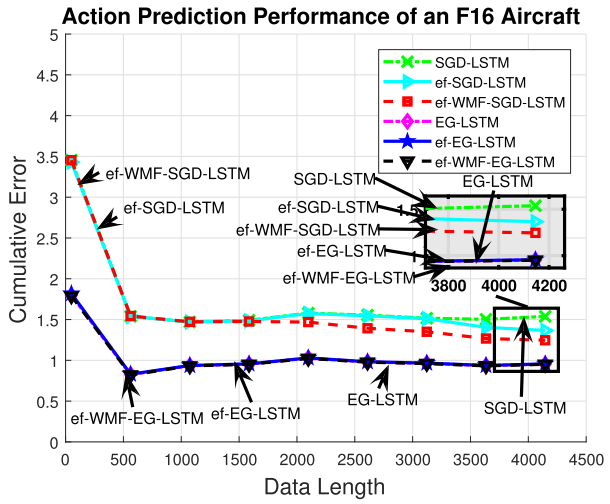

Fig. 9. Comparison of Movement prediction performances of the algorithms on the elevators data set.

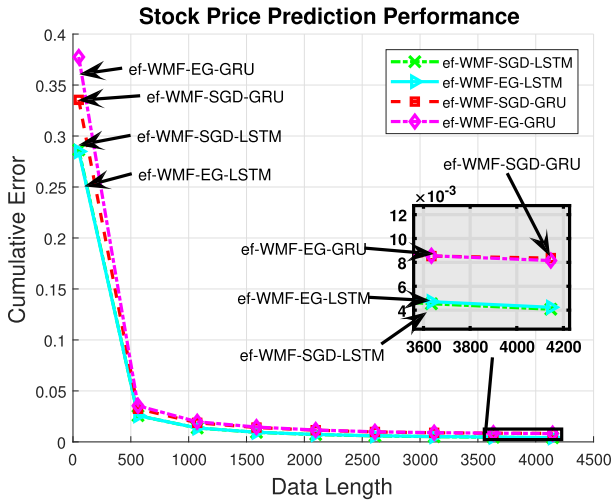

Fig. 10. Comparison of energy-efficient LSTM and GRU networks on the Alcoa Corporation data set.

TABLE I

TRAINING TIMES (IN S) FOR ONE TRIAL AND TIME ACCUMULATED ERRORS FOR THE WMF ALGORITHMS USING DIFFERENT RANK WEIGHT MATRICES ON THE ALCOA CORPORATION STOCK PRICE DATA SET. NOTE THAT THIS EXPERIMENT IS PERFORMED WITH A COMPUTER THAT HAS I5-6400 PROCESSOR, 2.7-GHZ CPU, AND 16-GB RAM

	Rank=2		Rank=3	
	Error	Time	Error	Time
WMF-SGD-LSTM	4.1×10^{-3}	103.15	4×10^{-3}	120.31
WMF-EG-LSTM	4.2×10^{-3}	115.38	4.1×10^{-3}	138.2

TABLE II

RELATIVE ENERGY CONSUMPTIONS (IN pJ) OF THE INTRODUCED RNN NETWORKS AT EACH TIME STEP. HERE, WE USE THE ENERGY CONSUMPTION DATA OF ARITHMETIC OPERATIONS FOR A 45-nm CMOS PROCESS [34]

Networks / Datasets	LSTM		GRU	
	LSTM	ef-LSTM	GRU	ef-GRU
Alcoa	980	526	741	390
Kinematic	2451.2	1187.2	1848	883.2
Elevators	12139	5263.2	9126	3931.2

one can significantly reduce the number of parameters to be trained in an LSTM network while enjoying high performance. Based on these observations, in all experiments, we select the rank as 2. Note that we do not reduce the rank to 1 since WMF significantly degrades the performance in that case.

D. LSTM and GRU Neural Networks

In this section, we evaluate the performances of the algorithms on the real-life and financial data sets. Since our approach is generic, in the sense, that it can be applied to any RNN structure, we also include the GRU-based algorithms to provide comparative analysis. The GRU network is defined by the following equations [26]:

$$\tilde{\boldsymbol{z}}_{t,j} = \sigma(\boldsymbol{W}^{(\tilde{z})}\boldsymbol{x}_{t,j} + \boldsymbol{R}^{(\tilde{z})}\boldsymbol{h}_{t,j-1}) \quad (43)$$

$$\boldsymbol{r}_{t,j} = \sigma(\boldsymbol{W}^{(r)}\boldsymbol{x}_{t,j} + \boldsymbol{R}^{(r)}\boldsymbol{h}_{t,j-1}) \quad (44)$$

$$\tilde{\boldsymbol{y}}_{t,j} = g(\boldsymbol{W}^{(y)}\boldsymbol{x}_{t,j} + \boldsymbol{r}_{t,j} \odot (\boldsymbol{R}^{(y)}\boldsymbol{h}_{t,j-1})) \quad (45)$$

$$\boldsymbol{y}_{t,j} = \tilde{\boldsymbol{y}}_{t,j} \odot \tilde{\boldsymbol{z}}_{t,j} + \boldsymbol{y}_{t,j-1} \odot (1 - \tilde{\boldsymbol{z}}_{t,j}) \quad (46)$$

where $\boldsymbol{x}_{t,j} \in \mathbb{R}^p$ is the input vector and $\boldsymbol{y}_{t,j} \in \mathbb{R}^m$ is the output vector. Here, $\tilde{\boldsymbol{z}}_{t,j}$ and $\boldsymbol{r}_{t,j}$ are the update and reset gates, respectively. The functions $g(\cdot)$ and $\sigma(\cdot)$ apply to vectors pointwise and commonly set to the $\tanh(\cdot)$ and sigmoid functions, respectively. In order to obtain an energy-efficient version of the GRU network, we apply the matrix factorization method and the ef-operator as in the LSTM case.

Here, we select the same parameters with Sections IV-A–IV-C. Since Model 3 provides the best performance in Sections IV-A–IV-C, we compare the LSTM and GRU Networks on three data sets using Model 3. For the Alcoa Corporation stock price data set, the LSTM-based algorithm with the SGD updates achieves the smallest steady-state error, as shown in Fig. 10. In Fig. 11, the GRU-based algorithm with the SGD updates outperforms the other network models. For the elevators data set, the LSTM-based algorithm with the EG updates achieves the smallest steady-state error among all the network models, as shown in Fig. 12. Overall, since the LSTM architecture has an output gate to control its memory content unlike the GRU network, it generally outperforms the GRU network on various real-life scenarios.

Moreover, in order to illustrate the energy efficiency of the introduced architectures, we provide energy consumption data for each data set in Table II. We observe that our approach almost halves the energy consumption for each case, and as the dimensionality of the data set increases, the provided energy efficiency even further increases.

E. Training Times and Structural Complexity

In this section, we first provide the training times (in s) of all the LSTM network-based models with both the SGD

TABLE III
TRAINING TIMES (IN S FOR ONE TRIAL) OF THE INTRODUCED ENERGY-EFFICIENT LSTM NETWORKS

Algorithms / Datasets	SGD-LSTM	ef-SGD-LSTM	ef-WMF-SGD-LSTM	EG-LSTM	ef-EG-LSTM	ef-WMF-EG-LSTM
Alcoa	55.71	144.98	103.15	66.43	155.07	115.38
Kinematic	99.84	181.10	167.50	117.01	200.61	178.17
Elevators	545.97	1052.59	384.57	676.73	1185.97	404.44

TABLE IV
TIME ACCUMULATED ERRORS OF THE INTRODUCED ALGORITHMS

Algorithms / Datasets	SGD-LSTM	ef-SGD-LSTM	ef-WMF-SGD-LSTM	EG-LSTM	ef-EG-LSTM	ef-WMF-EG-LSTM
Alcoa	0.0084	0.0041	0.0041	0.0069	0.0042	0.0042
Kinematic	0.3595	0.3621	0.3629	0.6139	0.6171	0.5975
Elevators	1.5607	1.367	1.2478	0.9587	0.9611	0.9564

TABLE V
TOTAL NUMBER OF PARAMETERS TO BE LEARNED FOR THE INTRODUCED NETWORKS

Algorithms / Datasets	LSTM	ef-LSTM	ef-WMF-LSTM	GRU	ef-GRU	ef-WMF-GRU
Alcoa	220	260	220	150	180	150
Kinematic	544	608	352	384	432	240
Elevators	2664	2808	792	1944	2052	540

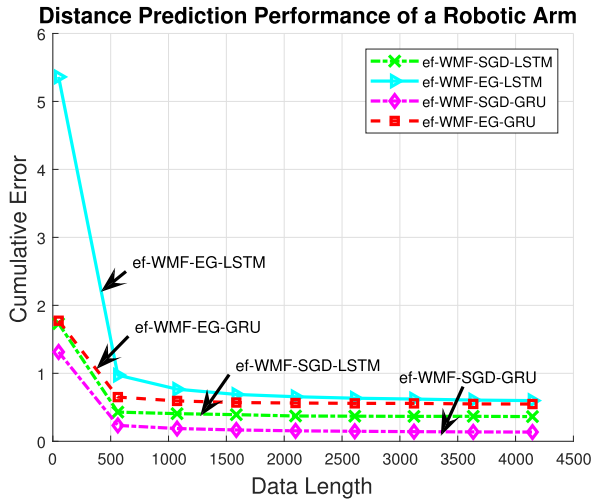

Fig. 11. Comparison of energy-efficient LSTM and GRU networks on the kinematic data set.

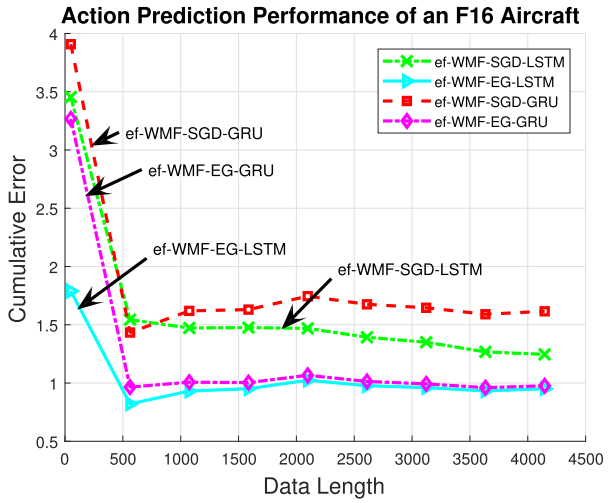

Fig. 12. Comparison of energy-efficient LSTM and GRU networks on the elevators data set.

and EG updates. We then give the total number of network parameters for each model, i.e., the structural complexity of the corresponding model. Finally, we compare all the models based on the training times, the number of network parameters, and error performance.

In Table III, we provide the training times of all the network models for each data set. Note that all the experiments are performed with a computer that has i5-6400 processor, 2.7-GHz CPU, and 16-GB RAM. Among all the network models, Model 1 has the fastest training performance when the data size is small, and however, it does not have the smallest time accumulated errors, i.e., defined as

$$\sum_{t=1}^{T}(d_t - \hat{d}_t)^2$$

for a data set with T samples, as stated in Table IV. Model 3 achieves intermediate training times (and the fastest training when the data size is large as in the elevators data set) and the smallest cumulative errors, as shown in Tables III and IV, respectively. In Table V, we provide the total number of network parameters for each model. We observe that Model 3 has the smallest number of network parameters among all

other models thanks to the weight matrix factorization. Overall, based on training times, error performances, and the number of network parameters, Model 3 is the best choice among all the models.

V. Conclusion

In this article, we have studied the variable-length data regression in an online framework and introduced an energy-efficient regression structure based on the LSTM network. In particular, we have introduced a generic LSTM-based regression structure to obtain fixed-length representations from variable-length data sequences. In order to reduce the complexity of this structure, we first eliminate the regular multiplications by replacing them with an energy-efficient operator, i.e., the ef-operator. We then apply a factorization method to all the matrices in the classical LSTM network in order to diminish the total number of parameters. For this energy-efficient and factorized LSTM network, we have introduced the online training algorithms based on the SGD [11] and EG [17] algorithms. Hence, we obtain highly efficient and effective online learning algorithms based on the LSTM network. Thanks to the generic structure of our approach, we have also introduced an energy-efficient GRU network in our simulations. Through several experiments involving real and financial data, we demonstrate significant performance improvements and complexity reductions achieved by the introduced algorithms with respect to the conventional methods.

References

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

[2] N. Wang, M. J. Er, and M. Han, "Generalized single-hidden layer feedforward networks for regression problems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 6, pp. 1161–1176, Jun. 2015.

[3] T. Ding and A. Hirose, "Fading channel prediction based on combination of complex-valued neural networks and chirp Z-transform," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 9, pp. 1686–1695, Sep. 2014.

[4] J.-T. Chien and Y.-C. Ku, "Bayesian recurrent neural network for language modeling," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 361–374, Feb. 2016.

[5] A. C. Tsoi and A. C. Tsoi, "Gradient based learning methods," in *Adaptive Processing of Sequences and Data Structures: International Summer School on Neural Networks 'E. R. Caianiello' Vietri sul Mare, Salerno, Italy September 6–13, 1997 Tutorial Lectures*, G. L. Giles and M. Gori, Eds. Berlin, Germany: Springer, 1998, pp. 27–92. doi: 10.1007/BFb0053994.

[6] 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.

[7] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. ICML*, vol. 28, 2013, pp. 1310–1318.

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

[9] 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.

[10] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach," *GMD-Forschungszentrum Informationstechnik*, vol. 5, Jan. 2002.

[11] A. W. Smith and D. Zipser, "Learning sequential structure with the real-time recurrent learning algorithm," *Int. J. Neural Syst.*, vol. 1, no. 2, pp. 125–131, 1989.

[12] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2016.

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

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

[15] D. Monner and J. A. Reggia, "A generalized LSTM-like training algorithm for second-order recurrent neural networks," *Neural Netw.*, vol. 25, pp. 70–83, Jan. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608011002036

[16] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Netw.*, vol. 18, no. 5, pp. 602–610, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608005001206

[17] J. Kivinen and M. K. Warmuth, "Exponentiated gradient versus gradient descent for linear predictors," *Inf. Comput.*, vol. 132, no. 1, pp. 1–63, 1997.

[18] S. I. Hill and R. C. Williamson, "Convergence of exponentiated gradient algorithms," *IEEE Trans. Signal Process.*, vol. 49, no. 6, pp. 1208–1215, Jun. 2001.

[19] H. Tuna, I. Onaran, and A. E. Cetin, "Image description using a multiplier-less operator," *IEEE Signal Process. Lett.*, vol. 16, no. 9, pp. 751–753, Sep. 2009.

[20] C. E. Akbaş, A. Bozkurt, A. E. Çetin, R. Çetin-Atalay, and A. Üner, "Multiplication-free neural networks," in *Proc. 23nd Signal Process. Commun. Appl. Conf. (SIU)*, May 2015, pp. 2416–2418.

[21] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, "LookNN: Neural network with no multiplication," in *Proc. Design Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1775–1780.

[22] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[23] A. Afrasiyabi, B. Nasir, O. Yildiz, F. T. Y. Vural, and A. E. Cetin, "An energy efficient additive neural network," in *Proc. 25th Signal Process. Commun. Appl. Conf. (SIU)*, May 2017, pp. 1–4.

[24] O. Kuchaiev and B. Ginsburg, "Factorization tricks for LSTM networks," *CoRR*, vol. abs/1703.10722, 2017. [Online]. Available: http://arxiv.org/abs/1703.10722

[25] N. Srinivasan, V. Ravichandran, K. L. Chan, J. R. Vidhya, S. Ramakirishnan, and S. M. Krishnan, "Exponentiated backpropagation algorithm for multilayer feedforward neural networks," in *Proc. 9th Int. Conf. Neural Inf. Process. (ICONIP)*, vol. 1, Nov. 2002, pp. 327–331.

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

[27] E. Levin, "A recurrent neural network: Limitations and training," *Neural Netw.*, vol. 3, no. 6, pp. 641–650, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089360809090054O

[28] A. Afrasiyabi, O. Yildiz, B. Nasir, F. T. Yarman-Vural, and A. E. Çetin, "Energy saving additive neural network," *CoRR*, 2017. [Online]. Available: https://arxiv.org/abs/1702.02676

[29] N. D. Vanli and S. S. Kozat, "A comprehensive approach to universal piecewise nonlinear regression based on trees," *IEEE Trans. Signal Process.*, vol. 62, no. 20, pp. 5471–5486, Oct. 2014.

[30] R. N. Bracewell and R. N. Bracewell, *The Fourier Transform & Its Applications*, vol. 31999. New York, NY, USA: McGraw-Hill, 1986.

[31] *Summary for Alcoa Inc. Common Stock*. Accessed: Apr. 1, 2018. [Online]. Available: http://finance.yahoo.com/quote/AA?ltr=1

[32] C. E. Rasmussen *et al. Delve Data Sets*. Accessed: Apr. 1, 2018. [Online]. Available: http://www.cs.toronto.edu/~delve/data/datasets.html

[33] 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.

[34] M. Horowitz, "Energy table for 45nm process," Stanford VLSI wiki.

Tolga Ergen received the B.S. and M.S. degrees in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2016 and 2018, respectively. He is currently pursuing the Ph.D. degree with the Electrical Engineering Department, Stanford University, Stanford, CA, USA.

His current research interests include machine learning, optimization, and neural networks.

Ali H. Mirza received the B.Sc. degree (Hons.) in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 2014. He is currently pursuing the Ph.D. degree with the Department of Electrical and Electronics Engineering, Bilkent University, Ankara, Turkey.

His current research interests include machine learning, big data signal processing, online learning, and deep neural networks.

Suleyman Serdar Kozat (A'10–M'11–SM'11) received the B.S. degree (Hons.) 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 a Professor with the Electrical and Electronics Engineering Department, Bilkent University. He has coauthored over 200 papers in refereed high-impact journals and conference proceedings and holds several patent inventions (used in several different Microsoft and IBM products). He holds several patent inventions due to his research accomplishments with the IBM Thomas J. Watson Research Center and Microsoft Research. His current research interests include cybersecurity, 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.