

ONLINE LEARNING WITH RECURRENT NEURAL NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

By
Tolga Ergen
July, 2018

Online Learning with Recurrent Neural Networks

By Tolga Ergen

July, 2018

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Süleyman Serdar Kozat(Advisor)

Sinan Gezici

Çağatay Candan

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

ONLINE LEARNING WITH RECURRENT NEURAL NETWORKS

Tolga Ergen

M.S. in Electrical and Electronics Engineering

Advisor: Süleyman Serdar Kozat

July, 2018

In this thesis, we study online learning with Recurrent Neural Networks (RNNs). Particularly, in Chapter 2, we investigate online nonlinear regression and introduce novel regression structures based on the Long Short Term Memory (LSTM) network, i.e., is an advanced RNN architecture. To train these novel LSTM based structures, we introduce highly efficient and effective Particle Filtering (PF) based updates. We also provide Stochastic Gradient Descent (SGD) and Extended Kalman Filter (EKF) based updates. Our PF based training method guarantees convergence to the optimal parameter estimation in the Mean Square Error (MSE) sense. In Chapter 3, we investigate online training of LSTM architectures in a distributed network of nodes, where each node employs an LSTM based structure for online regression. We first provide a generic LSTM based regression structure for each node. In order to train this structure, we introduce a highly effective and efficient Distributed PF (DPF) based training algorithm. We also introduce a Distributed EKF (DEKF) based training algorithm. Here, our DPF based training algorithm guarantees convergence to the performance of the optimal centralized LSTM parameters in the MSE sense. In Chapter 4, we investigate variable length data regression in an online setting and introduce an energy efficient regression structure build on LSTM networks. To reduce the complexity of this structure, we first replace the regular multiplication operations with an energy efficient operator. We then apply factorizations to the weight matrices so that the total number of parameters to be trained is significantly reduced. We then introduce online training algorithms. Through a set of experiments, we illustrate significant performance gains and complexity reductions achieved by the introduced algorithms with respect to the state of the art methods.

Keywords: Online learning, recurrent neural network (RNN), extended Kalman filtering (EKF), particle filtering (PF), stochastic gradient descent (SGD).

ÖZET

YİNELENEN SINIR AĞLARI İLE ÇEVİRİMİÇİ ÖĞRENİM

Tolga Ergen

Elektrik ve Elektronik Mühendisliği, Yüksek Lisans

Tez Danışmanı: Süleyman Serdar Kozat

Temmuz, 2018

Bu tezde, Yinelenen Sinir Ağları (YSA) ile çevrimiçi öğrenim problemini çalışmaktayız. Ayrıntılı olarak, ikinci bölümde, doğrusal olmayan çevrimiçi bağlanım problemini incelemekteyiz ve gelişmiş bir YSA olan Uzun Kısa Soluklu Bellek (UKSB) ağları tabanlı özgün bağlanım yapıları sunmaktayız. Bu yapıları eğitmek için, oldukça verimli ve etkili Parçacık Süzme (PS) tabanlı güncellemeler sunmaktayız. Buna ek olarak, Olasılıksal Gradyan İnişi (OGİ) ve Genişletilmiş Kalman Süzgeci (GKS) tabanlı güncellemeler sunmaktayız. PS tabanlı algoritmamız Ortalama Karesel Hata (OKH) performansı bakımından en iyi parametre tahminine yakınsamayı garanti etmektedir. Üçüncü bölümde, her bir düğümün çevrimiçi bağlanım için UKSB yapısını kullandığı dağınık ağlarda UKSB yapısının eğitimini incelemekteyiz. Öncelikle, her bir düğüm için genel bir UKSB yapısı sağlamaktayız. Bu yapıyı eğitmek için oldukça verimli ve etkili olan Dağınık PS (DPS) tabanlı eğitim algoritması sunmaktayız. Buna ek olarak, Dağınık GKS (DGKS) tabanlı eğitim algoritması sunmaktayız. Burada, DPS eğitim algoritmamız OKH performansı bakımından en iyi merkezi UKSB parametrelerine yakınsamayı garanti etmektedir. Dördüncü bölümde, değişken uzunluklu veri bağlanımını çevrimiçi bir düzende incelemekteyiz ve enerji tasarruflu bir UKSB yapısı sunmaktayız. Bu yapının karmaşıklığını düşürebilmek için öncelikle çarpım işlemlerini enerji tasarruflu bir işlem ile değiştirmekteyiz. Daha sonra ağırlık matrislerine faktörizasyon uygulayarak toplam parametre sayısını ciddi bir miktarda azaltmaktayız. Bundan sonra ise çevrimiçi eğitim algoritmaları sunmaktayız. Bir takım deneyler aracılığıyla, algoritmalarımız tarafından en gelişkin metotlara göre elde edilen performans kazançlarını ve karmaşıklık eksiltmelerini göstermekteyiz.

Anahtar sözcükler: Çevrimiçi öğrenim, yinelenen sinir ağı (YSA), genişletilmiş Kalman süzme (GKS), parçacık süzme (PS), olasılıksal gradyan inişi (OGİ).

Acknowledgement

I would like to present my sincere gratitudes to Assoc. Prof. Dr. Suleyman S. Kozat for his excellent guidance and support throughout my M.S. study. I would also like to thank for his insightful comments and feedbacks that enabled me to achieve this thesis.

I acknowledge that this work is supported by TÜBİTAK BİDEB 2210-A Scholarship Programme.

Contents

- 1 Introduction** **1**

- 2 Online Learning Algorithms Based on LSTM Networks** **10**
 - 2.1 Novel Learning Algorithms based on LSTM Neural Networks . . . 12
 - 2.1.1 Different Regression Architectures 12
 - 2.1.2 Conventional Online Training Algorithms 13
 - 2.1.3 Online Training based on the PF Algorithm 20
 - 2.2 Simulations 25
 - 2.2.1 Real Life Datasets 25
 - 2.2.2 Financial Datasets 29
 - 2.2.3 LSTM and GRU Networks 30
 - 2.2.4 Different Regression Architectures 32

- 3 Online Training of LSTM Networks in Distributed Systems** **34**
 - 3.1 Online Distributed Training Algorithms 36

3.1.1	Online Training Using the DEKF Algorithm	37
3.1.2	Online Training Using the DPF Algorithm	40
3.2	Simulations	45
4	Energy Efficient LSTM Networks for Online Learning	52
4.1	Online Learning with Energy Efficient RNN Architectures	55
4.1.1	RNN with ef-operator	55
4.1.2	LSTM with ef-operator	56
4.1.3	Energy Efficient RNN with Weight Matrix Factorization	57
4.1.4	Energy Efficient LSTM with Weight Matrix Factorization	59
4.1.5	Online Training Algorithms	60
4.2	Simulations	66
4.2.1	Financial Dataset	68
4.2.2	Real Life Datasets	69
4.2.3	Rank Effect on WMF	72
4.2.4	LSTM and GRU Neural Networks	73
4.2.5	Training Times and Structural Complexity	75
5	Conclusion	78

List of Figures

2.1	Detailed schematic of the proposed architecture in (2.11) for the regression tasks. Note that for the summations before the gate and $h(\cdot)$ functions, we multiply \mathbf{x}_t and \mathbf{y}_{t-1} by $\mathbf{W}^{(\cdot)}$ and $\mathbf{R}^{(\cdot)}$, respectively and also we add the weight vector $\mathbf{b}^{(\cdot)}$ to these summations. We omit these operations for presentation simplicity.	14
2.2	Sequential prediction performances of the algorithms for the kinematic dataset.	28
2.3	Comparison of the PF based algorithm with different number of particles for the kinematic dataset.	28
2.4	Comparison of the PF based algorithm with different N - m combinations for the kinematic dataset. Note that all the combinations in this figure has the same computation time.	29
2.5	Future price prediction performances of the algorithms for the Alcoa stock price dataset.	31
2.6	Exchange rate prediction performances of the algorithms for the Hong Kong exchange rate dataset.	31
2.7	Comparison of the LSTM and GRU architectures in terms of regression error performance for (a) the PF based algorithm, (b) the EKF based algorithm and (c) the SGD based algorithm.	33

3.1	Detailed schematic of each node k in our network.	35
3.2	Error performances over the Hong Kong exchange rate dataset.	46
3.3	Error performances for different N and s combinations of the DPF based algorithm. Here, we also provide computation times of the combinations (in seconds), i.e., denoted as T , where a computer with i5-6400 processor, 2.7 GHz CPU and 16 GB RAM is used.	47
3.4	Error performances over the sentence dataset.	48
3.5	Sequential prediction performance of the algorithms for the temperature dataset.	50
4.1	Detailed schematic of Energy Efficient LSTM based architecture.	54
4.2	Detailed schematic of energy efficient LSTM block at time t . Note that solid lines represent direct connections while dotted lines represent time lagged connections. For the sake of simplicity, bias terms are not shown in the figure.	58
4.3	Daily stock price prediction performances of the algorithms with the SGD updates on the Alcoa Corporation stock price dataset.	67
4.4	Daily stock price prediction performances of the algorithms with the EG updates on the Alcoa Corporation stock price dataset.	67
4.5	Comparison of all the algorithms with the SGD and EG updates on the Alcoa Corporation stock price dataset.	68
4.6	Distance prediction performances of the algorithms with the SGD updates on the kinematic dataset.	70
4.7	Distance prediction performances of the algorithms with the EG updates on the kinematic dataset.	71

4.8	Comparison of sequential prediction performances of the algorithms on the kinematic dataset.	71
4.9	Movement prediction performances of the algorithms on the elevators dataset.	72
4.10	Comparison of energy efficient LSTM and GRU networks on the Alcoa Corporation dataset.	74
4.11	Comparison of energy efficient LSTM and GRU networks on the kinematic dataset.	74
4.12	Comparison of energy efficient LSTM and GRU networks on the elevators dataset.	75

List of Tables

2.1	Comparison of the computational complexities of the proposed on-line training methods. In the table, p represents the dimensionality of the input space, m represents the dimensionality of the network's output space, and N represents the number of particles for the PF algorithm.	15
2.2	Time accumulated errors and the corresponding training times (in seconds) of the LSTM based algorithms for the elevators, pumadyn and bank datasets. Note that here, we use a computer with i5-6400 processor, 2.7 GHz CPU and 16 GB RAM.	30
2.3	Time accumulated errors of the LSTM based regression algorithms described in (2.10), (2.11) and (2.12) for each algorithm.	33
3.1	Comparison of the computational complexities of the introduced training algorithms for each node k . In this table, we also calculate the computational complexity of the SGD based algorithm by deriving exact gradient equations, however, we omit these calculations.	45
3.2	Time accumulated errors the algorithms for the speech, elevators, pumadyn and bank datasets.	50

4.1	Training times (in seconds) and time accumulated errors for the WMF algorithms using different rank weight matrices on the Alcoa Corporation stock price dataset. Note that this experiment is performed with a computer that has i5-6400 processor, 2.7 GHz CPU and 16 GB RAM.	73
4.2	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 $45nm$ CMOS process.	76
4.3	Training times (in seconds) of the introduced energy efficient LSTM networks.	76
4.4	Time accumulated errors of the introduced algorithms.	76
4.5	Total number of parameters to be learnt for the introduced networks.	77

Chapter 1

Introduction

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]. In this thesis, 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 only provide 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 Chapter 2, 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, then 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 back propagation recursion [4, 15]. However, these techniques provide poorer performance compared to 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 only exploits 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 to 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, in Chapter 2, 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 to 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 an extensive set of simulations, we illustrate significant performance improvements achieved by our algorithms compared to the conventional methods [18, 23, 24].

The main contributions of Chapter 2 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, 25] for each 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 to the both first and second order methods; 5) Through an 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 our simulation section.

In Chapter 3, we consider online training of the parameters of an LSTM structure in a distributed network of nodes. Here, we have a network of nodes, where each node has a set of neighboring nodes and can only exchange information with these neighbors. In particular, each node sequentially receives a variable length data sequence with its label and trains the parameters of the LSTM network. Each node can also communicate with its neighbors to share information in order to enhance the training performance since the goal is to train one set of LSTM coefficients using all the available data. As an example application, suppose that we have a database of labelled tweets and our aim is to train an emotion recognition engine based on an LSTM structure, where the training is performed in an online and distributed manner using several processing units. Words in each tweet are represented by word2vec vectors [26] and tweets are distributed to several processing units in an online manner.

The LSTM architectures are usually trained in a batch setting in the literature, where all data instances are present and processed together [5]. However, for applications involving big data, storage issues may arise due to keeping all the data in one place [27]. Additionally, in certain frameworks, all data instances are not available beforehand since instances are received in a sequential manner, which precludes batch training [27]. Hence, we consider online training, where

we sequentially receive the data to train the LSTM architecture without storing the previous data instances. Note that even though we work in an online setting, we may still suffer from computational power and storage issues due to large amount of data [28–30]. As an example, in tweet emotion recognition applications, the systems are usually trained using an enormous amount of data to achieve sufficient performance, especially for agglutinative languages [26]. For such tasks distributed architectures are used. In this basic distributed architectures, commonly named as centralized approach [28], the whole data is distributed to different nodes and trained parameters are merged later at a central node [5]. However, this centralized approach requires high storage capacity and computational power at the central node [28]. Additionally, centralized strategies have a potential risk of failure at the central node. To circumvent these issues, we distribute both the processing as well as the data to all the nodes and allow communication only between neighboring nodes, hence, we remove the need for a central node. In particular, each node sequentially receives a variable length data sequence with its label and exchanges information only with its neighboring nodes to train the common LSTM parameters.

For online training of the LSTM architecture in a distributed manner, one can employ one of the first order gradient based algorithms at each node due to their efficiency [5]. In the distributed implementation of the first order gradient based methods, each node exchanges either its estimate or its first order gradient with its neighboring nodes in order to compute the final estimate, e.g., [31] directly shares the estimates at each node with its neighbors and then updates the linear combination of the estimates to get the final estimate. However, since these training methods only exploit the first order gradient information, they suffer from poor performance and convergence issues. As an example, the Stochastic Gradient Descent (SGD) based algorithms usually have slower convergence compared to the second order methods [5, 31]. On the other hand, the second order gradient based methods require much higher computational complexity and communication load while providing superior performance compared to the first order methods [5]. Following the distributed implementation of the first order methods, one can implement the second order training methods in a distributed

manner, where we share not only the estimates but also the Jacobian matrix, e.g., the Distributed Extended Kalman Filtering (DEKF) algorithm [19,32]. However, as in the first order case, these sharing and combining the information at each node is adhoc, which does not provide the optimal training performance [32]. In addition to the EKF algorithm, the Hessian-Free (HF) [33] and Quasi-Newton (QN) [34] algorithms are also employed as the second order training methods for training RNNs. The HF algorithm avoids the direct Hessian computations, i.e., highly complex computations, so that it significantly reduces its computational complexity while enjoying high performance provided by a second order method [33]. Similarly, the QN algorithm approximately computes Hessian to save computational resources [34]. However, both of these algorithms provide restricted performances in online tasks [5,18]. In this chapter, to provide improved performance with respect to the second order methods while preserving both communication and computational complexity similar to the first order methods, we introduce a highly effective distributed online training method based on the particle filtering algorithm [35]. We first propose an LSTM based model for variable length data regression. We then put this model in a nonlinear state space form to train the model in an online and optimal manner [36].

The main contributions of Chapter 3 include: 1) We introduce distributed LSTM training methods in an online setting for variable length data sequences. Our Distributed Particle Filtering (DPF) based training algorithm guarantees convergence to the optimal centralized training performance in the Mean Square Error (MSE) sense; 2) We achieve this performance with a computational complexity and a communication load in the order of the first order gradient based methods; 3) Through simulations involving real life and financial data, we illustrate significant performance improvements with respect to the state of the art methods [18,23].

In Chapter 4, we investigate efficient training of LSTM networks for data regression. In the literature, LSTM networks are usually trained in a batch setting, where all data sequences are available and processed together for training [15,16]. However, in big data applications, such approaches might cause storage problems due to need to store all data sequences at one place [5,16,23]. Furthermore, in

certain settings, we sequentially receive data instances, which prevents training in a batch setting [16]. 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 [4, 5, 18, 23, 37]. Among these methods, the first order gradient based algorithms are generally employed due to their computational efficiency in training the LSTM network [4, 23, 38, 39]. The first order gradient based training algorithms usually perform additive updates, i.e., each parameter is updated through an addition operation, e.g., the SGD algorithm [4, 5, 23]. However, such algorithms suffer from slow convergence rate and poor performance, especially when only few components of the input data is related to the desired label [40]. To circumvent these issues, a first order training method with multiplicative updates, i.e., the Exponentiated Gradient (EG) algorithm, is introduced [40, 41]. However, since the EG algorithm employs multiplicative updates, it requires more computational resources compared to additive updates, especially for implementations on FPGAs [42, 43], which restricts its usage in real-life applications [40, 44, 45]. To be more precise, [43] shows that a multiplication operation consumes more than four times of the energy required by an addition operation. Furthermore, since the classical LSTM network has numerous vector matrix multiplications and several parameters to be trained, it requires high amount of energy and computational resources to provide an adequate performance [4, 44, 46, 47].

In order to address these issues, we introduce a novel energy efficient LSTM network and training methods based on the EG [40] and SGD [23] 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 significantly smaller number of the regular multiplication operations (only required for certain scaling operations) compared to the classical LSTM network. In order to further reduce the complexity, we also apply a matrix factorization method [48] to the LSTM parameters such that the number of parameters that needs to be learnt 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 [23, 49], we enjoy not only high performance provided by the LSTM network but also achieve low computational complexity in training. Here, thanks our generic approach, we also apply this approach to the GRU network [22] in our experiments. Through an extensive set of simulations, we illustrate significant performance gains and complexity reductions with respect to the conventional methods [23].

Our main contributions in Chapter 4 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 achieve substantial performance gains with respect to the classical LSTM architecture [12] trained with the conventional training methods [23]; 3) We achieve these substantial performance gains with a computational complexity that is significantly less than the conventional methods in the current literature [23]; 4) Through an extensive set of simulations, we demonstrate significant performance improvements achieved by the introduced methods with respect to the conventional methods [23]. Moreover, since our approach is generic, we also introduce an energy efficient GRU network in our simulation section.

Notation: In this thesis, all vectors are column vectors and denoted by boldface lower case letters. Matrices are represented by boldface upper case letters. For a matrix \mathbf{U} (or a vector \mathbf{u}), \mathbf{U}^T (or \mathbf{u}^T) is its ordinary transpose. The time index is given as subscript, e.g., \mathbf{u}_t is the vector at time t . For a vector \mathbf{u} , $|\mathbf{u}|$ and $\|\mathbf{u}\| = \sqrt{\mathbf{u}^T \mathbf{u}}$ are its ℓ_1 -norm and ℓ_2 -norm, respectively. For a vector \mathbf{u}_t , $u_{t,i}$ is the i^{th} element of that vector. Similarly, for a matrix \mathbf{U} , u_{ij} is the entry at the i^{th} row and j^{th} column of \mathbf{U} . Given a vector \mathbf{u} , $\text{diag}(\mathbf{u})$ is the diagonal matrix with the entries of \mathbf{u} at its diagonal. Moreover, $\mathbf{1}$ is a vector or matrix of all ones,

$\mathbf{0}$ is a vector or matrix of all zeros and \mathbf{I} is the identity matrix, where the sizes are understood from the context.

Chapter 2

Online Learning Algorithms Based on LSTM Networks

In this chapter, we sequentially receive $\{d_t\}_{t \geq 1}$, $d_t \in \mathbb{R}$ and regression vectors, $\{\mathbf{x}_t\}_{t \geq 1}$, $\mathbf{x}_t \in \mathbb{R}^p$ such that our goal is to estimate d_t based on our current and past observations $\{\dots, \mathbf{x}_{t-1}, \mathbf{x}_t\}$. Given our estimate \hat{d}_t , which can only be a function of $\{\dots, \mathbf{x}_{t-1}, \mathbf{x}_t\}$ and $\{\dots, d_{t-2}, d_{t-1}\}$, we suffer the loss $l(d_t, \hat{d}_t)$. This framework models a wide range of machine learning problems including financial analysis [50], tracking [51] 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 $\mathbf{x}_t = [x_t, x_{t-1} \dots, x_{t-p+1}]^T$ and then generate \hat{d}_t , after $d_t = x_{t+1}$ is observed, we suffer $l(d_t, \hat{d}_t) = (d_t - \hat{d}_t)^2$.

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

$$\mathbf{h}_t = \kappa \left(\mathbf{W}^{(h)} \mathbf{x}_t + \mathbf{R}^{(h)} \mathbf{h}_{t-1} \right) \quad (2.1)$$

$$\mathbf{y}_t = u \left(\mathbf{R}^{(y)} \mathbf{h}_t \right), \quad (2.2)$$

where $\mathbf{h}_t \in \mathbb{R}^m$ is the state vector, $\mathbf{x}_t \in \mathbb{R}^p$ is the input and $\mathbf{y}_t \in \mathbb{R}^m$ 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 $\mathbf{W}^{(h)} \in \mathbb{R}^{m \times p}$, $\mathbf{R}^{(h)} \in \mathbb{R}^{m \times m}$

and $\mathbf{R}^{(y)} \in \mathbb{R}^{m \times m}$.

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]

$$\mathbf{z}_t = h \left(\mathbf{W}^{(z)} \mathbf{x}_t + \mathbf{R}^{(z)} \mathbf{y}_{t-1} + \mathbf{b}^{(z)} \right) \quad (2.3)$$

$$\mathbf{i}_t = \sigma \left(\mathbf{W}^{(i)} \mathbf{x}_t + \mathbf{R}^{(i)} \mathbf{y}_{t-1} + \mathbf{b}^{(i)} \right) \quad (2.4)$$

$$\mathbf{f}_t = \sigma \left(\mathbf{W}^{(f)} \mathbf{x}_t + \mathbf{R}^{(f)} \mathbf{y}_{t-1} + \mathbf{b}^{(f)} \right) \quad (2.5)$$

$$\mathbf{c}_t = \Lambda_t^{(i)} \mathbf{z}_t + \Lambda_t^{(f)} \mathbf{c}_{t-1} \quad (2.6)$$

$$\mathbf{o}_t = \sigma \left(\mathbf{W}^{(o)} \mathbf{x}_t + \mathbf{R}^{(o)} \mathbf{y}_{t-1} + \mathbf{b}^{(o)} \right) \quad (2.7)$$

$$\mathbf{y}_t = \Lambda_t^{(o)} h(\mathbf{c}_t), \quad (2.8)$$

where $\Lambda_t^{(f)} = \text{diag}(\mathbf{f}_t)$, $\Lambda_t^{(i)} = \text{diag}(\mathbf{i}_t)$ and $\Lambda_t^{(o)} = \text{diag}(\mathbf{o}_t)$. Furthermore, $\mathbf{c}_t \in \mathbb{R}^m$ is the state vector, $\mathbf{x}_t \in \mathbb{R}^p$ is the input vector, $\mathbf{y}_t \in \mathbb{R}^m$ is the output vector. Here, \mathbf{i}_t , \mathbf{f}_t and \mathbf{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 $\mathbf{W}^{(z)} \in \mathbb{R}^{m \times p}$, $\mathbf{R}^{(z)} \in \mathbb{R}^{m \times m}$, $\mathbf{b}^{(z)} \in \mathbb{R}^m$, $\mathbf{W}^{(i)} \in \mathbb{R}^{m \times p}$, $\mathbf{R}^{(i)} \in \mathbb{R}^{m \times m}$, $\mathbf{b}^{(i)} \in \mathbb{R}^m$, $\mathbf{W}^{(f)} \in \mathbb{R}^{m \times p}$, $\mathbf{R}^{(f)} \in \mathbb{R}^{m \times m}$, $\mathbf{b}^{(f)} \in \mathbb{R}^m$, $\mathbf{W}^{(o)} \in \mathbb{R}^{m \times p}$, $\mathbf{R}^{(o)} \in \mathbb{R}^{m \times m}$ and $\mathbf{b}^{(o)} \in \mathbb{R}^m$. Given the output \mathbf{y}_t , we generate the final estimate as:

$$\hat{d}_t = \mathbf{w}_t^T \mathbf{y}_t, \quad (2.9)$$

where the final regression coefficients \mathbf{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 2.0.1. *The basic LSTM network can be extended by including last s outputs in the recursion, e.g., $\{\mathbf{y}_{t-s}, \dots, \mathbf{y}_{t-1}\}$, however this case corresponds to*

an extended output definition, i.e., an extended super output vector consisting of all $\{\mathbf{y}_{t-s}, \dots, \mathbf{y}_{t-1}\}$. We use only \mathbf{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 develop 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.

2.1 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.

2.1.1 Different Regression Architectures

We first consider the direct linear combination of the output \mathbf{y}_t with the weight vector \mathbf{w}_t . In this case, given (2.8), we generate the final estimate as

$$\begin{aligned} \hat{d}_t^{(1)} &= \mathbf{w}_t^T \mathbf{y}_t \\ &= \mathbf{w}_t^T \mathbf{\Lambda}_t^{(o)} h(\mathbf{c}_t), \end{aligned} \quad (2.10)$$

where $\mathbf{w}_t \in \mathbb{R}^m$. In (2.10), the final estimate of the system does not directly depend on \mathbf{x}_t . However, in generic nonlinear regression tasks, the final estimate usually depends on the current regression vector also [52]. 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. 2.1. Hence, we introduce the second regression architecture as

$$\hat{d}_t^{(2)} = \mathbf{w}_t^T \mathbf{\Lambda}_t^{(o)} h(\mathbf{c}_t) + \mathbf{v}_t^T \mathbf{\Lambda}_t^{(\alpha)} h(\mathbf{x}_t), \quad (2.11)$$

$\mathbf{v}_t \in \mathbb{R}^p$, in accordance with (2.10), where $\mathbf{\Lambda}_t^{(\alpha)} = \text{diag}(\boldsymbol{\alpha}_t)$ and

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

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

In (2.10) and (2.11), 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 (2.11) and introduce the third regression architecture as follows

$$\hat{d}_t^{(3)} = \mathbf{w}_t^T h(\mathbf{c}_t) + \mathbf{v}_t^T h(\mathbf{x}_t). \quad (2.12)$$

Note that $\hat{d}_t^{(2)}$ is our most general architecture to compute the final estimate since the updates for $\hat{d}_t^{(1)}$ is a special case when $\mathbf{\Lambda}_t^{(\alpha)} = \mathbf{0}$ and the updates for $\hat{d}_t^{(3)}$ is a special case when $\mathbf{\Lambda}_t^{(o)} = \mathbf{I}$ and $\mathbf{\Lambda}_t^{(\alpha)} = \mathbf{I}$. 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 $\hat{d}_t^{(3)}$.

2.1.2 Conventional Online Training Algorithms

In this subsection, 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 only exploits the first order gradient information so that it usually converges slower compared to 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

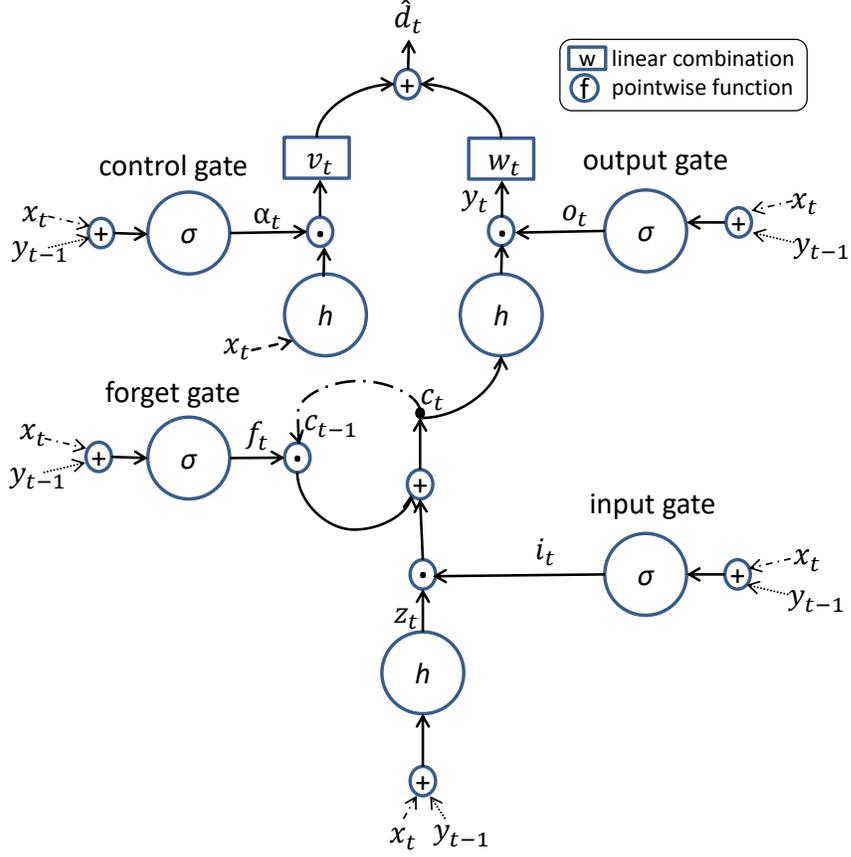


Figure 2.1: Detailed schematic of the proposed architecture in (2.11) for the regression tasks. Note that for the summations before the gate and $h(\cdot)$ functions, we multiply \mathbf{x}_t and \mathbf{y}_{t-1} by $\mathbf{W}^{(\cdot)}$ and $\mathbf{R}^{(\cdot)}$, respectively and also we add the weight vector $\mathbf{b}^{(\cdot)}$ to these summations. We omit these operations for presentation simplicity.

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]. Additionally, Table 2.1 shows that the EKF algorithm has high computational complexity compared to 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 (2.10), (2.11) and (2.12).

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

Table 2.1: Comparison of the computational complexities of the proposed online training methods. In the table, p represents the dimensionality of the input space, m represents the dimensionality of the network’s output space, and N represents the number of particles for the PF algorithm.

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

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t - \mu_t \nabla_{\mathbf{w}_t} l(d_t, \hat{d}_t) \\ &= \mathbf{w}_t + 2\mu_t (d_t - \hat{d}_t) \mathbf{\Lambda}_t^{(o)} h(\mathbf{c}_t), \end{aligned} \quad (2.13)$$

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

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

For notational simplicity, we derive the updates for each entry of $\mathbf{W}^{(z)}$ separately. We denote the entry in the i^{th} row and j^{th} column of $\mathbf{W}^{(z)}$ as $w_{ij}^{(z)}$. We have the following update for each entry of $\mathbf{W}^{(z)}$

$$w_{ij}^{(z)} = w_{ij}^{(z)} + 2\mu_t (d_t - \hat{d}_t) \mathbf{w}_t^T \frac{\partial \left(\mathbf{\Lambda}_t^{(o)} h(\mathbf{c}_t) \right)}{\partial w_{ij}^{(z)}}. \quad (2.14)$$

We write the partial derivative in (2.14) as

$$\frac{\partial \left(\mathbf{\Lambda}_t^{(o)} h(\mathbf{c}_t) \right)}{\partial w_{ij}^{(z)}} = \mathbf{\Lambda}_t \left(\frac{\partial o}{\partial w_{ij}^{(z)}} \right) h(\mathbf{c}_t) + \mathbf{\Lambda}_t^{(o)} \mathbf{\Lambda}_t^{(h'(c))} \frac{\partial \mathbf{c}_t}{\partial w_{ij}^{(z)}}, \quad (2.15)$$

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

$$\Lambda_t^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}}\right)} = \text{diag}\left(\frac{\partial \mathbf{o}_t}{\partial w_{ij}^{(z)}}\right).$$

Now, we compute the partial derivatives of \mathbf{o}_t and \mathbf{c}_t with respect to $w_{ij}^{(z)}$. Taking derivative of (2.7) gives

$$\frac{\partial \mathbf{o}_t}{\partial w_{ij}^{(z)}} = \Lambda_t^{(\sigma'(\zeta^{(o)}))} \left[\mathbf{R}^{(o)} \Lambda_{t-1}^{(o)} \Lambda_{t-1}^{(h'(c))} \boldsymbol{\psi}_{ij,t-1}^{(z)} + \mathbf{R}^{(o)} \Lambda_{t-1}^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}}\right)} h(\mathbf{c}_{t-1}) \right], \quad (2.16)$$

where

$$\zeta_t^{(o)} = \mathbf{W}^{(o)} \mathbf{x}_t + \mathbf{R}^{(o)} \Lambda_{t-1}^{(o)} h(\mathbf{c}_{t-1}) + \mathbf{b}^{(o)} \quad (2.17)$$

and

$$\boldsymbol{\psi}_{ij,t-1}^{(z)} = \frac{\partial \mathbf{c}_{t-1}}{\partial w_{ij}^{(z)}}. \quad (2.18)$$

To get (2.15), we also compute the partial derivative of \mathbf{c}_t with respect to $w_{ij}^{(z)}$. Using (2.18), we write the following recursive equation

$$\boldsymbol{\psi}_{ij,t}^{(z)} = \Lambda_t^{(z)} \frac{\partial \mathbf{i}_t}{\partial w_{ij}^{(z)}} + \Lambda_t^{(i)} \frac{\partial \mathbf{z}_t}{\partial w_{ij}^{(z)}} + \Lambda_{t-1}^{(c)} \frac{\partial \mathbf{f}_t}{\partial w_{ij}^{(z)}} + \Lambda_t^{(f)} \boldsymbol{\psi}_{ij,t-1}^{(z)}. \quad (2.19)$$

To obtain (2.19), we compute the partial derivatives of (2.3), (2.4) and (2.5) with respect to $w_{ij}^{(z)}$ as follows

$$\frac{\partial \mathbf{i}_t}{\partial w_{ij}^{(z)}} = \Lambda_t^{(\sigma'(\zeta^{(i)}))} \left[\mathbf{R}^{(i)} \Lambda_{t-1}^{(o)} \Lambda_{t-1}^{(h'(c))} \boldsymbol{\psi}_{ij,t-1}^{(z)} + \mathbf{R}^{(i)} \Lambda_{t-1}^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}}\right)} h(\mathbf{c}_{t-1}) \right] \quad (2.20)$$

$$\frac{\partial \mathbf{f}_t}{\partial w_{ij}^{(z)}} = \Lambda_t^{(\sigma'(\zeta^{(f)}))} \left[\mathbf{R}^{(f)} \Lambda_{t-1}^{(o)} \Lambda_{t-1}^{(h'(c))} \boldsymbol{\psi}_{ij,t-1}^{(z)} + \mathbf{R}^{(f)} \Lambda_{t-1}^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}}\right)} h(\mathbf{c}_{t-1}) \right] \quad (2.21)$$

$$\frac{\partial \mathbf{z}_t}{\partial w_{ij}^{(z)}} = \Lambda_t^{(h'(\zeta^{(z)}))} \left[\boldsymbol{\delta}_{ij} \mathbf{x}_t + \mathbf{R}^{(z)} \Lambda_{t-1}^{(o)} \Lambda_{t-1}^{(h'(c))} \boldsymbol{\psi}_{ij,t-1}^{(z)} + \mathbf{R}^{(z)} \Lambda_{t-1}^{\left(\frac{\partial o}{\partial w_{ij}^{(z)}}\right)} h(\mathbf{c}_{t-1}) \right], \quad (2.22)$$

where δ_{ij} is a $m \times p$ matrix with all entries zeros, except a 1 in the ij^{th} position. With these equations, we can compute (2.19) and then obtain (2.15) using (2.19) and (2.16). By this, we have all the required equations for the SGD update in (2.14).

Remark 2.1.1. *Here, we derive the updates just for the entries of $\mathbf{W}^{(z)}$. When we take the partial derivative of \hat{d}_t with respect to the entries of the other parameters as in (2.14), the equations (2.15), (2.18) and (2.19) still hold with a change of the derivative variable. For (2.16) and (2.20)–(2.22), we have also a change in the form and location of the $\delta_{ij}\mathbf{x}_t$ term. In particular, as in (2.22), when we take the derivative of $\mathbf{W}^{(\cdot)}$, $\mathbf{R}^{(\cdot)}$, and $\mathbf{b}^{(\cdot)}$ with respect to their entries, respectively, additional $\delta_{ij}\mathbf{x}_t$, $\delta_{ij}\mathbf{y}_{t-1}$ and δ_{ij} terms appear in the derivative equation of the corresponding structure, i.e., one of (2.16), (2.20), (2.21) and (2.22). Here, the size of δ_{ij} changes accordingly.*

Remark 2.1.2. *In case of $\hat{d}_t^{(2)}$, instead of (2.14), we have the following update*

$$w_{ij}^{(z)} = w_{ij}^{(z)} + 2\mu_t(d_t - \hat{d}_t) \left[\mathbf{w}_t^T \frac{\partial \left(\mathbf{\Lambda}_t^{(o)} h(\mathbf{c}_t) \right)}{\partial w_{ij}^{(z)}} + \mathbf{v}_t^T \mathbf{\Lambda}_t \left(\frac{\partial \alpha}{\partial w_{ij}^{(z)}} \right) h(\mathbf{x}_t) \right], \quad (2.23)$$

where the introduced partial derivative term $\partial \alpha / \partial w_{ij}^{(z)}$ is computed in the same manner with (2.16). Furthermore, we have an additional update for \mathbf{v}_t as follows

$$\mathbf{v}_{t+1} = \mathbf{v}_t + 2\mu_t(d_t - \hat{d}_t) \mathbf{\Lambda}_t^{(\alpha)} h(\mathbf{x}_t). \quad (2.24)$$

Then, we follow the derivations in (2.13), (2.15), (2.16), (2.19), (2.20), (2.21) and (2.22). For $\hat{d}_t^{(3)}$, we just set $\mathbf{\Lambda}_t^{(o)} = \mathbf{I}$ and $\mathbf{\Lambda}_t^{(\alpha)} = \mathbf{I}$ and then all the derivations in (2.13), (2.15), (2.16) and (2.19)–(2.24) follow as in $\hat{d}_t^{(2)}$.

According to the update equations in (2.15), (2.16) and (2.19), update of an entry of a parameter has a computational complexity $O(m^2 + mp)$ due to the matrix vector multiplications in (2.17). Since we have mp , m^2 and m entries for $\mathbf{W}^{(\cdot)}$, $\mathbf{R}^{(\cdot)}$ and $\mathbf{b}^{(\cdot)}$ respectively, this results in $O(m^4 + m^2p^2)$ computational complexity to update the entries of all parameters as given in Table 2.1.

2.1.2.2 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 (2.3)–(2.8) and (2.10). In the literature, there are certain EKF based methods to train LSTM, e.g., [18,37], however, these methods only estimate the parameters, i.e., $\boldsymbol{\theta}_t$. However, in our case, we also estimate the state and the output vector of LSTM, i.e., \mathbf{c}_t and \mathbf{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 [53]. 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 $\{\mathbf{w}, \mathbf{W}^{(z)}, \mathbf{R}^{(z)}, \mathbf{b}^{(z)}, \mathbf{W}^{(i)}, \mathbf{R}^{(i)}, \mathbf{b}^{(i)}, \mathbf{W}^{(f)}, \mathbf{R}^{(f)}, \mathbf{b}^{(f)}, \mathbf{W}^{(o)}, \mathbf{R}^{(o)}, \mathbf{b}^{(o)}\}$ 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

$$\mathbf{y}_t = \tau(\mathbf{c}_t, \mathbf{x}_t, \mathbf{y}_{t-1}) + \boldsymbol{\epsilon}_t \quad (2.25)$$

$$\mathbf{c}_t = \Omega(\mathbf{c}_{t-1}, \mathbf{x}_t, \mathbf{y}_{t-1}) + \mathbf{v}_t \quad (2.26)$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \mathbf{e}_t \quad (2.27)$$

$$d_t = \mathbf{w}_t^T \mathbf{y}_t + \varepsilon_t, \quad (2.28)$$

where $\tau(\cdot)$ and $\Omega(\cdot)$ are the nonlinear functions in (2.8) and (2.6) respectively, and $\boldsymbol{\epsilon}_t$, \mathbf{e}_t , \mathbf{v}_t and ε_t are zero mean Gaussian random variables. Additionally, $[\boldsymbol{\epsilon}_t^T, \mathbf{v}_t^T, \mathbf{e}_t^T]^T$ and ε_t are with variances \mathbf{Q}_t and R_t , respectively. Here, we assume that \mathbf{Q}_t and R_t are known or can be estimated from the data as detailed later in the chapter. We write (2.25), (2.26) and (2.27) in a compact form as

$$\begin{bmatrix} \mathbf{y}_t \\ \mathbf{c}_t \\ \boldsymbol{\theta}_t \end{bmatrix} = \begin{bmatrix} \tau(\mathbf{c}_t, \mathbf{x}_t, \mathbf{y}_{t-1}) \\ \Omega(\mathbf{c}_{t-1}, \mathbf{x}_t, \mathbf{y}_{t-1}) \\ \boldsymbol{\theta}_{t-1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\epsilon}_t \\ \mathbf{v}_t \\ \mathbf{e}_t \end{bmatrix} \quad (2.29)$$

$$d_t = \mathbf{w}_t^T \mathbf{y}_t + \varepsilon_t. \quad (2.30)$$

In the system described in (2.29) and (2.30), we are able to observe only d_t and we can estimate \mathbf{y}_t , \mathbf{c}_t and $\boldsymbol{\theta}_t$ based on the observed d_t values. Thus, we directly apply the EKF algorithm [19] to estimate \mathbf{y}_t , \mathbf{c}_t and $\boldsymbol{\theta}_t$ as follows

$$\begin{bmatrix} \mathbf{y}_{t|t} \\ \mathbf{c}_{t|t} \\ \boldsymbol{\theta}_{t|t} \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{t|t-1} \\ \mathbf{c}_{t|t-1} \\ \boldsymbol{\theta}_{t|t-1} \end{bmatrix} + \mathbf{L}_t (d_t - \mathbf{w}_{t|t-1}^T \mathbf{y}_{t|t-1}) \quad (2.31)$$

$$\mathbf{y}_{t|t-1} = \tau(\mathbf{c}_{t|t-1}, \mathbf{x}_t, \mathbf{y}_{t-1|t-1}) \quad (2.32)$$

$$\mathbf{c}_{t|t-1} = \Omega(\mathbf{c}_{t-1|t-1}, \mathbf{x}_t, \mathbf{y}_{t-1|t-1}) \quad (2.33)$$

$$\boldsymbol{\theta}_{t|t-1} = \boldsymbol{\theta}_{t-1|t-1} \quad (2.34)$$

$$\mathbf{L}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t (\mathbf{H}_t^T \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t + R_t)^{-1} \quad (2.35)$$

$$\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{L}_t \mathbf{H}_t^T \boldsymbol{\Sigma}_{t|t-1} \quad (2.36)$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_{t-1} \boldsymbol{\Sigma}_{t-1|t-1} \mathbf{F}_{t-1}^T + \mathbf{Q}_{t-1}, \quad (2.37)$$

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

$$\mathbf{H}_t^T = \begin{bmatrix} \frac{\partial \hat{d}_t}{\partial \mathbf{y}} & \frac{\partial \hat{d}_t}{\partial \mathbf{c}} & \frac{\partial \hat{d}_t}{\partial \boldsymbol{\theta}} \end{bmatrix} \bigg|_{\substack{\mathbf{y}=\mathbf{y}_{t|t-1} \\ \mathbf{c}=\mathbf{c}_{t|t-1} \\ \boldsymbol{\theta}=\boldsymbol{\theta}_{t|t-1}}} \quad (2.38)$$

and

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

where $\mathbf{F}_t \in \mathbb{R}^{(2m+n_\theta) \times (2m+n_\theta)}$ and $\mathbf{H}_t \in \mathbb{R}^{(2m+n_\theta)}$. For (2.35) and (2.37), we use \mathbf{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

$$\lambda_t = (d_t - \mathbf{w}_{t|t-1}^T \mathbf{y}_{t|t-1}). \quad (2.39)$$

For the estimation of \mathbf{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 \mathbf{Q}_t , we use the algorithm in [54], which provides a highly effective estimate of \mathbf{Q}_t .

Remark 2.1.3. For the EKF derivations of $\hat{d}_t^{(2)}$, we change the observation model in (2.30), the update in (2.31), the Jacobian computation in (2.38) and the definition in (2.39) according to the definition of the architecture in (2.11). Additionally, we also extend the parameter vector $\boldsymbol{\theta}_t$ by adding \mathbf{v}_t , $\mathbf{W}^{(\alpha)}$, $\mathbf{R}^{(\alpha)}$ and $\mathbf{b}^{(\alpha)}$. 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 (2.30), the update in (2.31), the Jacobian computation in (2.38) and the definition in (2.39) according to (2.12). Moreover, we modify $\boldsymbol{\theta}_t$ by removing $\mathbf{W}^{(\alpha)}$, $\mathbf{R}^{(\alpha)}$, $\mathbf{b}^{(\alpha)}$, $\mathbf{W}^{(o)}$, $\mathbf{R}^{(o)}$ and $\mathbf{b}^{(o)}$ 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 (2.31), (2.32), (2.33), (2.35), (2.36) and (2.37), 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 (2.35), (2.36) and (2.37).

2.1.3 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 to 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 2.1. 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 (2.29) and (2.30). Hence, we modify the system in (2.29) and

(2.30) as follows

$$\mathbf{a}_t = \varphi(\mathbf{a}_{t-1}, \mathbf{x}_t) + \boldsymbol{\eta}_t \quad (2.40)$$

$$d_t = \mathbf{w}_t^T \mathbf{y}_t + \xi_t, \quad (2.41)$$

where $\boldsymbol{\eta}_t$ and ξ_t are independent noise samples, $\varphi(\cdot, \cdot)$ is the nonlinear mapping in (2.29) and

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

For (2.40) and (2.41), we seek to obtain $\mathbf{E}[\mathbf{a}_t|d_{1:t}]$, i.e., the optimal state estimate in the mean square error (MSE) sense. For this purpose, we first find the posterior probability density function $p(\mathbf{a}_t|d_{1:t})$. 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 $\{\mathbf{a}_t^i, \omega_t^i\}_{i=1}^N$ denote the samples and the associated weights of the desired distribution, i.e., $p(\mathbf{a}_t|d_{1:t})$. Then, we obtain the desired distribution from its samples as follows

$$p(\mathbf{a}_t|d_{1:t}) \approx \sum_{i=1}^N \omega_t^i \delta(\mathbf{a}_t - \mathbf{a}_t^i), \quad (2.42)$$

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 $\{\mathbf{a}_t^i\}_{i=1}^N$, 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 $\mathbf{E}_p[\mathbf{a}_t|d_{1:t}]$, we use the following trick

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

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 $q(\mathbf{a}_t|d_{1:t})$ as our importance function to obtain the samples and the corresponding weights are calculated follows

$$\omega_t^i \propto \frac{p(\mathbf{a}_t^i|d_{1:t})}{q(\mathbf{a}_t^i|d_{1:t})}, \quad (2.43)$$

where the weights are normalized such that

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

To simplify the weight calculation, we can factorize (2.43) to obtain a recursive formulation for the update of the weights as follows [25]

$$\omega_t^i \propto \frac{p(d_t|\mathbf{a}_t^i)p(\mathbf{a}_t^i|\mathbf{a}_{t-1}^i)}{q(\mathbf{a}_t^i|\mathbf{a}_{t-1}^i, d_t)}\omega_{t-1}^i. \quad (2.44)$$

In (2.44), 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 non-negligible weights and contribute considerably to (2.42) [55]. In this sense, the optimal choice of the importance function is $p(\mathbf{a}_t|\mathbf{a}_{t-1}^i, d_t)$, however, this requires an integration that does not have an analytic form in most cases [56]. Thus, we choose $p(\mathbf{a}_t|\mathbf{a}_{t-1}^i)$ as the importance function, which provides a small variance for the weights but not zero as the optimal importance function does [21, 56]. This simplifies (2.44) as follows

$$\omega_t^i \propto p(d_t|\mathbf{a}_t^i)\omega_{t-1}^i. \quad (2.45)$$

We can now get the desired distribution to compute the conditional mean of the augmented state vector \mathbf{a}_t using (2.42) and (2.45). By this, we obtain the conditional mean for \mathbf{a}_t as follows

$$\begin{aligned} \mathbf{E}[\mathbf{a}_t|d_{1:t}] &= \int \mathbf{a}_t p(\mathbf{a}_t|d_{1:t}) d\mathbf{a}_t \\ &\approx \int \mathbf{a}_t \sum_{i=1}^N \omega_t^i \delta(\mathbf{a}_t - \mathbf{a}_t^i) d\mathbf{a}_t \\ &= \sum_{i=1}^N \omega_t^i \mathbf{a}_t^i. \end{aligned} \quad (2.46)$$

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 [55]. Due to this reason, although particles with very small weights have almost no contribution to our estimate in (2.46), we have to update them using (2.40) and (2.45). 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 [57], which is calculated as follows

$$N_{eff} = \frac{1}{\sum_{i=1}^N (\omega_t^i)^2}. \quad (2.47)$$

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 [55], then we apply the resampling algorithm introduced in [25], 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 $\mathbf{E}[\mathbf{a}_t|d_{1:t}]$, where the convergence is guaranteed under certain conditions as follows.

Remark 2.1.4. For the PF derivations of $\hat{d}_t^{(2)}$, we change the observation model in (2.41) according to the definition in (2.11). We also modify \mathbf{a}_t by adding \mathbf{v}_t , $\mathbf{W}^{(\alpha)}$, $\mathbf{R}^{(\alpha)}$ and $\mathbf{b}^{(\alpha)}$ to $\boldsymbol{\theta}_t$. For the PF derivations of $\hat{d}_t^{(3)}$, we modify (2.41) according to the definition in (2.12). Furthermore, we modify $\boldsymbol{\theta}_t$ by removing $\mathbf{W}^{(\alpha)}$, $\mathbf{R}^{(\alpha)}$, $\mathbf{b}^{(\alpha)}$, $\mathbf{W}^{(o)}$, $\mathbf{R}^{(o)}$ and $\mathbf{b}^{(o)}$ from its definition for $\hat{d}_t^{(2)}$.

Theorem 2.1.1. Let \mathbf{a}_t be the state vector such that

$$\sup_{\mathbf{a}_t} |\mathbf{a}_t|^4 p(d_t|\mathbf{a}_t) < K_t, \quad (2.48)$$

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

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

Proof of Theorem 2.1.1. From [58], we have

$$\mathbf{E} \left[\left| \mathbf{E}[\pi(\mathbf{a}_t)|d_{1:t}] - \sum_{i=1}^N \omega_t^i \pi(\mathbf{a}_t^i) \right|^4 \right] \leq C_t \frac{\|\pi\|_{t,4}^4}{N^2}, \quad (2.49)$$

where

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

$\pi \in B_t^4$, i.e., a class of functions with certain properties described in [58], and C_t represents a finite constant independent of N . With (2.48), $\pi(\mathbf{a}_t) = \mathbf{a}_t$ satisfies the conditions of B_t^4 . Therefore, applying $\pi(\mathbf{a}_t) = \mathbf{a}_t$ to (2.49) and then evaluating (2.49) as N goes to infinity concludes our proof. \square

This theorem provides a convergence result under (2.48). The inequality in (2.48) implies that the conditional distribution of the observations, i.e., $p(d_t | \mathbf{a}_t)$, decays faster than \mathbf{a}_t increases [58]. Since generic distributions usually decrease exponentially, e.g., Gaussian distribution, or they are nonzero only for bounded intervals, (2.48) is not a strict assumption for \mathbf{a}_t . Hence, we can conclude that Theorem 2.1.1 can be employed for most cases.

According to update equations in (2.40), (2.41), (2.45) and (2.46), each particles cost $O(m^2 + mp)$ due to the matrix vector multiplications in (2.40) and (2.41), and this results in $O(N(m^2 + mp))$ computational complexity to update all particles.

Algorithm 1 Online Training based on the PF Algorithm

- 1: **for** $i = 1 : N$ **do**
 - 2: Draw $\mathbf{a}_t^i \sim p(\mathbf{a}_t | \mathbf{a}_{t-1}^i)$
 - 3: Assign w_t^i according to (2.45)
 - 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 (2.47)
 - 10: **if** $N_{eff} < N_T$ **then** $\%N_T$ is a threshold for N_{eff}
 - 11: Apply the resampling algorithm in [25]
 - 12: Obtain new pairs $\{\bar{\mathbf{a}}_t^i, \bar{\omega}_t^i\}_{i=1}^N$, where $\bar{w}_t^i = 1/N, \forall i$
 - 13: **end if**
 - 14: Using $\{\bar{\mathbf{a}}_t^i, \bar{\omega}_t^i\}_{i=1}^N$, compute the estimate according to (2.46)
-

2.2 Simulations

In this section, we illustrate the performances of our algorithms on different benchmark real datasets under various scenarios. We first consider the regression performance for real life datasets such as kinematic [59], elevators [60], bank [61] and pumadyn [60]. We then consider the regression performance for financial datasets, e.g., Alcoa stock price [62] and Hong Kong exchange rate data [63]. 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 (2.10), (2.11) and (2.12).

Throughout this section, “Architecture 1” represents the LSTM network with (2.10) as the final estimate equation, similarly “Architecture 2” represents the LSTM network with (2.11) and “Architecture 3” represents the LSTM network with (2.12).

2.2.1 Real Life Datasets

In this subsection, we evaluate the performances of the algorithms for the real life datasets. We first evaluate the performances of the algorithms for the kinematic dataset [59]. We then examine the effect of the number of particles on the convergence rate of the PF based algorithm using the same dataset. Furthermore, in order to illustrate the effects of model size while keeping the computation time same, we perform another experiment on the same dataset for the PF based algorithm. Finally, we consider three benchmark real datasets, i.e., elevators [60], bank [61] and pumadyn [60], to evaluate the regression performances of our algorithms.

We first consider the kinematic dataset [59], i.e., a simulation of 8 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 reaches their maximum performance for fair comparison. To provide this fair setup, we have the following parameters. For this dataset, the input vector is $\mathbf{x}_t \in \mathbb{R}^8$ 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 $N = 1500$. Additionally, we choose $\boldsymbol{\eta}_t$ and ξ_t as zero mean Gaussian random variables with the covariance $\text{Cov}[\boldsymbol{\eta}_t] = 0.01\mathbf{I}$ and the variance $\text{Var}[\xi_t] = 0.25$, respectively. For the EKF based algorithm, we choose the initial error covariance as $\boldsymbol{\Sigma}_{0|0} = 0.01\mathbf{I}$. Moreover, we choose $\mathbf{Q}_t = 0.01\mathbf{I}$ and $R_t = 0.25$. For the SGD based algorithm, we set the learning rate as $\mu = 0.03$. As seen in Fig. 2.2, the PF based algorithm converges to a much smaller final MSE level, 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 dataset, where we use the same setup except the number of particles. In Fig. 2.3, 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 to 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 dataset in order to observe the effects of model size while keeping the computation time the same. To provide this setup, we choose four different the output dimension, 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. 2.4, 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. 2.4 matches with our interpretation for the PF based algorithm.

Other than the kinematic dataset, we also consider the elevators [60], bank [61] and pumadyn [60] datasets. For all of these datasets, we again select a fixed architecture, i.e., Architecture 1. Additionally, 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 dataset. The elevators dataset is obtained from the procedure that is related to controlling a F16 aircraft and our aim is to predict the variable that expresses the actions of the aircraft. For this dataset, we have $\mathbf{x}_t \in \mathbb{R}^{18}$ 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 dataset case except that we choose $N = 100$, $\mathbf{Q}_t = 0.0016\mathbf{I}$, $\text{Cov}[\boldsymbol{\eta}_t] = 0.0016\mathbf{I}$ and $\mu = 0.7$. Moreover, the pumadyn dataset 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 $\mathbf{x}_t \in \mathbb{R}^{32}$ and we set the output dimension of the neural network as $m = 32$. Additionally, 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 dataset case. Finally, the bank dataset 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 $\mathbf{x}_t \in \mathbb{R}^{32}$ 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 dataset case. As shown in Table 2.2, the PF based algorithm achieves a smaller time accumulated error value while consuming less training time compared to its competitors, therefore, it has superior performance compared to the other algorithms in these real life tasks.

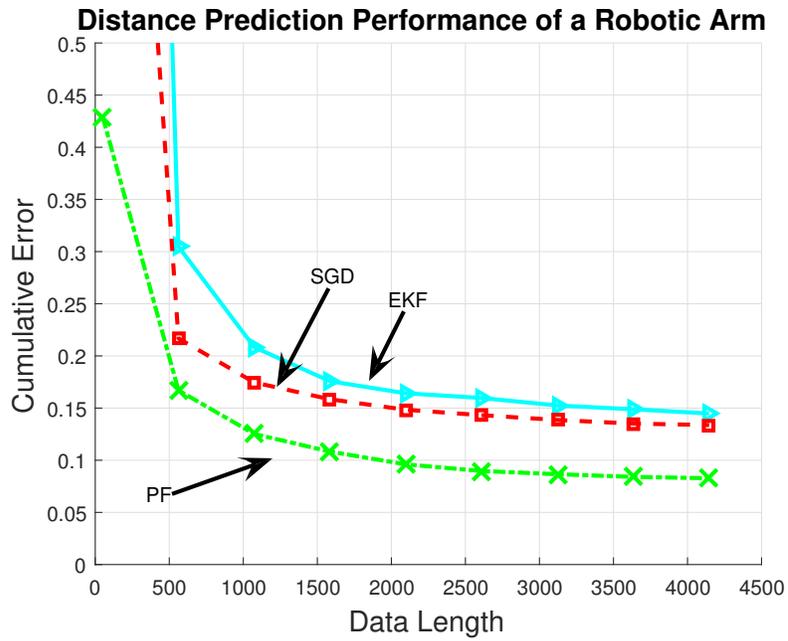


Figure 2.2: Sequential prediction performances of the algorithms for the kinematic dataset.

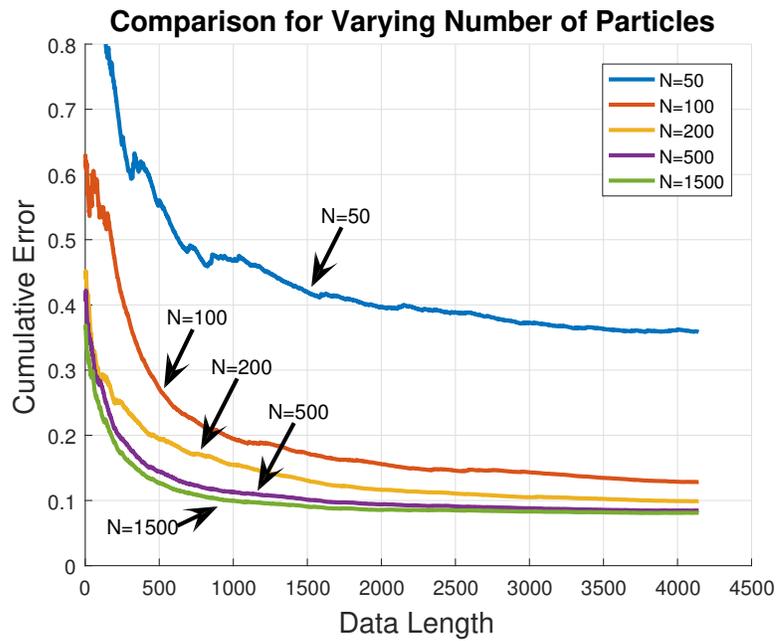


Figure 2.3: Comparison of the PF based algorithm with different number of particles for the kinematic dataset.

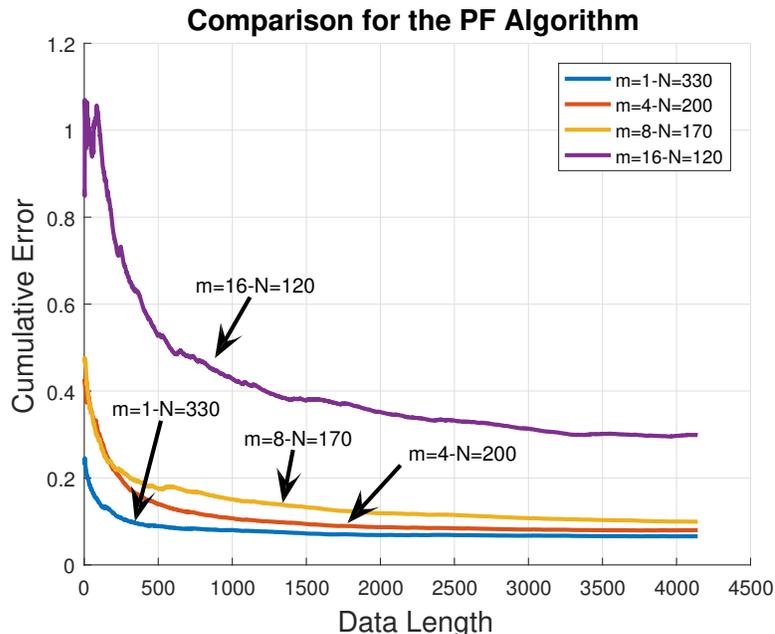


Figure 2.4: Comparison of the PF based algorithm with different N - m combinations for the kinematic dataset. Note that all the combinations in this figure has the same computation time.

2.2.2 Financial Datasets

In this subsection, we evaluate the performances of the algorithms under two different financial scenarios. We first consider the Alcoa stock price dataset [62], which contains the daily stock price values. Our goal is to predict the future prices by examining the past prices. As in the previous subsection, 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 dataset, we choose to examine the price of the previous five days, so that we have the input $\mathbf{x}_t \in \mathbb{R}^5$ 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 $N = 2000$. Additionally, we choose $\boldsymbol{\eta}_t$ and ξ_t as zero mean Gaussian random variables with $\text{Cov}[\boldsymbol{\eta}_t] = 0.0036\mathbf{I}$ and $\text{Var}[\xi_t] = 0.01$. For the EKF based algorithm, we choose

Algorithms Datasets	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

Table 2.2: Time accumulated errors and the corresponding training times (in seconds) of the LSTM based algorithms for the elevators, pumadyn and bank datasets. Note that here, we use a computer with i5-6400 processor, 2.7 GHz CPU and 16 GB RAM.

$\Sigma_{0|0} = 0.0036\mathbf{I}$, $\mathbf{Q}_t = 0.0036\mathbf{I}$ and $R_t = 0.01$. For the SGD based algorithm, we set the learning rate as $\mu = 0.1$. With these fair settings, Fig. 2.5 illustrates that the PF based algorithm converges much faster.

Aside from the Alcoa stock price dataset, we also consider the Hong Kong exchange rate dataset [63], for which we have the amount of Hong Kong dollars that one is able to buy for one U.S. dollar 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 rate of the algorithms are the same. We use the same parameters with the Alcoa stock price dataset case except $\mathbf{Q}_t = 0.0004\mathbf{I}$ and $\text{Cov}[\boldsymbol{\eta}_t] = 0.0004\mathbf{I}$. In this case, Fig. 2.6 shows that the PF based algorithm converges to a much smaller steady state error value.

2.2.3 LSTM and GRU Networks

In this subsection, 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

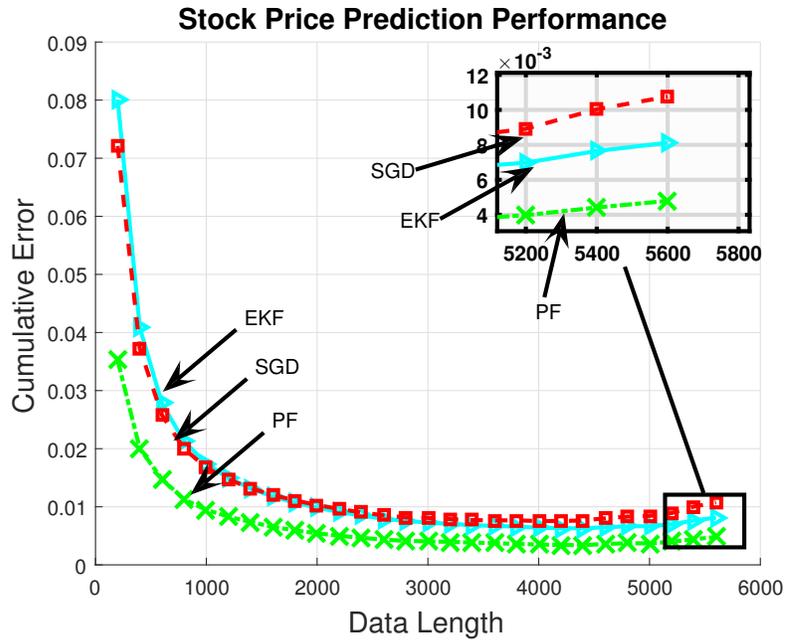


Figure 2.5: Future price prediction performances of the algorithms for the Alcoa stock price dataset.

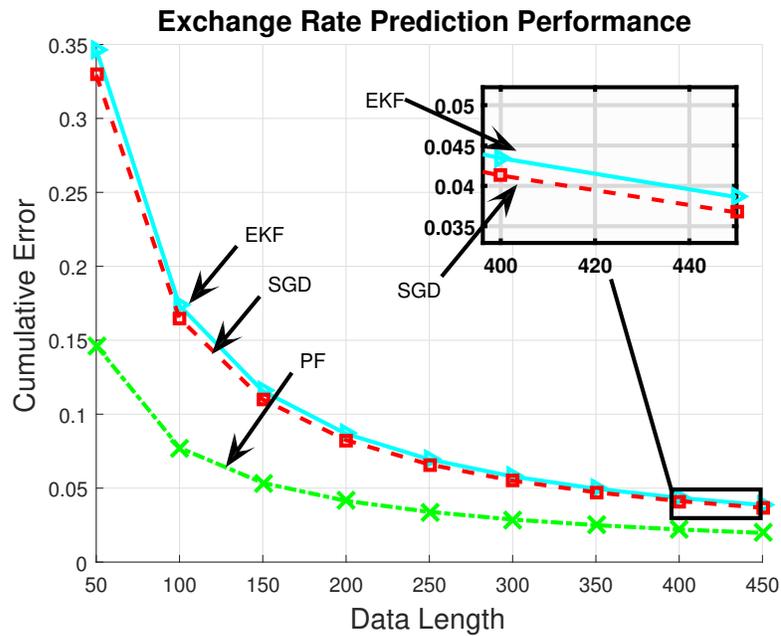


Figure 2.6: Exchange rate prediction performances of the algorithms for the Hong Kong exchange rate dataset.

described by the following set of equations [22]:

$$\tilde{\mathbf{z}}_t = \sigma \left(\mathbf{W}^{(\tilde{\mathbf{z}})} \mathbf{x}_t + \mathbf{R}^{(\tilde{\mathbf{z}})} \mathbf{y}_{t-1} \right) \quad (2.50)$$

$$\mathbf{r}_t = \sigma \left(\mathbf{W}^{(r)} \mathbf{x}_t + \mathbf{R}^{(r)} \mathbf{y}_{t-1} \right) \quad (2.51)$$

$$\tilde{\mathbf{y}}_t = g \left(\mathbf{W}^{(y)} \mathbf{x}_t + \mathbf{r}_t \odot (\mathbf{R}^{(y)} \mathbf{y}_{t-1}) \right) \quad (2.52)$$

$$\mathbf{y}_t = \tilde{\mathbf{y}}_t \odot \tilde{\mathbf{z}}_t + \mathbf{y}_{t-1} \odot (\mathbf{1} - \tilde{\mathbf{z}}_t), \quad (2.53)$$

where $\mathbf{x}_t \in \mathbb{R}^p$ is the input vector and $\mathbf{y}_t \in \mathbb{R}^m$ is the output vector. The functions $g(\cdot)$ and $\sigma(\cdot)$ are set to the hyperbolic tangent and sigmoid functions, respectively. For the coefficient matrices, we have $\mathbf{W}^{(\tilde{\mathbf{z}})} \in \mathbb{R}^{m \times p}$, $\mathbf{R}^{(\tilde{\mathbf{z}})} \in \mathbb{R}^{m \times m}$, $\mathbf{W}^{(r)} \in \mathbb{R}^{m \times p}$, $\mathbf{R}^{(r)} \in \mathbb{R}^{m \times m}$, $\mathbf{W}^{(y)} \in \mathbb{R}^{m \times p}$, $\mathbf{R}^{(y)} \in \mathbb{R}^{m \times m}$. Here, $\tilde{\mathbf{z}}_t$ and \mathbf{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. Hence, they have significant differences. To compare them, we use the Hong Kong exchange rate dataset as in the previous subsection. 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 rate of the algorithms are the same. With this fair setup, Fig. 2.7a, 2.7b and 2.7c show 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.

2.2.4 Different Regression Architectures

In this subsection, we compare the performances of different LSTM based regression architectures. For this purpose, we use the Hong Kong exchange rate dataset as in the previous subsection. For a fair comparison, we select the parameters

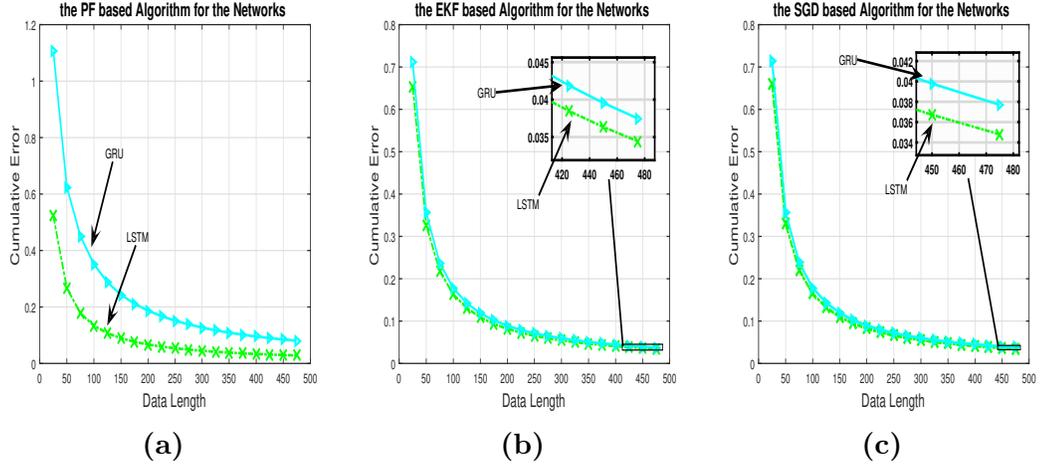


Figure 2.7: Comparison of the LSTM and GRU architectures in terms of regression error performance for (a) the PF based algorithm, (b) the EKF based algorithm and (c) the SGD based algorithm.

Architectures Algorithms	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

Table 2.3: Time accumulated errors of the LSTM based regression algorithms described in (2.10), (2.11) and (2.12) for each algorithm.

such that the convergence rate of the algorithms are the same. We choose the same parameter with the previous subsection except $\Sigma_{0|0} = 0.01\mathbf{I}$. Under this fair setup, Table 2.3 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 has also the direct contribution of the regression vector, it has a greater error value compared to 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.

Chapter 3

Online Training of LSTM Networks in Distributed Systems

In this chapter, we consider a network of K nodes. In this network, we declare two nodes that can exchange information as neighbors and denote the neighborhood of each node k as \mathcal{N}_k that also includes the node k , i.e., $k \in \mathcal{N}_k$. At each node k , we sequentially receive $\{d_{k,t}\}_{t \geq 1}$, $d_{k,t} \in \mathbb{R}$ and matrices, $\{\mathbf{X}_{k,t}\}_{t \geq 1}$, defined as $\mathbf{X}_{k,t} = [\mathbf{x}_{k,t}^{(1)} \mathbf{x}_{k,t}^{(2)} \dots \mathbf{x}_{k,t}^{(m_t)}]$, where $\mathbf{x}_{k,t}^{(l)} \in \mathbb{R}^p$, $\forall l \in \{1, 2, \dots, m_t\}$ and $m_t \in \mathbb{Z}^+$ is the number of columns in $\mathbf{X}_{k,t}$, which can change with respect to t . In our network, each node k aims to learn a certain relation between the desired value $d_{k,t}$ and matrix $\mathbf{X}_{k,t}$. After observing $\mathbf{X}_{k,t}$ and $d_{k,t}$, each node k first updates its belief about the relation and then exchanges an updated information with its neighbors. After receiving $\mathbf{X}_{k,t+1}$, each node k estimates the next signal $d_{k,t+1}$ as $\hat{d}_{k,t+1}$. Based on $d_{k,t+1}$, each node k suffers the loss $l(d_{k,t+1}, \hat{d}_{k,t+1})$ at time instance $t+1$. This framework models a wide range of applications in the machine learning and signal processing literatures, e.g., sentiment analysis [26]. As an example, in tweet emotion recognition application [26], each $\mathbf{X}_{k,t}$ corresponds to a tweet, i.e., the t^{th} tweet at the node (processing unit) k . For the t^{th} tweet at the node k , one can construct $\mathbf{X}_{k,t}$ by finding word2vec representation of each word, i.e., $\mathbf{x}_{k,t}^{(l)}$ for the l^{th} word. After receiving $d_{k,t}$, i.e., the desired emotion label for the t^{th} tweet at the node k , each node k first updates its belief about the relation between the

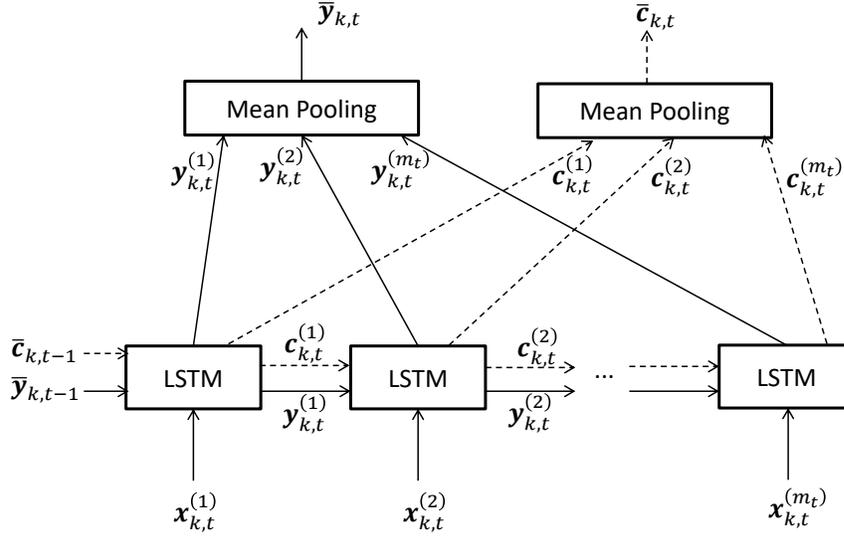


Figure 3.1: Detailed schematic of each node k in our network.

tweet and its emotion label, and then exchanges information, e.g., the trained system parameters, with its neighboring units to estimate the next label.

Here, each node k generates an estimate $\hat{d}_{k,t}$ using the LSTM architecture. Although there exist different variants of LSTM, we use the most widely used variant [12], i.e., the LSTM architecture without peephole connections. The input $\mathbf{X}_{k,t}$ is first fed to the LSTM architecture as illustrated in Fig. 3.1, where the internal equations are given as [12]:

$$\mathbf{i}_{k,t}^{(l)} = \sigma(\mathbf{W}_k^{(i)} \mathbf{x}_{k,t}^{(l)} + \mathbf{R}_k^{(i)} \mathbf{y}_{k,t}^{(l-1)} + \mathbf{b}_k^{(i)}) \quad (3.1)$$

$$\mathbf{f}_{k,t}^{(l)} = \sigma(\mathbf{W}_k^{(f)} \mathbf{x}_{k,t}^{(l)} + \mathbf{R}_k^{(f)} \mathbf{y}_{k,t}^{(l-1)} + \mathbf{b}_k^{(f)}) \quad (3.2)$$

$$\mathbf{c}_{k,t}^{(l)} = \mathbf{i}_{k,t}^{(l)} \odot g(\mathbf{W}_k^{(z)} \mathbf{x}_{k,t}^{(l)} + \mathbf{R}_k^{(z)} \mathbf{y}_{k,t}^{(l-1)} + \mathbf{b}_k^{(z)}) + \mathbf{f}_{k,t}^{(l)} \odot \mathbf{c}_{k,t}^{(l-1)} \quad (3.3)$$

$$\mathbf{o}_{k,t}^{(l)} = \sigma(\mathbf{W}_k^{(o)} \mathbf{x}_{k,t}^{(l)} + \mathbf{R}_k^{(o)} \mathbf{y}_{k,t}^{(l-1)} + \mathbf{b}_k^{(o)}) \quad (3.4)$$

$$\mathbf{y}_{k,t}^{(l)} = \mathbf{o}_{k,t}^{(l)} \odot h(\mathbf{c}_{k,t}^{(l)}), \quad (3.5)$$

where $\mathbf{x}_{k,t}^{(l)} \in \mathbb{R}^p$ is the input vector, $\mathbf{y}_{k,t}^{(l)} \in \mathbb{R}^n$ is the output vector and $\mathbf{c}_{k,t}^{(l)} \in \mathbb{R}^n$ is the state vector for the l^{th} LSTM unit. Moreover, $\mathbf{o}_{k,t}^{(l)}$, $\mathbf{f}_{k,t}^{(l)}$ and $\mathbf{i}_{k,t}^{(l)}$ represent the output, forget and input gates, respectively. $g(\cdot)$ and $h(\cdot)$ are set to the hyperbolic tangent function and apply vectors pointwise. Likewise, $\sigma(\cdot)$ is the pointwise sigmoid function. The operation \odot represents the elementwise multiplication of two vectors of the same size. As the coefficient matrices and the weight vectors

of the LSTM architecture, we have $\mathbf{W}_k^{(\cdot)}$, $\mathbf{R}_k^{(\cdot)}$ and $\mathbf{b}_k^{(\cdot)}$, where the sizes are chosen according to the input and output vectors. Given the outputs of LSTM for each column of $\mathbf{X}_{k,t}$ as seen in Fig. 3.1, we generate the estimate for each node k as follows

$$\hat{d}_{k,t} = \mathbf{w}_{k,t}^T \bar{\mathbf{y}}_{k,t}, \quad (3.6)$$

where $\mathbf{w}_{k,t} \in \mathbb{R}^n$ is a vector of the regression coefficients and $\bar{\mathbf{y}}_{k,t} \in \mathbb{R}^n$ is a vector obtained by taking average of the LSTM outputs for each column of $\mathbf{X}_{k,t}$, i.e., known as the mean pooling method, as described in Fig. 3.1.

Remark 3.0.1. *In (3.6), we use the mean pooling method to generate $\bar{\mathbf{y}}_{k,t}$. One can also use the other pooling methods by changing the calculation of $\bar{\mathbf{y}}_{k,t}$ and then generate the estimate as in (3.6). As an example, for the max and last pooling methods, we use $\bar{\mathbf{y}}_{k,t} = \max_i \mathbf{y}_{k,t}^{(i)}$ and $\bar{\mathbf{y}}_{k,t} = \mathbf{y}_{k,t}^{(m_t)}$, respectively. All our derivations hold for these pooling methods and the other LSTM architectures. We provide the required updates for different LSTM architectures in the next section.*

3.1 Online Distributed Training Algorithms

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

Considering our model in Fig. 3.1 and the LSTM equations in (3.1)–(3.5), we have the following nonlinear state space form for each node k

$$\bar{\mathbf{c}}_{k,t} = \Omega(\bar{\mathbf{c}}_{k,t-1}, \mathbf{X}_{k,t}, \bar{\mathbf{y}}_{k,t-1}) \quad (3.7)$$

$$\bar{\mathbf{y}}_{k,t} = \Theta(\bar{\mathbf{c}}_{k,t}, \mathbf{X}_{k,t}, \bar{\mathbf{y}}_{k,t-1}) \quad (3.8)$$

$$\boldsymbol{\theta}_{k,t} = \boldsymbol{\theta}_{k,t-1} \quad (3.9)$$

$$d_{k,t} = \mathbf{w}_{k,t}^T \bar{\mathbf{y}}_{k,t} + \varepsilon_{k,t}, \quad (3.10)$$

where $\Omega(\cdot)$ and $\Theta(\cdot)$ represent the nonlinear mappings performed by the consecutive LSTM units and the mean pooling operation as illustrated in Fig. 3.1,

and $\boldsymbol{\theta}_{k,t} \in \mathbb{R}^{n_\theta}$ is a parameter vector consisting of $\{\mathbf{w}_k, \mathbf{W}_k^{(z)}, \mathbf{R}_k^{(z)}, \mathbf{b}_k^{(z)}, \mathbf{W}_k^{(i)}, \mathbf{R}_k^{(i)}, \mathbf{b}_k^{(i)}, \mathbf{W}_k^{(f)}, \mathbf{R}_k^{(f)}, \mathbf{b}_k^{(f)}, \mathbf{W}_k^{(o)}, \mathbf{R}_k^{(o)}, \mathbf{b}_k^{(o)}\}$, where $n_\theta = 4n(n+p) + 5n$. Since the LSTM parameters are the states of the network to be estimated, we also include the static equation (3.9) as our state. Furthermore, $\varepsilon_{k,t}$ represents the error in observations and it is a zero mean Gaussian random variable with variance $R_{k,t}$.

Remark 3.1.1. *We can also apply the introduced algorithms to different implementations of the LSTM architecture [12]. For this purpose, we modify the function $\Omega(\cdot)$ and $\Theta(\cdot)$ in (3.7) and (3.8) according to the chosen LSTM architecture. We also alter $\boldsymbol{\theta}_{k,t}$ in (3.9) by adding or removing certain parameters according to the chosen LSTM architecture.*

3.1.1 Online Training Using the DEKF Algorithm

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

The EKF algorithm is based on the assumption that the state distribution given the observations is Gaussian [19]. To meet this assumption, we introduce Gaussian noise to (3.7), (3.8) and (3.9). By this, we have the following model for each node k

$$\begin{bmatrix} \bar{\mathbf{c}}_{k,t} \\ \bar{\mathbf{y}}_{k,t} \\ \boldsymbol{\theta}_{k,t} \end{bmatrix} = \begin{bmatrix} \Omega(\bar{\mathbf{c}}_{k,t-1}, \mathbf{X}_{k,t}, \bar{\mathbf{y}}_{k,t-1}) \\ \Theta(\bar{\mathbf{c}}_{k,t}, \mathbf{X}_{k,t}, \bar{\mathbf{y}}_{k,t-1}) \\ \boldsymbol{\theta}_{k,t-1} \end{bmatrix} + \begin{bmatrix} \mathbf{e}_{k,t} \\ \boldsymbol{\epsilon}_{k,t} \\ \mathbf{v}_{k,t} \end{bmatrix} \quad (3.11)$$

$$d_{k,t} = \mathbf{w}_{k,t}^T \bar{\mathbf{y}}_{k,t} + \varepsilon_{k,t}, \quad (3.12)$$

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

3.1.1.1 Online Training with the EKF Algorithm

In this subsection, we derive the online training method based on the EKF algorithm when we do not allow communication between the neighbors. Since the system in (3.11) and (3.12) is already in a nonlinear state space form, we can directly apply the EKF algorithm [19] as follows

Time Update:

$$\bar{\mathbf{c}}_{k,t|t-1} = \Omega(\bar{\mathbf{c}}_{k,t-1|t-1}, \mathbf{X}_{k,t}, \bar{\mathbf{y}}_{k,t-1|t-1}) \quad (3.13)$$

$$\bar{\mathbf{y}}_{k,t|t-1} = \Theta(\bar{\mathbf{c}}_{k,t-1|t-1}, \mathbf{X}_{k,t}, \bar{\mathbf{y}}_{k,t-1|t-1}) \quad (3.14)$$

$$\boldsymbol{\theta}_{k,t|t-1} = \boldsymbol{\theta}_{k,t-1|t-1} \quad (3.15)$$

$$\boldsymbol{\Sigma}_{k,t|t-1} = \mathbf{F}_{k,t-1} \boldsymbol{\Sigma}_{k,t-1|t-1} \mathbf{F}_{k,t-1}^T + \mathbf{Q}_{k,t-1} \quad (3.16)$$

Measurement Update:

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

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

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

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

$$\mathbf{H}_{k,t}^T = \begin{bmatrix} \frac{\partial \hat{d}_{k,t}}{\partial \bar{\mathbf{c}}} & \frac{\partial \hat{d}_{k,t}}{\partial \bar{\mathbf{y}}} & \frac{\partial \hat{d}_{k,t}}{\partial \boldsymbol{\theta}} \end{bmatrix} \Bigg|_{\substack{\bar{\mathbf{c}} = \bar{\mathbf{c}}_{k,t|t-1} \\ \bar{\mathbf{y}} = \bar{\mathbf{y}}_{k,t|t-1} \\ \boldsymbol{\theta} = \boldsymbol{\theta}_{k,t|t-1}} \quad (3.17)$$

and

$$\mathbf{F}_{k,t} = \begin{bmatrix} \frac{\partial \Omega(\bar{\mathbf{c}}, \mathbf{X}_{k,t}, \bar{\mathbf{y}})}{\partial \bar{\mathbf{c}}} & \frac{\partial \Omega(\bar{\mathbf{c}}, \mathbf{X}_{k,t}, \bar{\mathbf{y}})}{\partial \bar{\mathbf{y}}} & \frac{\partial \Omega(\bar{\mathbf{c}}, \mathbf{X}_{k,t}, \bar{\mathbf{y}})}{\partial \boldsymbol{\theta}} \\ \frac{\partial \Theta(\bar{\mathbf{c}}, \mathbf{X}_{k,t}, \bar{\mathbf{y}})}{\partial \bar{\mathbf{c}}} & \frac{\partial \Theta(\bar{\mathbf{c}}, \mathbf{X}_{k,t}, \bar{\mathbf{y}})}{\partial \bar{\mathbf{y}}} & \frac{\partial \Theta(\bar{\mathbf{c}}, \mathbf{X}_{k,t}, \bar{\mathbf{y}})}{\partial \boldsymbol{\theta}} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \Bigg|_{\substack{\bar{\mathbf{c}} = \bar{\mathbf{c}}_{k,t|t} \\ \bar{\mathbf{y}} = \bar{\mathbf{y}}_{k,t|t} \\ \boldsymbol{\theta} = \boldsymbol{\theta}_{k,t|t}}, \quad (3.18)$$

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

3.1.1.2 Online Training with the DEKF Algorithm

In this subsection, we introduce our online training method based on the DEKF algorithm for the network described by (3.11) and (3.12). In our network of K nodes, we denote the number of neighbors for the node k as η_k , i.e., also called as the degree of the node k [32]. With this structure, the time update equations in (3.13)–(3.16) still hold for each node k . However, since we have information exchange between the neighbors, the measurement update equations of each node k adopt the iterative scheme [32] as the following.

For the node k at time t :

$$\phi_{k,t} \leftarrow [\bar{\mathbf{c}}_{k,t|t-1}^T \bar{\mathbf{y}}_{k,t|t-1}^T \boldsymbol{\theta}_{k,t|t-1}^T]^T$$

$$\Phi_{k,t} \leftarrow \Sigma_{k,t|t-1}$$

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

$$R \leftarrow \mathbf{H}_{l,t}^T \Phi_{k,t} \mathbf{H}_{l,t} + R_{l,t}$$

$$\phi_{k,t} \leftarrow \phi_{k,t} + \Phi_{k,t} \mathbf{H}_{l,t} R^{-1} (d_{l,t} - \mathbf{w}_{k,t|t-1}^T \bar{\mathbf{y}}_{k,t|t-1})$$

$$\Phi_{k,t} \leftarrow \Phi_{k,t} - \Phi_{k,t} \mathbf{H}_{l,t} R^{-1} \mathbf{H}_{l,t}^T \Phi_{k,t}.$$

Now, we update the state and covariance matrix estimate as

$$\begin{aligned} \Sigma_{k,t|t} &= \Phi_{k,t} \\ [\bar{\mathbf{c}}_{k,t|t}^T \bar{\mathbf{y}}_{k,t|t}^T \boldsymbol{\theta}_{k,t|t}^T]^T &= \sum_{l \in \mathcal{N}_k} c(k, l) \phi_{l,t}, \end{aligned}$$

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

$$c(k, l) = \begin{cases} 1/\max(\eta_k, \eta_l) & \text{if } l \in \mathcal{N}_k/k \\ 1 - \sum_{l \in \mathcal{N}_k/k} c(k, l) & \text{if } k = l \\ 0 & \text{if } l \notin \mathcal{N}_k \end{cases}. \quad (3.19)$$

With these steps, we can update all the nodes in our network as illustrated in Algorithm 2.

According to the procedure in Algorithm 2, the computational complexity of our training method results in $\mathcal{O}(\eta_k(n^8 + n^4 p^4))$ computations at each node k due to matrix and vector multiplications on lines 8 and 19 as shown in Table 3.1.

Algorithm 2 Training based on the DEKF Algorithm

- 1: According to (3.17), compute $\mathbf{H}_{k,t}$, $\forall k \in \{1, 2, \dots, K\}$
 - 2: **for** $k = 1 : K$ **do**
 - 3: $\phi_{k,t} \leftarrow [\bar{\mathbf{c}}_{k,t|t-1}^T \bar{\mathbf{y}}_{k,t|t-1}^T \boldsymbol{\theta}_{k,t|t-1}^T]^T$
 - 4: $\Phi_{k,t} \leftarrow \Sigma_{k,t|t-1}$
 - 5: **for** $l \in \mathcal{N}_k$ **do**
 - 6: $R \leftarrow \mathbf{H}_{l,t}^T \Phi_{k,t} \mathbf{H}_{l,t} + R_{l,t}$
 - 7: $\phi_{k,t} \leftarrow \phi_{k,t} + \Phi_{k,t} \mathbf{H}_{l,t} R^{-1} (d_{l,t} - \mathbf{w}_{k,t|t-1}^T \bar{\mathbf{y}}_{k,t|t-1})$
 - 8: $\Phi_{k,t} \leftarrow \Phi_{k,t} - \Phi_{k,t} \mathbf{H}_{l,t} R^{-1} \mathbf{H}_{l,t}^T \Phi_{k,t}$
 - 9: **end for**
 - 10: **end for**
 - 11: **for** $k = 1 : K$ **do**
 - 12: Using (3.19), calculate $c(k, l)$, $\forall l \in \mathcal{N}_k$
 - 13: $[\bar{\mathbf{c}}_{k,t|t}^T \bar{\mathbf{y}}_{k,t|t}^T \boldsymbol{\theta}_{k,t|t}^T]^T \leftarrow \sum_{l \in \mathcal{N}_k} c(k, l) \phi_{l,t}$
 - 14: $\Sigma_{k,t|t} \leftarrow \Phi_{k,t}$
 - 15: According to (3.18), compute $\mathbf{F}_{k,t}$
 - 16: $\bar{\mathbf{c}}_{k,t+1|t} \leftarrow \Omega(\bar{\mathbf{c}}_{k,t|t}, \mathbf{X}_{k,t}, \bar{\mathbf{y}}_{k,t|t})$
 - 17: $\bar{\mathbf{y}}_{k,t+1|t} \leftarrow \Theta(\bar{\mathbf{c}}_{k,t+1|t}, \mathbf{X}_{k,t}, \bar{\mathbf{y}}_{k,t|t})$
 - 18: $\boldsymbol{\theta}_{k,t+1|t} \leftarrow \boldsymbol{\theta}_{k,t|t}$
 - 19: $\Sigma_{k,t+1|t} \leftarrow \mathbf{F}_{k,t} \Sigma_{k,t|t} \mathbf{F}_{k,t}^T + \mathbf{Q}_{k,t}$
 - 20: **end for**
-

3.1.2 Online Training Using the DPF Algorithm

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

The PF algorithm only requires the independence of the noise samples in (3.11) and (3.12). Thus, we modify our system in (3.11) and (3.12) for the node k as follows

$$\mathbf{a}_{k,t} = \varphi(\mathbf{a}_{k,t-1}, \mathbf{X}_{k,t}) + \boldsymbol{\gamma}_{k,t} \quad (3.20)$$

$$d_{k,t} = \mathbf{w}_{k,t}^T \bar{\mathbf{y}}_{k,t} + \varepsilon_{k,t}, \quad (3.21)$$

where $\boldsymbol{\gamma}_{k,t}$ and $\varepsilon_{k,t}$ are independent state and measurement noise samples, respectively, $\varphi(\cdot, \cdot)$ is the nonlinear mapping in (3.11) and $\mathbf{a}_{k,t} \triangleq [\bar{\mathbf{c}}_{k,t}^T \bar{\mathbf{y}}_{k,t}^T \boldsymbol{\theta}_{k,t}^T]^T$.

3.1.2.1 Online Training with the PF Algorithm

For the system in (3.20) and (3.21), our aim is to obtain $\mathbf{E}[\mathbf{a}_{k,t}|d_{k,1:t}]$, i.e., the optimal estimate for the hidden state in the MSE sense. To achieve this, we first obtain posterior distribution of the states, i.e., $p(\mathbf{a}_{k,t}|d_{k,1:t})$. Based on the posterior density function, we then calculate the conditional mean estimate. In order to obtain the posterior distribution, we apply the PF algorithm [21].

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

$$p(\mathbf{a}_{k,t}|d_{k,1:t}) \approx \sum_{i=1}^N \omega_{k,t}^i \delta(\mathbf{a}_{k,t} - \mathbf{a}_{k,t}^i). \quad (3.22)$$

Sampling from the desired distribution $p(\mathbf{a}_{k,t}|d_{k,1:t})$ is intractable in general so that we obtain the samples from $q(\mathbf{a}_{k,t}|d_{k,1:t})$, which is called as importance function [21]. To calculate the weights in (3.22), we use the following formula

$$w_{k,t}^i \propto \frac{p(\mathbf{a}_{k,t}^i|d_{k,1:t})}{q(\mathbf{a}_{k,t}^i|d_{k,1:t})}, \text{ where } \sum_{i=1}^N \omega_{k,t}^i = 1. \quad (3.23)$$

We can factorize (3.23) such that we obtain the following recursive formula [21]

$$\omega_{k,t}^i \propto \frac{p(d_{k,t}|\mathbf{a}_{k,t}^i)p(\mathbf{a}_{k,t}^i|\mathbf{a}_{k,t-1}^i)}{q(\mathbf{a}_{k,t}^i|\mathbf{a}_{k,t-1}^i, d_{k,t})} \omega_{k,t-1}^i. \quad (3.24)$$

In (3.24), we choose the importance function so that the variance of the weights is minimized. By this, we obtain particles that have nonnegligible weights and significantly contribute to (3.22) [21]. In this sense, since $p(\mathbf{a}_{k,t}^i|\mathbf{a}_{k,t-1}^i)$ provides a small variance for the weights [21], we choose it as our importance function. With this choice, we alter (3.24) as follows

$$\omega_{k,t}^i \propto p(d_{k,t}|\mathbf{a}_{k,t}^i) \omega_{k,t-1}^i. \quad (3.25)$$

By (3.22) and (3.25), we obtain the state estimate as follows

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

Although we choose the importance function to reduce the variance of the weights, the variance inevitably increases over time [21]. Hence, we apply the resampling algorithm introduced in [21] such that we eliminate the particles with small weights and prevent the variance from increasing.

3.1.2.2 Online Training with the DPF Algorithm

In this subsection, we introduce our online training method based on the DPF algorithm when the nodes share information with their neighbors. We employ the Markov Chain Distributed Particle Filter (MCDPF) algorithm [35] to train our distributed system. In the MCDPF algorithm, particles move around the network according to the network topology. In every step, each particle can randomly move to another node in the neighborhood of its current node. While randomly moving, the weight of each particle is updated using $p(d_{k,t}|\mathbf{a}_{k,t})$ at the node k , hence, particles use the observations at different nodes.

Suppose we consider our network as a graph $G = (V, E)$, where the vertices V represent the nodes in our network and the edges E represent the connections between the nodes. In addition to this, we denote the number of visits to each node k in s steps by each particle i as $M^i(k, s)$. Here, each particle moves to one of its neighboring nodes with a certain probability, where the movement probabilities of each node to the other nodes are represented by the adjacency matrix, i.e., denoted as \mathcal{A} . In this framework, at each visit to each node k , each particle multiplies its weight with $p(d_{k,t}|\mathbf{a}_{k,t})^{\frac{2|E(G)|}{s\eta_k}}$ in a run of s steps [35], where $|E(G)|$ is the number of edges in G and η_k is the degree of the node k . From

(3.25), we have the following update for each particle i at the node k after s steps

$$w_{k,t}^i = w_{k,t-1}^i \prod_{j=1}^K p(d_{j,t} | \mathbf{a}_{k,t}^i)^{\frac{2|E(G)|}{s\eta_j} M^i(j,s)}. \quad (3.26)$$

We then calculate the posterior distribution at the node k as

$$p(\mathbf{a}_{k,t} | O_{k,t}) \approx \sum_{i=1}^{N(k)} w_{k,t}^i \delta(\mathbf{a}_{k,t} - \mathbf{a}_{k,t}^i), \quad (3.27)$$

where $O_{k,t}$ represents the observations seen by the particles at the node k until t and $w_{k,t}^i$ is obtained from (3.26). After we obtain (3.27), we calculate our estimate for $\mathbf{a}_{k,t}$ as follows

$$\begin{aligned} \mathbf{E}[\mathbf{a}_{k,t} | O_{k,t}] &= \int \mathbf{a}_{k,t} p(\mathbf{a}_{k,t} | O_{k,t}) d\mathbf{a}_{k,t} \\ &\approx \int \mathbf{a}_{k,t} \sum_{i=1}^{N(k)} \omega_{k,t}^i \delta(\mathbf{a}_{k,t} - \mathbf{a}_{k,t}^i) d\mathbf{a}_{k,t} \\ &= \sum_{i=1}^{N(k)} \omega_{k,t}^i \mathbf{a}_{k,t}^i. \end{aligned} \quad (3.28)$$

We can obtain the estimate for each node using the same procedure as illustrated in Algorithm 3. In Algorithm 3, $N(j)$ represents the number of particles at the node j and $\mathcal{I}_{i \rightarrow j}$ represents the indices of the particles that move from the node i to the node j . Thus, we obtain a distributed training algorithm that guarantees convergence to the optimal centralized parameter estimation as illustrated in Theorem 1.

Theorem 3.1.1. *For each node k , let $\mathbf{a}_{k,t}$ be the bounded state vector with a measurement density function that satisfies the following inequality*

$$0 < p_0 \leq p(d_{k,t} | \mathbf{a}_{k,t}) \leq \|p\|_\infty < \infty, \quad (3.29)$$

where p_0 is a constant and

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

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

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

Proof of Theorem 3.1.1. Using (3.29), from [35], we obtain

$$\mathbf{E}\left[\left(\mathbf{E}[\pi(\mathbf{a}_t)|\{d_{j,1:t}\}_{j=1}^K] - \sum_{i=1}^N \omega_{k,t}^i \pi(\mathbf{a}_{k,t}^i)\right)^2\right] \leq \|\pi\|_\infty^2 \left(C_t \sqrt{U(s,v)} + \sqrt{\frac{\zeta_t}{N}}\right)^2, \quad (3.30)$$

where π is a bounded function, v is the second largest eigenvalue modulus of \mathcal{A} , ζ_t and C_t are time dependent constants and $U(s, v)$ is a function of s as described in [35] such that $U(s, v)$ goes to zero as s goes to infinity. Since the state vector $\mathbf{a}_{k,t}$ is bounded, we can choose $\pi(\mathbf{a}_{k,t}) = \mathbf{a}_{k,t}$. With this choice, evaluating (3.30) as N and s go to infinity yields the results. This concludes our proof. \square

According to the update procedure illustrated in Algorithm 3, the computational complexity of our training method results in $\mathcal{O}(N(k)(n^2 + np))$ computations at each node k due to matrix vector multiplications in (3.20) and (3.21) as shown in Table 3.1.

Algorithm 3 Training based on the DPF Algorithm

- 1: Sample $\{\mathbf{a}_{j,t}^i\}_{i=1}^{N(j)}$ from $p(\mathbf{a}_t|\{\mathbf{a}_{j,t-1}^i\}_{i=1}^{N(j)})$, $\forall j$
 - 2: Set $\{w_{j,t}^i\}_{i=1}^{N(j)} = 1$, $\forall j$
 - 3: **for** s steps **do**
 - 4: Move the particles according to \mathcal{A}
 - 5: **for** $j = 1 : K$ **do**
 - 6: $\{\mathbf{a}_{j,t}^i\}_{i=1}^{N(j)} \leftarrow \bigcup_{l \in \mathcal{N}_j} \{\mathbf{a}_{l,t}^i\}_{i \in \mathcal{I}_{l \rightarrow j}}$
 - 7: $\{w_{j,t}^i\}_{i=1}^{N(j)} \leftarrow \bigcup_{l \in \mathcal{N}_j} \{w_{l,t}^i\}_{i \in \mathcal{I}_{l \rightarrow j}}$
 - 8: $\{w_{j,t}^i\}_{i=1}^{N(j)} \leftarrow \{w_{j,t}^i\}_{i=1}^{N(j)} p(d_{j,t}|\{\mathbf{a}_{j,t}^i\}_{i=1}^{N(j)})^{\frac{2|E(G)|}{s\eta_j}}$
 - 9: **end for**
 - 10: **end for**
 - 11: **for** $j=1:K$ **do**
 - 12: Resample $\{\mathbf{a}_{j,t}^i, w_{j,t}^i\}_{i=1}^{N(j)}$
 - 13: Compute the estimate for node j using (3.28)
 - 14: **end for**
-

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

Table 3.1: Comparison of the computational complexities of the introduced training algorithms for each node k . In this table, we also calculate the computational complexity of the SGD based algorithm by deriving exact gradient equations, however, we omit these calculations.

3.2 Simulations

We evaluate the performance of the introduced algorithms on different benchmark real datasets. We first consider the prediction performance on Hong Kong exchange rate dataset [63]. We then evaluate the regression performance on emotion labelled sentence dataset [64]. For these experiments, to observe the effects of communication among nodes, we also consider the EKF and PF based algorithms without communication over a network of multiple nodes, where each node trains LSTM based on only its observations. Throughout this section, we denote the EKF and PF based algorithms without communication over a network of multiple nodes as “EKF” and “PF”, respectively. Moreover, we denote the EKF and PF based algorithms with communication over a network of multiple nodes as “DEKF” and “DPF”, respectively. We also consider the SGD based algorithm without communication over a network of multiple nodes as a benchmark algorithm and denote it by “SGD”.

We first consider the Hong Kong exchange rate dataset [63]. For this dataset, we have the amount of Hong Kong dollars that can buy one United States dollar on certain days. Our aim is to estimate future exchange rate by using the values in the previous two days. In online applications, one can demand a small steady state error or fast convergence rate based on the requirements of application [17]. In this experiment, we evaluate the convergence rates of the algorithms. For this purpose, we select the parameters such that the algorithms converge to the same steady state error level. In this setup, we choose the parameters for each node k as follows. Since $\mathbf{X}_{k,t} \in \mathbb{R}^2$ is our input, we set the output dimension as $n = 2$.

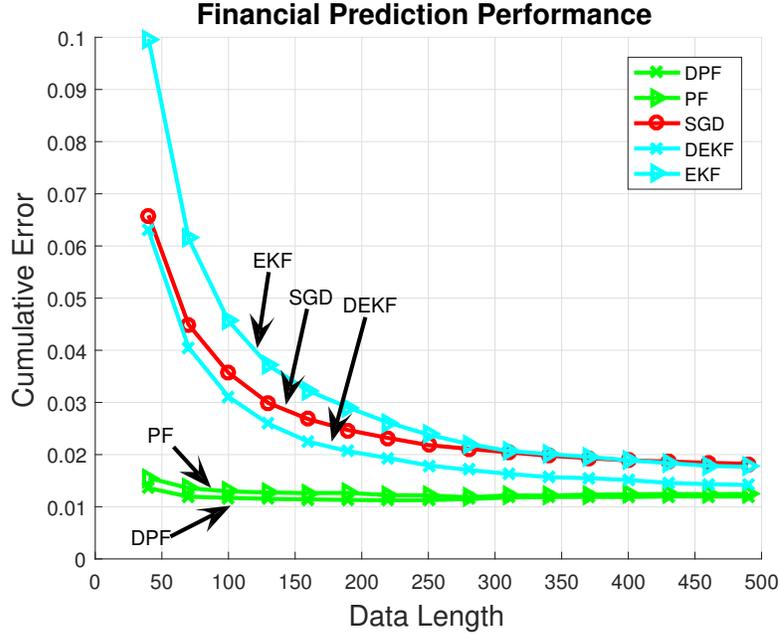


Figure 3.2: Error performances over the Hong Kong exchange rate dataset.

In addition to this, we consider a network of four nodes.

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

For the EKF based algorithms, we select $\Sigma_{k,0|0} = 0.0004\mathbf{I}$, $\mathbf{Q}_{k,t} = 0.0004\mathbf{I}$ and $R_{k,t} = 0.01$. Moreover, according to (3.19), the weights between nodes are calculated as $1/3$.

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

In Fig. 3.2, we illustrate the prediction performance of the algorithms. Due to the highly nonlinear structure of our model, the EKF and DEKF based algorithms have slower convergence compared to the other algorithms. Moreover, due to only exploiting the first order gradient information, the SGD based algorithm has also slower convergence compared to the PF based algorithms. Unlike the SGD and EKF based methods, the PF based algorithms perform parameter estimation

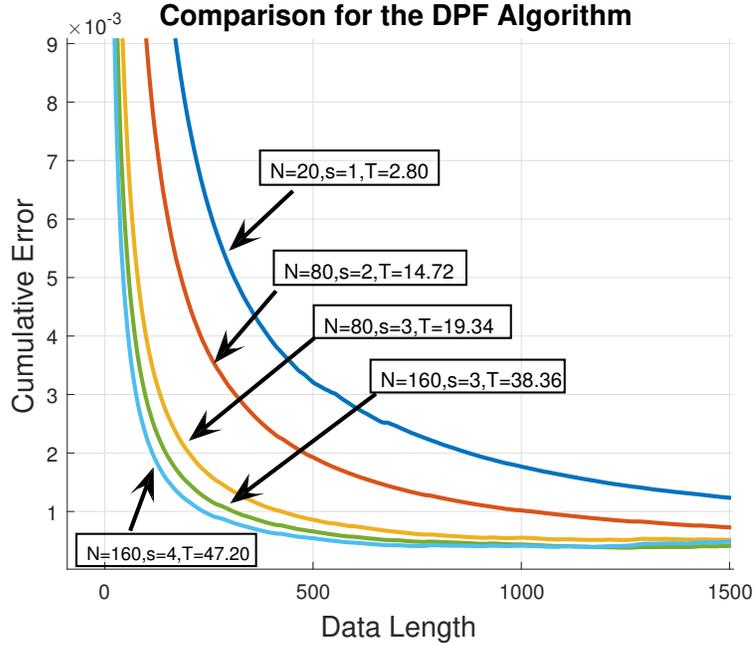


Figure 3.3: Error performances for different N and s combinations of the DPF based algorithm. Here, we also provide computation times of the combinations (in seconds), i.e., denoted as T , where a computer with i5-6400 processor, 2.7 GHz CPU and 16 GB RAM is used.

through a high performance gradient free density estimation technique, hence, they converge much faster to the final MSE level, i.e., defined as $\frac{1}{T} \sum_{t=1}^T (d_t - \hat{d}_t)^2$ for a sequence of length T . Among the PF based methods, due to its distributed structure the DPF based algorithm has the fastest convergence rate.

In order to demonstrate the effects of the number of particles N and the number of Markov steps s , we perform another experiment using the Hong Kong exchange rate dataset. In this experiment, we use the same setting with the previous case except $\text{Cov}[\gamma_{k,t}] = 0.0001\mathbf{I}$, $\Sigma_{k,0|0} = 0.0001\mathbf{I}$ and $\mathbf{Q}_{k,t} = 0.0001\mathbf{I}$. In Fig. 3.3, we observe that as s and N increase, the DPF based algorithm obtains a faster convergence rate and a lower final MSE value. However, as s and N increase, the marginal performance improvement becomes smaller with respect to the previous s and N values. Furthermore, the computation time of the algorithm increases with increasing s and N values. Thus, after a certain selection, a further increase does not worth the additional computational load. Therefore, we use $N(k) = 80$ and $s = 3$ in our previous simulation.

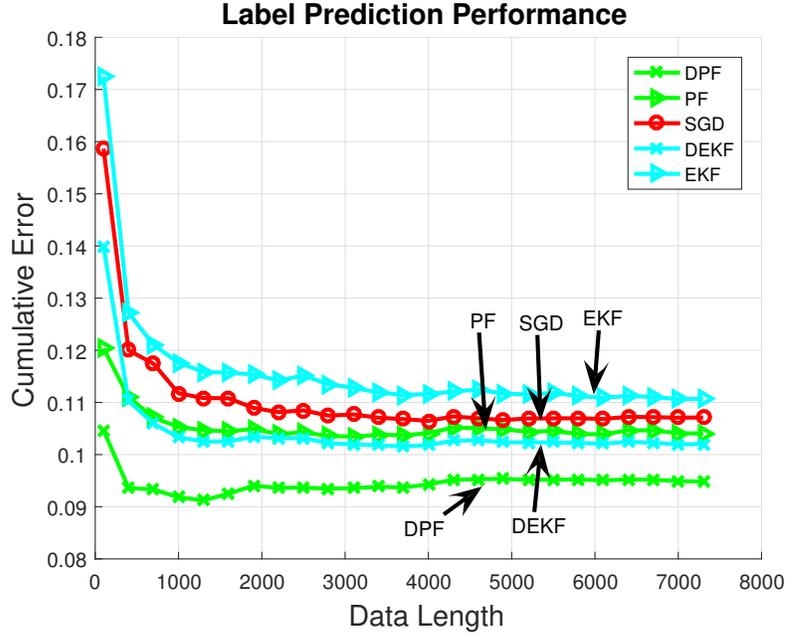


Figure 3.4: Error performances over the sentence dataset.

Other than the Hong Kong exchange rate dataset, we consider the emotion labelled sentence dataset [64]. In this dataset, we have the vector representation of each word in an emotion labelled sentence. In this experiment, we evaluate the steady state error performance of the algorithms. Thus, we choose the parameters such that the convergence rate of the algorithms are similar. To provide this setup, we select the parameters for each node k as follows. Since the number of words varies from sentence to sentence in this case, we have a variable length input regressor, i.e., defined as $\mathbf{X}_{k,t} \in \mathbb{R}^{2 \times m_t}$, where m_t represents the number of words in a sentence. For the other parameters, we use the same setting with the Hong Kong exchange rate dataset except $N(k) = 50$, $\text{Cov}[\boldsymbol{\gamma}_{k,t}] = (0.025)^2 \mathbf{I}$, $\boldsymbol{\Sigma}_{k,0|0} = (0.025)^2 \mathbf{I}$, $\mathbf{Q}_{k,t} = (0.025)^2 \mathbf{I}$ and $\mu = 0.055$. In Fig. 3.4, we illustrate the label prediction performance of the algorithms. Again due to the highly non-linear structure of our model, the EKF based algorithm has the highest steady state error value. Additionally, the SGD based algorithm also has a high final MSE value compared to the other algorithms. Furthermore, the DEKF based algorithm achieves a lower final MSE value than the PF based method thanks to its distributed structure. However, since the DPF based method utilizes a

powerful gradient free density estimation method while effectively sharing information between the neighboring nodes, it achieves a much smaller steady state error value.

We also perform another experiment that includes a larger dataset, where we include two additional algorithms, namely the Hessian-free [33] and quasi-Newton [34] algorithms to illustrate their performances. For this purpose, we use the temperature dataset [65] and in this dataset, we have temperature data that was collected from 2006 to 2013 by a weather station in Amherst, Massachusetts. Our aim is to predict tomorrow’s temperature value by examining the temperature of the previous four days. Since we aim to compare the convergence rates of the algorithms, we select the parameters such that all the algorithms converge to the same steady state error level. Here, we use the same setting with the Hong Kong exchange rate dataset except $N(k) = 80$, $\text{Cov}[\boldsymbol{\gamma}_{k,t}] = 0.0004\mathbf{I}$, $\boldsymbol{\Sigma}_{k,0|0} = 0.0004\mathbf{I}$, $\mathbf{Q}_{k,t} = 0.0004\mathbf{I}$ and $\mu = 0.001$. Furthermore, we set the learning rate of the quasi-Newton algorithm as 0.0003 and for the other parameters of the Hessian-free and quasi-Newton algorithms, we follow [33,34]. Here, we denote the Hessian-free and quasi-Newton algorithms as “HF” and “QN”, respectively. In Fig. 3.5, we demonstrate the temperature prediction performances of the algorithms. Since the SGD based algorithm only exploits the first order gradient information, it has the slowest convergence rate compared to the others. The QN and HF algorithms perform similarly and achieve slightly slower convergence rate compared to the EKF based algorithm. The PF based algorithm achieves the fastest rate among the algorithms that lack communication thanks to its powerful gradient free density estimation technique. However, overall the distributed algorithms achieves much faster rates due to their communication capability, among which the DPF based algorithm converges to the final MSE level in a much faster manner.

In addition to the temperature dataset, we also perform experiments on four different large datasets. For these datasets, we select the parameters such that all the algorithms have similar convergence rates. We first consider the CMU ARCTIC dataset [66], where an English male speaker reads 1132 different utterances and it contains 4×10^5 samples. Our aim is to predict the next sample based on

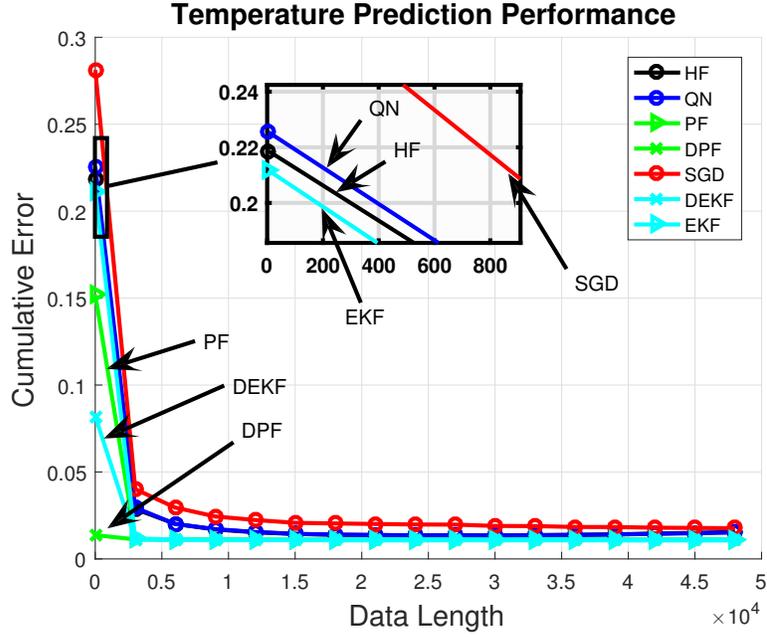


Figure 3.5: Sequential prediction performance of the algorithms for the temperature dataset.

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

Table 3.2: Time accumulated errors the algorithms for the speech, elevators, pumadyn and bank datasets.

the previous four samples. For this experiments, we use the same setting with the temperature dataset except $N(k) = 50$, $\text{Cov}[\gamma_{k,t}] = 0.0001\mathbf{I}$, $\Sigma_{k,0|0} = 0.0001\mathbf{I}$, $\mathbf{Q}_{k,t} = 0.0001\mathbf{I}$ and $\mu = 0.003$. We then perform another experiment on the bank dataset [61], where we have feature vectors related to the queues in banks and we aim to estimate the fraction of the people that leaves the bank because of the full queues. This dataset contains 8192 samples. For this experiment, we use the same setting with the speech dataset except $n = 32$, $N(k) = 150$, $\text{Cov}[\gamma_{k,t}] = 0.0016\mathbf{I}$, $\text{Var}[\varepsilon_{k,t}] = 0.25$, $\Sigma_{k,0|0} = 0.0016\mathbf{I}$, $\mathbf{Q}_{k,t} = 0.0016\mathbf{I}$, $R_{k,t} = 0.25$ and $\mu = 0.07$. As the third dataset, we use the elevator dataset [60], where we have feature vectors related to an F16 aircraft and our aim is to predict a certain variable that explains the aircraft’s actions. This dataset contains 16000 samples and we use the same setting with the bank dataset except $n = 18$, $N(k) = 100$ and $\mu = 0.7$.

Finally, we perform an experiment on the pumadyn dataset [60], where we have feature vectors related to the action of a robotic arm and we aim to predict its angular acceleration. Here, we have 8192 samples and we use the same setting with the bank dataset except $N(k) = 170$ and $\mu = 0.4$. As seen in Table 3.2, in all experiments, the DPF based algorithm achieves much smaller time accumulated error thanks to its distributed architecture and high performance gradient free density estimation technique.

Chapter 4

Energy Efficient LSTM Networks for Online Learning

In this chapter, we sequentially receive $\{d_t\}_{t \geq 1}$, $d_t \in \mathbb{R}$ and matrices $\{\mathbf{X}_t\}_{t \geq 1}$, i.e., defined as $\mathbf{X}_t = [\mathbf{x}_{t,1}, \mathbf{x}_{t,2} \dots, \mathbf{x}_{t,n_t}]$, where $\mathbf{x}_{t,j} \in \mathbb{R}^p, \forall j \in \{1, 2, \dots, n_t\}$ and $n_t \in \mathbb{Z}^+$ is the number of columns in \mathbf{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 \mathbf{X}_t . To find this relation, after receiving each \mathbf{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 [67]. As an example, in sequential prediction under the square loss, at each time t , we receive a set of features, i.e., \mathbf{X}_t in our case, related to the desired label d_t . We then generate the estimate through a function, i.e., $\hat{d}_t = \kappa(\mathbf{X}_t)$. After the desired label d_t is observed, we suffer the square loss $L(\hat{d}_t, d_t) = (d_t - \hat{d}_t)^2$.

Here, we use RNNs to obtain the final estimate \hat{d}_t . Since we have variable length data sequences, we use a structure as in Fig. 4.1 to obtain fixed length

sequences. The basic RNN architecture is defined by the following equations [16]:

$$\mathbf{h}_{t,j} = f(\mathbf{W}\mathbf{x}_{t,j} + \mathbf{R}\mathbf{h}_{t,j-1}) \quad (4.1)$$

$$\mathbf{z}_{t,j} = g(\mathbf{U}\mathbf{h}_{t,j}), \quad (4.2)$$

where $\mathbf{x}_{t,j} \in \mathbb{R}^p$ is the input vector, $\mathbf{h}_{t,j} \in \mathbb{R}^m$ is the state vector and $\mathbf{z}_{t,j} \in \mathbb{R}^m$ is the output vector for the j^{th} RNN unit. Here, $f(\cdot)$ and $g(\cdot)$ usually set to the hyperbolic tangent function and they apply to vectors pointwise. Moreover, \mathbf{W} , \mathbf{R} and \mathbf{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 the final estimate \hat{d}_t . Among different implementations of LSTM network, we choose the most widely used one, i.e., the LSTM network without peephole connections [4]. Since we receive variable length data sequence \mathbf{X}_t at time t , we apply the LSTM network to each column of \mathbf{X}_t as illustrated in Fig. 4.1, where the internal LSTM equations for the j^{th} unit are as follows [4, 12]:

$$\tilde{\mathbf{c}}_{t,j} = g\left(\mathbf{W}^{(\tilde{c})}\mathbf{x}_{t,j} + \mathbf{R}^{(\tilde{c})}\mathbf{h}_{t,j-1} + \mathbf{b}^{(\tilde{c})}\right) \quad (4.3)$$

$$\mathbf{i}_{t,j} = \sigma\left(\mathbf{W}^{(i)}\mathbf{x}_{t,j} + \mathbf{R}^{(i)}\mathbf{h}_{t,j-1} + \mathbf{b}^{(i)}\right) \quad (4.4)$$

$$\mathbf{f}_{t,j} = \sigma\left(\mathbf{W}^{(f)}\mathbf{x}_{t,j} + \mathbf{R}^{(f)}\mathbf{h}_{t,j-1} + \mathbf{b}^{(f)}\right) \quad (4.5)$$

$$\mathbf{c}_{t,j} = \mathbf{D}_{t,j}^{(i)}\tilde{\mathbf{c}}_{t,j} + \mathbf{D}_{t,j}^{(f)}\mathbf{c}_{t,j-1} \quad (4.6)$$

$$\mathbf{o}_{t,j} = \sigma\left(\mathbf{W}^{(o)}\mathbf{x}_{t,j} + \mathbf{R}^{(o)}\mathbf{h}_{t,j-1} + \mathbf{b}^{(o)}\right) \quad (4.7)$$

$$\mathbf{h}_{t,j} = \mathbf{D}_{t,j}^{(o)}g(\mathbf{c}_{t,j}), \quad (4.8)$$

where $\mathbf{c}_{t,j} \in \mathbb{R}^m$ is the state vector, $\mathbf{x}_{t,j} \in \mathbb{R}^p$ is the input vector, $\mathbf{h}_{t,j} \in \mathbb{R}^m$ is the output vector. Here, $\mathbf{i}_{t,j}$, $\mathbf{f}_{t,j}$ and $\mathbf{o}_{t,j}$ 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. Moreover, $\mathbf{D}_{t,j}^{(i)} = \text{diag}(\mathbf{i}_{t,j})$, $\mathbf{D}_{t,j}^{(f)} = \text{diag}(\mathbf{f}_{t,j})$ and $\mathbf{D}_{t,j}^{(o)} = \text{diag}(\mathbf{o}_{t,j})$. 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 in Fig. 4.1, 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,

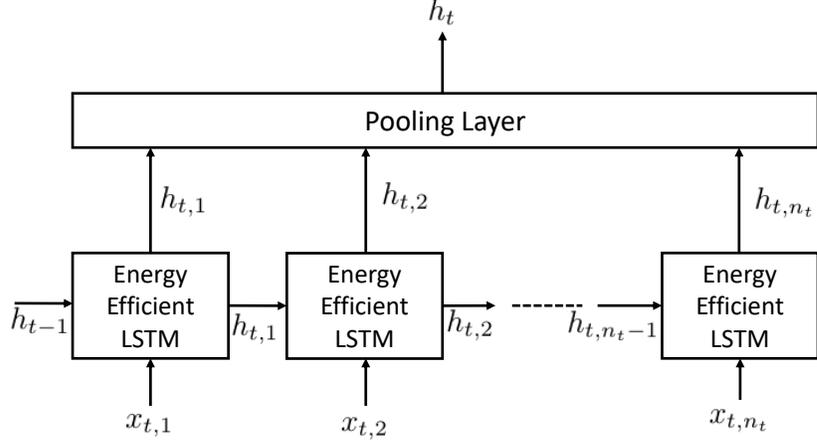


Figure 4.1: Detailed schematic of Energy Efficient LSTM based architecture.

i.e., denoted as $\mathbf{h}_t \in \mathbb{R}^m$ at time t . Using the fixed length vectors, we generate the final estimate as

$$\hat{d}_t = \mathbf{w}_t^T \mathbf{h}_t, \quad (4.9)$$

where $\mathbf{w}_t \in \mathbb{R}^m$ represents a vector of 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. 4.1, we use the mean pooling method to obtain fixed length output vectors as $\mathbf{h}_t = \frac{1}{n_t} \sum_{j=1}^{n_t} \mathbf{h}_{t,j}$. 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 $\mathbf{h}_t = \max_j \mathbf{h}_{t,j}$ and $\mathbf{h}_t = \mathbf{h}_{t,n_t}$, respectively. With such changes, our derivations can be extended to the other pooling methods.

4.1 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.

4.1.1 RNN with ef-operator

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

Let $\mathbf{a}, \mathbf{b} \in \mathbb{R}^p$, the ef-operator [44] on \mathbf{a} and \mathbf{b} is defined as

$$\mathbf{a} \diamond \mathbf{b} := \sum_{i=1}^p \text{sign}(a_i \times b_i)(|a_i| + |b_i|), \quad (4.10)$$

where the $\text{sign}(\cdot)$ function returns the sign of its input. (4.10) can also be written as

$$\mathbf{a} \diamond \mathbf{b} := \sum_{i=1}^p \text{sign}(a_i)b_i + \text{sign}(b_i)a_i. \quad (4.11)$$

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

By applying the ef-operator, (4.1) can be written as

$$\mathbf{h}_{t,j} = f\left(\mathbf{a}_h \odot (\mathbf{W} \diamond \mathbf{x}_{t,j}) + \mathbf{b}_h \odot (\mathbf{R} \diamond \mathbf{h}_{t,j-1})\right), \quad (4.12)$$

where $\mathbf{a}_h \in \mathbb{R}^m$ and $\mathbf{b}_h \in \mathbb{R}^m$ are the scaling coefficients introduced in [46, 47] and \odot is the element by element multiplication of two vectors of the same size. In (4.12), $\mathbf{W} \diamond \mathbf{x}_{t,j}$ and $\mathbf{R} \diamond \mathbf{h}_{t,j-1}$ are given as follows

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

where \mathbf{w}_i represents the transpose of the i^{th} row of \mathbf{W} and $\mathbf{w}_i \diamond \mathbf{x}_{t,j}$ is given as

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

where $x_{t,jk}$ is the k^{th} element of $\mathbf{x}_{t,j}$. Similarly,

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

where \mathbf{r}_i represents the transpose of the i^{th} row of \mathbf{R} and $\mathbf{r}_i \diamond \mathbf{h}_{t,j}$ is given as

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

where $h_{t,jk}$ is the k^{th} element of $\mathbf{h}_{t,j}$. Likewise, (4.2) is modified as follows

$$\mathbf{z}_{t,j} = g(\mathbf{b}_z \odot (\mathbf{U} \diamond \mathbf{h}_{t,j})), \quad (4.13)$$

where \mathbf{b}_z is the scaling coefficient.

4.1.2 LSTM with ef-operator

In this subsection, we replace the regular multiplication operators in the classical LSTM architecture with the ef-operator. Based on this modification, (4.3)–(4.8) are written as follows

$$\tilde{\mathbf{c}}_{t,j} = g\left(\mathbf{a}_{\tilde{c}} \odot (\mathbf{W}^{(\tilde{c})} \diamond \mathbf{x}_{t,j}) + \mathbf{b}_{\tilde{c}} \odot (\mathbf{R}^{(\tilde{c})} \diamond \mathbf{h}_{t,j-1}) + \mathbf{b}^{(\tilde{c})}\right) \quad (4.14)$$

$$\mathbf{i}_{t,j} = \sigma\left(\mathbf{a}_i \odot (\mathbf{W}^{(i)} \diamond \mathbf{x}_{t,j}) + \mathbf{b}_i \odot (\mathbf{R}^{(i)} \diamond \mathbf{h}_{t,j-1}) + \mathbf{b}^{(i)}\right) \quad (4.15)$$

$$\mathbf{f}_{t,j} = \sigma\left(\mathbf{a}_f \odot (\mathbf{W}^{(f)} \diamond \mathbf{x}_{t,j}) + \mathbf{b}_f \odot (\mathbf{R}^{(f)} \diamond \mathbf{h}_{t,j-1}) + \mathbf{b}^{(f)}\right) \quad (4.16)$$

$$\mathbf{c}_{t,j} = \mathbf{i}_{t,j} \blacklozenge \tilde{\mathbf{c}}_{t,j} + \mathbf{f}_{t,j} \blacklozenge \mathbf{c}_{t,j-1} \quad (4.17)$$

$$\mathbf{o}_{t,j} = \sigma\left(\mathbf{a}_o \odot (\mathbf{W}^{(o)} \diamond \mathbf{x}_{t,j}) + \mathbf{b}_o \odot (\mathbf{R}^{(o)} \diamond \mathbf{h}_{t,j-1}) + \mathbf{b}^{(o)}\right) \quad (4.18)$$

$$\mathbf{h}_{t,j} = \mathbf{o}_{t,j} \blacklozenge g(\mathbf{c}_{t,j}), \quad (4.19)$$

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

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

In (4.14), $\mathbf{W}^{(\bar{c})} \diamond \mathbf{x}_{t,j}$ is written as

$$\mathbf{W}^{(\bar{c})} \diamond \mathbf{x}_{t,j} = [\mathbf{w}_1^{(\bar{c})} \diamond \mathbf{x}_{t,j} \ \mathbf{w}_2^{(\bar{c})} \diamond \mathbf{x}_{t,j} \ \dots \ \mathbf{w}_m^{(\bar{c})} \diamond \mathbf{x}_{t,j}]^T, \quad (4.20)$$

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

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

and

$$\mathbf{R}^{(\bar{c})} \diamond \mathbf{h}_{t,j-1} = [\mathbf{r}_1^{(\bar{c})} \diamond \mathbf{h}_{t,j-1} \ \mathbf{r}_2^{(\bar{c})} \diamond \mathbf{h}_{t,j-1} \ \dots \ \mathbf{r}_m^{(\bar{c})} \diamond \mathbf{h}_{t,j-1}]^T, \quad (4.21)$$

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

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

where $\mathbf{w}_i^{(\bar{c})}$ and $\mathbf{r}_i^{(\bar{c})}$ are the i^{th} row of $\mathbf{W}^{(\bar{c})}$ and $\mathbf{R}^{(\bar{c})}$ respectively.

For the other multiplications, we change the parameters in either (4.20) or (4.21) according to the chosen coefficient matrix. Other than that, we follow the same procedures in (4.20) and (4.21).

Remark 4.1.1. *Compared to the original LSTM network in (4.3)–(4.8), 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 (4.14)–(4.16) and (4.18), 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, which provides substantial decrease in the computational complexity and energy consumption compared to the classical LSTM network.*

4.1.3 Energy Efficient RNN with Weight Matrix Factorization

In this subsection, we apply the matrix factorization method [48] to the weight matrices of the basic RNN architecture in order to reduce the number of parameters to be trained.

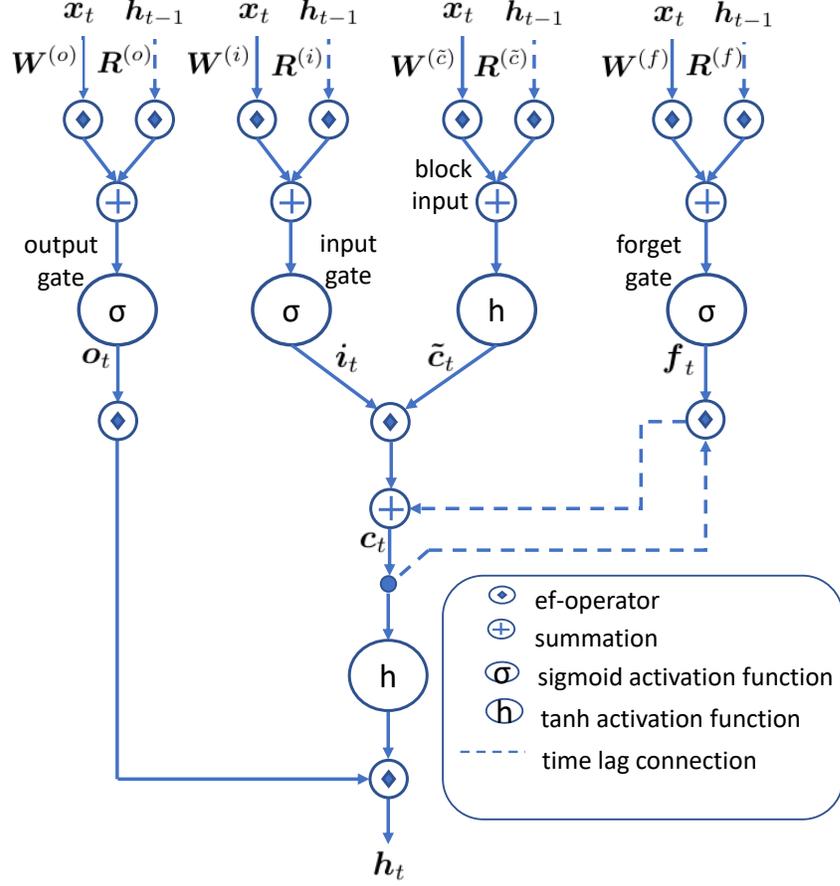


Figure 4.2: Detailed schematic of energy efficient LSTM block at time t . Note that solid lines represent direct connections while dotted lines represent time lagged connections. For the sake of simplicity, bias terms are not shown in the figure.

We first factorize the weight matrices in (4.12) as $\mathbf{W} \approx \mathbf{MN}$ and $\mathbf{R} \approx \mathbf{PQ}$, where $\mathbf{W} \in \mathbb{R}^{m \times p}$, $\mathbf{M} \in \mathbb{R}^{m \times d}$, $\mathbf{N} \in \mathbb{R}^{d \times p}$, $\mathbf{R} \in \mathbb{R}^{m \times m}$, $\mathbf{P} \in \mathbb{R}^{m \times f}$ and $\mathbf{Q} \in \mathbb{R}^{f \times m}$.

Remark 4.1.2. We factorize the RNN weight matrices into two smaller matrices. The rank of these two smaller matrices are selected such that $d, f \ll \min(m, p)$. Thus, we significantly reduce the number of parameters that needs to be learnt, e.g., \mathbf{W} has mp entries while \mathbf{M} and \mathbf{N} have $d(m + p) \ll mp$.

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

$$\mathbf{h}_{t,j} = f\left(\mathbf{a}_h \odot (\mathbf{M} \diamond \mathbf{N} \diamond \mathbf{x}_{t,j}) + \mathbf{b}_h \odot (\mathbf{P} \diamond \mathbf{Q} \diamond \mathbf{h}_{t,j-1})\right). \quad (4.22)$$

In (4.22), $\mathbf{M} \diamond \mathbf{N} \diamond \mathbf{x}_{t,j}$ is given as follows

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

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

$$\boldsymbol{\mu}_i \diamond \mathbf{x}_{t,j} = \sum_{k=1}^p \text{sign}(x_{t,jk}) \mu_{ik} + \text{sign}(\mu_{ik}) x_{t,jk} \quad (4.23)$$

and

$$\mathbf{P} \diamond \mathbf{Q} \diamond \mathbf{h}_{t,j-1} = [\boldsymbol{\nu}_1 \diamond \mathbf{h}_{t,j-1} \ \boldsymbol{\nu}_2 \diamond \mathbf{h}_{t,j-1} \ \dots \ \boldsymbol{\nu}_m \diamond \mathbf{h}_{t,j-1}]^T,$$

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

$$\boldsymbol{\nu}_i \diamond \mathbf{h}_{t,j} = \sum_{k=1}^m \text{sign}(h_{t,jk}) \nu_{ik} + \text{sign}(\nu_{ik}) h_{t,jk}. \quad (4.24)$$

In a similar manner, (4.13) is modified as follows

$$\mathbf{z}_{t,j} = g(\mathbf{b}_z \odot (\mathbf{S} \diamond \mathbf{T} \diamond \mathbf{h}_{t,j})), \quad (4.25)$$

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

4.1.4 Energy Efficient LSTM with Weight Matrix Factorization

In this subsection, we apply the matrix factorization method [48] 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 $\mathbf{W}^{(\cdot)} \approx \mathbf{M}^{(\cdot)}\mathbf{N}^{(\cdot)}$ and $\mathbf{R}^{(\cdot)} \approx \mathbf{P}^{(\cdot)}\mathbf{Q}^{(\cdot)}$, where $\mathbf{W}^{(\cdot)} \in \mathbb{R}^{m \times p}$, $\mathbf{M}^{(\cdot)} \in \mathbb{R}^{m \times d}$, $\mathbf{N}^{(\cdot)} \in \mathbb{R}^{d \times p}$, $\mathbf{R}^{(\cdot)} \in \mathbb{R}^{m \times m}$, $\mathbf{P}^{(\cdot)} \in \mathbb{R}^{m \times f}$ and $\mathbf{Q}^{(\cdot)} \in \mathbb{R}^{f \times m}$ such that $d, f \ll \min(p, m)$. We then apply this factorization to the LSTM neural

network in (4.14)–(4.19) by replacing the weight matrices with their factorized forms. The modifications for the j^{th} LSTM unit in Fig. 4.1 are as in the following.

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

$$\mathbf{M}^{(\tilde{c})} \diamond \mathbf{N}^{(\tilde{c})} \diamond \mathbf{x}_{t,j} = [\boldsymbol{\mu}_1^{(\tilde{c})} \diamond \mathbf{x}_{t,j} \boldsymbol{\mu}_2^{(\tilde{c})} \diamond \mathbf{x}_{t,j} \dots \boldsymbol{\mu}_m^{(\tilde{c})} \diamond \mathbf{x}_{t,j}]^T, \quad (4.26)$$

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

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

and

$$\mathbf{P}^{(\tilde{c})} \diamond \mathbf{Q}^{(\tilde{c})} \diamond \mathbf{h}_{t,j-1} = [\boldsymbol{\nu}_1^{(\tilde{c})} \diamond \mathbf{h}_{t,j-1} \boldsymbol{\nu}_2^{(\tilde{c})} \diamond \mathbf{h}_{t,j-1} \dots \boldsymbol{\nu}_m^{(\tilde{c})} \diamond \mathbf{h}_{t,j-1}]^T, \quad (4.27)$$

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

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

For the other weight matrices, we use the factorized form of the chosen weight matrix in (4.26) and (4.27). Then, we follow the same operations in (4.26) and (4.27).

Remark 4.1.3. *We reduce the total number of LSTM network parameters by applying weight matrix factorization. In the original LSTM equations in (4.3)–(4.8), we have $4m(m+p)$ scalar parameter in the weight matrices, i.e., $\mathbf{W}^{(\cdot)}$ and $\mathbf{R}^{(\cdot)}$. However, in our energy efficient LSTM network, we have $4d(m+p) + 8mf$, which is significantly less than $4m(m+p)$ provided that $d, f \ll \min(m, p)$.*

4.1.5 Online Training Algorithms

In this subsection, 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 [23] to obtain the online updates for each parameter. For \mathbf{w}_t , the SGD update is computed as follows

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}_t} L, \quad (4.28)$$

where $\nabla_{\mathbf{w}_t}$ represent the gradient of a certain function with respect to \mathbf{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., (4.28), are additive updates since the gradient information is being added at each time step.

On the other hand, for \mathbf{w}_t , the EG update [40] is computed as follows

$$w_{t+1,i} = \frac{w_{t,i} r_{t,i}}{\sum_{j=1}^m w_{t,j} r_{t,j}}, \quad (4.29)$$

where

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

and $w_{t,i}$ is the i^{th} component of \mathbf{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 (4.29), 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 (4.29), we use the first order Taylor series expansion along with the ef operator as follows

$$w_{t+1,i} = \frac{w_{t,i} \diamond \hat{r}_{t,i}}{\sum_{j=1}^m (w_{t,j} \diamond \hat{r}_{t,j})}, \quad (4.30)$$

where

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

Note that since the division in (4.30) 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 4.1.4. *Since the weight vector \mathbf{w}_t might contain negative components, we use the slightly modified version of the original EG algorithms, i.e., the EG⁺ algorithm [40], that uses a weight vector $\mathbf{w}_t^+ - \mathbf{w}_t^-$. In this algorithm, the weight vector is updated as follows*

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

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

where

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

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

In the following subsection, we derive the updates for the parameters of the proposed energy efficient RNN and LSTM networks.

4.1.5.1 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 $\mathbf{z}_t = \sum_{j=1}^{n_t} \mathbf{z}_{t,j}/n_t$ as the output at time t . Although our structure in Fig. 4.1 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 required extensions to obtain the equations for generic n_t values. With this modification, we have $\mathbf{z}_t = \mathbf{z}_{t,1}$, thus, we generate the estimate as $\hat{d}_t = \mathbf{w}_t^T \mathbf{z}_{t,1}$.

Under the square loss, we compute the first order derivative of the loss function with respect to b_{zi} , i.e., the i^{th} element of \mathbf{b}_z , as follows

$$\frac{\partial L}{\partial b_{zi}} = \frac{\partial L}{\partial \hat{d}_t} \frac{\partial \hat{d}_t}{\partial \mathbf{z}_{t,1}} \frac{\partial \mathbf{z}_{t,1}}{\partial b_{zi}} = -2(d_t - \hat{d}_t) \mathbf{w}_t^T \left[g'(\boldsymbol{\varphi}_t) \odot \left((\mathbf{u}_i \diamond \mathbf{h}_{t,1}) \mathbf{e}_i + \mathbf{b}_z \odot \boldsymbol{\lambda}_t^{(Uh)} \right) \right], \quad (4.31)$$

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

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

Moreover, for the i^{th} element of $\boldsymbol{\lambda}_t^{(Uh)} = \partial(\mathbf{U} \diamond \mathbf{h}_{t,1})/\partial b_{zi}$, we use the following formula in (4.31)

$$\lambda_{t,i}^{(Uh)} = \sum_{j=1}^m (u_{ij} 2\delta(h_{t,1j}) \gamma_{t,ij}^{(b_z)} + \text{sign}(u_{ij}) \gamma_{t,ij}^{(b_z)}), \quad (4.32)$$

where $\delta(\cdot)$ is the dirac delta function, we compute the derivative of the sign function as $d(\text{sign}(x))/dx = 2\delta(x)$ [68] and

$$\gamma_{t,ij}^{(b_z)} = \frac{\partial h_{t,1j}}{\partial b_{zi}} = f'_j(\boldsymbol{\theta}_t) b_{hj} \lambda_{t-1,j}^{(Rh)}, \quad (4.33)$$

where f'_i is the derivative of the i^{th} element of $f(\cdot)$ with respect to its argument and

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

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

$$\frac{\partial L}{\partial u_{ik}} = \frac{\partial L}{\partial \hat{d}_t} \frac{\partial \hat{d}_t}{\partial \mathbf{z}_{t,1}} \frac{\partial \mathbf{z}_{t,1}}{\partial u_{ik}} = -2(d_t - \hat{d}_t) \mathbf{w}_t^T \left[g'(\boldsymbol{\varphi}_t) \odot \left(b_{zi} (\text{sign}(h_{t,1k}) + 2\delta(u_{ik}) h_{t,1k}) \mathbf{e}_i + \mathbf{b}_z \odot \boldsymbol{\alpha}_t^{(Uh)} \right) \right], \quad (4.34)$$

where

$$\alpha_{t,i}^{(Uh)} = \sum_{j=1}^m (u_{ij} 2\delta(h_{t,1j}) \gamma_{t,ij}^{(u_{ik})} + \text{sign}(u_{ij}) \gamma_{t,ij}^{(u_{ik})}), \quad (4.35)$$

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 4.1.5. *When we take the derivative of L with respect to the other parameters, the position of the term with \mathbf{e}_i changes. As an example, for the derivative of L with respect to b_{hi} , $\mathbf{r}_i \diamond \mathbf{h}_{t-1,1}$ term appears in (4.33) when $j = i$, otherwise, (4.33) does not change. If we write (4.33) in a vector form, the contribution of $\mathbf{r}_i \diamond \mathbf{h}_{t-1,1}$ can be written as $(\mathbf{R} \diamond \mathbf{h}_{t-1,1})\mathbf{e}_i$. As seen in this case, for the other derivatives, the position and form of the term with \mathbf{e}_i slightly changes, other than that we follow the same procedure in (4.31)–(4.35).*

Remark 4.1.6. *For the other n_t values, i.e., $n_t \neq 1$, the recursion in (4.33) is performed through the outputs of the different RNN blocks at a certain time t as in Fig. 4.1. Thus, rather than having $t - 1$ in (4.33), 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 4.1.7. *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 (4.34), instead of only taking the derivative with respect to the parameters of \mathbf{U} , we compute the derivatives of the loss with respect to the entries of both \mathbf{S} and \mathbf{T} , i.e., the factorized versions of \mathbf{U} . Other than such changes, we follow the same procedures in (4.31)–(4.35).*

With the derived gradients, we can update each parameter of the basic RNN architecture as in (4.28) and (4.30).

4.1.5.2 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 required extensions to obtain the equations for generic n_t values. With this modification, we have $\mathbf{h}_t = \mathbf{h}_{t,1}$, hence, we generate the estimate $\hat{d}_t = \mathbf{w}_t^T \mathbf{h}_{t,1}$.

We first compute the derivative of L with respect to $w_{ij}^{(\bar{c})}$, i.e., the element at

the i^{th} row and the j^{th} column of $\mathbf{W}^{(\tilde{c})}$, as follows

$$\frac{\partial L}{\partial w_{ij}^{(\tilde{c})}} = \frac{\partial L}{\partial \hat{d}_t} \frac{\partial \hat{d}_t}{\partial \mathbf{h}_{t,1}} \frac{\partial \mathbf{h}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} = -2(d_t - \hat{d}_t) \mathbf{w}_t^T \frac{\partial(\mathbf{o}_{t,1} \blacklozenge g(\mathbf{c}_{t,1}))}{\partial w_{ij}^{(\tilde{c})}}. \quad (4.36)$$

In (4.36), we calculate the partial derivative as

$$\begin{aligned} \frac{\partial(\mathbf{o}_{t,1} \blacklozenge g(\mathbf{c}_{t,1}))}{\partial w_{ij}^{(\tilde{c})}} &= \frac{\partial \mathbf{o}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \odot \text{sign}(g(\mathbf{c}_{t,1})) + \mathbf{o}_{t,1} \odot 2\delta(g(\mathbf{c}_{t,1})) \odot g'(\mathbf{c}_{t,1}) \odot \frac{\partial \mathbf{c}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \\ &\quad + 2\delta(\mathbf{o}_{t,1}) \odot \frac{\partial \mathbf{o}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \odot g(\mathbf{c}_{t,1}) + \text{sign}(\mathbf{o}_{t,1}) \odot g'(\mathbf{c}_{t,1}) \odot \frac{\partial \mathbf{c}_{t,1}}{\partial w_{ij}^{(\tilde{c})}}. \end{aligned} \quad (4.37)$$

For (4.37), we now compute the derivatives of $\mathbf{o}_{t,1}$ and $\mathbf{c}_{t,1}$ with respect to $w_{ij}^{(\tilde{c})}$. With $\boldsymbol{\lambda}_{t-1}^{(R^{(o)h})} = \partial(\mathbf{R}^{(o)} \diamond \mathbf{h}_{t-1,1}) / \partial w_{ij}^{(\tilde{c})}$ as in (4.32), the derivative of (4.18) is as follows

$$\frac{\partial \mathbf{o}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} = \mathbf{D}_{t,1}^{(\sigma'(\zeta^{(o)}))} \left(\mathbf{b}_o \odot \boldsymbol{\lambda}_{t-1}^{(R^{(o)h})} \right), \quad (4.38)$$

where

$$\boldsymbol{\zeta}_{t,1}^{(o)} = \mathbf{a}_o \odot (\mathbf{W}^{(o)} \diamond \mathbf{x}_{t,1}) + \mathbf{b}_o \odot (\mathbf{R}^{(o)} \diamond \mathbf{h}_{t-1,1}) + \mathbf{b}^{(o)}. \quad (4.39)$$

In order to calculate (4.37), we also compute the derivative of $\mathbf{c}_{t,1}$ with respect to $w_{ij}^{(\tilde{c})}$. For this derivative, we obtain the following recursive relation from (4.17)

$$\begin{aligned} \frac{\partial \mathbf{c}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} &= \text{sign}(\tilde{\mathbf{c}}_{t,1}) \odot \frac{\partial \mathbf{i}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} + 2\delta(\tilde{\mathbf{c}}_{t,1}) \odot \frac{\partial \tilde{\mathbf{c}}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \odot \mathbf{i}_{t,1} \\ &\quad + \text{sign}(\mathbf{i}_{t,1}) \odot \frac{\partial \tilde{\mathbf{c}}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} + 2\delta(\mathbf{i}_{t,1}) \odot \frac{\partial \mathbf{i}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \odot \tilde{\mathbf{c}}_{t,1} \\ &\quad + \text{sign}(\mathbf{c}_{t-1,1}) \odot \frac{\partial \mathbf{f}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} + 2\delta(\mathbf{c}_{t-1,1}) \odot \frac{\partial \mathbf{c}_{t-1,1}}{\partial w_{ij}^{(\tilde{c})}} \odot \mathbf{f}_{t,1} \\ &\quad + \text{sign}(\mathbf{f}_{t,1}) \odot \frac{\partial \mathbf{c}_{t-1,1}}{\partial w_{ij}^{(\tilde{c})}} + 2\delta(\mathbf{f}_{t,1}) \odot \frac{\partial \mathbf{f}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} \odot \mathbf{c}_{t-1,1}. \end{aligned} \quad (4.40)$$

For (4.40), we compute the derivatives of (4.14), (4.15) and (4.16) with respect

to $w_{ij}^{(\tilde{c})}$ as in the following

$$\frac{\partial \mathbf{i}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} = \mathbf{D}_{t,1}^{(\sigma'(\zeta^{(i)}))} \left(\mathbf{b}_i \odot \boldsymbol{\lambda}_{t-1}^{(R^{(i)}h)} \right) \quad (4.41)$$

$$\frac{\partial \mathbf{f}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} = \mathbf{D}_{t,1}^{(\sigma'(\zeta^{(f)}))} \left(\mathbf{b}_f \odot \boldsymbol{\lambda}_{t-1}^{(R^{(f)}h)} \right) \quad (4.42)$$

$$\frac{\partial \tilde{\mathbf{c}}_{t,1}}{\partial w_{ij}^{(\tilde{c})}} = \mathbf{D}_{t,1}^{(g'(\zeta^{(\tilde{c})}))} \left((\text{sign}(x_{t,1j}) + 2\delta(w_{ij}^{(\tilde{c})})x_{t,1j})\mathbf{e}_i + \mathbf{b}_{\tilde{c}} \odot \boldsymbol{\lambda}_{t-1}^{(R^{(\tilde{c})}h)} \right). \quad (4.43)$$

Using (4.41)–(4.43), we compute (4.40). Then, we compute (4.37) using (4.40) and (4.38) in order to calculate (4.36). After obtaining (4.36), we update the parameter using the SGD and EG based algorithms as in (4.28) and (4.30).

As in Remark 4.1.5, when we take the derivative with respect to the other parameters, only the location of the term with \mathbf{e}_i changes. Similar to the RNN case, when $n_t \neq 1$, the recursion in (4.33) and (4.40) is performed through the outputs of the different LSTM blocks at a certain time t as in Fig. 4.1. Moreover, for the factorized LSTM network, we compute the derivatives of the loss function with respect to each factorized matrix parameter as in Remark 4.1.7.

With the derived gradients, we can update each parameter of the energy efficient LSTM architecture as in (4.28) and (4.30).

4.2 Simulations

In this section, we illustrate the performances of our algorithms on various datasets under different scenarios. We first compare the regression performances of our algorithms on a financial dataset, i.e., the Alcoa Corporation stock price rate dataset [62]. We then evaluate the regression performances on real-life datasets, i.e., the kinematic [59] and elevators [60] datasets. Since our approach is generic, we also compare the performances of our training algorithms on two different recurrent neural networks, i.e., the LSTM and GRU neural networks. We then compare the structural complexity of our energy efficient algorithms with the conventional structures.

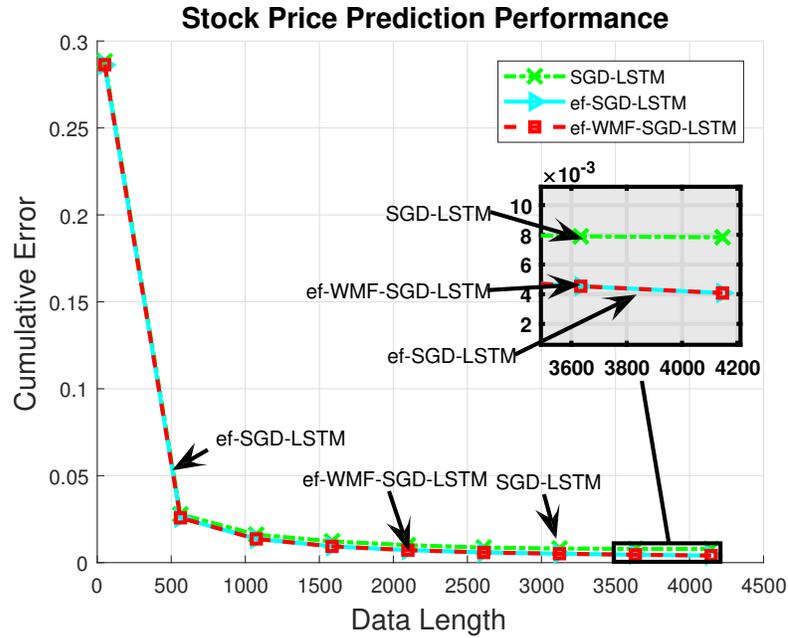


Figure 4.3: Daily stock price prediction performances of the algorithms with the SGD updates on the Alcoa Corporation stock price dataset.

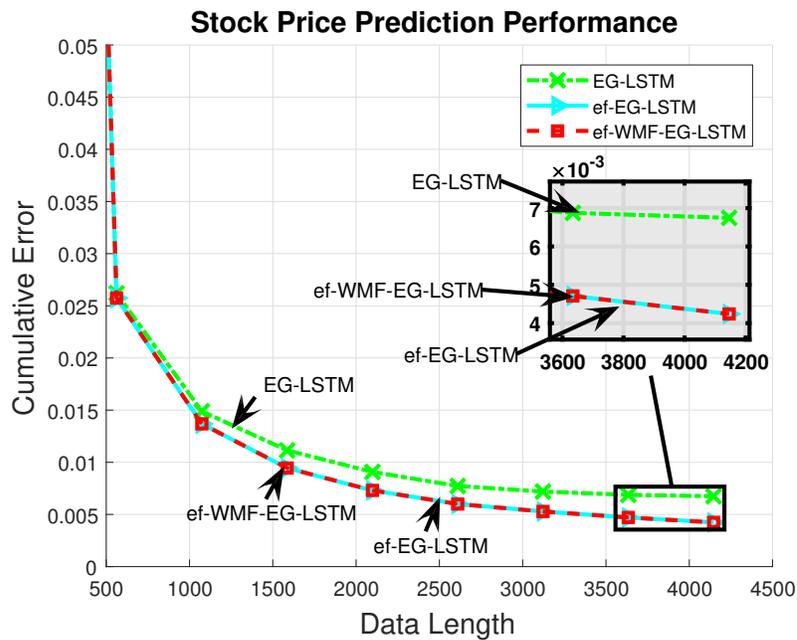


Figure 4.4: Daily stock price prediction performances of the algorithms with the EG updates on the Alcoa Corporation stock price dataset.

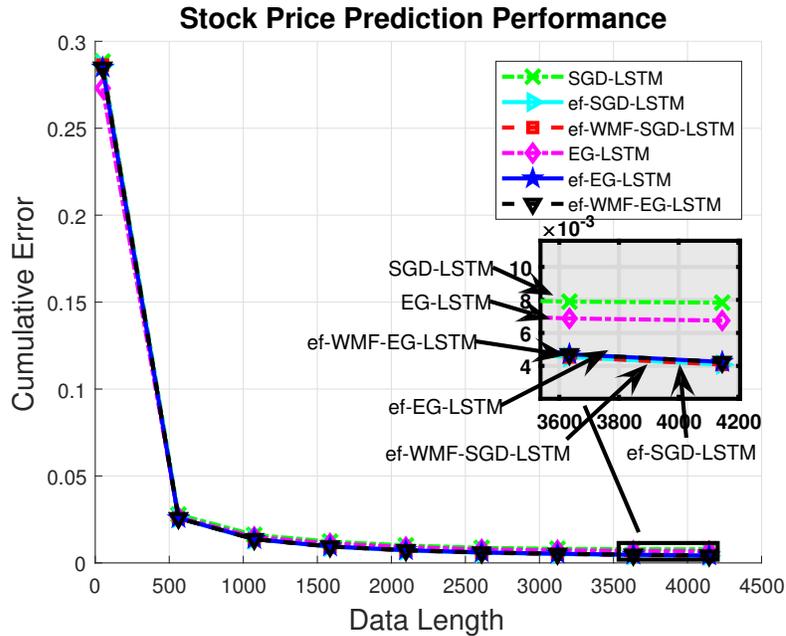


Figure 4.5: Comparison of all the algorithms with the SGD and EG updates on the Alcoa Corporation stock price dataset.

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).

4.2.1 Financial Dataset

In this subsection, we compare the performances of our algorithms on a financial dataset. We consider the Alcoa Corporation stock price dataset [62], 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 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. Additionally, for Model 3, we set the rank of factorized LSTM weight matrices as 2 based on our observations in the next

sections. For all these experiments, we set the learning rate as $\eta = 0.1$. Moreover, we have $\mathbf{X}_t \in \mathbb{R}^5$ and set the output dimensionality as $m = 5$.

Since Model 2 and Model 3 have a multiplication free structure, the gradient of each parameter becomes more robust against the vanishing and exploding gradient problems. Thus, in Fig. 4.3, Model 2 and Model 3 outperform Model 1 in terms of the error performance. Although both models, i.e., Model 2 and Model 3, perform similarly, we consider Model 3 as superior due to less operational complexity and smaller number of network parameters to be trained (see details in the next sections). Likewise, in Fig. 4.4, Model 2 and Model 3 have smaller error than Model 1. In Fig. 4.5, we evaluate the combined results of all the models with both SGD and EG updates. Model 2 and Model 3 with SGD updates provide slightly smaller steady state errors compared to all other models. Overall, Model 3 with the SGD updates outperforms its competitors in terms of both error performance and complexity issues.

4.2.2 Real Life Datasets

In this subsection, we compare the performances of our algorithms using two real-life datasets, i.e., the kinematic [59] and elevators [60] datasets. We first evaluate the performances of the models on the kinematic dataset [59], which contains data related to a realistic simulation of the forward dynamics of an 8 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 common parameters in all models to be same, e.g., the learning rate η . For this dataset, the input vector is $\mathbf{X}_t \in \mathbb{R}^8$, $m = 8$ 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 seen in Fig. 4.6, all the models perform similarly. However, in terms of computational complexity and the total number of network parameters, Model 3 has the lowest operational complexity and the total number of parameters. In Fig. 4.7, Model 3 outperforms all other models thanks to having smaller number of parameters and less complicated optimization problem for the parameters. In

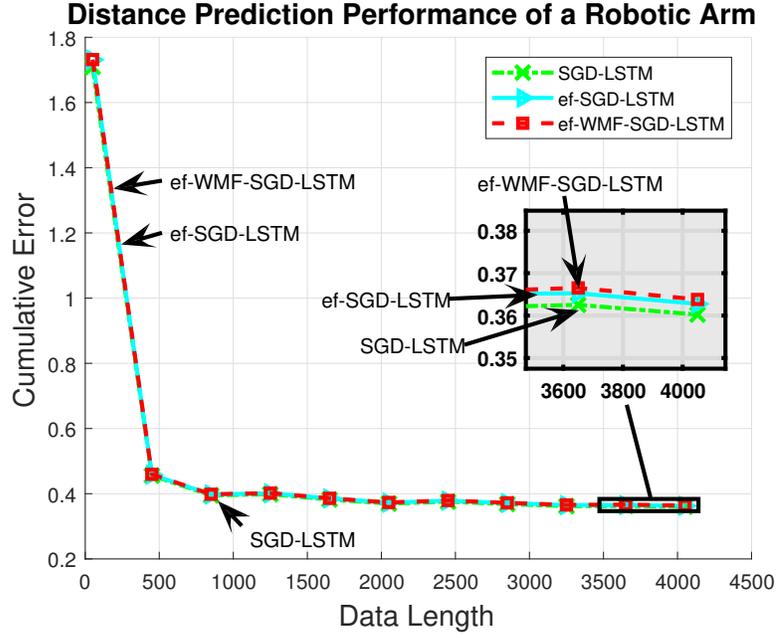


Figure 4.6: Distance prediction performances of the algorithms with the SGD updates on the kinematic dataset.

Fig. 4.8, we compare the models with both the SGD and EG updates. We observe that Model 1 with the SGD updates achieves a slightly smaller time accumulated error compared to Model 2 and Model 3.

In addition to the kinematic dataset, we also evaluate the performances on the elevators dataset [60], 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 $\mathbf{X}_t \in \mathbb{R}^{18}$, $m = 18$ and select the learning rate as $\eta = 0.1$. The rank for Model 3 is 2. In Fig. 4.9, all the models using the EG updates outperform the models using the SGD updates, which arises from the sparseness of \mathbf{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 due to having less operational complexity and smaller number of network parameters compared to the other models.

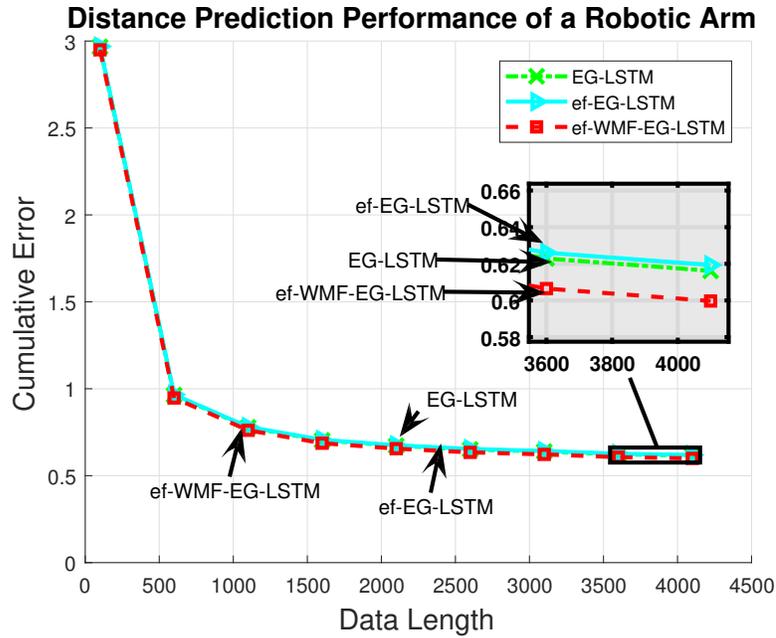


Figure 4.7: Distance prediction performances of the algorithms with the EG updates on the kinematic dataset.

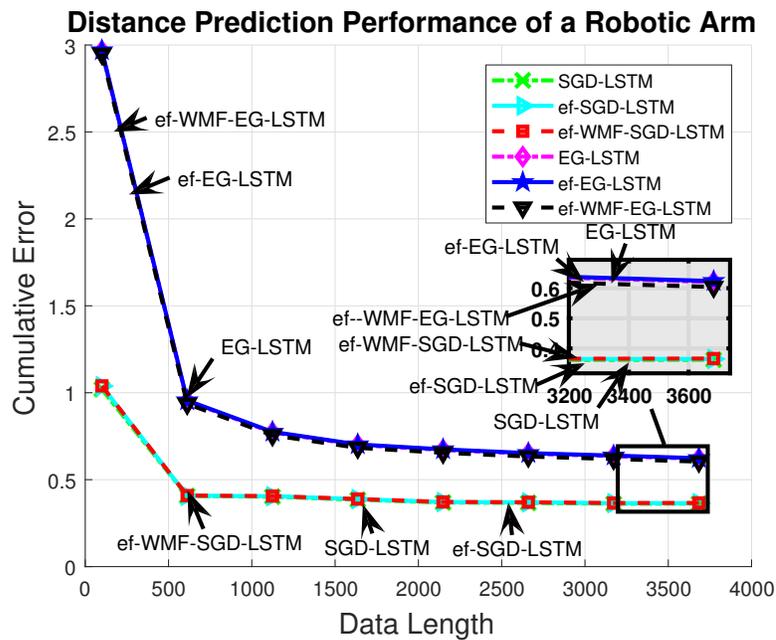


Figure 4.8: Comparison of sequential prediction performances of the algorithms on the kinematic dataset.

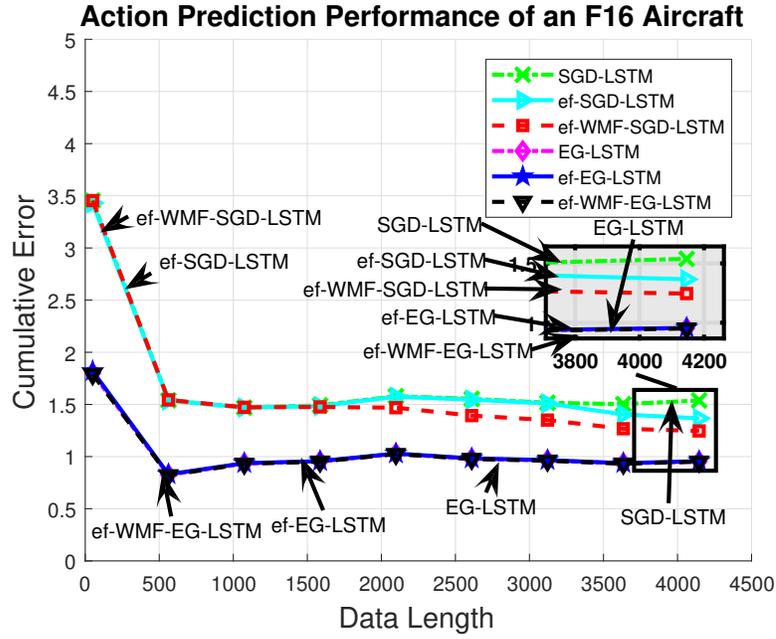


Figure 4.9: Movement prediction performances of the algorithms on the elevators dataset.

4.2.3 Rank Effect on WMF

In this subsection, we illustrate the effects of the rank on the performances of the introduced models. For this purpose, we use the Alcoa Corporation stock price dataset. In Table 4.1, we observe that as the rank decreases, the training time also decreases for the LSTM based WMF models with both SGD and EG updates and the time accumulated 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, one can significantly reduce the number of parameters to be trained in an LSTM network while enjoying high performance. Based on this 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.

	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 4.1: Training times (in seconds) and time accumulated errors for the WMF algorithms using different rank weight matrices on the Alcoa Corporation stock price dataset. Note that this experiment is performed with a computer that has i5-6400 processor, 2.7 GHz CPU and 16 GB RAM.

4.2.4 LSTM and GRU Neural Networks

In this subsection, we evaluate performances of the algorithms on the real-life and financial datasets. Since our approach is generic in the sense that it can be applied to any RNN structure, we also include GRU based algorithms to provide comparative analysis. The GRU network is defined by the following equations [22]:

$$\tilde{\mathbf{z}}_{t,j} = \sigma \left(\mathbf{W}^{(\tilde{z})} \mathbf{x}_{t,j} + \mathbf{R}^{(\tilde{z})} \mathbf{h}_{t,j-1} \right) \quad (4.44)$$

$$\mathbf{r}_{t,j} = \sigma \left(\mathbf{W}^{(r)} \mathbf{x}_{t,j} + \mathbf{R}^{(r)} \mathbf{h}_{t,j-1} \right) \quad (4.45)$$

$$\tilde{\mathbf{y}}_{t,j} = g \left(\mathbf{W}^{(y)} \mathbf{x}_{t,j} + \mathbf{r}_{t,j} \odot (\mathbf{R}^{(y)} \mathbf{h}_{t,j-1}) \right) \quad (4.46)$$

$$\mathbf{y}_{t,j} = \tilde{\mathbf{y}}_{t,j} \odot \tilde{\mathbf{z}}_{t,j} + \mathbf{y}_{t,j-1} \odot (1 - \tilde{\mathbf{z}}_{t,j}), \quad (4.47)$$

where $\mathbf{x}_{t,j} \in \mathbb{R}^p$ is the input vector, $\mathbf{y}_{t,j} \in \mathbb{R}^m$ is the output vector. Here, $\tilde{\mathbf{z}}_{t,j}$ and $\mathbf{r}_{t,j}$ 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 energy efficient version of the GRU network, we apply the matrix factorization method and ef-operator as in the LSTM case.

Here, we select the same parameters with the previous subsections. Since Model 3 provides the best performance in the previous sections, we compare the LSTM and GRU Networks on three datasets using Model 3. For the Alcoa Corporation stock price dataset, the LSTM based algorithm with the SGD updates achieves the smallest steady state error as shown in Fig. 4.10. In Fig. 4.11, the GRU based algorithm with the SGD updates outperforms the other network

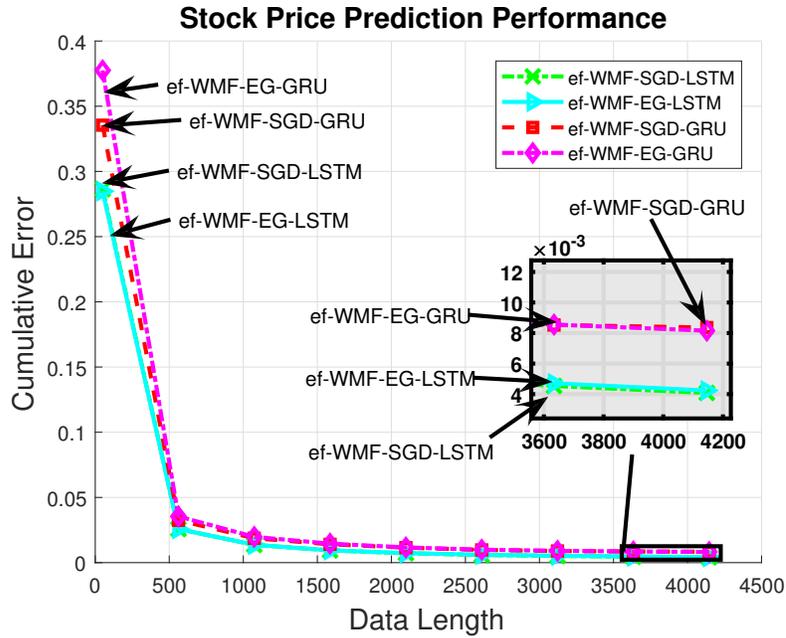


Figure 4.10: Comparison of energy efficient LSTM and GRU networks on the Alcoa Corporation dataset.

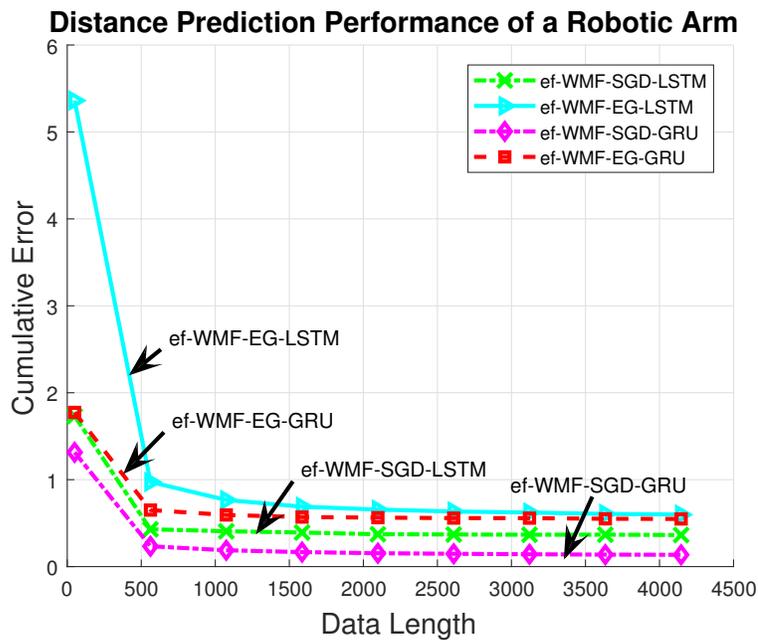


Figure 4.11: Comparison of energy efficient LSTM and GRU networks on the kinematic dataset.

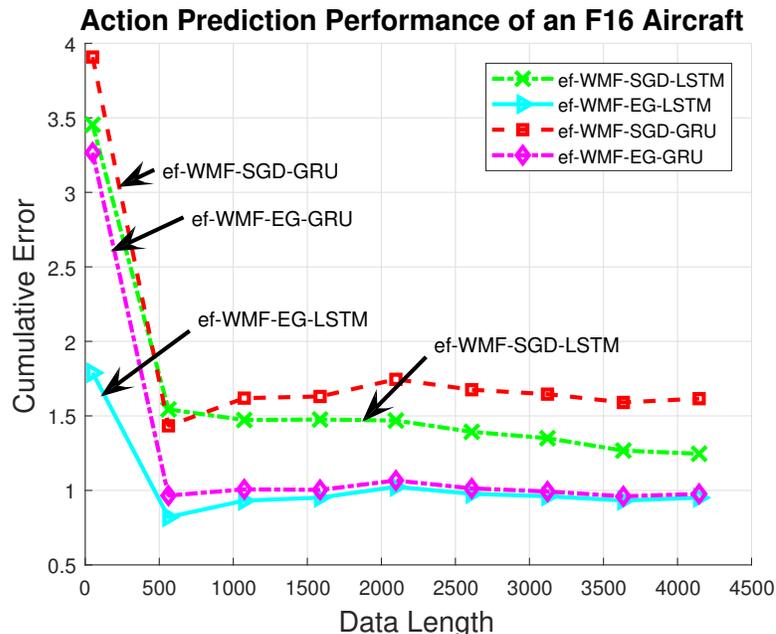


Figure 4.12: Comparison of energy efficient LSTM and GRU networks on the elevators dataset.

models. For the elevators dataset, the LSTM based algorithm with the EG updates achieves the smallest steady state error among all the network models as illustrated in Fig 4.12. 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 [69] for each dataset in Table 4.2. We observe that our approach almost halves the energy consumption for each case and as the dimensionality of the dataset increases, the provided energy efficiency even further increases.

4.2.5 Training Times and Structural Complexity

In this subsection, we first provide the training times (in seconds) of all the LSTM network based models with both the SGD and EG updates. We then give the total number of network parameters for each model, i.e., the structural complexity

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

Table 4.2: 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 $45nm$ CMOS process.

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 4.3: Training times (in seconds) of the introduced energy efficient LSTM networks.

of the corresponding model. Finally, we compare all the models based on both training times, the number of network parameters and error performance.

In Table 4.3, we provide the training times of all the network models for each dataset. 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, however, it does not have the smallest cumulative errors as stated in Table 4.4. Model 3 achieves intermediate training times (and the fastest training when the data size is large as in the elevators dataset) and the smallest cumulative errors as shown in Table 4.3 and Table 4.4 respectively. In Table 4.5, 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

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 4.4: Time accumulated errors of the introduced algorithms.

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

Table 4.5: Total number of parameters to be learnt for the introduced networks.

models.

Chapter 5

Conclusion

In this thesis, we investigate online learning with RNNs. Particularly, in Chapter 2, we study the nonlinear regression problem in an online setting and introduce novel LSTM, i.e., an advanced RNN architecture, based online algorithms for data regression. We then introduce low complexity and effective online training methods for these algorithms. We achieve these by first proposing novel regression algorithms to compute the final estimate, where we introduce 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 [17,19,25] 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 our simulation section, thanks to the generic structure of our approach, we also introduce 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 chapter. Furthermore, we demonstrate significant performance improvements achieved by the introduced algorithms with respect to the conventional methods [18,23] over

several different datasets (used in this chapter).

In Chapter 3, we study online training of the LSTM architecture in a distributed network of nodes for regression and introduce online distributed training algorithms for variable length data sequences. We first propose a generic LSTM based model for variable length data inputs. In order to train this model, we put the model equations in a nonlinear state space form. Based on this form, we introduce distributed extended Kalman and particle filtering based online training algorithms. In this way, we obtain effective training algorithms for our LSTM based model. Here, our distributed particle filtering algorithm guarantees convergence to the optimal centralized parameter estimation in the MSE sense under certain conditions. We achieve this performance with communication and computational complexity in the order of the first order methods [5]. Through simulations involving real life and financial data, we illustrate significant performance improvements with respect to the state of the art methods [18, 23].

In Chapter 4, we study variable length data regression in an online framework and introduce an energy efficient regression structure based on the LSTM network. In particular, we introduce 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 introduce online training algorithms based on the SGD [23] and EG [40] 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 also introduce 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.

Bibliography

- [1] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*. Cambridge university press, 2006.
- [2] D. F. Specht, “A general regression neural network,” *IEEE Transactions on Neural Networks*, vol. 2, 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,” *Digital Signal Processing*, vol. 4, no. 4, pp. 207–221, 1994.
- [4] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A search space odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–11, 2016.
- [5] A. C. Tsoi, “Gradient based learning methods,” in *Adaptive processing of sequences and data structures*, pp. 27–62, Springer, 1998.
- [6] S. Hochreiter, *Untersuchungen zu dynamischen neuronalen Netzen*. PhD thesis, diploma thesis, institut für informatik, lehrstuhl prof. brauer, technische universität münchen, 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 Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–13, 2016.
- [8] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

- [9] U. Shaham, A. Cloninger, and R. R. Coifman, “Provable approximation properties for deep neural networks,” *CoRR*, vol. abs/1509.07385, 2015.
- [10] M. Hermans and B. Schrauwen, “Training and analysing deep recurrent neural networks,” in *Advances in Neural Information Processing Systems*, pp. 190–198, 2013.
- [11] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, pp. 157–166, Mar 1994.
- [12] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.
- [13] F. A. Gers, J. A. Schmidhuber, and F. A. Cummins, “Learning to forget: Continual prediction with LSTM,” *Neural Comput.*, vol. 12, pp. 2451–2471, Oct. 2000.
- [14] *ARMA Modeling and Forecasting*, pp. 89–123. New York, NY: Springer New York, 2003.
- [15] J. Mazumdar and R. G. Harley, “Recurrent neural networks trained with backpropagation through time algorithm to estimate nonlinear load harmonic currents,” *IEEE Transactions on Industrial Electronics*, vol. 55, no. 9, pp. 3484–3491, 2008.
- [16] H. Jaeger, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the “echo state network” approach*. GMD-Forschungszentrum Informationstechnik, 2002.
- [17] A. H. Sayed, *Fundamentals of adaptive filtering*. John Wiley & Sons, 2003.
- [18] J. A. Pérez-Ortiz *et al.*, “Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets,” *Neural Networks*, vol. 16, no. 2, pp. 241–250, 2003.
- [19] B. D. Anderson and J. B. Moore, *Optimal filtering*. 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 *Proceedings of the 27th International Conference on Neural Information Processing Systems, NIPS’14*, (Cambridge, MA, USA), pp. 2933–2941, MIT Press, 2014.
- [21] P. M. Djuric, J. H. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M. F. Bugallo, and J. Miguez, “Particle filtering,” *IEEE signal processing magazine*, vol. 20, no. 5, pp. 19–38, 2003.
- [22] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.
- [23] A. W. Smith and D. Zipser, “Learning sequential structure with the real-time recurrent learning algorithm,” *International Journal of Neural Systems*, vol. 1, no. 02, pp. 125–131, 1989.
- [24] T. Ergen and S. S. Kozat, “Efficient online learning algorithms based on LSTM neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2018.
- [25] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking,” *IEEE Transactions on signal processing*, vol. 50, no. 2, pp. 174–188, 2002.
- [26] S. Vosoughi, P. Vijayaraghavan, and D. Roy, “Tweet2vec: Learning tweet embeddings using character-level CNN-LSTM encoder-decoder,” *CoRR*, vol. abs/1607.07514, 2016.
- [27] D. Wilson and T. R. Martinez, “The general inefficiency of batch training for gradient descent learning,” *Neural Networks*, vol. 16, no. 10, pp. 1429 – 1451, 2003.
- [28] X. Wu *et al.*, “Data mining with big data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, pp. 97–107, Jan 2014.

- [29] C. Yan, Y. Zhang, J. Xu, F. Dai, L. Li, Q. Dai, and F. Wu, “A highly parallel framework for hevc coding unit partitioning tree decision on many-core processors,” *IEEE Signal Processing Letters*, vol. 21, pp. 573–576, May 2014.
- [30] C. Yan, Y. Zhang, J. Xu, F. Dai, J. Zhang, Q. Dai, and F. Wu, “Efficient parallel framework for hevc motion estimation on many-core processors,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, pp. 2077–2089, Dec 2014.
- [31] K. Yuan *et al.*, “On the convergence of decentralized gradient descent,” *SIAM Journal on Optimization*, vol. 26, no. 3, pp. 1835–1854, 2016.
- [32] F. S. Cattivelli and A. H. Sayed, “Diffusion strategies for distributed kalman filtering and smoothing,” *IEEE Transactions on Automatic Control*, vol. 55, pp. 2069–2084, Sept 2010.
- [33] R. Kiros, “Training neural networks with stochastic hessian-free optimization,” *CoRR*, vol. abs/1301.3641, 2013.
- [34] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, “A stochastic quasi-newton method for large-scale optimization,” *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 1008–1031, 2016.
- [35] S. H. Lee and M. West, “Convergence of the markov chain distributed particle filter (MCDPF),” *IEEE Transactions on Signal Processing*, vol. 61, pp. 801–812, Feb 2013.
- [36] T. Ergen and S. S. Kozat, “Online training of LSTM networks in distributed systems for variable length data sequences,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–7, 2017.
- [37] F. A. Gers, J. A. Pérez-Ortiz, D. Eck, and J. Schmidhuber, “DEKF-LSTM,” in *ESANN*, pp. 369–376, 2002.
- [38] D. Monner and J. A. Reggia, “A generalized LSTM-like training algorithm for second-order recurrent neural networks,” *Neural Networks*, vol. 25, pp. 70 – 83, 2012.

- [39] A. Graves and J. Schmidhuber, “Frame-wise phoneme classification with bidirectional LSTM and other neural network architectures,” *Neural Networks*, vol. 18, no. 5, pp. 602 – 610, 2005. IJCNN 2005.
- [40] J. Kivinen and M. K. Warmuth, “Exponentiated gradient versus gradient descent for linear predictors,” *Information and Computation*, vol. 132, no. 1, pp. 1–63, 1997.
- [41] S. I. Hill and R. C. Williamson, “Convergence of exponentiated gradient algorithms,” *IEEE Transactions on Signal Processing*, vol. 49, pp. 1208–1215, Jun 2001.
- [42] A. R. Omondi and J. C. Rajapakse, *FPGA implementations of neural networks*, vol. 365. Springer, 2006.
- [43] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.
- [44] H. Tuna, I. Onaran, and A. E. Cetin, “Image description using a multiplier-less operator,” *IEEE Signal Processing Letters*, vol. 16, pp. 751–753, Sept 2009.
- [45] C. E. Akbas, A. Bozkurt, A. E. Cetin, R. Cetin-Atalay, and A. Uner, “Multiplication-free neural networks,” in *2015 23rd Signal Processing and Communications Applications Conference (SIU)*, pp. 2416–2418, May 2015.
- [46] A. Afrasiyabi, B. Nasir, O. Yildiz, F. T. Y. Vural, and A. E. Cetin, “An energy efficient additive neural network,” in *2017 25th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, May 2017.
- [47] A. Afrasiyabi, O. Yildiz, B. Nasir, F. T. Yarman-Vural, and A. E. Çetin, “Energy saving additive neural network,” *CoRR*, vol. abs/1702.02676, 2017.
- [48] O. Kuchaiev and B. Ginsburg, “Factorization tricks for LSTM networks,” *CoRR*, vol. abs/1703.10722, 2017.

- [49] N. Srinivasan, V. Ravichandran, K. L. Chan, J. R. Vidhya, S. Ramakrishnan, and S. M. Krishnan, “Exponentiated backpropagation algorithm for multilayer feedforward neural networks,” in *Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on*, vol. 1, pp. 327–331 vol.1, Nov 2002.
- [50] Z. Li, Y. Li, F. Yu, and D. Ge, “Adaptively weighted support vector regression for financial time series prediction,” in *2014 International Joint Conference on Neural Networks (IJCNN)*, pp. 3062–3065, July 2014.
- [51] I. Patras and E. Hancock, “Regression-based template tracking in presence of occlusions,” in *Image Analysis for Multimedia Interactive Services, 2007. WIAMIS '07. Eighth International Workshop on*, pp. 15–15, June 2007.
- [52] M. B. Douglas and G. Donald, *Nonlinear regression analysis and its applications*. John Wiley & Sons, 1988.
- [53] Y. Ho and R. Lee, “A bayesian approach to problems in stochastic estimation and control,” *IEEE Transactions on Automatic Control*, vol. 9, no. 4, pp. 333–339, 1964.
- [54] M. Enescu, M. Sirbu, and V. Koivunen, “Recursive estimation of noise statistics in kalman filter based mimo equalization,” *the proceedings of XXVI-Ith General Assembly of the International Union of Radio Science (URSI), Maastricht the Netherlands*, pp. 17–24, 2002.
- [55] A. Kong, J. S. Liu, and W. H. Wong, “Sequential imputations and bayesian missing data problems,” *Journal of the American statistical association*, vol. 89, no. 425, pp. 278–288, 1994.
- [56] A. Doucet, S. Godsill, and C. Andrieu, “On sequential monte carlo sampling methods for bayesian filtering,” *Statistics and computing*, vol. 10, no. 3, pp. 197–208, 2000.
- [57] N. Bergman, “Recursive bayesian estimation,” *Department of Electrical Engineering, Linköping University, Linköping Studies in Science and Technology. Doctoral dissertation*, vol. 579, 1999.

- [58] X. L. Hu, T. B. Schon, and L. Ljung, “A basic convergence result for particle filtering,” *IEEE Transactions on Signal Processing*, vol. 56, pp. 1337–1348, April 2008.
- [59] C. E. Rasmussen, R. M. Neal, G. Hinton, D. Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani, “Delve data sets.” <http://www.cs.toronto.edu/~delve/data/datasets.html>.
- [60] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, “KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework,” *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, no. 2-3, pp. 255–287, 2011.
- [61] L. Torgo, “Regression data sets.” <http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html>.
- [62] “Summary for alcoa inc. common stock.” <http://finance.yahoo.com/quote/AA?ltr=1>.
- [63] E. W. Frees, “Regression modelling with actuarial and financial applications.” <http://instruction.bus.wisc.edu/jffrees/jffreesbooks/Regression%20Modeling/BookWebDec2010/data.html>.
- [64] M. Lichman, “UCI machine learning repository.” <http://archive.ics.uci.edu/ml>, 2013.
- [65] M. Liberatore, “Umass trace repository.” <http://traces.cs.umass.edu/index.php/Sensors/Sensors>.
- [66] J. Kominek and A. W. Black, “Cmu arctic database.” <http://www.festvox.org/cmuarctic/index.html>.
- [67] N. D. Vanli and S. S. Kozat, “A comprehensive approach to universal piecewise nonlinear regression based on trees,” *IEEE Transactions on Signal Processing*, vol. 62, pp. 5471–5486, Oct 2014.
- [68] R. N. Bracewell and R. N. Bracewell, *The Fourier transform and its applications*, vol. 31999. McGraw-Hill New York, 1986.

[69] M. Horowitz, “Energy table for 45nm process.”