EFFICIENT ONLINE TRAINING ALGORITHMS FOR 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
Nuri Mert Vural
December 2020

EFFICIENT ONLINE TRAINING ALGORITHMS FOR RECURRENT NEURAL NETWORKS

By Nuri Mert Vural

December 2020

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)

Nail Akar

Salih Ergüt

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

EFFICIENT ONLINE TRAINING ALGORITHMS FOR RECURRENT NEURAL NETWORKS

Nuri Mert Vural
M.S. in Electrical and Electronics Engineering
Advisor: Süleyman Serdar Kozat
December 2020

Recurrent Neural Networks (RNNs) are widely used for online regression due to their ability to learn nonlinear temporal dependencies. As an RNN model, Long-Short-Term-Memory Networks (LSTMs) are commonly preferred in practice, since these networks are capable of learning long-term dependencies while avoiding the exploding gradient problem. On the other hand, the performance improvement of LSTMs usually comes with the price of their large parameter size, which makes their training significantly demanding in terms of computational and data requirements.

In this thesis, we address the computational challenges of LSTM training. We introduce two training algorithms, designed for obtaining the online regression performance of LSTMs with less computational requirements than the state-of-the-art. The introduced algorithms are truly online, i.e., they do not assume any underlying data generating process and future information, except that the dataset is bounded. We discuss theoretical guarantees of the introduced algorithms, along with their asymptotic convergence behavior. Finally, we demonstrate their performance through extensive numerical studies on real and synthetic datasets, and show that they achieve the regression performance of LSTMs with significantly shorter training times.

Keywords: Long-Short-Term-Memory, Recurrent Neural Networks, Online Optimization, Kalman Filtering, Sequential Learning.

ÖZET

YİNELEYİCİ SİNİR AĞLARI İÇİN VERİMLİ ÇEVRİMİCİ EĞİTİM ALGORİTMALARI

Nuri Mert Vural
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Aralık 2020

Yineleyici sinir ağları doğrusal olmayan zamansal bağıntılarını öğrenebilmeleri nedeniyle çevrimiçi öğrenme uygulamalarında yaygın olarak kullanılmaktadır. Uzun-Kısa-Soluklu Bellek (UKSB) mimarisi uzun süreçli bağıntıları gradyan patlaması yaşamadan öğrenebildiği için uygulamada en sık tercih edilen yinelemeli ağ yapılarından biridir. Ancak, UKSB'ler bu performans kazanımlarını oldukça yüksek sayıda parametre ile sağlamaktadır. Bu durum UKSB eğitimini veri ve hesaplama gereksinimleri açısından oldukça pahalı bir işleme dönüştürmektedir.

Tezimizde UKSB eğitiminin hesaplama zorluklarını ele alıyoruz. UKSB'lerin çevrimiçi regresyon performansına daha düşük hesap yüküyle ulaşan iki adet eğitim algoritması tanıtıyoruz. Tanıttığımız algoritmalar tamamiyle çevrimiçi olup, verinin sınırlı bir dizi olması dışında veriye dair hiçbir ön kabul yapmamaktadır. Tezimizin devamında tanıttığımız algoritmaların teorik garantilerini ve asimptotik yakınsama özelliklerini tartışacağız. Ayrıca, gerçek ve sentetik veriler üzreinde yaptığımız çok sayıda deneyle algoritmalarımızın literatürdeki en son gelişmeleri yansıtan algoritmalara göre performans üstünlüğünü göstereceğiz.

Anahtar sözcükler: Uzun-Kısa-Soluklu Bellek, Yineleyici Sinir Ağları, Çevrimiçi Eniyileme, Kalman Filtresi, Dizi Öğrenimi.

Acknowledgement

I would like to thank my advisor, Prof. Süleyman Serdar Kozat, for his wise supervision and endless encouragement throughout my M.S. study. I would like to express my profound gratitude to him as I learned how to be professional and productive thanks to the work ethics in Prof. Kozat's team.

I would like to thank Prof. Nail Akar and Prof. Salih Ergüt for allocating their time to investigate my work. Moreover, I would like to thank them also for all their help, support, and encouragement throughout my M.S. study.

I am grateful to all faculty and staff at the Department of Electrical and Electronics Engineering at Bilkent for creating a great academic environment and a second home. I thank all my office mates, Selin Coşan, Emir Ceyani, Hakan Gökçesu, Fatih İlhan, Fatih Aslan, Dogan Can Çiçek, and Selim Fırat Yılmaz for their help and collaborations. Selin Coşan deserves a special mention. I was extremely fortunate to share my office with her as she was one of the most helpful people to me at Bilkent.

I am fortunate to have had great friends during my time in Ankara. I am thankful to Uğur Çakıcı, Sena Kocabörek, Hilal Öztürk, Damla Akoluk, and Burak Yılmaz, who made this period of my life very enjoyable. Additionally, I would like to express my sincerest thanks to Mert Diner and Nihan Önder Kürklü for their care and support. Thanks to their effort, I have improved my life quality significantly.

Finally, I am grateful to my family, my parents Fatma and Lütfi, and my sister Merve. There are no words to describe their support. This thesis is dedicated to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Estimating an unknown desired signal online is one of the main subjects of interest in the contemporary machine learning literature [1–4]. In this problem, a learner sequentially receives a data sequence related to a desired signal to predict the signal's next value. This problem (also known as online regression) is extensively studied due to its applications in a wide set of problems, such as neural network training [2, 4–6], signal processing [7, 8], or machine learning [1, 3].

For online regression, there exists a wide range of established methods in machine learning literature [1, 3, 9, 10]. Adopting neural networks is one of the most preferred among them, due to their relative ease of training and high generalization capacity [11, 12]. Among several neural network architectures, recurrent neural networks (RNNs) are the most commonly used ones for online regression, since they are able to identify sequential patterns and learn temporal behavior as well [13]. Despite their performance improvements, however, simple RNNs have been shown to fail to capture long-term dependencies due to vanishing and exploding gradient problems, which significantly restricts their performance in real-life applications [14]. To resolve this issue, a sophisticated RNN architecture with several control structures, namely long-short-term-memory network (LSTM), was introduced [15]. Through many variants, LSTMs have seen remarkable empirical

success in a broad range of sequential learning tasks, including online regression [16–19].

LSTMs maintain constant backward flow in the error signal to overcome the vanishing gradient problem. They utilize three gates, i.e., input, forget, and output gates, to regulate the information flow through the next steps. In each gate, an independent set of weights, i.e., gate weights, are employed to learn the optimal information regulation in a data-dependent manner. With this structure, LSTMs have been shown to improve the performance of simple RNNs dramatically [15, 20]. To provide this improvement, however, LSTMs require significantly larger number of parameters than simple RNNs, which makes their training more demanding in terms of computational and data requirements.

In this thesis, we address the computational challenges of the LSTM training. We introduce two online optimization algorithms that achieve the online regression performance of LSTMs with a significantly shorter training time. To develop our algorithms, we use several tools from the fields of convex optimization [1, 3] and nonconvex sequential learning [4].

In Chapter 2, we consider second-order optimization. Since the computational requirements of the existing online second-order training algorithms is at least quadratic in the number of the network parameters, the large parameter size of LSTMs makes it especially difficult to train them with second-order algorithms [21, 22]. To resolve this issue, we introduce a highly efficient extended Kalman Filter (EKF) based training algorithm for LSTM-based online learning. To design our algorithm, we use the independent EKF approach, i.e., a variant of the EKF learning algorithm that uses an approximation to the state covariance matrix in a block-diagonal form. We develop an IEKF-based algorithm for LSTMs, which guarantees errors to converge to a small interval.

The main contributions of Chapter 2 are as follows: 1) We introduce a second-order training algorithm with a performance guarantee for LSTM-based online learning algorithm. 2) Since we construct our algorithm on the independent EKF method [23, 24], our algorithm provides significant computational savings in

comparison to the state-of-the-art second-order optimization methods [21, 25]. 3) Our algorithm can be used in a broad range of adaptive signal processing applications, since it does not assume any underlying data generating process or future information, except that the target sequence is bounded [19, 26]. 4) Through an extensive set of experiments involving synthetic and real data, we demonstrate significant performance gains achieved by the proposed algorithm with respect to the state-of-the-art algorithms. 4) In the experiments, we particularly show that our algorithm provides a considerable increase in the accuracy of the widely-used adaptive learning method, i.e., Adam [6], RMSprop [5], and DEKF [24], and very close performance to EKF [21] with a 10 to 15 times reduction in the run-time.

In Chapter 3, we consider first-order optimization. Due to our efficiency concerns, here, we use simple RNNs (SRNNs) [13, 27]. In this part, we introduce a first-order training algorithm for SRNNs, which achieves the online regression performance of LSTMs in a considerably shorter training time. We also provide strong theoretical analysis to support our experimental results by providing regret bounds on the converge rate of our algorithm.

The main contributions of Chapter 3 are as follows: 1) To the best of our knowledge, our study is the first study that obtains the online regression performance of LSTMs with SRNNs by using a first-order training algorithm. 2) Since SRNNs have four times fewer parameters compared to LSTMs for a fixed hidden layer size, our algorithm obtains the performance of LSTMs in a 2 to 3 times shorter training time. 3) We support our experimental results with strong theoretical analysis and prove that our algorithm converges to the local optimum parameters. Moreover, we do not make any assumption on the data sequence to guarantee convergence, except that the input/target sequences are bounded. Therefore, our algorithm can be safely used in any online regression scenario in a theoretically justified sense. 4) Through an extensive set of experiments on real and synthetic data, we verify our theoretical work and demonstrate significant performance improvements of our algorithm with respect to LSTMs and the state-of-the-art training methods.

Notation: All vectors are column vectors and denoted by boldface lower case

letters. Matrices are represented by boldface capital letters. The $\mathbf{0}$ (respectively $\mathbf{1}$) represents a matrix or a vector of all zeros (respectively ones), whose dimensions are understood from on the context. $\mathbf{I}$ is the identity matrix, whose dimensions are understood from the context. $\|\cdot\|$ and $\mathrm{Tr}(\cdot)$ denote the Euclidean norm and trace operators. Given two matrices $\mathbf{A}$ and $\mathbf{B}$, $\mathbf{A} > \mathbf{B}$ (respectively $\geq$) means that $(\mathbf{A} - \mathbf{B})$ is a positive definite (respectively semi-positive definite) matrix. Given two vectors $\mathbf{x}$ and $\mathbf{y}$, $[\mathbf{x}; \mathbf{y}]$ is their vertical concatenation. We use bracket notation $[n]$ to denote the set of the first n positive integers, i.e., $[n] = \{1, \cdots, n\}$.

Chapter 2

An Efficient and Effective Second-Order Training Algorithm for LSTM-based Online Learning

In this chapter, we introduce a highly efficient and effective extended Kalman filter (EKF) based second-order training algorithm for LSTM-based online learning. Our algorithm is truly online, i.e., it does not assume any underlying data generating process and future information, except that the target sequence is bounded. To design an efficient algorithm, we use the independent EKF (IEKF) approach [23], i.e., we approximate the state covariance matrix in a block-diagonal form by neglecting the covariance terms between the weights belonging to different nodes. Differing from the previous work on IEKF [23, 24], we provide strong theoretical analysis for our algorithm by deriving an adaptive hyper-parameter selection strategy that guarantees errors to converge to a small interval under the only assumption that the target sequence is bounded. Through simulations, we demonstrate significant performance gains achieved by our algorithm with respect to the state-of-the-art methods. We mainly show that our algorithm provides a considerable improvement in the accuracy of the widely-used adaptive methods Adam [6], RMSprop [5], and DEKF [24], and very close performance to EKF with a 10 to 15 times reduction in the run-time.

2.1 Related Work

Various optimization algorithms in the deep learning literature, such as Adam [6], RMSprop [5] or SGD [28], can be used to train LSTMs online. Among the widely used first-order algorithms, the adaptive learning methods, e.g., Adam and RMSprop, are observed to provide faster convergence than the plain SGD [29]. The performance gains of Adam and RMSprop are justified with their approximate Hessian-based preconditioning, which let them use the second-order properties of the error surface to improve the convergence speed [30]. However, the approximation employed in the first-order adaptive methods considers only the diagonal elements of the Hessian matrix, which severely limits their performance in comparison to the second-order algorithms that utilize a full Hessian matrix estimate [25].

For LSTMs, the Hessian-Free and EKF algorithms are capable of utilizing a full Hessian matrix estimate with reasonable time complexity [21, 22]. However, the Hessian-Free algorithm requires a large size of batches to approximate the Hessian matrix, making it impractical for the online settings. On the other hand, the EKF learning algorithm is higly suitable for adaptive learning, as it recursively updates its Hessian estimate, i.e., its state covariance matrix, without using mini-batch statistics. Moreover, EKF is extensively studied in the neural network literature, where it has been repeatedly demonstrated to have faster convergence and better accuracy than the first-order methods [19]. However, EKF requires a quadratic computational complexity in the number of parameters, which is prohibitive for most of the practical applications.

To reduce the computational complexity of EKF, it is common in practice to use a block-digonal approximation for the state covariance matrix, which we refer to as the IEKF approach. The algorithms using the IEKF approach, such as DEKF [24] or MEKA [23], have been shown to provide better performance than the first-order methods with an acceptable reduction in the error performance (due to their computational savings) compared to EKF [23]. Despite their empirical success, however, the existing methods are heuristic-based; hence, they

have no guarantee to converge to an optimum set of weights during the training [24]. Moreover, due to the neglected covariance terms, they are sensitive to the hyperparameter selection and initialization, which degrades their performance compared to that of EKF [21]. In this study, we are interested in reducing the performance difference between EKF and IEKF to provide an efficient and effective second-order training algorithm for LSTM-based adaptive learning. To this end, we introduce an IEKF-based algorithm with an adaptive hyperparameter selection strategy that guarantees errors to converge to a small interval.

There are only a few recent works studying convergence analysis of RNN training with EKF [31, 32]. However, these studies consider basic RNN training with full Hessian approximation, which cannot be easily extended to LSTMs due to high computational complexity of EKF. Hence, differing from the previous works, we use the IEKF approach and introduce an IEKF-based training algorithm with a performance guarantee. Endowed with a theoretical guarantee, our algorithm performs very closely to EKF in a considerably shorter running time. As noted earlier, our algorithm is truly online, i.e., it does not require a priori knowledge on the environment for its performance guarantee, making the proposed algorithm of practical value for a wide range of adaptive signal processing applications, such as time-series prediction [19] or object tracking [26].

2.2 Model and Problem Description

In this chapter, we study the online regression problem, defined as as follows: We sequentially receive bounded target vectors, $\{\mathbf{d}_t\}_{t\geq 1}$, $\mathbf{d}_t \in [-1,1]^{n_d}$, and input vectors, $\{\mathbf{x}_t\}_{t\geq 1}$, $\mathbf{x}_t \in \mathbb{R}^{n_x}$ such that our goal is to estimate $\mathbf{d}_t$ based on our current and past observations $\{\cdots, \mathbf{x}_{t-1}, \mathbf{x}_t\}$.[1] Given our estimate $\hat{\boldsymbol{d}}_t$, which can only be a function of $\{\cdots, \mathbf{x}_{t-1}, \mathbf{x}_t\}$ and $\{\cdots, \mathbf{d}_{t-2}, \mathbf{d}_{t-1}\}$, we suffer the loss $\ell(\mathbf{d}_t, \hat{\boldsymbol{d}}_t)$. The aim is to optimize the network with respect to the loss function $\ell(\cdot,\cdot)$. In this part, we particularly work with the squared error, i.e., $\ell(\mathbf{d}_t, \hat{\boldsymbol{d}}_t) = \|\mathbf{d}_t - \hat{\boldsymbol{d}}_t\|^2$. We note

[1] We assume $\mathbf{d}_t \in [-1,1]^{n_d}$ for notational simplicity; however, our derivations hold for any bounded desired data sequence after shifting and scaling in magnitude.

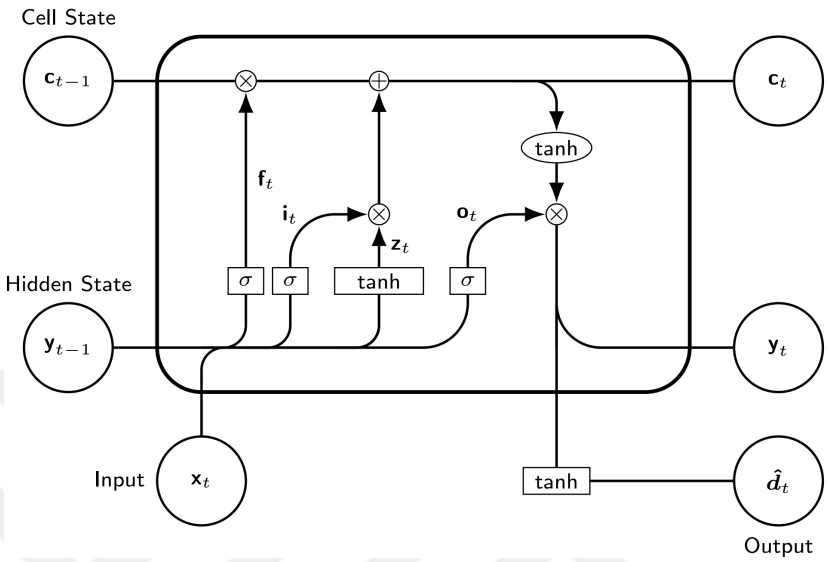

Figure 2.1: The detailed schematic of the equations given in (2.1)-(2.7).

that since we observe the target value $\mathbf{d}_t$ at each time step, i.e., full information setting, all of our results hold in the deterministic sense. We additionally note that our work can be extended to a wide range of cost functions (including the cross-entropy) using the analysis in [33, Section 3].

In this chapter, we study online regression with LSTM-based networks due to their succes in modelling highly nonlinear sequential tasks [15]. As illustrated in Fig. 2.1, we use the most widely used LSTM model, where the activation functions are set to the hyperbolic tangent function and the peep-hole connections are eliminated. As in Fig. 2.1, we use a single hidden layer based on the LSTM structure, and an output layer with the hyperbolic tangent function[2]. Hence, we

[2]We use the hyperbolic tangent function in the output layer to ensure that the model estimates also lay in $[-1, 1]^{n_d}$. However, our results hold for any output activation function, given that the function is differentiable, and its range is sufficient large to include the target vectors.

have:

$$\mathbf{z}_t = \tanh(\mathbf{W}_z[\mathbf{x}_t; \mathbf{y}_{t-1}]) \tag{2.1}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{x}_t; \mathbf{y}_{t-1}]) \tag{2.2}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{x}_t; \mathbf{y}_{t-1}]) \tag{2.3}$$

$$\mathbf{c}_t = \mathbf{i}_t \odot \mathbf{z}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1} \tag{2.4}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{x}_t; \mathbf{y}_{t-1}]) \tag{2.5}$$

$$\mathbf{y}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.6}$$

$$\hat{\boldsymbol{d}}_t = \tanh(\mathbf{W}_d \mathbf{y}_t) \tag{2.7}$$

where $\odot$ denotes the element-wise multiplication, $\mathbf{c}_t \in \mathbb{R}^{n_s}$ is the state vector, $\mathbf{x}_t \in \mathbb{R}^{n_x}$ is the input vector, and $\mathbf{y}_t \in \mathbb{R}^{n_s}$ is the output vector, and $\hat{\boldsymbol{d}}_t \in [-1, 1]^{n_d}$ is our final estimation. Furthermore, $\mathbf{i}_t$, $\mathbf{f}_t$ and $\mathbf{o}_t$ are the input, forget and output gates respectively. The sigmoid function $\sigma(.)$ and the hyperbolic tangent function $\tanh(.)$ applies point wise to the vector elements. The weight matrices are $\mathbf{W}_z, \mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{n_s \times (n_x + n_s)}$ and $\mathbf{W}^{(d)} \in \mathbb{R}^{n_d \times n_s}$. We note that although we do not explicitly write the bias terms, they can be included in (2.1)-(2.7) by augmenting the input vector with a constant dimension.

2.3 Review of EKF-Based Online Training Algorithms

In this section, we review the EKF learning algorithm and derive the update rules of the IEKF learning algorithm within the LSTM-based online learning framework. We note that in the neural network literature, there are two approaches to derive EKF-based learning algorithms: the parallel EKF approach, where both the network weights and hidden state vectors are treated as the states to be estimated by EKF, and the parameter-based EKF approach, where only the network weights are viewed as states to be estimated [24]. In the following, we derive our algorithms by using the parameter-based EKF approach. We prefer to use the parameter-based EKF approach since, unlike the parallel EKF approach, the

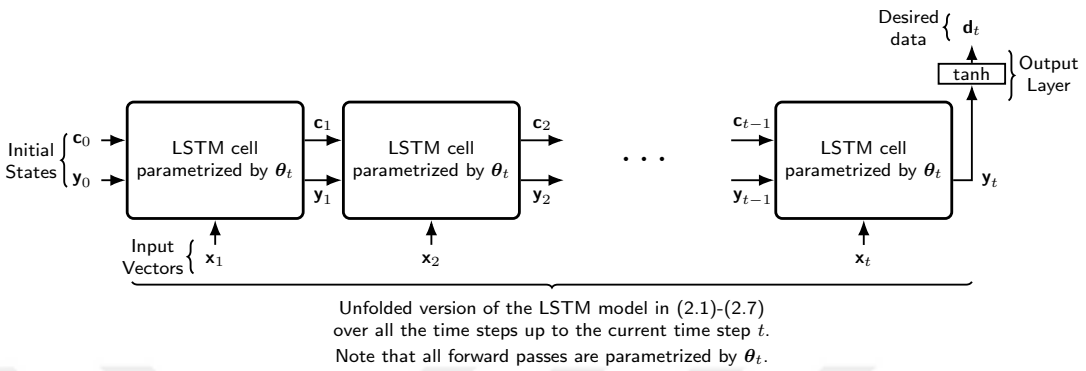

Figure 2.2: The detailed schematic of $h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta}_t)$ given in (2.9). The figure represents the unfolded version of the LSTM model in (2.1)-(2.7) over all the time steps up to the current time step t. Here, the iterations start with the predetermined initial states $\mathbf{y}_0$ and $\mathbf{c}_0$, which are independent of the network weights. Then, the same LSTM forward pass (given in (2.1)-(2.6)) is repeatedly applied to the input sequence $\{\mathbf{x}_t\}$, where the LSTM cells are parametrized by their corresponding weights in $\boldsymbol{\theta}_t$. Finally, the resulting hidden state vector $\mathbf{y}_t$ goes through the output layer function in (2.7), which is parametrized by its corresponding weights in $\boldsymbol{\theta}_t$, and generates the desired data $\mathbf{d}_t$. We note that by the given $h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta}_t)$, the data generating process in (2.8)-(2.9) is fully characterized only by the network weights $\boldsymbol{\theta}_t$, as the parameter-based EKF approach requires.

parameter-based EKF approach allows us to use the Truncated Backpropagation Through Time algorithm [13] to approximate the derivatives efficiently [24]. However, we emphasize that it is possible to adapt our analysis to the parallel EKF approach by using the state-space representation in [34] and changing our mathematical derivations accordingly.

For notational convenience in the following derivations, we introduce two new notations: *1)* We group all the LSTM parameters, i.e., $\mathbf{W}_z, \mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{n_s \times (n_x + n_s)}$ and $\mathbf{W}_d \in \mathbb{R}^{n_d \times n_s}$, into a vector $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$, where $n_\theta = 4n_s(n_s + n_x) + n_s n_d$. *2)* We use $\{\mathbf{x}_t\}$ to denote the input sequence up to time t, i.e., $\{\mathbf{x}_t\} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t\}$.

Now, we are ready to derive the EKF and IEKF learning algorithms.

2.3.1 Online Learning with EKF

In order to convert the LSTM training into a state estimation problem, we model the desired signal as a nonlinear process realized by the LSTM network in (2.1)-(2.7). We note that since we use the parameter-based EKF approach, our desired signal model should be fully characterized by only the network weights $\boldsymbol{\theta}$. Therefore, we describe the underlying process of the incoming data with the following dynamical system:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} \tag{2.8}$$

$$\mathbf{d}_t = h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta}_t). \tag{2.9}$$

Here, we represent the LSTM weights that realize the incoming data stream with a vector $\boldsymbol{\theta}_t \in \mathbb{R}^{n_\theta}$, which is modeled as a stationary process. As detailed in Fig. 2.2, we use $h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta}_t)$ to represent the unfolded version of the LSTM model in (2.1)-(2.7) over all the time steps up to the current time step t, where all forward passes are parametrized by $\boldsymbol{\theta}_t$. Note that the dependence of $h_t(\cdot)$ on t is due to the increased length of the recursion at each time step. The EKF learning algorithm is the EKF applied to the state-space model in (2.8)-(2.9) to estimate the network parameters $\boldsymbol{\theta}_t$. From the optimization perspective, EKF performs an online optimization procedure with the squared loss, aiming to predict the time-series with performance close to the best state-space model formulated as (2.8)-(2.9), i.e., the LSTM model with the (locally) optimum parameters [34].

In the algorithm, we first perform the forward LSTM-propagation with (2.1)-(2.7) by using the parameters $\hat{\boldsymbol{\theta}}_t \in \mathbb{R}^{n_\theta}$, which is our estimate for the optimum weights at time step t. Then, we perform the weight updates with the following formulas:

$$\hat{\boldsymbol{\theta}}_{t+1} = \hat{\boldsymbol{\theta}}_t + \mathbf{G}_t(\mathbf{d}_t - \hat{\boldsymbol{d}}_t) \tag{2.10}$$

$$\mathbf{P}_{t+1} = (\mathbf{I} - \mathbf{G}_t\mathbf{H}_t)\mathbf{P}_t + \mathbf{Q}_t \tag{2.11}$$

$$\mathbf{H}_t = \frac{\partial h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_t} \tag{2.12}$$

$$\mathbf{G}_t = \mathbf{P}_t\mathbf{H}_t^T(\mathbf{H}_t\mathbf{P}_t\mathbf{H}_t^T + \mathbf{R}_t)^{-1}. \tag{2.13}$$

Here, $\mathbf{P}_t \in \mathbb{R}^{n_\theta \times n_\theta}$ is the state covariance matrix, which models the interactions between each pair of the LSTM parameters, $\mathbf{G}_t \in \mathbb{R}^{n_\theta \times n_d}$ is the Kalman gain matrix, and $\mathbf{H}_t \in \mathbb{R}^{n_d \times n_\theta}$ is the Jacobian matrix of $h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta})$ evaluated at $\hat{\boldsymbol{\theta}}_t$. The noise covariance matrices $\mathbf{Q}_t \in \mathbb{R}^{n_\theta \times n_\theta}$ and $\mathbf{R}_t \in \mathbb{R}^{n_d \times n_d}$ are artifically introduced to the algorithm to enhance the training performance [21]. In order to efficiently implement the algorithm, we use diagonal matrices for the artificial noise terms, i.e., $\mathbf{Q}_t = q_t \mathbf{I}$ and $\mathbf{R}_t = r_t \mathbf{I}$, where $q_t, r_t > 0$. Due to (2.11) and (2.13), the computational complexity of the EKF learning algorithm is $O(n_\theta^2)$, which is usually prohibitive for the online settings.[3]

2.3.2 Online Learning with IEKF

In this study, we use IEKF to develop an efficient EKF-based training algorithm for LSTM-based online learning. In IEKF, we approximate to the state covariance matrix by using a block-diagonal matrix approximation. To this end, we assume each neural node in LSTM as an independent subsystem, and use a different (and independent) EKF learning algorithm to learn the weight in each node. Let us denote the LSTM nodes with the first $(4n_s + n_d)$ integers, i.e., $[(4n_s + n_d)] = \{1, \cdots, (4n_s + n_d)\}$, and use i to index the nodes, i.e., $i \in [(4n_s + n_d)]$. Then, we perform the weight updates in IEKF with the following:

$$\text{for } i = 1, \cdots, (4n_s + n_d)$$

$$\hat{\boldsymbol{\theta}}_{i,t+1} = \hat{\boldsymbol{\theta}}_{i,t} + \mathbf{G}_{i,t}(\mathbf{d}_t - \hat{\boldsymbol{d}}_t) \tag{2.14}$$

$$\mathbf{P}_{i,t+1} = (\mathbf{I} - \mathbf{G}_{i,t}\mathbf{H}_{i,t})\mathbf{P}_{i,t} + q_t \mathbf{I} \tag{2.15}$$

$$\mathbf{H}_{i,t} = \frac{\partial h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i}\Big|_{\boldsymbol{\theta}_i = \hat{\boldsymbol{\theta}}_{i,t}} \tag{2.16}$$

$$\mathbf{G}_{i,t} = \mathbf{P}_{i,t}\mathbf{H}_{i,t}^T(\mathbf{H}_{i,t}\mathbf{P}_{i,t}\mathbf{H}_{i,t}^T + r_{i,t}\mathbf{I})^{-1}. \tag{2.17}$$

Here, $\boldsymbol{\theta}_{i,t}$ and $\hat{\boldsymbol{\theta}}_{i,t}$ denote the weights in the LSTM node with index i. $\mathbf{P}_{i,t} \in \mathbb{R}^{\frac{n_\theta}{(4n_s+n_d)} \times \frac{n_\theta}{(4n_s+n_d)}}$ is the state covariance matrix, $\mathbf{H}_{i,t} \in \mathbb{R}^{n_d \times \frac{n_\theta}{(4n_s+n_d)}}$ is the Jacobian matrix of $h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta}_t)$ and $\mathbf{G}_{i,t} \in \mathbb{R}^{\frac{n_\theta}{(4n_s+n_d)} \times n_d}$ is the Kalman gain matrix corresponding to the LSTM weights in node i. Since we perform (2.15) and (2.17)

[3] We use big-O notation, i.e., $O(f(x))$, to ignore constant factors.

$(4n_s + n_d)$ times, the computational complexity of the IEKF learning algorithm is $O(n_\theta^2/(4n_s + n_d))$. We note that since LSTMs include a large number of nodes in practice, the reduction in the computational requirement with IEKF leads to considerable run-time savings in LSTM-based online learning. However, as noted earlier, the existing IEKF methods are heuristic-based, and thus, sensitive to hyperparameter selection and initialization, which reduces their performance in comparison to EKF. Therefore, to provide an efficient and effective online learning procedure, we introduce an IEKF-based LSTM training algorithm with a theoretical performance guarantee in the following section.

2.4 Algorithm Development

In this section, we introduce an online IEKF-based training algorithm with a performance guarantee. We present our algorithm in two subsections: In the first subsection, we develop our algorithm by assuming that we have prior information about the data (in the form of the non-linear term in the error dynamics) before the training. In the second subsection, we drop this assumption and extend our algorithm to a truly online form, where the final algorithm sequentially learns the non-linear term without a priori knowledge of the data while preserving its performance guarantee.

For the analysis in the following subsections, we write the error dynamics of the independent EKF structures. To this end, we first write the Taylor series expansion of $h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta}_t)$ around $\hat{\boldsymbol{\theta}}_t$:

$$h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta}_t) = h_t(\{\mathbf{x}_t\}; \hat{\boldsymbol{\theta}}_t) + \mathbf{H}_t(\boldsymbol{\theta}_t - \hat{\boldsymbol{\theta}}_t) + \boldsymbol{\zeta}_t, \tag{2.18}$$

where $\mathbf{H}_t \in \mathbb{R}^{n_d \times n_\theta}$ is the Jacobian matrix of $h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta})$ evaluated at $\hat{\boldsymbol{\theta}}_t$, and $\boldsymbol{\zeta}_t \in \mathbb{R}^{n_d}$ is the non-linear term in the expansion. Note that $\mathbf{d}_t = h_t(\{\mathbf{x}_t\}; \boldsymbol{\theta}_t)$ and $\hat{\boldsymbol{d}}_t = h_t(\{\mathbf{x}_t\}; \hat{\boldsymbol{\theta}}_t)$. For notational simplicity, we introduce two shorthand notations: $\boldsymbol{\chi}_{i,t} = (\boldsymbol{\theta}_{i,t} - \hat{\boldsymbol{\theta}}_{i,t})$, and $\mathbf{e}_t = (\mathbf{d}_t - \hat{\boldsymbol{d}}_t)$. Then, the error dynamics of the

EKF learning algorithm applied to node i can be written as:

$$\mathbf{e}_t = \sum_{i=1}^{(4n_s+n_d)} \mathbf{H}_{i,t}\boldsymbol{\chi}_{i,t} + \boldsymbol{\zeta}_t = \mathbf{H}_{i,t}\boldsymbol{\chi}_{i,t} + \boldsymbol{\zeta}_{i,t}, \tag{2.19}$$

where we consider the effect of partitioning the weights as additional non-linearity for node i, i.e.,

$$\boldsymbol{\zeta}_{i,t} = \sum_{\substack{j=1 \\ j\neq i}}^{(4n_s+n_d)} \mathbf{H}_{j,t}\boldsymbol{\chi}_{j,t} + \boldsymbol{\zeta}_t. \tag{2.20}$$

For now, let us assume that the norm of $\boldsymbol{\zeta}_{i,t}$ is bounded by a scalar value $\overline{\zeta}$ for all the nodes throughout the training, i.e.,

$$\overline{\zeta} \geq \|\boldsymbol{\zeta}_{i,t}\| \quad \text{for all } t \in [T] \text{ and } i \in [(4n_s + n_d)]. \tag{2.21}$$

We note that we will prove (2.21) in the following part (in Theorem 2.3).

2.4.1 Performance Guarantee with Known Parameters

In this subsection, we present an IEKF-based algorithm that guarantees errors to converge to a small interval under the assumption that $\overline{\zeta}$ is known, i.e., Algorithm 1.

In Algorithm 1, we take the upper bound of the residual terms $\overline{\zeta}$ as the input. We initialize the state covariance matrix of each independent EKF as $\mathbf{P}_{i,1} = p_1\mathbf{I}$, where $p_1 \in \mathbb{R}^+$. In each time step, we first generate a prediction $\hat{\boldsymbol{d}}_t$, then receive the desired data $\mathbf{d}_t$, and suffer the loss $\|\mathbf{e}_t\|^2 = \ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})$. We perform the parameter updates only if the loss is bigger than $4\overline{\zeta}^2$, i.e., $\|\mathbf{e}_t\|^2 > 4\overline{\zeta}^2$. If so, we calculate the Jacobian matrix $\mathbf{H}_{i,t}$, measurement noise level $r_{i,t}$, and the Kalman gain matrix $\mathbf{G}_{i,t}$ for each $i \in [(4n_s + n_d)]$ in lines 7-9 of Algorithm 1. We, then, update the weights and state covariance matrix of the weights belonging to node i in lines 10 and 11.

In the following lemma, we present several propositions that will be used to prove the theoretical guarantees of Algorithm 1.

Algorithm 1

Parameters: $\overline{\zeta} \in \mathbb{R}^+$.
Initialization: $\mathbf{P}_{i,1} = p_1\mathbf{I}$ for $i \in [(4n_s + n_d)]$, where $p_1 \in \mathbb{R}^+$.
for $t = 1$ **to** T **do**
  Generate $\hat{\boldsymbol{d}}_t$, observe $\mathbf{d}_t$, and suffer the loss $\|\mathbf{e}_t\|^2 = \ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})$.
  if $\|\mathbf{e}_t\|^2 > 4\overline{\zeta}^2$ **then**
    for $i = 1$ **to** $(4n_s + n_d)$ **do**
      Calculate the Jacobian matrix $\mathbf{H}_{i,t}$
      $r_{i,t} = 3\,\mathrm{Tr}(\mathbf{H}_{i,t}\mathbf{P}_{i,t}\mathbf{H}_{i,t}^T)/n_d$
      $\mathbf{G}_{i,t} = \mathbf{P}_{i,t}\mathbf{H}_{i,t}^T(\mathbf{H}_{i,t}\mathbf{P}_{i,t}\mathbf{H}_{i,t}^T + r_{i,t}\mathbf{I})^{-1}$
      $\hat{\boldsymbol{\theta}}_{i,t+1} = \hat{\boldsymbol{\theta}}_{i,t} + \mathbf{G}_{i,t}(\mathbf{d}_t - \hat{\boldsymbol{d}}_t)$
      $\mathbf{P}_{i,t+1} = (\mathbf{I} - \mathbf{G}_{i,t}\mathbf{H}_{i,t})\mathbf{P}_{i,t} + q_t\mathbf{I}$
    end for
  else
    $\hat{\boldsymbol{\theta}}_{i,t+1} = \hat{\boldsymbol{\theta}}_{i,t}$, for $i \in [(4n_s + n_d)]$
    $\mathbf{P}_{i,t+1} = \mathbf{P}_{i,t}$, for $i \in [(4n_s + n_d)]$
  end if
end for

Lemma 2.1. *For* $\{t : \|\boldsymbol{e}_t\|^2 > 4\overline{\zeta}^2\}$*, Algorithm 1 guarantees the following statements:*

1. *For each node* i*, the difference between the locally optimal weights and LSTM weights is governed with the following equation:*

$$\boldsymbol{\chi}_{i,t+1} = (\boldsymbol{I} - \boldsymbol{G}_{i,t}\boldsymbol{H}_{i,t})\boldsymbol{\chi}_{i,t} - \boldsymbol{G}_{i,t}\boldsymbol{\zeta}_{i,t}, \tag{2.22}$$

   which can also be written as

$$\boldsymbol{\chi}_{i,t+1} - \boldsymbol{\chi}_{i,t} = -\boldsymbol{G}_{i,t}\boldsymbol{H}_{i,t}\boldsymbol{\chi}_{i,t} - \boldsymbol{G}_{i,t}\boldsymbol{\zeta}_{i,t}. \tag{2.23}$$

2. *For each node* i*,* $\boldsymbol{P}_{i,t}^{-1}$ *and* $(\boldsymbol{P}_{i,t+1} - q_t\boldsymbol{I})^{-1}$ *exist and they are always positive definite as such*

$$(\boldsymbol{P}_{i,t+1} - q_t\boldsymbol{I})^{-1} = \boldsymbol{P}_{i,t}^{-1}(\boldsymbol{I} - \boldsymbol{G}_{i,t}\boldsymbol{H}_{i,t})^{-1} \tag{2.24}$$

$$= \boldsymbol{P}_{i,t}^{-1} + r_{i,t}^{-1}\boldsymbol{H}_{i,t}^T\boldsymbol{H}_{i,t} \geq \boldsymbol{0}. \tag{2.25}$$

3. *As a result of the previous two statements,*

$$(\boldsymbol{P}_{i,t+1} - q_t\boldsymbol{I})^{-1}\boldsymbol{\chi}_{i,t+1} = \boldsymbol{P}_{i,t}^{-1}\boldsymbol{\chi}_{i,t} - (\boldsymbol{P}_{i,t+1} - q_t\boldsymbol{I})^{-1}\boldsymbol{G}_{i,t}\boldsymbol{\zeta}_{i,t} \tag{2.26}$$

holds for each node $i \in [(4n_s + n_d)]$.

In the following theorem, we state the theoretical guarantees of Algorithm 1.

Theorem 2.2. *If* $\sum_{i=1}^{(4n_s+n_d)} Tr(\boldsymbol{P}_{i,t})$ *stays bounded during training, Algorithm 1 guarantees the following statements:*

1. *The LSTM weights stay bounded during training.*
2. *The loss sequence* $\{\|\boldsymbol{e}_t\|^2\}_{t\geq 1}$ *is guaranteed to converge to the interval* $[0, 4\overline{\zeta}^2]$.

Remark 2.1. *Due to the Kalman gain matrix formulation (line 9 in Algorithm 1),* $Tr((\boldsymbol{I} - \boldsymbol{G}_{i,t}\boldsymbol{H}_{i,t})\boldsymbol{P}_{i,t})$ *is always smaller than or equal to* $Tr(\boldsymbol{P}_{i,t})$ *for each node* i*, i.e.,* $Tr((\boldsymbol{I} - \boldsymbol{G}_{i,t}\boldsymbol{H}_{i,t})\boldsymbol{P}_{i,t}) \leq Tr(\boldsymbol{P}_{i,t})$ *for all* $i \in [(4n_s + n_d)]$*. Since* $\boldsymbol{P}_{i,t+1} = (\boldsymbol{I} - \boldsymbol{G}_{i,t}\boldsymbol{H}_{i,t})\boldsymbol{P}_{i,t} + q_t\boldsymbol{I}$*, and the artificial process noise level* q_t *is a user-dependent parameter, the condition in Theorem 2.2 can be satisfied by the user by selecting sufficiently small* q_t*.*

We note that due to $\mathbf{d}_t, \hat{\boldsymbol{d}}_t \in [-1, 1]^{n_d}$, $\|\mathbf{e}_t\|^2 \leq 4n_d$. Therefore, to ensure that the 2nd statement in Theorem 2.2 is not a trivial interval, we must show that $\overline{\zeta} \in [0, \sqrt{n_d}]$. In the following theorem we show that choosing small initial weights, i.e., $\hat{\boldsymbol{\theta}}_1 \approx \mathbf{0}$, guarantees $\overline{\zeta} \in [0, \sqrt{n_d}]$.

Theorem 2.3. *For any bounded data sequence* $\{\boldsymbol{d}_t\}_{t\geq 1}$ *with* $\boldsymbol{d}_t \in [-1, 1]^{n_d}$*, there exists a small positive number* ϵ *such that* $\|\hat{\boldsymbol{\theta}}_1\| \leq \epsilon$ *ensures that* $\overline{\zeta}$ *is in the interval* $[0, \sqrt{n_d}]$.

Remark 2.2. *We note that although Theorem 2.3 guarantees the existence of* ϵ*, which guarantees* $\overline{\zeta} \in [0, \sqrt{n_d}]$ *under the condition of* $\|\hat{\boldsymbol{\theta}}_1\| \leq \epsilon$*, it does not provide us a specific* ϵ *value. However, in the simulations, we observe that* $\hat{\boldsymbol{\theta}}_1 \sim \mathcal{N}(\boldsymbol{0}, 0.01\boldsymbol{I})$ *gives us both small error rates and fast convergence speed at the same time. Therefore, in the following we assume that* $\hat{\boldsymbol{\theta}}_1 \sim \mathcal{N}(\boldsymbol{0}, 0.01\boldsymbol{I})$ *is sufficiently small to ensure* $\overline{\zeta} \in [0, \sqrt{n_d}]$ *for practical applications.*

Note that by Theorem 2.3, we guarantee an interval for $\overline{\zeta}$. In the following section, we utilize this interval to extend our algorithm to a truly online form.

Algorithm 2

Parameters: $\overline{\zeta}_{\min} \in \mathbb{R}^+$.
Initialize $\overline{\boldsymbol{\zeta}} = [\sqrt{n_d}, 0.5\sqrt{n_d}, 0.25\sqrt{n_d}, \cdots, \overline{\zeta}_{\min}]^T$.
Set $N = \log(\sqrt{n_d}/\overline{\zeta}_{\min}) + 1$.
Initialize N independent Algorithm 1 instances with the entries of $\overline{\boldsymbol{\zeta}}$.
Let the indices of the instances be $j \in [N]$.
Initialize the weight of the instances as $w_{j,1} = 1/N$, for $j \in [N]$.
for $t = 1$ **to** T **do**
    Receive $\{\hat{\boldsymbol{d}}_{j,t}\}_{j\in[N]}$ vectors of the Algorithm 1 instances.
    Calculate the prediction vector as $\hat{\boldsymbol{d}}_t = \frac{\sum_{j=1}^{N} w_{j,t}\hat{\boldsymbol{d}}_{j,t}}{\sum_{k=1}^{N} w_{k,t}}$.
    Observe $\mathbf{d}_t$ and suffer the loss $\|\mathbf{e}_t\|^2 = \|\mathbf{d}_t - \hat{\boldsymbol{d}}_t\|^2$.
    Update all the Algorithm 1 instances by using $\mathbf{d}_t$ and their own $\hat{\boldsymbol{d}}_{j,t}$ vector.
    $w_{j,t+1} = w_{j,t}\exp(-\frac{1}{8n_d}\|\mathbf{d}_t - \hat{\boldsymbol{d}}_{j,t}\|^2)$, for $j \in [N]$.
end for

2.4.2 Truly Online Form

In this section, we extend Algorithm 1 to a truly online form, where we do not necessarily know $\overline{\zeta}$ a priori. To this end, we introduce Algorithm 2, where we run multiple instances of Algorithm 1 with carefully selected $\overline{\zeta}$ values, and mix their predictions with the exponential weighting algorithm to find the smallest $\overline{\zeta}$ efficiently in a truly online manner.

In Algorithm 2, we use the fact that the effective range of $\overline{\zeta}$ is $[0, \sqrt{n_d}]$. Here, we specify a small $\overline{\zeta}_{\min}$ and run multiple Algorithm 1 instances independently with $\overline{\zeta}$ values from $\sqrt{n_d}$ to $\overline{\zeta}_{\min}$ decreasing with powers of 2, i.e., $\overline{\boldsymbol{\zeta}} = [\sqrt{n_d}, 0.5\sqrt{n_d}, 0.25\sqrt{n_d}, \cdots, \overline{\zeta}_{\min}]^T$, where we have a total of $N = \log(\sqrt{n_d}/\overline{\zeta}_{\min}) + 1$ number of independent Algorithm 1 instances. In each round, we receive the prediction of the instances, i.e., $\{\hat{\boldsymbol{d}}_{j,t}\}_{j\in[N]}$, and take the weighted average of $\{\hat{\boldsymbol{d}}_{j,t}\}_{j\in[N]}$ to determine $\hat{\boldsymbol{d}}_t$, i.e., $\hat{\boldsymbol{d}}_t = (\sum_{j=1}^{N} w_{j,t}\hat{\boldsymbol{d}}_{j,t})/\sum_{k=1}^{N} w_{k,t}$. In its following, we observe the target value $\mathbf{d}_t$, and suffer the loss $\|\mathbf{e}_t\|^2 = \|\mathbf{d}_t - \hat{\boldsymbol{d}}_t\|^2$. We, then, update the Algorithm 1 instances with their own predictions $\hat{\boldsymbol{d}}_{j,t}$, and update the weights as $w_{j,t+1} = w_{j,t}\exp(-\frac{1}{8n_d}\|\mathbf{d}_t - \hat{\boldsymbol{d}}_{j,t}\|^2)$ for $j \in [N]$.

In the following theorem, we derive the theoretical guarantees of Algorithm 2.

Theorem 2.4. *Let us use* $\overline{\zeta}_{best}$ *to denote the smallest possible value for* $\overline{\zeta}$ *that guarantees the tightest possible interval for* $\{\|\boldsymbol{e}_t\|^2\}_{t\geq 1}$*. Assuming* $\overline{\zeta}_{best} \in [\overline{\zeta}_{min}, \sqrt{n_d}]$*, Algorithm 2 guarantees that* $\{\|\boldsymbol{e}_t\|^2\}_{t\geq 1}$ *converges to the interval* $[0, 16\overline{\zeta}^{-2}_{best}]$ *in a truly online manner.*

Remark 2.3. *We note that the results so far do not assume any specific distribution for the data nor any topology for the error surface. Therefore, our performance guarantee holds globally regardless of the distribution of the data, given that the target sequence is bounded (required in Theorem 2.3). The difference between different noise levels or two local optima demonstrates itself in* $\overline{\zeta}$*, or in the asymptotic error interval in Theorem 2.2, which is guaranteed to be learned in a data-dependent manner in Theorem 2.4.*

Now that we have proved the performance guarantee of our algorithm, in the following remark, we present the computational complexity of Algorithm 2, and compare the presented complexity with the computational requirement of EKF.

Remark 2.4. *We maintain that* $\overline{\zeta}_{min} = 0.01$ *is practically sufficient for* $\overline{\zeta}_{best} \in [\overline{\zeta}_{min}, \sqrt{n_d}]$ *(or to guarantee a tight interval for error to converge). In this case, the number of independent Algorithm 1 instances in Algorithm 2, i.e.,* N*, becomes* $\log(100\sqrt{n_d})$*, equivalently,* $(7 + 0.5 \log n_d)$*. Hence, the computational complexity of Algorithm 2 becomes* $O\big((7 + 0.5 \log n_d) n_\theta^2/(4n_s + n_d)\big)$ *in the worst case, where we assume that all the Algorithm 1 instances perform the IEKF updates (lines 7-11 in Algorithm 1) in every time step.*

As noted earlier, the computational requirement of EKF is $O(n_\theta^2)$*. Therefore, the asymptotic efficiency gain of Algorithm 2 over EKF is* $(4n_s + n_d)/(7 + 0.5 \log n_d)$*. Since the number of nodes in practical LSTMs is usually more than* 50*, in the worst case, Algorithm 2 reduces the computational requirement of the EKF-based training by* $5 - 7$ *times [19, 26, 34, 35]. However, we note that in practice, Algorithm 1 with a high* $\overline{\zeta}$ (≥ 0.2) *performs a small number of updates, usually around* $20 - 30$ *updates in* 1000 *time steps. Therefore, the ratio between the run-times of EKF and Algorithm 2 is generally considerably higher than the worst-case ratio derived above. In fact, in the following section, we demonstrate*

that Algorithm 2 provides a $10-15$ *times reduction in the training time compared to EKF while yielding very similar performance.*

2.5 Simulations

In this section, we illustrate the performance improvements of our algorithm, i.e., Algorithm 2, with respect to the state-of-the-art algorithms. To this end, we compare Algorithm 2 (abbreviated as Alg2) with five widely used optimization algorithms, i.e., SGD, Adam, RMSprop, DEKF, and EKF, on six real-world and three synthetic datasets. We linearly map the target vectors between $[-1, 1]$ to satisfy the condition in Theorem 2.4.

In all simulations, we randomly draw the initial weights from a Gaussian distribution with zero mean and standard deviation of 0.1, i.e., $\hat{\boldsymbol{\theta}}_1 \sim \mathcal{N}(\mathbf{0}, 0.01\mathbf{I})$. We set the initial values of all state variables to 0 and $\overline{\zeta}_{\text{min}}$ to 0.01. To provide a robust comparison, we report results with 90% confidence intervals. To obtain the confidence intervals, we repeat the experiments twenty times with randomly chosen initial parameters and calculate the 5th and 95th percentiles of the errors in each time step. Then, we report the average of the calculated intervals between the calculated percentiles.

We search the hyperparameters of the learning algorithms (except EKF) over a dense grid and report the results using the best hyperparameters in that setting, i.e., the hyperparameters minimizing the mean squared error. Since the grid search for EKF takes a very long time, in EKF, we use the same hyperparameters used in DEKF. To provide a consistent scale for the results, we normalize the squared errors with the variance of the target stream. To evaluate the generalization performance of the algorithms, we report the confidence intervals of both one-step normalized squared errors (shortly NSE) and k-step normalized squared errors (shortly kNSE), where kNSE is the normalized squared error of the k-step ahead forecast. In the experiments, we use $k = 50$. We share the source code of our experiments on GitHub at `https:`

	kin8nm ($n_h = 16$, $n_x = 9$)			kin32fm ($n_h = 12$, $n_x = 33$)			elevators ($n_h = 12$, $n_x = 19$)		
	NSE	kNSE	Run-time	NSE	kNSE	Run-time	NSE	kNSE	Run-time
SGD	0.695 ± 0.25	0.658 ± 0.26	0.58	0.483 ± 0.28	0.486 ± 0.31	1.15	0.488 ± 0.22	0.494 ± 0.24	0.62
RMSprop	0.430 ± 0.17	0.427 ± 0.17	0.59	0.228 ± 0.12	0.232 ± 0.13	1.03	0.326 ± 0.16	0.336 ± 0.16	0.55
Adam	0.423 ± 0.20	0.426 ± 0.20	0.58	0.243 ± 0.16	0.249 ± 0.17	1.06	0.393 ± 0.25	0.396 ± 0.25	0.57
DEKF	0.419 ± 0.22	0.417 ± 0.22	1.68	0.198 ± 0.12	0.209 ± 0.13	2.51	0.247 ± 0.12	0.249 ± 0.12	1.74
EKF	$\mathbf{0.331 \pm 0.18}$	$\mathbf{0.324 \pm 0.18}$	62.17	0.171 ± 0.10	0.181 ± 0.10	112.11	0.216 ± 0.10	0.215 ± 0.10	53.74
Alg2	0.345 ± 0.20	0.335 ± 0.20	4.48	$\mathbf{0.148 \pm 0.06}$	$\mathbf{0.164 \pm 0.10}$	9.04	$\mathbf{0.213 \pm 0.09}$	$\mathbf{0.215 \pm 0.09}$	4.45

(a)

	puma8nm ($n_h = 16$, $n_x = 9$)			puma8nh ($n_h = 16$, $n_x = 9$)			puma32fm ($n_h = 12$, $n_x = 33$)		
	NSE	kNSE	Run-time	NSE	kNSE	Run-time	NSE	kNSE	Run-time
SGD	0.459 ± 0.22	0.448 ± 0.22	0.58	0.708 ± 0.22	0.725 ± 0.23	0.61	0.383 ± 0.22	0.395 ± 0.25	1.11
RMSprop	0.252 ± 0.13	0.254 ± 0.13	0.59	0.517 ± 0.14	0.524 ± 0.13	0.61	0.185 ± 0.10	0.187 ± 0.10	1.03
Adam	0.291 ± 0.21	0.293 ± 0.21	0.58	0.523 ± 0.23	0.526 ± 0.23	0.65	0.172 ± 0.11	0.181 ± 0.11	1.06
DEKF	0.216 ± 0.11	0.211 ± 0.11	1.72	0.509 ± 0.23	0.502 ± 0.23	1.76	0.148 ± 0.08	0.156 ± 0.09	2.51
EKF	$\mathbf{0.153 \pm 0.08}$	$\mathbf{0.152 \pm 0.08}$	59.38	0.480 ± 0.24	0.474 ± 0.24	68.23	0.136 ± 0.08	0.144 ± 0.08	113.14
Alg2	0.179 ± 0.11	0.171 ± 0.11	4.31	$\mathbf{0.464 \pm 0.21}$	$\mathbf{0.457 \pm 0.21}$	4.60	$\mathbf{0.122 \pm 0.05}$	$\mathbf{0.129 \pm 0.08}$	9.21

(b)

Table 2.1: Empirical 90% confidence intervals for the one-step and k-step normalized squared errors, i.e., NSE and kNSE, with the run-times (in seconds) of the compared algorithms. The bold font shows the best NSE and kNSE results (in terms of the median value) for each dataset. The simulations are performed on a computer with i7-7500U processor, 2.7-GHz CPU, and 8-GB RAM.

`//github.com/nurimertvural/EfficientEffectiveLSTM`.

2.5.1 Real Data Benchmark

In the first part, we evaluate the performance of our algorithm with six real-world datasets. Since the EKF simulations require very high running times, we consider only the first 2500 input/output pairs in each dataset, i.e., $T = 2500$.

2.5.1.1 Kinematic Family of Datasets

In the first set of experiments, we use the kinematic family of datasets [36], which are obtained through the simulations of eight-link all revolute robotic arms with different forward-dynamics. The aim is to estimate the distance of the end-effector from a target by using the input predictors, i.e., $n_d = 1$. Here, we consider two datasets from the kinematic family: kin8nm, which is generated with nonlinear

dynamics and moderate noise, and kin32fm, which is generated with fairly linear dynamics and moderate noise.

For the kin8nm dataset, we use 8-dimensional input vectors of the dataset with an additional bias dimension, i.e., $n_x = 9$, and 16-dimensional state vectors, i.e., $n_h = 16$. In Adam, RMSprop, and SGD, we use the learning rates of 0.003, 0.004, and 0.3 In EKF and DEKF, we choose the initial state covariance matrix as $100\mathbf{I}$ and anneal the measurement and process noise levels from 30 to 3, and 10^{-2} to 10^{-6}, respectively. In Alg2, we initialize the state covariance matrices as $10\mathbf{I}$ and anneal the process noise level from 10^{-4} to 10^{-8}.

In the leftmost column of Table 2.1a, we present the experiment results for the kin8nm dataset. Here, we observe that SGD provides the worst NSE and kNSE performance, followed by Adam, RMSprop, and DEKF. Moreover, Alg2 and EKF outperform the other algorithms in NSE and kNSE while providing very similar error intervals. However, Alg2 provides that equivalent performance in approximately 15 times smaller run-time.

For the kin32fm dataset, we use 32-dimensional input vectors of the dataset with an additional bias dimension, i.e., $n_x = 33$, and 12-dimensional state vectors, i.e., $n_h = 12$. In Adam, RMSprop, and SGD, we use the learning rates of 0.001, 0.002 and 0.5. In EKF and DEKF, we use the same hyperparameters used in the previous experiment. In Alg2, we initialize the state covariance matrices as $0.5\mathbf{I}$ and choose the process noise as $q_t = 10^{-8}$ for all $t \in [T]$.

In the middle column of Table 2.1a, we present the experiment results for the kin32fm dataset. Here, we see that EKF and Alg2 provide lower NSE and kNSE values compared to DEKF, Adam, RMSprop and SGD. Differing from the previous experiment, Alg2 achieves slightly better NSE and kNSE performance than EKF. Similar to the previous experiment, Alg2 provides very close performance to EKF in 12 times smaller run-time.

2.5.1.2 Elevators Dataset

In the second part, we consider the elevators dataset, which is obtained from the controlling procedure of an F16 aircraft [37]. Here, the aim is to predict the scalar variable that expresses the actions of the aircraft, i.e., $n_d = 1$. For this dataset, we use 18-dimensional input vectors of the dataset with an additional bias dimension, i.e., $n_x = 19$, and 12-dimensional state vectors, i.e., $n_h = 12$. In Adam, RMSprop, and SGD, we choose the learning rates as 0.005, 0.005, and 0.35. In EKF and DEKF, we initialize the state covariance matrix as $100\mathbf{I}$ and anneal the measurement and process noise levels from 50 to 3, and 10^{-2} to 10^{-6}. In Alg2, we set the initial state covariance matrices to $10\mathbf{I}$ and anneal the process noise from 10^{-4} to 10^{-8}.

In the rightmost column of Table 2.1a, we present the experiment results for the elevators dataset. Here, we see that as in the previous two experiments, Adam and RMSprop obtain smaller NSE and kNSE values compared to SGD. Furthermore, EKF-based methods, i.e., DEKF, EKF, and Alg2, improve the accuracy of the first-order methods considerably. Here also, EKF and Alg2 achieve the smallest errors while Alg2 reduces the training time of EKF 13 times without compromising the error performance.

2.5.1.3 Pumadyn Family of Datasets

In the last set of experiments, we use the pumadyn family of datasets obtained through realistic simulations of the dynamics of a Puma 560 robot arm [36]. The aim is to estimate the angular acceleration of the arm, i.e., $n_d = 1$, by using the angular position and angular velocity of the links. In this part, we consider three datasets from the pumadyn family: puma8nm, which is generated with nonlinear dynamics and moderate noise, puma8nh, which is generated with nonlinear dynamics and high noise, and puma32fm, which is generated with fairly linear dynamics and moderate noise

For the puma8nm dataset, we use 8-dimensional input vectors of the dataset

with an additional bias dimension, i.e., $n_x = 9$, and 16-dimensional state vectors, i.e., $n_h = 16$. In Adam, RMSprop, and SGD, we use the learning rates of 0.009, 0.01 and 0.4. In EKF and DEKF, we initialize the state covariance matrix as $100\mathbf{I}$ and anneal the measurement and process noise levels from 30 to 3, and 10^{-2} to 10^{-6}, respectively. In Alg2, we set the initial state covariance matrices to $10\mathbf{I}$ and anneal the process noise level from 10^{-4} to 10^{-8}.

In the puma8nh experiment, we left the hyperparameters of the puma8nm experiment unchanged with the following exceptions: Here, we choose the learning rates of Adam, RMSprop and SGD as 0.006, 0.006, and 0.25. In EKF and DEKF, we anneal the measurement and process noise levels from 20 to 3, and 10^{-3} to 10^{-6}, respectively.

For the puma32fm dataset, we use 32-dimensional input vectors of the dataset with an additional bias dimension, i.e., $n_x = 33$, and 12-dimensional state vectors, i.e., $n_h = 12$. In Adam, RMSprop, and SGD, we use the learning rates of 0.003, 0.004, and 0.35. In EKF and DEKF, we choose the initial state covariance matrix as $100\mathbf{I}$ and anneal the measurement noise and process noise levels from 20 to 3, and 10^{-3} to 10^{-6}, respectively. In Alg2, we initialize the state covariance matrices as $0.5\mathbf{I}$ and choose the process noise as $q_t = 10^{-8}$ for all $t \in [T]$.

In Table 2.1b, we present the experiment results for the puma8nm, puma8nh, and puma32fm datasets. Here, we observe that the error values in the puma8nh experiment are relatively higher due to the nonlinear dynamics and high noise of the puma8nh dataset, whereas the error values in the puma32fm experiment are comparably smaller due to the linearity and moderate noise of the puma32fm dataset. As in the previous parts, Alg2 and EKF achieve lower NSE and kNSE values than the other algorithms in all experiments. However, Alg2 provides the comparable error values in 12 to 15 times smaller running times than EKF.

2.5.2 Binary Addition

In this part, we compare the performance of the algorithms on a synthesized dataset that requires learning long-term dependencies. We show that our algorithm learns the long-term dependencies comparably well with EKF, which explains its success in the previous experiments.

To compare the algorithms, we construct a synthesized experiment in which we can control the length of temporal dependence. To this end, we train the network to learn adding n number of binary sequences, where the carry bit is the temporal dependency that the models need to learn. We note that the number of added sequences, i.e., n, controls how long the carry bit is propagated on average, hence, the average length of the temporal dependence. We learn binary addition with a purely online approach, i.e., there is only one input stream, and learning continues even when the network makes a mistake.

In the experiments, we consider $n \in \{3, 4, 5\}$. For each n, we repeat the experiments with five different input streams, where all input bits are generated randomly by using independent Bernoulli random trials with success probability 0.5. We assume that the network decides 1 when its output is positive, and 0 in vice versa. For the performance comparison, we count the number of symbols needed to attain error-free predictions for 500 subsequent symbols, i.e., sustainable prediction. We note that as the algorithms demonstrate a faster rate of convergence, the number of steps required to obtain sustainable prediction is expected to be smaller.

In this part, we use 12-dimensional state vectors, i.e., $n_h = 12$. As the input vectors, we use the incoming n bits with an additional bias, i.e., $n_x = n + 1$. In EKF and DEKF, we initialize the state covariance matrix as $100\mathbf{I}$, choose the measurement noise level as $r_t = 3$ for all $t \in [T]$, and anneal the process noise level from 10^{-3} to 10^{-6}. In Alg2, we set all initial state covariance matrices to $10\mathbf{I}$ and choose the process noise as $q_t = 10^{-7}$ for all $t \in [T]$. In RMSprop, we use the learning rate of 0.02. Due to space constraints, we compare the EKF-based methods only with RMSprop, which is observed to demonstrate a faster

rate of convergence compared to Adam and SGD during the experiments. In the following, we say that an algorithm is failed if it is unable to obtain sustainable prediction after 10^5 timesteps.

In the first experiment, we consider adding three sequences, i.e., $n = 3$. The experiment results are presented in Table 2.2a. Here, we see that Alg2 and EKF achieve 500 subsequent error-free predictions with considerably fewer data compared to RMSprop and DEKF. On the other hand, RMSprop and DEKF achieve sustainable prediction with shorter running times in general due to their relatively small computational complexity.

In the second and third experiments, we consider adding four and five sequences, i.e., $n = 4$ and $n = 5$, respectively. The results are presented in Table 2.2b and Table 2.2c. Here, we see that in all experiments, RMSprop and DEKF are unable to obtain subsequent 500 error-free predictions even after 10^5 timesteps. On the other hand, EKF and Alg2 consistently achieve the sustainable prediction parallel to their success in the previous experiments. Moreover, we see that EKF requires slightly fewer data to complete the task in general. However, Alg2 requires significantly shorter running times due to its relative efficiency in comparison to EKF.

RMSprop		DEKF		EKF		Alg2 (This work)	
Timestep	Run-time	Timestep	Run-time	Timestep	Run-time	Timestep	Run-time
43778	15.61	29995	11.24	**5995**	37.93	6906	**11.21**
44472	15.74	11464	**5.21**	**2651**	17.16	6107	10.26
22912	**8.47**	25382	9.72	**5411**	33.74	7065	11.79
27264	7.46	12242	**5.25**	**4205**	26.5	8245	14.48
21073	**6.69**	19491	7.67	**3244**	20.65	5510	9.26

(a) Adding three binary sequences ($n_h = 12$, $n_x = 4$).

RMSprop		DEKF		EKF		Alg2 (This work)	
Timestep	Run-time	Timestep	Run-time	Timestep	Run-time	Timestep	Run-time
Failed		Failed		**28978**	197.03	31353	**52.05**
Failed		Failed		**30578**	207.38	36606	**61.41**
Failed		Failed		59835	407.63	**53204**	**88.03**
Failed		Failed		50549	343.84	**18083**	**30.99**
Failed		Failed		**15871**	107.75	18165	**32.96**

(b) Adding four binary sequences ($n_h = 12$, $n_x = 5$).

RMSprop		DEKF		EKF		Alg2 (This work)	
Timestep	Run-time	Timestep	Run-time	Timestep	Run-Time	Timestep	Run-Time
Failed		Failed		**27977**	202.27	33912	**52.34**
Failed		Failed		**61709**	477.21	78583	**128.53**
Failed		Failed		**54763**	412.44	54777	**85.38**
Failed		Failed		**32647**	245.28	93080	**174.21**
Failed		Failed		**44492**	334.85	57919	**86.36**

(c) Adding five binary sequences ($n_h = 12$, $n_x = 6$).

Table 2.2: Timesteps and the run-times (in seconds) required to achieve 500 subsequent error-free predictions, i.e., sustainable prediction. The experiments are repeated with five different input streams and the results are presented in order. The bold font shows the best result (in terms of both timestep and run-time) for each input stream. The simulations are performed on a computer with i7-7500U processor, 2.7-GHz CPU, and 8-GB RAM.

Chapter 3

Achieving Online Regression Performance of LSTMs with Simple RNNs

In this chapter, we achieve the online regression performance of LSTMs with SRNNs efficiently. To this end, we introduce a first-order training algorithm with a linear time complexity in the number of parameters. We show that when SRNNs are trained with our algorithm, they provide very similar regression performance with the LSTMs in two to three times shorter training time. We provide strong theoretical analysis to support our experimental results by providing regret bounds on the convergence rate of our algorithm. Through simulations, we verify our theoretical work and demonstrate significant performance improvements of our algorithm with respect to LSTMs and the state-of-the-art training methods.

3.1 Related Work

LSTMs have recently seen remarkable empirical success in a broad range of sequential learning tasks, including online regression [16, 34]. To achieve such performance, however, LSTMs require a comparably large number of parameters, which make them more demanding in terms of computational and data requirements compared to simpler models, such as SRNNs [17]. There exists a wide range of work for obtaining the performance of the LSTMs with simpler models [38,39]. One of the most common approaches to that end is allowing SRNNs to have direct connections from the distant past to enable them to handle long-term dependencies. One model in this line is the NARX-RNN, which introduces an additional set of recurrent connections with time lags of $2, 3, \cdots, k$ time steps [39]. However, NARX-RNNs are extremely inefficient, as both parameter counts and computation counts grow by the same factor k [38]. A more efficient model with a similar structure is Clockwork RNNs, i.e., CW-RNNs [20]. The main idea of CW-RNNs is splitting the weights and hidden units of SRNNs into partitions, each with a distinct period, to maintain an additive error flow. However, CW-RNNs require hidden units to be partitioned *a priori*, which in practice is difficult to do in any meaningful way, especially in online regression where we usually have a very small amount of *a priori* information on the data sequence.

Another approach to improve the performance of SRNNs is to use higher-order properties of the error surface during training. For example, RMSprop [5] utilizes approximate Hessian-based preconditioning, which significantly improves the error performance of SGD [30]. Due to its efficiency and performance improvement, RMSprop and its follow-on methods, such as Adam [6], are being frequently used in the current deep learning applications. However, these techniques are not sufficient to close the gap between SRNNs and LSTMs in general. As a second-order technique, Kalman filters have shown advantages in bridging long time lags as well [19]. However, this approach requires a quadratic time complexity in the parameter size, which is computationally unfeasible for larger networks. Another attempt to bridge the performance of LSTMs and SRNNs is Hessian Free optimization, an adapted second-order training method that has been demonstrated

to work well with SRNNs [22]. However, the Hessian Free algorithm requires a large size of batches to approximate the Hessian matrix, which makes it impractical for the online settings.

In this chapter, we obtain the online regression performance of LSTMs with SRNNs by using a first-order training algorithm. Our study differs from the model-based studies since we do not use any complex model rather than the standard Elman Network. Moreover, our algorithm is fundamentally different from the Kalman Filter and Hessian Free algorithms since it is a truly online first-order training algorithm. In the experiments, we show that our algorithm provides very similar performance with LSTMs trained with the state-of-the-art first-order methods, in two to three times shorter training time. Therefore, we introduce a highly practical algorithm, providing comparable performance with the state-of-the-art methods in a highly efficient manner.

3.2 Model and Problem Definition

We consider the following online regression setting: We observe input variables $\{\mathbf{x}_t\}_{t\geq 1}$ at each time step in a sequential manner. After receiving the vector $\mathbf{x}_t$, we produce an estimate $\hat{d}_t$ for the next unobserved target value d_t.[1] After declaring our estimate, we observe d_t and suffer the loss $\ell(d_t, \hat{d}_t)$. Our aim is to optimize the network with respect to the loss function $\ell(\cdot, \cdot)$ in an online manner. In this part, we particularly work with the squared loss, i.e., $\ell(\hat{d}_t, d_t) = 0.5(d_t - \hat{d}_t)^2$, where we scale the squared loss with 0.5 for mathematical convenience when computing the derivatives. However, our work can be directly extended to the cross-entropy loss, which is detailed in Appendix B.1.

We study online regression with continually running RNNs. For this task, we

[1]For mathematical convenience, we assume $\mathbf{x}_t \in [-1, 1]^{n_x}$ and $d_t \in [-\sqrt{n_h}, \sqrt{n_h}]$, where $n_h \in \mathbb{N}$ is the hidden-size of the network. However, our derivations can be extended to any bounded input and output sequences after shifting and scaling the magnitude.

use the SRNN model (or the standard Elman network model), i.e.,

$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t) \tag{3.1}$$

$$\hat{d}_t = \boldsymbol{\vartheta}^T \mathbf{h}_t. \tag{3.2}$$

Here, we have $\mathbf{W} \in \mathbb{R}^{n_h \times n_h}$ and $\mathbf{U} \in \mathbb{R}^{n_h \times n_x}$ as the hidden layer weight matrices. We have $\boldsymbol{\vartheta} \in \mathbb{R}^{n_h}$, with $\|\boldsymbol{\vartheta}\| \leq 1$, as the output layer weights.[2] Moreover, $\mathbf{h}_t \in [-1,1]^{n_h}$ is the hidden state vector, $\mathbf{x}_t \in [-1,1]^{n_x}$ is the input vector, $\hat{d}_t \in [-\sqrt{n_h}, \sqrt{n_h}]$ is our estimation, and tanh applies to vectors point-wise. We note that although we do not explicitly write the bias terms, they can be included in (3.1)-(3.2) by augmenting the input vectors with a constant dimension.

3.3 Algorithm Development

In this section, we introduce the main contribution of our paper, i.e., an online first-order algorithm that achieves the online regression performance of LSTMs with SRNNs. We develop our algorithm in three subsections. In the first subsection, we describe our approach and provide definitions for the following analysis. In the second subsection, we present the auxiliary results that will be used in the development of the main algorithm. In the last subsection, we introduce our algorithm and provide a regret bound on its convergence rate.

3.3.1 Sequential Learning Approach

We investigate the SRNN-based regression problem in the sequential learning framework, i.e., we make no statistical assumptions on the data in order to model chaotic, non-stationary or even adversarial environments. Hence, we model our problem as a game between a learner and an adversary, where the learner is tasked with predicting the weight matrices from some convex sets in each round t.

[2]Note that for the convergence of the training, the output-layer weights, i.e., $\boldsymbol{\vartheta}$, should stay bounded. To this end, our algorithm performs projection onto a convex set to keep $\boldsymbol{\vartheta}$ bounded. As the convex set, we use $\{\boldsymbol{\vartheta} : \|\boldsymbol{\vartheta}\| \leq 1\}$ for mathematical convenience in our proofs. However, our derivations can be extended to any bounded set of $\boldsymbol{\vartheta}$ by shifting and scaling the magnitude.

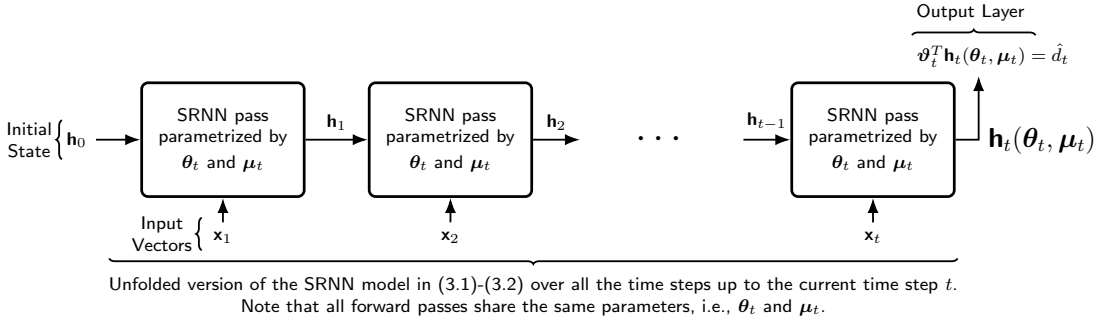

Figure 3.1: In this figure, we visually describe $\mathbf{h}_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$. $\mathbf{h}_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$ is defined as the hidden state vector obtained by running the model in (3.1)-(3.2) with $\boldsymbol{\theta}_t$ and $\boldsymbol{\mu}_t$ from the initial time step up to the current time step t. In the figure, the SRNN sequence is initialized with a predetermined initial state $\mathbf{h}_0$, which is independent of the network weights. Then, the same SRNN forward pass (given in (3.1)) is repeatedly applied to the input sequence $\{\mathbf{x}_t\}_{t\geq 1}$, where all the iterations are parametrized by $\boldsymbol{\theta}_t$ and $\boldsymbol{\mu}_t$. The resulting hidden vector after t iterations is defined to be $\mathbf{h}_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$. Here, we note that the dependence of $\mathbf{h}_t(\cdot, \cdot)$ on t is due to the increased length of the recursion at each time step.

To formulate the game, we first introduce three new notations: We use $\mathbf{W}_t$, $\mathbf{U}_t$ and $\boldsymbol{\vartheta}_t$ to denote the weights learned by the first $t-1$ data/input pairs. For mathematical convenience, we work with the vectorized form of the weight matrices, i.e., $\boldsymbol{\theta}_t = \text{vec}(\mathbf{W}_t)$ and $\boldsymbol{\mu}_t = \text{vec}(\mathbf{U}_t)$. Moreover, we use $\mathbf{h}_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$ to denote the hidden state vector obtained by running the SRNN model in (3.1)-(3.2) with $\boldsymbol{\theta}_t$ and $\boldsymbol{\mu}_t$ from the initial time step up to the current time step t (for the detailed description of $\mathbf{h}_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$ see Fig. 3.1).

Then, we construct the game as follows: At each round t, the learner declares his prediction $\boldsymbol{\theta}_t$ and $\boldsymbol{\mu}_t$; concurrently, the adversary chooses a target value $d_t \in [-\sqrt{n_h}, \sqrt{n_h}]$, an input $\mathbf{x}_t \in [-1, 1]^{n_x}$, and a weight vector $\boldsymbol{\vartheta}_t$, $\|\boldsymbol{\vartheta}_t\| \leq 1$; then, the learner observes the loss function

$$\ell_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t) \coloneqq 0.5\big(d_t - \underbrace{\boldsymbol{\vartheta}_t^T \mathbf{h}_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}_{\hat{d}_t}\big)^2 \tag{3.3}$$

and suffers the loss $\ell_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$. This procedure of play is repeated across T rounds, where T is the total number of input instances. We note that we constructed our setting for adversarial $\boldsymbol{\vartheta}_t$ selections for mathematical convenience in our proofs. However, since the selected $\ell_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$ is convex with respect to $\boldsymbol{\vartheta}_t$, we will use the online gradient descent algorithm [40] to learn the optimal output layer weights

(simultaneously with the hidden layer weights) during the training.

Since we assume no statistical assumptions on the input/output sequences, we analyze our performance with the notion of regret. However, the standard regret definition for the convex problems is intractable in the non-convex settings due to the NP-hardness of the non-convex global optimization [4]. Therefore, we use the notion of local regret recently introduced by Hazan et al. [4], which quantifies the objective of predicting points with a small gradient on average.

To formulate the local regret for our setting, we first define the projected partial derivatives of $\ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})$ with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ as follows:

$$\frac{\partial_{\mathcal{K}_\theta} \ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})}{\partial \boldsymbol{\theta}} \coloneqq \frac{1}{\eta}\Big(\boldsymbol{\theta} - \Pi_{\mathcal{K}_\theta}\Big[\boldsymbol{\theta} - \eta \frac{\partial \ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})}{\partial \boldsymbol{\theta}}\Big]\Big) \tag{3.4}$$

$$\frac{\partial_{\mathcal{K}_\mu} \ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})}{\partial \boldsymbol{\mu}} \coloneqq \frac{1}{\eta}\Big(\boldsymbol{\mu} - \Pi_{\mathcal{K}_\mu}\Big[\boldsymbol{\mu} - \eta \frac{\partial \ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})}{\partial \boldsymbol{\mu}}\Big]\Big), \tag{3.5}$$

where $\partial_{\mathcal{K}}$ denotes the projected partial derivative operator defined with some convex set $\mathcal{K}$ and some learning rate η. The operators $\Pi_{\mathcal{K}_\theta}[\cdot]$ and $\Pi_{\mathcal{K}_\mu}[\cdot]$ denote the orthogonal projections onto $\mathcal{K}_\theta$ and $\mathcal{K}_\mu$.

We define the time-smoothed loss at time t, parametrized by some window-size $w \in [T]$, as

$$L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu}) \coloneqq \frac{1}{w} \sum_{i=0}^{w-1} \ell_{t-i}(\boldsymbol{\theta}, \boldsymbol{\mu}). \tag{3.6}$$

Then, we define the local regret as

$$R_w(T) \coloneqq \sum_{t=1}^{T} \Big(\Big\| \frac{\partial_{\mathcal{K}_\theta} L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}{\partial \boldsymbol{\theta}} \Big\|^2 + \Big\| \frac{\partial_{\mathcal{K}_\mu} L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}{\partial \boldsymbol{\mu}} \Big\|^2 \Big). \tag{3.7}$$

Our aim is to derive a sublinear upper bound for $R_w(T)$ in order to ensure the convergence of our algorithm to the locally optimum weights. However, before the derivations, we first present some auxiliary results, i.e., the Lipschitz and smoothness properties of $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$, which will be used in the convergence proof of our algorithm.

3.3.2 Lipschitz and Smoothness Properties

In this section, we derive the Lipschitz and smoothness properties of the time-smoothed loss function $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$.

We note that $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ is defined as the average of the most recent w instant loss functions (see (3.6)), where the loss function $\ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})$ recursively depends on $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ due to $\mathbf{h}_t(\boldsymbol{\theta}, \boldsymbol{\mu})$ (see (3.3) and Fig. 3.1). We emphasize that since we are interested in online learning, this recursion can be infinitely long, which might cause $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ to have unboundedly large derivatives. On the other hand, online algorithms naturally require loss functions with bounded gradients to guarantee convergence.

Therefore, in this part, we analyze the recursive dependency of $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ on $\boldsymbol{\mu}$ and $\boldsymbol{\theta}$. We derive sufficient conditions for its derivatives to be bounded and find the explicit formulations of the smoothness constants of $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ in terms of the model parameters. To this end, we first derive the Lipschitz properties of $\mathbf{h}_t(\boldsymbol{\theta}, \boldsymbol{\mu})$, and observe the effect of infinitely long recursion on the derivatives of $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$.

Lemma 3.1. *Let $\boldsymbol{W}$, $\boldsymbol{W}'$, $\boldsymbol{U}$, $\boldsymbol{U}'$ satisfy $\|\boldsymbol{W}\|, \|\boldsymbol{W}'\| \leq \lambda$, and $\|\boldsymbol{U}\|, \|\boldsymbol{U}'\| \leq \lambda$. Let $\boldsymbol{h}_t(\boldsymbol{\theta}, \boldsymbol{\mu})$ and $\boldsymbol{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu}')$ be the state vectors obtained at time t by running the model in (3.1)-(3.2) with the matrices $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{W}'$, $\boldsymbol{U}'$ on common input sequence $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_t\}$, respectively. If $\boldsymbol{h}_0(\boldsymbol{\theta}, \boldsymbol{\mu}) = \boldsymbol{h}_0(\boldsymbol{\theta}', \boldsymbol{\mu}')$, then*

$$\|\boldsymbol{h}_t(\boldsymbol{\theta}, \boldsymbol{\mu}) - \boldsymbol{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu}')\| \leq \sum_{i=0}^{t-1} \lambda^i \big(\sqrt{n_h}\|\boldsymbol{\theta} - \boldsymbol{\theta}'\| + \sqrt{n_x}\|\boldsymbol{\mu} - \boldsymbol{\mu}'\|\big). \tag{3.8}$$

Remark 3.1. *We note that, by (3.8), to ensure $\boldsymbol{h}_t(\boldsymbol{\theta}, \boldsymbol{\mu})$ has a bounded gradient with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ in an infinite time horizon, λ should be in $[0, 1)$, i.e., $\lambda \in [0, 1)$. In this case, the right hand side of (3.8) becomes bounded, i.e.,*

$$\|\boldsymbol{h}_t(\boldsymbol{\theta}, \boldsymbol{\mu}) - \boldsymbol{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu}')\| \leq \frac{\sqrt{n_h}}{1-\lambda}\|\boldsymbol{\theta} - \boldsymbol{\theta}'\| + \frac{\sqrt{n_x}}{1-\lambda}\|\boldsymbol{\mu} - \boldsymbol{\mu}'\| \tag{3.9}$$

for any $t \in [T]$.

We recall that $L_{t_w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ is dependent on $\boldsymbol{h}_t(\boldsymbol{\theta}, \boldsymbol{\mu})$ due to (3.3) and (3.6). Hence, to ensure the derivatives of $L_{t_w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ stay bounded, we need to constrain

our parameter space as $\mathcal{K}_\theta = \{vec(\boldsymbol{W}) : \|\boldsymbol{W}\| \leq \lambda\}$ *and* $\mathcal{K}_\mu = \{vec(\boldsymbol{U}) : \|\boldsymbol{U}\| \leq \lambda\}$ *for some* $\lambda \in [0, 1)$*. Note that since* $\mathcal{K}_\theta$ *and* $\mathcal{K}_\mu$ *are convex sets for any* $\lambda \in [0, 1)$*, our constraint does not violate the setting described in the previous subsection.*

Now that we have found $\lambda \in [0, 1)$ is sufficient for $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ to have bounded derivatives in any $t \in [T]$, in the following theorem, we provide its smoothness constants.

Theorem 3.2. *Let* $\boldsymbol{\theta} = vec(\boldsymbol{W})$ *and* $\boldsymbol{\mu} = vec(\boldsymbol{U})$ *, where* $\boldsymbol{W}$ *and* $\boldsymbol{U}$ *satisfy* $\|\boldsymbol{W}\| \leq \lambda$*, and* $\|\boldsymbol{U}\| \leq \lambda$ *for some* $\lambda \in [0, 1)$*. Then,* $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ *has the following Lipschitz and smoothness properties:*

$$1) \left\| \frac{\partial^2 L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})}{\partial \boldsymbol{\theta}^2} \right\| \leq \beta_\theta, \text{ where } \beta_\theta = \frac{4n_h\sqrt{n_h}}{(1-\lambda)^3}. \tag{3.10}$$

$$2) \left\| \frac{\partial^2 L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})}{\partial \boldsymbol{\mu}^2} \right\| \leq \beta_\mu, \text{ where } \beta_\mu = \frac{4n_x\sqrt{n_h}}{(1-\lambda)^3}. \tag{3.11}$$

$$3) \left\| \frac{\partial^2 L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\mu}} \right\| \leq \beta_{\theta\mu}, \text{ where } \beta_{\theta\mu} = \frac{4n_h\sqrt{n_x}}{(1-\lambda)^3}. \tag{3.12}$$

In the following section, we use these properties to develop an SRNN training algorithm with a convergence guarantee.

3.3.3 Main Algorithm

In this part, we present our algorithm, namely the Windowed Online Gradient Descent Algorithm (WOGD), shown in Algorithm 3.

In the algorithm, we take the learning rate $\eta \in [0, 1)$, window-size $w \in [T]$ and $\lambda \in [0, 1)$ as the inputs. We, then, define the parameter spaces $\mathcal{K}_\theta$, $\mathcal{K}_\mu$, and $\mathcal{K}_\vartheta$ in line 3. Here, we define $\mathcal{K}_\theta$ and $\mathcal{K}_\mu$ as given in Remark 3.1 to ensure that the derivatives of the loss functions are bounded. Furthermore, we define $\mathcal{K}_\vartheta$ as $\mathcal{K}_\vartheta = \{\boldsymbol{\vartheta} : \|\boldsymbol{\vartheta}\| \leq 1\}$ to satisfy our assumption of $\|\boldsymbol{\vartheta}\| \leq 1$.

Algorithm 3 Windowed Online Gradient Descent Algorithm (WOGD)

1: **Parameters:**

- Learning rate $\eta \in [0, 1)$
- Window-size $w \in [T]$
- $\lambda \in [0, 1)$

2: Initialize $\boldsymbol{\theta}_1$, $\boldsymbol{\mu}_1$, $\boldsymbol{\vartheta}_1$ and $\mathbf{h}_0$.

3: Let

- $\mathcal{K}_\theta = \{\text{vec}(\mathbf{W}) : \|\mathbf{W}\| < \lambda\}$
- $\mathcal{K}_\mu = \{\text{vec}(\mathbf{U}) : \|\mathbf{U}\| \leq \lambda\}$
- $\mathcal{K}_\vartheta = \{\boldsymbol{\vartheta} : \|\boldsymbol{\vartheta}\| \leq 1\}$

4: **for** $t = 1$ **to** T **do**

5: Predict $\boldsymbol{\theta}_t$, $\boldsymbol{\mu}_t$ and $\boldsymbol{\vartheta}_t$.

6: Receive $\mathbf{x}_t$ and generate $\hat{d}_t$.

7: Observe d_t and the cost function $\ell_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$.

8: Updates:

$$\boldsymbol{\vartheta}_{t+1} = \Pi_{\mathcal{K}_\vartheta}\Big[\boldsymbol{\vartheta}_t - \frac{1}{\sqrt{t}}\frac{\partial L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}{\partial \boldsymbol{\vartheta}_t}\Big] \tag{3.13}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\frac{\partial_{\mathcal{K}_\theta} L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}{\partial \boldsymbol{\theta}} \tag{3.14}$$

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t - \eta\frac{\partial_{\mathcal{K}_\mu} L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}{\partial \boldsymbol{\mu}}. \tag{3.15}$$

9: **end for**

In the learning part, we first predict the hidden layer weight matrices, i.e., $\boldsymbol{\theta}_t$ and $\boldsymbol{\mu}_t$, and the output layer weights, i.e., $\boldsymbol{\vartheta}_t$ (see line 5). Then, we receive the input vector $\mathbf{x}_t$ and generate our estimate $\hat{d}_t$ by running the model in (3.1)-(3.2). We next observe ground truth value d_t and the loss function $\ell_t(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$ in line 7. Having observed the label, we update the weight matrices in line 8 (or in (3.13)-(3.15)). Here, we update the output layer weights $\boldsymbol{\vartheta}_t$ with the projected online gradient descent algorithm [40]. We update the hidden weights in (3.14)-(3.15) by using their projected partial derivatives defined with $(\mathcal{K}_\theta, \eta)$ and $(\mathcal{K}_\mu, \eta)$.

Note that since we constructed our setting for adversarial $\boldsymbol{\vartheta}_t$ selections, the update rule for the output layer in (3.13) does not contradict with our analysis. Moreover, since $L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$ is the average of last w losses, which are all convex with respect to $\boldsymbol{\vartheta}_t$, $L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)$ is also convex with respect to $\boldsymbol{\vartheta}_t$. Then, by using [40, Theorem 1], we can prove the update rule in (3.13) converges to the best possible output layer weights satisfying $\|\boldsymbol{\vartheta}\| \leq 1$. Therefore, in the following theorem, we provide the convergence guarantee of WOGD specifically for the hidden layer weights.

Theorem 3.3. *Let $\ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})$ and $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ be the loss and time-smoothed loss functions defined in (3.3) and (3.6), respectively. Moreover, let β be the maximum possible smoothness constant, i.e.,*

$$\beta = \max\{\beta_\theta, \beta_\mu, \beta_{\theta\mu}\}, \tag{3.16}$$

where β_θ, β_μ, and $\beta_{\theta\mu}$ are defined in (3.10), (3.11) and (3.12). Then, if WOGD is run with the parameters

$$0 < \eta \leq \frac{1}{\beta} \tag{3.17}$$

it ensures that

$$R_w(T) \leq \frac{16\sqrt{n_h}}{\eta}\frac{T}{w} + \frac{16\sqrt{n_h}}{\eta}, \tag{3.18}$$

where $R_w(T)$ is the local regret defined in (3.7). By selecting a window-size w such that $\frac{T}{w} = o(T)$, one can bound $R_w(T)$ with a sublinear bound, hence, guarantee convergence of the hidden layer weights to the locally optimum parameters.[3]

Theorem 3.3 shows that with appropriate parameter selections, WOGD guarantees to learn the locally optimum SRNN parameters for any bounded input/output sequences. Moreover, here, we observe that the window-size of the algorithm directly controls its convergence rate (see (3.18)). That will the key property of our algorithm to obtain the regression performance of LSTMs with SRNNs.

[3] *We use little-o notation, i.e., $g(x) = o(f(x))$, to describe an upper-bound that cannot be tight, i.e., $\lim_{x\to\infty} g(x)/f(x) = 0$.*

Now that we have proved the convergence guarantee of WOGD, in the following remark, we investigate the computational requirement of WOGD.

Remark 3.2. *The most expensive operation of WOGD is the update rule of the hidden layer weights, i.e., (3.14)-(3.15), which can also be written in the form as (see (3.4)-(3.5))*

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{t+1} &= \boldsymbol{\theta}_t - \eta \frac{\partial L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}{\partial \boldsymbol{\theta}} \quad \text{- Update} \\ \boldsymbol{\theta}_{t+1} &= \Pi_{\mathcal{K}_\theta}\left[\hat{\boldsymbol{\theta}}_{t+1}\right] \qquad\qquad \text{- Projection.}\end{aligned}$$

Note that the update part of requires the computation of the partial derivatives of $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$. Additionally, the projection part projects $\hat{\boldsymbol{\theta}}_{t+1}$ onto the spectral unit ball $\mathcal{K}_\theta$, which requires a matrix projection operation.

To compute the partial derivatives, we use the Truncated Backpropagation Through Time algorithm [13], which has $O\big(hn_h(n_h+n_x)\big)$ computational complexity with a truncation length h. Since WOGD uses the partial derivatives of the last w losses, we can approximate to these partial derivatives with a single backpropagation by using a truncation length w, which results in $O\big(wn_h(n_h+n_x)\big)$ computational requirement for computing the partial derivatives.

Furthermore, the projection step can be written as

$$\boldsymbol{\theta}_{t+1} = \underset{\boldsymbol{\theta}\in\mathcal{K}_\theta}{\arg\min} \|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{t+1}\|. \tag{3.19}$$

By [41, Proposition 9], (3.19) can be performed by computing the singular value decomposition (SVD) of the matrix form of $\hat{\boldsymbol{\theta}}_{t+1}$ and clipping its singular values with λ. Moreover, we can reduce the computational requirement of this part by performing projection only when the ℓ_2 norm of the weights exceed some predetermined threshold α, i.e.,

$$\boldsymbol{\theta}_{t+1} = \begin{cases} \arg\min_{\boldsymbol{\theta}\in\mathcal{K}_\theta} \|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{t+1}\| & \|\hat{\boldsymbol{\theta}}_{t+1}\| > \alpha \\ \hat{\boldsymbol{\theta}}_{t+1} & \|\hat{\boldsymbol{\theta}}_{t+1}\| \le \alpha. \end{cases} \tag{3.20}$$

Note that since the weight matrices are initialized close to the origin, (3.20) efficiently ensures the stability of the weight matrices with a reasonable α selection, such as $\alpha \in [5, 10]$.

During the experiments, we observe that when the hidden size is chosen in a reasonable range, i.e., $5 \leq n_h \leq 20$*, WOGD performs the projection step very rarely -at most* 3 *times in a single simulation. Therefore, the main computational bottleneck of our algorithm is to compute the partial derivatives of* $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ *with respect to* $\boldsymbol{\theta}$ *and* $\boldsymbol{\mu}$*, which requires* $O\big(wn_h(n_h+n_x)\big)$*, i.e., a linear time complexity in the number of weights. With this complexity, our algorithm has the same computational requirement as SGD [13].*

In the next remark, we discuss the effect of choosing higher learning rate than the theoretically guaranteed one in Theorem 2.

Remark 3.3. *We note that WOGD is constructed by assuming the worst-case Lipschitz constants derived in Theorem 3.2. On the other hand, our experiments suggest that in practice, the landscape of the objective function is generally nicer than what is predicted by our theoretical development. For example, in the simulations, we observe that the smoothness of the error surface is usually* 10^4 *to* 10^5 *times smaller than their theoretical upper-bounds given in (3.10)-(3.12). Therefore, it is practically possible to obtain vanishing regret with WOGD by using much higher learning rate than the theoretically guaranteed one. Since the regret bound of WOGD is inversely proportional with the learning rate (see (3.18)), in the following, we use WOGD with the higher learning rates than suggested in Theorem 3.3 to obtain faster convergence.*

3.4 Experiments

In this section, we verify our theoretical results and demonstrate the performance improvements of our algorithm. To this end, we use four real-life and two synthetic datasets. We compare our algorithm with three widely used neural-network training algorithms: Adam [6], RMSprop [5], and SGD [28]. To illustrate performance differences between the models, we apply Adam, RMSprop, and SGD both to SRNNs [27] and LSTMs [15]. Since the main focus of our paper is to obtain the regression performance of LSTMs with SRNNs, we apply WOGD only

to SRNNs. In the following, we use the prefixes "SRNN-" and "LSTM-" to denote to which model the optimization algorithm is applied, such as SRNN-Adam, LSTM-RMSprop, or SRNN-WOGD.

In the experiments, we use the most widely used LSTM model, where the activation functions are set to the hyperbolic tangent function, and the peep-hole connections are eliminated [34]. As the SRNN model, we use the standard Elman model given in (3.1)-(3.2). Moreover, we implement our algorithm slightly differently than Algorithm 3 to obtain its maximum performance. Differing from Algorithm 3, we use $8/\sqrt{t}$ as the output layer learning rate and choose the maximum ℓ_2 norm of the output layer weights as 2.5, i.e., $\|\boldsymbol{\vartheta}_t\| \leq 2.5$ for $t \in [T]$. We emphasize that these changes do not hurt our convergence guarantee as they alter the regret bound in Theorem 3.3 only by a constant factor.

In all simulations, we randomly draw the initial weights from a Gaussian distribution with zero mean and standard deviation of 0.1. In all SRNN-WOGD runs, we use $\alpha = 7.5$ and $\lambda = 0.95$. We search the hyperparameters of the learning algorithms over a dense grid and report the results using the best hyperparameters in that setting, i.e., the hyperparameters that provide the smallest mean squared error. We run each experiment 30 times and provide the mean performance of the algorithms.

3.4.1 Real Data Benchmark

3.4.1.1 Pumadyn Datasets

In the first part, we consider the pumaydn dataset [36], which includes 7000 input/output pairs obtained from the simulation of Unimation Puma 560 robotic arm, i.e., $T = 7000$. Here, we aim to estimate the angular acceleration of the arm by using the angular position and angular velocity of the links. To compare the algorithms under various scenarios, we use two variants of the pumaydn

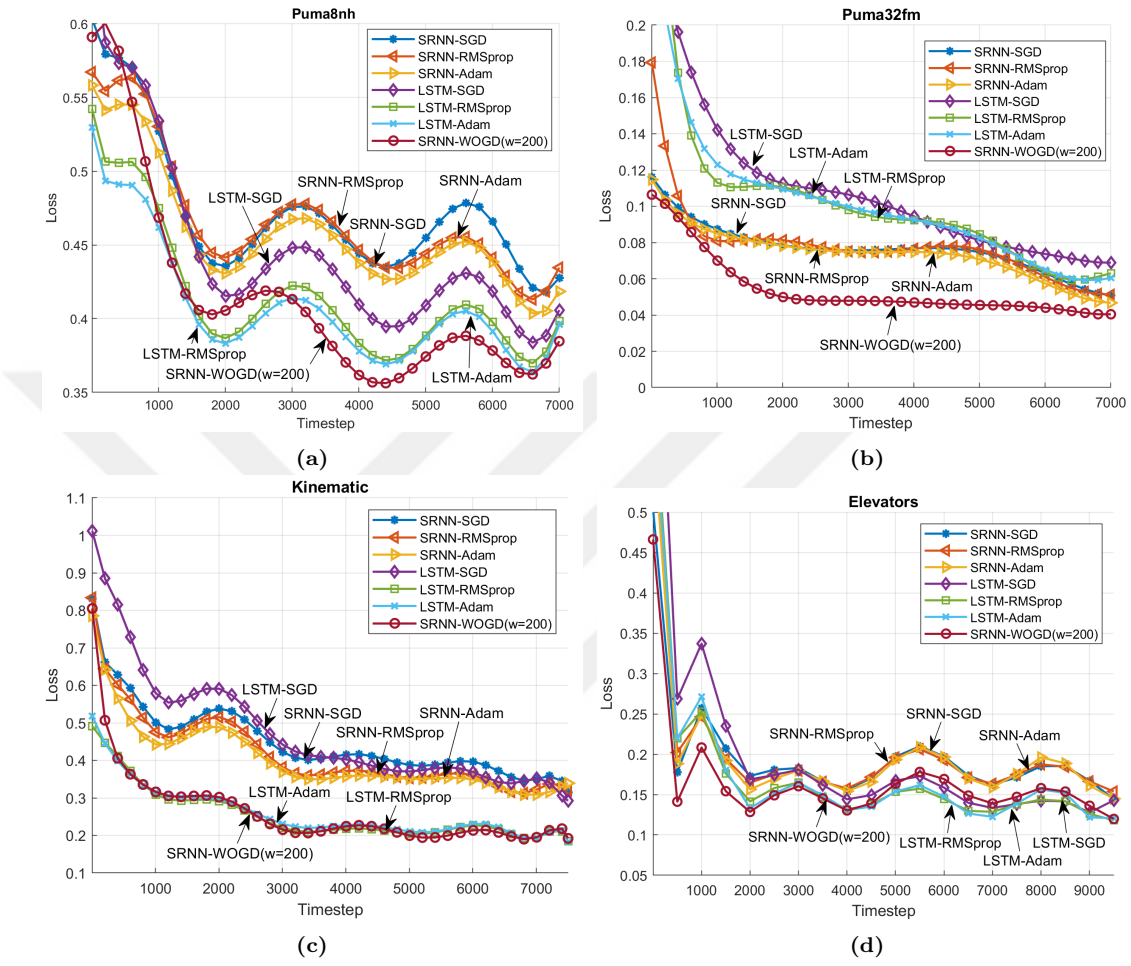

Figure 3.2: Sequential prediction performances of the algorithms on the (a) puma8nh, (b) puma32fm, (c) kinematic, and (d) elevators datasets.

dataset with different difficulties, i.e., puma8nh, which is generated with nonlinear dynamics and high noise, and puma32fm, which is generated with fairly linear dynamics and moderate noise.

For the puma8nh dataset, we use 8-dimensional input vectors of the dataset with an additional bias dimension, i.e., $n_x = 9$, and 12-dimensional state vectors, i.e., $n_h = 12$. In SRNN-Adam, SRNN-RMSprop, and SRNN-SGD, we use the learning rates of 0.007, 0.005, and 0.03, respectively. For LSTM-Adam, LSTM-RMSprop, and LSTM-SGD, we choose the learning rates as 0.01, 0.01, and 0.007. In SRNN-WOGD, we use $\eta = 0.03$. To test the effect of the window-size on performance, we run SRNN-WOGD with three different window-sizes, i.e., $w \in$

Datasets	Puma8nh		Puma32fm		Kinematics		Elevators	
Algorithms	MSE	Run-time(s)	MSE	Run-time(s)	MSE	Run-time(s)	MSE	Run-time(s)
SRNN-WOGD(w = 50)	0.456	1.32	0.068	1.40	0.346	1.31	0.183	1.72
SRNN-WOGD(w = 100)	0.428	2.14	0.060	2.29	0.293	2.06	0.168	2.80
SRNN-WOGD(w = 200)	**0.408**	**4.16**	**0.053**	**4.66**	**0.263**	**4.08**	**0.158**	**5.42**
SRNN-SGD	0.471	0.87	0.075	0.95	0.448	0.83	0.186	1.09
SRNN-RMSprop	0.467	0.93	0.076	0.98	0.419	0.88	0.188	1.15
SRNN-Adam	0.451	0.98	**0.072**	**1.05**	0.407	0.94	0.186	1.28
LSTM-SGD	0.453	7.32	0.111	7.77	0.316	7.33	0.189	9.65
LSTM-RMSprop	0.412	7.80	0.101	7.85	**0.261**	**7.55**	0.162	10.13
LSTM-Adam	**0.408**	**7.60**	0.100	7.97	0.265	7.91	**0.160**	**10.10**

Table 3.1: Mean squared errors and the corresponding run-times (in seconds) of the compared algorithms. The two algorithms with the lowest two mean errors are emphasized. The simulations are performed on a computer with i7-7500U processor, 2.7-GHz CPU, and 8-GB RAM.

$\{50, 100, 200\}$.

We plot the learning curves of the puma8nh experiment in Fig. 3.2a. Here, we see that the LSTM-based methods and SRNN-WOGD(w = 200) outperform the SRNN-based state-of-the-art methods while providing very similar performances. We present the mean errors and the run-times of this part in the first column of Table 3.1. In the table, we observe that as consistent with our theoretical results, the performance of SRNN-WOGD improves as its window-size, i.e., w, gets larger. Moreover, we see that LSTM-RMSprop and SRNN-WOGD(w = 200) are on a par error-wise, while SRNN-WOGD(w = 200) achieves the equivalent performance in almost two times shorter run-time.

For the puma32fm dataset, we use 32-dimensional input vectors of the dataset with an additional bias dimension, i.e., $n_x = 33$, and 12-dimensional state vectors, i.e., $n_h = 12$. In SRNN-Adam, SRNN-RMSprop, and SRNN-SGD, we choose the learning rates as 0.001, 0.001, and 0.01. For LSTM-Adam, LSTM-RMSprop, and LSTM-SGD, we use the learning rates of 0.002, 0.002, and 0.065. In SRNN-WOGD, we use $\eta = 0.08$ and $w \in \{50, 100, 200\}$.

We plot the learning curves of the puma32fm experiment in Fig. 3.2b. In the figure, we see that SRNN-WOGD(w = 200) converges to the small loss values

much faster than the other algorithms. Additionally, we observe that the LSTM-based methods converge to the same loss values as the SRNN-based models, yet more slowly due to their higher number of parameters. We present the mean errors and the run-times of this part in the second column of Table 3.1. Here, we see that as in the previous experiment, the error of SRNN-WOGD reduces as its window size increases. Moreover, we observe that SRNN-WOGD(w = 200) provides a considerably smaller mean error compared to the other models due to its relatively fast convergence.

3.4.1.2 Kinematic and Elevators Datasets

In the second part, we consider the kinematic and elevators datasets, which include 7500 and 9500 input/output pairs, respectively [36, 37]. The kinematic dataset is obtained from a simulation of an eight-link all-revolute robotic arm, and the aim is to predict the distance of the effector from a target. The elevators dataset is obtained from a procedure related to controlling an F16 aircraft, and the aim is to predict the variable that expresses the actions of the aircraft.

For the kinematic dataset, we use 8-dimensional input vectors of the dataset with an additional bias dimension, i.e., $n_x = 9$, and 15-dimensional state vectors, i.e., $n_h = 15$. In SRNN-Adam, SRNN-RMSprop, and SRNN-SGD, we choose the learning rates as 0.007, 0.007, and 0.035. For LSTM-Adam, LSTM-RMSprop, and LSTM-SGD, we use the learning rates of 0.009, 0.01, and 0.15. In SRNN-WOGD, we use $\eta = 0.075$ and $w \in \{50, 100, 200\}$.

For the elevators dataset, we use 18-dimensional input vectors of the dataset with an additional bias dimension, i.e., $n_x = 19$, and 15-dimensional state vectors, i.e., $n_h = 15$. In SRNN-Adam, SRNN-RMSprop, and SRNN-SGD, we choose the learning rates as 0.002, 0.002, and 0.02. For LSTM-Adam, LSTM-RMSprop, and LSTM-SGD, we use the learning rates of 0.004, 0.004, and 0.05. In SRNN-WOGD, we use $\eta = 0.04$ and $w \in \{50, 100, 200\}$.

We plot the learning curves of the kinematic and elevators experiments in

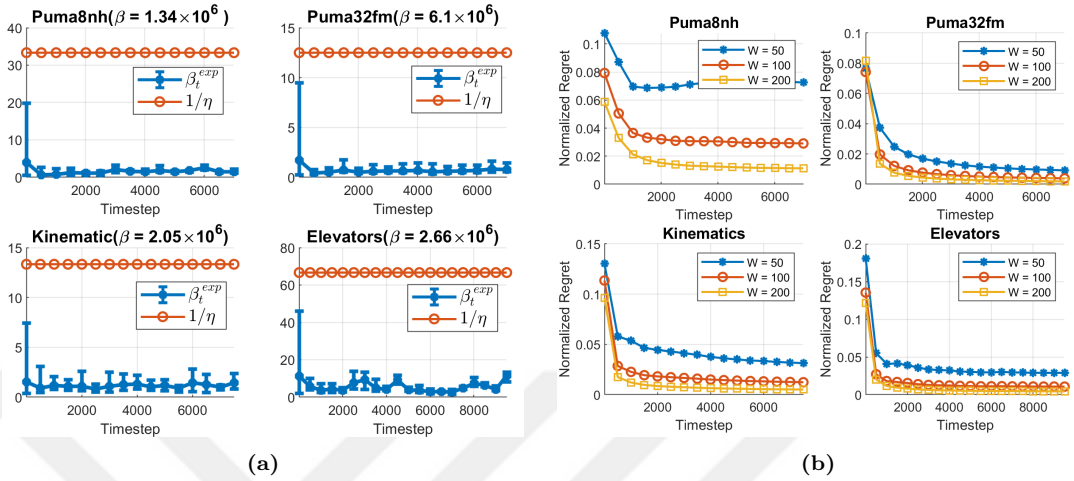

Figure 3.3: (a) Comparison between smothnesses the error surfaces and their theoretical upper-bounds formulated in (3.16)-(3.17). (b) The normalized regret bounds of SRNN-WOGD, i.e., $R_w(t)/t$ for $t \in [T]$, with varying window-sizes.

Fig. 3.2c and Fig. 3.2d. Here again, we observe that the SRNNs trained with the state-of-the-art algorithms perform the worst, followed by the LSTM-based models and SRNN-WOGD(w = 200), which give substantially better results. We present the mean errors and the run-times of this part in the last two columns of Table 3.1. Here, we see that as in the previous experiments, the larger window-size SRNN-WOGD has, the lower mean error it attains. Moreover, we observe that in both experiments, SRNN-WOGD(w = 200) provides very similar mean error with the best LSTM models, yet within two times shorter training time due to efficiency improvements of our algorithm achieved by utilizing SRNNs.

3.4.1.3 Experimental Verification of Theoretical Results

In this section, we further verify our theoretical results by analyzing the smoothness of the error surface and behavior of the normalized regret in the simulations. To observe the smoothness parameters efficiently without calculating the Hessian matrix, we use the finite differences formulated as

$$\beta_t^{exp} = \max\{\beta_{\theta,t}^{exp}, \beta_{\mu,t}^{exp}\},$$

where

$$\beta_{\theta,t}^{exp} = \frac{\left\| \frac{\partial L_{t,w}(\boldsymbol{\theta}_{t+1},\boldsymbol{\mu}_{t+1})}{\partial \boldsymbol{\theta}} - \frac{\partial L_{t,w}(\boldsymbol{\theta}_{t},\boldsymbol{\mu}_{t})}{\partial \boldsymbol{\theta}} \right\|}{\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|} \text{ and } \beta_{\mu,t}^{exp} = \frac{\left\| \frac{\partial L_{t,w}(\boldsymbol{\theta}_{t+1},\boldsymbol{\mu}_{t+1})}{\partial \boldsymbol{\mu}} - \frac{\partial L_{t,w}(\boldsymbol{\theta}_{t},\boldsymbol{\mu}_{t})}{\partial \boldsymbol{\mu}} \right\|}{\|\boldsymbol{\mu}_{t+1} - \boldsymbol{\mu}_t\|}.$$

We note that since we use very small learning rates in SRNN-WOGD, the given finite differences closely approximate the smoothness of the error surface in the direction of the gradient update.

In Fig. 3.3a, we plot the smoothness parameters of the error surface that is obtained from the simulations of SRNN-WOGD(w = 200). In the plots, error bars extend between the minimum and maximum smoothness values observed in 30 simulations. Moreover, we provide the theoretical upper bounds (formulated in Theorem 3.2) in the title of the plots. In the figure, we observe that as indicated in Remark 3.3, the error surface is much smoother than what is predicted by its theoretical upper-bounds. We note that this gap is expected, since we derived the upper-bounds by considering the worst-case scenario, i.e., saturation region of SRNNs, which is rarely encountered in practice due to variations in real-world data.

Additionally, we see in the plots that the selected learning rates satisfy the condition of Theorem 3.3 given in (3.17). To verify the regret bound, we plot the normalized regret, i.e., $R_w(t)/t$ for $t \in [T]$, of all SRNN-WOGD runs in Fig. 3.3b. Here, we see that as consistent with our theoretical derivation, the normalized regret vanishes and its convergence rate increases as the window-size gets larger, which agrees with the results of the previous four experiments.

3.4.2 Binary Addition

In this part, we compare the performance of the algorithms on a synthesized dataset that requires learning long-term dependencies. We show that SRNN-WOGD learns the long-term dependencies comparably well with LSTMs, which explains its success in the previous experiments.

To compare the algorithms, we construct a synthesized experiment in which we

Algorithms	SRNN-WOGD(w=200)		LSTM-RMSprop		SRNN-RMSprop	
Net	Sustainable Prediction	Run-Time(s)	Sustainable Prediction	Run-Time(s)	Sustainable Prediction	Run-Time(s)
1	**1911**	0.73	2787	2.23	5325	**0.51**
2	**2050**	**0.85**	2828	2.24	9535	0.93
3	**1454**	**0.56**	2588	1.99	14331	1.39
4	**1934**	**0.79**	3799	2.87	9079	0.87
5	**1891**	**0.74**	2607	2.03	10125	0.96

(a)

Algorithms	SRNN-WOGD(w=200)		LSTM-RMSprop		SRNN-RMSprop	
Net	Sustainable Prediction	Run-Time(s)	Sustainable Prediction	Run-Time(s)	Sustainable Prediction	Run-Time(s)
1	**19398**	9.53	22553	16.97	32902	**3.57**
2	**18891**	**9.03**	30062	23.48	Failed	
3	27173	11.99	**15171**	**11.42**	Failed	
4	**21499**	**10.46**	42526	32.38	Failed	
5	**23718**	**11.64**	25623	19.57	Failed	

(b)

Table 3.2: Time steps and run-times required to achieve 1000 subsequent correct predictions, i.e., sustainable prediction. The algorithm requiring the smallest number of time step for sustainable prediction is emphasized. In (a), we consider adding 2 sequences. In (b), we consider adding 3 sequences.

can control the length of temporal dependence. To this end, we train the network to learn the summation of n number of binary sequences, where the carry bit is the temporal dependency that the models need to learn. We note that the number of added sequences, i.e., n, controls how long the carry bit is propagated on average, hence, the average length of the temporal dependence. We learn binary addition with a purely online approach, i.e., there is only one single input stream, and learning continues even when the network makes a mistake.

In the experiments, we consider $n \in \{2, 3\}$. For each n, we repeat the experiments with 5 different input streams. We generate the input sequences randomly, where 0 and 1's are drawn with equal probabilities. In the models, we use the sigmoid function as the output layer activation function and the cross-entropy loss as the loss function. We assume that the network decides 1 when its output is bigger than 0.5, and 0 in vice versa. For the performance comparison, we count the number of symbols needed to attain error-free predictions for 1000 subsequent symbols, i.e., sustainable prediction. We note that as the models are more capable of learning long-term dependencies, the number of steps required to obtain sustainable prediction is expected to be smaller.

In the first experiment, we consider adding 2 sequences. The results are presented in Table 3.2a. Due to space constraints, we compare our algorithm only with RMSprop since we observe that RMSprop generally provides the fastest convergence in this task. In the table, we see that SRNN-WOGD(w = 200) consistently achieves the sustainable prediction with considerably less number of time steps compared to both SRNNs and LSTMs. Moreover, due to the efficiency of our algorithm, SRNN-WOGD(w = 200) requires almost three times smaller run-time compared to LSTMs.

In the second experiment, we consider adding 3 sequences. The results are presented in Table 3.2b. Here, we see that SRNN-RMSprop fails in the four out of five simulations, i.e., it could not achieve the sustainable prediction within 5×10^4-time steps. Moreover, as in the previous experiment, SRNN-WOGD(w = 200) obtains the sustainable prediction within a relatively small number of time steps and considerably shorter run-time compared to LSTMs in almost all experiments.

Chapter 4

Conclusion

In this thesis, we studied online nonlinear regression with continually running RNNs. We have introduced two efficient online learning algorithms that achieve the regression performance of LSTMs with significantly shorter training times.

In Chapter 2, we introduced a highly efficient EKF-based second-order training algorithm. To design our algorithm, we first modeled the LSTM-based adaptive regression problem with a state-space model to derive the update equations of EKF and IEKF. Next, we derived an adaptive hyper-parameter selection strategy for IEKF that guarantees errors to converge to a small interval under the assumption that all the data-dependent parameters are known *a priori*. Finally, we extended our algorithm to a truly online form, where the algorithm learns the data-dependent parameters by sequentially observing the data sequence.

In Chapter 3, we introduced a first-order gradient-based optimization algorithm with a linear time complexity in the number of parameters. To design our algorithm, we modeled the SRNN-based online regression problem as a sequential learning problem, where we assumed each time step as a separate loss function assigned by an adversary. We characterized the Lipschitz properties of these loss functions with respect to the network weights and derived sufficient conditions for our model to have bounded derivatives. Then, by using these results, we

have introduced an online gradient descent algorithm, guaranteed to converge to the locally optimum parameters in a strong deterministic sense, i.e., without any stochastic assumptions.

In both chapters, we demonstrated significant performance improvements of our algorithms with respect to the state-of-the-art training methods. To be specific, in Chapter 2, our algorithm provided a considerable improvement in accuracy with respect to the widely-used adaptive methods Adam, RMSprop, and DEKF, and very close performance to EKF with a 10 to 15 times reduction in the training time. In Chapter 3, our algorithm achieved the online regression performance of the LSTMs trained with Adam, RMSprop and SGD, in a 2 to 3 times shorter run-time.

As future work, the use of adaptive learning rate schemes and momentum methods can be considered to improve the performance of the presented algorithms further. Another possible research direction is to explore the performance of our algorithm in different practical problems, such as sequence to sequence learning or sequence generation.

Bibliography

[1] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games.* Cambridge University Press, 2006.

[2] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[3] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems.," *Foundations and Trends in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

[4] E. Hazan, K. Singh, and C. Zhang, "Efficient regret minimization in nonconvex games.," in *ICML*, vol. 70 of *Proceedings of Machine Learning Research*, pp. 1433–1441, PMLR, 2017.

[5] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural Networks for Machine Learning, 2012.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[7] S. S. Kozat and A. C. Singer, "Universal switching linear least squares prediction," *IEEE Transactions on Signal Processing*, vol. 56, pp. 189–204, Jan 2008.

[8] R. Laxhammar and G. Falkman, "Online learning and sequential anomaly detection in trajectories.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 6, pp. 1158–1173, 2014.

[9] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," *IEEE Transactions on Signal Processing*, vol. 52, pp. 2165–2176, Aug 2004.

[10] 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, no. 20, pp. 5471–5486, 2014.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[13] R. J. Williams and D. Zipser, "Backpropagation," ch. Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity, pp. 433–486, Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1995.

[14] Y. Bengio, P. Y. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5 2, pp. 157–166, 1994.

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

[16] F. A. Gers, N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Journal of Machine Learning Research*, vol. 3, 2002.

[17] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. Presented in NIPS 2014 Deep Learning and Representation Learning Workshop.

[18] 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. 28, no. 10, pp. 2222–2232, 2017.

[19] J. Antonio Perez-Ortiz, F. Gers, D. Eck, and J. Schmidhuber, "Kalman filters improve lstm network performance in problems unsolvable by traditional recurrent nets," *Neural networks : the official journal of the International Neural Network Society*, vol. 16, pp. 241–50, 04 2003.

[20] J. Koutnik, K. Greff, F. J. Gomez, and J. Schmidhuber, "A clockwork rnn.," in *ICML*, vol. 32 of *JMLR Workshop and Conference Proceedings*, JMLR.org, 2014.

[21] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd ed., 1998.

[22] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, (USA), pp. 1033–1040, Omnipress, 2011.

[23] S. A. Shah, F. Palmieri, and M. Datum, "Optimal filtering algorithms for fast learning in feedforward neural networks," *Neural Networks*, vol. 5, no. 5, pp. 779–787, 1992.

[24] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks," *IEEE Transactions on Neural Networks*, vol. 5, pp. 279–297, March 1994.

[25] J. Martens, *Second-order Optimization for Neural Networks*. PhD thesis, University of Toronto, 2016.

[26] H. Coskun, F. Achilles, R. S. DiPietro, N. Navab, and F. Tombari, "Long short-term memory kalman filters: Recurrent neural estimators for pose regularization.," in *ICCV*, pp. 5525–5533, IEEE Computer Society, 2017.

[27] J. L. Elman, "Finding structure in time," *COGNITIVE SCIENCE*, vol. 14, no. 2, pp. 179–211, 1990.

[28] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[29] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning.," in *ICML (3)*, vol. 28 of *JMLR Workshop and Conference Proceedings*, pp. 1139–1147, JMLR.org, 2013.

[30] Y. N. Dauphin, H. de Vries, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization.," in *NIPS* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 1504–1512, 2015.

[31] J. Rubio Avila and W. Yu Liu, "Nonlinear system identification with recurrent neural networks and dead-zone kalman filter algorithm," *Neurocomputing*, vol. 70, pp. 2460–2466, 8 2007.

[32] X. Wang and Y. Huang, "Convergence study in extended kalman filter-based training of recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 22, pp. 588–600, April 2011.

[33] G. V. Puskorius and L. A. Feldkamp, "Extensions and enhancements of decoupled extended kalman filter training," in *Proceedings of International Conference on Neural Networks (ICNN'97)*, vol. 3, pp. 1879–1883 vol.3, June 1997.

[34] T. Ergen and S. S. Kozat, "Efficient online learning algorithms based on lstm neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, pp. 3772–3783, Aug 2018.

[35] H. Yang, Z. Pan, and Q. Tao, "Robust and adaptive online time series prediction with long short-term memory.," *Comput. Intell. Neurosci.*, 2017.

[36] C. E. Rasmussen, "Delve data sets." `http://www.cs.toronto.edu/~delve/data/datasets.html`. Accessed: 2020-06-15.

[37] J. Alcala-Fdez, A. Fernandez, J. Luengo, J. Derrac, and S. Garcia, "Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework.," *Multiple-Valued Logic and Soft Computing*, vol. 17, no. 2-3, pp. 255–287, 2011.

[38] I. Sutskever and G. E. Hinton, "Temporal-kernel recurrent neural networks.," *Neural Networks*, vol. 23, no. 2, pp. 239–243, 2010.

[39] T. Lin, B. G. Horne, P. Tiño, and C. L. Giles, "Learning long-term dependencies in NARX recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 7, pp. 1329–1338, November 1996.

[40] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent.," in *ICML* (T. Fawcett and N. Mishra, eds.), pp. 928–936, 2003.

[41] H. Sedghi, V. Gupta, and P. M. Long, "The singular values of convolutional layers.," in *ICLR (Poster)*, OpenReview.net, 2019.

[42] Y. Hsieh and V. Cevher, "Dimension-free information concentration via exp-concavity," *CoRR*, vol. abs/1802.09301, 2018.

[43] P. Lancaster and H. Farahat, "Norms on direct sums and tensor products," *Mathematics of Computation - Math. Comput.*, vol. 26, 05 1972.

[44] S. Bubeck, "Convex optimization: Algorithms and complexity," *Found. Trends Mach. Learn.*, vol. 8, pp. 231–357, Nov. 2015.

Appendix A

Supplement for Chapter 2

A.1 Proof of Lemma 2.1

In the following, we manipulate the IEKF update rules in (2.14)-(2.17) to obtain the statements in Lemma 2.1. We note that since Algorithm 1 performs the IEKF updates if $\|\mathbf{e}_t\|^2 > 4\bar{\zeta}^2$, it guarantees the following statements for $\{t : \|\mathbf{e}_t\|^2 > 4\bar{\zeta}^2\}$.

1. By multiplying both sides of (2.14) with -1, we write:

$$-\hat{\boldsymbol{\theta}}_{i,t+1} = -\hat{\boldsymbol{\theta}}_{i,t} - \mathbf{G}_{i,t}(\mathbf{d}_t - \hat{\boldsymbol{d}}_t). \tag{A.1}$$

Then by using (2.8), we add $\boldsymbol{\theta}_{i,t+1}$ and $\boldsymbol{\theta}_{i,t}$ to both sides of (A.1) respectively:

$$\boldsymbol{\theta}_{i,t+1} - \hat{\boldsymbol{\theta}}_{i,t+1} = (\boldsymbol{\theta}_{i,t} - \hat{\boldsymbol{\theta}}_{i,t}) - \mathbf{G}_{i,t}(\mathbf{d}_t - \hat{\boldsymbol{d}}_t). \tag{A.2}$$

By using the Taylor series expansion in (2.19) and using the notation $\boldsymbol{\chi}_{i,t} = (\boldsymbol{\theta}_{i,t} - \hat{\boldsymbol{\theta}}_{i,t})$, we write

$$\boldsymbol{\chi}_{i,t+1} = \boldsymbol{\chi}_{i,t} - \mathbf{G}_{i,t}\mathbf{H}_{i,t}\boldsymbol{\chi}_{i,t} - \mathbf{G}_{i,t}\boldsymbol{\zeta}_{i,t}. \tag{A.3}$$

The statements in (2.22) and (2.23) follow (A.3).

2. By (2.15), for all $i \in [(4n_s + n_d)]$,

$$\mathbf{P}_{i,t+1} - q_t\mathbf{I} = (\mathbf{I} - \mathbf{G}_{i,t}\mathbf{H}_{i,t})\mathbf{P}_{i,t} \tag{A.4}$$

$$= \big(\mathbf{I} - \mathbf{P}_{i,t}\mathbf{H}_{i,t}^T(\mathbf{H}_{i,t}\mathbf{P}_{i,t}\mathbf{H}_{i,t}^T + r_{i,t}\mathbf{I})^{-1}\mathbf{H}_{i,t}\big)\mathbf{P}_{i,t} \tag{A.5}$$

$$= \mathbf{P}_{i,t} - \mathbf{P}_{i,t}\mathbf{H}_{i,t}^T(\mathbf{H}_{i,t}\mathbf{P}_{i,t}\mathbf{H}_{i,t}^T + r_{i,t}\mathbf{I})^{-1}\mathbf{H}_{i,t}\mathbf{P}_{i,t} \tag{A.6}$$

where we use the formulation of $\mathbf{G}_{i,t}$ in (2.17) to write (A.4) from (A.5). By applying the matrix inversion lemma[1] to (A.6), we write

$$(\mathbf{P}_{i,t+1} - q_t\mathbf{I})^{-1} = \mathbf{P}_{i,t}^{-1} + r_{i,t}^{-1}\mathbf{H}_{i,t}^T\mathbf{H}_{i,t}. \tag{A.7}$$

By noting that $\mathbf{P}_{i,1}^{-1} > \mathbf{0}$, and using (A.7) as the induction hypothesis, it can be shown that $(\mathbf{P}_{i,t+1} - q_t\mathbf{I})^{-1}$ exists and $(\mathbf{P}_{i,t+1} - q_t\mathbf{I})^{-1} > \mathbf{0}$, for all $t \in [T]$. Since $q_t \geq 0$, $\mathbf{P}_{i,t}^{-1}$ has the same properties, which leads to (2.25). Also, (2.24) can be reached by taking the inverse of both sides in (A.4).

3. By multiplying both sides of (2.22) with $(\mathbf{P}_{i,t+1} - q_t\mathbf{I})^{-1}$, and using (2.24), (2.26) can be obtained.

A.2 Proof of Theorem 2.2

To prove Theorem 2.2, we use the second method of Lyapunov. Let us fix an arbitrary node $i \in [(4n_s + n_d)]$, and choose the Lyapunov function as

$$V_{i,t} = \boldsymbol{\chi}_{i,t}^T\mathbf{P}_{i,t}^{-1}\boldsymbol{\chi}_{i,t}. \tag{A.8}$$

Let us say that $\Delta V_{i,t} = V_{i,t+1} - V_{i,t}$. Since we update $\mathbf{P}_{i,t}$, and $\hat{\boldsymbol{\theta}}_{i,t}$ only when $\|\mathbf{e}_t\|^2 > 4\bar{\zeta}^2$, for $\{t : \|\mathbf{e}_t\|^2 \leq 4\bar{\zeta}^2\}$, $\Delta V_{i,t} = 0$. Therefore, in the following, we only consider the time steps in which we perform the weight update, i.e, $\{t : \|\mathbf{e}_t\|^2 > 4\bar{\zeta}^2\}$.

[1]Matrix inversion lemma: $(\mathbf{A} - \mathbf{BCD})^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{C}^{-1} - \mathbf{DA}^{-1}\mathbf{B})^{-1}\mathbf{DA}^{-1}$.

To begin with, we write the open formula of $\Delta V_{i,t}$:

$$\Delta V_{i,t} = \boldsymbol{\chi}_{i,t+1}^T \mathbf{P}_{i,t+1}^{-1} \boldsymbol{\chi}_{i,t+1} - \boldsymbol{\chi}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} \tag{A.9}$$

$$\leq \boldsymbol{\chi}_{i,t+1}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \boldsymbol{\chi}_{i,t+1} - \boldsymbol{\chi}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} \tag{A.10}$$

$$= \boldsymbol{\chi}_{i,t+1}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} - \boldsymbol{\chi}_{i,t+1}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t} - \boldsymbol{\chi}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} \tag{A.11}$$

$$= (\boldsymbol{\chi}_{i,t+1} - \boldsymbol{\chi}_{i,t})^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} - \boldsymbol{\chi}_{i,t+1}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t} \tag{A.12}$$

$$= (-\mathbf{G}_{i,t} \mathbf{H}_{i,t} \boldsymbol{\chi}_{i,t} - \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t})^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} - \boldsymbol{\chi}_{i,t+1}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t} \tag{A.13}$$

$$= -\boldsymbol{\chi}_{i,t}^T \mathbf{H}_{i,t}^T \mathbf{G}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} - \boldsymbol{\zeta}_{i,t}^T \mathbf{G}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} - \boldsymbol{\chi}_{i,t+1}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t}, \tag{A.14}$$

where we use the 2nd statement in Lemma 2.1 for (A.10), (2.26) for (A.11), and (2.23) for (A.13). For the sake of notational simplicity, we introduce $\mathbf{M}_{i,t} = (\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T + r_{i,t} \mathbf{I})$, where $\mathbf{G}_{i,t} = \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T \mathbf{M}_{i,t}^{-1}$. Then, we write (A.14) as

$$\Delta V_{i,t} \leq -\boldsymbol{\chi}_{i,t}^T \mathbf{H}_{i,t}^T \mathbf{M}_{i,t}^{-1} \mathbf{H}_{i,t} \boldsymbol{\chi}_{i,t} - \boldsymbol{\zeta}_{i,t}^T \mathbf{M}_{i,t}^{-1} \mathbf{H}_{i,t} \boldsymbol{\chi}_{i,t} - \boldsymbol{\chi}_{i,t+1}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t}. \tag{A.15}$$

We write the last term in (A.15) as

$$-\boldsymbol{\chi}_{i,t+1}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t} = -\boldsymbol{\zeta}_{i,t}^T \mathbf{G}_{i,t}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \boldsymbol{\chi}_{i,t+1} \tag{A.16}$$

$$= -\boldsymbol{\zeta}_{i,t}^T \mathbf{G}_{i,t}^T \Big(\mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} - (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t} \Big) \tag{A.17}$$

$$= -\boldsymbol{\zeta}_{i,t}^T \mathbf{G}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} + \boldsymbol{\zeta}_{i,t}^T \mathbf{G}_{i,t}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t} \tag{A.18}$$

$$= -\boldsymbol{\zeta}_{i,t}^T \mathbf{M}_{i,t}^{-1} \mathbf{H}_{i,t} \boldsymbol{\chi}_{i,t} + \boldsymbol{\zeta}_{i,t}^T \mathbf{G}_{i,t}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t}, \tag{A.19}$$

where we use (2.26) to obtain (A.17). By adopting (A.19) in (A.15), we write

$$\Delta V_{i,t} \leq -\boldsymbol{\chi}_{i,t}^T \mathbf{H}_{i,t}^T \mathbf{M}_{i,t}^{-1} \mathbf{H}_{i,t} \boldsymbol{\chi}_{i,t} - 2\boldsymbol{\zeta}_{i,t}^T \mathbf{M}_{i,t}^{-1} \mathbf{H}_{i,t} \boldsymbol{\chi}_{i,t} + \boldsymbol{\zeta}_{i,t}^T \mathbf{G}_{i,t}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} \boldsymbol{\zeta}_{i,t}. \tag{A.20}$$

We add $\pm \boldsymbol{\zeta}_{i,t}^T \mathbf{M}_{i,t}^{-1} \boldsymbol{\zeta}_{i,t}$ to (A.20), and group the terms as

$$\Delta V_{i,t} \leq \boldsymbol{\zeta}_{i,t}^T \Big(\mathbf{G}_{i,t}^T (\mathbf{P}_{i,t+1} - q_t \mathbf{I})^{-1} \mathbf{G}_{i,t} + \mathbf{M}_{i,t}^{-1} \Big) \boldsymbol{\zeta}_{i,t} - (\mathbf{H}_{i,t} \boldsymbol{\chi}_{i,t} + \boldsymbol{\zeta}_{i,t})^T \mathbf{M}_{i,t}^{-1} (\mathbf{H}_{i,t} \boldsymbol{\chi}_{i,t} + \boldsymbol{\zeta}_{i,t}). \tag{A.21}$$

By using (2.25), the definition of $\mathbf{e}_t$ in (2.19), and formulation of $\mathbf{G}_{i,t}$, we write (A.21) as

$$\begin{aligned}\Delta V_{i,t} \leq & \boldsymbol{\zeta}_{i,t}^T \Big(\mathbf{M}_{i,t}^{-1} \mathbf{H}_{i,t} \mathbf{P}_{i,t} (\mathbf{P}_{i,t}^{-1} + r_{i,t}^{-1} \mathbf{H}_{i,t}^T \mathbf{H}_{i,t}) \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T \mathbf{M}_{i,t}^{-1} + \mathbf{M}_{i,t}^{-1} \Big) \boldsymbol{\zeta}_{i,t} - \mathbf{e}_t^T \mathbf{M}_{i,t}^{-1} \mathbf{e}_t \\ = & \boldsymbol{\zeta}_{i,t}^T \Big(r_{i,t}^{-1} \mathbf{M}_{i,t}^{-1} \mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T \mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T \mathbf{M}_{i,t}^{-1} + \mathbf{M}_{i,t}^{-1} \mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T \mathbf{M}_{i,t}^{-1} + \mathbf{M}_{i,t}^{-1} \Big) \boldsymbol{\zeta}_{i,t} \\ & - \mathbf{e}_t^T \mathbf{M}_{i,t}^{-1} \mathbf{e}_t. \end{aligned} \tag{A.22}$$

Note that since $\mathbf{M}_{i,t}^{-1} = (\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T + r_{i,t} \mathbf{I})^{-1}$, $\mathbf{M}_{i,t}^{-1} \leq r_{i,t}^{-1} \mathbf{I}$, and $\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T \mathbf{M}_{i,t}^{-1} \leq \mathbf{I}$. By using these two inequalities in (A.22), we get

$$\Delta V_{i,t} \leq \frac{3 \|\boldsymbol{\zeta}_{i,t}\|^2}{r_{i,t}} - \frac{\|\mathbf{e}_t\|^2}{\max_{j \in [n_\theta]} \lambda_j (\mathbf{M}_{i,t})}, \tag{A.23}$$

where $\lambda_j(\mathbf{M}_{i,t})$ denotes the jth eigenvalue of $\mathbf{M}_{i,t}$. We note that $\mathrm{Tr}(\mathbf{M}_{i,t})/n_d \leq \max_{j \in [n_\theta]} \lambda_j(\mathbf{M}_{i,t})$, and $\mathrm{Tr}(\mathbf{M}_{i,t}) = \mathrm{Tr}(\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T) + n_d r_{i,t}$. Then, by using $\|\boldsymbol{\zeta}_{i,t}\|^2 \leq \overline{\zeta}^2$ and (A.23), we write

$$\Delta V_{i,t} \leq \frac{3 \overline{\zeta}^2}{r_{i,t}} - \frac{n_d \|\mathbf{e}_t\|^2}{\mathrm{Tr}(\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T) + n_d r_{i,t}}. \tag{A.24}$$

To ensure stability, we need $\Delta V_{i,t} < 0$. For this, it is sufficient to guarantee that the right hand side of (A.24) is smaller than 0, i.e.,

$$\Delta V_{i,t} \leq \frac{3 \overline{\zeta}^2}{r_{i,t}} - \frac{n_d \|\mathbf{e}_t\|^2}{\mathrm{Tr}(\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T) + n_d r_{i,t}} < 0. \tag{A.25}$$

To guarantee (A.25), we need to ensure

$$\frac{3 \mathrm{Tr}(\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T) \overline{\zeta}^2 / n_d}{\|\mathbf{e}_t\|^2 - 3 \overline{\zeta}^2} < r_{i,t}. \tag{A.26}$$

Since we only consider $\{t : \|\mathbf{e}_t\|^2 > 4 \overline{\zeta}^2\}$, we can bound the left hand side of (A.26) as

$$\frac{3 \mathrm{Tr}(\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T) \overline{\zeta}^2 / n_d}{\|\mathbf{e}_t\|^2 - 3 \overline{\zeta}^2} < \frac{3 \mathrm{Tr}(\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T) \overline{\zeta}^2 / n_d}{4 \overline{\zeta}^2 - 3 \overline{\zeta}^2}. \tag{A.27}$$

$$= 3 \mathrm{Tr}(\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T) / n_d \tag{A.28}$$

Therefore $r_{i,t} = 3 \mathrm{Tr}(\mathbf{H}_{i,t} \mathbf{P}_{i,t} \mathbf{H}_{i,t}^T) / n_d$ ensures stability.

As we ensure $\Delta V_{i,t} < 0$ for $\{t : \|\mathbf{e}_t\|^2 > 4\overline{\zeta}^2\}$ and $\Delta V_{i,t} = 0$ for $\{t : \|\mathbf{e}_t\|^2 \leq 4\overline{\zeta}^2\}$, Algorithm 1 guarantees

$$\boldsymbol{\chi}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} \leq \boldsymbol{\chi}_{i,1}^T \mathbf{P}_{i,1}^{-1} \boldsymbol{\chi}_{i,1}^T = p_1^{-1} \|\boldsymbol{\chi}_{i,1}\|^2, \tag{A.29}$$

for all $t \in [T]$. S

Since $\|\boldsymbol{\chi}_{i,1}\|$ is finite, as a result of (A.29), $\boldsymbol{\chi}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t}$ is also finite. Under the condition that $\mathrm{Tr}(\mathbf{P}_{i,t})$ stays bounded, $\|\boldsymbol{\chi}_{i,t}\|$ should stay bounded, which proves the 1st statement of Theorem 2.2. Moreover, since $\Delta V_{i,t} < 0$ for $\{t : \|\mathbf{e}_t\|^2 > 4\overline{\zeta}^2\}$, as the cardinality of $\{t : \|\mathbf{e}_t\|^2 > 4\overline{\zeta}^2\}$ approaches to infinity, $\boldsymbol{\chi}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t}$ approaches to 0, i.e., $\boldsymbol{\chi}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} \to 0$. If $\mathrm{Tr}(\mathbf{P}_{i,t})$ stays bounded, $\boldsymbol{\chi}_{i,t}^T \mathbf{P}_{i,t}^{-1} \boldsymbol{\chi}_{i,t} \to 0$ implies that $\|\boldsymbol{\chi}_{i,t}\| \to 0$. Since we learn the LSTM parameters $\hat{\boldsymbol{\theta}}_t$ only when $\{t : \|\mathbf{e}_t\|^2 > 4\overline{\zeta}^2\}$, by the desired data model in (2.8)-(2.9), Algorithm 1 converges to the weights, which guarantees a loss value lower than or equal to $4\overline{\zeta}^2$. This proves the 2nd statement of Theorem 2.2.

A.3 Proof of Theorem 2.3

To prove the statement in the theorem, we construct the scenario that maximizes $\overline{\zeta}$ for $\hat{\boldsymbol{\theta}}_1 \approx \mathbf{0}$ and show that $\overline{\zeta} \in [0, \sqrt{n_d}]$ in this scenario. We construct this scenario in three steps:

1. Due to our LSTM model in (2.1)-(2.7), as $\|\hat{\boldsymbol{\theta}}_t\|$ approaches to 0, the entries of $\hat{\boldsymbol{d}}_t$ and $\mathbf{H}_t$ approaches to 0 as well, i.e., as $\|\hat{\boldsymbol{\theta}}_t\| \to 0$, $\hat{\boldsymbol{d}}_t \to \mathbf{0}$ and $\mathbf{H}_t \to \mathbf{0}$. Hence, by (2.19), as $\|\hat{\boldsymbol{\theta}}_t\| \to 0$, $\boldsymbol{\zeta}_{i,t} \to \mathbf{e}_t$ for all $t \in [T]$ and $i \in [(4n_s + n_d)]$.

2. Let us assume $\overline{\zeta} \leq \sqrt{n_d}$, which we will validate in the following, and use $\overline{\zeta} = \sqrt{n_d}$. Since $\overline{\zeta} = \sqrt{n_d}$ does not allow any weight update, $\|\hat{\boldsymbol{\theta}}_1\| \leq \epsilon$ and $\overline{\zeta} = \sqrt{n_d}$ result in $\|\hat{\boldsymbol{\theta}}_t\| \leq \epsilon$ for all $t \in [T]$, where ϵ is a small positive number.

3. By the 1st step, $\mathbf{d}_t = \pm\mathbf{1}$ maximizes $\overline{\zeta}$ in the case of $\|\hat{\boldsymbol{\theta}}_t\| \leq \epsilon$. By the 2nd step, if $\|\hat{\boldsymbol{\theta}}_1\| \leq \epsilon$ and $\overline{\zeta} = \sqrt{n_d}$, then $\|\hat{\boldsymbol{\theta}}_t\| \leq \epsilon$. This implies that as $\|\hat{\boldsymbol{\theta}}_1\| \to 0$, $\|\boldsymbol{\zeta}_{i,t}\| \to \|\mathbf{e}_t\| = \sqrt{n_d}$. Thus, our assumption in the 2nd step holds at some

point for any bounded data sequence. Therefore, we can choose sufficiently small initial weights to ensure $\overline{\zeta} \in [0, \sqrt{n_d}]$.

A.4 Proof of Theorem 2.4

For the proof, we use the notion of exp-concavity, which is defined in [42, Definition 2]. Note that by using [42, Definition 2], we can show that $F(\hat{\boldsymbol{d}}_t) = \|\mathbf{d}_t - \hat{\boldsymbol{d}}_t\|^2$, where $\mathbf{d}_t, \hat{\boldsymbol{d}}_t \in [-1, 1]^{n_d}$, is $\frac{1}{8n_d}$-exp-concave.

We note that since we assume $\overline{\zeta}_{\text{best}} \in [\overline{\zeta}_{\min}, \sqrt{n_d}]$, there is an Algorithm 1 instance in Algorithm 2, which uses $\overline{\zeta} \in [\overline{\zeta}_{\text{best}}, 2\overline{\zeta}_{\text{best}}]$. Let us use $\tilde{j}$ for the index of that instance, and $\ell_{j,t}$ to denote the loss of the jth Algorithm 1 instance at time t, i.e, $\ell_{j,t} = \|\mathbf{d}_t - \hat{\boldsymbol{d}}_{j,t}\|^2$. Let us also use $\ell_{A,t}$ to denote the loss of Algorithm 2 at time t, where $\ell_{A,t} = \|\mathbf{d}_t - \hat{\boldsymbol{d}}_t\|^2$. In this proof, our aim is to show that $\lim_{t\to\infty} \ell_{A,t} - \ell_{\tilde{j},t} = 0$. For the proof, we introduce three shorthand notations:

$$W_t = \sum_{j=1}^{N} w_{j,t}, \qquad L_{j,T} = \sum_{t=1}^{T} \ell_{j,t}, \qquad L_{A,T} = \sum_{t=1}^{T} \ell_{A,t}.$$

To begin with, we find a lower bound for $\ln(W_{T+1}/W_1)$:

$$\ln \frac{W_{T+1}}{W_1} = \ln \Big(\sum_{j=1}^{N} \exp(-\frac{1}{8n_d} L_{j,T}) \Big) - \ln N \tag{A.30}$$

$$\geq -\frac{1}{8n_d} L_{\tilde{j},T} - \ln N. \tag{A.31}$$

Then, we find an upper bound for $\ln(W_{t+1}/W_t)$:

$$\ln \frac{W_{t+1}}{W_t} = \ln \Big(\frac{\sum_{j=1}^{N} w_{j,t} \exp(-\frac{1}{8n_d} \|\mathbf{d}_t - \hat{\boldsymbol{d}}_{j,t}\|^2)}{\sum_{k=1}^{N} w_{k,t}} \Big) \tag{A.32}$$

$$\leq \ln \Big(\exp(-\frac{1}{8n_d} \|\mathbf{d}_t - \frac{\sum_{j=1}^{N} w_{j,t} \hat{\boldsymbol{d}}_{j,t}}{\sum_{k=1}^{N} w_{k,t}} \|^2) \Big) \tag{A.33}$$

$$= -\frac{1}{8n_d} \|\mathbf{d}_t - \hat{\boldsymbol{d}}_t\|^2 = -\frac{1}{8n_d} \ell_{A,t}, \tag{A.34}$$

where we use the exp-concavity of $F(\hat{\boldsymbol{d}}_t) = \|\mathbf{d}_t - \hat{\boldsymbol{d}}_t\|^2$ and Jensen's inequality for (A.33). Since $\sum_{t=1}^{T} \ln \frac{W_{t+1}}{W_t} = \ln \frac{W_{T+1}}{W_1}$, by using (A.31) and (A.34), we write

$$-\frac{1}{8n_d} L_{\tilde{j},T} - \ln N \leq -\frac{1}{8n_d} \sum_{t=1}^{T} \ell_{A,t}, \tag{A.35}$$

which can be equivalently written as

$$L_{A,T} - L_{\tilde{j},T} \leq 8n_d \ln N. \tag{A.36}$$

Note that by (A.19) the series $L_{A,T} - L_{\tilde{j},T}$ is convergent, which means $\lim_t \ell_{A,t} - \ell_{\tilde{j},t} = 0$. Since the Algorithm 1 instance with index $\tilde{j}$ has $\overline{\zeta} \in [\overline{\zeta}_{\text{best}}, 2\overline{\zeta}_{\text{best}}]$, by Theorem 2.2, the loss sequence of Algorithm 2 converges to $[0, 16\overline{\zeta}^2_{\text{best}}]$.

Appendix B

Supplement for Chapter 3

B.1 Extension to the Cross-Entropy Loss

In this part, we explain that our algorithm can be used with the cross-entropy loss, without any change. Since the cross-entropy loss is mainly used to learn binary target values, we naturally assume the following RNN architecture:

$$\begin{aligned}\mathbf{h}_t &= \tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t)\\ \hat{p}_t &= \sigma(\boldsymbol{\vartheta}^T \mathbf{h}_t)\\ \ell_t(p_t, \hat{p}_t) &= RE(p_t, \hat{p}_t).\end{aligned}$$

Here, σ is the sigmoid function, i.e., $\sigma(x) = 1/(1+e^{-x})$, $\mathbf{h}_t \in [-1,1]^{n_h}$ is the hidden vector, $\mathbf{x}_t \in [-1,1]^{n_x}$ is the input vector, and $p_t, \hat{p}_t \in [0,1]$ are the target and estimated values. Moreover, RE denotes the cross-entropy loss, i.e., $RE(p_t, \hat{p}_t) = -p_t \log \hat{p}_t - (1-p_t)\log(1-\hat{p}_t)$, and ℓ_t denotes the instantaneous loss at time step t.

As in the squared loss, the cross-entropy is convex with respect to output layer weights $\boldsymbol{\vartheta}$. Therefore, we can use the projected online gradient descent – as in (3.13)– to ensure the convergence of the output layer learning rule. Moreover, the formulation of the derivative of the cross-entropy function with respect to $\boldsymbol{\vartheta}$

is the same with that of the squared loss, i.e.,

$$\frac{0.5(\mathbf{d}_t - \hat{\boldsymbol{d}}_t)^2}{\partial \boldsymbol{\vartheta}} = (\hat{\boldsymbol{d}}_t - \mathbf{d}_t)\boldsymbol{\vartheta} \text{ and } \frac{\partial RE(p_t, \hat{p}_t)}{\partial \boldsymbol{\vartheta}} = (\hat{p}_t - p_t)\boldsymbol{\vartheta},$$

where $\mathbf{d}_t$ and $\hat{\boldsymbol{d}}_t$ are the outputs of the regression model in (3.1)-(3.2). Therefore, the Lipschitz properties derived in Theorem 3.2 applies to the the cross-entropy loss as well. Since Theorem 3.3 uses only the Lipschitz properties, it can be extended for the cross-entropy loss with the same learning rules in (3.14)-(3.15). As a result, Algorithm 3 can be used for the cross-entropy loss without any change.

B.2 Preliminaries for the Proofs

In the proofs, we use $\|\cdot\|_\infty$ for the ℓ_∞ norm. We denote the derivative of tanh as $\tanh'$, where $\tanh'(x) = 1 - \tanh(x)^2$. We denote the elementary row scaling operation with $\odot$, i.e., $\mathbf{x} \odot \mathbf{W} = \text{diag}(\mathbf{x})\mathbf{W}$. Here, $\text{diag}(\mathbf{x}) \in \mathbb{R}^{n \times n}$ is the elementary scaling matrix whose diagonal elements are the components of $\mathbf{x} \in \mathbb{R}^n$.

For the following analysis, we reformulate our network model as

$$\hat{\boldsymbol{d}}_t = \boldsymbol{\vartheta}^T \mathbf{h}_t. \tag{B.1}$$

$$\mathbf{h}_{t+1} = \tanh(\mathbf{W}\mathbf{h}_t + \mathbf{U}\mathbf{x}_{t+1}). \tag{B.2}$$

Moreover, we write the hidden state update in (B.2) with the vectorized weight matrices as

$$\mathbf{h}_{t+1} = \tanh(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}) \tag{B.3}$$

where $\mathbf{H}_t = \mathbf{I} \otimes \mathbf{h}_t^T$, $\mathbf{X}_t = \mathbf{I} \otimes \mathbf{x}_t^T$, and $\otimes$ is the Kronecker product.

B.3 Auxiliary Propositions

Proposition 1. *For any* $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{W} \in \mathbb{R}^{n\times m}$, *where* $n, m \in \mathbb{N}$, *the following statements hold:*

$$1)\ \|\boldsymbol{x} \odot \boldsymbol{W}\| \leq \|\boldsymbol{x}\|_\infty \|\boldsymbol{W}\| \tag{B.4}$$

$$2)\ \|\tanh'(\boldsymbol{x}) - \tanh'(\boldsymbol{y})\|_\infty \leq 2\|\boldsymbol{x} - \boldsymbol{y}\| \tag{B.5}$$

$$3)\ \|\boldsymbol{I} \otimes \boldsymbol{x}^T\| = \|\boldsymbol{x}\|. \tag{B.6}$$

Proof of Proposition 1. 1. Since $\mathbf{x} \odot \mathbf{W} = \mathrm{diag}(\mathbf{x})\mathbf{W}$, we have $\|\mathbf{x} \odot \mathbf{W}\| \leq \|\mathrm{diag}(\mathbf{x})\|\|\mathbf{W}\|$, where we use the Cauchy-Schwarz inequality. Since by definition $\|\mathrm{diag}(\mathbf{x})\| = \|\mathbf{x}\|_\infty$, $\|\mathbf{x} \odot \mathbf{W}\| \leq \|\mathrm{diag}(\mathbf{x})\|\|\mathbf{W}\| = \|\mathbf{x}\|_\infty\|\mathbf{W}\|$.

2. Recall that $\tanh'(x) = 1 - \tanh(x)^2$. Since $\tanh(x) \in [-1, 1]$, tanh is 1-Lipschitz 2-smooth. Then, by using $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|$ for any $\mathbf{x} \in \mathbb{R}^n$, we have $\|\tanh'(\mathbf{x}) - \tanh'(\mathbf{y})\|_\infty \leq \|\tanh'(\mathbf{x}) - \tanh'(\mathbf{y})\|$. Since tanh is 2-smooth, we have $\|\tanh'(\mathbf{x}) - \tanh'(\mathbf{y})\| \leq 2\|\mathbf{x} - \mathbf{y}\|$.

3. See [43, Theorem 8].

□

Proposition 2. *Let* $\boldsymbol{W}$ *and* $\boldsymbol{U}$ *be the hidden-layer weight matrices in (B.2) satisfying* $\|\boldsymbol{W}\| \leq \lambda$, *and* $\|\boldsymbol{U}\| \leq \lambda$ *for some* $\lambda \in \mathbb{R}$. *By using the formulation given in (B.3), the Lipschitz and smoothness properties of the single SRNN iteration*

can be written as:

$$1) \left\| \frac{\partial \tanh(\boldsymbol{H}_t\boldsymbol{\theta} + \boldsymbol{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{h}_t} \right\| \leq \lambda, \tag{B.7}$$

$$2) \left\| \frac{\partial^2 \tanh(\boldsymbol{H}_t\boldsymbol{\theta} + \boldsymbol{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{h}_t^2} \right\| \leq 2\lambda^2, \tag{B.8}$$

$$3) \left\| \frac{\partial^2 \tanh(\boldsymbol{H}_t\boldsymbol{\theta} + \boldsymbol{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{h}_t \partial \boldsymbol{\theta}} \right\| \leq 2\lambda\sqrt{n_h}, \tag{B.9}$$

$$4) \left\| \frac{\partial^2 \tanh(\boldsymbol{H}_t\boldsymbol{\theta} + \boldsymbol{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{h}_t \partial \boldsymbol{\mu}} \right\| \leq 2\lambda\sqrt{n_x}, \tag{B.10}$$

$$5) \left\| \frac{\partial^2 \tanh(\boldsymbol{H}_t\boldsymbol{\theta} + \boldsymbol{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\mu}} \right\| \leq 2\sqrt{n_x}\sqrt{n_h}. \tag{B.11}$$

$$6) \left\| \frac{\partial \tanh(\boldsymbol{H}_t\boldsymbol{\theta} + \boldsymbol{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\theta}} \right\| \leq \sqrt{n_h}, \tag{B.12}$$

$$7) \left\| \frac{\partial \tanh(\boldsymbol{H}_t\boldsymbol{\theta} + \boldsymbol{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\mu}} \right\| \leq \sqrt{n_x}, \tag{B.13}$$

$$8) \left\| \frac{\partial^2 \tanh(\boldsymbol{H}_t\boldsymbol{\theta} + \boldsymbol{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\theta}^2} \right\| \leq 2n_h, \tag{B.14}$$

$$9) \left\| \frac{\partial^2 \tanh(\boldsymbol{H}_t\boldsymbol{\theta} + \boldsymbol{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\mu}^2} \right\| \leq 2n_x. \tag{B.15}$$

Proof of Proposition 2. We prove each statement separately:

1. We note that (B.2) and (B.3) are equivalent. By using (B.4) and $\tanh'(x) \leq 1$ on (B.2), we write

$$\begin{aligned} \left\| \frac{\partial \tanh(\mathbf{W}\mathbf{h}_t + \mathbf{U}\mathbf{x}_{t+1})}{\partial \mathbf{h}_t} \right\| &= \|\tanh'(\mathbf{W}\mathbf{h}_t + \mathbf{U}\mathbf{x}_{t+1}) \odot \mathbf{W}\| \\ &\leq \|\tanh'(\mathbf{W}\mathbf{h}_t + \mathbf{U}\mathbf{x}_{t+1})\|_\infty \|\mathbf{W}\| \\ &\leq \lambda. \end{aligned}$$

2. By using (B.4) and (B.5), we write

$$\begin{aligned} &\left\| \frac{\partial \tanh(\mathbf{W}\mathbf{h}_t + \mathbf{U}\mathbf{x}_{t+1})}{\partial \mathbf{h}_t} - \frac{\partial \tanh(\mathbf{W}'\mathbf{h}_t + \mathbf{U}\mathbf{x}_{t+1})}{\partial \mathbf{h}_t} \right\| \\ &\qquad = \|\tanh'(\mathbf{W}\mathbf{h}_t + \mathbf{U}\mathbf{x}_{t+1}) \odot \mathbf{W} - \tanh'(\mathbf{W}\mathbf{h}'_t + \mathbf{U}\mathbf{x}_{t+1}) \odot \mathbf{W}\| \\ &\qquad \leq \|\tanh'(\mathbf{W}\mathbf{h}_t + \mathbf{U}\mathbf{x}_{t+1}) - \tanh'(\mathbf{W}\mathbf{h}'_t + \mathbf{U}\mathbf{x}_{t+1})\|_\infty \|\mathbf{W}\| \\ &\qquad \leq 2\|\mathbf{W}\|\|\mathbf{h}_t - \mathbf{h}'_t\|\|\mathbf{W}\| \leq 2\lambda^2\|\mathbf{h}_t - \mathbf{h}'_t\|. \end{aligned}$$

3. We note that since tanh is twice differentiable, the order of partial derivatives is not important. Then, by using (B.4), (B.5), (B.6), and $\|\mathbf{h}_t\| \leq \sqrt{n_h}$, we write

$$\begin{aligned}
&\Big\|\frac{\partial \tanh(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\theta}} - \frac{\partial \tanh(\mathbf{H}'_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\theta}}\Big\| \\
&\qquad = \|\tanh'(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}) \odot \mathbf{H}_t - \tanh'(\mathbf{H}'_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}) \odot \mathbf{H}'_t\| \qquad \text{(B.16)} \\
&\qquad \leq \|\tanh'(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}) - \tanh'(\mathbf{H}'_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu})\|_\infty \|\mathbf{H}_t\| \\
&\qquad\quad + \|\tanh'(\mathbf{H}'_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu})\|_\infty \|\mathbf{H}_t - \mathbf{H}'_t\| \qquad \text{(B.17)} \\
&\qquad \leq \|\tanh'(\mathbf{W}\mathbf{h}_t + \mathbf{U}\mathbf{x}_{t+1}) - \tanh'(\mathbf{W}\mathbf{h}'_t + \mathbf{U}\mathbf{x}_{t+1})\|_\infty \|\mathbf{H}_t\| \\
&\qquad\quad + \|\mathbf{h}_t - \mathbf{h}'_t\| \qquad \text{(B.18)} \\
&\qquad \leq \big(2\|\mathbf{W}\|\sqrt{n_h} + 1\big)\|\mathbf{h}_t - \mathbf{h}'_t\| \leq (2\lambda\sqrt{n_h} + 1)\|\mathbf{h}_t - \mathbf{h}'_t\|,
\end{aligned}$$

where we add $\pm \tanh'(\mathbf{H}'_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}) \odot \mathbf{H}_t$ inside of the norm in (B.16), and use the triangle inequality for (B.17). Here, we omit $+1$ term for mathematical convenience in the following derivations.

4. This can be obtained by repeating the steps in the proof of (B.9) for $\boldsymbol{\mu}$ and $\mathbf{h}_t$.

5. By using (B.4), (B.5), (B.6), $\|\mathbf{h}_t\| \leq \sqrt{n_h}$, and $\|\mathbf{x}_t\| \leq \sqrt{n_x}$, we write

$$\begin{aligned}
&\Big\|\frac{\partial \tanh(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\theta}} - \frac{\partial \tanh(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}')}{\partial \boldsymbol{\theta}}\Big\| \\
&\qquad = \|\tanh'(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}) \odot \mathbf{H}_t - \tanh'(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}') \odot \mathbf{H}_t\| \\
&\qquad \leq \|\tanh'(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}) - \tanh'(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}')\|_\infty \|\mathbf{H}_t\| \\
&\qquad \leq 2\|\mathbf{X}_{t+1}\|\|\boldsymbol{\mu} - \boldsymbol{\mu}'\|\|\mathbf{H}_t\| \\
&\qquad \leq 2\sqrt{n_h}\sqrt{n_x}\|\boldsymbol{\mu} - \boldsymbol{\mu}'\|.
\end{aligned}$$

6. By using (B.4), $\tanh'(x) \leq 1$, and $\|\mathbf{h}_t\| \leq \sqrt{n_h}$,

$$\begin{aligned}
\Big\|\frac{\partial \tanh(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\theta}}\Big\| &= \|\tanh'(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}) \odot \mathbf{H}_t\| \\
&\leq \|\tanh'(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu})\|_\infty \|\mathbf{H}_t\| \\
&\leq \sqrt{n_h}.
\end{aligned}$$

7. This can be obtained by repeating the steps in the proof of (B.12) for $\boldsymbol{\mu}$.

8. By using (B.4), (B.5), (B.6), and $\|\mathbf{h}_t\| \leq \sqrt{n_h}$, we write

$$\begin{aligned}
&\Big\| \frac{\partial \tanh(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\theta}} - \frac{\partial \tanh(\mathbf{H}_t\boldsymbol{\theta}' + \mathbf{X}_{t+1}\boldsymbol{\mu})}{\partial \boldsymbol{\theta}} \Big\| \\
&\qquad = \|\tanh'(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}) \odot \mathbf{H}_t - \tanh'(\mathbf{H}_t\boldsymbol{\theta}' + \mathbf{X}_{t+1}\boldsymbol{\mu}) \odot \mathbf{H}_t\| \\
&\qquad \leq \|\tanh'(\mathbf{H}_t\boldsymbol{\theta} + \mathbf{X}_{t+1}\boldsymbol{\mu}) - \tanh'(\mathbf{H}_t\boldsymbol{\theta}' + \mathbf{X}_{t+1}\boldsymbol{\mu})\|_\infty \|\mathbf{H}_t\| \\
&\qquad \leq 2\|\mathbf{H}_t\|\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|\|\mathbf{H}_t\| \\
&\qquad \leq 2n_h\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|.
\end{aligned}$$

9. This can be obtained by repeating the steps in the proof of (B.14) for $\boldsymbol{\mu}$.

□

B.4 Proof of Lemma 3.1

Before the proof, let $\mathbf{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu})$ be the state vector obtained at time t by running the model in (B.2) with the matrices $\mathbf{W}'$, $\mathbf{U}$, input sequence $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t\}$, and the initial condition $\mathbf{h}_1(\boldsymbol{\theta}', \boldsymbol{\mu}) = \mathbf{h}_1(\boldsymbol{\theta}, \boldsymbol{\mu}) = \mathbf{h}_1(\boldsymbol{\theta}', \boldsymbol{\mu}')$. Then,

$$\|\mathbf{h}_t(\boldsymbol{\theta}, \boldsymbol{\mu}) - \mathbf{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu}')\| \leq \|\mathbf{h}_t(\boldsymbol{\theta}, \boldsymbol{\mu}) - \mathbf{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu})\| + \|\mathbf{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu}) - \mathbf{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu}')\|, \tag{B.19}$$

where we add $\pm\mathbf{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu})$ inside of the norm and use the triangle inequality.

We will bound the terms in (B.19) separately. We begin with the first term.

Since $\mathbf{h}_t(\boldsymbol{\theta}, \boldsymbol{\mu})$ and $\mathbf{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu})$ include the same $\boldsymbol{\mu}$, in the following (between (B.20)-(B.23)), we abbreviate them as $\mathbf{h}_t(\boldsymbol{\theta})$ and $\mathbf{h}_t(\boldsymbol{\theta}')$:

$$\|\mathbf{h}_t(\boldsymbol{\theta}) - \mathbf{h}_t(\boldsymbol{\theta}')\| = \|\tanh(\mathbf{W}\mathbf{h}_{t-1}(\boldsymbol{\theta}) + \mathbf{U}\mathbf{x}_t) - \tanh(\mathbf{W}'\mathbf{h}_{t-1}(\boldsymbol{\theta}') + \mathbf{U}\mathbf{x}_t)\| \tag{B.20}$$

$$\leq \|\tanh(\mathbf{W}\mathbf{h}_{t-1}(\boldsymbol{\theta}) + \mathbf{U}\mathbf{x}_t) - \tanh(\mathbf{W}\mathbf{h}_{t-1}(\boldsymbol{\theta}') + \mathbf{U}\mathbf{x}_t)\| + \|\tanh(\mathbf{W}\mathbf{h}_{t-1}(\boldsymbol{\theta}') + \mathbf{U}\mathbf{x}_t) - \tanh(\mathbf{W}'\mathbf{h}_{t-1}(\boldsymbol{\theta}') + \mathbf{U}\mathbf{x}_t)\| \tag{B.21}$$

$$\leq \lambda\|\mathbf{h}_{t-1}(\boldsymbol{\theta}) - \mathbf{h}_{t-1}(\boldsymbol{\theta}')\| + \sqrt{n_h}\|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \tag{B.22}$$

$$\leq \sum_{i=0}^{t-1} \left(\lambda^i \sqrt{n_h}\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|\right) \tag{B.23}$$

Here, to obtain (B.21), we add $\pm\tanh(\mathbf{W}\mathbf{h}_{t-1}(\boldsymbol{\theta}') + \mathbf{U}\mathbf{x}_t)$ inside of the norm in (B.20), and use the triangle inequality. Then, we use (B.7) and (B.12) to get (B.22). Until we reach (B.23), we repeatedly apply the same bounding technique to bound the norm of the differences between the state vectors.

Now, we bound the second term in (B.19). Since $\mathbf{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu})$ and $\mathbf{h}_t(\boldsymbol{\theta}', \boldsymbol{\mu}')$ include the same $\boldsymbol{\theta}'$, in the following (between (B.24)-(B.27)), we abbreviate them as $\mathbf{h}_t(\boldsymbol{\mu})$ and $\mathbf{h}_t(\boldsymbol{\mu}')$:

$$\|\mathbf{h}_t(\boldsymbol{\mu}) - \mathbf{h}_t(\boldsymbol{\mu}')\| = \|\tanh(\mathbf{W}'\mathbf{h}_{t-1}(\boldsymbol{\mu}) + \mathbf{U}\mathbf{x}_t) - \tanh(\mathbf{W}'\mathbf{h}_{t-1}(\boldsymbol{\mu}') + \mathbf{U}'\mathbf{x}_t)\| \tag{B.24}$$

$$\leq \|\tanh(\mathbf{W}'\mathbf{h}_{t-1}(\boldsymbol{\mu}) + \mathbf{U}\mathbf{x}_t) - \tanh(\mathbf{W}'\mathbf{h}_{t-1}(\boldsymbol{\mu}') + \mathbf{U}\mathbf{x}_t)\| + \|\tanh(\mathbf{W}'\mathbf{h}_{t-1}(\boldsymbol{\mu}') + \mathbf{U}\mathbf{x}_t) - \tanh(\mathbf{W}'\mathbf{h}_{t-1}(\boldsymbol{\mu}') + \mathbf{U}'\mathbf{x}_t)\| \tag{B.25}$$

$$\leq \lambda\|\mathbf{h}_{t-1}(\boldsymbol{\mu}) - \mathbf{h}_{t-1}(\boldsymbol{\mu}')\| + \sqrt{n_x}\|\boldsymbol{\mu} - \boldsymbol{\mu}'\| \tag{B.26}$$

$$\leq \sum_{i=0}^{t-1} \left(\lambda^i \sqrt{n_x}\|\boldsymbol{\mu} - \boldsymbol{\mu}'\|\right), \tag{B.27}$$

where for (B.25), we add $\pm\tanh(\mathbf{W}'\mathbf{h}_{t-1}(\boldsymbol{\mu}') + \mathbf{U}\mathbf{x}_t)$ and use the triangle inequality. We, then, use (B.7) and (B.13) to get (B.26). Until we reach (B.27), we repeatedly apply the same technique to bound the norm of the differences between state vectors. In the end, we use (B.23) and (B.27) to bound (B.19), which

yields the statement in the lemma.

B.5 Proof of Theorem 3.2

Recall that

$$L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu}) = \frac{1}{w} \sum_{i=0}^{w-1} \ell_{t-i}(\boldsymbol{\theta}, \boldsymbol{\mu}).$$

Then, if we bound the derivative of $\ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})$ for an arbitrary $t \in [T]$, the resulting bound will be valid for $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$ as well. Therefore, here, we analyze the Lipschitz properties of $\ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})$ for an arbitrary $t \in [T]$ and extend the result to $L_{t,w}(\boldsymbol{\theta}, \boldsymbol{\mu})$. In the following, we prove each statement of the theorem separately.

1. Let us use $\mathbf{h}_t$ and $\mathbf{h}'_t$ for the state vectors obtained by running the model in (B.2) from the initial step up to current time step t with the same initial condition, same input layer matrix $\boldsymbol{\mu}$, common input sequence $\{\mathbf{x}_1, \cdots, \mathbf{x}_t\}$ but different $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, respectively. Let us also say $\ell_t(\boldsymbol{\theta}', \boldsymbol{\mu}) = 0.5(\mathbf{d}_t - \hat{\boldsymbol{d}}'_t)^2$, where $\hat{\boldsymbol{d}}'_t$ is the prediction of the second model producing $\mathbf{h}'_t$. Then,

$$\begin{aligned}
&\Big\| \frac{\partial \ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})}{\partial \boldsymbol{\theta}} - \frac{\partial \ell_t(\boldsymbol{\theta}', \boldsymbol{\mu})}{\partial \boldsymbol{\theta}} \Big\| \\
&\qquad = \Big\| (\mathbf{d}_t - \hat{\boldsymbol{d}}'_t) \boldsymbol{\vartheta}^T \Big(\sum_{\tau=1}^{t} \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}} \Big) - (\mathbf{d}_t - \hat{\boldsymbol{d}}_t) \boldsymbol{\vartheta}^T \Big(\sum_{\tau=1}^{t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} \frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\theta}} \Big) \Big\| && \text{(B.28)} \\
&\qquad \leq 2\sqrt{n_h} \sum_{\tau=1}^{t} \Big\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} \frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\theta}} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}} \Big\| && \text{(B.29)} \\
&\qquad \leq 2\sqrt{n_h} \sum_{\tau=1}^{t} \Big(\Big\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} \frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\theta}} - \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}} \Big\| + \Big\| \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}} \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \Big\| \Big) && \text{(B.30)} \\
&\qquad \leq 2\sqrt{n_h} \sum_{\tau=1}^{t} \Big(\Big\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} \Big\| \Big\| \frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\theta}} - \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}} \Big\| + \Big\| \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}} \Big\| \Big\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \Big\| \Big) && \text{(B.31)} \\
&\qquad \leq 2\sqrt{n_h} \sum_{\tau=1}^{t} \lambda^{t-\tau} \Big\| \frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\theta}} - \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}} \Big\| + 2n_h \sum_{\tau=1}^{t} \Big\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \Big\|. && \text{(B.32)}
\end{aligned}$$

Here, we use the bounds of $\mathbf{d}_t$, $\hat{\boldsymbol{d}}_t$ and $\boldsymbol{\vartheta}$ for (B.28).[1] To get (B.30), we add $\pm\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau}\frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}}$ inside the norm of (B.29), and use the triangle inequality. To get (B.32), we use (B.7) and (B.12).

In the following, we will bound the terms in (B.32) separately. We begin with the first term. Note that $\mathbf{h}_\tau = \tanh(\mathbf{W}\mathbf{h}_{\tau-\,1}+\mathbf{U}\mathbf{x}_\tau)$, and $\mathbf{h}'_\tau = \tanh(\mathbf{W}'\mathbf{h}'_{\tau-\,1}+\mathbf{U}\mathbf{x}_\tau)$. Then,

$$\begin{aligned}
\sum_{\tau=1}^{t}\lambda^{t-\tau}\Big\|\frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\theta}}-\frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}}\Big\| &\le \sum_{\tau=1}^{t}\lambda^{t-\tau}\Big\|\frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\theta}}-\frac{\partial \tanh(\mathbf{W}\mathbf{h}'_{\tau-\,1}+\mathbf{U}\mathbf{x}_{\tau-1})}{\partial \boldsymbol{\theta}}\Big\| \\
&\quad+\sum_{\tau=1}^{t}\lambda^{t-\tau}\Big\|\frac{\partial \tanh(\mathbf{W}\mathbf{h}'_{\tau-\,1}+\mathbf{U}\mathbf{x}_{\tau-1})}{\partial \boldsymbol{\theta}}-\frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}}\Big\| \qquad \text{(B.33)}\\
&\le \sum_{\tau=1}^{t}\lambda^{t-\tau}\big(2\lambda\sqrt{n_h}\|\mathbf{h}_{\tau-\,1}-\mathbf{h}'_{\tau-\,1}\|+2n_h\|\boldsymbol{\theta}-\boldsymbol{\theta}'\|\big) \qquad \text{(B.34)}\\
&\le \Big(\sum_{\tau=1}^{t}\lambda^{t-\tau}\Big)\Big(\frac{2\lambda n_h}{1-\lambda}+2n_h\Big)\|\boldsymbol{\theta}-\boldsymbol{\theta}'\| \qquad \text{(B.35)}\\
&\le \frac{2n_h}{1-\lambda}\Big(\frac{\lambda}{1-\lambda}+1\Big)\|\boldsymbol{\theta}-\boldsymbol{\theta}'\|, \qquad \text{(B.36)}
\end{aligned}$$

Here, to get (B.33), we add $\pm\frac{\partial}{\partial \boldsymbol{\theta}}\tanh(\mathbf{W}\mathbf{h}'_{\tau-\,1}+\mathbf{U}\mathbf{x}_{\tau-1})$ inside of the term in the left hand side and use the triangle inequality. We use (B.9) and (B.14) for (B.34). We use Lemma 3.1 and $\lambda\in[0,1)$ for (B.35).

Now, we bound the second term in (B.32). To bound the sum, we first focus on

[1]Note that since $\mathbf{d}_t, \hat{\boldsymbol{d}}_t \in [-\sqrt{n_h}, \sqrt{n_h}]$ and $\|\boldsymbol{\vartheta}_t\| \le 1$, the ℓ_2 norm of $(\mathbf{d}_t - \hat{\boldsymbol{d}}_t)\boldsymbol{\vartheta}$ is bounded by $2\sqrt{n_h}$, i.e., $\|(\mathbf{d}_t - \hat{\boldsymbol{d}}_t)\boldsymbol{\vartheta}\| \le 2\sqrt{n_h}$.

the individual terms, i.e., $\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \right\|$:

$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \right\| \leq \left\| \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_\tau} \right\| \left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_{t-1}} \right\| + \left\| \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_{t-1}} \right\| \left\| \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_{t-1}}{\partial \mathbf{h}_\tau} \right\| \tag{B.37}$$

$$\leq \lambda^{t-\tau-1} \big(2\lambda^2 \|\mathbf{h}_{t-1} - \mathbf{h}'_{t-1}\| + 2\lambda \sqrt{n_h} \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \big) + \lambda \left\| \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_{t-1}}{\partial \mathbf{h}_\tau} \right\| \tag{B.38}$$

$$\leq \lambda^{t-\tau-1} \Big(\frac{2\lambda^2 \sqrt{n_h}}{1-\lambda} + 2\lambda \sqrt{n_h} \Big) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| + \lambda \left\| \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_{t-1}}{\partial \mathbf{h}_\tau} \right\| \tag{B.39}$$

$$\leq (t-\tau) \lambda^{t-\tau-1} \Big(\frac{2\lambda^2 \sqrt{n_h}}{1-\lambda} + 2\lambda \sqrt{n_h} \Big) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|, \tag{B.40}$$

where we add $\pm \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_{t-1}} \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_\tau}$, and use the triangle inequality for (B.37). We use (B.8) and (B.9) for (B.38). We use Lemma 3.1 and $\lambda \in [0, 1)$ for (B.39). We, then, repeat the same manipulations in (B.37)-(B.39) to bound the terms with partial derivatives.

Then, the second term in (B.32) can be bound as:

$$\sum_{\tau=1}^{t} \left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \right\| \leq \sum_{\tau=1}^{t} (t-\tau) \lambda^{t-\tau-1} \Big(\frac{2\lambda^2 \sqrt{n_h}}{1-\lambda} + 2\lambda \sqrt{n_h} \Big) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \tag{B.41}$$

$$= \frac{2\sqrt{n_h}}{1-\lambda} \Big(\frac{\lambda^2}{(1-\lambda)^2} + \frac{\lambda}{1-\lambda} \Big) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|. \tag{B.42}$$

Here, to get (B.42), we use the fact that

$$\sum_{\tau=1}^{t} (t-\tau) \lambda^{t-\tau-1} \leq \frac{1}{(1-\lambda)^2} \text{ for any } t \in \mathbb{N}.$$

Then, by using (B.36), (B.42), and (B.32), we can write

$$\left\| \frac{\partial \ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})}{\partial \boldsymbol{\theta}} - \frac{\partial \ell_t(\boldsymbol{\theta}', \boldsymbol{\mu})}{\partial \boldsymbol{\theta}} \right\| \leq 2\sqrt{n_h} \sum_{\tau=1}^{t} \lambda^{t-\tau} \left\| \frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\theta}} - \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\theta}} \right\| + 2n_h \sum_{\tau=1}^{t} \left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \right\| \tag{B.43}$$

$$\leq \frac{4n_h \sqrt{n_h}}{1-\lambda} \Big(\frac{\lambda^2}{(1-\lambda)^2} + \frac{2\lambda}{1-\lambda} + 1 \Big) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \tag{B.44}$$

$$= \frac{4n_h \sqrt{n_h}}{(1-\lambda)^3} \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|. \tag{B.45}$$

By realizing that (B.45) holds for an arbitrary t, the statement in the theorem can be obtained.

2. This part can be obtained by adapting the steps in the previous proof for $\boldsymbol{\mu}$, and use the Lipschitz conditions in (B.10), (B.13), and (B.15), accordingly.

3. We use the same notation as in the proof of the 1st statement. Then,

$$\Big\|\frac{\partial \ell_t(\boldsymbol{\theta},\boldsymbol{\mu})}{\partial \boldsymbol{\mu}} - \frac{\partial \ell_t(\boldsymbol{\theta}',\boldsymbol{\mu})}{\partial \boldsymbol{\mu}}\Big\|$$

$$= \Big\|(\mathbf{d}_t - \hat{\boldsymbol{d}}_t')\boldsymbol{\vartheta}^T\Big(\sum_{\tau=1}^{t}\frac{\partial \mathbf{h}_t'}{\partial \mathbf{h}_\tau'}\frac{\partial \mathbf{h}_\tau'}{\partial \boldsymbol{\mu}}\Big) - (\mathbf{d}_t - \hat{\boldsymbol{d}}_t)\boldsymbol{\vartheta}^T\Big(\sum_{\tau=1}^{t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau}\frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\mu}}\Big)\Big\| \tag{B.46}$$

$$\leq 2\sqrt{n_h}\sum_{\tau=1}^{t}\Big\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau}\frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\mu}} - \frac{\partial \mathbf{h}_t'}{\partial \mathbf{h}_\tau'}\frac{\partial \mathbf{h}_\tau'}{\partial \boldsymbol{\mu}}\Big\| \tag{B.47}$$

$$\leq 2\sqrt{n_h}\sum_{\tau=1}^{t}\Big(\Big\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau}\frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\mu}} - \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau}\frac{\partial \mathbf{h}_\tau'}{\partial \boldsymbol{\mu}}\Big\| + \Big\|\frac{\partial \mathbf{h}_\tau'}{\partial \boldsymbol{\mu}}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}_\tau'}{\partial \boldsymbol{\mu}}\frac{\partial \mathbf{h}_t'}{\partial \mathbf{h}_\tau'}\Big\|\Big) \tag{B.48}$$

$$\leq 2\sqrt{n_h}\sum_{\tau=1}^{t}\Big(\Big\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau}\Big\|\Big\|\frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\mu}} - \frac{\partial \mathbf{h}_\tau'}{\partial \boldsymbol{\mu}}\Big\| + \Big\|\frac{\partial \mathbf{h}_\tau'}{\partial \boldsymbol{\mu}}\Big\|\Big\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}_t'}{\partial \mathbf{h}_\tau'}\Big\|\Big) \tag{B.49}$$

$$\leq 2\sqrt{n_h}\sum_{\tau=1}^{t}\lambda^{t-\tau}\Big\|\frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\mu}} - \frac{\partial \mathbf{h}_\tau'}{\partial \boldsymbol{\mu}}\Big\| + 2\sqrt{n_h n_x}\sum_{\tau=1}^{t}\Big\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}_t'}{\partial \mathbf{h}_\tau'}\Big\|. \tag{B.50}$$

Here, to get (B.48), we add $\pm\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau}\frac{\partial \mathbf{h}_\tau'}{\partial \boldsymbol{\mu}}$ inside the norm in (B.47), and use the triangle inequality. To get (B.50), we use (B.7) and (B.13).

We bound the terms in (B.50) separately. We begin with the first term. Note

that $\mathbf{h}_\tau = \tanh(\mathbf{W}\mathbf{h}_{\tau-1} + \mathbf{U}\mathbf{x}_\tau)$, and $\mathbf{h}'_\tau = \tanh(\mathbf{W}'\mathbf{h}'_{\tau-1} + \mathbf{U}\mathbf{x}_\tau)$. Then,

$$\sum_{\tau=1}^{t} \lambda^{t-\tau} \Big\| \frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\mu}} - \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\mu}} \Big\| \leq \sum_{\tau=1}^{t} \lambda^{t-\tau} \Big\| \frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\mu}} - \frac{\partial \tanh(\mathbf{W}\mathbf{h}'_{\tau-1} + \mathbf{U}\mathbf{x}_{\tau-1})}{\partial \boldsymbol{\theta}} \Big\|$$
$$+ \sum_{\tau=1}^{t} \lambda^{t-\tau} \Big\| \frac{\partial \tanh(\mathbf{W}\mathbf{h}'_{\tau-1} + \mathbf{U}\mathbf{x}_{\tau-1})}{\partial \boldsymbol{\theta}} - \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\mu}} \Big\| \quad \text{(B.51)}$$
$$\leq \sum_{\tau=1}^{t} \lambda^{t-\tau} \big(2\lambda \sqrt{n_x} \|\mathbf{h}_{\tau-1} - \mathbf{h}'_{\tau-1}\| + 2\sqrt{n_x n_h} \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \big) \quad \text{(B.52)}$$
$$\leq \Big(\sum_{\tau=1}^{t} \lambda^{t-\tau} \Big) \Big(\frac{2\lambda \sqrt{n_x n_h}}{1-\lambda} + 2\sqrt{n_x n_h} \Big) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \quad \text{(B.53)}$$
$$\leq \frac{2\sqrt{n_h n_x}}{1-\lambda} \Big(\frac{\lambda}{1-\lambda} + 1 \Big) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|, \quad \text{(B.54)}$$

where to get (B.51), we add $\pm \frac{\partial}{\partial \boldsymbol{\theta}} \tanh(\mathbf{W}\mathbf{h}'_{\tau-1} + \mathbf{U}\mathbf{x}_{\tau-1})$ inside of the term in the left hand side and use the triangle inequality,. We use (B.10) and (B.11) for (B.52). We use Lemma 3.1 and $\lambda \in [0, 1)$ for (B.53).

Now, we bound the second term in (B.50):

$$\sum_{\tau=1}^{t} \Big\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \Big\| \leq \sum_{\tau=1}^{t} (t-\tau) \lambda^{t-\tau-1} \Big(\frac{2\lambda^2 \sqrt{n_h}}{1-\lambda} + 2\lambda \sqrt{n_h} \Big) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \quad \text{(B.55)}$$
$$= \frac{2\sqrt{n_h}}{1-\lambda} \Big(\frac{\lambda^2}{(1-\lambda)^2} + \frac{\lambda}{1-\lambda} \Big) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|, \quad \text{(B.56)}$$

where we use (B.40) to bound the terms $\Big\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \Big\|$.

Then, by using (B.54), (B.56), and (B.50), we can write

$$\Big\| \frac{\partial \ell_t(\boldsymbol{\theta}, \boldsymbol{\mu})}{\partial \boldsymbol{\mu}} - \frac{\partial \ell_t(\boldsymbol{\theta}', \boldsymbol{\mu})}{\partial \boldsymbol{\mu}} \Big\| \leq 2\sqrt{n_h} \sum_{\tau=1}^{t} \lambda^{t-\tau} \Big\| \frac{\partial \mathbf{h}_\tau}{\partial \boldsymbol{\mu}} - \frac{\partial \mathbf{h}'_\tau}{\partial \boldsymbol{\mu}} \Big\| + 2\sqrt{n_h n_x} \sum_{\tau=1}^{t} \Big\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} - \frac{\partial \mathbf{h}'_t}{\partial \mathbf{h}'_\tau} \Big\|$$
$$\leq \frac{4n_h \sqrt{n_x}}{1-\lambda} \Big(\frac{\lambda^2}{(1-\lambda)^2} + \frac{2\lambda}{1-\lambda} + 1 \Big) \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|$$
$$= \frac{4n_h \sqrt{n_x}}{(1-\lambda)^3} \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|. \quad \text{(B.57)}$$

By realizing that (B.57) holds for an arbitrary t, the statement in the theorem can be obtained.

B.6 Proof of Theorem 3.3

In the following, we use $\langle \cdot, \cdot \rangle$ to denote the inner product. Due to the space constraints, we omit the arguments in the partial derivative terms, i.e.,

$$\frac{\partial L_{t,w}}{\partial \boldsymbol{\theta}} \coloneqq \frac{\partial L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}{\partial \boldsymbol{\theta}}, \qquad \frac{\partial_{\mathcal{K}_\theta} L_{t,w}}{\partial \boldsymbol{\theta}} \coloneqq \frac{\partial_{\mathcal{K}_\theta} L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}{\partial \boldsymbol{\theta}},$$
$$\frac{\partial L_{t,w}}{\partial \boldsymbol{\mu}} \coloneqq \frac{\partial L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}{\partial \boldsymbol{\mu}}, \qquad \frac{\partial_{\mathcal{K}_\mu} L_{t,w}}{\partial \boldsymbol{\mu}} \coloneqq \frac{\partial_{\mathcal{K}_\mu} L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}_t)}{\partial \boldsymbol{\mu}}.$$

In the following, we bound the sums $\sum_{t=1}^{T} \left\| \frac{\partial_{\mathcal{K}_\theta} L_{t,w}}{\partial \boldsymbol{\theta}} \right\|^2$ and $\sum_{t=1}^{T} \left\| \frac{\partial_{\mathcal{K}_\mu} L_{t,w}}{\partial \boldsymbol{\mu}} \right\|^2$ separately. To bound the first sum, we first fix an arbitrary $\boldsymbol{\mu} \in \mathcal{K}_\mu$ and derive an upper-bound for the difference $L_{t,w}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\mu}) - L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu})$, i.e.,

$$\begin{aligned}
L_{t,w}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\mu}) - L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}) &\leq \Big\langle \frac{\partial L_{t,w}}{\partial \boldsymbol{\theta}}, \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t \Big\rangle + \frac{\beta_\theta}{2} \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|^2 && \text{(B.58)} \\
&\leq -\eta \Big\langle \frac{\partial L_{t,w}}{\partial \boldsymbol{\theta}}, \frac{\partial_{\mathcal{K}_\theta} L_{t,w}}{\partial \boldsymbol{\theta}} \Big\rangle + \frac{\beta \eta^2}{2} \Big\| \frac{\partial_{\mathcal{K}_\theta} L_{t,w}}{\partial \boldsymbol{\theta}} \Big\|^2 && \text{(B.59)} \\
&\leq -\eta \Big\| \frac{\partial_{\mathcal{K}_\theta} L_{t,w}}{\partial \boldsymbol{\theta}} \Big\|^2 + \frac{\beta \eta^2}{2} \Big\| \frac{\partial_{\mathcal{K}_\theta} L_{t,w}}{\partial \boldsymbol{\theta}} \Big\|^2 && \text{(B.60)} \\
&= -\big(\eta - \frac{\beta \eta^2}{2}\big) \Big\| \frac{\partial_{\mathcal{K}_\theta} L_{t,w}}{\partial \boldsymbol{\theta}} \Big\|^2 && \text{(B.61)}
\end{aligned}$$

where we use [44, Lemma 3.4] for (B.58), Theorem 3.2 for (B.59), and [4, Lemma 3.2] for (B.60).

Then, we swap the left and right hand sides of (B.61), add $\pm L_{t+1,w}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\mu})$ to the right hand side and upper-bound the terms, i.e.,

$$\begin{aligned}
\big(\eta - \frac{\beta \eta^2}{2}\big) \Big\| \frac{\partial_{\mathcal{K}_\theta} L_{t,w}}{\partial \boldsymbol{\theta}} \Big\|^2 &\leq L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}) - L_{t+1,w}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\mu}) \\
&\quad + L_{t+1,w}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\mu}) - L_{t,w}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\mu}) && \text{(B.62)} \\
&\leq L_{t,w}(\boldsymbol{\theta}_t, \boldsymbol{\mu}) - L_{t+1,w}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\mu}) + \frac{4\sqrt{n_h}}{w}, && \text{(B.63)}
\end{aligned}$$

where we use $L_{t,w} \in [-\sqrt{n_h}, \sqrt{n_h}]$ for (B.63).

Next, we sum both sides of (B.63) over T and get

$$\begin{aligned}
\big(\eta - \frac{\beta \eta^2}{2}\big) \sum_{t=1}^{T} \Big\| \frac{\partial_{\mathcal{K}_\theta} L_{t,w}}{\partial \boldsymbol{\theta}} \Big\|^2 &\leq L_{1,w}(\boldsymbol{\theta}_1, \boldsymbol{\mu}) - L_{T+1,w}(\boldsymbol{\theta}_{T+1}) + \frac{4\sqrt{n_h} T}{w} \\
&\leq 4\sqrt{n_h} + \frac{4\sqrt{n_h} T}{w}. && \text{(B.64)}
\end{aligned}$$

By realizing that $\left(\eta - \frac{\beta\eta^2}{2}\right) > \frac{\eta}{2}$ for $0 < \eta \leq 1/\beta$, we simplify (B.64) as

$$\sum_{t=1}^{T} \left\| \frac{\partial_{\mathcal{K}_\theta} L_{t,w}}{\partial \boldsymbol{\theta}} \right\|^2 \leq \frac{8\sqrt{n_h}}{\eta} \frac{T}{w} + \frac{8\sqrt{n_h}}{\eta}. \tag{B.65}$$

We note that we can bound $\sum_{t=1}^{T} \left\| \frac{\partial_{\mathcal{K}_\mu} L_{t,w}}{\partial \boldsymbol{\mu}} \right\|^2$ by fixing $\boldsymbol{\theta} \in \mathcal{K}_\theta$ and adapting steps between (B.58)-(B.64) for $\boldsymbol{\mu}$ accordingly. In the end, we get the same bound, i.e.,

$$\sum_{t=1}^{T} \left\| \frac{\partial_{\mathcal{K}_\mu} L_{t,w}}{\partial \boldsymbol{\mu}} \right\|^2 \leq \frac{8\sqrt{n_h}}{\eta} \frac{T}{w} + \frac{8\sqrt{n_h}}{\eta}. \tag{B.66}$$

By summing the inequalities in (B.65) and (B.66), we get

$$R_w(T) \leq \frac{16\sqrt{n_h}}{\eta} \frac{T}{w} + \frac{16\sqrt{n_h}}{\eta}. \tag{B.67}$$