Non-Uniformly Sampled Data Processing Using LSTM Networks

S. Onur Sahin and Suleyman S. Kozat, *Senior Member, IEEE*

***Abstract*—We investigate classification and regression for non-uniformly sampled variable length sequential data and introduce a novel Long Short-Term Memory (LSTM) architecture. In particular, we extend the classical LSTM network with additional time gates, which incorporate the time information as a nonlinear scaling factor on the conventional gates. We also provide forward pass and backward pass update equations for the proposed LSTM architecture. We show that our approach is superior to the classical LSTM architecture, when there is correlation between time samples. In our experiments, we achieve significant performance gains with respect to the classical LSTM and Phased-LSTM (PLSTM) architectures. In this sense, the proposed LSTM architecture is highly appealing for the applications involving non-uniformly sampled sequential data.**

***Index Terms*—Non-uniform Sampling, Long Short-Term Memory, Recurrent Neural Networks, Supervised Learning, Classification, Regression**

I. Introduction

A. Preliminaries

We study classification and regression of non-uniformly sampled variable length data sequences, where we sequentially receive a non-uniformly sampled data sequence and estimate an unknown desired signal related to this sequence. In the classical data processing applications, data sequences are usually assumed to be uniformly sampled, however, this is not the case in many real life applications. For example, non-uniform sampling is used in many medical imaging applications [1], measurements in astronomy due to day and night conditions [2] and financial data [3], where the stock market values are redetermined by each transaction. Although non-uniformly sampled data frequently arises in these problems, there exist a few studies on non-uniformly sampled sequential data processing in neural networks [4], [5], machine learning [6] and signal processing literatures [7], [8]. Nonlinear approaches are usually used in these studies since linear approaches are usually incapable of capturing highly complex underlying structures [9]. Here, we study classification and regression problems particularly for non-uniformly sampled variable length data sequences in a supervised framework. We sequentially receive a data sequence with the corresponding desired data or signal and we find a nonlinear relation between them.

Even though there exist several nonlinear modeling approaches to process the sequential data [10], [9], neural network based methods are more practical in general because of their capability of modeling highly nonlinear and complex underlying relations [11]. Especially, recurrent neural networks (RNNs) are employed to process sequential data since they are able to identify sequential patterns and learn temporal behaviour, thanks to their internal memory exploiting past information. Although simple RNNs improve the performance in sequential processing tasks, they fail to capture long term dependencies due to vanishing and exploding gradient problems [12]. The LSTM networks are introduced as a special class of the RNNs to remedy these vanishing and exploding gradient problems and capture the long term dependencies [12]. The LSTM networks provide performance gains with their gating mechanisms, which control the amount of the information entering the network and the past information stored in the memory [11].

Even though the classical LSTM networks have satisfactory performance in the applications using uniformly sampled sequential data, they usually perform poorly in the case of non-uniformly sampled data [13], [5]. To circumvent this issue, one can convert non-uniformly sampled data to uniformly sampled data by employing a preprocessing technique, e.g., [13], [14]. However, such approaches result in computational load and provide restricted performance [15].

In this paper, we resolve these problems by introducing a sequential nonlinear learning algorithm based on the LSTM network, which is extended with additional gates incorporating the time information. Our structure provides additional time dependent control while keeping the computational load in the same level. Through extensive set of simulations, we demonstrate significant performance improvements compared to the state of the art architectures in several different regression and classification tasks.

B. Prior Art and Comparisons

RNN based learning methods are extensively used in processing sequential data and modeling time series [16], [17], [18]. Especially complex RNNs, e.g., LSTM networks, have a satisfactory performance thanks to their memory capabilities to exploit past information and gating mechanism to control the flow of the information entering the network. However, this performance of the LSTM networks depends on how the data is sampled, i.e., uniform or non-uniform sampling, existence of missing samples and time intervals between the samples change the performance of the network [5]. In [19], [20], which provide VLSI RNNs with continuous-time dynamical systems approaches, the authors state their networks require

This work is supported by the Turkish Academy of Sciences Outstanding Researcher Programme.

The authors are with the Department of Electrical and Electronics Engineering, Bilkent University, Bilkent, Ankara 06800, Turkey, Tel: +90 (312) 290-2336, Fax: +90 (312) 290-1223, (contact e-mail: ssahin@ee.bilkent.edu.tr, kozat@ee.bilkent.edu.tr)

uniformly sampled data and have limitations in the case of non-uniformly sampled data. Moreover, in [21], [22], it has been shown that the size of the time intervals between samples are important and convey significant information for many sequential data processing tasks such as rhythm detection and motor control.

Among few proposed solutions for processing non-uniformly sampled sequential data, [13] and [14] first convert non-uniformly sampled data to uniformly sampled data by windowing and averaging the samples on the large intervals. Then, they process this uniformly sampled data using the LSTM network. In this setup, they still feed the LSTM network with uniformly sampled data, which is obtained after preprocessing the non-uniformly sampled data. These windowing and averaging operations cause information loss in data entering the LSTM network. As an example, this preprocessing may cause failure in the corresponding tasks, where the aim is to detect whether a value is greater than a certain threshold or not, since averaging smooths the peaks. Furthermore, they also lose the time information contained in the sampling intervals instead of incorporating this information to the network. On the contrary, our LSTM network uses the whole sequence to generate the output, therefore, it exploits all information in the sequence. In addition, it also incorporates the time information to capture the relationship between the underlying model and the sampling times.

The sampling time information should be used in the network to enhance the performance in the applications using non-uniformly sampled data [5]. One can add the time intervals between consecutive data samples to the input vectors as another feature [5]. However, extending the input vector by the time differences contributes only as an additive term with a constant linear scaling deeming this approach as insufficient to model the effect of non-uniform sampling as we demonstrate in this paper. On the contrary, in our LSTM architecture the time differences appear as an adaptive nonlinear scaling factor on the conventional gates of the classical LSTM architecture, which sufficiently model the effect of the non-uniform sampling.

In [5], the authors provide a new LSTM architecture, namely, the PLSTM architecture, which basically learns a periodic sampler function and responds to only a small portion of the input sequence, which is sampled by this function. The sampler function is described by three parameters: period, shift and on-ratio. In each period, the network is updated by only the samples corresponding to its open phase, where on-ratio is ratio of open phase to the period, and shift is the initial time of the open phase. Processing only a small portion of the data accelerates the learning process and provides capability to work on non-uniformly sampled data by incorporating the time information. Although the PLSTM network performs better compared to the classical LSTM network in the classification tasks using non-uniformly sampled data, our approach has two significant contributions over this. Firstly, an important amount of information is lost since the PLSTM architecture processes only a small percentage of the data sequence corresponding to its open phase, where we use the whole sequence. Secondly, the PLSTM network generates the output only at the end of the sequence, therefore, in the vanilla form, it can only be used for the sequential data processing tasks requiring only one output for the whole sequence. On the other hand, our LSTM architecture can generate the output at each time step as well as the end of the sequence, hence, it can also be employed in the tasks such as time series prediction and online regression.

We emphasize that the conventional LSTM based methods [13], [5], [11], [14] are inadequate to process non-uniformly sampled sequential data since they suffer from certain restrictions such as loss in the information exploited by the network. Furthermore, [13] and [14] lose the time information in the preprocessing step due to windowing and averaging operations instead of incorporating it. [5] has restricted application areas since it can generate output only at the end of the sequence. In this paper, we employ a novel LSTM network, which is extended with additional time gates, for classification and sequential regression tasks. These time gates incorporate the time information into the network as an adaptive nonlinear scaling factor on the conventional gates. Since we use the whole data sequence there is no loss in the incoming information unlike [13], [14], [5]. Moreover, our LSTM architecture can generate output at each time step unlike PLSTM, hence, it has a wide range of application areas from sequence labelling to online regression.

C. Contributions

Our contributions are as follows.

1) We introduce a novel LSTM network architecture for processing non-uniformly sampled sequential data, where for the first time in the literature we incorporate the time information as a nonlinear scaling factor using additional time gates.
2) We show that the sampling intervals have a scaling effect on the conventional gates of the classical LSTM architecture. To show this, we first model non-uniform sampling with the missing input case and then extend it to the arbitrary non-uniform sampling case.
3) Our architecture can generate output at each time step as well as at the end of the input sequence unlike the PLSTM network. Therefore, our LSTM architecture has a wide range of application areas from online regression to sequence labelling.
4) Our architecture contains the classical LSTM network and simplifies to it when the time intervals do not carry any information related to the underlying model.
5) Our LSTM architecture enables us to use the whole data sequence without any loss in the information entering to the LSTM network unlike [13], [14] and [5].
6) We achieve this substantial performance improvement with the same order of computational complexity with the vanilla LSTM network. The computational cost due to the time gates is only linear in the number of hidden neurons in the LSTM network.
7) Through extensive set of experiments involving synthetic and real datasets, we demonstrate significant performance gains achieved by our algorithm for both regression and classification problems.

D. Organization of the Paper

The organization of the paper as follows. We formally define our problem setting in Section II. In Section III, we first provide the derivations for the effect of the time information on the conventional gates and then introduce our LSTM architecture. In Section IV, we compare the performance of our architecture with respect to the state of the art architectures. The paper concludes with several remarks in Section V.

II. Problem Description

In this paper, all vectors are column vectors and denoted by boldface lower case letters. For a vector $\boldsymbol{x}$, $||\boldsymbol{x}|| = \sqrt{\boldsymbol{x}^T\boldsymbol{x}}$ is the ℓ^2-norm, where $\boldsymbol{x}^T$ is the ordinary transpose. $\langle\cdot,\cdot\rangle$ represents the outer product of two vectors, i.e., $\langle \boldsymbol{x}_1, \boldsymbol{x}_2\rangle = \boldsymbol{x}_1\boldsymbol{x}_2^T$. Vector sequences are denoted by boldface upper case letters, e.g., $\boldsymbol{X}$. $\boldsymbol{X}^{(i)}$ represents the i^{th} vector sequence in the dataset $\{\boldsymbol{X}^{(1)},\ldots,\boldsymbol{X}^{(N)}\}$, where N is the number of vector sequences in the set. $\mathcal{X}$ is the space of variable length vector sequences, i.e., $\boldsymbol{X}^{(i)} \in \mathcal{X}$. $\boldsymbol{X}^{(i)} = [\boldsymbol{x}_{t_1}^{(i)}, ..., \boldsymbol{x}_{t_{n_i}}^{(i)}]$ are the ordered sequence of vectors with length n_i, where $\boldsymbol{x}_{t_k}^{(i)}$ stands for the vector of $\boldsymbol{X}^{(i)}$ at time t_k, and k is the time index. x_j and $x_{t_k,j}$ represent the j^{th} elements in the vector $\boldsymbol{x}$ and $\boldsymbol{x}_{t_k}$, respectively. $\mathbf{1}_n \in \mathbb{R}^n$ stands for the vector, where all elements equal to 1. $W_{i,\{j,k\}}$ represents the element of the matrix $\boldsymbol{W}_i$ in j^{th} row and k^{th} column.

We study nonlinear regression and classification of non-uniformly sampled sequential data. We observe variable length vector sequences $\boldsymbol{X}^{(i)} = [\boldsymbol{x}_{t_1}^{(i)}, ..., \boldsymbol{x}_{t_{n_i}}^{(i)}] \in \mathcal{X}$, $\boldsymbol{x}_{t_k}^{(i)} \in \mathbb{R}^m$. The corresponding desired signal is given by $d_{t_k}^{(i)} \in \mathbb{R}$ in regression and $d_{n_i}^{(i)} \in \{1,\ldots,C\}$ for classification, where C is the number of classes. Our goal is to estimate $d_{t_k}^{(i)}$ by

$$\hat{d}_{t_k}^{(i)} = f_{t_k}(\boldsymbol{x}_{t_1}^{(i)},\ldots,\boldsymbol{x}_{t_k}^{(i)}),$$

where $f_{t_k}(\cdot)$ is a possibly time varying and adaptive nonlinear function at time step t_k. For the input vector $\boldsymbol{x}_{t_k}^{(i)}$, we suffer the loss $l(d_{t_k}^{(i)}, \hat{d}_{t_k}^{(i)})$ and the loss for the vector sequence $\boldsymbol{X}^{(i)}$ is the average of individual losses, which is denoted by $L^{(i)} = \frac{1}{n_i}\sum_{k=1}^{n_i} l(d_{t_k}^{(i)}, \hat{d}_{t_k}^{(i)})$. The total performance of the network is evaluated by the mean of the losses over all sequences:

$$L = \frac{1}{N}\sum_{i=1}^{N} L^{(i)}. \tag{1}$$

Since the data is non-uniformly sampled, the sampling times of the input vectors $\boldsymbol{x}_{t_k}$ are not regular, i.e., the time intervals between the consecutive input vectors, $\boldsymbol{x}_{t_k}$ and $\boldsymbol{x}_{t_{k+1}}$, may vary and we denote these sampling intervals by Δt_k's,

$$\Delta t_k \triangleq t_{k+1} - t_k.$$

As an example, in target tracking and position estimation application with a camera system [23], we sequentially receive position vectors of a target $\boldsymbol{x}_{t_k}$ and estimate its distance from a certain point $\boldsymbol{p}$ in the next position by $\hat{d}_{t_k}$. Here, the desired signal is given by $d_{t_k} = ||\boldsymbol{x}_{t_{k+1}} - \boldsymbol{p}||$ and under squared error loss, $l(d_{t_k}, \hat{d}_{t_k}) = (d_{t_k} - \hat{d}_{t_k})^2$. In the case of occlusions or when the camera misses frames, we do not receive position vectors and time intervals between consecutively received position vectors change, which corresponds to non-uniform sampling.

We use recurrent neural networks to process the sequential data. A generic RNN is given by [24]

$$\begin{aligned}\boldsymbol{h}_{t_k} &= f(\boldsymbol{W}_h\boldsymbol{x}_{t_k} + \boldsymbol{R}_h\boldsymbol{h}_{t_{k-1}})\\ \boldsymbol{y}_{t_k} &= g(\boldsymbol{R}_y\boldsymbol{h}_{t_k}),\end{aligned} \tag{2}$$

where $\boldsymbol{x}_{t_k} \in \mathbb{R}^m$ is the input vector, $\boldsymbol{h}_{t_k} \in \mathbb{R}^q$ is the state vector and $\boldsymbol{y}_{t_k} \in \mathbb{R}^p$ is the output at time t_k. $\boldsymbol{W}_h \in \mathbb{R}^{q\times m}$, $\boldsymbol{R}_h \in \mathbb{R}^{q\times q}$ and $\boldsymbol{R}_h \in \mathbb{R}^{q\times q}$ are the input weight matrices. $f(\cdot)$ and $g(\cdot)$ are point-wise nonlinear functions. We drop the sample index i to simplify the notation.

We focus on a special kind of the RNNs, the LSTM networks without the peephole connections. The LSTM network is described by the following equations [25]:

$$\boldsymbol{z}_{t_k} = g(\boldsymbol{W}_z\boldsymbol{x}_{t_k} + \boldsymbol{R}_z\boldsymbol{y}_{t_{k-1}}) \tag{3}$$

$$\boldsymbol{i}_{t_k} = \sigma(\boldsymbol{W}_i\boldsymbol{x}_{t_k} + \boldsymbol{R}_i\boldsymbol{y}_{t_{k-1}}) \tag{4}$$

$$\boldsymbol{f}_{t_k} = \sigma(\boldsymbol{W}_f\boldsymbol{x}_{t_k} + \boldsymbol{R}_f\boldsymbol{y}_{t_{k-1}}) \tag{5}$$

$$\boldsymbol{o}_{t_k} = \sigma(\boldsymbol{W}_o\boldsymbol{x}_{t_k} + \boldsymbol{R}_o\boldsymbol{y}_{t_{k-1}}) \tag{6}$$

$$\boldsymbol{c}_{t_k} = \boldsymbol{i}_{t_k} \odot \boldsymbol{z}_{t_k} + \boldsymbol{f}_{t_k} \odot \boldsymbol{c}_{t_{k-1}} \tag{7}$$

$$\boldsymbol{y}_{t_k} = \boldsymbol{o}_{t_k} \odot h(\boldsymbol{c}_{t_k}), \tag{8}$$

where $\boldsymbol{x}_{t_k} \in \mathbb{R}^m$ is the input vector, $\boldsymbol{c}_{t_k} \in \mathbb{R}^q$ is the state vector and $\boldsymbol{y}_{t_k} \in \mathbb{R}^q$ is the output vector at time t_k. $\boldsymbol{z}_{t_k}$ is the block input, $\boldsymbol{i}_{t_k}$, $\boldsymbol{f}_{t_k}$ and $\boldsymbol{o}_{t_k}$ are the input, forget and output gates, respectively. Nonlinear activation functions $g(\cdot)$, $h(\cdot)$ and $\sigma(\cdot)$ apply the point-wise operations. $\tanh(\cdot)$ is commonly used for $g(\cdot)$ and $h(\cdot)$ functions and $\sigma(\cdot)$ is the sigmoid function, i.e., $\sigma(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{x}}}$. $\odot$ is the element-wise (Hadamard) product and operates on the two vectors of the same size. $\boldsymbol{W}_z$, $\boldsymbol{W}_i$, $\boldsymbol{W}_f$, $\boldsymbol{W}_o \in \mathbb{R}^{q\times m}$ are the input weight matrices and $\boldsymbol{R}_z$, $\boldsymbol{R}_i$, $\boldsymbol{R}_f$, $\boldsymbol{R}_o \in \mathbb{R}^{q\times q}$ are the recurrent weight matrices. With the abuse of notation, we incorporate the bias weights, $\boldsymbol{b}_z$, $\boldsymbol{b}_i$, $\boldsymbol{b}_f$, $\boldsymbol{b}_o \in \mathbb{R}^q$, into the input weight matrices and denote them by $\boldsymbol{W}_\theta = [\boldsymbol{W}_\theta; \boldsymbol{b}_\theta]$, $\theta \in \{z, i, f, o\}$, where $\boldsymbol{x}_{t_k} = [\boldsymbol{x}_{t_k}; 1]$. For the regression problem, we generate the estimate $\hat{d}_{t_k}$ as

$$\hat{d}_{t_k} = \boldsymbol{w}_{t_k}^T\boldsymbol{y}_{t_k},$$

where $\boldsymbol{w}_{t_k} \in \mathbb{R}^q$ is the final regression coefficients, which can be trained in an online or batch manner depending on the application.

For the classification problem, we focus on the sequence classification, i.e., we have only one desired signal $d^{(i)}$ for each vector sequence $\boldsymbol{X}^{(i)}$. As shown in Fig. 1, our final decision $\hat{d}^{(i)}$ is given by

$$\hat{d}^{(i)} = \max_j \text{softmax}(\boldsymbol{W}\tilde{\boldsymbol{y}}^{(i)})_j,$$

where $\boldsymbol{W} \in \mathbb{R}^{q\times c}$ is the weight matrix, c is the number of classes, and $\tilde{\boldsymbol{y}}^{(i)}$ is the combination of the LSTM network outputs, $\boldsymbol{y}_{t_1}^{(i)},\ldots,\boldsymbol{y}_{t_{n_i}}^{(i)}$. To obtain $\tilde{\boldsymbol{y}}^{(i)}$, we may use three

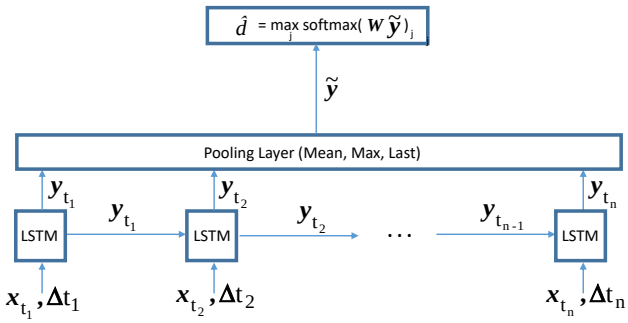


Fig. 1: Detailed schematic of the classifiacation architecture. Note that the index i is dropped in order to simplify the notation.

different pooling methods: mean, max and last pooling as

$$\tilde{\boldsymbol{y}}^{(i)}_{mean} = \frac{1}{n_i}\sum_{k=1}^{n_i} \boldsymbol{y}^{(i)}_{t_k}$$
$$\tilde{y}^{(i)}_{max_j} = \max_k (y^{(i)}_{t_k,j})$$
$$\tilde{\boldsymbol{y}}^{(i)}_{last} = \boldsymbol{y}^{(i)}_{t_{n_i}}.$$

In Section III, we introduce a novel LSTM architecture working on non-uniformly sampled data, and also provide its forward-pass and backward-pass update formulas.

III. A Novel LSTM Architecture

We need to incorporate the time information into the LSTM network to enhance the performance [5]. For this purpose, one can directly append the sampling intervals, Δt_k's, to the input vector, i.e., $\tilde{\boldsymbol{x}}_{t_k} = [\boldsymbol{x}_{t_k}; \Delta t_k]$. However, in this solution, Δt_k is incorporated as an additional feature and its effect is only additive to the weighted sum of the other features, e.g., as multiplied by $\tilde{\boldsymbol{W}}\tilde{\boldsymbol{x}_{t_k}}$, where $\tilde{\boldsymbol{W}} \in \mathbb{R}^{q\times(m+1)}$ is the extended weight matrix. For example, the input gate $\boldsymbol{i}_{t_k}$ is calculated by

$$\boldsymbol{i}_{t_k} = \sigma(\tilde{\boldsymbol{W}}_i \tilde{\boldsymbol{x}}_{t_k} + \boldsymbol{R}_i \boldsymbol{y}_{t_{k-1}}), \tag{9}$$

instead of (4), where $\boldsymbol{W}_i \boldsymbol{x}_{t_k}$ term changes as $\tilde{\boldsymbol{W}}_i \tilde{\boldsymbol{x}}_{t_k}$. In that case, the only difference between (9) and (4) is the additive term of $\tilde{W}_{i,\{j,m+1\}}\Delta t_k$ inside $\sigma(\cdot)$. In the following, we will demonstrate that the Δt_k should also have a scaling effect on the conventional gates, i.e., the input, forget and output gates.

To this end, in subsection III-A, we first consider a special case of non-uniform sampling, where $\boldsymbol{X}^{(i)}$ is uniformly sampled, however, certain columns of $\boldsymbol{X}^{(i)}$ are missing. We then extend our approach to arbitrary non-uniform sampling case in subsection III-B.

A. Modeling Non-uniform Sampling with Missing Input Case

We make our derivations first for the RNN case for one step ahead prediction problem, i.e., the aim is to estimate the next signal $\boldsymbol{x}_{t_{k+1}}$, where the current input is $\boldsymbol{x}_{t_k}$. We first consider the case when we have uniform sampling, i.e., $t_{k+1} - t_k = \Delta$ for all time steps, where Δ is some fixed time interval. In this framework, we simply combine the RNN equations (2), then, the RNN model estimates the next sample as

$$\begin{aligned}\hat{\boldsymbol{x}}_{t_{k+1}} &= g\left(\boldsymbol{R}_y f\left(\boldsymbol{W}_h \boldsymbol{x}_{t_k} + \boldsymbol{R}_h \boldsymbol{h}_{t_{k-1}}\right)\right) \\ &= \bar{f}(\boldsymbol{x}_{t_k}, \boldsymbol{h}_{t_{k-1}}),\end{aligned} \tag{10}$$

where $\bar{f}(\cdot)$ is a composite function, which includes $f(\cdot)$ and $g(\cdot)$. Assume that $\boldsymbol{x}_{t_k}$ are the samples of an infinitely differentiable continuous function of time, $\boldsymbol{x}$. In this case, $\boldsymbol{x}_{t_{k+1}}$ is calculated by the Taylor series expansion of $\boldsymbol{x}$ around $\boldsymbol{x}_{t_k}$ as

$$\begin{aligned}\boldsymbol{x}_{t_{k+1}} &= \boldsymbol{x}_{t_k+\Delta} \\ &= \boldsymbol{x}_{t_k} + \Delta\frac{\partial \boldsymbol{x}_{t_k}}{\partial t} + \Delta^2\frac{\partial^2 \boldsymbol{x}_{t_k}}{\partial t^2} + \Delta^3\frac{\partial^3 \boldsymbol{x}_{t_k}}{\partial t^3} + \dots\end{aligned} \tag{11}$$

We now model the non-uniform sampling case with missing instances, i.e., any Δt_k is an integer multiple of the fixed time interval Δ. For example, if the next input $\boldsymbol{x}_{t_{k+1}}$ is not missing, then the time interval $\Delta t_k = \Delta$. Similarly, if $\boldsymbol{x}_{t_{k+1}}$ is missing, but we have $\boldsymbol{x}_{t_{k+2}}$, then $\Delta t_k = 2\Delta$. Assume that the $\boldsymbol{x}_{t_{k-1}}$ and $\boldsymbol{x}_{t_{k+1}}$ are available, while the $\boldsymbol{x}_{t_k}$ is missing from our data sequence. In this case, we cannot directly apply the same Taylor series expansion in (11) to calculate $\boldsymbol{x}_{t_{k+1}}$ since the data $\boldsymbol{x}_{t_k}$ is missing. However, we have an estimate $\hat{\boldsymbol{x}}_{t_k}$, which is obtained by the model in (10) using the input $\boldsymbol{x}_{t_{k-1}}$. Therefore, we estimate $\boldsymbol{x}_{t_{k+1}}$ by using $\hat{\boldsymbol{x}}_{t_k}$ instead of $\boldsymbol{x}_{t_k}$ in (11) as

$$\begin{aligned}\boldsymbol{x}_{t_{k+1}} &\approx \sum_{n=0}^{\infty} \Delta^n \frac{\partial^n \hat{\boldsymbol{x}}_{t_k}}{\partial t^n} \\ &= \hat{\boldsymbol{x}}_{t_k} + \Delta\frac{\partial \hat{\boldsymbol{x}}_{t_k}}{\partial t} + \Delta^2\frac{\partial^2 \hat{\boldsymbol{x}}_{t_k}}{\partial t^2} + \Delta^3\frac{\partial^3 \hat{\boldsymbol{x}}_{t_k}}{\partial t^3} + \dots\end{aligned} \tag{12}$$

We next substitute $\hat{\boldsymbol{x}}_{t_k}$ with $\bar{f}(\boldsymbol{x}_{t_{k-1}}, \boldsymbol{h}_{t_{k-2}})$ by using (10) to yield

$$\begin{aligned}\hat{\boldsymbol{x}}_{t_{k+1}} = \bar{f}(\boldsymbol{x}_{t_{k-1}}, \boldsymbol{h}_{t_{k-2}}) &+ \frac{\Delta}{1!}\frac{\partial \bar{f}(\boldsymbol{x}_{t_{k-1}}, \boldsymbol{h}_{t_{k-2}})}{\partial t} \\ &+ \frac{\Delta^2}{2!}\frac{\partial^2 \bar{f}(\boldsymbol{x}_{t_{k-1}}, \boldsymbol{h}_{t_{k-2}})}{\partial t^2} + \dots\end{aligned} \tag{13}$$

We write (13) in the vector form as

$$\hat{x}_{t_{k+1},j} = \begin{bmatrix} 1 & \frac{\Delta}{1!} & \frac{\Delta^2}{2!} & \frac{\Delta^3}{3!} & \dots \end{bmatrix} \begin{bmatrix} \bar{f}\left(\boldsymbol{x}_{t_{k-1}}, \boldsymbol{h}_{t_{k-2}}\right)_j \\ \bar{f}'\left(\boldsymbol{x}_{t_{k-1}}, \boldsymbol{h}_{t_{k-2}}\right)_j \\ \bar{f}''\left(\boldsymbol{x}_{t_{k-1}}, \boldsymbol{h}_{t_{k-2}}\right)_j \\ \bar{f}'''\left(\boldsymbol{x}_{t_{k-1}}, \boldsymbol{h}_{t_{k-2}}\right)_j \\ \vdots \end{bmatrix}, \tag{14}$$

where $\bar{f}'(\cdot)$ represents the derivative with respect to t and similarly for the other derivative terms. We approximate this equation as

$$\hat{\boldsymbol{x}}_{t_{k+1}} \approx f_\Delta(\Delta) \odot f_{\boldsymbol{x},\boldsymbol{h}}(\boldsymbol{x}_{t_{k-1}}, \boldsymbol{h}_{t_{k-2}}), \tag{15}$$

where $f_\Delta(\cdot)$ is a nonlinear function of Δ, whereas $f_{\boldsymbol{x},\boldsymbol{h}}(\cdot)$ represents a nonlinear function of $\boldsymbol{x}_{t_{k-1}}$ and $\boldsymbol{h}_{t_{k-2}}$. Note that both $f_\Delta(\cdot)$ and $f_{\boldsymbol{x},\boldsymbol{h}}(\cdot)$ return vectors as their outputs in the length of $\boldsymbol{x}_{t_k}$. This derivation can be extended to any length of missing instances such as for 2Δ this yields

$$\hat{\boldsymbol{x}}_{t_{k+2}} \approx f_\Delta(2\Delta) \odot f_{\boldsymbol{x},\boldsymbol{h}}(\boldsymbol{x}_{t_{k-1}}, \boldsymbol{h}_{t_{k-2}}). \tag{16}$$

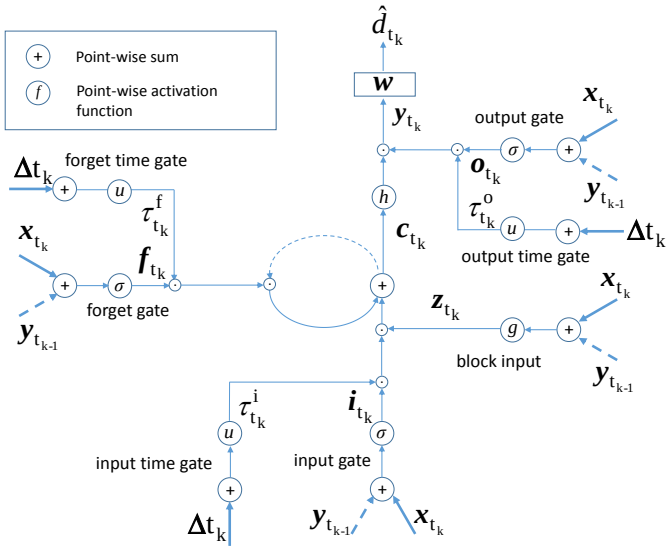


Fig. 2: Detailed schematic of an LSTM block with additional time gates. Note that $\boldsymbol{x}_{t_k}$, $\boldsymbol{y}_{t_{k-1}}$ and $\boldsymbol{\Delta} t_k$ are multiplied with their weights, $\boldsymbol{W}_{(\cdot)}$ and $\boldsymbol{R}_{(\cdot)}$, according to (3)-(5) and (20)-(23). Also corresponding biases $\boldsymbol{b}_{(\cdot)}$ is added.

Hence, the time interval Δ has a nonlinear scaling effect on $f_{\boldsymbol{x},\boldsymbol{h}}(\cdot)$. Note that in uniform sampling case, the classical RNNs use only $f_{\boldsymbol{x},\boldsymbol{h}}(\cdot)$ to estimate the next sample, i.e., $\bar{f}(\cdot)$ in (10), although the scaling effect of time interval still exists. However, $f_{\boldsymbol{x},\boldsymbol{h}}(\cdot)$ is able to handle this scaling effect since Δ's and $f_\Delta(\Delta)$ are constant.

In subsection III-B, we focus on the arbitrary non-uniform sampling case.

B. Arbitrary Non-uniform Sampling

In this subsection, we consider the arbitrary non-uniform sampling, i.e., sampling without any constant sampling interval and Δt_k is not an integer multiple of a fixed time interval Δ. The Taylor series expansion for the missing data case is similarly extended to arbitrary non-uniform sampling case for the one step ahead estimation problem, i.e.,

$$\begin{aligned}\boldsymbol{x}_{t_{k+1}} &= \sum_{n=0}^{\infty} {\Delta t_k}^n \frac{\partial^n \boldsymbol{x}_{t_k}}{\partial t^n} \\ &= \boldsymbol{x}_{t_k} + \Delta t_k \frac{\partial \boldsymbol{x}_{t_k}}{\partial t} + \Delta t_k^2 \frac{\partial^2 \boldsymbol{x}_{t_k}}{\partial t^2} + \Delta t_k^3 \frac{\partial^3 \boldsymbol{x}_{t_k}}{\partial t^3} + \dots\end{aligned} \tag{17}$$

Similar derivations lead to

$$\hat{\boldsymbol{x}}_{t_{k+1}} = f_\Delta(\Delta t_k) \odot f_{\boldsymbol{x},\boldsymbol{h}}(\boldsymbol{x}_{t_k}, \boldsymbol{h}_{t_{k-1}}). \tag{18}$$

In the non-uniform sampling case, $f_\Delta(\Delta t_k)$ have unique scaling effect on $f_{\boldsymbol{x},\boldsymbol{h}}(\cdot)$ at each time step since Δt_k differs. Therefore, ignoring the time information in estimation process results in a limited performance. Extending input vector with time intervals makes only an additive contribution to the $f_{\boldsymbol{x},\boldsymbol{h}}(\boldsymbol{x}_{t_k}, \boldsymbol{h}_{t_{k-1}})$ term, which is insufficient to model the effect of $f_\Delta(\Delta t_k)$. To circumvent this issue, we introduce a new RNN structure, particularly, an LSTM architecture, which includes the effect of $f_\Delta(\Delta t_k)$. The new LSTM architecture is explained in the next subsection.

C. Time Gated - LSTM Architecture

We present a novel LSTM architecture to incorporate the time information into our estimation function as a nonlinear scaling factor, i.e., it learns the time dependent scaling function $f_\Delta(\cdot)$. In the classical LSTM architecture, $f_{\boldsymbol{x},\boldsymbol{h}}(\boldsymbol{x}_{t_k}, \boldsymbol{h}_{t_{k-1}})$ is already modelled as $\sigma(\boldsymbol{W}\boldsymbol{x}_{t_k} + \boldsymbol{R}\boldsymbol{y}_{t_{k-1}})$ in the specialized gate structures as in (3)-(6). Therefore, we focus on modeling $f_\Delta(\cdot)$. In accordance with (18), we can straightforwardly incorporate the time information into the LSTM architecture by altering (8) as

$$\boldsymbol{y}_{t_k} = \boldsymbol{o}_{t_k} \odot h(\boldsymbol{c}_{t_k}) \odot f_\Delta(\Delta t_k). \tag{19}$$

Here, we incorporate $f_\Delta(\Delta t_k)$ to the LSTM architecture as a scaling factor only on the output gate. Since the gate structures in the LSTM architecture are specialized for different tasks, such as forgetting the last state, their responses to the time intervals need to be different. For example, when the input $\boldsymbol{x}_{t_k}$ arrives after a long time interval Δt_k, while the forget gate needs to keep a small amount of the past state, the input gate needs to incorporate more from the new input. To this end, we decompose $f_\Delta(\Delta t_k)$ into three different functions, $f_\Delta^{(i)}(\Delta t_k)$, $f_\Delta^{(f)}(\Delta t_k)$ and $f_\Delta^{(o)}(\Delta t_k)$, and use these functions on the conventional gates in order to allow them to give different responses depending on the time intervals. In particular, we introduce new time gates to the LSTM network in order to model the scaling effect of $f_\Delta(\cdot)$. This LSTM architecture is named as Time Gated LSTM (TG-LSTM) in this paper.

We introduce three different time gates, which use sampling intervals, Δt_k's, as their inputs as shown in Fig. 2. The first time gate is the input time gate and denoted by $\boldsymbol{\tau}^i_{t_k}$, the second time gate is the forget time gate and represented by $\boldsymbol{\tau}^f_{t_k}$. Similarly, the third time gate is the output time gate and denoted by $\boldsymbol{\tau}^o_{t_k}$. Note that there is no time gate $\boldsymbol{\tau}^z_{t_k}$, since $\boldsymbol{i}_{t_k}$ and $\boldsymbol{z}_{t_k}$ participate to the network as multiplied with each other and only one time gate $\boldsymbol{\tau}^i_{t_k}$ is sufficient to scale both. The input gate $\boldsymbol{i}_{t_k}$, forget gate $\boldsymbol{f}_{t_k}$ and output gate $\boldsymbol{o}_{t_k}$ are multiplied by $\boldsymbol{\tau}^i_{t_k}$, $\boldsymbol{\tau}^f_{t_k}$, $\boldsymbol{\tau}^o_{t_k}$ respectively as shown in Fig. 2. In addition to (3)-(8), the forward-pass process of the new LSTM architecture in Fig. 2 is modelled by the following set of equations:

$$\boldsymbol{\tau}^i_{t_k} = u(\boldsymbol{W}_{\tau^i}\boldsymbol{\Delta} t_k) \tag{20}$$
$$\boldsymbol{\tau}^f_{t_k} = u(\boldsymbol{W}_{\tau^f}\boldsymbol{\Delta} t_k) \tag{21}$$
$$\boldsymbol{c}_{t_k} = \boldsymbol{i}_{t_k} \odot \boldsymbol{z}_{t_k} \odot \boldsymbol{\tau}^i_{t_k} + \boldsymbol{f}_{t_k} \odot \boldsymbol{c}_{t_{k-1}} \odot \boldsymbol{\tau}^f_{t_k} \tag{22}$$
$$\boldsymbol{\tau}^o_{t_k} = u(\boldsymbol{W}_{\tau^o}\boldsymbol{\Delta} t_k) \tag{23}$$
$$\boldsymbol{y}_{t_k} = \boldsymbol{o}_{t_k} \odot \boldsymbol{\tau}^o_{t_k} \odot h(\boldsymbol{c}_{t_k}), \tag{24}$$

where $\boldsymbol{W}_{\tau^i}$, $\boldsymbol{W}_{\tau^f}$ and $\boldsymbol{W}_{\tau^o} \in \mathbb{R}^{q \times n_\tau}$ are the weight matrices of the time gates $\boldsymbol{\tau}^i$, $\boldsymbol{\tau}^f$ and $\boldsymbol{\tau}^o$, respectively. $u(\cdot)$ is the point-wise nonlinearity, which is set to $\sigma(\cdot)$. $\boldsymbol{\Delta} t_k \in \mathbb{R}^{n_\tau}$ is the input for the time gates and one can append different functions of Δt_k such as $(\Delta t_k)^2$ and $\frac{1}{\Delta t_k}$ in addition to Δt_k. Here, (20), (21) and (23) are added to the set of forward-pass equations of the classical LSTM architecture, (22) and (24) are replaced with (7) and (8), respectively.

Architecture	Computational Load
LSTM-1	$4q^2 + 4qm + 3q$
LSTM-2	$4q^2 + 4qm + 7q$
PLSTM	$4q^2 + 4qm + 3q$
TG-LSTM	$4q^2 + 4qm + 6q$

TABLE I: The number of multiplication operations in the forward pass of the TG-LSTM, PLSTM and the classical LSTM architectures for one time step. LSTM-1 is the network that time intervals are not used. LSTM-2 represents the LSTM network, where the time intervals are added to the input vector as another feature.

D. Training of the New Model

For the training of the TG-LSTM architecture, we employ the back-propagation through time (BPTT) algorithm to update the weight matrices of our LSTM network, i.e., the input weight matrices $\boldsymbol{W}_z, \boldsymbol{W}_i, \boldsymbol{W}_f, \boldsymbol{W}_o, \boldsymbol{W}_{\tau^i}, \boldsymbol{W}_{\tau^f}, \boldsymbol{W}_{\tau^o}$, and the recurrent weight matrices $\boldsymbol{R}_z, \boldsymbol{R}_i, \boldsymbol{R}_f, \boldsymbol{R}_o$. To write the update equations in a notationally simplified form, we first define a new notation for the gates before the nonlinearity is applied, e.g.,

$$\begin{aligned}\bar{\boldsymbol{i}}_{t_k} &= \boldsymbol{W}_i \boldsymbol{x}_{t_k} + \boldsymbol{R}_i \boldsymbol{y}_{t_{k-1}} \\ \bar{\boldsymbol{\tau}}^i_{t_k} &= \boldsymbol{W}_{\tau^i} \boldsymbol{x}_{t_k},\end{aligned}$$

where $\bar{\boldsymbol{i}}_{t_k} \in \mathbb{R}^q$ and $\bar{\boldsymbol{\tau}}^i_{t_k} \in \mathbb{R}^q$ are the sum terms before the nonlinearity for the input gate and input time gate, respectively. The terms for the other gates $\bar{\boldsymbol{z}}_{t_k}$, $\bar{\boldsymbol{f}}_{t_k}$, $\bar{\boldsymbol{o}}_{t_k}$, $\bar{\boldsymbol{\tau}}^f_{t_k}$, $\bar{\boldsymbol{\tau}}^o_{t_k} \in \mathbb{R}^q$ have similar formulations. Then, we first calculate the local gradients as follows:

$$\begin{aligned}\delta \boldsymbol{y}_{t_k} =& \frac{\partial L}{\partial y_{t_k}} + \boldsymbol{R}_z^T \delta \boldsymbol{z}_{t_{k+1}} + \boldsymbol{R}_i^T \delta \boldsymbol{i}_{t_{k+1}} \\ &+ \boldsymbol{R}_f^T \delta \boldsymbol{f}_{t_{k+1}} + \boldsymbol{R}_o^T \delta \boldsymbol{o}_{t_{k+1}} \\ \delta \boldsymbol{o}_{t_k} =& \ \delta \boldsymbol{y}_{t_k} \odot h(\boldsymbol{c}_{t_k}) \odot \boldsymbol{\tau}^o_{t_k} \odot \sigma'(\bar{\boldsymbol{o}}_{t_k}) \\ \delta \boldsymbol{\tau}^o_{t_k} =& \ \delta \boldsymbol{y}_{t_k} \odot h(\boldsymbol{c}_{t_k}) \odot \boldsymbol{o}_{t_k} \odot u'(\bar{\boldsymbol{\tau}}^o{}_{t_k}) \\ \delta \boldsymbol{c}_{t_k} =& \ \delta \boldsymbol{y}_{t_k} \odot \boldsymbol{o}_{t_k} \odot \boldsymbol{\tau}^o_{t_k} \odot h'(\boldsymbol{c}_{t_k}) + \boldsymbol{f}_{t_{k+1}} \odot \delta \boldsymbol{c}_{t_{k+1}} \\ \delta \boldsymbol{f}_{t_k} =& \ \delta \boldsymbol{c}_{t_k} \odot \boldsymbol{c}_{t_{k-1}} \odot \boldsymbol{\tau}^f_{t_k} \odot \sigma'(\bar{\boldsymbol{f}}_{t_k}) \\ \delta \boldsymbol{\tau}^f_{t_k} =& \ \delta \boldsymbol{c}_{t_k} \odot \boldsymbol{c}_{t_{k-1}} \odot \boldsymbol{f}_{t_k} \odot \sigma'(\bar{\boldsymbol{\tau}}^f{}_{t_k}) \\ \delta \boldsymbol{i}_{t_k} =& \ \delta \boldsymbol{c}_{t_k} \odot \boldsymbol{z}_{t_k} \odot \boldsymbol{\tau}_{t_k} \odot \sigma'(\bar{\boldsymbol{i}}_{t_k}) \\ \delta \boldsymbol{z}_{t_k} =& \ \delta \boldsymbol{c}_{t_k} \odot \boldsymbol{i}_{t_k} \odot \boldsymbol{\tau}_{t_k} \odot g'(\bar{\boldsymbol{z}}_{t_k}) \\ \delta \boldsymbol{\tau}^i_{t_k} =& \ \delta \boldsymbol{c}_{t_k} \odot \boldsymbol{i}_{t_k} \odot \boldsymbol{z}_{t_k} \odot u'(\bar{\boldsymbol{\tau}}^i_{t_k}),\end{aligned}$$

where $\delta \boldsymbol{y}_{t_k}$, $\delta \boldsymbol{o}_{t_k}$, $\delta \boldsymbol{\tau}^o_{t_k}$, $\delta \boldsymbol{c}_{t_k}$, $\delta \boldsymbol{f}_{t_k}$, $\delta \boldsymbol{\tau}^f_{t_k}$ $\delta \boldsymbol{c}_{t_k}$, $\delta \boldsymbol{i}_{t_k}$, $\delta \boldsymbol{z}_{t_k}$, $\delta \boldsymbol{\tau}^i_{t_k} \in \mathbb{R}^q$ are the local gradients for corresponding nodes. The gradients for the input and the recurrent weight matrices are calculated by

$$\begin{aligned}\delta \boldsymbol{W}_\theta &= \sum_{k=0}^{n} \langle \delta \boldsymbol{\theta}_{t_k}, \boldsymbol{x}_{t_k} \rangle \\ \delta \boldsymbol{R}_\theta &= \sum_{k=0}^{n-1} \langle \delta \boldsymbol{\theta}_{t_{k+1}}, \boldsymbol{y}_{t_k} \rangle,\end{aligned}$$

where $\theta \in \{z, i, f, o\}$, and the gradient for weights of the time gates are calculated by

$$\delta \boldsymbol{W}_{\tau^*} = \sum_{k=0}^{n} \langle \delta \boldsymbol{\tau}^*_{t_k}, \boldsymbol{\Delta} t_k \rangle,$$

where $* \in \{i, f, o\}$ and $\boldsymbol{\Delta} t_k = [\Delta t_k; 1]$. $\langle \cdot, \cdot \rangle$ represents the outer product of two vectors, i.e., $\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle = \boldsymbol{x}_1 \boldsymbol{x}_2^T$.

Remark 1: Our TG-LSTM architecture has additional time gates on top of the vanilla LSTM architecture. One can remove any time gate by setting its all elements to 1, for example, to close input time gate, $\boldsymbol{\tau}^i = \boldsymbol{1}_q$. In the worst case, the time intervals have no correlation with the underlying model, all time gates converge to $\boldsymbol{1}_q$ vector and our TG-LSTM architecture simplifies to the classical LSTM architecture.

Remark 2: The complexity of the new architecture is in the same order of the complexity of the classical LSTM architecture. In Table I, we provide the computational loads in terms of the number of required multiplication operations in the forward pass for the classical LSTM, PLSTM and TG-LSTM architectures. In the table, LSTM-1 represents the LSTM network in which the time intervals are not incorporated to the input vector, i.e., the input vector is merely $\boldsymbol{x}_{t_k}$. LSTM-2 is the network in which the input vectors are extended with the time intervals between the samples as another feature, i.e., $\tilde{\boldsymbol{x}}_{t_k} = [\boldsymbol{x}_{t_k}; \Delta t_k]$. Four matrix vector multiplications for the input, i.e., $\boldsymbol{W}\boldsymbol{x}_{t_k}$, four matrix vector multiplications for the last hidden state, i.e., $\boldsymbol{R}\boldsymbol{h}_{t_{k-1}}$, and three vector-vector multiplications between gates, i.e., (7) and (8) in the response of the previous comment, are included in the basic LSTM architecture, which need $4q^2+4qm+3q$ multiplication operations. Since the LSTM-2 architecture has an extended input vector, it has $4q(m+1)$ multiplications instead of $4qm$ from the $\boldsymbol{W}\boldsymbol{x}_{t_k}$ terms. The PLSTM architecture has additional scalar operations for the sampler functions, however, since we include only vectorial multiplications, it has $4q^2 + 4qm + 3q$ multiplication operations in one time step. The TG-LSTM architecture has additional $3q$ multiplications due to the multiplications of time gates with the conventional gates.

IV. Simulations

In this section, we illustrate the performance of the proposed LSTM architecture under different scenarios with respect to the state of the art methods through several experiments. In the first part, we focus on the regression problem for various real life datasets such as kinematic [26], bank [27] and pumadyn [28]. In the second part, we compare our method with the LSTM structures on several different classification tasks over real life datasets such as Pen-Based Recognition of Handwritten Digits [29] and UJI Pen (Version 2) [29] datasets.

A. Regression Task

In this subsection, we evaluate the performances of the TG-LSTM and the vanilla LSTM architectures for the regression problem. The classical LSTM architecture uses the time intervals as another feature in the input vectors, i.e., the LSTM-2 architecture defined in III-D. Therefore, for a dataset with the input size m, the classical LSTM architecture has the

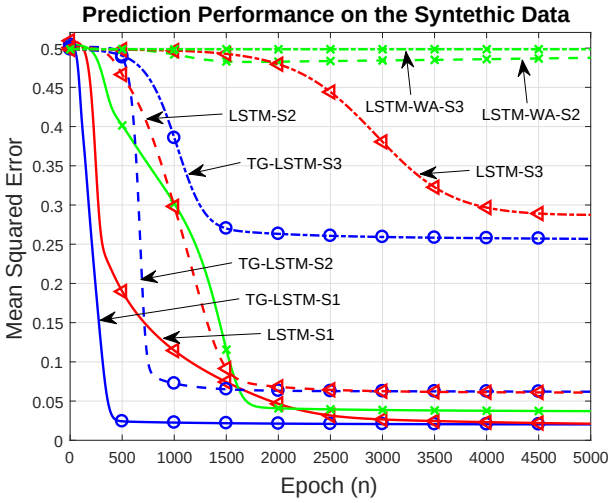


Fig. 3: Regression performance of the TG-LSTM and the LSTM networks on the syntehtic sine dataset with different sampling intervals. The sampling intervals Δt_k are drawn uniformly from the range $[2, 10]$, $[5, 20]$ and $[20, 50]$ ms for S1, S2 and S3 simulations, respectively. LSTM represents the classical LSTM architecture in [5], which uses the time intervals as another feature. LSTM-WA is the classical LSTM architecture in [13], [14], which uses windowing and averaging operations on the data before entering the LSTM network.

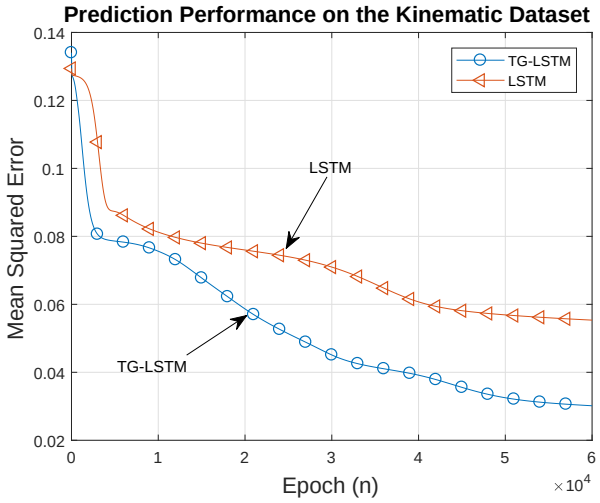


Fig. 4: Regression performance of the TG-LSTM and LSTM networks on the Kinematic dataset.

input size $m + 1$. LSTM-WA represents the classical LSTM architecture in [13], [14], which uses windowing and averaging operations on the data before entering the LSTM network. We train the networks with Stochastic Gradient Descent (SGD) algorithm using the constant learning rate.

We first consider a sine wave with frequency 10 Hz and length $n = 1000$ for training and $n = 500$ for testing. The sampling intervals Δt_k are drawn uniformly from the range $[2, 10]$, $[5, 20]$ and $[20, 50]$ ms for S1, S2 and S3 simulations, respectively. Our aim is to predict the next sample $x_{t_{k+1}}$. For this data, the input is scalar $x_{t_k} \in \mathbb{R}$, i.e., the input size $m = 1$, and the output $d_{t_k} \in \mathbb{R}$, where $d_{t_k} = x_{t_{k+1}}$. For the parameter selection, we perform a grid search on the number of hidden neurons and learning rate in the intervals $q = [3, 20]$ and $\eta = [10^{-3}, 10^{-6}]$, respectively. For the window size of the classical LSTM architecture with preprocessing method, we search on the interval $[\frac{\Delta_{max}}{2}, \Delta_{max}]$, where Δ_{max} equals to 10, 20 and 50 ms, respectively. We choose the parameters with 5-fold cross validation, however, we only use the first and last folds for validation to keep the sequential pattern of the data. Otherwise, the sequence is corrupted, e.g., the last sample of the first fold is followed by the first sample of the third fold instead of the second fold. We choose the learning rate as $\eta = 10^{-4}$ for S1 and S2 simulations and $\eta = 2 \times 10^{-5}$ for S3 simulations. The number of hidden neurons are chosen as $q = 20$ for all simulations. The window sizes for the method using windowing and averaging technique are 5, 20 and 50 ms for S1, S2 and S3, respectively. We initiate the weights of the time gates of the TG-LSTM architecture from the distribution $\mathcal{N}(\frac{1}{\mathbb{E}[\Delta t_k]}, 0.01)$ to start the time gates in the smooth area of the sigmoid activation function and prevent the gradients from diminishing due to multiplication. Other weights are initiated from the distribution $\mathcal{N}(0, 0.01)$.

In Fig. 3, we demonstrate the regression performance of the algorithms under different sampling interval ranges in terms of the mean squared error on the test set per epoch. LSTM represents the classical LSTM architecture in [5], which uses the time intervals as another feature. LSTM-WA is the classical LSTM architecture in [13], [14], which uses windowing and averaging operations on the data before entering the LSTM network. In Fig. 3, one can see that the performance improvement by the TG-LSTM architecture becomes more evident for the larger time intervals. While all three architectures achieve similar results in terms of the steady-state error in S1 simulations, the performance difference between the TG-LSTM architecture and the classical LSTM architecture using preprocessing technique significantly increases in S2 simulations. Furthermore, in S3 simulations, we observe a higher performance difference between TG-LSTM and the classical LSTM architectures. Moreover, the TG-LSTM architecture outperforms the other architectures in terms of the convergence rate in all cases.

Other than the sine wave, we compare the TG-LSTM and the classical LSTM architectures on kinematic [26], bank [27] and pumadyn [28] datasets. The results for the LSTM-WA method are not included due to comparatively much better performances of the other methods. Each dataset contains an input vector sequence and the corresponding desired signals for each time step. These datasets do not have separate training and test sets, therefore, we split the sequences in each dataset such that first 60% of the sequence is used for the training and the following 40% is used for the test. Since the datasets contain uniformly sampled sequences, we first need to convert them to the non-uniformly sampled sequences. For this purpose, we sequentially under-sample the sequences based on a probabilistic model. Assume that we have the uniformly sampled input sequence $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l]$. If we receive $\boldsymbol{x}_j$ from the original sequence as $\boldsymbol{x}_{t_k}$, the next sample $\boldsymbol{x}_{t_{k+1}}$ is chosen from the remaining sequence $[\boldsymbol{x}_{j+1}, \ldots, \boldsymbol{x}_l]$ according

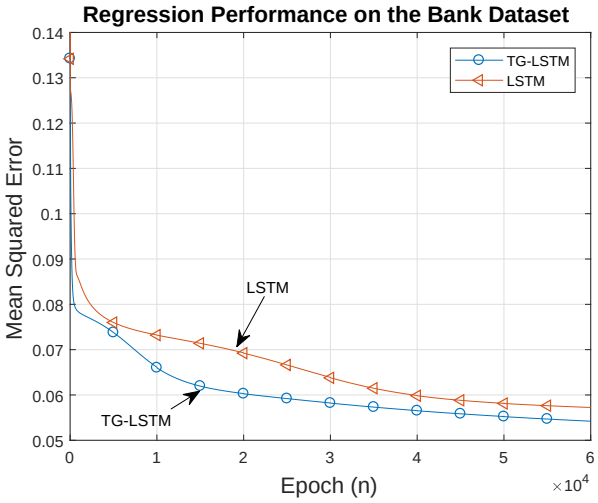


Fig. 5: Regression performance of the TG-LSTM and LSTM networks on the Bank dataset.

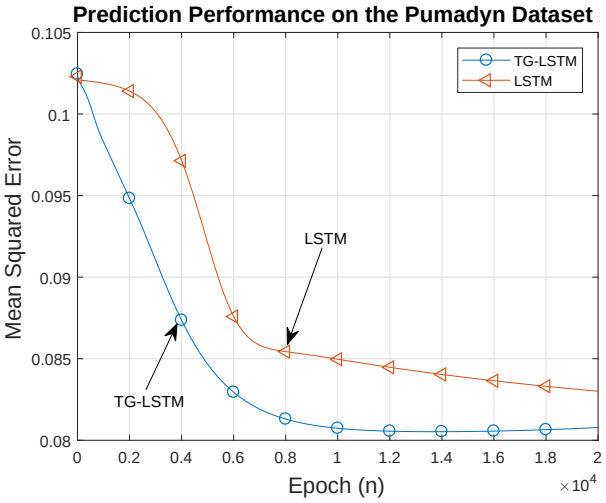


Fig. 6: Regression performance of the TG-LSTM and LSTM networks on the Pumadyn dataset.

to the probabilistic model. In our simulations, we use

$$\mathrm{p}(\Delta t_k) = \begin{cases} 0.4, & \text{if } \Delta t_k = 1 \\ 0.4, & \text{if } \Delta t_k = 2 \\ 0.2, & \text{if } \Delta t_k = 3 \\ 0 & \text{otherwise} \end{cases}, \tag{25}$$

where $\mathrm{p}(\Delta t_k)$ is the probability mass function for the time difference $\Delta t_k = t_{k+1} - t_k$, e.g., $P(\boldsymbol{x}_{t_{k+1}} = \boldsymbol{x}_{j+1} | \boldsymbol{x}_{t_k} = \boldsymbol{x}_j) = 0.4$ and $P(\boldsymbol{x}_{t_{k+1}} = \boldsymbol{x}_{j+3} | \boldsymbol{x}_{t_k} = \boldsymbol{x}_j) = 0.2$. Using (25), we generate the non-uniformly sampled sequence $\boldsymbol{X}_{nu} = [\boldsymbol{x}_{t_1}, \ldots, \boldsymbol{x}_{t_n}]$, $n < l$, and use this sequence in our simulations. Note that, there is no fine tuning on the under-sampling function. We observed similar results with the probabilistic models using different probability mass functions. For each simulation, we used the same number of hidden neurons for both LSTM architectures and set q to the original input size of the dataset, m. Note that, input size for the classical LSTM becomes $m+1$ since we extend the input vector with the time differences, i.e., LSTM-2 architecture.

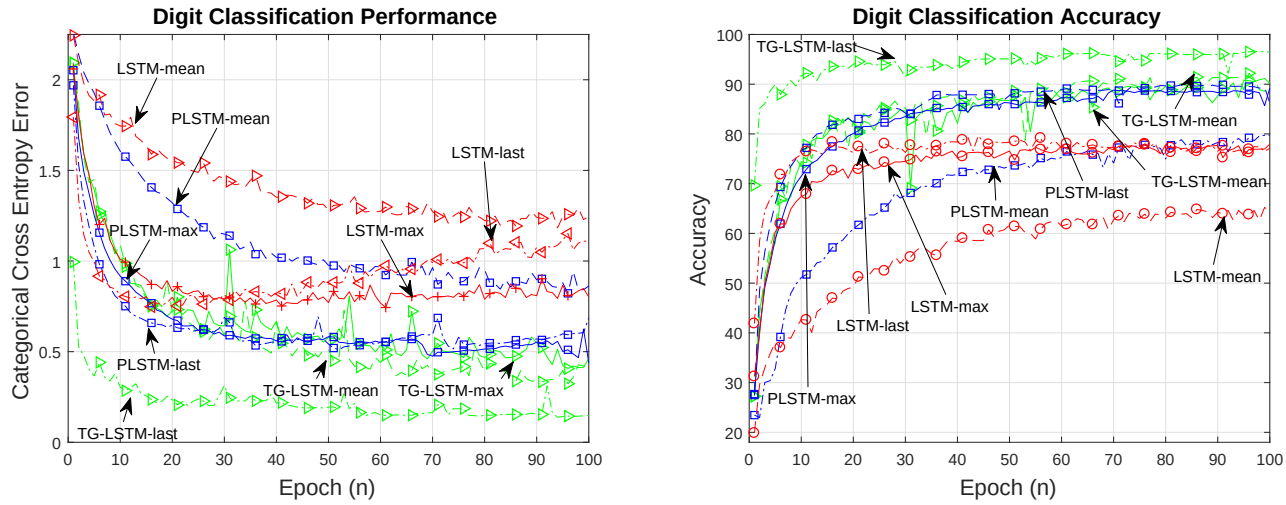
- Kinematic dataset is a simulation of 8-link all-revolute robotic arm, where the aim is to predict the distance of the effector from the target. The original input vector size $m = 8$ and we set the number of hidden neurons $q = 8$ for both LSTM and TG-LSTM networks. For the SGD algorithm, we select the constant learning rate $\eta = 10^{-5}$ from the interval $[10^{-6}, 10^{-3}]$ using the cross-validation.
- Bank dataset is generated by a simulator, which simulates the queues in banks. Our goal is to predict the fraction of the customers leaving the bank due to long queues. The input vector $\boldsymbol{x}_{t_k} \in \mathbb{R}^{32}$. We set the number of hidden neurons $q = 32$, and the constant learning rate $\eta = 10^{-5}$ from the interval $[10^{-6}, 10^{-3}]$.
- Pumadyn dataset is obtained from a simulation of Unimation Puma 560 robotic arm, which simulates the queues in banks. Our goal is to predict the angular acceleration of one of the arms. For this dataset, the input vector size $m = 32$ and we set the number of hidden neurons $q = 32$ for both TG-LSTM and LSTM networks. The constant learning rate is set as $\eta = 10^{-5}$ from the interval $[10^{-6}, 10^{-3}]$.

In Fig. 4, Fig. 5 and Fig.6, we illustrate the regression performances of the TG-LSTM and the classical LSTM architectures in terms of mean squared error per epoch for the kinematic, bank and pumadyn datasets, respectively. The TG-LSTM architecture has an outstandingly faster convergence rate compared to the classical LSTM architecture. In addition, the TG-LSTM architecture significantly outperforms the classical LSTM architecture in terms of the steady-state performance. These results show that the time gates, which incorporate the time differences as a nonlinear scaling factor, successfully model the effect of the non-uniform sampling. Both faster convergence and better steady-state performance are achieved by the TG-LSTM architecture. In these simulations no decaying factor is used for the learning rate of SGD algorithm since the architectures are able to converge. In the tasks, which requires a decaying factor for the convergence, the performance difference of the TG-LSTM and the classical LSTM architectures significantly increases since our algorithm has a faster convergence rate.

B. Classification Task

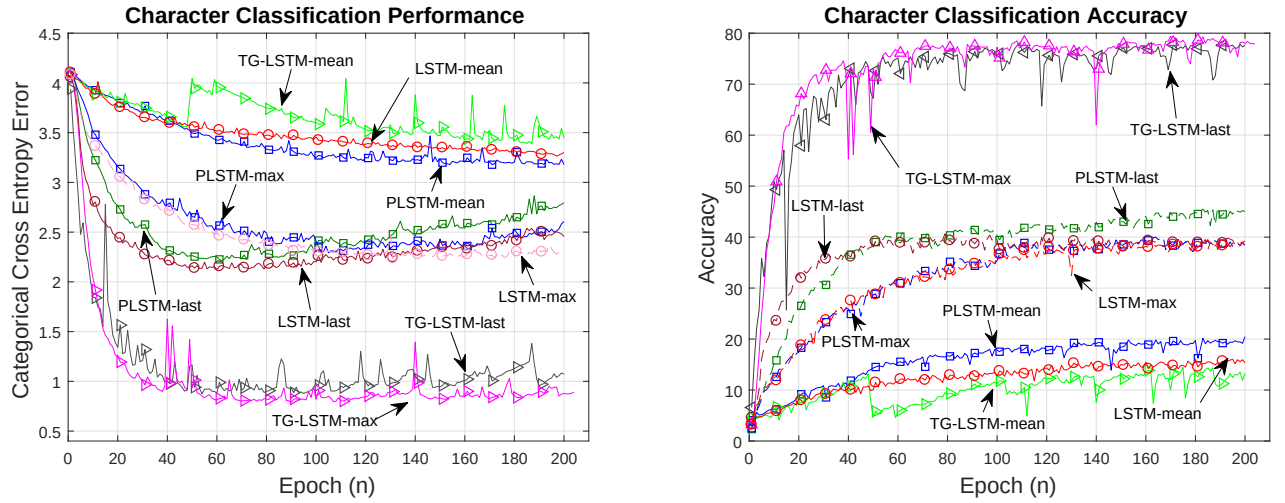
In this subsection, we evaluate the performances of the TG-LSTM, PLSTM [5] and the classical LSTM architectures for the classification tasks. For this task, we used the real life datasets Pen-Based Recognition of Handwritten Digits [29] and UJI Pen (Version 2) [29]. For the SGD algorithm, we use Adam optimizer [30] with the initial learning rate $\eta = 10^{-3}$.

In the first experiment, we demonstrate the classification performance of the LSTM architrectures on the Pen-Based Recognition of Handwritten Digits [29] dataset. This dataset contains handwritten digits from 44 different writers, where each writer draws 250 digits. These digits are drawn on a 500x500 tablet and uniformly sampled with 100 milliseconds. We non-uniformly under-sample these uniform samples by using (25). The input vector $\boldsymbol{x}_{t_k} = [x, y]^T$, where x and y



(a) **(b)**

Fig. 7: Classification performances based on (a) the categorical cross entropy error (b) the accuracy on the Pen-Based Recognition of Handwritten Digits [29] dataset.



(a) **(b)**

Fig. 8: Classification performances based on (a) the categorical cross entropy error (b) the accuracy on the UJI Pen (Version 2) [29] dataset.

are the coordinates, and the desired signal $d_{t_k} \in \{0, \ldots, 9\}$. For the parameter selection, we use 5-fold cross validation, and set the number of the hidden neurons $q = 100$, which is selected from the set $\{10, 25, 50, 100\}$. For the PLSTM architecture, all three parameters, i.e., the period, shift and on-ratio, are set as trainable to employ the network with full capacity. In Fig. 7, we illustrate the cross-entropy loss and the accuracy plots for the architectures with three different pooling methods. We observe from these figures that the TG-LSTM architecture outperforms both the PLSTM and the classical LSTM architectures. In particular, the TG-LSTM architecture using last pooling method significantly improves the performance for both convergence rate and the steady-state accuracy, which shows that the time gates in our method successfully model the the effect of the non-uniform sampling.

We also compare the performance of the architectures on the relatively more difficult dataset, UJI Pen (Version 2) [29]. This dataset is created by the same method with Pen-Based Recognition of Handwritten Digits [29] dataset. Although there are many other characters in the dataset, we used only upper-case and lower-case letters in the English alphabet, and the digits. The input vector $\boldsymbol{x}_{t_k} = [x, y]^T$, where x and y are the coordinates, and the desired signal $d_{t_k} \in \{a, \ldots, z, A, \ldots, Z, 1, \ldots, 9\}$, where we consider the digit "0" and the upper-case letter "O" as the same label. In this setup, we have 61 different labels, therefore, this a relatively difficult dataset. For all architectures, we set the number of the hidden neurons $q = 100$, which is selected from the set $\{10, 25, 50, 100\}$ by 5-fold cross validation. For the PLSTM architecture, all three parameters are trainable as in the first experiment. In Fig. 8, we illustrate the performance of the architectures in terms of the categorical cross entropy error and accuracy, respectively. We observe that the TG-LSTM architecture with max and last pooling methods significantly

improve the performance. Since the dataset is more difficult with 61 different classes, the performance increase is more observable in this simulation.

V. Conclusion

We studied nonlinear classification and regression problems on variable length non-uniformly sampled sequential data in a batch setting and introduced a novel LSTM architecture, namely, TG-LSTM. In the TG-LSTM architecture, we incorporate the sampling time information to enhance the performance for applications involving non-uniformly sampled sequential data. In particular, the input, forget and output gates of the LSTM architecture are scaled by these time gates using the sampling intervals. When the time intervals do not convey information related to the underlying task, our architecture simplifies to the vanilla LSTM architecture. The TG-LSTM architecture has a wide range of application areas since it can generate output at each time step as well as at the end of the input sequence unlike the other state of the art methods. We achieve significant performance gains in various applications, while our approach has nearly the same computational complexity with the classical LSTM architecture. In our simulations, covering several different classification and regression tasks, we demonstrate significant performance gains achieved by the introduced LSTM architecture with respect to the conventional LSTM architectures [11], [5] over several synthetic and real life datasets.

References

[1] J. J. Benedetto and H. C. Wu, "Nonuniform sampling and spiral mri reconstruction," in *Wavelet Applications in Signal and Image Processing VIII*, vol. 4119. International Society for Optics and Photonics, 2000, pp. 130–142.

[2] R. Vio, T. Strohmer, and W. Wamsteker, "On the reconstruction of irregularly sampled time series," *Publications of the Astronomical Society of the Pacific*, vol. 112, no. 767, p. 74, 2000.

[3] F. Eng and F. Gustafsson, "Algorithms for downsampling non-uniformly sampled data," in *Signal Processing Conference, 2007 15th European*. IEEE, 2007, pp. 1965–1969.

[4] D. Rastovic, "From non-markovian processes to stochastic real time control for tokamak plasma turbulence via artificial intelligence techniques," *Journal of Fusion Energy*, vol. 34, no. 2, pp. 207–215, Apr 2015. [Online]. Available: https://doi.org/10.1007/s10894-014-9791-5

[5] D. Neil, M. Pfeiffer, and S.-C. Liu, "Phased lstm: Accelerating recurrent network training for long or event-based sequences," in *Advances in Neural Information Processing Systems*, 2016, pp. 3882–3890.

[6] D. Rastovic, "Investigation of stabilities and instabilities at tokamak plasma behavior and machine learning with big data," vol. 1, pp. 1–5, 11 2017.

[7] L. Xie, Y. Liu, H. Yang, and F. Ding, "Modelling and identification for non-uniformly periodically sampled-data systems," *IET Control Theory & Applications*, vol. 4, no. 5, pp. 784–794, 2010.

[8] J. Sjölund, A. Eklund, E. Özarslan, and H. Knutsson, "Gaussian process regression can turn non-uniform and undersampled diffusion mri data into diffusion spectrum imaging," in *Biomedical Imaging (ISBI 2017), 2017 IEEE 14th International Symposium on*. IEEE, 2017, pp. 778–782.

[9] A. C. Singer, G. W. Wornell, and A. V. Oppenheim, "Nonlinear autoregressive modeling and estimation in the presence of noise," *Digital signal processing*, vol. 4, no. 4, pp. 207–221, 1994.

[10] D. Rastovic, "Tokamak design as one sustainable system," *Neural Network World*, vol. 21, no. 6, p. 493, 2011.

[11] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, 2017.

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

[13] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzell, "Learning to diagnose with lstm recurrent neural networks," *arXiv preprint arXiv:1511.03677*, 2015.

[14] Z. C. Lipton, D. C. Kale, and R. Wetzel, "Modeling missing data in clinical time series with rnns," *Machine Learning for Healthcare*, 2016.

[15] F. Rasheed and R. Alhajj, "Periodic pattern analysis of non-uniformly sampled stock market data," *Intelligent Data Analysis*, vol. 16, no. 6, pp. 993–1011, 2012.

[16] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying lstm to time series predictable through time-window approaches," in *Neural Nets WIRN Vietri-01*. Springer, 2002, pp. 193–200.

[17] M. Assaad, R. Boné, and H. Cardot, "A new boosting algorithm for improved time-series forecasting with recurrent neural networks," *Information Fusion*, vol. 9, no. 1, pp. 41–55, 2008.

[18] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[19] G. Cauwenberghs, "An analog vlsi recurrent neural network learning a continuous-time trajectory," *IEEE Transactions on Neural Networks*, vol. 7, no. 2, pp. 346–361, 1996.

[20] B. A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 263–269, 1989.

[21] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[22] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 3. IEEE, 2000, pp. 189–194.

[23] J. Pan and B. Hu, "Robust occlusion handling in object tracking," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.

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

[25] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[26] C. E. Rasmussen, R. M. Neal, G. Hinton, D. Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani, "Delve data sets." [Online]. Available: http://www.cs.toronto.edu/ delve/data/datasets.html

[27] L. Torgo, "Regression data sets." [Online]. Available: http://www.dcc.fc.up.pt/ ltorgo/Regression/DataSets.html

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

[29] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[30] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.