End-to-End Incomplete Time-Series Modeling From Linear Memory of Latent Variables

Qianli Ma, *Member, IEEE*, Sen Li, Lifeng Shen, Jiabing Wang, Jia Wei, Zhiwen Yu, *Senior Member, IEEE*, and Garrison W. Cottrell

***Abstract*—Time series with missing values (incomplete time series) are ubiquitous in real life on account of noise or malfunctioning sensors. Time-series imputation (replacing missing data) remains a challenge due to the potential for nonlinear dependence on concurrent and previous values of the time series. In this paper, we propose a novel framework for modeling incomplete time series, called a linear memory vector recurrent neural network (LIME-RNN), a recurrent neural network (RNN) with a learned linear combination of previous history states. The technique bears some similarity to residual networks and graph-based temporal dependency imputation. In particular, we introduce a linear memory vector [called the residual sum vector (RSV)] that integrates over previous hidden states of the RNN, and is used to fill in missing values. A new loss function is developed to train our model with time series in the presence of missing values in an end-to-end way. Our framework can handle imputation of both missing-at-random and consecutive missing inputs. Moreover, when conducting time-series prediction with missing values, LIME-RNN allows imputation and prediction simultaneously. We demonstrate the efficacy of the model via extensive experimental evaluation on univariate and multivariate time series, achieving state-of-the-art performance on synthetic and real-world data. The statistical results show that our model is significantly better than most existing time-series univariate or multivariate imputation methods.**

***Index Terms*—Missing values, recurrent neural networks (RNNs), temporal dependency, time-series imputation.**

I. Introduction

Time-series modeling is a central issue in a wide range of applications involving time-series prediction (TSP) [1]–[3] and time-series classification (TSC) [4]–[6], such as health care [7], action recognition [8], financial markets [9], and urban traffic control [10]. However, real-world time-series data inevitably contain missing values due to noise or malfunctioning sensors. Missing values make any kind of inference more difficult [11]. Therefore, most methods have to impute missing values before performing inference. Time-series imputation is a challenging task, since it is necessary to model temporal dependencies using incomplete data. In the case of missing a continuous segment of data, some form of long-term memory will be required. How to estimate these missing values is an active research topic.

Using a graph to model temporal dependencies between the missing item and its previously revealed points is an explicit and natural strategy for time-series imputation. A recent representative work is temporal-regularized matrix factorization (TRMF) [3], in which a graph-based temporal regularization is introduced to model temporal dependencies. These dependencies are further simplified into an autoregressive (AR) structure. For example, assuming the missing variable at time step t is x_t, the AR dependencies can be formulated as $x_t = \sum_{l\in\mathcal{L}} W^{(l)}x_{t-l}$, where the $W^{(l)}s$ are the weights between time steps at different lags l, and $\mathcal{L}$ denotes the lag set. Although TRMF demonstrated the effectiveness of graph-based modeling in time-series imputation, the graph-based dependency structure (such as $\mathcal{L}$) still requires manual design, and cannot capture complex dynamic correlations in an automatic way.

With the revolution in deep learning, recurrent neural networks (RNNs) have shown great potential for learning temporal dependencies within sequence data. However, the standard RNN is designed to model temporal dependencies from the complete data. Thus, the incomplete data are a challenge for the learning mechanism of RNNs. Simple methods,

Manuscript received September 15, 2018; revised November 29, 2018 and January 22, 2019; accepted March 17, 2019. Date of publication April 11, 2019; date of current version December 3, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61502174, Grant 61872148, Grant 61722205, Grant 61751205, Grant 61572199, and Grant 61751202, in part by the Natural Science Foundation of Guangdong Province under Grant 2017A030313355 and Grant 2017A030313358, in part by the Guangdong Province Higher Vocational Colleges and Schools Pearl River Scholar Funded Scheme (2018), in part by the Key Research and Development Program of Guangdong Province under Grant 2018B010107002, in part by the Guangzhou Science and Technology Planning Project under Grant 201704030051, in part by the Opening Project of Guangdong Province Key Laboratory of Big Data Analysis and Processing under Grant 2017014, in part by the Guangdong University of Finance and Economics Big Data and Educational Statistics Application Laboratory under Grant 2017WSYS001, and in part by the National Science Foundation, USA, to the Temporal Dynamics of Learning Center, an NSF Science of Learning Center under Grant SMA 1041755. This paper was recommended by Associate Editor N. Zhang. *(Corresponding author: Qianli Ma.)*

Q. Ma is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, also with the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, South China University of Technology, Guangzhou 510006, China, and also with the Guangdong Key Laboratory of Big Data Analysis and Processing, South China University of Technology, Guangzhou 510006, China (e-mail: qianlima@scut.edu.cn).

S. Li, L. Shen, J. Wang, J. Wei, and Z. Yu are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: awslee@foxmail.com; scuterlifeng@foxmail.com; jbwang@scut.edu.cn; csjwei@scut.edu.cn; zhwyu@scut.edu.cn).

G. W. Cottrell is with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093 USA (e-mail: gary@ucsd.edu).

This paper has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the author.

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

Digital Object Identifier 10.1109/TCYB.2019.2906426

such as replacing the missing values with their mean or the previously revealed values, introduces bias which could mislead the RNN in the task of TSP or TSC.

Recently, the residual network (ResNet) [12] has had a far-reaching impact for its simplicity and effectiveness. One-to-one connections of weight 1 between inputs and outputs of internal modules (or weights W when dimensionality must be changed between input and output) are used to copy information from previous hidden unit representations forward, so that the internal module only has to learn a residual between its input and output. During learning, the fixed one-to-one connections prevent the vanishing gradient problem.

Inspired by graph-based methods and ResNets, in this paper, we introduce a weighted linear memory vector into RNNs to address the problem of time-series imputation. We propose a novel end-to-end imputation framework we call the linear memory vector RNN (LIME-RNN). The RNN is trained for TSP, and we add a vector that is a weighted sum of all of the previous hidden states of the model. We call this the residual sum vector (RSV) in analogy to the residual connections in a ResNet. The weights of this memory are learned, and then it is used to impute missing values through a second weight matrix. The model is also inspired by graph-based models, as the weights between the RSVs allow information from the previous time steps to influence the imputation.

There are two major differences in this paper from previous graph-based models and ResNets: first, we use the RSV, a weighted sum of the previous hidden unit states of a predictive RNN as the regressor for the missing values, rather than previous elements of the time series itself. Second, we learn the residual connection weights using back propagation through time (BPTT) in an end-to-end way. To the best of our knowledge, this residual structure has not been previously considered as an approach to the imputation problem in TSP.

From the point of view of graph dependencies, the RSV directly integrates the information from its historical states via weighted paths, much as a weighted graph does. In this way, it can take full advantage of the previous observed information and reduce the negative impact of missing values. We develop a loss function to train the LIME-RNN with incomplete time series in an end-to-end way. The network learns to regress the values of the input variables when they are present, and when the values are missing, they are filled in from this regressor. In this way, the LIME-RNN can be trained with incomplete time series, simultaneously imputing the missing values and conducting TSP. Moreover, our framework is suitable for both "missing at random" data and consecutive missing data. The RNN itself can be a "vanilla" RNN [13], an LSTM network [14], or a GRU network [15]. We explore all three of these variants here.

Our contributions can be summarized as follows.

1) We unify the idea of a residual structure with the method of graph-based modeling of temporal dependencies in the proposed end-to-end framework LIME-RNN.
2) We introduce a loss function to train the LIME-RNN in the presence of missing values, which is applicable for both the random and consecutive missing data settings, and in univariate or multivariate time series. Moreover, when the task is prediction, the LIME-RNN can simultaneously achieve imputation and prediction.
3) The LIME-RNN is evaluated empirically on several synthetic and real-world time series, and the results show that our model obtains state-of-the-art imputation and prediction accuracy.

The remainder of this paper is organized as follows. Section II discusses related work on time-series imputation. Section III introduces the preliminaries on RNNs, graph-based temporal dependencies, and residual short paths. Section IV presents our method formally. Section V describes the detailed experimental settings and Section VI reports results and analysis. We conclude in Section VII.

II. Related Work

The demand for imputing missing data arises in many areas, giving rise to many relevant studies. Traditional time-series imputation methods, such as interpolation, splines, and moving averages (MAs), are commonly used to impute missing values in time series. All of them estimate the missing value from immediately preceding or succeeding values. Hence, they will achieve poor performance when encountering consecutive missing values. The expectation maximization (EM) algorithm [16] is also widely applied in dealing with missing values in time series. Sinopoli *et al.* [17] combined it with a Kalman filter. Oba *et al.* [18] combined it with PCA and variational Bayes methods. Both of them reconstruct the missing values by iterative EM steps over the available values.

Similar to the Kalman filter, Li *et al.* [19] proposed DynaMMo, using a sequence of latent variables to model the underlying linear dynamical system and hidden patterns of the observation sequences for multivariate time-series imputation. White *et al.* [20] proposed MICE, a sequential linear regression multivariate imputation method, in which the variable with a missing value is regressed on other available variables and draws from the corresponding posterior predictive distribution to replace the missing value. Anava *et al.* [21] used an AR model to address online TSP with missing values. In particular, they assume that the missing item can be represented as a recursive AR form of its previous nonmissing points and missing ones. However, all of them assume the time series has underlying linear dynamics, while nonlinear dynamics is more common in time series [22]–[24].

Recently, modeling temporal dependencies with graph-based regularization provides a new insight into time-series imputation. The aforementioned TRMF [3] employed a low-rank matrix factorization to deal with the correlation among multiple variables and further generalized the AR model as a weighted dependency graph-based regularizer to learn the temporal dependencies between nonmissing observations and missing values at different time steps, which allows for simultaneous imputation and prediction. However, TRMF is still limited to linear dependency with manually designed structures.

RNNs are suitable for modeling nonlinear temporal dependencies for both univariate and multivariate time series. However, conventional RNNs are based on sequential memory and cannot be trained in the presence of missing values. Although, Brakel *et al.* [25] presented a training strategy for time-series imputation, their method still required the guidance of ground truth in the training stage. Recently, Lipton *et al.* [26] used an RNN with an additional binary variable to indicate whether the value is missing or not, and set the missing value to zero when it is missing. This allowed them to train a recurrent network with missingness information, which was especially important in their medical domain. However, their use case was not filling in the missing variables. In medical data, a lack of data is actually useful information (e.g., that a test was not run). Che *et al.* [27] combined this indicator variable approach with learned decays for variables. However, the problem of RNNs being trained in the presence of missing values, where the values are missing at random (and in the case where their "missingness" is not informative), has not yet been addressed.

Our method is related to TRMF with a temporal dependency graph and RNNs. We address time-series imputation from the viewpoint of modeling graph dependencies with weighted residual short paths. Moreover, our framework is an end-to-end network, adopting a novel learning mechanism, which takes full advantage of the previous observed information of incomplete time series and reduces the negative impact of missing values to the memory in RNNs.

III. Preliminaries

To clarify the proposed method and make the paper more compact, we briefly introduce some basic concepts about RNNs, graph-based temporal dependencies, and residual short paths in this section.

A. Recurrent Neural Networks

RNNs are one of the most popular deep learning network structures. They are especially suitable for dealing with time and spatial correlation information because of their recursive processing of historical information and modeling historical memory. Some variants of RNNs, such as LSTMs and GRUs, are widely used in sequence-related scenarios. Given a sequential T-step time series $x = \{x_1, x_2, \ldots, x_T\}$, an RNN encodes it as a hidden representation $h = \{h_1, h_2, \ldots, h_T\}$, where the inputs are $x_t \in \mathbb{R}^n$, and the previous hidden state $h_{t-1} \in \mathbb{R}^m$. More specifically, RNNs generate h_t by the current input x_t and the previous hidden representation h_{t-1}

$$z_t = \mathbf{W_{in}} x_t + \mathbf{W_h} h_{t-1} + \mathbf{b_{hidden}} \quad (1)$$

$$h_t = f(z_t) \quad (2)$$

where z_t is an internal intermediate state and the model parameters are symbolized by $\mathbf{W_{in}}$, $\mathbf{W_h}$, and $\mathbf{b_{hidden}}$. f is a nonlinear transfer function [e.g., $f(\cdot)$ is the logistic, tanh, or ReLU]. Furthermore, we can simplify RNNs at time step t as an $\mathcal{F}_{\text{RNN}}$ function formulated by

$$h_t = \mathcal{F}_{\text{RNN}}(h_{t-1}, x_t; \mathbf{W}) \quad (3)$$

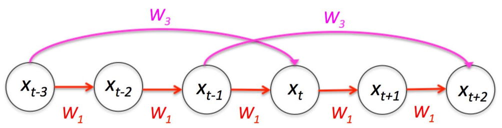


Fig. 1. Graph-based regularization for temporal dependencies illustrated in [3].

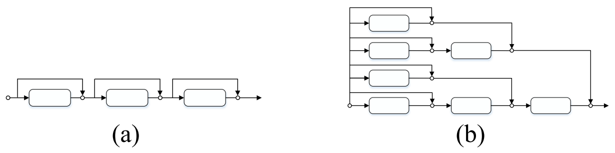


Fig. 2. As analyzed in [28], (a) ResNets enable very deep networks by leveraging the short paths shown in (b). There are $2^3 = 8$ short paths in a 3-block ResNet.

where $\mathbf{W}$ denotes all of the parameters. $\mathcal{F}_{\text{RNN}}$ encapsulates the different RNN variants.

B. Graph-Based Temporal Dependencies

Yu *et al.* [3] have elaborated on how to introduce graph-based regularization into conventional matrix factorization to capture temporal dependencies. These dependencies are simplified to an AR structure illustrated in Fig. 1. For example, assuming the missing variable at time step t is x_t, the AR dependencies can be formulated as

$$x_t = \sum_{l \in \mathcal{L}} W^{(l)} x_{t-l} \quad (4)$$

where W is the weights between different time steps and $\mathcal{L}$ denotes a lag set that needs to be manually specified.

C. Residual Short Paths

The ResNet [12] adds an identity mapping between the input and output of a module that allows the layer to just learn a residual difference between its input and output. This structure allows gradient information to flow backwards through the network, allowing very deep networks to be learned. Recently, Veit *et al.* [28] performed an enlightening analysis of ResNets, and they argued that a ResNet can be regarded as an ensemble of relatively shallow networks. As seen in Fig. 2, adapted from their paper, a 3-block ResNet is a collection of $2^3 = 8$ short paths with different lengths. In their view, with the structure of short paths, the flow of the gradient information can be efficiently propagated in this corresponding shallow network. This is the main reason why ResNets works so well.

IV. Proposed Methods

In this section, we present our new framework for time-series-related tasks with missing values. We first introduce our LIME-RNN framework. Then, we review how our RSV idea relates to the graph-based dependency framework.

A. Proposed LIME-RNN Framework

We define a time series of length T as $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\}$, where each $\boldsymbol{x}_t \in \mathbb{R}^n$. Since we are focusing on incomplete

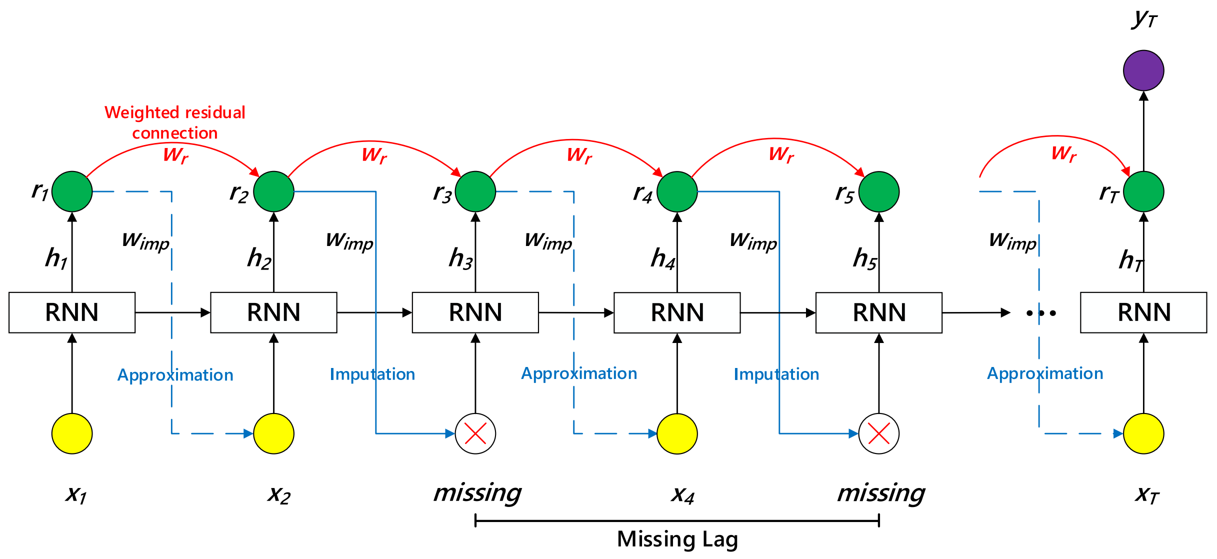


Fig. 3. Proposed LIME-RNN framework. We use green units to denote the RSV, yellow ones for input, purple for the task-related output, and the red "X" for missing inputs (randomly generated), which may be all or part of the input vector. In particular, when the missing lag drops to 1 (the missing lag is 2 in the above diagram), we have the challenging case of consecutive missing inputs. Experiments in this setting are presented in Section VI. The mapping from $\boldsymbol{r_T}$ to $\boldsymbol{y_T}$ may be linear, or a feedforward network.

time series, we require a corresponding set of indicator vectors, $\mathcal{M} = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_T\}$, $\boldsymbol{m}_t \in \{0, 1\}^n$, where $\boldsymbol{m}_{it} = 0$ indicates x_{it} is given, and $\boldsymbol{m}_{it} = 1$ indicates x_{it} is missing, where x_{it} is the ith component of the vector x_t. The framework of our LIME-RNN is shown in Fig. 3. Note that the RNN layer can be composed of any kind of RNN unit.

We introduce the RSV to the RNN for integrating the history information flow from hidden states, and draw it as the green units in Fig. 3. The output of RSV at time step t is an RSV denoted by $\boldsymbol{r}_t \in \mathbb{R}^m$ (i.e., the same dimension as h_t). $\boldsymbol{r}_t$ is defined in general as

$$\boldsymbol{r}_t = \begin{cases} f(\boldsymbol{h}_t) & \text{if } t = 1 \\ f(\boldsymbol{h}_t + g(\mathbf{W_r}\boldsymbol{r}_{t-1})) & \text{if } t = 2, 3 \ldots, T \end{cases} \tag{5}$$

where g and f are vector-valued functions, $\mathbf{W_r} \in \mathbb{R}^{m\times m}$, and $h_t \in \mathbb{R}^m$ denotes the output of the RNN hidden layer at time t.

However, in this paper, we assume f and g are the identity function, so we can write this simply as

$$\boldsymbol{r}_t = \begin{cases} \boldsymbol{h}_t & \text{if } t = 1 \\ \boldsymbol{h}_t + \mathbf{W_r}\boldsymbol{r}_{t-1} & \text{if } t = 2, 3 \ldots, T \end{cases} \tag{6}$$

which can be written in closed form as

$$\boldsymbol{r}_t = \sum_{i=0}^{t-1} (\mathbf{W}_r)^i \boldsymbol{h}_{t-i} \tag{7}$$

where we assume $(\mathbf{W}_r)^0 = \mathbf{I}$. Assuming the spectral radius of $\mathbf{W_r}$ is less than 1, this represents an exponentially decaying weighted history of the hidden unit vectors. Hence, we use a weighted transformation $\mathbf{W_r}$ in Fig. 3 instead of the identity structure used in ResNet [12]. Due to the weighted transformation, we can regard the residual short paths as a weighted graph model. More specifically, it takes into account the temporal dependencies of neighboring nodes, fusing RNNs's hidden states, and history information flow of the previous RSV with weighted residual connections.

We then learn to approximate the next input using a weighted sum of the RSV as follows:

$$z_{t+1} = \mathbf{W_{imp}}\boldsymbol{r}_t \tag{8}$$

where $\mathbf{W_{imp}} \in \mathbb{R}^{n\times m}$ is a learned transformation matrix, and z_{t+1} is trained to approximate x_{t+1} when present, and is used to impute it when missing.

A novel learning mechanism that guides LIME-RNN to take advantage of the previous observed information flow can be divided into two stages: 1) forward propagation and 2) error back propagation.

1) Forward Propagation: As seen in Fig. 3, two kinds of links enable LIME-RNN to directly model time series in the presence of missing values: 1) dashed blue links and 2) solid blue links. Our training process runs under two cases: 1) approximation and 2) imputation. Dashed blue links are for approximation and solid blue ones are for imputation. If the next input $\boldsymbol{x}_{t+1}$ is revealed, we train the output $\boldsymbol{z_{t+1}}$ of the RSV to approximate $\boldsymbol{x}_{t+1}$, aiming to model temporal dependencies between $\boldsymbol{x}_{t+1}$ (including the case of missing terms) and the history vector. When $\boldsymbol{x}_{t+1}$ is missing, we directly copy z_{t+1} to $\boldsymbol{x}_{t+1}$.

More formally, the input $\boldsymbol{u}_t$ to LIME-RNN is obtained by integrating the ground truth $\boldsymbol{x}_t$ and the imputed value z_t, as indicated by $\boldsymbol{m}_t$

$$\boldsymbol{u}_t = \boldsymbol{x}_t \circ \neg\boldsymbol{m_t} \,\oplus\, z_t \circ \boldsymbol{m_t} \tag{9}$$

where $\boldsymbol{m_t}$ is the indicator vectors as defined above, $\circ$ is the element-wise product, $\oplus$ is element-wise addition, and $\neg$ is the negation operator. This expression simply copies the elements of z_t to the locations of missing items in $\boldsymbol{x}_t$ during forward propagation.

2) Error Back Propagation: At each time step t, according to the existence of input $\boldsymbol{x}_t$ or not, the approximating loss $\mathcal{L}_{\mathbf{t\_approx}}$ is simply the squared error loss between the approximation and the existing values

$$\mathcal{L}_{\mathbf{t\_approx}}(z_t, \boldsymbol{x}_t) = \|(z_t - \boldsymbol{x}_t) \circ \neg \boldsymbol{m}_t\|_2^2. \tag{10}$$

Here, $\neg \boldsymbol{m}_t$ simply masks off missing data from the approximation loss.

Let the superscript k denote the kth sample of time-series collections ($k = 1, 2, \ldots, N$), and the overall training loss $\mathcal{L}_{\mathbf{total}}$ has two terms: 1) the total approximating loss term $\mathcal{L}_{\mathbf{total\_approx}}$ and 2) the task-related loss term $\mathcal{L}_{\mathbf{total\_target}}$

$$\mathcal{L}_{\mathbf{total\_approx}} = \sum_{k=1}^{N} \sum_{t=1}^{T-1} \underbrace{\mathcal{L}_{\mathbf{t\_approx}}\left(z_{t+1}^{(k)}, \boldsymbol{x}_{t+1}^{(k)}\right)}_{\text{Approximating Loss}} \tag{11}$$

$$\mathcal{L}_{\mathbf{total\_target}} = \sum_{k=1}^{N} \underbrace{\mathcal{L}_{\mathbf{target}}\left(\boldsymbol{d}^{(k)}, \boldsymbol{y}_T^{(k)}\right)}_{\text{Task-related Loss}} \tag{12}$$

where $d^{(k)}$ and $y_T^{(k)}$ denote the task-related target and output of the kth sample. The term $\mathcal{L}_{\mathbf{total\_target}}$ will depend on the task. For example, if the task is TSC, $\mathcal{L}_{\mathbf{target}}$ will be cross-entropy loss.

Therefore, the overall training loss $\mathcal{L}_{\mathbf{total}}$ is obtained by combining (11) and (12)

$$\mathcal{L}_{\mathbf{total}} = \mathcal{L}_{\mathbf{total\_approx}} + \lambda_{\text{target}} \mathcal{L}_{\mathbf{total\_target}} \tag{13}$$

where λ_{target} is a coefficient weighting the importance of the task loss (we will usually simply set this to 1). This loss function can be optimized by the standard BPTT algorithm.

Finally, the network update of the hidden unit activations can be described by using the unified input form of $\boldsymbol{u}_t$

$$\boldsymbol{h}_t = f(\mathbf{W_{in}} \boldsymbol{u}_t + \mathbf{W_h} \boldsymbol{h}_{t-1} + \mathbf{b_{hidden}}) \tag{14}$$

where $\mathbf{W_{in}}$, $\mathbf{W_h}$, and $\mathbf{b_{hidden}}$ denote the framework parameters and f is the activation function. Furthermore, a generalized function $\mathcal{F}_{\text{RNN}}$ is used to summarize our framework

$$\boldsymbol{h}_t = \mathcal{F}_{\text{RNN}}(\boldsymbol{h}_{t-1}, \boldsymbol{u}_t; \mathbf{W}). \tag{15}$$

Here, we are purposely being agnostic about how the output $\boldsymbol{y}_T$ is computed. There can be any pathway from $\boldsymbol{h}_T$ and/or $\boldsymbol{r}_T$ to $\boldsymbol{y}_T$—a simple learned weight matrix or a feedforward network, which would provide a pathway for error propagation due to the target loss. With the end-to-end loss function of (13) and the generalized recursive update form of (15), our framework models incomplete time series from the residual information flow for missing items (represented here by $\boldsymbol{u}_t$) and the hidden state, which can be trained in an end-to-end manner in the presence of missing values. Furthermore, when the task is prediction of the next input, the model can simultaneously achieve both imputation and prediction.

B. Discussion: Temporal Dependencies and Residual Short Paths in LIME-RNN

LIME-RNN combines the merits of graph-based models with explicitly modeled temporal dependencies via weighted residual connection between nodes, with the merits of RNNs that can accumulate historical residual information and learn the underlying patterns of incomplete time series automatically. Compared to other general graph methods (such as [3], shown in Fig. 1), our LIME-RNN has several advantages.

1) The dependency between input variables can be nonlinear, and is mediated in our model by the learned hidden unit representation through z_t.
2) The temporal dependency graph in LIME-RNN considers all direct connections among hidden variables [e.g., given K previous points, the number of residual short paths is 2^K, as in Fig. 2(b)], which avoids the handcrafted design of the dependency structure.
3) These residual short paths in LIME-RNN can be automatically learned in an end-to-end way using BPTT, which does not limit the system to some set of user-intuited assumptions, such as the dependency length (delay) in autoregression (AR).

V. Experimental Details

In this section, we conduct a comparison between our framework and several state-of-the-art methods for time-series imputation of missing values, and empirically evaluate the performance of different RNN unit types in the LIME-RNN model. First, we introduce the datasets. Second, we describe our data-preprocessing procedures, experimental details, and the comparison methods. Third, we show the experimental results on univariate and multivariate time series. Then, we visualize the imputation results of more realistic cases where data are consecutively missing on both univariate and multivariate datasets. Finally, we investigate the effects of hyper-parameters, investigate error accumulation on a particular dataset, and, finally, report the runtime of our algorithm.

A. Datasets

The datasets we use are summarized in Table I, and include five univariate time series and three multivariate ones. The datasets are as follows.

1) *Sanity check* [21] is a synthetic time series generated from a fifth-order AR equation: $\mathbf{x}_t = \phi_0 + \sum_{i=1}^{5} \phi_i \mathbf{x}_{t-i} + \epsilon_t$, where ϕ_0 and $\{\phi_i\}(i \in 1, \ldots, 5)$ are set to 0, 0.6, -0.5, 0.4, -0.4, and 0.3, respectively. The noise terms $\{\epsilon_t\}$ are sampled from a distribution of $\mathcal{N}(0, 0.3^2)$. The first five points $\{\mathbf{x}_i\}(i \in 1, 2, \ldots, 5)$ are initialized by 1, 2, 3, 4, and 5, respectively.
2) *Monthly temperature* [29] is a real-world univariate time series of the average monthly temperature in England between January 1723 and December 1970.
3) *Daily births* [30] is a time series of the number of daily births in Quebec from January 1977 to December 1990.
4) *ElectricityLoadDiagrams* [31] records clients' electricity consumption every 15 min with 140 256 data points. We select one of the client's data as a univariate time series and downsample to 17 536 points, called `Electricity_MT124`.
5) *Ozone concentration* [32] collects the monthly Ozone concentration data from the city of Azusa, CA, USA, from 1956 to 1970.

TABLE I
EXPERIMENTAL DATASET AND IMPLEMENTATION DETAILS

Data set	Dimensionality	Length	Preprocessing	Data source	Missing type	Missing rate	ILR/MBS	T
Sanity check	univariate(1)	496	mean	Synthetic	random	from 5% to 50%, in increments of 5%.	0.015/1	10
Monthly temperature		2976	min-max	Real world			0.01/8	19
Daily births		5113	min-max				0.01/8	19
Electricity_MT124		17536	min-max				0.01/8	24
Ozone concentration		180	mean		random & consecutive	16.0%	0.015/1	10
DSIM	multivariate(16)	1440	min-max	Synthetic	random	from 5% to 50%, in increments of 5%	0.015/4	9
SCITOS G5	multivariate(24)	5456	min-max	Real world			0.01/4	12
Traffic volume	multivariate(10)	4272	smoothing		consecutive	10.0%	0.015/4	36

6) *DSIM* [33] is a simulated diabetes multivariate dataset. 16-D data with additive Gaussian noise is generated for each simulated minute, yielding 1440 data points.
7) *SCITOS G5* [34] is a real-world dataset, which consists of the measurements of the 24 ultrasound sensors of a SCITOS G5 robot navigating a room. The 5456 sensor readings were sampled at a rate of 9 Hz as the robot was following the wall of the room in a clockwise direction, making four trips around the room.
8) *Traffic volume* is a real-world Traffic volume dataset collected from ten stations in the freeway network in a province of China. Each station records flow every 5 min from February to April, resulting in 25 632 records. Considering that the ten stations are in the same highway network, we treat it as a multivariate time series and downsample the recording interval every 30 min, obtaining 4272 10-D records.

Each dataset is preprocessed as shown in Table I. We either subtract the mean value of the entire dataset or normalize the raw data by a linear transformation using the maximum and minimum (min–max normalization) [33] of the dataset. For the Traffic volume dataset, we smooth the data using a 5-step sliding-average smoothing (also called rectangular boxcar smoothing) [35], [36], that is, each point is replaced by the mean of the two points before and after it and the point itself.

B. Creating Missing Data

As shown in Table I, we implemented two methods of creating missing values: 1) missing at random and 2) consecutive missing. Missing at random simply means we randomly remove some data (at the level of an individual component of a data vector in the multivariate case) with a certain probability (the missing rate in Table I). For comparison purposes, we use the same missing rates as in [3], [19], and [33]. Note that with high missing rates (e.g., 50%), there will be frequent occurrences of consecutive missing data as well.

Consecutive missing means that we remove a sequence of data points, which is often more realistic, mimicking device failure or lost records of a traffic toll-collection station. First, we set a fixed length L for each missing section. For Ozone, the length is 12, for Traffic volume, the length is 24 (this corresponds to half of the day). Based on the proportion of the given missing values and the given length L, we calculate the corresponding number of segments N_{seg}. We then randomly choose N_{seg} starting points and remove those from the sequence, ensuring that no missing segments overlap.

As noted in Table I, for Ozone, we used both random and consecutive missing. We removed one segment of length 12 and then 13 missing at random points from the first 156 data points, while holding out the last 24 points to evaluate prediction performance. We use Ozone to visualize the results on a univariate dataset (see Fig. 7).

C. Tasks

On the univariate datasets, we conduct both the imputation and prediction tasks. We divide the time series into two parts: the first 70% for training and the remaining 30% for testing prediction accuracy. We do not use a holdout set, but train until the error flattens out. Since our model is trained to impute missing data based on the data that are not missing, when we train the model, we can leave data out, and give the results of the imputation on the missing data in the training set. In this manner, we can measure performance on the training set, since the ground truth for the missing data is not used in training. Thus, we report the imputation error on the training set while focusing on the prediction performance on the test set.

On the multivariate datasets and with the following previous work [3], [19], [33], we only conduct the imputation task and compute the imputation performance on the entire dataset.

D. LIME-RNN Implementation Details

For the LIME implementation, we used a single-layer RNN with 128 neurons. We tried all three recurrent unit types: vanilla RNN, GRU, and LSTM. For brevity, we only report the results of LIME-LSTM. LIME-GRU performed comparably to LIME-LSTM, and occasionally better, but only by small amounts. LIME-vRNN performed significantly worse than the other two. The Nemenyi test (statistical test) [37] on all three recurrent unit types is conducted in Section VI.

The initial learning rates (ILR) and mini-batch size (MBS) are shown in the penultimate column of Table I. The learning rates are annealed during training. Initial weights were uniformly distributed in the range [−0.1, 0.1]. For training, we used the Adam optimizer [38] and stopped training when the loss flattens out. The coefficient λ_{target} of (13) is set to 1 for the goal of one-step-ahead prediction; this is identical to the approximation loss, and both are trained to estimate the next input, $\boldsymbol{x}_{t+1}$. In the one-step-ahead prediction task, we must slice the time series into fixed length subsequences; hence, we have to choose T, the length of the subsequences (as in Fig. 3). The choice of T is shown in the last column of Table I, and the relevant details are discussed in Section VI-D (hyper-parameter analysis). These subsequences overlap, which is commonly called "many-to-one" training. For consistency, we apply this training technique to all RNNs, including both univariate and multivariate data.

The experiments were run on the tensorflow platform using an Intel Core i5-6500, 3.20-GHz CPU with 32-GB RAM, and a GeForce GTX 980-Ti 6G. All of the following experiments are repeated five times with different initial random weights and their average results reported.

E. Comparison Methods

Most imputation algorithms are designed for interattribute correlations, while univariate time series only have one attribute. Therefore, not all algorithms apply to both univariate and multivariate imputation. We divide the comparison techniques into univariate methods and multivariate ones, and introduce them separately.

1) Univariate Methods: Here, we compare LIME-RNN with six representative time-series imputation methods, on both imputation and prediction tasks. We include the prediction task because if we compared the methods solely on imputation, the efficacy of the method in the service of a task would not be apparent. However, the imputation methods described below do not have a mechanism for prediction, so they need to be combined with predictors, such as ARMA [39] or LSTM.

For each dataset, we use the same predictor for each method, but different datasets will use different predictors depending on the characteristics of the dataset. In particular, for `Sanity check`, we use ARMA as the predictor since the time series is derived from the AR equation. For `Monthly temperature` with strong seasonal patterns, we replace ARMA with seasonal ARIMA (SARIMA) [40]. For `Daily births`, `Electricity_MT124`, and `Ozone`, we use an LSTM network as the predictor due to the nonlinearity of these datasets.

The six imputation methods for univariate time series are as follows.

1) *Forward Imputation:* This method simply replaces the missing value with its last observed value.[1]
2) *Indicator Variable Approach [26]:* This method adds a Boolean indicator variable at each time step that is 1 if the data are missing, and 0 otherwise. As in the work of [26], missing values are set to 0.
3) *Spline Imputation [41]:* Missing values are estimated with spline interpolation.
4) *MA Imputation [42]:* Missing values are replaced with the mean of the values in a window around the missing value. For example, a missing value at time t will be imputed by the mean of observations at $t-2$, $t-1$, $t+1$, and $t+2$ (assuming the window size is 2).
5) *Regularized EM (RegEM) Imputation [43]:* This is a regularized variant of the EM algorithm for imputation.
6) *Kalman Imputation [17]:* This algorithm assumes an underlying structural time-series model and employs the Kalman filter to estimate its parameters. The estimation is done by maximizing the log-likelihood using iterative steps, and the resulting Kalman smoothed estimator is used to complete the missing values.

[1]There are two other simple imputation methods, zero and mean imputation. Since their overall performance is similar to forward imputation, we omit them for brevity.

The imputation methods Spline, MA, and Kalman were implemented using the imputeTS toolkit in R [44].

2) Multivariate Methods: We compare seven representative multivariate imputation methods with our model. These include the following techniques.

1) *BPCA [18]:* This method jointly conducts PCA regression, Bayesian estimation, and EM learning to fill missing values with estimated values.
2) *MICE [20]:* This method is sequential regression multivariate imputation, in which the variable with missing values is regressed on other available variables, and draws from the corresponding posterior predictive distribution to replace the missing value.
3) *Fourier [33]:* This method is based on the Fourier transform, and uses the past values of each variable to impute each missing value.
4) *Lagged k-Nearest Neighbors (Lk-NN) [33]:* This method is based on the k-nearest neighbor imputation (k-NNI) [45]. They first obtain the top-k neighbors by calculating the Euclidean distance between the remaining available attributes with other complete inputs. Then, they impute the missing item with the average (or weighted average) of the corresponding item of the top-k neighbors. This paper also introduces time-lagged correlations between variables.
5) *FL k-NN [33]:* This is an ensemble method that combines Lk-NN and the Fourier transform.
6) *Dynammo [19]:* This method is based on EM and the Kalman filter. It learns a linear dynamical system in the presence of missing values and imputes them.
7) *TRMF [3]:* This is a recently proposed framework for multivariate time-series imputation and prediction, which introduces temporal graph regularization into matrix factorization.

We construct a baseline method called H-LSTM to verify the effectiveness of the graph-based residual paths. H-LSTM learns a matrix $\mathbf{W_{imp}}$ as in LIME-RNN, but directly tries to predict the next input value $\boldsymbol{x}_{t+1}$ from the hidden unit activations at time t

$$z_{t+1} = \mathbf{W_{imp}} \boldsymbol{h}_t \tag{16}$$

where $\mathbf{W_{imp}} \in \mathbb{R}^{n\times m}$ is a learned transformation matrix. This should be compared to (8). This method uses the same end-to-end loss function of (13). Finally, to further verify the benefit of the graph-based residual paths versus simply skipping connections, we also constructed a baseline called RH-LSTM, which introduces skip connections into H-LSTM. This strategy was proposed by Wang and Tian [46].

We evaluate the results by using root mean square error (RMSE) for all time series except for `DSIM`. In order to directly compare with the published results, we use mean absolute error (MAE) for `DSIM` [33]

$$\text{RMSE} = \sqrt{\frac{\sum_{i\in\Omega} (x_i - \widehat{x_i})^2}{|\Omega|}} \tag{17}$$

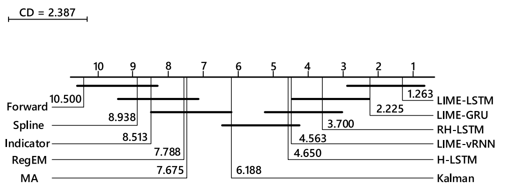


Fig. 4. Critical difference diagram for our framework on univariate imputation with other algorithms using the Nemenyi test. The bold solid horizontal lines group algorithms into cliques, within which there is no significant difference in rank at a 0.05 significance level.

$$\mathrm{MAE} = \frac{\sum_{i\in\Omega} |x_i - \widehat{x_i}|}{|\Omega|} \tag{18}$$

where Ω denotes the index set of the missing values and $|\cdot|$ denotes its size. x_i and $\widehat{x_i}$ are the ground truth and imputed value of the ith missing item, respectively. Similarly, we use RMSE to evaluate the prediction performance.

VI. Results and Analysis

We first give quantitative results comparing our algorithm on imputation and prediction on the univariate datasets, and then imputation on the multivariate datasets. In general, we find our algorithm is state of the art. Following this, for qualitative comparison, we provide visualizations of the performance of our algorithm and the competing algorithms on these tasks on the Ozone (univariate) and Traffic (multivariate) datasets. We then examine the effect of hyperparameters on our algorithm. We also show how our algorithm is able to recover from errors, and give an empirical comparison of the runtime of our algorithm against a standard LSTM network.

A. Performance Comparison on Univariate Datasets

The imputation performance of each algorithm on the univariate datasets with different missing rates is reported in Table I of the supplementary material (due to space constraints, we only show statistical results here). As shown, LIME-LSTM outperforms the other algorithms on all but 3 of the 40 examples, and is second best on those. LIME-LSTM is better than RH-LSTM, showing that the RSV is indeed more effective than simply adding skip connections. In addition, LIME-LSTM is superior to H-LSTM, showing that the RSV is more effective than simply learning a mapping from the hidden unit vector at the previous time step. On the other hand, H-LSTM does provide a strong baseline, and demonstrates the effectiveness of our end-to-end loss function.

In addition, the Nemenyi test (a nonparametric statistical test) [37] was conducted on the rank-order performance of the algorithms. Fig. 4 shows the critical difference diagram. In this diagram, groups of algorithms that are not significantly different at the $p \leq 0.05$ level in rank are connected. As mentioned above, we also tried our method with GRU and vanilla RNN units. The statistical analysis shows that the LSTM and GRU versions of our algorithm are not statistically significantly different in this experiment, while LIME-LSTM is superior to all others, including RH-LSTM, H-LSTM, and LIME-vRNN. We conclude that at this significance level (the critical difference is 2.142), our method is clearly the winner. The closest competitor that is not one of our models is the Kalman filter method, which is not statistically different than the RegEM or MA method. According to the ranking, the forward imputation, spline, and indicator methods performed worst.

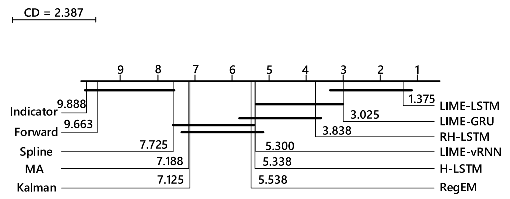


Fig. 5. Critical difference diagram of one-step-ahead prediction for our framework and the other algorithms on the Nemenyi test. The significance level is 0.05.

As mentioned in Section V-E, after performing imputation, we then combined these methods with suitable predictors to perform one-step-ahead prediction on the remaining 30% of datasets. The performance is shown in Table II of the supplementary material. Our method gives the best results in every case except one, and is second best on that. To check that this was not by chance, we again performed the Nemenyi test to obtain a statistical comparison of the methods.

The statistical results are shown in Fig. 5. Again, we find that the LSTM and GRU performance are not significantly different. While RH-LSTM, LIME-vRNN, and H-LSTM are not significantly different in rank from LIME-GRU, LIME-LSTM is better at the $p \leq 0.05$ level. Again, the forward imputation, spline, and indicator variable methods performed worst. The connection between forward imputation and the indicator variable approach shows that although the latter adds an additional variable indicating whether the current value is missing, experimental results show that this variable provides little gain on the prediction task. This makes sense in this context, as the indicator variable was actually informative in the medical setting where it was used by Lipton *et al.*, and there is no reason to believe it will be informative here, where values are missing at random.

B. Performance Comparison on Multivariate Datasets

Following the previous research, we performed imputation on the entire DSIM and SCITOS G5 datasets. Table III of the supplementary material shows the results. The results of BPCA, MICE, RegEM, Fourier, L*k*-NN, and FL*k*-NN on DSIM were taken from [33]. In other cases, the results are obtained by running the original authors' source code on the data. Again, our method achieves the best results on 17 out of the 20 comparisons, and performs better as the missing rate increases. TRMF performs poorly because it relies on traditional matrix factorization techniques, which are more suitable for the linear case, while here we have nonlinear data. DynaMMo is better on SCITOS when the missing rate is 5%–15%, but LIME-LSTM is not far behind.

We performed the same statistical test (Nemenyi test) as before. The results are shown in Fig. 6. The performance of two instances of our framework, LIME-LSTM and LIME-GRU, are similar and ranked 1 and 2, respectively. This indicates that they have similar modeling capabilities for univariate

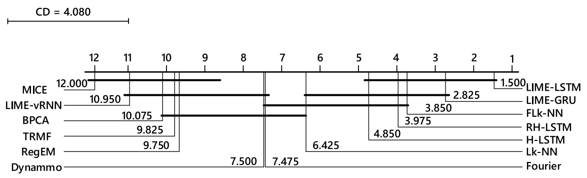


Fig. 6. Critical difference diagram of multivariate imputation for our framework and the other algorithms on the Nemenyi test. The significance level is 0.05.

and multivariate imputation while, as before, LIME-vRNN shows inferior performance.

In this analysis, however, while our results are numerically superior, FL*k*-NN, RH-LSTM, and H-LSTM are not significantly different from LIME-LSTM or LIME-GRU. It is worth examining FL*k*-NN a little more closely. It actually uses two methods: 1) Fourier and 2) L*k*-NN. The Fourier transform is fit to the available variables, and then the inverse Fourier transform is used to impute the missing values.

L*k*-NN is a lagged *k*-nearest neighbors algorithm, where the lag is determined by the highest correlations with the missing variables. This part of the algorithm has two parameters that must be chosen by hand: k is the number of nearest neighbors and p is the number of time lags. The algorithm takes the p lags with the strongest correlation for each pair of variables, and the k nearest neighbors across all lags (weighted by the strength of the correlation), and averages the results. The results of these two methods are then averaged. This is a complex method compared to ours, with a number of hand-chosen parameters. Our method is clearly more elegant, and numerically outperforms this method.

RH-LSTM and H-LSTM, while not significantly different from the LIME models at the $p \leq 0.05$ level, show the importance of learning representations to predict missing variables, and suggest that deep networks are better able to take advantage of interactions between components of a multivariate time series than the remaining methods.

C. Imputation Visualization

In this section, we provide a visualization of our model's performance compared with the other methods for both a univariate case (Ozone) and a multivariate case (Traffic). Here, we consider the setting in which the data are consecutively missing, which is closer to what is encountered in real-world data. In the following discussion, we compare the results based on one instance of LIME-LSTM.

1) Univariate Results: In Fig. 7, we visualize the imputation and prediction results of six methods on the `Ozone concentration` dataset. As noted above, there are 13 randomly missing points, as well as a section of 12 consecutive missing data points, starting around the 58th time step. The data shown in the figure start after the first 40 points in the sequence, because there are no missing data in that section of the time series.

As shown in Fig. 7(f), LIME-LSTM achieves excellent imputation and prediction performance. The imputed values (red points) are located on the original curves. Due to the end-to-end loss function, H-LSTM also works well [Fig. 7(e)], but does not quite reach the peak of the consecutively missing data around time step 60. The imputation performance of the remaining algorithms is poor for the consecutively missing section of the data. None of them are able to capture the fact that there should be a peak here. The reason is that these algorithms impute the missing values from immediately preceding and succeeding values, which are not available in the case of consecutively missing data. Also, the existing previous and succeeding values are both low, so the imputed values by these algorithms are also low. This is especially evident in the Kalman model's imputation [Fig. 7(d)].

The pink shaded region of Fig. 7 shows the prediction performance of the algorithms. All use an LSTM network for the prediction portion. As such, all work reasonably well, but H-LSTM and LIME-LSTM predict curves that are smoother than the other methods. Between the two, LIME-LSTM is more accurate at predicting peaks and valleys. It is clear here that these methods would also benefit from using such a model to impute consecutively missing data, as this would clearly have helped around time step 58.

2) Multivariate Results: We visualize the imputation results of six different methods on the `Traffic volume` dataset, in Fig. 8. The data contain 18 consecutive missing blocks of length 24. This dataset is 10-D, so here we just show the results on one of the ten variables. The others are similar.

To show the differences between the methods clearly, we have zoomed in on the imputation results of the pink region. DynaMMo and TRMF completely fail in this setting. TRMF models the missing values in two ways: first, using correlations between the ten variables at the same time step, and second, by the dependency graph regularization. Since the values of all ten variables at the same time interval are deleted, there is nothing to correlate with (this also causes the failure of DynaMMo). Moreover, since the missing time interval is relatively large, the dependency graph regularization is not effective. The Fourier approach uses past values to impute missing values; FLk-NN combines this with using correlations between variables. As a result, these two algorithms essentially repeat the previous values, just shifting some number of time steps, as can be seen in Fig. 8(c) and (d). This completely distorts the original characteristics of the data. Since both H-LSTM and LIME-LSTM are learning to model the dynamics of the time series, they perform much better, with a small offset from the missing data. However, H-LSTM fails to maintain the dynamics across the entire interval in the pink region of Fig. 8(e). On the other hand, as seen in Fig. 8(f), LIME-LSTM is superior to the other methods both in accuracy and stability. The H-LSTM models the temporal dependencies via the hidden unit state in the LSTM, while the LIME-LSTM model is able to use the longer-term history, as captured by the RSV, similar to the residual-short-path structure in a ResNet.

D. Hyper-Parameter Analysis

The above results used a fixed hyper-parameter setting of a single layer of 128 neurons. Here, we perform an empirical

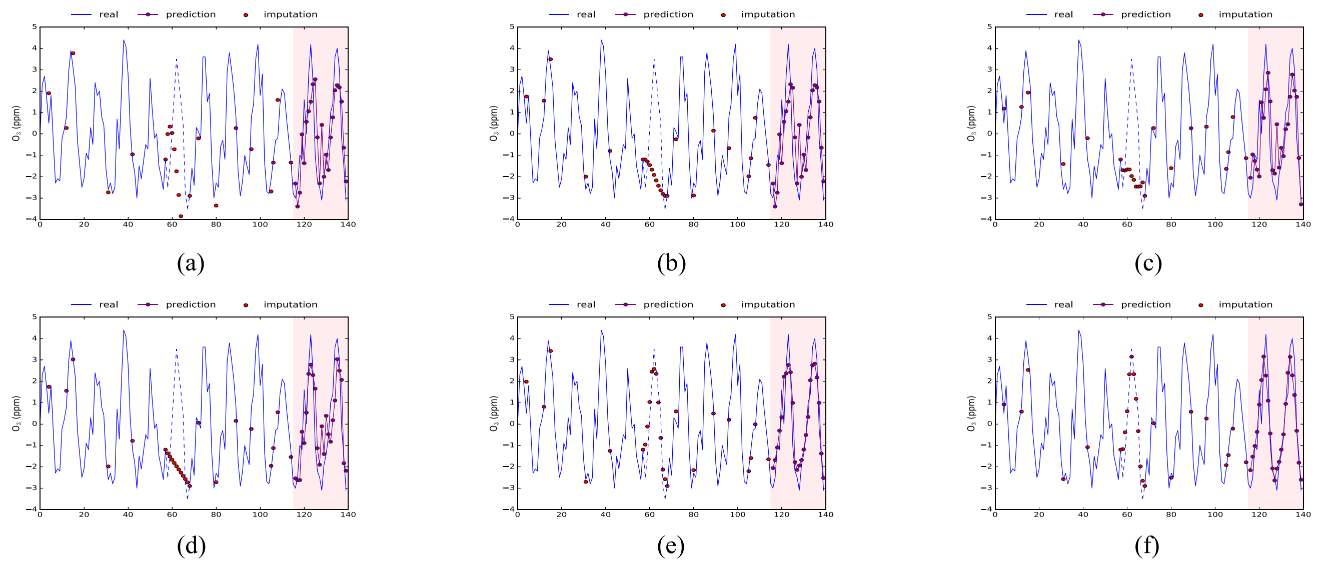


Fig. 7. Visualization results of imputation and prediction on the univariate Ozone concentration dataset (after mean subtraction). The dashed lines (around step 60) denote consecutively missing cases. The predicted values are in the pink region. (a) Spline + LSTM. (b) RegEM + LSTM. (c) MA + LSTM. (d) Kalman + LSTM. (e) H-LSTM. (f) LIME-LSTM.

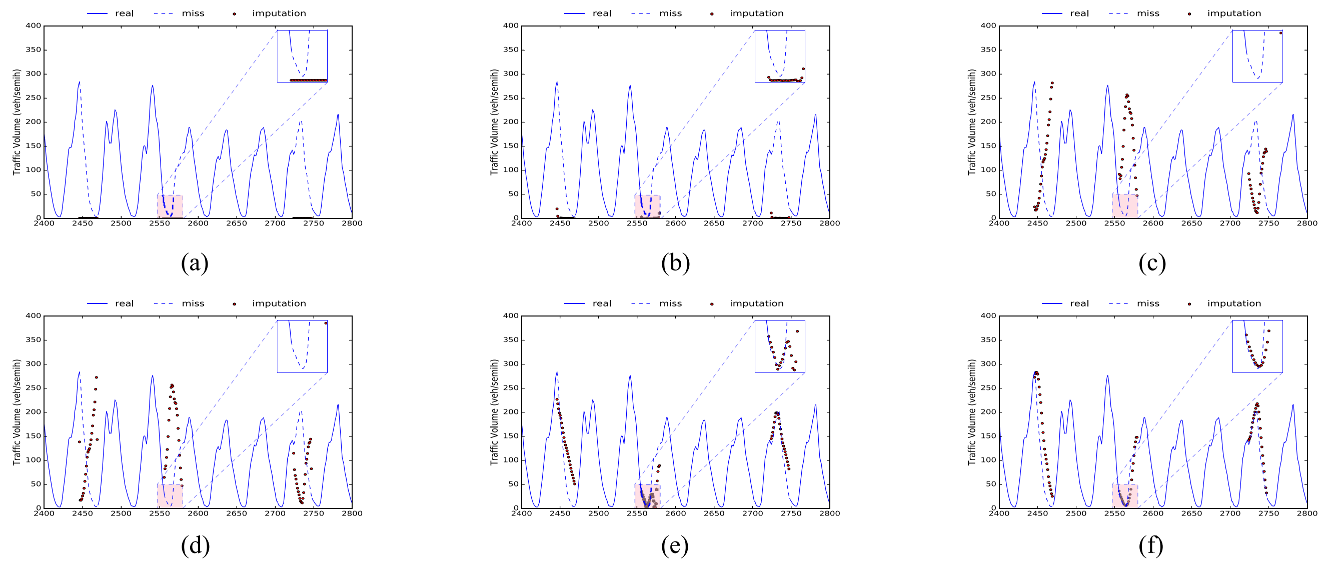


Fig. 8. Plots of one dimension imputed by different methods on the multivariate Traffic volume dataset (after smooth). The dashed lines denote consecutively missing cases. We zoom in on the imputed values in the pink region. (a) Dynammo. (b) TRMF. (c) Fourier. (d) FL*k*NN. (e) H-LSTM. (f) LIME-LSTM.

evaluation of the effects of two hyper-parameters: 1) the number of hidden units and 2) the number of recurrent layers. We perform this evaluation using the imputation task on four of the time series (with missing rates shown in parentheses in Fig. 9) to demonstrate why the particular hyper-parameter setting was used. We do the evaluation by varying one parameter at a time while maintaining the other parameters fixed.

As shown in the first row of Fig. 9, the influence of the number of hidden units on the imputation performance is slight. As shown in the second row, adding additional recurrent layers quickly leads to overfitting. As shown in the third row, for each time series, there is a "sweet spot" for T that is relatively stable across a range of values. If the time slice is too short, there may be not enough history in the RSV for the network to pick up on. If the time slice is too long, we hypothesize that the poorer results are due to the exploding gradient problem. For stable training, we have to clip the maximum norm of the gradient to 10, following common practice. As a result of these analyses, we used one layer of 128 neurons, while making T as small as possible while maintaining performance.

E. Error Accumulation Analysis

Inevitably, the missing values imputed by the network should yield errors. Here, we argue that these errors will not accumulate quickly along the forward propagation since the network is able to recover from its previous errors. Assume that x_t is missing and x_{t+1} is revealed in our model, we feed

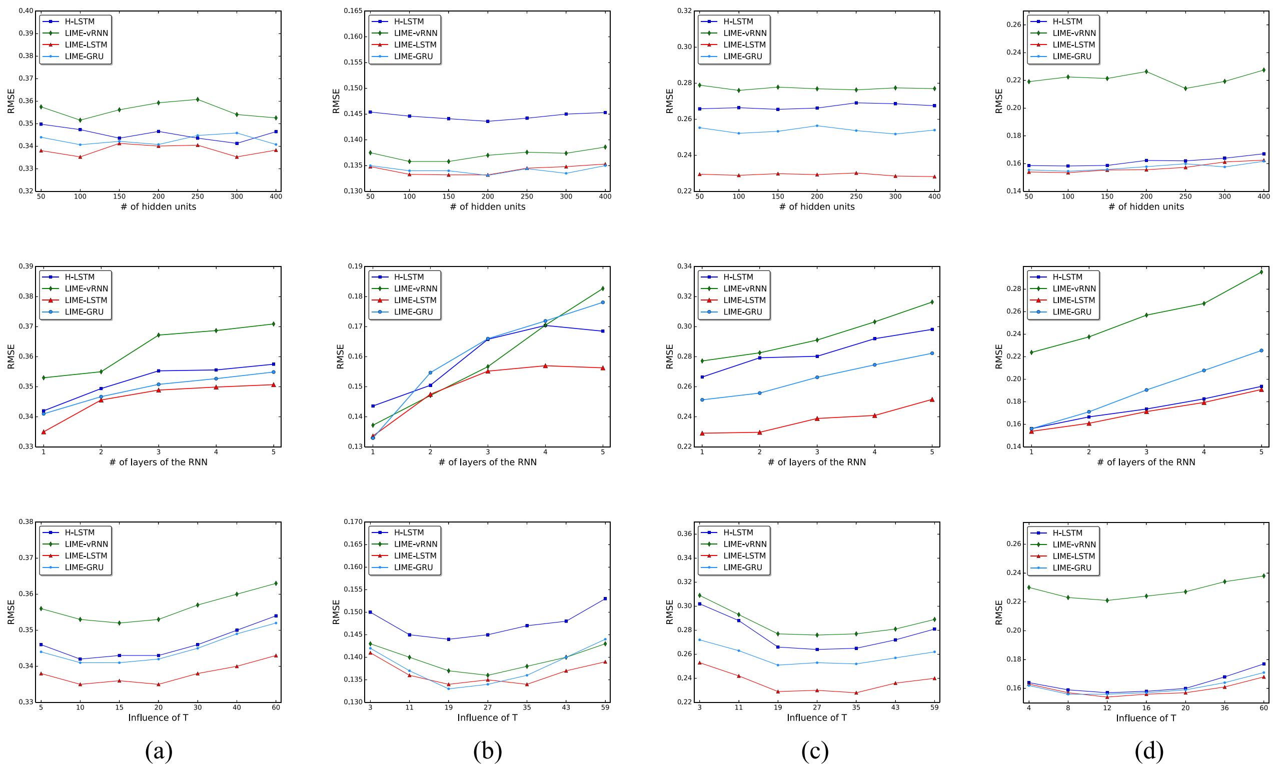


Fig. 9. Plots show the imputation performance of the four datasets under different hyper-parameters. The missing rates are shown in parentheses. The first row shows the influence of the number (#) of hidden units, the second row shows the influence of additional layers, and the third row shows the influence of T. (a) Sanity check (50%). (b) Monthly temperature (15%). (c) Daily births (40%). (d) SCITOS G5 (15%).

the imputed value of x_t (which will have some amount of error) into the network and then the RSV at time t is trained to predict the revealed value x_{t+1}. In this way, the network can learn to recover from previous errors. In other words, the network has the ability to reduce the errors by approximating the revealed value.

We perform some experiments to verify this ability. The setup is as follows: first, we train LIME-LSTM on the `Monthly temperature(20%)` dataset, and then save the model and record the imputation and prediction results (denoted as R_i and R_p, respectively). Next, we test the saved model with the same data, but we artificially amplify the error by adding uniform random noise U in increasing amounts at each imputation and record the imputation and prediction results (denoted as $R_i^{'}$ and $R_p^{'}$, respectively). Finally, the deviation of the original result and the noise-added result is calculated (i.e., $|R_i - R_i^{'}|/R_i \times 100\%$) to measure the model's ability to recover from errors. For comparison purposes, the same settings are applied to H-LSTM.

From Table II, on the one hand, at a noise level of $U(-0.1, 0.1)$, the deviation is small, which verifies that our assumption that the model is able to recover from errors, but as the noise level increases, the deviation increases at a superlinear rate. On the other hand, compared with H-LSTM, LIME-LSTM is more capable of recovering from errors, benefiting from the residual short path which correlates sufficient history information with the missing item, further improving its ability to recover from errors.

TABLE II
COMPARISON OF H-LSTM AND LIME-LSTM'S DEVIATIONS ON THE `Monthly Temperature(20%)` DATASET

Noise Level	The deviation (%)			
	H-LSTM		LIME-LSTM	
$U(a,b)$*	imputation	prediction	imputation	prediction
(-0.1,0.1)	3.54	0.28	0.45	0.20
(-0.2,0.2)	5.04	4.26	1.86	1.96
(-0.3,0.3)	11.73	8.35	3.82	4.46
(-0.4,0.4)	14.96	13.90	5.38	7.55
(-0.5,0.5)	21.90	22.00	9.17	13.24

*a and b are the minimum and maximum values of the distribution U.

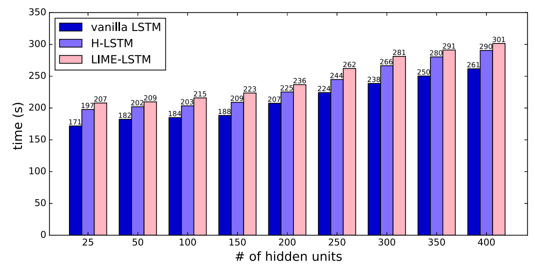


Fig. 10. This plot shows a runtime comparison of training times with an increasing number (#) of hidden units between vanilla LSTM, H-LSTM, and LIME-LSTM on the `SCITOS G5(20%)` dataset.

F. Runtime Analysis

It is impossible to compare the runtime of our method with other methods since most methods use a CPU and a variety of platforms, while we use GPUs on tensorflow. However, under the same setting, the runtime comparison between vanilla LSTM, H-LSTM, and LIME-LSTM with an increasing number of hidden units for 100 epochs of training on the `SCITOS G5(20%)` dataset is shown in Fig. 10. First, comparing vanilla LSTM with H-LSTM, our proposed loss function incurs extra cost for detecting missing values. Comparing H-LSTM with LIME-LSTM, the computation of the RSV only slightly increases the runtime. In general, the runtime complexity does not increase significantly.

VII. Conclusion

In this paper, we presented a novel framework, LIME-RNN, that combines the idea of a graph-based structure with residual short paths, and learns temporal dependencies from incomplete time series in an end-to-end way. A simple linear exponentially decaying memory trace of the hidden unit vector is introduced in LIME-RNN, which enhances the temporal modeling capabilities for incomplete time series. LIME-RNN was evaluated on several synthetic and real-world time series with different missing rates and with both missing at random and consecutively missing data. Extensive experimental results on both random and consecutive missing data demonstrate that LIME-RNN outperforms other state-of-the-art methods on imputation and prediction with missing values.

Throughout this paper, the missing pattern we studied here is random missing (meaning that the missingness is not informative). In other cases, such as time-series data from medical applications, the fact that a test has *not* been run, so the test results are missing, can actually be informative [26]. Investigating how to combine these types of missing data with our approach is left for future work. Also, we only concerned ourselves here with TSP. It also remains in future work to apply our framework to the case of TSC in the presence of missing values.

Acknowledgment

The authors are grateful for the constructive advice received from the anonymous reviewers of this paper.

References

[1] S. Sivakumar and S. Sivakumar, "Marginally stable triangular recurrent neural network architecture for time series prediction," *IEEE Trans. Cybern.*, vol. 48, no. 10, pp. 2836–2850, Oct. 2018.

[2] M. Perez-Ortiz, P. A. Gutierrez, and C. Hervas-Martinez, "Projection-based ensemble learning for ordinal regression," *IEEE Trans. Cybern.*, vol. 44, no. 5, pp. 681–694, May 2014.

[3] H.-F. Yu, N. Rao, and I. S. Dhillon, "Temporal regularized matrix factorization for high-dimensional time series prediction," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 847–855.

[4] J. Mei, M. Liu, Y.-F. Wang, and H. Gao, "Learning a Mahalanobis distance-based dynamic time warping measure for multivariate time series classification," *IEEE Trans. Cybern.*, vol. 46, no. 6, pp. 1363–1374, Jun. 2016.

[5] L. Chi, B. Li, X. Zhu, S. Pan, and L. Chen, "Hashing for adaptive real-time graph stream classification with concept drifts," *IEEE Trans. Cybern.*, vol. 48, no. 5, pp. 1591–1604, May 2018.

[6] Y. Li, H. Hu, Y. Wen, and J. Zhang, "Can we speculate running application with server power consumption trace?" *IEEE Trans. Cybern.*, vol. 48, no. 5, pp. 1500–1512, May 2017.

[7] H. Elmoaqet, D. M. Tilbury, and S. K. Ramachandran, "Multi-step ahead predictions for critical levels in physiological time series," *IEEE Trans. Cybern.*, vol. 46, no. 7, pp. 1704–1714, Jul. 2016.

[8] Q. Ma *et al.*, "WALKING WALKING walking: Action recognition from action echoes," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017, pp. 2457–2463.

[9] Y. Cao, Y. Li, S. Coleman, A. Belatreche, and T. M. Mcginnity, "Detecting wash trade in financial market using digraphs and dynamic programming," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 11, pp. 2351–2363, Nov. 2016.

[10] T. Anwar, C. Liu, H. L. Vu, M. S. Islam, and T. Sellis, "Capturing the spatiotemporal evolution in road traffic networks," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 8, pp. 1426–1439, Aug. 2018.

[11] D. B. Rubin, "Inference and missing data," *Biometrika*, vol. 63, no. 3, pp. 581–592, 1976.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 630–645.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

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

[15] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[16] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Soc. B Methodol.*, vol. 39, no. 1, pp. 1–38, 1977.

[17] B. Sinopoli *et al.*, "Kalman filtering with intermittent observations," *IEEE Trans. Autom. Control*, vol. 49, no. 9, pp. 1453–1464, Sep. 2004.

[18] S. Oba *et al.*, "A Bayesian missing value estimation method for gene expression profile data," *Bioinformatics*, vol. 19, no. 16, pp. 2088–2096, 2003.

[19] L. Li, J. Mccann, N. S. Pollard, and C. Faloutsos, "DynaMMo: Mining and summarization of coevolving sequences with missing values," in *Proc. ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2009, pp. 507–516.

[20] I. R. White, P. Royston, and A. M. Wood, "Multiple imputation using chained equations: Issues and guidance for practice," *Stat. Med.*, vol. 30, no. 4, pp. 377–399, 2011.

[21] O. Anava, E. Hazan, and A. Zeevi, "Online time series prediction with missing data," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2191–2199.

[22] H. Lei, Y. Xia, and X. Qin, "Estimation of semivarying coefficient time series models with ARMA errors," *Ann. Stat.*, vol. 44, no. 4, pp. 1618–1660, 2016.

[23] Z. Cai, J. Fan, and Q. Yao, "Functional-coefficient regression models for nonlinear time series," *J. Amer. Stat. Assoc.*, vol. 95, no. 451, pp. 941–956, 2000.

[24] D. Tjøstheim and B. H. Auestad, "Nonparametric identification of nonlinear time series: Projections," *J. Amer. Stat. Assoc.*, vol. 89, no. 428, pp. 1398–1409, 1994.

[25] P. Brakel, D. Stroobandt, and B. Schrauwen, "Training energy-based models for time-series imputation," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 2771–2797, 2013.

[26] Z. C. Lipton, D. Kale, and R. Wetzel, "Directly modeling missing data in sequences with RNNs: Improved classification of clinical time series," in *Proc. Mach. Learn. Healthcare Conf.*, 2016, pp. 253–270.

[27] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Sci. Rep.*, vol. 8, no. 1, p. 6085, 2018.

[28] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 550–558.

[29] Hipel and McLeod. (1994). *Time Series Data Library*. [Online]. Available: https://datamarket.com/data/set/22vp

[30] Hipel and McLeod. (1994). *Time Series Data Library*. [Online]. Available: https://datamarket.com/data/set/235j

[31] D. Dheeru and E. K. Taniskidou. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

[32] Hipel and McLeod. (1994). *Time Series Data Library*. [Online]. Available: https://datamarket.com/data/set/22vk

[33] S. A. Rahman, Y. Huang, J. Claassen, and S. Kleinberg, "Imputation of missing values in time series with lagged correlations," in *Proc. IEEE Int. Conf. Data Mining Workshop*, 2015, pp. 753–762.

[34] A. L. Freire, G. A. Barreto, M. Veloso, and A. T. Varela, "Short-term memory mechanisms in neural network learning of robot navigation tasks: A case study," in *Proc. Robot. Symp.*, 2009, pp. 1–6.

[35] J. Archer *et al.*, "Temporally separating cherenkov radiation in a scintillator probe exposed to a pulsed X-ray beam," *Physica Medica Eur. J. Med. Phys.*, vol. 42, pp. 185–188, Oct. 2017.

[36] Q. Ma, L. Shen, and G. W. Cottrell, "Deep-ESN: A multiple projection-encoding hierarchical reservoir computing framework," *arXiv preprint arXiv:1711.05255*, 2017.

[37] J. Ar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, no. 1, pp. 1–30, 2006.

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

[39] R. L. Kashyap, "Optimal choice of AR and MA parts in autoregressive moving average models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-4, no. 2, pp. 99–104, Mar. 1982.

[40] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, vol. 37. San Francisco, CA, USA: Holden-Day, 1976, pp. 238–242.

[41] I. J. Schoenberg, *Cardinal Spline Interpolation*, vol. 12. Philadelphia, PA, USA: SIAM, 1973.

[42] E. G. Booth, J. F. Mount, and J. H. Viers, "Hydrologic variability of the cosumnes river floodplain," *San Francisco Estuary Watershed Sci.*, vol. 4, no. 2, 2006, Art. no. 2.

[43] T. Schneider, "Analysis of incomplete climate data: Estimation of mean values and covariance matrices and imputation of missing values," *J. Climate*, vol. 14, no. 5, pp. 853–871, 2001.

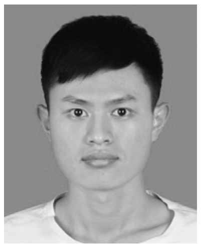
[44] S. Moritz and T. Bartz-Beielstein, "ImputeTS: Time series missing value imputation in *R*," *R J.*, vol. 9, no. 1, pp. 207–218, 2017.

[45] G. A. P. A. Batista and M. C. Monard, "An analysis of four missing data treatment methods for supervised learning," *Appl. Artif. Intell.*, vol. 17, nos. 5–6, pp. 519–533, 2003.

[46] Y. Wang and F. Tian, "Recurrent residual learning for sequence classification," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, 2016, pp. 938–943.

Qianli Ma (M'17) received the Ph.D. degree in computer science from the South China University of Technology, Guangzhou, China, in 2008.

He is an Associate Professor with the School of Computer Science and Engineering, South China University of Technology. From 2016 to 2017, he was a Visiting Scholar with the University of California at San Diego, La Jolla, CA, USA. His current research interests include machine-learning algorithms, data-mining methodologies, and time-series modeling and their applications.



Sen Li is currently pursuing the master's degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China.

His current research interests include machine learning and deep learning.

Lifeng Shen received the bachelor's degree in mathematics from Jinan University, Guangzhou, China, in 2015 and the master's degree in computer science from the School of Computer Science and Engineering, South China University of Technology, Guangzhou, under the supervision of Prof. M. Qianli.

He is currently a Research Assistant with the Hong Kong University of Science and Technology, Hong Kong, working with Prof. J. T. Kwok. His current research interests include machine learning, deep learning, and time-series mining.

Jiabing Wang received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2003.

He is currently an Associate Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His current research interests include machine learning, algorithm design for data mining and its applications to social network analysis, natural language processing, bioinformatics, and computer vision.

Jia Wei received the B.Sc. and M.Sc. degrees in computer science from the Harbin Institute of Technology, Harbin, China, in 2003 and 2006, respectively, and the Ph.D. degree in computer science from the South China University of Technology, Guangzhou, China, in 2009.

He is currently an Associate Professor with the School of Computer Science and Engineering, South China University of Technology. His current research interests include machine learning and artificial intelligence.

Zhiwen Yu (S'06–M'08–SM'14) received the Ph.D. degree in computer science from the City University of Hong Kong, Hong Kong, in 2008.

He is a Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China, and an Adjunct Professor with Sun Yat-sen University, Guangzhou. He has published over 100 refereed journal papers and international conference papers. His current research interests include data mining, machine learning, pattern recognition, and intelligent computing.

Dr. Yu is a Distinguished Member of the China Computer Federation, a Senior Member of ACM, and the Vice Chair of the ACM Guangzhou Chapter.

Garrison W. Cottrell received the Ph.D. degree in computer science from the University of Rochester, Rochester, NY, USA.

He was a Postdoctoral Fellow with D. E. Rumelhart with the Institute for Cognitive Science, University of California at San Diego, La Jolla, CA, USA, where he is currently a Professor of computer science and engineering. His current research interests include cognitive modeling, neural networks, and deep network modeling of the primate visual system.