Neural ODEs for Informative Missingess in Multivariate Time Series

Mansura Habiba
Cloud Solutions Architect
IBM in Ireland
Dublin, Ireland
0000-0001-9051-1370

Barak A. Pearlmutter
Dept of Computer Science & Hamilton Institute
Maynooth University
Maynooth, Ireland
0000-0003-0521-4553

***Abstract*—Informative missingness is unavoidable in the digital processing of continuous time series, where the value for one or more observations at different time points are missing. Such missing observations are one of the major limitations of time series processing using deep learning. Practical applications, e.g., sensor data, healthcare, weather, generates data that is in truth continuous in time, and informative missingness is a common phenomenon in these datasets. These datasets often consist of multiple variables, and often there are missing values for one or many of these variables. This characteristic makes time series prediction more challenging, and the impact of missing input observations on the accuracy of the final output can be significant. A recent novel deep learning model called GRU-D is one early attempt to address informative missingness in time series data. On the other hand, a new family of neural networks called Neural ODEs (Ordinary Differential Equations) are natural and efficient for processing time series data which is continuous in time. In this paper, a deep learning model is proposed that leverages the effective imputation of GRU-D, and the temporal continuity of Neural ODEs. A time series classification task performed on the PhysioNet dataset demonstrates the performance of this architecture.**

***Index Terms*—GRU-D, Informative missingness, Neural Ordinary Differentiation Equations**

I. Introduction

Continuous time series usually consists of a series of data collected on different time points at a different sampling frequencies. Based on the sampling frequency as well as the availability of data, some time points in the series can have missing observations for many of the time point in a continuous time series. These missing values can influence the result of different time series problems, e.g. classification, prediction using deep learning. Che et al. [2] demonstrates some findings using the MIMIC-III datasets [6] which reflects the significance of missing patterns in time series prediction tasks. If the value for certain variables in a multivariate time series is missing for a significant time, the impact of corresponding input variables fades away over time and can result in inaccurate result. Over time various experiments are being used to process data with missing value in order to overcome the imposed challenges. These techniques are suitable for small as well as a simple time series. As the complexity, dynamics, length, sampling frequencies, number of the variable of the time series increases, these techniques become irrelevant. Some of the methods are as following

- Omit the missing data and perform the task on only available observations of the time series. This approach helps to ignore the unavailable data. However, if the missing rate is high and a significant amount of data is ignored, this solution often results in an inaccurate outcome and misguiding prediction.
- Use data imputation to fill out the missing observation with substitutes value. There are several imputation techniques to determine substitute value. The limitation of this approach is that its only suitable for simple time series. It is often too hard to find right substitute values for complex time series. For complex time series, rather than using a single data imputation method, multiple imputation methods are often used combinedly to deal with the complexity of data series as well as to reduce the uncertainty.
- Another practice is to apply data imputation multiple times iteratively with a target to reach an average value for the missing observations. However, data imputation only effective for simple data series with fixed missing rate and fixed time series length. In reality, time series are often variable in length and missing observations occur in completely random order.

Among all different Neural network families, Recurrent Neural Networks (RNN) have shown significant efficiency in case of solving missing pattern in time series with different gating units as well as their capacity of storing memories. These days, various time series tasks such as classification, prediction, generation are usually solved using RNN models. Different gating units for RNNs (e.g., Long Short-Term Memory (LSTM) [5], Gated Recurrent Units (GRU) [4], and GRU-D [2]) typically consider time series as a dynamical system of the discrete and fixed time step. Theoretically, fixed step can be enough for time series modelling if it is too small. However, real-world time series data are usually sampled at an irregular rate. For some applications such as sensors, a short time step is necessary to cope with the higher data sampling frequencies. On the other hand, patients health record needs to be sampled at a higher time step, as the time gap between two consecutive visits of a patient can be very long. Due to the irregular data sampling rate and variable length, multivariate

978-1-7281-9418-9/20/$31.00 ©2020 IEEE

time series are very complex in nature. Therefore, it requires a structured and dynamic model to defeat the uncertainty of informative missingness.

Chen et al. [3], Rubanova et al. [8] try to handle missing pattern in continuous time series with data imputation. GRU-D [2] successfully exploits the strength of RNN models for time series to capture the long term dependencies in multivariate time series. Another recent family of neural networks, Neural Ordinary Differential Equations (ODE-NN) [1, 3, 7] helps to solve time series by using black-box differential equation solver. This kind of neural network leverages the initial value problem to compute the hidden dynamics (f) as a function of continuous time at any time t. As shown in (1), ODE-NN can compute the hidden state of time t (h_t) from the initial hidden state(h_0) and the hyperparameters (θ_{t-1}) update over time. Here $t \in \{0, \ldots, T\}$ and $h_t \in \mathbb{R}$.

$$h_t = h_{t-1} + f(h_0, \theta_{t-1}) \tag{1}$$

In this paper, two different neural network models are introduced. Both models leverage the differential equation solver in order to compute the hidden dynamics of the model. They uses differential equation solver to impute data, where the missing observations of variables are replaced by the derivative of the value of available observations of corresponding variables. Over time the decay in hidden dynamics, as well as input, has a significant impact on the final output on multivariate time series. This work computes the decay rate as the derivatives of time (t), therefore, the decay rate can control the gradient optimization of the model over time. First model use to generate the hidden dynamics of GRU-D model as continuous time dynamics using differential equation solver. Second model compute decay rate in addition to continuous hidden dynamics of GRU-D model. In this work, the time series are considered a function of time t, rather than discrete sequence. The second model proposed in this paper,shows an efficient way to generate both hidden and input decay rate based on the dynamics of the data. Experiment on Physio net dataset demonstrate that the proposed models outperformed existing GRU-D model in time series classification task. These experiments show that ODE based neural network model can successfully solve the informative missingness. The main goal of this work is to use an ODE solver as a black box ODE solver and compute the gradient for the optimizer, which is in charge of optimizing the parameters of the neural network model.

II. State of Art

A multivariate time series (X_t) with D variables and of T length can be described as in (2). Time series (X_t) can have missing observation. Any observation for time series (X_t) is $x_t \in \mathbb{R}^D$ which represents t-th observation of all variables and x_t^d represents the value of the d-th variable at time t.

$$X_t = x_1, x_2, \ldots, {x_T}^D \in \mathbb{R}^{T \times D} \tag{2}$$

The problem domain can be described as in Fig. 1. Here X is a continuous multivariate time series with some missing values. The masking vector M identifies missing observations. The timestamp vector S records the time of the t-th observation of each variable. The time interval vector Δ measures the duration of unavailability. The strength of the existing GRU-D implementation is that it uses the masking and time interval vectors to characterize the missing pattern rather than adopting the traditional missing-completely-at-random model. This approach helps quantify the influence of missing observations and adapt accordingly.

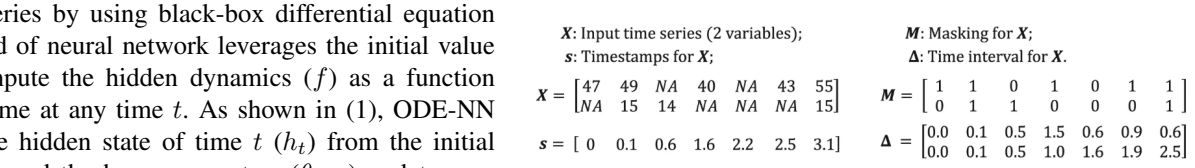


Fig. 1: Continuous time series with missing values, from Che et al. [2]

A. Existing GRU-D implementation

The main contribution of GRU-D is that it can identify the long term dependencies as well as useful missing pattern in data. It can utilize long term temporal missing pattern in time series. Generally, GRU-D outperforms the other two RNN (GRU and LSTM) in terms of prediction experiments. The performance evaluation and comparison described in Che et al. [2] shows that GRU-D scored 2.5% higher accuracy than other RNN implementation. Using GRU-D model, can predict of time series without relying on previous missing observations in the time series. This model achieves higher accuracy within less time in case of robust prediction of multivariate time series data. The core components of GRU-D are as following :

1) Masking Vector: A masking vector (M) $m_t \in [0, 1]^D$ indicates either the observation is available with value 1 or missing with value 0 at any time step t.

$$m_t^d = \begin{cases} 0 & \text{if } x_t^d \text{ is missing} \\ 1 & \text{if } x_t^d \text{ is available} \end{cases} \tag{3}$$

2) Time Interval Vector: This vector computes the time interval ${\delta_t}^d$ for each d-th variable since its last observation. If the first observation for d-th variable is measured at time $s_0 = 0$, then the time interval can be measured by (4).

$$\delta_t^d = \begin{cases} s_t - s_{t-1} + \delta_{t-1}^d & t > 1, m_{t-1}^d = 0 \\ s_t - s_{t-1} & t > 1, m_{t-1}^d = 1 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

3) Data Imputation Value: For replacing the missing data, three different approaches, GRU-Mean, GRU-Froward and GRU-Simple, are used in the GRU-D model.

GRU-Mean: : Each missing observation is replaced with the mean (x_t^d) of the individual variable across the training examples. Both training and testing datasets is processed to compute the dynamics ($\tilde{x}^d$) of GRU-Mean as shown in (6). Here, N is the length of time series.

$$x_t^d \leftarrow m_t^d x_t^d + \left(1 - m_t^d\right) \tilde{x}^d \tag{5}$$

$$\text{Where, } \tilde{x}^d = \sum_{n=1}^{N}\sum_{t=1}^{T_n} m_{t,n}^d x_{t,n}^d / \sum_{n=1}^{N}\sum_{t=1}^{T_n} m_{t,n}^d \tag{6}$$

GRU-Forward: : Missing value is replaced with last available observations as shown in (7) . Here $t' < t$ is the last time when the d-th variable was observed.

$$x_t^d \leftarrow m_t^d x_t^d + \left(1 - m_t^d\right) x_t^d \tag{7}$$

GRU-Simple: : This approach identifies the missing variable along with the duration of missingness by concatenating the measurement, masking and time interval vectors as shown in (8).

$$x_t^{(n)} \leftarrow \left[x_t^{(n)}; m_t^{(n)}; \delta_t^{(n)}\right] \tag{8}$$

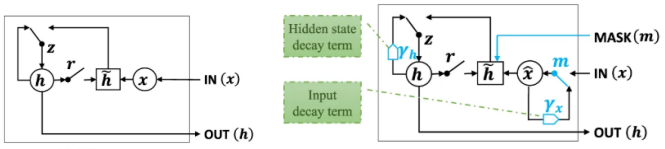
4) Hidden State Decay: The importance of decay rate is that it can control the decay mechanism of the model considering underlying properties associated with each variable. As a result, individual variable of a multivariate time series can influence the prediction based on its actual weight of decay. The decay rate (γ_t) is modelled as (9) with its associated weight and bias parameter W_γ and b_γ. The negative rectifier force decay rate to decrease monotonically between range [0,1].

$$\gamma_t = \exp\left\{-\max\left(0, W_\gamma \delta_t + b_\gamma\right)\right\} \tag{9}$$

There are two different decay rate, hidden decay rate (γ_{h_t}) and input decay rate (γ_{x_t}). Hidden decay rate controls the decrease in decays of the previous hidden state h_{t-1} before computing current hidden state h_t as like (10). Here, $\hat{h}_{t-1}$ represents a state between observations, but it has less control over the raw input variable. Therefore, a second decay rate, input decay rate, controls the decay of raw input variables directly.

$$\hat{h}_{t-1} = \gamma_{h_t} \odot h_{t-1} \tag{10}$$

As shown in Figs. 2a and 2b, the main difference between traditional GRU and GRU-D is that it uses two different decay mechanism hidden state and raw input. Using additional hidden state decay helps to capture the complete missing patterns.



(a) GRU Model Cell (b) GRU-D Model Cell [2]

Fig. 2: GRU vs GRU-D Model Cells

The update functions for different gate units and intermediate states of GRU-D are explained by (11a) (Reset Gate), (11b) (Update Gate), (11c) (Intermediate state), and (11d) (Hidden state). The typical GRU cell update functions [4] are modified in GRU-D. The input (x_t) and hidden (h_t) vectors are replaced by $\hat{y}_t$ and $\hat{h}_t$ respectively. Besides, a new set of parameter vectors, V_r, V_z and V are introduced for mask vector m_t.

$$r_t = \sigma(W_r \hat{x}_t + U_r \hat{h}_{t-1} + V_r m_t + b_r) \tag{11a}$$
$$z_t = \sigma(W_z \hat{x}_t + U_z \hat{h}_{t-1} + V_z m_t + b_z) \tag{11b}$$
$$\tilde{h}_t = \tanh(W \hat{x}_t + U(r_t \odot \hat{h}_{t-1}) + V m_t + b) \tag{11c}$$
$$h_t = (1 - z_t) \odot \hat{h}_{t-1} + z_t \odot \tilde{h}_t \tag{11d}$$

The neural network architecture of GRU-D shown in Fig. 3 shows that the input variable, masking vector and time interval vector is used as input for each cell in a GRU-D model. In Fig. 3, the GRU-Mean data imputation method is used. Both decay rates (γ_{h_t} and γ_{x_t}) shape the input and hidden state.

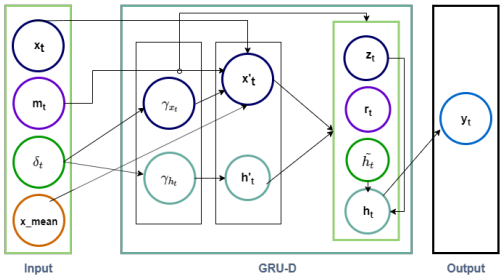


Fig. 3: Architecture of GRU-D

B. ODE-RNN

This neural network defines the hidden state between observations as the solution of an Ordinary Differential Equation (ODE) as shown in (12a)

$$h'_t = \mathit{ODESolve}(f_\theta, h_0, (t-1, t)) \tag{12a}$$
$$h_t = \mathit{RNNCell}(h'_t, x_t) \tag{12b}$$

Here the function f_θ describes the dynamics of the hidden state using a neural network with parameters θ. The previous hidden state at time $t-1$ can be computed at any time using a differential equation solver. ODE-RNN uses a standard RNN update function as shown in (12b) to compute current hidden state with input the intermediate state h'_t. The hidden state can be computed as (12b) without depending on the time interval implicitly. The hidden state is defined by an ODE solver and later updated by another RNN network at each observation. ODE-RNN [8] is an auto-regressive model which makes one step ahead prediction conditioned on the sequence of previous observation. This model consists of two different neural networks: (i) an encoder, (ii) a decoder and ODE solver, as shown in Fig. 4.

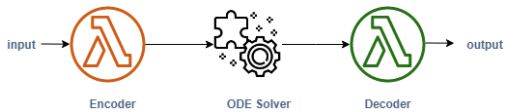


Fig. 4: Architecture of ODE-RNN

C. Latent-ODE

latent-ODE is a variational auto-encoder based continuous time model. Both training and prediction use this auto-encoder. ODE-RNN is the encoder for Latent-ODE. This model can be characterized as encoder-decoder or fully ODE based sequence-to-sequence based architecture. Initial value problem is used to compute the latent state (z_t) at any time t. A neural network (g) generates the final hidden state of ODE-RNN encoder as the mean and stranded deviation initial latent state z_0 as shown in (13a). To get the approximate posterior both of the encoder and decoder are trained jointly. The posterior distribution over latent states are a function of final hidden state as shown in (13b). This indicates the explicit uncertainty which is not available in traditional RNN or ODE-RNNs.

$$\mu_{z_0}, \sigma_{z_0} = g(\text{ODE-RNN}_\phi(\{y_i, t_i\}_{i=0}^N)) \tag{13a}$$

$$q(z_0|) = \eta(\mu_{z_0}, \sigma_{z_0}) \tag{13b}$$

The major benefit of latent-ODE is the hidden dynamics of the system and distribution of observations get decoupled from each other. As a result, each of the hidden state at any time t can be computed independently. In addition, the posterior distribution over latent states can measure uncertainty which is a unique feature for latent-ODE.

III. Methodology

Two different models are proposed in this paper to solve the informative missingness of multivariate continuous time series as shown in Fig. 1.

A. Continuous GRU-D

This model leverages an ODE solver in order to compute the hidden dynamics of continuous time series at any time t. Similar to Che et al. [2], the hidden decay rate (γ_{h_t}) and input decay rate (γ_{x_t}) are computed using (9). These decay rates are used as parameters for the network. However, instead of computing the hidden state as a dynamical system of discrete sequential value, this proposed model uses the initial value problem to calculate the continuous dynamics using derivatives generated by ODE solver. Therefore, the value of d-th variable at time t (y_t) is computed using (14). (8) shows that y_t is vector concatenating the value of last available observation at time t (x_t), masking vector (m_t) value and hidden state value (h_t) at time t. The hidden dynamics y_0 is the initial value of the vector, $y_0^n \leftarrow [x_0^n; m_0^n; h_0^n]$. Here n is the number of variable and y_0 is an $n \times 3$ vector. The resulting vector y_t provides the value of hidden state at time t and observation value of n variables at time t, $y_t \leftarrow [x_t^n; m_t^n; h_t^n]$.

$$(y_1, y_2, \ldots, y_t) = \mathit{ODESolve}(\mathscr{N}, y_0, T, h_0, \theta = [\gamma_{h_t}, \gamma_{x_t}]) \tag{14}$$

Here $\mathscr{N}$ is a neural network similar to the update functions of existing GRU-D model cell, described in (11a), (11b), (11c), and (11d). An ODE neural network, as shown in Fig. 5 is used

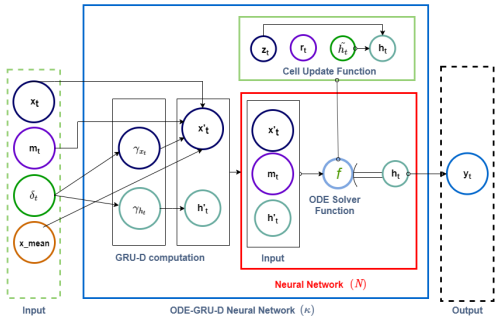


Fig. 5: Architecture of ODE-GRU-D

to compute the update in hidden state and generate the target output.

This model leverage the ODE solver and uses the update functions as described in (11a), (11b), (11c) and (11d). No additional training of encoder or decoder, unlike in Chen et al. [3] or Rubanova et al. [8], is required in this proposed model. ODE-RNN [8] separately generates the hidden dynamics using a Neural ODE [3] and then computed hidden state is updated using standard RNN cell as shown in (1). However,this proposed model continuously update the hidden dynamics using a Neural ODE network $(\mathscr{N})$ which use the update functions of GRU-D. (15)shows that the proposed neural network (κ) takes the same inputs $X = (InputVecotor(X), MaskingVector(M), Decay(\Delta), T)$ as like existing GRU-D as shown in Fig. 1. In addition to these inputs, the proposed model has additional components (i) an ODE based neural network $(\mathscr{N})$, and (ii) a loss function $(\mathscr{L})$ which helps the optimizer to leverage the dynamics of ODE in order to compute the update of hidden state. As shown in (15), the time interval between two consecutive observations is not considered in hidden dynamics computation. Therefore, the length of time interval is irrelevant and the proposed model work for both higher as well as shorter sampling frequencies of any continuous time series.

$$y_t = \kappa(y_0, T, \mathscr{L}, \mathscr{N}) \tag{15}$$

Algorithm 1 shows that a ODE based neural network $\mathscr{N}$ updates the hidden state of the neural network. Here, $\mathscr{N}$ uses the update function that constructs the dynamics of hidden as well as derivative of the cell state and call the ODE solver to update all the parameters for optimizer at once. Algorithm 2 describes the steps for computing the derivative of hidden dy. The parameters of this neural network are optimized based on the output of loss function $(\mathscr{L})$. The input for $\gamma_{x_t}, \gamma_{h_t}$ are computed using the existing GRU-D computation described in (9). Similarly, x_t^d and $\tilde{h}_t$ are computed using (7) and (10).

Algorithm 2 uses the same update functions as Che et al. [2] to compute the continuous hidden dynamics of the neural network. Here f is a GRU-D neural network, which uses x_t^d and $\hat{h}_t$ as input for a ODE based implementation of GRU-D cell update function. T is the time series. *ODESolve* uses f to compute the changes in cell state over time.

Algorithm 1 Compute the state of an ODE-GRU-D cell

procedure $\mathscr{N}(x_t{}^d, \tilde{h}_t, T, hiddenDynamics)$
$y_0 \leftarrow tuple(x_0^d, m_0, \hat{h}_0)$
$\frac{\delta y}{\delta t}, \frac{\delta L}{\delta \theta} \leftarrow ODESolve(f, y_0, T, h, \theta)$
return y_t

Algorithm 2 Update the derivative, the hidden state, and parameters for the optimizer

procedure $f(y, T, \theta)$
$x, h, m \leftarrow y$
Compute r_t using (11a)
Compute z_t using (11b)
Compute $\tilde{h}_t$ using (11c)
$h \leftarrow (h - \tilde{h}_t) * z_t$
$\frac{\delta y}{\delta t} \leftarrow tuple(\frac{\delta x}{\delta t}, \frac{\delta h}{\delta t}, m)$
return $\frac{\delta y}{\delta t}$

B. Extended ODE-GRU-D

The second proposed model ($\mathscr{Q}$) characterized the missing pattern of the time series using Adjoint ODE solver. Fig. 6 shows the architecture of proposed Extended ODE-GRU-D neural network model. Existing GRU-D model computes decay rate as shown in (9), where W_γ and b_γ are parameters of the same GRU model, and they are trained jointly with all the other parameters. However, if the dynamics of the decay rate can be computed as the derivative original input Δ, with respect to time, it can provide the accurate value of decay rate over time. Therefore, in this proposed model, a Filter Linear neural network *FL* is used to compute the derivative of decay rate over time. W_γ and b_γ are parameters of *FL*, and they are trained separately to compute the derivative of decay rate ($\frac{\partial \gamma}{\partial t}$) as shown in (16a). ODE solver solves the neural network (*FL*) with respect to Δ over time to compute the derivative. This derivative is used for computing the decay rate as shown in (16b).

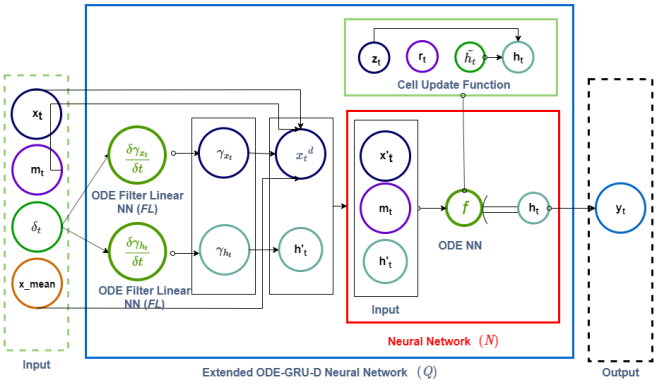


Fig. 6: Architecture of Extended ODE-GRU-D Model

Fig. 6 shows that two ODE Solver using Filter Linear neural network as function is used in proposed model to compute hidden state decay rate and input decay rate. The third ODE Solver compute the update in hidden state of the model.

$$\frac{\delta \gamma}{\delta t} = ODESolve(FL, M, h_0, T, \mathscr{L}, \mathscr{N}) \tag{16a}$$

$$\gamma_t = \exp\left\{-\max\left(0, \frac{\delta \gamma}{\delta t}\right)\right\} \tag{16b}$$

This γ_t from (16b) compute $x_t^d \in X^D$ using (17).

$$x_t^d \leftarrow m_t^d x_t^d + \left(\gamma_{x_t^d} x_t^d + \left(1 - \gamma_{x_t^d}\right)\right) x_{mean_t}^d \tag{17}$$

$X^D = \{x_0^d, x_1^d, ..., x_T^D\}$ is used as the input along with the masking vector (M) and the initial hidden state (h_0) for the proposed *Extended ODE-GRU-D* model. Similar to proposed ODE-GRU-D described in §III-A, *Extended ODE-GRU-D* also uses ODE based neural network ($\mathscr{N}$) to compute the update in hidden dynamics of the continuous time series as shown in (18).

$$y_t = \mathscr{Q}(X^D, M, h_0, T, \mathscr{L}, \mathscr{N}) \tag{18}$$

Proposed Extended ODE-GRU-D exploits ODE solver to understand the decay dynamics of the time series, therefore, controlling of change in the decay rate is more accurate in comparison to existing GRU-D model.this model learns the decay mechanism for raw input as well as the hidden dynamics of the continuous time series. The main advantages of this proposed models is that it naturally handle the decay mechanism and the gap between consecutive observations.

IV. Results

As neural ODE is used to compute the hidden state at any time (t), no additional data processing is required for these tow proposed neural network models. Unlike ODE-RNN [8], the proposed models uses single neural network. it does not require separate encoder and decoder.

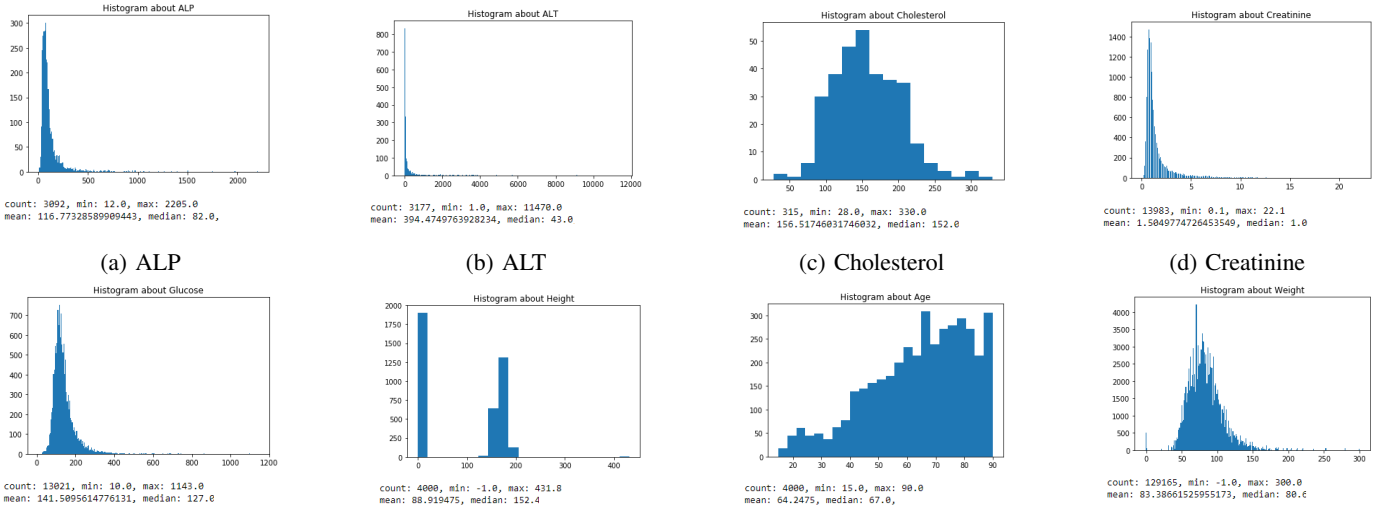
A. Dataset and Task Description

For demonstrating the performance evaluation of proposed models, we use the PhysioNet Challenges dataset [9], a collection of multivariate time series with missing observations. It contains 8000 intensive care unit records (ICU), each of them a time series of 48 hours with 33 parameters. Fig. 7 shows the histogram for some of the parameters, i.e., ALP, glucose, age, height, weight, etc. This experiment uses Training Set A for training as the output of this set is available only. As in the original paper, two prediction tasks, below, are attempted using this dataset.

Mortality task (Binary classification problem): Predict patient's death in the hospital. There are 554 patients with positive mortality label.

Multi-task classification problem: Predict in-hospital mortality, length-of-stay less than 3 days, patient's chance of having a cardiac condition, and patient' recovery state from surgery.

Table I shows that the proposed ODE-GRU-D and Extended ODE-GRU-D significantly outperform the existing GRU-D and Neural ODEs.For the comparative analysis area under curve (AUC) comparison.



(a) ALP (b) ALT (c) Cholesterol (d) Creatinine

(e) Glucose (f) Height (g) Age (h) Weight

Fig. 7: Different parameters of the PhysioNet Challenges 2012 dataset

TABLE I: AUC on PhysioNet

Method	AUC
GRU-D	0.824+0.0.12
ODE-RNN	0.833+0.009
Latent-ODE (RNN Encoder)	0.781+0.018
Latent-ODE (ODE Encoder)	0.829+0.004
Latent-ODE + Poisson	0.826+0.007
ODE-GRU-D	0.8947+0.001
Extended ODE-GRU-D	0.9147+0.005

V. Conclusions

Most available solutions for handling missing value, in continuous time series, use data imputation. However, data imputation requires an extensive amount of data pre-processing; it also can impair the performance. This proposed model learns the hidden dynamics of time series as progress. It can compute the hidden state at any time t without directly depending on the previous value. As the proposed model learns the derivative of changes it does not depends on additional data imputation. The derivatives of hidden state help to differentiate between longer and shorter dependencies; therefore, the proposed model works evenly fine for data with a higher sampling rate as well as a shorter sampling rate. As for using ODE based neural network, this proposed model requires more training time in comparison to testing time. Training time is relative to the dataset. Also, based on implementation, memory cost can be either fixed or incremental. Future extension of this work mainly focuses on using a similar model when the informative missingness in any dataset is not random. Instead, it follows a particular distribution. If the time pattern of occurrence of the missing observation can be identified, it is possible to switch on or off the computation of derivative. That can help to predict the missingness of the information and design the model accordingly.

References

[1] A. E. Bryson, Jr. A steepest ascent method for solving optimum programming problems. *Journal of Applied Mechanics*, 29(2):247, 1962.

[2] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8(1):1–12, 2018.

[3] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, pages 6571–83, 2018.

[4] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. Technical Report arXiv:1406.1078, arXiv, 2014.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–80, 1997.

[6] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark. MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3:160035, 2016.

[7] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–9, 1989.

[8] Y. Rubanova, T. Q. Chen, and D. K. Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in Neural Information Processing Systems*, pages 5321–31, 2019.

[9] I. Silva, G. Moody, D. J. Scott, L. A. Celi, and R. G. Mark. Predicting in-hospital mortality of ICU patients: The physionet/computing in cardiology challenge. *In CinC*, 2012.