GRU-M: A Joint Impute and Learn Approach for Sequential Prediction under Missing Data

Soumen Pachal* S.PACHAL@TCS.COM
Avinash Achar* ACHAR.AVINASH@TCS.COM
Nancy Bhutani† NANCYBHUTANI96@GMAIL.COM
Akash Das† DAS.AKASH3@TCS.COM
Tata Consultancy Services Research, IITM Research Park, Chennai, India

Editors: Vu Nguyen and Hsuan-Tien Lin

Abstract

Sequential Prediction in presence of missing data is an old research problem. Classically, researchers have tackled this by imputing data first and then building predictive models. This 2-stage process is typically prone to errors and to circumvent this, researchers have provided a variety of techniques which employ a joint impute and learn approach before prediction. Among these, Recurrent Neural Networks (RNNs) have been very popular given their natural ability to tackle sequential data efficiently. Existing state-of-art approaches either (i)do not impute (ii) do not completely factor the available information around a gap, (iii)ignore position information within a gap and so on. Our approach intelligently addresses these gaps by proposing a novel architecture which jointly imputes and learns by taking into account (i)information from either end of the gap (ii)proximity to the left/right-end of a gap (iii)the length of the gap. In context of this work, prediction means either sequence classification or forecasting. In this paper, we demonstrate the utility of the proposed architecture on forecasting tasks. We benchmark against a range of state-of-art baselines and in scenarios where data is either (a)naturally missing or (b)synthetically masked.

Keywords: Sequential Prediction; Missing Data; RNN; GRU; Time-series;

1. Introduction

Sequence based prediction (classification Xing et al. (2010) and forecasting Lim and Zohren (2021)), is a classic research problem with numerous applications. Given constant methodological enhancements and newer emerging applications, it continues to be very relevant. Recurrent Neural Networks (RNNs) have been a popular state-of-the-art approach for the same. Deep RNNs have been highly successful in diverse domains like computer vision, NLP, time series classification/forecasting, speech and audio processing and so on. RNNs have exhibited state-of-the-art performance (and much more) in tasks like machine translation Sutskever et al. (2014), handwriting recognition Graves et al. (2009) etc.

When data is partly missing, predictive modelling becomes further challenging. This is the problem tackled in this paper. Data could be missing due to many factors like sensor failure, noise (under high noise levels, its better to ignore the data), cost of sensing, maintenance and so on. Sequence prediction under missing data also has a vast literature. RNNs in particular have also been explored for this. This paper addresses sequence prediction

*. Both authors contributed equally.
†. Both authors contributed equally.

under missing data using RNNs in a novel way. Our proposed ideas can be employed for both classification and forecasting. In the classification context, data can appear as variable length sequences, where each sequence is mapped to a class. In forecasting, data is typically in the form of one long sequence. We consider a general multi-step forecasting scenario with possibly additional exogenous variables to be handled.

1.1. Problem & Motivation

We consider real-valued time series with missing values. For sequence classification applications, time-stamps can be either discrete or continuous (real-valued). Here data comes as multiple sequences, where each sequence is real-valued (possibly vector valued). Further, each of these sequences can have different lengths. The classification task is to categorize each sequence into one of many predefined classes. For example, each vector sequence could indicate essential parameter measurements like blood pressure, heart rate, sugar level etc. from a patient, while the classification task may mean detecting a certain disease.

If the task is forecasting, then the time-stamps are assumed discrete (integers or natural numbers). The forecasting task considered here involves, predicting one or more endogenous variables over a multi-step forecast horizon, in the presence of possible exogenous inputs, which influence the evolution of the endogenous variables. Such forecasting tasks arise in diverse applications. A retailer may be interested in forecasting one or more products sales (modelled as endogenous variables) in the presence of exogenous price variation. In electricity markets, forecasting power demand (endogenous) by factoring temperature influences/fluctuations (exogenous) is another example.

1.2. Contributions

Early approaches for prediction under missing data have restricted themselves to a 2-step approach where data is first imputed and subsequently a predictive model is built. However this 2-step approach is very sensitive to the imputation quality and can suffer from excess errors due to poor imputation. To address this, diverse approaches which avoid an explicit imputation step have been proposed.

RNN approaches form a dominant class of techniques for sequential prediction under missing data. A small subclass of these techniques build a predictive model by avoiding any form of imputation during learning by capturing the pattern of missingness directly Lipton et al. (2016); Pachal and Achar (2022), where the missingness pattern to be captured could be very complex. On the other hand, most other techniques employ a joint impute and train strategy. Here, missing gaps in data are imputed during predictive model building, by utilizing the neighboring data that is available. Our proposed approach falls under this class of approaches. Some of these approaches ignore information from the right-end of a data gap Che et al. (2016), while others which consider information from either end, do not explicitly consider time distances from either side of the data gap Cao et al. (2018). Our approach proposes a novel joint and learn strategy while addressing the above issues. Specifically,

- We propose a novel RNN based joint impute and learn strategy which factors both the (closest) left-end and right-end values of a gap, for imputation.

- Our novel strategy takes into account the position of the missing point and its distance from both the left and right end of a gap. In the regions closer to the data gap end, it can impute by maximally factoring the influence from the respective end values. For large gaps, it can ignore the right and left end values while imputing in the mid-region.
- We demonstrate effectiveness of the proposed architecture on real time-series where data is either (i)naturally missing or (ii)artificially masked.

2. Related Work

Prediction under missing data has a long literature. A natural approach to tackle this is to impute first before doing predictive modeling. Data imputation itself has an extensive literature. The early and simpler methods for data imputation include smoothing, interpolation, splines Johnson (2013) etc. which capture straightforward patterns in the data. The advanced imputation approaches include spectral analysis Mondal and Percival (2010), matrix factorization Koren et al. (2009), kernel methods Rehfeld et al. (2011), EM algorithm García-Laencina et al. (2010), GANs (Generative Adversarial Networks) Luo et al. (2018), Tranformers Nie et al. (2024) etc. Hence, the imputation step can be computationally expensive. Further, imputation methods typically make strong, restrictive assumptions like missing at random, low missing rates and so on which may not hold in practice. The two step sequential process of imputation followed by prediction can suffer from imputation errors which can in-turn render poor predictive models.

To circumvent this, literature has seen techniques where data imputation is not separately carried out. When performed, it is coupled with the predictive model building process. There have been a wide range of techniques proposed along these lines. These include RNNs, GANs, Gaussian processes (GP) etc. Gaussian processes essentially do a Bayesian estimation where the predictive distribution given the observed values is inferred. They are a natural data-driven model to consider when data is irregularly sampled with many missing values. There has been some recent work along this direction Futoma (2018); Li (2020) mostly applied in the health care domain (clinical time series). In the rest of the section, we elaborate on RNN techniques only, as our work is RNN based.

RNNs have been explored to tackle missing data scenarios even before the deep learning surge Bengio and Gingras (1995); Tresp and Briegel (1997); Parveen and D. Green (2001). Bengio and Gingras (1995) is perhaps the earliest approach which unfolds the RNN to allow the missing values to settle down (converge) while the other inputs are fixed at the observed values. Parveen and D. Green (2001) employs a slightly modified architecture of Bengio and Gingras (1995). While both use non-gated RNNs, Bengio and Gingras (1995) uses a feedback structure based on a Jordan network while Parveen and D. Green (2001) uses an Elmann network.

Among the RNN approaches, a couple of them impute neither before nor during model building. Lipton et al. (2016) encodes the missingness pattern as a simple binary vector indicating the presence or absence of a data point and then trains. A more involved approach of encoding the missingness pattern using two RNN layers in a lossless fashion was proposed in Pachal and Achar (2022). Both these approaches do not bother to impute even during model building. To capture all the complex dependencies based on the patterns of missingness can be hard on these RNN models.

There is a recent attention-based RNN approach which first imputes using some padding (from left) OR some form of interpolation Cinar et al. (2018). During subsequent model building it doesn't use the imputed values as they are, but assumes some form of weighted influence from these imputed valued on the prediction. These weights are suitably parameterized and the associated parameters are learnt parallely with the predictive model.

Among the recent joint impute and learn strategies, GRU-D Che et al. (2016) imputes a missing point as weighted combination of the overall (time-series) mean and the closest available point at the left end of a data gap. The weight that controls the influence from the left end is assumed to decay exponentially with the time distance from the left end. The decay/weight factor is learnt in conjunction with the predictive model.

While GRU-D totally ignores the right-end of a data gap, BRITS is a more recent approach incorporating influence of both right and left end of a gap towards imputation, using a bidirectional layer. In the state evolution from left, there is a state decay (inspired from GRU-D) incorporated which is a function of how far the current event is to the closest available point to its left. The state at each time-step is non-linearly transformed via a feed-forward network to capture the input at the next time-step (to the right). Further, the state of the backward layer is transformed to capture the input at the next step to the left. If the input at the current time-step is missing, then its predicted to be the average of the output of the previous step of both the forward and backward layer. In BRITS, equal importance is given to both forward and backward imputed values (computed by passing the available information from either end of the gap through bidirectional layer) irrespective of its position in the gap. In our GRU-M model, we consider the left-end and right-end available values directly. Further, we have considered a convex combination of left-end, right-end and mean values where the coefficients are learned along with the other model parameters during training. BRITS has been recently extended to tackle multivariate time-series with selective dependencies, using graph neural networks in GRIN Cini et al. (2021).

Recently, Bi-GAN, a GAN approach Gupta et al. (2021) was considered using a joint impute and learn strategy. It also factors both the left and right end of a gap for imputation similar to BRITS. But unlike BRITS, it adopts a generator-discriminator framework where the generator is a bidirectional RNN as in BRITS, while training is done in an adversarial setting jointly with a discriminator which predicts the presence or absence of data. In addition, Bi-GAN differs from BRITS in the last step where a missing input is modelled as a weighted average (instead of standard average) of the outputs of the previous step from the forward and backward layers. These weights are assumed to be a function of the distance from the left-end/right-end gap.

2.1. In Perspective of Other RNN Approaches

Unlike GRU-D, our approach (denoted as GRU-M) factors information from both the left and right end of a data gap in a novel and intelligent fashion. As explained in detail in the next section, our approach can be viewed as a non-trivial extension of GRU-D. The attention-based approach of Cinar et al. (2018) while intelligent in learning a weighted influence of the imputed values during model building, can still be prone to imputation errors due to its separate imputation step.

Our approach uses the closest left and right available inputs directly to impute. BRITS and Bi-GAN use the closest left and right values in an indirect fashion by (a)passing the closest left available input through the forward layer (b)passing the closest right available input through the backward layer and (c) by taking an average of these two arrived values from either direction. Unlike our method, both are prone to error accumulation as imputation is carried out sequentially from either direction. Further, unlike BRITS or Bi-GAN, our approach additionally allows for using the mean as a possible impute option, which makes sense when imputing in the middle region of large data gaps.

3. Proposed Methodology

3.1. GRU

We start by describing the GRU unit gating equations in detail as our proposed approach builds on it. Also, GRU unit as the building block for RNNs is currently very popular across sequence prediction applications Ravanelli et al. (2018); Che et al. (2016); Gruber and Jockisch (2020). GRU based cell computes its hidden state (for one layer) at the i^{th} time-step as follows.

$$z_i = \sigma(W^z u_i + U^z h_{i-1} + b_z) \tag{1}$$
$$r_i = \sigma(W^r u_i + U^r h_{i-1} + b_r) \tag{2}$$
$$\tilde{h}_i = tanh(U(r_i \odot h_{i-1}) + W u_i + b) \tag{3}$$
$$h_i = (1 - z_i) \odot h_{i-1} + z_i \odot \tilde{h}_i \tag{4}$$

where h_{i-1} is state at the previous time-instant, (u_i) is current input, $\sigma(.)$ denotes sigmoid function, z_i is update gate vector and r_i is the reset gate vector. $\tilde{h}_i$ is the additional memory (summary of all inputs so far) which is a function of u_i and h_{i-1}, the previous hidden state. The reset signal controls the influence of the previous state on the new memory. The final current hidden state is a convex combination (controlled by z_i) of the new memory and the memory at the previous step, h_{i-1}. All associated weight matrices W^z, W^r, W, U^z, U^r, U and vectors b_z, b_r and b are trained using back-propagation through time.

3.2. Proposed GRU-M Unit

As explained earlier in related work, GRU-D in addition to using the masking binary vector also adopts a novel joint impute-learn strategy. We first describe the essentials of GRU-D, then point out its drawbacks and finally describe our method, GRU-M which can be viewed as a non-trivial extension of GRU-D.

A multivariate time-series X with D variables of length N is denoted as

$$\langle (x_1, t_1), (x_2, t_2), \ldots (x_i, t_i) \ldots (x_N, t_N) \rangle \in \mathbb{R}^{D \times T} \tag{5}$$

Each $x_i \in \mathbb{R}^D$ represents the i^{th} observation, while x_i^d represents its d^{th} component. As evident, $t_i \in \mathbb{R}$ denote the time-stamp of x_i, the time instant at which the measurement happens.

To capture which variables are missing at which time-instant, we define formally the binary masking variable, m_i^d, as

$$m_i^d = \begin{cases} 1, & \text{if } x_i^d \text{ is observed} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

GRU-D also maintains a time-interval δ_i^{dL}, denoting the distance from the closest available data-point on the left for the d^{th} variable. More formally,

$$\delta_i^{dL} = \begin{cases} t_i - t_{i-1} + \delta_{i-1}^{dL}, & \text{if } i > 1, m_{i-1} = 0 \\ t_i - t_{i-1}, & i > 1, m_{i-1} = 1 \\ 0, & i = 1. \end{cases} \tag{7}$$

GRU-D essentially models a *decay* mechanism from the left end of a gap and works with δ_i^L ($D \times 1$ vector of δ_i^{dL}s) only. When data is missing at time-step i at the d^{th} input variable, the input is hypothesized to be a weighted combination of the closest available input to the left and the time average of the d^{th} component (across the sequence). The decay/weight factor is modelled using a monotonically decreasing function of δ_i^L and ranges between 0 and 1. The vector of decay/weight factors ($D \times 1$ vector) at the i^{th} time-step would be

$$\gamma_i^L = exp\{-max(0, W_\gamma \delta_i^L + b_\gamma\} \tag{8}$$

where b_γ is $D \times 1$ vector and W_γ is a $D \times D$ matrix assumed to be diagonal. This assumes a component-wise independence of decay rates which is pretty realistic. The modified input that is input to the GRU unit is now

$$\hat{x}_i^d = m_i x_i^d + (1 - m_i)(\gamma_i^{dL} x_i^{dL} + (1 - \gamma_i^{dL})\tilde{x}^d) \tag{9}$$

where x_i^{dL} is the last available data point to the left of the d^{th} component of i^{th} observation, $\tilde{x}^d$ is the empirical mean across all available data points among the N observations of the d^{th} variable. Replacing u_i by $\hat{x}_i^d$ in eqns.(1)-(4) is essentially the GRU-D architecture by incorporating input decay.

Drawback of GRU-D: GRU-D's hypothesis is that closer a missing observation is to the left end of the gap ($\delta_i^{dL} \to 0$), closer will $\hat{x}_i^d$ be to x_i^{dL}. The farther away the missing observation is from gap's left end, closer will $\hat{x}_i^d$ be to $\tilde{x}^d$. In this process, GRU-D totally ignores information from right end of the gap. This means even though the missing observation is far away from the left end, it may actually be very close to the right end. In which case, $\hat{x}_i^d$ must be close to x_i^{dR}, (the closest available observation to the right of i^{th} observation at d^{th} component) instead of $\tilde{x}^d$, especially when the gap length is large. In a bid to address this lacunae in GRU-D and beyond, we propose our novel approach termed GRU-M.

GRU-M unlike GRU-D takes into account the right-end of a gap as well while computing $\hat{x}_i^d$, the modified input at each time-step. Towards this, we additionally maintain another time-interval vector δ_i^R denoting the distance from the closest available data-point on the right of any observation. This can be efficiently computed recursively as follows.

$$\delta_i^{dR} = \begin{cases} t_{i+1} - t_i + \delta_{i+1}^{dR}, & \text{if } i < N, m_{i+1} = 0 \\ t_{i+1} - t_i, & i < N, m_{i+1} = 1 \\ 0, & i = N. \end{cases} \tag{10}$$

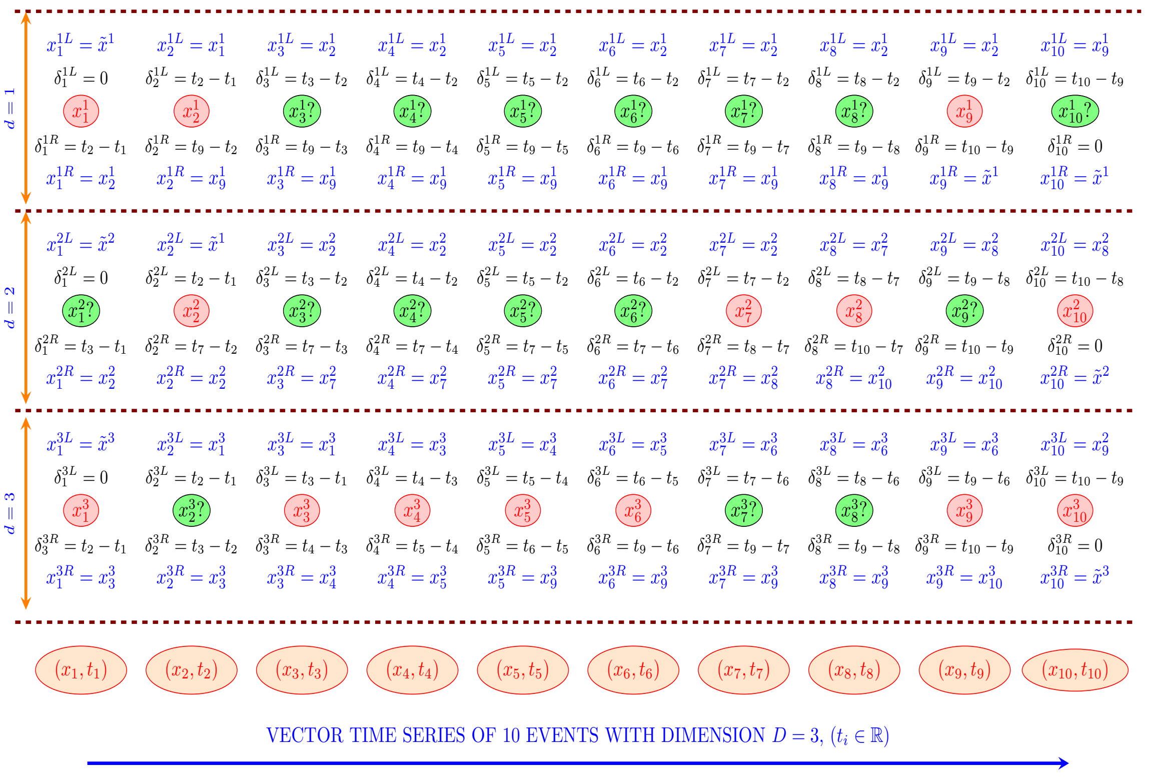
$d = 1$
$x_1^{1L} = \tilde{x}^1$ $x_2^{1L} = x_1^1$ $x_3^{1L} = x_2^1$ $x_4^{1L} = x_2^1$ $x_5^{1L} = x_2^1$ $x_6^{1L} = x_2^1$ $x_7^{1L} = x_2^1$ $x_8^{1L} = x_2^1$ $x_9^{1L} = x_2^1$ $x_{10}^{1L} = x_9^1$
$\delta_1^{1L} = 0$ $\delta_2^{1L} = t_2 - t_1$ $\delta_3^{1L} = t_3 - t_2$ $\delta_4^{1L} = t_4 - t_2$ $\delta_5^{1L} = t_5 - t_2$ $\delta_6^{1L} = t_6 - t_2$ $\delta_7^{1L} = t_7 - t_2$ $\delta_8^{1L} = t_8 - t_2$ $\delta_9^{1L} = t_9 - t_2$ $\delta_{10}^{1L} = t_{10} - t_9$
x_1^1 x_2^1 $x_3^1?$ $x_4^1?$ $x_5^1?$ $x_6^1?$ $x_7^1?$ $x_8^1?$ x_9^1 $x_{10}^1?$
$\delta_1^{1R} = t_2 - t_1$ $\delta_2^{1R} = t_9 - t_2$ $\delta_3^{1R} = t_9 - t_3$ $\delta_4^{1R} = t_9 - t_4$ $\delta_5^{1R} = t_9 - t_5$ $\delta_6^{1R} = t_9 - t_6$ $\delta_7^{1R} = t_9 - t_7$ $\delta_8^{1R} = t_9 - t_8$ $\delta_9^{1R} = t_{10} - t_9$ $\delta_{10}^{1R} = 0$
$x_1^{1R} = x_2^1$ $x_2^{1R} = x_9^1$ $x_3^{1R} = x_9^1$ $x_4^{1R} = x_9^1$ $x_5^{1R} = x_9^1$ $x_6^{1R} = x_9^1$ $x_7^{1R} = x_9^1$ $x_8^{1R} = x_9^1$ $x_9^{1R} = \tilde{x}^1$ $x_{10}^{1R} = \tilde{x}^1$
$d = 2$
$x_1^{2L} = \tilde{x}^2$ $x_2^{2L} = \tilde{x}^1$ $x_3^{2L} = x_2^2$ $x_4^{2L} = x_2^2$ $x_5^{2L} = x_2^2$ $x_6^{2L} = x_2^2$ $x_7^{2L} = x_2^2$ $x_8^{2L} = x_7^2$ $x_9^{2L} = x_8^2$ $x_{10}^{2L} = x_8^2$
$\delta_1^{2L} = 0$ $\delta_2^{2L} = t_2 - t_1$ $\delta_3^{2L} = t_3 - t_2$ $\delta_4^{2L} = t_4 - t_2$ $\delta_5^{2L} = t_5 - t_2$ $\delta_6^{2L} = t_6 - t_2$ $\delta_7^{2L} = t_7 - t_2$ $\delta_8^{2L} = t_8 - t_7$ $\delta_9^{2L} = t_9 - t_8$ $\delta_{10}^{2L} = t_{10} - t_8$
$x_1^2?$ x_2^2 $x_3^2?$ $x_4^2?$ $x_5^2?$ $x_6^2?$ x_7^2 x_8^2 $x_9^2?$ x_{10}^2
$\delta_1^{2R} = t_3 - t_1$ $\delta_2^{2R} = t_7 - t_2$ $\delta_3^{2R} = t_7 - t_3$ $\delta_4^{2R} = t_7 - t_4$ $\delta_5^{2R} = t_7 - t_5$ $\delta_6^{2R} = t_7 - t_6$ $\delta_7^{2R} = t_8 - t_7$ $\delta_8^{2R} = t_{10} - t_7$ $\delta_9^{2R} = t_{10} - t_9$ $\delta_{10}^{2R} = 0$
$x_1^{2R} = x_2^2$ $x_2^{2R} = x_2^2$ $x_3^{2R} = x_7^2$ $x_4^{2R} = x_7^2$ $x_5^{2R} = x_7^2$ $x_6^{2R} = x_7^2$ $x_7^{2R} = x_8^2$ $x_8^{2R} = x_{10}^2$ $x_9^{2R} = x_{10}^2$ $x_{10}^{2R} = \tilde{x}^2$
$d = 3$
$x_1^{3L} = \tilde{x}^3$ $x_2^{3L} = x_1^3$ $x_3^{3L} = x_1^3$ $x_4^{3L} = x_3^3$ $x_5^{3L} = x_4^3$ $x_6^{3L} = x_5^3$ $x_7^{3L} = x_6^3$ $x_8^{3L} = x_6^3$ $x_9^{3L} = x_6^3$ $x_{10}^{3L} = x_9^2$
$\delta_1^{3L} = 0$ $\delta_2^{3L} = t_2 - t_1$ $\delta_3^{3L} = t_3 - t_1$ $\delta_4^{3L} = t_4 - t_3$ $\delta_5^{3L} = t_5 - t_4$ $\delta_6^{3L} = t_6 - t_5$ $\delta_7^{3L} = t_7 - t_6$ $\delta_8^{3L} = t_8 - t_6$ $\delta_9^{3L} = t_9 - t_6$ $\delta_{10}^{3L} = t_{10} - t_9$
x_1^3 $x_2^3?$ x_3^3 x_4^3 x_5^3 x_6^3 $x_7^3?$ $x_8^3?$ x_9^3 x_{10}^3
$\delta_1^{3R} = t_2 - t_1$ $\delta_2^{3R} = t_3 - t_2$ $\delta_3^{3R} = t_4 - t_3$ $\delta_4^{3R} = t_5 - t_4$ $\delta_5^{3R} = t_6 - t_5$ $\delta_6^{3R} = t_9 - t_6$ $\delta_7^{3R} = t_9 - t_7$ $\delta_8^{3R} = t_9 - t_8$ $\delta_9^{3R} = t_{10} - t_9$ $\delta_{10}^{3R} = 0$
$x_1^{3R} = x_3^3$ $x_2^{3R} = x_3^3$ $x_3^{3R} = x_4^3$ $x_4^{3R} = x_5^3$ $x_5^{3R} = x_9^3$ $x_6^{3R} = x_9^3$ $x_7^{3R} = x_9^3$ $x_8^{3R} = x_9^3$ $x_9^{3R} = x_{10}^3$ $x_{10}^{3R} = \tilde{x}^3$
(x_1, t_1) (x_2, t_2) (x_3, t_3) (x_4, t_4) (x_5, t_5) (x_6, t_6) (x_7, t_7) (x_8, t_8) (x_9, t_9) (x_{10}, t_{10})
VECTOR TIME SERIES OF 10 EVENTS WITH DIMENSION $D = 3$, $(t_i \in \mathbb{R})$

Figure 1: Illustration of various variables (δ_i^{dL}, δ_i^{dR}, x_i^{dL}, x_i^{dR}) that determine the modified input into GRU-unit (in GRU-M) at each time-step. Green - missing, Red - available.

At the right end of the time-series, the closest available data point must be at zero distance, hence $\delta_N^{dR} = 0$. For any other observation i, if the right immediate observation is available ($m_{i+1} = 0$), then the time distance $\delta_i^{dR} = t_{i+1} - t_i$. If the right immediate observation is not available ($m_{i+1} = 0$), then one can recursively compute this as the sum of (i)time distance to the immediate right observation ($t_{i+1} - t_i$), (ii) distance from the immediate right observation to *its* immediately available data point to the right, δ_{i+1}^{dR}.

Please refer to Fig. 1 for a detailed illustration of δ_i^{dR} and δ_i^{dL} on an example data sequence. For instance, consider the first component ($d = 1$) of the input at time-step 4 (x_4^1), in Fig. 1. The associated entry is marked in green indicating a missing entry. The closest available data point to the left in the first component is at time-step 2. Note that available data points are marked in red. Hence δ_i^{dL}, the time distance to the closest available data point to the left is $(t_4 - t_2)$. The closest available data point to the right in the first component is at time-step 9. Hence δ_i^{dR}, the time distance to the closest available data point to the right is $(t_9 - t_4)$.

Instead of learning one decay or weight factor between 0 and 1, we learn three factors as follows. Let $\delta_i^d = [\delta_i^{dL}, \delta_i^{dR}]$.

$$\Gamma_i^d = F(\delta_i^d), \text{where } F(.) \text{ denotes a feed-forward map}, \Gamma_i^d \text{ is } (3 \times 1) \text{ vector.} \qquad (11)$$

$$\gamma_i^{dL} = \frac{exp(\Gamma_i^d(1))}{\sum_{j=1}^3 exp(\Gamma_i^d(j))},\ \gamma_i^{dR} = \frac{exp(\Gamma_i^d(2))}{\sum_{j=1}^3 exp(\Gamma_i^d(j))},\ \gamma_i^{dm} = \frac{exp(\Gamma_i^d(3))}{\sum_{j=1}^3 exp(\Gamma_i^d(j))} \tag{12}$$

where

$$\gamma_i^{dL} + \gamma_i^{dR} + \gamma_i^{dm} = 1 \tag{13}$$

$F(.)$ denotes a general feed-forward map with a linear activation at the output layer. Hence, $F(.)$ in its simplest form with no hidden nodes would be a linear transformation, namely

$$F(\delta_i^d) = W_M^d \delta_i^d + b_M^d. \quad (\text{where } W_M^d \text{is } (3\times 2), b_M^d \text{is } (3\times 1)) \tag{14}$$

Essentially, the above equations model a (potentially) general feed-forward structure (FFN blocks in Fig. 2) with two inputs $[\delta_i^L, \delta_i^R]$, and three outputs $[\Gamma_i^d(1), \Gamma_i^d(2), \Gamma_i^d(3)]$ followed by a *softmax* transformation. The modified input at time i to the GRU unit is now a convex combination of three quantities, x_i^{dL}, x_i^{dR} (the closest observation to the right of i) and $\tilde{x}^d$, where the co-efficients come from the above described 3-output softmax layer.

$$\hat{x}_i^d = m_i^d x_i^d + (1 - m_i^d)(\gamma_i^{dL} x_i^{dL} + \gamma_i^{dR} x_i^{dR} + \gamma_i^{dm} \tilde{x}^d) \tag{15}$$

Entities x_i^{dR}, x_i^{dL} can be efficiently computed recursively as follows.

$$x_i^{dL} = \begin{cases} x_{i-1}^d & \text{if } i > 1, m_{i-1} = 1 \\ x_{i-1}^{dL} & i > 1, m_{i-1} = 0 \\ \tilde{x}^d, & i = 1 \end{cases} \tag{16}$$

$$x_i^{dR} = \begin{cases} x_{i+1}^d & \text{if } i < N, m_{i+1} = 1 \\ x_{i+1}^{dR} & i < N, m_{i+1} = 0 \\ \tilde{x}^d, & i = N \end{cases} \tag{17}$$

At the right end, x_i^{dR} has strictly no closed observation to the right. It would not be unreasonable to assume x_N^{dR} to be $\tilde{x}^d$, mean of the d^{th} variable and we adopt this convention even on the left (in particular on x_1^{dL}). This can be verified in Fig. 1 where x_1^{dL} and x_{10}^{dR} ($N = 10$ here) are chosen to be $\tilde{x}^d$. If the immediate observation to the right is available, then $x_i^{dR} = x_{i+1}^d$. If not, then x_i^{dR} must be same as the immediate observation to the right of the $(i+1)^{th}$ observation, and hence is updated recursively to x_{i+1}^{dR}. An analogous update is performed from the left for x_i^{dL}.

As an illustration look at time-step 4 in the first component ($d = 1$) of the input in Fig. 1. $x_4^{1L} = x_2^1$ as the closest available point to the left is at t_2. Similarly $x_4^{1R} = x_9^1$ as the closest available point to the right is at t_9. Fig. 1 illustrates the various variables on an example time-series with 10 observations. The input dimension, D is chosen to be 3. The red circles represent available observations while the green ones indicated missing values. The various variables δ_i^{jL}, δ_i^{jR} (eqn. (7) & eqn. (10)) and x_i^{dL}, x_i^{dR} (eqn. (16) & eqn. (17)) are clearly explained in this figure via the explicit values it takes on in all 3 components.

Fig. 2 explains the overall scheme of the modified input at the i^{th} step of GRU-M. It considers a 2-dimensional ($D = 2$) input. If $m_i^d = 1$, i.e. data is available, then $\hat{x}_i^d$ is just x_i^d. If $m_i^d = 0$, i.e. data is missing, then the entire architecture comes into play. Lets stick to $d = 1$ for illustration. Both δ_i^{1L} and δ_i^{1R} are processed by a feed forward network (FFN1)

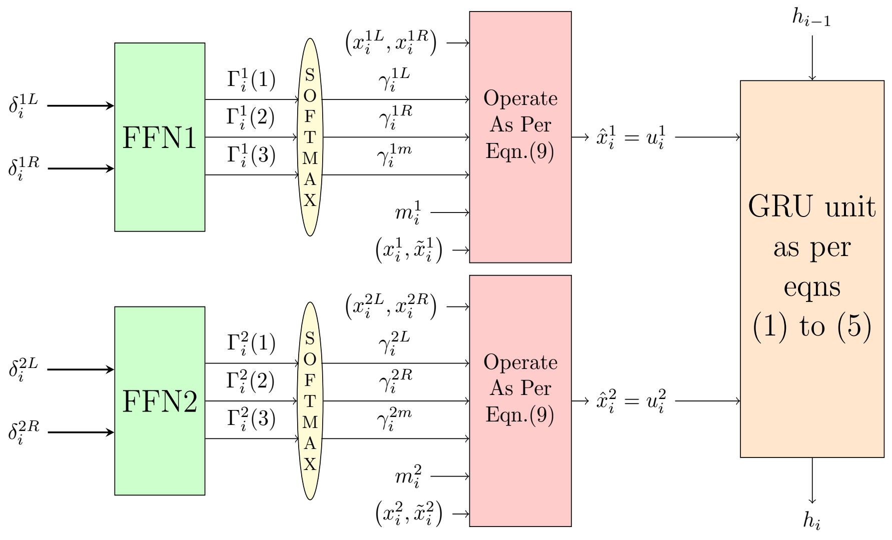

Figure 2: Illustration of computation of the modified input ($\hat{x}_i^d$) which is fed at the input (u_i^d) position of the GRU unit at i^{th} time-step of unfolded RNN, Input Dimension, $D = 2$ here, h_{i-1} comes from previous ($(i-1)^{th}$) time-step of unfolded RNN.

which implements eqn. (11) with a linear activation at the output. The 3 outputs of the FFN1 are passed through a softmax layer to obtain the 3 factors γ_i^{dL} γ_i^{dR} and γ_i^{dm}. These 3 factors in conjunction with x_i^{1L} (closest available value to the left), x_i^{1R} (closest available value to the right), $\tilde{x}_i^1$ (mean of the 1^{st} component) and m_i^1 (masking variable) are fed to the next block which implements eqn. (15). The output of this block is $\tilde{x}_i^1$ which is fed at u_i^1, the 1^{st} input position of the GRU block. A similar story holds for the 2^{nd} input position of the GRU-block at the i^{th} time-step.

Advantages of GRU-M: The idea of the above parametrization is as follows. For large gaps, when the missing observation (where $m_i = 0$) is closer to the left end of the gap, $\hat{x}_i^d$ must be close to x_i^{dL}. Similarly when missing observation closer to the right end, $\hat{x}_i^d$ should be close to x_i^{dR}. When missing observation is around the center of the gap, the true value may have no bearing with the left-end or right-end value. Hence in the mid-gap region, it may be best to impute with $\tilde{x}^d$, which our GRU-M scheme can capture. On the other hand, when the gap length ($\delta_i^L + \delta_i^R$) is small or medium, the above parametrization should ideally ignore $\tilde{x}^d$ completely, as the time-series true value may actually not come anywhere close to the mean $\tilde{x}^d$. For instance, when both x_i^{dL} and x_i^{dR} are close in value, while $\tilde{x}^d$ is relatively much farther from either of them, $\tilde{x}^d$ must be ignored while evaluating $\hat{x}_i^d$. *Our proposed parametrization can capture all such important features.*

Model Parameters: Compared to GRU-D, while GRU-M captures many more influences during imputation as explained above, this comes at a cost of some additional parameters. While the standard GRU unit parameters are same in both GRU-D and GRU-M, the difference lies in how $\hat{x}_i$ is evaluated. GRU-D, uses two $2D$ extra parameters for this, where D parameters come from the bias vector b_γ and remaining D parameters from the diagonal of W_γ matrix (refer eqn. (8)). In contrast, in GRU-M, for each component (or the d^{th} component), we need to additionally store at least a matrix W_M^d and a vector b_M^d. Recall from eqn. (14), that W_M^d is (3×2) while b_d^M is (3×1), hence contributing to 9 parameters. Hence in total GRU-M uses $9D$ extra parameters at least. In comparison to GRU-D, GRU-M needs $7D$ extra parameters to capture the additional influences that GRU-D completely misses. This is not much, while training with these extra parameters gives us superior predictions as demonstrated in our experiments.

GRU-D in addition to input decay captures missingness patterns by incorporating a hidden state decay from the left. This aspect of GRU-D can also be extended. However in this paper, for purposes of testing GRU-M and GRU-D, we stick to input-decay only.

3.3. GRU-M unit for sequence prediction (classification/forecasting)

We have till now discussed how sequential inputs with missing entries from a (vector) time-series are processed by an RNN layer with the proposed GRU-M unit. Suppose one wants to perform classification of a sequence (with missing entries). This sequence needs to be processed by an RNN layer with GRU-M unit. One needs to feed the state output from the last time-step to a soft-max layer and train using a standard cross-entropy loss, for instance.

Forecasting: As mentioned earlier we use a multi-step forecasting setting. *For accurate multi-step forecasting, we use an encoder-decoder Sutskever et al. (2014) model which consists of two RNN layers of the proposed GRU-M units.* For forecasting Wen et al. (2017), a historical window (of some fixed size) into the past starting from the forecast horizon is used as input for forecasting. This window also referred to as input window is fed sequentially into the encoder or the first RNN layer. The last state of the encoder RNN layer is typically fed as the initial state into the second RNN layer, which is the decoder. The decoder is unfolded into as many steps as the width of the forecast horizon. *Exogenous inputs (if any) of the input window are placed at the encoder inputs, while the exogenous inputs in the forecast horizon are placed as decoder inputs in a sequential fashion.*

4. Results

In this paper, we restrict our experimental validation of the GRU-M unit to multi-step forecasting task using an ED architecture as described above. *The data sets we use are all publicly available. We are unfortunately not in a position to share our codes due to proprietary reasons, but we believe we have shared full details of our proposed architecture using which any reader can implement the same.*

4.1. Data Sets, Error Metrics and Hyper-parameters

D1: Air Quality[1] data-set was recorded for a period of 1 year (Mar 2004 to Feb 2005) on an hourly basis. The data is *naturally missing* here. It contains 9358 points of average response from 5 metal oxide sensors. We use CO(GT) data, from one of these sensors (denoted as D1 from now on) to measure the performance of our model.
D2: Electricity data-set[2] is a publicly available data-set which measured electricity consumption in kWH of 321 clients for every hour. We divided the whole data-set into 3 groups

1. github.com/mehak25/BiGAN/tree/master/data/air/initial
2. https://github.com/laiguokun/multivariate-time-series-data

namely low, medium and high based on average consumption. From each group, we choose one sequence randomly and denote this collection of 3 sequences as D2.
D3: M5[3] is a publicly available data-set distributed across 12 aggregation levels which contains daily sales of different products for 5.4 years. We pick sequences from level 10 which is the aggregated sales for each product across all stores and states. While this level of data contains 3049 sequences, we pick 4 sequences (referred to as $D3$ from now on) based on sufficient total variation. The idea is higher the total variation of the sequence, more challenging would be the forecasting. We employ synthetic masking on these 4 sequences. The total variation we employ is defined below:

$$TV(f) = \frac{1}{L-h+1}\sum_{i=1}^{L-h+1}\frac{1}{h}\sum_{j=i}^{i+h-1}|f(j+1)-f(j)|,$$

where L is the total length of the sequence and h is the size of the moving window ($h \ll L$). TV can be defined without a moving window with just the inner summation, but a high TV based on this definition will not necessarily capture sufficient variation across the sequence. We use a small/local moving window to check for sufficient variation across the length of the sequence.

We consider following well-known error metrics to measure performance: (1)**MSE** (Mean Square Error) (2)**MAPE** (Mean Absolute Percentage Error) (3)**MASE** (Mean Absolute Scale Error Hyndman and Koehler (2006)). While MSE is a scale dependent metric, the other two are scale independent, while being complementary to each other.

The APE is relative error (RE) expressed in percentage. If $\widehat{X}$ is predicted value, while X is the true value, RE $= (\widehat{X} - X)/X$. In the multi-step setting, APE is computed for each step and is averaged over all steps to obtain the MAPE for one window of the prediction horizon. APE while has the advantage of being a scale independent metric, can assume abnormally high values and can be misleading when the true value is very low. An alternative complementary error metric which is scale-free could be MASE.

The MASE is computed with reference to a baseline metric. The choice of baseline is typically the *copy previous* predictor, which just replicates the previous observed value as the prediction for the next step. For a given window of one prediction horizon of K steps ahead, let us denote the i^{th} step error by $|\widehat{X}_i - X_i|$. The i^{th} scaled error is defined as

$$e_s^i = \frac{|\widehat{X}_i - X_i|}{\frac{1}{n-K}\sum_{j=K+1}^{n}|X_j - X_{j-K}|} \tag{18}$$

where n is no. of data points in the training set. The normalizing factor is the average i^{th} step-ahead error of the copy-previous baseline on the training set. Hence the MASE on a multi-step prediction window w of size K will be

$$MASE(w,K) = \frac{1}{K}\sum_{j=1}^{K} e_s^j \tag{19}$$

Table 1 represents the broad choice of hyper-parameters used for training.

3. https://www.kaggle.com/competitions/m5-forecasting-accuracy/data

Table 1: Hyper parameters during training.

Parameters	**Description (D1/D2/D3)**
Batch size	256/512/256
Learning rate	0.01/0.05/0.001
Hidden state vector size	16/16/20
Optimizer	Adam/RMSProp/Adam

4.2. Baselines and Masking

We consider a wide range of baselines to benchmark our method, GRU-M against: (1) EDC - An Encoder-Decoder (Seq2Seq) predictive model which first imputes using cubic spline interpolation (using a piece-wise cubic polynomial that is twice differentiable) before model building. (2) BRITS Cao et al. (2018) - A bi-directional RNN approach for joint imputation and prediction exploiting information from both ends of a data gap. (3) RITS Cao et al. (2018) - unidirectional version of BRITS. (4) Bi-GAN Gupta et al. (2021) - A GAN based jointly impute and learn technique incorporating information from either ends of a gap. (5) GRU-D Che et al. (2016) - An ED model with a GRU-D unit (adopting a joint impute and learn strategy) in both encoder and decoder.

The first approach above is based on imputing first (using a strong technique) followed by predictive model building using an ED model on complete imputed data. The rest of the baselines adopt a joint impute and learn approach using some distinct form of an RNN architecture and training approach. Hence our bench-marking enables a diverse comparison with state-of-art baselines.

4.2.1. Assessing significance of mean error differences statistically

We have conducted a Welch t-test (unequal variance) based significance assessment (across all relevant experiments) under all the mean metrics (MSE,MASE,MAPE) differences (Proposed vs Baseline) with a significance level of 0.05 for null hypothesis rejection. The best performing method's error is highlighted in bold if its MASE/MAPE improvement over every other method is statistically significant. We allow for highlighting the second/third best errors in situations when the mean error differences between the best and second/third best errors are statistically insignificant.

4.2.2. Synthetic Masking

We traverse the data sequentially and at each step we ask whether to mask at the current step or not. To perform this, we toss a coin at every step with an adaptive/varying q (probability of head). On seeing a tail, we do nothing and move ahead. If heads, we need to mask, in particular decide the number of consecutive steps ahead (τ_w) that need to be masked. For this, we consider a bag of window lengths T_w, from which we uniformly sample to obtain τ_w. Depending on the length τ_w sampled, the heads probability q is updated as $q = \frac{c}{\tau_w}$, where c is a control parameter. If $c = 1$, the strategy masks about 50% of the points, because for every τ_w points masked, the next τ_w points (in expectation) are retained. In both D2 and D3, we have chosen $c = 0.2$, which means we mask about 16% of the points.

Synthetic masking advantage: Since underlying actual data is known, one can test the performance of trained models on *all* points of a separate test set in the forecast horizon.

4.3. Results on D1 (data naturally missing)

For D1, input window size was set as 20 while the prediction horizon was varied over $8, 12$ and 16 steps. We separately kept aside 10% data for testing our model. This means we measure the performance of our method on 909, 905 and 901 test examples for prediction horizon $8, 12$ and 16 respectively. Table 2 gives a detailed account of the various errors, vindicating GRU-M's superior performance.

Specifically, in MASE terms, GRU-M shows a minimum (statistically significant) improvement of 0.15 over all baselines in the best case scenario (Prediction horizon = 12). Please note in other two scenarios also, the minimum improvement over all other baselines is a very healthy 0.14. In MAPE terms, GRU-M shows a minimum improvement of 8% across all baselines in the best case scenario (Prediction horizon = 8). In other two scenarios too, the minimum improvement over all baselines is a statistically significant 7% and 5% respectively. In MSE terms, GRU-M outperforms all baselines across all prediction horizons.

Table 2: Results on D1 (naturally missing) for different forecast horizons.

	Prediction horizon = 8			Prediction horizon = 12			Prediction horizon = 16		
Method	MASE	MAPE	MSE	MASE	MAPE	MSE	MASE	MAPE	MSE
EDC	1.07	39	1.78	0.91	35	1.72	0.86	37	1.63
Bi-GAN	1.14	32	2.48	1.22	38	3.33	1.39	48	4.39
RITS	1.05	32	1.97	0.94	32	1.95	0.88	32	1.95
BRITS	1.05	32	1.97	0.94	32	1.97	0.88	32	1.95
GRU-D	0.79	42	1.76	0.78	42	1.76	0.74	42	1.76
GRU-M	**0.65**	**24**	**0.77**	**0.63**	**25**	**0.87**	**0.62**	**27**	**0.94**

4.4. Results on D2 (data synthetically masked)

We perform masking on D2 and D3 to generate missingness as discussed above. For D2, we set width of input window as 20 and prediction horizon at 12. This implies we are predicting for the next 12 hours ahead. A test size of 10% was separately set aside for each of the 3 series to measure performance of our model. Table 3 demonstrates GRU-Ms superior performance compared to the baselines. *In both the current and next experiment, performance of Bi-GAN was very poor (in particular MAPE was > 100%). Hence we have not reported Bi-GAN results in Tab. 3 and Tab. 4.*

In particular, in terms of MASE/MAPE, GRU-M shows a minimum improvement of 0.05 (Medium consumption) and 13% (Medium) respectively over all baselines in the best case scenario for the medium category. Even in high category, GRU-M outperforms all baselines based on all metrics in a statistically significant fashion. In the low category too, except based on MASE metric where the performance is comparable to GRU-D, GRU-M outperforms all baselines in a statistically significant fashion.

Table 3: Results on D2: Masking Window Lengths $T_w = \{5, 6, \ldots, 12\}$, Parameter $c = 0.2$

	Low			Medium			High		
Method	MASE	MAPE	MSE	MASE	MAPE	MSE	MASE	MAPE	MSE
EDC	0.55	23	1135	0.68	25	14941	0.37	14	110529
RITS	0.89	31	3222	1.27	39	45221	1.36	45	977696
BRITS	0.88	33	2804	1.03	33	35518	1.04	36	637439
GRU-D	**0.43**	23	1167	0.36	24	14468	0.32	22	198071
GRU-M	**0.51**	**22**	**1046**	**0.31**	**11**	**4739**	**0.28**	**11**	**74424**

4.5. Results on D3 (data synthetically masked)

Here, the input window size is set to 20, while forecast horizon was chosen to be 5 days ahead. A separate 97 days out of 1941 was kept aside for testing which implies we use 73 examples to measure the performance of our method. Table 4 represents the errors of all relevant methods, which indicates the superior or comparable performance of GRU-M in terms of all three error metrics, compared to all baselines.

In particular, under MSE metric, GRU-M though is one of the best performing methods, it has comparable performance with EDC (also indicated in bold) on sequences $1, 2$. In terms of MASE/MAPE, GRU-M shows a minimum improvement of 0.04 and 7% respectively over all baselines, in the best case scenario (Seq 3). Our results based on MASE/MAPE indicate that GRU-D and EDC have comparable performance with GRU-M on some sequences.

Table 4: Results on D3: Masking Window Lengths $T_w = \{5, 6, \ldots, 12\}$, Parameter $c = 0.2$

	Seq 1			Seq 2			Seq 3			Seq 4		
Method	MASE	MAPE	MSE	MASE	MAPE	MSE	MASE	MAPE	MSE	MASE	MAPE	MSE
EDC	0.65	**19**	**3293**	0.43	14	**1732**	0.62	22	9366	0.63	41	1759
RITS	0.72	24	3740	1.45	50	13219	1.08	36	33177	0.71	65	2143
BRITS	0.85	30	5325	1.17	40	8750	0.98	31	33945	0.54	49	1273
GRU-D	**0.59**	23	4098	0.38	16	1874	0.40	19	7198	**0.42**	**39**	**1106**
GRU-M	**0.56**	**17**	**2589**	**0.34**	**10**	**1706**	**0.38**	**12**	**4647**	0.48	**39**	**1064**

5. Conclusions

We proposed a novel architecture (GRU-M) for sequential learning under missing data. Its a joint impute and learn approach where the parameterized functions which impute and perform predictive modelling are simultaneously learnt. It factors in information not only from both ends of a data gap, but also the distance from the left and right-end of the gap in a novel fashion, distinct from existing approaches. It can also incorporate state-decay from the right if necessary whenever there are bidirectional layers, which can arise during forecasting. Overall, it can be viewed as a non-trivial generalization of GRU-D Che et al. (2016), which is a state-of-art technique for the same problem. Our proposed approach can be employed for both sequence classification and sequence forecasting. Based on 3 diverse metrics, we bench-marked our approach on data sets where data was either naturally missing or synthetically masked. We compared against a range of diverse, but closely related state-of-art RNN approaches for sequence forecasting under missing data. Our results clearly vindicate the viability of our proposed approach. As future work, we would like to test our proposed architecture on sequential classification.

References

Yoshua Bengio and Francois Gingras. Recurrent neural networks for missing or asynchronous data. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, page 395–401, Cambridge, MA, USA, 1995. MIT Press.

Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. Brits: Bidirectional recurrent imputation for time series. *Advances in Neural Information Processing Systems 31*, pages 6775–6785, 2018.

Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8, Jun 2016. doi: 10.1038/s41598-018-24271-9.

Yagmur Gizem Cinar, Hamid Mirisaee, Parantapa Goswami, Eric Gaussier, and Ali Aït-Bachir. Period-aware content attention rnns for time series forecasting with missing values. *Neurocomputing*, 312:177–186, 2018. ISSN 0925-2312.

Andrea Cini, Ivan Marisca, and Cesare Alippi. Filling the g_ap_s: Multivariate time series imputation by graph neural networks. In *International Conference on Learning Representations*, 2021.

Joseph David Futoma. *Gaussian Process-Based Models for Clinical Time Series in Healthcare.* PhD thesis, Statistical Science, Duke University, 2018.

Pedro J. García-Laencina, José-Luis Sancho-Gómez, and Aníbal R. Figueiras-Vidal. Pattern classification with missing data: A review. *Neural Comput. Appl.*, 19(2):263–282, March 2010. ISSN 0941-0643. doi: 10.1007/s00521-009-0295-6.

Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(5):855–868, May 2009.

Nicole Gruber and Alfred Jockisch. Are gru cells more specific and lstm cells more sensitive in motive classification of text? *Front. Artif. Intell.*, 2020. doi: 10.3389/frai.2020.00040.

Mehak Gupta, Thao-Ly T. Phan, H. Timothy Bunnell, and Rahmatollah Beheshti. Concurrent imputation and prediction on ehr data using bi-directional gans: Bi-gans for ehr imputation and prediction. *In Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB '21)*, pages 1–9, 2021.

Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

Sharon A. Johnson. *Splines*, pages 1443–46. Springer US, 2013. ISBN 978-1-4419-1153-7.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

Steven Cheng-Xian Li. *Learning from Irregularly-Sampled Time Series.* PhD thesis, Steven Cheng-Xian Li, University of Massachusetts Amherst, 2020.

Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Phil. Trans. R. Soc. A.*, 379(2194):40–48, 2021.

Zachary C Lipton, David Kale, and Randall Wetzel. Directly modeling missing data in sequences with rnns: Improved classification of clinical time series. In *Proc. of the 1st Machine Learning for Healthcare Conference*, volume 56, pages 253–270. PMLR, 2016.

Yonghong Luo, Xiangrui Cai, Ying ZHANG, Jun Xu, and Yuan xiaojie. Multivariate time series imputation with generative adversarial networks. In *Advances in Neural Information Processing Systems*, volume 31, pages 1603–1614, 2018.

Debashis Mondal and Donald Percival. Wavelet variance analysis for gappy time series. *Annals of the Institute of Statistical Mathematics*, 62(5):943–966, October 2010.

Tong Nie, Guoyang Qin, Wei Ma, Yuewen Mei, and Jian Sun. Imputeformer: Low rankness-induced transformers for generalizable spatiotemporal imputation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 2260–2271, New York, NY, USA, 2024.

Soumen Pachal and Avinash Achar. Sequence prediction under missing data: An rnn approach without imputation. In *Proc. of the 31st ACM Int. Conf. on Information & Knowledge Management*, CIKM '22, page 1605–1614, 2022.

S. Parveen and P. D. Green. Speech recognition with missing data using recurrent neural nets. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, page 1189–1195, Cambridge, MA, USA, 2001.

Mirco Ravanelli, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio. Light gated recurrent units for speech recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(2):92–102, Apr 2018.

Kira Rehfeld, Norbert Marwan, Jobst Heitzig, and Jürgen Kurths. Comparison of correlation analysis techniques for irregularly sampled time series. *Nonlinear Processes in Geophysics*, 18(3):389–404, July 2011. doi: 10.5194/npg-18-389-2011.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, pages 3104–3112, 2014.

Volker Tresp and Thomas Briegel. A solution for missing data in recurrent neural networks with an application to blood glucose prediction. In *Proceedings of the 10th International Conference on Neural Information Processing Systems*, page 971–977, Cambridge, MA, USA, 1997. MIT Press.

Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. In *The 31st Conference on Neural Information Processing Systems (NIPS 2017), Time Series Workshop*, 2017.

Zhengzheng Xing, Jian Pei, and Eamonn Keogh. A brief survey on sequence classification. *SIGKDD Explor. Newsl.*, 12(1):40–48, November 2010. ISSN 1931-0145.