# MEMORY LEARNING OF MULTIVARIATE ASYNCHRONOUS TIME SERIES

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Sequential observations from complex systems are usually collected irregularly and asynchronously across variables. Besides, they are typically both serially and cross-sectionally dependent. Recurrent networks are always used to model such sequential data, trying to simultaneously capture marginal dynamics and dependence dynamics with one shared memory. This leads to two problems. First, some heterogeneous marginal information is difficult to be preserved in the shared memory. Second, in an asynchronous setting, missing values across variables will introduce bias in the shared memory. To solve these problems, this paper designs a new architecture that seamlessly integrates continuous-time ODE solvers with a set of memory-aware GRU blocks. It learns memory profiles separately and addresses the issue of asynchronous observations. Numerical results confirm that this new architecture outperforms a variety of state-of-the-art baseline models on datasets from various fields.

## 1 INTRODUCTION

Multivariate asynchronous time series (MATS) widely exist in various domains. For example, in astronomy, a periodic signal hidden in noise is usually detected unevenly (Scargle, 1982). In health care, patients can only take some measurements but not all at each time point (Jensen et al., 2014). In activity recognition, data are collected from wearable sensors on devices at different frequencies (Kaluža et al., 2010). However, there are some challenges when modeling MATS.

First, a single variable in the MATS is sampled irregularly. In such sequences, observations appear randomly, and the observed time intervals are not uniform. Recurrent Neural Networks (RNNs) perform not ideally when they are applied to this kind of data since RNNs do not have any built-in notions of time

Second, the randomness of observed time points in univariate time series will make the multivariate time series asynchronous. Compared to the multivariate synchronous time series (see Figure 1(a)), the multivariate asynchronous time series (see Figure 1(b)) is more common in real life. However, it is more difficult to model since some variables have missing values at a particular observed time point (e.g., variables $X_t^1$ and $X_t^3$ have no observations at time point $t_1$ in Figure 1(b))).

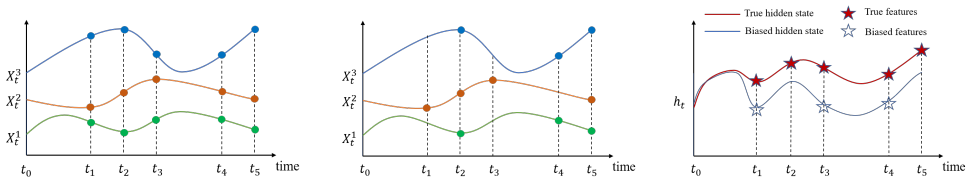


Figure 1: (a), (b) are the examples for multivariate synchronous/asynchronous time series. The multivariate time series contains three variables $X_t^1, X_t^2, X_t^3$, which are observed at different time points (e.g., $t_0, ..., t_5$). The corresponding color points suggest that variables have observations. (c) is the hidden state/memory for asynchronous time series in (b).

Third, each variable in MATS is serial dependent, and the critical quantity that we want to capture is the dynamics of the marginal conditional distribution of each variable. The serial dependencies are heterogeneous across variables (e.g., some variables have strong serial dependencies while some have weak serial dependencies, or some variables have long-range dependencies while some have short-range dependencies).

Fourth, each variable in MATS is not independent but depends on each other, and the target quantity that we want to model is the dynamics of the joint conditional distribution. In general, the joint conditional distribution contains additional information compared to the marginal distribution of each variable. This additional information is called the dependence structure, which describes the dependence or inter-correlation among variables. A typical way to extract the dependence structure from joint distributions is the Copula function (Trivedi & Zimmer, 2007) in statistics. However, modeling MATS is more complicated as the marginal distribution for each variable and the dependence structure among the variables are constantly changing over time.

Generally, there are two classes of approaches to fit multivariate time series. The first class models individual variables with the independent univariate model, which can capture the marginal dynamics perfectly even though the serial dependencies are heterogeneous across variables. Some statistical methods in this class include the ARMA models using the Box-Jenkins methodology (Makridakis & Hibon, 1997), and state space models (Hyndman et al., 2008). Some deep learning models also attempt to train each variable separately to reduce overfitting to noisy observations (Zhang et al., 2019). However, this class of methods abandons the dependence structure among variables.

Another class of approaches tries to capture marginal and joint dynamics simultaneously, where all variables are jointly modeled and share the same memory (also called hidden state). The most popular models used in this class are RNNs, which excel at extracting complex dynamic patterns across variables. Since the serial dependencies are heterogeneous across variables, the shared memory is hard to preserve some critical marginal information, such as long-term dependencies (Belletti et al., 2018), or statistical features (e.g., max/min/mean) of each variable (Harutyunyan et al., 2019). For example, when predicting the sales of $N$ products, each product has different characteristics (e.g., popularity or seasonality). Using shared memory to store the data of all products may forget some critical information about the individual product.

In the context of MATS, another problem of jointly modeling all variables is that the missing values in some variables will result in biased features (Zhang et al., 2021). If all the variables share one memory, the biased features will enter the shared memory. Thus predictions based on this shared memory are unreliable. For example, in Figure 1(c), the missing values at time points $t_1, t_3, t_4$ generate biased features at each time point such that the constructed memory is biased.

Based on the above analysis, we propose a new model architecture (CoGRUODE) targeting modeling the MATS from both marginal and joint perspectives. By seamlessly integrating two co-evolved GRUODE (Brouwer et al., 2019) networks, called marginal block and dependence block, our model takes advantage of the above two approaches at the same time. The marginal block captures the marginal dynamics and forms a unique marginal memory for each variable. In this way, the marginal information (such as long-range dependence and statistical features) of univariate are well preserved. Meanwhile, the evolution and update of the memory of variable $i$ will not affect the memories of variable $j$ ($j \neq i$), which also solves the problem of the adverse effect of missing values across variables. To avoid losing the information of the dependency structure, the dependence block extracts the updated information from the marginal block and forms a shared dependence memory. Ultimately, the memories from both blocks are used for prediction/classification. To demonstrate the superiority of CoGRUODE, we extensively test the model with three real-world datasets, i.e., the LSST dataset in UEA Repository, the Human Activity Dataset, and the Physionet 2012 Challenge Dataset. Our proposed model makes significant improvements compared with the various baselines.

## 2 Related Work

**Learning Multivariate Asynchronous Time Series**. Recurrent networks are the most widely used models for sequential data. However, it is inappropriate to handle MATS because of no built-in notion of time. A simple way to solve this problem is to input the time interval of each variable as an additional feature directly when training models (e.g., GRU-$\Delta t$). Furthermore, some papers propose to modify the architecture of classical RNNs by embedding the time intervals into models. For

example, CT-GRU (Mozer et al., 2017) extends standard GRU by incorporating multiple time scales of memory and performing a context-dependent selection of time scales for information storage and retrieval. GRU-D (Che et al., 2018) adds a decay mechanism in the memory, where the memory will decay exponentially as the time interval grows.

Another stream of methods to tackle this kind of data is based on Neural Ordinary Differential Equation (NODE) (Chen et al., 2018). Neural ODE converts Neural Networks into continuous-time models and integrates the dynamics at arbitrarily small intervals. Following this idea, Latent ODE (Rubanova et al., 2019) embeds the Neural ODE into Variational Auto-encoder (VAE) (Kingma & Welling, 2013). However, Latent ODE only derives the dynamics in the generation phase rather than directly deriving the dynamics. The framework that overcomes this issue is the GRU-ODE-Bayes (GRUODE) (Brouwer et al., 2019), which converts the classical discrete-time GRU cell into the continuous-time GRU cell and derives the update formula for the Neural ODE part when there are no observations. Furthermore, Lechner & Hasani (2020) propose ODELSTM by converting standard LSTM to a continuous version to solve the problem of gradient vanishing and exploding.

**Learning Marginal and Dependence Dynamics Separately**. Multivariate time series contains marginal dynamics for each variable and dependence dynamics among variables simultaneously. Some statistical models thus analyze the multivariate time series by separating the marginal conditional distribution and dependence structure between each variable, such as VARMA models (Lütkepohl, 2006) (diagonal values represent the marginal information of individual variables and off-diagonal values stand for the dependence information among variables). However, VARMA models only apply linear dependence on past history marginally and linear dependence cross-sectionally.

In the machine learning community, some studies try to model the marginal dynamics and dependence dynamics separately. For example, to preserve long-range dependencies of individual variables, the block-diagonal RNN (Belletti et al., 2018) divides variables into a few groups and applies RNNs to each group, and then the output of each block is processed by a fully-connected layer. To store the statistical features of each variable, Channel-wise LSTM (Harutyunyan et al., 2019) trains each variable separately at the first LSTM layer and then concatenates the output and feeds it into another LSTM layer. Wang et al. (2019) propose a new global-local framework named Deep Factor for time series forecasting, where the global part captures the common latent factor for all variables and the local part captures the uncertainty factor for each variable, conditioned on the common latent factor. Zhang et al. (2020) propose mGRN to model the marginal conditional distribution and joint distribution with two different GRU directly. The models in all of the above studies outperform standard RNNs in a fair setting, proving the significance of the marginal-dependence training process.

## 3 BACKGROUND

**Notations**. Consider a dataset that includes $N$ multivariate time series with time length $T$. Each time series $i \in \{1, 2, ..., N\}$ is denoted as $\mathbf{X}_{i,t}$ which contains $D$ variables such that $\mathbf{X}_{i,t} = [X^1_{i,t}, X^2_{i,t}, ..., X^D_{i,t}]^T$. Each time series is measured at $K_i$ different time points which are specified by a time vector $\mathbf{t}_i = [t_{i,1}, t_{i,2}, ..., t_{i,K_i}]$, where $t_{i,K_i} \leq T$. The time intervals between two observational time points (e.g., $[t_{i,1}, t_{i,2}]$ and $[t_{i,2}, t_{i,3}]$) are not equally spaced. Given $i$ and a particular observed time point $t$, not all variables are measured, but at least one of the $D$ variables should have an observed value. We thus denote a mask matrix $\mathbf{m}_t = [m^1_t, \cdots, m^D_t]$ to indicate whether a value in $\mathbf{X}_t$ is observed. Specifically, if variable $X^d_t$ has an observation, we denote its value as $x^d_t$ and set mask $m^d_t = 1$. Otherwise, we denote its value as 0 and set mask $m^d_t = 0$. We denote $\sigma$ and $\tanh$ as the sigmoid function and the hyperbolic tangent function. We also denote $\odot$ as the Hadamard product. A typical example for this dataset is the Electronic Medical Records (EHR) from $N$ different patients. The data of one patient is a multivariate time series, which includes different measurements, such as heartbeat, pulse, blood pressure, etc. Each measurement occurs at different time points, and at each observational time point, there is at least one measurement that has observations.

**GRUODE.** Our model is built upon GRUODE (Brouwer et al., 2019), a continuous-time recurrent neural network for irregularly sampled time series. The evolution and update of memories in GRUODE are shown in Figure 2. GRUODE evolves the memory using a Neural ODE when there are no observations. Specifically, GRUODE divides the evolution of the memory into the ODE part and the Bayes part.

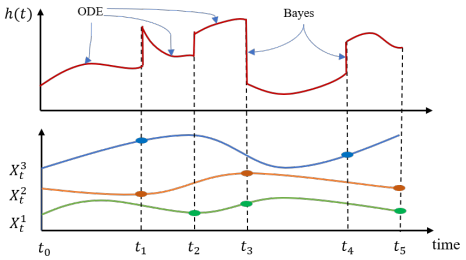


Figure 2: Example of evolution and update for the memory of GRUODE of asynchronous time series with three variables.

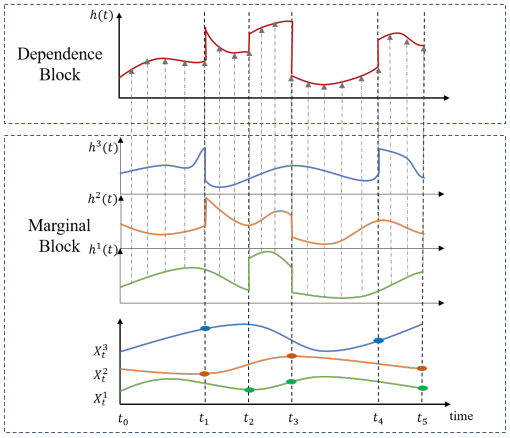


Figure 3: Example of evolution and update for the memory of CoGRUODE of asynchronous time series with three variables.

*ODE part* evolves the memory during two observational time points using

$$\begin{aligned}
\mathbf{r}(t) &= \sigma\left(W_r\mathbf{X}(t) + U_r\mathbf{h}(t) + b_r\right), \\
\mathbf{z}(t) &= \sigma\left(W_z\mathbf{X}(t) + U_z\mathbf{h}(t) + b_z\right), \qquad (1) \\
\tilde{\mathbf{h}}(t) &= \tanh\left(W_h\mathbf{X}(t) + U_h(\mathbf{r}(t)\odot\mathbf{h}(t)) + b_h\right), \\
\frac{d\mathbf{h}(t)}{dt} &= (1-\mathbf{z}(t))\odot(\tilde{\mathbf{h}}(t) - \mathbf{h}(t)), \qquad (2)
\end{aligned}$$

where $W_{(r,z,h)} \in \mathbb{R}^{H\times D}$, $U_{(r,z,h)} \in \mathbb{R}^{H\times H}$, $b_{(r,z,h)} \in \mathbb{R}^H$ are trainable parameters ($H$ denotes the size of memory). $\mathbf{r}(t)$, $\mathbf{z}(t)$, and $\tilde{\mathbf{h}}(t)$ are the reset gate, the update gate, and the candidate update gate which are all in $\mathbb{R}^H$. $\mathbf{h}(t)$ is the continuous memory and $\mathbf{X}(t)$ is the continuous observations (if continuous input is not available, then $\mathbf{X}(t) = 0$ and equation 2 is autonomous).

*Bayes part* updates the memory using a standard GRU if there are observations for variables at a observed time point (such as time point $t_0, t_1, ..., t_5$ in Figure 2). The updated formula is

$$\mathbf{h}\left(t_+\right) = \mathbf{GRU}\left(\mathbf{h}\left(t_-\right), f_{prep}\left(\mathbf{X}_t, \mathbf{m}_t, \mathbf{h}\left(t_-\right)\right)\right), \qquad (3)$$

where $\mathbf{GRU}$ represents the operation of standard GRU cell. $\mathbf{X}_t$ and $\mathbf{m}_t$ are the observations and masks at time point $t = t_k$. $\mathbf{h}(t_-)$ and $\mathbf{h}(t_+)$ denote the memory before and after the update. Note that some variables in $\mathbf{X}_t$ are missing and the preprocessing function $f_{prep}$ is designed to solve this problem by imputing missing values with prediction values (e.g., prediction mean/variance/error) (details are shown in Appendix A). However, $f_{prep}$ can only be applied in regression tasks, and is not applicable in classification problem. It is because the loss/error is only computed at the final time point and no prediction values will be obtained to impute before the final time point.

## 4 Architecture of CoGRUODE

To alleviate the challenges mentioned above, we propose the model CoGRUODE. It consists of two blocks: marginal block and dependence block. An illustration of CoGRUODE is presented in Figure 3 (A more detailed architecture of CoGRUODE is shown in Appendix C). The marginal block is designed to capture the marginal dynamics of each variable. Instead of sharing one memory, each variable has its own memory in marginal block. Therefore, the information of the univariate will be fully preserved and missing values in some variables will not affect other variables. The dependence block is designed to model the dependence structure among variables by extracting the information from the marginal block. The memory in dependence block is shared across all variables to capture the interaction among variables.

**Marginal block**. We use the Neural ODE to evolve marginal memory when there are no observations for all variables (such as time interval $t_0 - t_1, \cdots, t_4 - t_5$ in Figure 3). However, we only update the marginal memory of observed variables while keep the marginal memory of other unobserved variables unchanged. For example, at time point $t_1$ in Figure 3, the marginal memories of $X_t^2$ and $X_t^3$ are updated but the the marginal memory of $X_t^1$ is not updated and it still be evolved with Neural ODE. One intuitive idea to learn one unique memory for each variable is to train each variable with different GRUODEs, but the training time will increase as the number of variables increases. It is thus a challenge when applying to high-dimensional time series.

To tackle the training time problem, we propose two approaches, namely *hidden matrix (HM) approach* and *hidden vector (HV) approach*, in constructing the marginal blocks. Specifically, the marginal block consists of two parts, namely the *ODE part* (**GRU-ODE-M**) and the *Bayes part* (**GRU-Bayes-M**). Mathematically, the two parts are given as follows:

*ODE part* (**GRU-ODE-M**):

$$\begin{aligned}
\mathbf{r}^M(t) &= \sigma\left(W_r^M \circledast \mathbf{X}(t) + U_r^M \circledast \mathbf{h}^M(t) + b_r^M\right), \\
\mathbf{z}^M(t) &= \sigma\left(W_z^M \circledast \mathbf{X}(t) + U_z^M \circledast \mathbf{h}^M(t) + b_z^M\right), \\
\tilde{\mathbf{h}}^M(t) &= \tanh\left(W_h^M \circledast \mathbf{X}(t) + U_h^M \circledast (\mathbf{r}^M(t) \odot \mathbf{h}^M(t)) + b_h^M\right), \\
\frac{d\mathbf{h}^M(t)}{dt} &= (1 - \mathbf{z}^M(t)) \odot (\tilde{\mathbf{h}}^M(t) - \mathbf{h}^M(t)).
\end{aligned} \tag{4}$$

$$\tag{5}$$

*Bayes part* (**GRU-Bayes-M**):

$$\mathbf{h}^{M,d}(t_+) = \begin{cases} \mathbf{GRU}\left(\mathbf{h}^{M,d}(t_-), X_t^d\right), & \text{if } m_t^d = 1 \\ \mathbf{h}^{M,d}(t_-), & \text{if } m_t^d = 0 \end{cases} \tag{6}$$

*HM approach*: For each variable $d$ where $1 \leq d \leq D$, the corresponding marginal memory $\mathbf{h}^{M,d}(t)$, the reset gate $\mathbf{r}^{M,d}(t)$, the update gate $\mathbf{z}^{M,d}(t)$, and the candidate update gate $\tilde{\mathbf{h}}^{M,d}(t)$ are vectors in $\mathbb{R}^s$. The corresponding weight parameters $W_{(r,z,h)}^{M,d} \in \mathbb{R}^{s\times 1}$, $U_{(r,z,h)}^{M,d} \in \mathbb{R}^{s\times s}$ and bias parameters $b_{(r,z,h)}^{M,d} \in \mathbb{R}^s$. Hence, $\mathbf{h}^M(t)$, $\mathbf{r}^M(t)$, $\mathbf{z}^M(t)$, and $\tilde{\mathbf{h}}^M(t)$, are all $D \times s$ *matrices*, e.g., $\mathbf{h}^M(t) = [\mathbf{h}^{M,1}(t), \cdots, \mathbf{h}^{M,D}(t)]^T$. The total weight parameters $W_{(r,z,h)}^M$ $U_{(r,z,h)}^M$ are tensors such that $W_{(r,z,h)}^M = [W_{(r,z,h)}^{M,1}, \cdots, W_{(r,z,h)}^{M,D}]^T \in \mathbb{R}^{D\times s\times 1}$ and $U_{(r,z,h)}^M = [U_{(r,z,h)}^{M,1}, \cdots, U_{(r,z,h)}^{M,D}]^T \in \mathbb{R}^{D\times s\times s}$. Therefore, $W_{(r,z,h)}^M \circledast \mathbf{X}(t) = [W_{(r,z,h)}^{M,1} X^1(t), \cdots, W_{(r,z,h)}^{M,D} X^D(t)]^T$ and $U_{(r,z,h)}^M \circledast \mathbf{h}^M(t) = [U_{(r,z,h)}^{M,1} \mathbf{h}^{M,1}(t), \cdots, U_{(r,z,h)}^{M,D} \mathbf{h}^{M,D}(t)]^T$.

The marginal block built on HM approach learns marginal memory for each variable independently. However, if the missing rate of variables is too high, the serial structure of each variable is completely destroyed, and it is impossible to learn marginal memory based on such sparse observations. In this case, the marginal block will lose its function and HM approach is not suitable. To form an effective marginal block, we thus relax the constraints that variables must be trained entirely independently.

*HV approach*: The marginal memory $\mathbf{h}^M(t)$, the reset gate $\mathbf{r}^M(t)$, the update gate $\mathbf{z}^M(t)$, and the candidate update gate $\tilde{\mathbf{h}}^M(t)$ are all *vectors* in $\mathbb{R}^H$. $W_{(r,z,h)}^M \in \mathbb{R}^{H\times D}$, $U_{(r,z,h)}^M \in \mathbb{R}^{H\times H}$, $b_{(r,z,h)}^M \in \mathbb{R}^H$ are trainable parameters. Here, $\circledast$ represents the usual matrix multiplication. Generally, $W_{(r,z,h)}^M$ and $U_{(r,z,h)}^M$ are initialized using uniform Xavier (Glorot & Bengio, 2010) and Orthogonal (Saxe et al., 2013) initialization respectively. However, we choose to initialize $U_{(r,z,h)}^M$ as the diagonal matrices and $W_{(r,z,h)}^M$ as the block diagonal matrices $[A_{ij}]_{D\times D}$, where

$$A_{ij} = \begin{cases} [a, a, \cdots, a]_{1\times s} & \text{if } i = j \\ [0, 0, \cdots, 0]_{1\times s} & \text{if } i \neq j \end{cases}. \tag{7}$$

The block $A_{ij}$ when $i = j$ reflects the marginal information for variable $i$, while the block $A_{ij}$ when $i \neq j$ reflects the correlation between variables $i$ and $j$. Initializing 0 to $A_{ij}$ when $i \neq j$ means that no interrelation between variables. $a$ and $s$ are hyperparameters. We set $a = 1$ by default in our experiments. Each variable occupies $s$ dimensions memory and thus the overall size of marginal memory $H$ equals $s \cdot D$. More details about the two approaches are given in Appendix B.

Although we initialize the weight matrix to diagonal blocks in the HV approach, entries in the off-diagonal blocks will slowly move away from 0 during the training process. This means that information about one variable will leak to other variables. We experimentally find that entries on the off-diagonal blocks are much smaller than those on the diagonal blocks, indicating the marginal block mainly focuses on capturing the temporal information for each variable while allowing some small correlation (see Appendix G).

In marginal block, when the variable has no observations, the memory for this variable will be evolved using equation 5 and if this variable has an observation, the memory will be updated using equation 6. Moreover, in equation 6, we do not need the $f_{prep}$ function in equation 3 when updating the memories, and thus our method can also apply to classification tasks compared to GRUODE.

**Dependence block**. The dependence block also consists of two parts, namely the *ODE part* (**GRU-ODE-P**) and the *Bayes part* (**GRU-Bayes-P**). The updated formulae are
*ODE part* (**GRU-ODE-P**):

$$\begin{aligned}
\mathbf{r}^P(t) &= \sigma\left(W_r^P \tilde{\mathbf{h}}^M(t) + U_r^P \mathbf{h}^P(t) + b_r^P\right), \\
\mathbf{z}^P(t) &= \sigma\left(W_z^P \tilde{\mathbf{h}}^M(t) + U_z^P \mathbf{h}^P(t) + b_z^P\right), \qquad (8) \\
\tilde{\mathbf{h}}^P(t) &= \tanh\left(W_h^P \tilde{\mathbf{h}}^M(t) + U_h^P \mathbf{h}^P(t) + b_h^P\right), \\
\frac{d\mathbf{h}^P(t)}{dt} &= \left(1 - \mathbf{z}^P(t)\right) \odot \left(\tilde{\mathbf{h}}^P(t) - \mathbf{h}^P(t)\right), \qquad (9)
\end{aligned}$$

*Bayes part* (**GRU-Bayes-P**) :

$$\mathbf{h}^P\left(t_+\right) = \mathbf{GRU}\left(\mathbf{h}^P\left(t_-\right), \tilde{\mathbf{h}}^M\left(t_+\right)\right), \qquad (10)$$

where $W^P_{(r,z,h)} \in \mathbb{R}^{H\times D}$, $U^P_{(r,z,h)} \in \mathbb{R}^{H\times H}$, and $b^P_{(r,z,h)} \in \mathbb{R}^H$ are trainable parameters. $\mathbf{r}^P(t) \in \mathbb{R}^H$, $\mathbf{z}^P(t) \in \mathbb{R}^H$, and $\tilde{\mathbf{h}}^P(t) \in \mathbb{R}^H$ are the reset gate, update gate, and candidate update gate of dependence block. Unlike the marginal memory, the dependence memory $\mathbf{h}^P(t) \in \mathbb{R}^H$ is shared by all variables. It evolves as equation 9 when all variables have no observations, and updates as equation 10 when one variable has an observation.

Note that we use the candidate update gate $\tilde{\mathbf{h}}^M(t)$ obtained from the marginal block instead of $\mathbf{X}(t)$ as the input in the dependence block at each time point. The reason is that there are missing values in $\mathbf{X}(t)$ across variables. However, $\tilde{\mathbf{h}}^M(t)$ is generated by marginal block based on observations $\mathbf{X}(t)$. It contains the most information of observations and it is a continuous-time variable without any missing values. We use $\tilde{\mathbf{h}}^M(t)$ instead of $\mathbf{h}^M(t)$ as an input in the dependence block. It is because the candidate update gate contains more information about $\mathbf{X}_t$ while $\mathbf{h}^M(t)$ is generated by $\tilde{\mathbf{h}}^M(t)$ and $\mathbf{h}^M(t_-)$.

At last, the marginal memory $\mathbf{h}^M(t)$ and dependence memory $\mathbf{h}^P(t)$ capture marginal and dependent dynamics, respectively. We thus concatenate both memory and feed it into a MLP, to make predictions (i.e., $\hat{\mathbf{X}}_t = \text{MLP}(\mathbf{h}^M(t), \mathbf{h}^P(t))$) or classification (i.e., $\hat{C} = \text{MLP}(\mathbf{h}^M(T), \mathbf{h}^P(T))$).

We verify the properties above by further ablation studies. We conclude our model CoGRUODE in Algorithm 1. The explanation of this algorithm and the comparison between the algorithms of GRUODE and CoGRUODE are displayed in Appendix D.

## 5 Experiments

In this section, we conduct three empirical studies using public datasets to examine the superiority of CoGRUODE. We defer the descriptions of all experiment datasets and training details in Appendix F. We compare our model with various baseline models including GRU-$\Delta t$, CT-GRU (Mozer et al., 2017), GRU-D (Che et al., 2018), ODERNN (Rubanova et al., 2019), GRUODE / mGRUODE (Brouwer et al., 2019), and ODELSTM (Lechner & Hasani, 2020). Hyperparameters, including the dimension of memory, learning rate, batch size, dropout, and weight decay, are tuned using a random search for all experiments. To ensure fairness, the number of parameters in all models are adjusted to

**Algorithm 1** Training process of CoGRUODE

**Input:** Observations: $\mathbf{X}_t$; Observed time points: $\mathbf{t} = [t_1, \cdots, t_K]$; Masks: $\mathbf{m}_t$; time length: $T$
**Initialize:** time $= 0$, $\mathbf{h}_0^M$, $\mathbf{h}_0^P$, and all trainable parameters
**Set:** $\mathbf{h}^M = \mathbf{h}_0^M$, $\mathbf{h}^P = \mathbf{h}_0^P$
**for** $k = 1$ **to** $K$ **do**
  $\mathbf{h}^M, \tilde{\mathbf{h}}^M = \textbf{GRU-ODE-M}(\mathbf{h}^M, \text{time}, t_k)$ % marginal memory evolves to $t_k$
  $\mathbf{h}^P = \textbf{GRU-ODE-P}(\mathbf{h}^P, \tilde{\mathbf{h}}^M, \text{time}, t_k)$ % dependence memory evolves to $t_k$
  time $= t_k$
  $\mathbf{h}^M, \tilde{\mathbf{h}}^M = \textbf{GRU-Bayes-M}(\mathbf{h}^M, \mathbf{X}_t, \mathbf{m}_t, \text{time}, t_k)$ % marginal memory updates at $t_k$
  $\mathbf{h}^P = \textbf{GRU-Bayes-P}(\mathbf{h}^P, \tilde{\mathbf{h}}^M, \text{time}, t_k)$ % dependence memory updates at $t_k$
**end for**
$\mathbf{h}^M, \tilde{\mathbf{h}}^M = \textbf{GRU-ODE-M}(\mathbf{h}^M, t_K, T)$% marginal memory evolves to end
$\mathbf{h}^P = \textbf{GRU-ODE-P}(\mathbf{h}^P, \tilde{\mathbf{h}}^M, t_K, T)$ % dependence memory evolves to end
$\hat{\mathbf{X}}_t = \textbf{MLP}(\mathbf{h}^M, \mathbf{h}^P)$ % prediction/classification with marginal and dependence memory
**return** $\mathbf{h}^M, \mathbf{h}^P, \hat{\mathbf{X}}_t$

Table 1: Correlation matrix for six variables in LSST dataset

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1.000 | 0.169 | 0.125 | 0.126 | 0.122 | 0.074 |
| 2 | 0.169 | 1.000 | 0.800 | 0.833 | 0.820 | 0.479 |
| 3 | 0.125 | 0.800 | 1.000 | 0.929 | 0.883 | 0.736 |
| 4 | 0.126 | 0.833 | 0.929 | 1.000 | 0.900 | 0.652 |
| 5 | 0.122 | 0.820 | 0.883 | 0.900 | 1.000 | 0.606 |
| 6 | 0.074 | 0.479 | 0.736 | 0.652 | 0.606 | 1.000 |

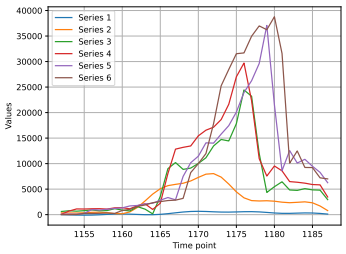


Figure 4: One sample trajectories

be approximately equal. For all ODE-based models, we numerically solve the ODE using "Euler" method with time step $= 0.02$ in LSST dataset, time step $= 0.01$ in Human Activity dataset, and time step $= 0.1$ in Physionet dataset. For all models, a two-layer MLP is used to map the memories at the final time point to the corresponding number of classes in all datasets.

### 5.1 LSST DATASET

The Photometric LSST Astronomical Time Series Classification Challenge is a public dataset that aims to classify simulated astronomical multivariate time-series data (Bagnall et al., 2018). The dataset possesses a strong dependence structure (see Table 1 and Figure 4) and is collected by measuring the photon flux using different astronomical filters. Based on the 6 physical processes, this task aims to classify celestial objects into 14 astronomical classes.

We divide the total samples (4925) into 70% for training, 15% for validation, and 15% for test. Since the dataset is regular and synchronous, we resample three new datasets at different missing rate (25%, 50%, and 75%). The loss function used in this dataset is loss $= \frac{1}{N}\sum_{n=1}^{N} \textbf{CE}\,(y_n, \hat{y}_n)$, where $N$ is the number of time series. $y_n$ denotes the label of $n^{\text{th}}$ time series and $\hat{y}_n$ is the prediction for this label. **CE** denotes the cross-entropy loss.

### 5.2 HUMAN ACTIVITY DATASET

Human Activity dataset (Kaluža et al., 2010) is a challenge to classify the human activities at each time into 7 classes based on the data from sensors attached to people's belts, chests, and ankles. The data from sensors are 6554 sequences of multivariate time series with 12 variables and 211 time points. We follow all data preprocessing steps from (Rubanova et al., 2019) and we normalize the time length and all time series values into 0 to 1. We split all data (6,554) into 70% (4,587) for training, 15% (983) for validation, and 15% (984) for test. As the activities at each time need to be

Table 2: Classification Test Accuracy (mean ± std, $N$ = 5) on LSST Datasets.

| | 25% missingness | 50% missingness | 75% missingness | # params (M) |
|---|---|---|---|---|
| GRU-$\Delta$t | 61.568% ± 0.406% | 56.811% ± 0.939% | 52.162% ± 1.670% | 0.290 |
| CT-GRU | 57.730% ± 1.561% | 53.838% ± 0.804% | 49.108% ± 0.899% | 0.284 |
| GRU-D | 59.514% ± 1.477% | 53.324% ± 1.261% | 49.757% ± 1.038% | 0.280 |
| ODERNN | 50.649% ± 0.563% | 43.270% ± 1.828% | 40.216% ± 0.857% | 0.285 |
| GRUODE | 59.973% ± 1.102% | 57.054% ± 1.257% | 50.892% ± 1.331% | 0.283 |
| mGRUODE | 60.000% ± 1.510% | 56.622% ± 1.809% | 50.676% ± 0.684% | 0.288 |
| ODELSTM | 59.649% ± 1.715% | 55.757% ± 0.980% | 50.027% ± 1.537% | 0.283 |
| CoGRUODE-HV | 62.081% ± 0.665% | 57.270% ± 1.219% | 52.351% ± 0.781% | 0.291 |
| CoGRUODE-HM | **62.514% ± 1.563%** | **58.000% ± 1.031%** | **53.973% ± 0.919%** | 0.281 |

Table 3: Per-time Classification Test Accuracy (mean ± std, $N$ = 5) on Human Activity Dataset.

| | ACCURACY | # params(M) |
|---|---|---|
| GRU-$\Delta t$ | 87.126% ± 0.364% | 0.978 |
| CT-GRU | 82.288% ± 0.218% | 0.956 |
| GRU-D | 85.514% ± 0.858% | 0.992 |
| ODERNN | 54.244% ± 11.443% | 0.964 |
| GRUODE | 87.261% ± 0.281% | 1.012 |
| mGRUODE | 87.286% ± 0.387% | 0.986 |
| ODELSTM | 86.584% ± 0.366% | 1.016 |
| CoGRUODE-HV | **89.006% ± 0.205%** | 0.955 |
| CoGRUODE-HM | 88.346% ± 0.367% | 0.989 |

Table 4: Per-sequence Classification Test AUC-ROC and AUC-PR (mean ± std, $N$ = 5) on Physionet Dataset.

| | AUC-ROC | AUC-PR | # params(M) |
|---|---|---|---|
| GRU-$\Delta t$ | 0.823 ± 0.000 | 0.474 ± 0.001 | 1.065 |
| CT-GRU | 0.780 ± 0.004 | 0.434 ± 0.004 | 1.176 |
| GRU-D | 0.853 ± 0.005 | 0.539 ± 0.014 | 0.959 |
| ODERNN | 0.688 ± 0.031 | 0.273 ± 0.041 | 1.030 |
| GRUODE | 0.821 ± 0.001 | 0.480 ± 0.002 | 0.984 |
| mGRUODE | 0.826 ± 0.001 | 0.484 ± 0.001 | 1.051 |
| ODELSTM | 0.781 ± 0.021 | 0.420 ± 0.046 | 0.995 |
| CoGRUODE-HV | **0.913 ± 0.006** | **0.692 ± 0.016** | 0.962 |
| CoGRUODE-HM | 0.831 ± 0.001 | 0.485 ± 0.003 | 1.023 |

classified, the loss function is $\text{loss} = \frac{1}{N}\frac{1}{K}\sum_{n=1}^{N}\sum_{k=1}^{K}\mathbf{CE}\left(y_{n,k}, \hat{y}_{n,k}\right)$, where $N$ is the number of time series, $K$ is the total number of observational time points. $y_{n,k}$ denotes the label of $n$-th time series at time point $t$ and $\hat{y}_{n,k}$ is the prediction class for this label.

### 5.3 PHYSIONET DATASET

The Physionet 2012 challenge dataset (Goldberger et al., 2000) contains 12,000 multivariate irregularly sampled time series, and each time series consists of several measurements lasting for 48 hours of particular patient admission to ICU. The goal of this dataset is to predict the patients' in-hospital mortality based on the measurements data, which is a binary classification problem. In total, 41 variables are included in this dataset, and we select 37 variables by excluding age, gender, height, and ICUTtype, which are four time-invariant variables. Following the data preprocessing from Rubanova et al. (2019), we round up the timestamps to one minute. Furthermore, we use the data saved in files (a) and (b) to train (7,200 samples) and validate (800 samples) the models, and use data saved in file (c) to test (4,000 samples) the models (Horn et al., 2020). We normalize each variable across all patients to the interval [0, 1]. The loss function used in this dataset is $\text{loss} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{CE}\left(y_n, \hat{y}_n\right)$, where $N$ is the number of time series. $y_n$ denotes the label of $n$-th time series, and $\hat{y}_n$ is the prediction for this label. The datasets are highly imbalanced, with a positive class rate (death class) of around 14%. Thus, we provide AUC-ROC and AUC-PR as the model metric instead of accuracy.

### 5.4 ABLATION STUDIES

1. We perform four ablation studies on LSST dataset with missing rate 50% to validate the efficiency of each component of our model. **(a)** (Original CoGRUODE) using both memories for classification (i.e., $c = \text{MLP}(h^M(T), h^P(T))$). **(b)** Using only marginal memory for classification (i.e., $c = \text{MLP}(h^M(T))$) while keeping the rest of the model unchanged. **(c)** Using only dependence memory for classification (i.e., $c = \text{MLP}(h^P(T))$) while keeping the rest of the model unchanged. **(d)** Using $h^M(t)$ to carry marginal information to dependence block instead of using $\tilde{h}^M(t)$ in equation 8. **(e)** Inputting data $X_t$ into dependence block directly instead of using $\tilde{h}^M(t)$. The results in Figure 5 show that **(1)** using only marginal memory does not perform well since it loses the dependence information across variables; **(2)** if only uses marginal memory, the HV approach gives better results than the HM approach since it allows a little correlation among variables when constructing marginal memory; **(3)**

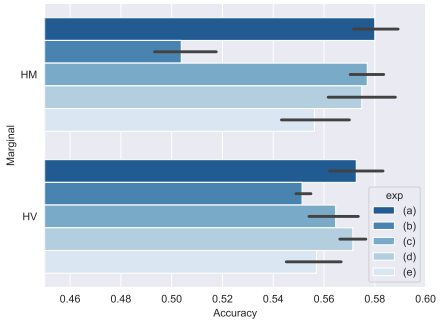


Figure 5: The results of ablation studies of CoGRUODE on LSST dataset with missing rate 50%. (a) is the original CoGRUODE, (b)-(e) are ablation results

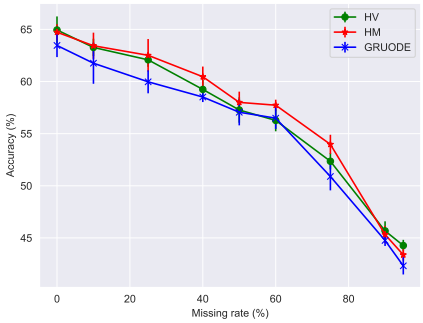


Figure 6: The results of experiments on LSST dataset with various missing rate (0, 10%, 25%, 40%, 50%, 60%, 75%, 90%, 95%).

using only dependence memory performs slightly worse than using two memories since the unique marginal memory of each variable contains the heterogeneous serial dependence, which is hard to be preserved in dependence memory; **(4)** using $\tilde{h}^M(t)$ to carry marginal information to dependence block is slightly better than using $h^M(t)$; **(5)** inputting observed data $X_t$ into dependence block gives poor accuracy since the missing values across variabes introduce the bias into memory, indicating the marginal block mitigates the problem of missing values.

2. We further conduct more experiments on LSST dataset with missing rate ranging from 0 to 95% (see results in Figure 6). We notice that **(1)** CoGRUODE always outperforms than GRUODE, showing the superioty of marginal-dependence framework; **(2)** the HM approach generally gives better results than HV approach when missing rate is low. However, when missing rate is high (e.g., greater that 90%), the serial dependece is broken and thus the marginal block built on HM approach is less efficient. On the contrary, marginal block built on HV approach proforms well since it allows a little correlation to construct marginal memory. This result is also proved in the Physionet experiments where CoGRUODE-HM performs worse than CoGRUODE-HV. The reason is that 23 variables of the 36 variables have a missing rate of more than 90% (the details of missing rate for each variable are listed in Appendix F).

## 6 Conclusion

This paper presents a new model architecture called CoGRUODE for multivariate asynchronous time series prediction/classification. It is a continuous-time model built on the GRUODE and contains two blocks: marginal block and dependence block. The marginal block and the dependence block capture the serial dependence of each variable and the dependence structure across all variables respectively. Two approaches (HM / HV) are proposed to build the marginal block. HM approach forms independent marginal memory of each variable. HV approach allows a little correlation between variables in building the marginal memory of each variable. We find that the HM approach give better results than the HV approach when the missing rate is low. However, when missing rate is high, the serial dependence is destroyed. The HM approach is less efficient in this case but HV approach also performs well. The dependence block aims to model the dependence structure among variables. It retrieves the marginal memory of each variable in the marginal block and forms the dependence memory. At last, we make prediction/classification using the memories from both blocks. Our model outperforms various baselines based on three public empirical datasets.

## REFERENCES


Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

Francois Belletti, Alex Beutel, Sagar Jain, and Ed Chi. Factorized recurrent neural architectures for longer range dependence. In *international conference on artificial intelligence and statistics*, pp. 1522–1530. PMLR, 2018.

Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. GRU-ODE-Bayes: continuous modeling of sporadically-observed time series. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 7379–7390, 2019.

Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 6572–6583, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.

Hrayr Harutyunyan, Hrant Khachatrian, David C Kale, Greg Ver Steeg, and Aram Galstyan. Multitask learning and benchmarking with clinical time series data. *Scientific data*, 6(1):1–18, 2019.

Max Horn, Michael Moor, Christian Bock, Bastian Rieck, and Karsten Borgwardt. Set functions for time series. In *International Conference on Machine Learning*, pp. 4353–4363. PMLR, 2020.

Rob Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media, 2008.

Anders Boeck Jensen, Pope L Moseley, Tudor I Oprea, Sabrina Gade Ellesøe, Robert Eriksson, Henriette Schmock, Peter Bjødstrup Jensen, Lars Juhl Jensen, and Søren Brunak. Temporal disease trajectories condensed from population-wide registry data covering 6.2 million patients. *Nature communications*, 5(1):1–10, 2014.

Boštjan Kaluža, Violeta Mirchevska, Erik Dovgan, Mitja Luštrek, and Matjaž Gams. An agent-based approach to care in independent living. In *International joint conference on ambient intelligence*, pp. 177–186. Springer, 2010.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Mathias Lechner and Ramin Hasani. Learning long-term dependencies in irregularly-sampled time series. *arXiv preprint arXiv:2006.04418*, 2020.

Helmut Lütkepohl. Forecasting with varma models. *Handbook of economic forecasting*, 1:287–325, 2006.

Spyros Makridakis and Michele Hibon. Arma models and the box–jenkins methodology. *Journal of forecasting*, 16(3):147–163, 1997.

Michael C Mozer, Denis Kazakov, and Robert V Lindsey. Discrete event, continuous time rnns. *arXiv preprint arXiv:1710.04110*, 2017.

Yulia Rubanova, Ricky TQ Chen, and David Duvenaud. Latent ODEs for irregularly-sampled time series. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 5320–5330, 2019.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Jeffrey D Scargle. Studies in astronomical time series analysis. ii-statistical aspects of spectral analysis of unevenly spaced data. *The Astrophysical Journal*, 263:835–853, 1982.

Ikaro Silva, George Moody, Daniel J Scott, Leo A Celi, and Roger G Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In *2012 Computing in Cardiology*, pp. 245–248. IEEE, 2012.

Pravin K Trivedi and David M Zimmer. *Copula modeling: an introduction for practitioners*. Now Publishers Inc, 2007.

Yuyang Wang, Alex Smola, Danielle Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. Deep factors for forecasting. In *International conference on machine learning*, pp. 6607–6617. PMLR, 2019.

Kun Zhang, Yuan Xue, Gerardo Flores, Alvin Rajkomar, Claire Cui, and Andrew M Dai. Modelling ehr timeseries by restricting feature interaction. *arXiv preprint arXiv:1911.06410*, 2019.

Yaquan Zhang, Qi Wu, Nanbo Peng, Min Dai, Jing Zhang, Hu Wang, et al. Memory-gated recurrent networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)*, 2020.

Zhao-Yu Zhang, Shao-Qun Zhang, Yuan Jiang, and Zhi-Hua Zhou. Life: Learning individual features for multivariate time series prediction with missing values. In *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 1511–1516. IEEE, 2021.

Guo-Bing Zhou, Jianxin Wu, Chen-Lin Zhang, and Zhi-Hua Zhou. Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, 13(3):226–234, 2016.