FULLY CONTINUOUS GATED RECURRENT UNITS FOR PROCESSING TIME SERIES

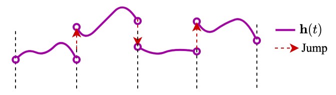
Anonymous authors
Paper under double-blind review

ABSTRACT

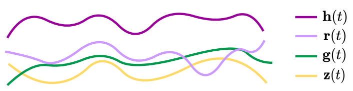
For a long time, RNN-based models, such as RNNs, LSTMs, and GRUs, have been used to process time series data. However, RNN-based models do not fit well with real-world sporadically observed data. As a result, many researchers have suggested various enhancements to overcome the limitation. Among them, differential equation-based models, e.g., GRU-ODE-Bayes, ODE-RNN, and so forth, show good accuracy in many cases. Those methods try to continuously model the hidden state of RNNs (or GRUs). However, existing methods' hidden states are *piece-wise continuous*. In this paper, we represent GRUs as delay differential equations and present fully continuous GRUs. To our knowledge, we propose the first model that continuously generalizes all the parts of GRUs, including their hidden state and various gates. After reconstructing a continuous path $\mathbf{x}(t)$ from discrete time series observations $\{(\mathbf{x}_i, t_i)\}_{i=0}^{N-1}$ (with an appropriate interpolation algorithm), we calculate the time derivatives of the reset gate $\mathbf{r}(t)$, the update gate $\mathbf{z}(t)$, the update vector $\mathbf{g}(t)$, and the hidden state $\mathbf{h}(t)$. Then, we develop an augmented delay differential equation (DDE) that continuously generalizes all the parts. In our experiments with 3 real-world datasets and 13 baselines, our fully continuous GRU method outperforms existing baselines by non-trivial margins.

1 INTRODUCTION

Recurrent neural networks (RNNs) are one of the most popular neural networks in deep learning. They have been actively used for processing sequential data (Yin et al., 2017; Fu, 2011). Several different recurrent units have been proposed, ranging from long short-term memory (LSTMs (Hochreiter & Schmidhuber, 1997)) to gated recurrent units (GRUs (Cho et al., 2014)). Among them, GRUs are one of the most popular units and have advantages in terms of processing efficiency and efficacy (Brouwer et al., 2019).



(a) Existing piece-wise continuous approach (e.g., GRU-ODE-Bayes ODE-RNN, NJODE, etc.), where the jump mechanism is mandatory to read new time series input.



(b) Our proposed fully continuous approach, called Cont-GRU

Figure 1: Existing vs. Cont-GRU

Among various enhancements for GRUs, GRU-ODE-Bayes, ODE-RNN, NJODE, and some others (Herrera et al., 2021; Lukoševičius & Uselis, 2022; Schirmer et al., 2021; Brouwer et al., 2019) make a fundamental and unique contribution in continuously generalizing GRUs and therefore, they have a strong point in processing irregular time series. However, we found that they (piece-wise) continuously generalize only the hidden state $\mathbf{h}(t)$ of GRUs, but the reset gate $\mathbf{r}(t)$, the update gate $\mathbf{z}(t)$, and the update vector $\mathbf{g}(t)$ of GRUs are not properly continuously generalized. For instance, the continuous generalization of GRUs by GRU-ODE-Bayes can be written as follows:

$$\begin{aligned}\frac{d\mathbf{h}(t)}{dt} &:= \big(1-\mathbf{z}(t)\big)\odot\big(\mathbf{g}(t)-\mathbf{h}(t)\big),\\ \mathbf{z}(t) &:= \sigma\big(\mathbf{W}_z\mathbf{x}(t)+\mathbf{U}_z\mathbf{h}(t-1)+\mathbf{b}_z\big),\\ \mathbf{g}(t) &:= \phi\big(\mathbf{W}_g\mathbf{x}(t)+\mathbf{U}_g\big(\mathbf{r}(t)\odot\mathbf{h}(t-1)\big)+\mathbf{b}_g\big),\\ \mathbf{r}(t) &:= \sigma\big(\mathbf{W}_r\mathbf{x}(t)+\mathbf{U}_r\mathbf{h}(t-1)+\mathbf{b}_r\big),\end{aligned} \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{\dim(\mathbf{h}) \times \dim(\mathbf{x})}$ and $\mathbf{U} \in \mathbb{R}^{\dim(\mathbf{h}) \times \dim(\mathbf{h})}$ are weight matrices, and $\mathbf{b} \in \mathbb{R}^{\dim(\mathbf{h})}$ is a bias vector. σ is a sigmoid function and ϕ is a hyperbolic tangent function. $\odot$ means an element-wise product. We note that only the hidden state is modeled as an ODE.

As noted above, GRU-ODE-Bayes does not calculate the time derivatives of the reset gate, the update gate, and the update vector, denoted $\frac{d\mathbf{r}(t)}{dt}$, $\frac{d\mathbf{z}(t)}{dt}$, and $\frac{d\mathbf{g}(t)}{dt}$ respectively, since the input time series sample $\{(\mathbf{x}_i, t_i)\}_{i=0}^{N-1}$ is discrete and it is impossible to calculate $\frac{d\mathbf{x}(t)}{dt}$. When estimating $\frac{d\mathbf{h}(t)}{dt}$ for arbitrary time t, one can circumvent the situation by reusing the most recently observed input $\mathbf{x}_i$, where t_i is the closest observation time to t. In addition, GRU-ODE-Bayes uses the jump mechanism to read the discrete observations in $\{(\mathbf{x}_i, t_i)\}_{i=0}^{N-1}$ (cf. Figure 1 (a)).

In all those continuous methods, only the hidden state is (piece-wise) continuously generalized. In addition, such ODE-based continuous generalizations inherit the limitation of ODEs, i.e., all ODEs are homeomorphic and sometimes lack in their representation learning capabilities (Dupont et al., 2019). In this paper, however, we completely continuously generalize GRUs based on delay differential equations (DDEs), including the reset gate, the update gate, and the update vector (cf. Figure 1 (b)) — our method neither use the jump mechanism and nor have the ODE's limitations. To our knowledge, our model, called ***Continuous GRU*** (Cont-GRU), is the first fully continuous interpretation of GRUs. Our method can be summarized as follows:

1. We calculate the time derivatives of the hidden state $\mathbf{h}(t)$, the reset gate $\mathbf{r}(t)$, the update gate $\mathbf{z}(t)$, and the update vector $\mathbf{g}(t)$ of GRUs.
2. We then define an augmented delay differential equation (DDE) in Equation 10 after combining all those time derivative terms. The advantages of our model are as follows:
   (a) DDEs share the same base philosophy as that of GRUs, which is past information influences current output. DDEs are for modeling these time-delay systems.
   (b) Using an interpolation algorithm, we convert the discrete time series sample $\{(\mathbf{x}_i, t_i)\}_{i=0}^{N-1}$ into a continuous path $\mathbf{x}(t)$, where $\mathbf{x}(t_i) = (\mathbf{x}_i, t_i)$ at each observation time point t_i and for other non-observed time points, the interpolation algorithm fills out appropriate values.
   (c) There is no need to use the jump mechanism since $\mathbf{x}(t)$ is continuous and therefore, our method is *fully* continuous rather than being *piece-wise* continuous.
   (d) Our DDE-based method does not have the limitation of ODEs — DDEs are not homeomorphic — and therefore, we expect better representation learning capability which is important for downstream tasks. In addition, our Cont-GRU model is relatively lightweight in terms of computation time. See Appendix I for more discussion.

2 Related work

RNN-based models Machine learning with time series has a long history. RNNs, LSTMs, and GRUs have been typically used for processing time series data. However, many real-world time series applications suffer from the irregularity of time series.

As a result, typical RNN-based models show good performance for regular time series datasets only. In order to overcome the limitation, their recent enhancements have attempted to implement simple heuristic methods to handle irregular time series. GRU-D (Che et al., 2018) is one of such models that can handle missing observations of time series.

Differential Eqauation-based models Deep learning models based on differential equations are commonly utilized for processing irregular time series. Many of them rely on a technology called neural ordinary differential equations (NODEs (Chen et al., 2018)), which solve the following initial value problem:

$$\mathbf{h}(t_i) = \mathbf{h}(t_{i-1}) + \int_{t_{i-1}}^{t_i} f(t, \mathbf{h}(t), \boldsymbol{\theta}_f)dt, \tag{2}$$

where f, called *ODE function*, is a neural network which is parameterized by $\boldsymbol{\theta}_f$ and approximates $\frac{d\mathbf{h}(t)}{dt}$. We can get $\mathbf{h}(t_i)$ by solving the initial value problem with various ODE solvers. However, NODEs are homeomorphic. In other words, the mapping from $\mathbf{h}(t_{i-1})$ to $\mathbf{h}(t_i)$ continuously changes in a bijective manner, which is too restrictive in some cases for complicated tasks and therefore, augmenting $\mathbf{h}(t)$ with zeros had been proposed in (Dupont et al., 2019). We also consider this augmentation technique to enhance some baselines for thorough experiments. However, this increases the model size and computation amount.

NODEs overcome the limitation posed by discrete time RNNs. In other words, utilizing Equation 2 between two observations allows us to model the evolutionary process of the hidden state between the two observations in a continuous time. Following this idea, various differential equation-based time series models have been recently proposed. Most of them use a mechanism called *jump*. A jump means that we read a new observation after solving the previous initial value problem and before solving the next initial value problem (cf. Figure 1 (a)).

ODE-RNN (Rubanova et al., 2019) is a hybrid NODE model that combines RNNs and NODEs with the jump mechanism. Unlike ordinary RNNs, ODE-RNN can calculate the hidden state at any time by controlling the terminal integral time. There are various efforts for enhancing ODE-RNN, such as Latent-ODE, Augmented-ODE, ACE-NODE and GRU-ODE-Bayes. These models are the ones that piece-wise continuously generalize the hidden state of RNNs. Among them, GRU-ODE-Bayes (Brouwer et al., 2019) is a representative continuous method with the jump mechanism. Compared to others, it piece-wise continuously generalizes GRUs. GRU-ODE-Bayes consists of two parts, GRU-ODE and GRU-Bayes.

Neural jump ODEs (NJODEs) (Herrera et al., 2021) are data-driven and continuous in time. In NJODEs, the ODE function f transforms the hidden state between observations and jumps the hidden state according to the jump neural network when a new observation is available.

Table 1: Comparison among continuous RNN models

Model	Is it continuous?	What are continuous?
ODE-RNN	Piece-wise	$\mathbf{h}(t)$
GRU-ODE-Bayes	Piece-wise	$\mathbf{h}(t)$
NJODE	Piece-wise	$\mathbf{h}(t)$
NCDE	Fully	$\mathbf{h}(t)$
Cont-GRU	Fully	$\mathbf{h}(t), \mathbf{r}(t), \mathbf{z}(t), \mathbf{g}(t)$

Various methods have been proposed to address the issue. In (Kidger et al., 2020), neural controlled differential equations (NCDEs) were proposed to resolve the issue by using the controlled differential equation paradigm. To continuously generalize the hidden state, NCDEs read a continuous path and evolve the hidden state $\mathbf{h}(t)$ continuously over time. There also exist several variants of NCDES, such as ANCDE and EXIT, which focus on how to account for hidden states. In the case of ANCDE, they added an attention to NCDEs to selectively drop (or select) observations. EXIT proposed an effective way to construct a continuous path from a discrete time series input. However, those works are also interested in continuously generalizing the hidden state only. In Table 1, we compare existing popular continuous RNN models. All previous methods (piece-wise) continuously generalize the hidden state only. Our Cont-GRU is the first model that fully continuously generalizes all the parts of GRUs.

Delay Differential Equations Delay differential equations (DDEs) are a type of differential equation in mathematics that uses the value of a function from a previous time to determine the derivative of the function at a given time. DDEs are also known as time-delay systems. They are a functional state system, similar to partial differential equations (PDEs) with infinite-dimensional state vectors, as opposed to ordinary differential equations (ODEs) with finite-dimensional state vectors. DDEs overcome the limitations of NODEs. In particular, it overcomes the limitations of NODEs well in physical or physiological systems where the effect of time delay cannot be avoided. In this context, two interesting papers (Zhu et al., 2021; 2022) have been published. In them, the simplest form of neural delay differential equations (NDDEs) can be written as follows:

$$\mathbf{h}(t_i) = \mathbf{h}(t_{i-1}) + \int_{t_{i-1}}^{t_i} f(t, \mathbf{h}(t), \mathbf{h}(t-\tau), \boldsymbol{\theta}_f)dt, \tag{3}$$

where τ is a delay effect. The difference between Equation 2 and Equation 3 is that NDDEs consider the function evaluations of previous times.

3 Proposed method

In this section, we describe our proposed fully continuous GRU concept. Unlike the previous methods that only make the hidden state $\mathbf{h}(t)$ continuous, we continuously generalize all the hidden state. the update gate, the update vector, and the reset gate — to our knowledge, we are the first fully continuously generalizing GRUs. We first redefine GRUs to construct their fully continuous counterparts as follows — we simply replace $t-1$ in GRUs with $t-\Delta t$:

$$\begin{aligned}
\mathbf{h}(t) &:= \mathbf{z}(t) \odot \mathbf{h}(t-\Delta t) + (1-\mathbf{z}(t)) \odot \mathbf{g}(t), \\
\mathbf{z}(t) &:= \sigma\big(\mathbf{W}_z\mathbf{x}(t) + \mathbf{U}_z\mathbf{h}(t-\Delta t) + \mathbf{b}_z\big) = \sigma(\mathbf{A}(t, t-\Delta t)) \\
\mathbf{g}(t) &:= \phi\big(\mathbf{W}_g\mathbf{x}(t) + \mathbf{U}_g\big(\mathbf{r}(t) \odot \mathbf{h}(t-\Delta t)\big) + \mathbf{b}_g\big) = \phi(\mathbf{B}(t, t-\Delta t)), \\
\mathbf{r}(t) &:= \sigma\big(\mathbf{W}_r\mathbf{x}(t) + \mathbf{U}_r\mathbf{h}(t-\Delta t) + \mathbf{b}_r\big) = \sigma(\mathbf{C}(t, t-\Delta t)).
\end{aligned} \tag{4}$$

We note that we adopt $\mathbf{A}(t, t-\Delta t)$, $\mathbf{B}(t, t-\Delta t)$, and $\mathbf{C}(t, t-\Delta t)$ to simplify our definition, which is helpful later in defining the time derivative terms of all GRU parts.

3.1 Overall workflow

Figure 2 (b) shows the overall workflow diagram of our method, Cont-GRU, which is defined as follows:

1. A continuous path $\mathbf{x}(t)$ is created from a discrete time series sample by an interpolation algorithm — one can choose any interpolation method, e.g., natural cubic spline.
2. All the reset gate $\mathbf{r}(t)$, the update gate $\mathbf{z}(t)$, the update vector $\mathbf{g}(t)$, and the hidden state $\mathbf{h}(t)$ of GRUs are modeled as an augmented DDE, which means that they are all continuous in our framework.
3. After that, there is one more fully connected layer to further process $\mathbf{h}(t)$ for a downstream task, i.e., output layer.

3.2 Fully continuous GRUs

In order to continuously generalize GRUs, we need to calculate the time derivatives of GRU's various parts. Considering Equation 4, we can define them as an augmented DDE.

Time derivative of $\mathbf{h}(t)$ Since the hidden state $\mathbf{h}(t)$ is a composite function of $\mathbf{r}(t)$, $\mathbf{z}(t)$, and $\mathbf{g}(t)$, the derivative of $\mathbf{h}(t)$ can be written as follows:

$$\begin{aligned}
\frac{d\mathbf{h}(t)}{dt} &= \frac{d\mathbf{z}(t)}{dt} \odot \mathbf{h}(t-\Delta t) + \mathbf{z}(t) \odot \frac{d\mathbf{h}(t-\Delta t)}{dt} - \frac{d\mathbf{z}(t)}{dt} \odot \mathbf{g}(t) + (1-\mathbf{z}(t)) \odot \frac{d\mathbf{g}(t)}{dt}, \\
&= \frac{d\mathbf{z}(t)}{dt} \odot \big(\mathbf{h}(t-\Delta t) - \mathbf{g}(t)\big) + \mathbf{z}(t) \odot \big(\frac{d\mathbf{h}(t-\Delta t)}{dt} - \frac{d\mathbf{g}(t)}{dt}\big) + \frac{d\mathbf{g}(t)}{dt}, \\
&= \frac{d\mathbf{z}(t)}{dt} \odot \zeta(t, t-\Delta t) + \mathbf{z}(t) \odot \frac{d\zeta(t, t-\Delta t)}{dt} + \frac{d\mathbf{g}(t)}{dt},
\end{aligned} \tag{5}$$

where $\zeta(t, t-\Delta t) = \mathbf{h}(t-\Delta t) - \mathbf{g}(t)$. So, we can write $\frac{d\mathbf{h}(t)}{dt}$ as follows:

$$\frac{d\mathbf{h}(t)}{dt} = \frac{d(\mathbf{z}(t) \odot \zeta(t, t-\Delta t))}{dt} + \frac{d\mathbf{g}(t)}{dt} \tag{6}$$

Time derivative of $\mathbf{z}(t)$ The *continuous* update gate is written as $\mathbf{z}(t) = \sigma\big(\mathbf{W}_z\mathbf{x}(t) + \mathbf{U}_z\mathbf{h}(t-\Delta t) + \mathbf{b}_z\big)$, and its derivative, denoted $\frac{d\mathbf{z}(t)}{dt}$, is as follows:

$$\frac{d\mathbf{z}(t)}{dt} = \sigma\big(\mathbf{A}(t, t-\Delta t))(1 - \sigma(\mathbf{A}(t, t-\Delta t))\big)\frac{d\mathbf{A}(t, t-\Delta t)}{dt}, \tag{7}$$

where $\mathbf{A}(t, t-\Delta t) = \mathbf{W}_z\mathbf{x}(t) + \mathbf{U}_z\mathbf{h}(t-\Delta t) + \mathbf{b}_z$, and $\frac{d\mathbf{A}(t,t-\Delta t)}{dt} = \mathbf{W}_z\frac{d\mathbf{x}(t)}{dt} + \mathbf{U}_z\frac{d\mathbf{h}(t-\Delta t)}{dt}$.

Time derivative of $\mathbf{g}(t)$ The *continuous* update vector has the form of $\mathbf{g}(t) = \phi\big(\mathbf{W}_g\mathbf{x}(t) + \mathbf{U}_g\big(\mathbf{r}(t) \odot \mathbf{h}(t-\Delta t)\big) + \mathbf{b}_g\big)$, and its derivative, $\frac{d\mathbf{g}(t)}{dt}$, can be calculate as follows:

$$\frac{d\mathbf{g}(t)}{dt} = \big(1 - \phi(\mathbf{B}(t, t-\Delta t))\big)\big(1 + \phi(\mathbf{B}(t, t-\Delta t))\big)\frac{d\mathbf{B}(t, t-\Delta t)}{dt}, \tag{8}$$

where $\mathbf{B}(t, t-\Delta t) = \mathbf{W}_g\mathbf{x}(t) + \mathbf{U}_g\big(\mathbf{r}(t) \odot \mathbf{h}(t-\Delta t)\big) + \mathbf{b}_g$, and $\frac{d\mathbf{B}(t,t-\Delta t)}{dt} = \mathbf{W}_g\frac{d\mathbf{x}(t)}{dt} + \mathbf{U}_g\frac{d\mathbf{r}(t)}{dt}\mathbf{h}(t-\Delta t) + \mathbf{U}_g\mathbf{r}(t)\frac{d\mathbf{h}(t-\Delta t)}{dt}$.

Algorithm 1: How to train Cont-GRU

```
Input: Training data $D_{train}$, Validating data $D_{val}$, Maximum iteration numbers $max\_iter$
1 Initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta}$ means the parameters of Cont-GRU, e.g., $\mathbf{W}_h$, $\mathbf{U}_h$, etc.);
2 Create a continuous path $\mathbf{x}(t)$ for each discrete time series sample $\{(\mathbf{x}_i, t_i)\}_{i=0}^{N-1}$ in $D_{train}$ and $D_{val}$ with
    an interpolation algorithm;
3 $i \leftarrow 0$;
4 while $i < max\_iter$ do
5     Train $\boldsymbol{\theta}$ and using a task loss $L$;
6     Validate and update the best parameters $\boldsymbol{\theta}^*$ with $D_{val}$;
7     $i \leftarrow i + 1$;
8 return $\boldsymbol{\theta}^*$;
```

Time derivative of $\mathbf{r}(t)$ The *continuous* reset gate is defined as $\mathbf{r}(t) = \sigma\big(\mathbf{W}_r\mathbf{x}(t) + \mathbf{U}_r\mathbf{h}(t - \Delta t) + \mathbf{b}_r\big)$, and its derivative $\frac{d\mathbf{r}(t)}{dt}$ is derived as follows:

$$\frac{d\mathbf{r}(t)}{dt} = \sigma\big(\mathbf{C}(t))(1 - \sigma(\mathbf{C}(t, t - \Delta t))\big)\frac{d\mathbf{C}(t, t - \Delta t)}{dt}, \tag{9}$$

where $\mathbf{C}(t, t - \Delta t) = \mathbf{W}_r\mathbf{x}(t) + \mathbf{U}_r\mathbf{h}(t - \Delta t) + \mathbf{b}_r$, and $\frac{d\mathbf{C}(t,t-\Delta t)}{dt} = \mathbf{W}_r\frac{d\mathbf{x}(t)}{dt} + \mathbf{U}_r\frac{d\mathbf{h}(t-\Delta t)}{dt}$.

Finally, the time derivatives of $\mathbf{h}(t), \mathbf{r}(t), \mathbf{z}(t)$, and $\mathbf{g}(t)$ is written as follows :

$$\frac{d}{dt}\begin{bmatrix}\mathbf{h}(t)\\ \mathbf{z}(t)\\ \mathbf{g}(t)\\ \mathbf{r}(t)\end{bmatrix} = \begin{bmatrix}\frac{d(\mathbf{z}(t)\odot\zeta(t,t-\Delta t))}{dt} + \frac{d\mathbf{g}(t)}{dt}\\ \sigma\big(\mathbf{A}(t, t - \Delta t))(1 - \sigma(\mathbf{A}(t, t - \Delta t))\big)\frac{d\mathbf{A}(t,t-\Delta t)}{dt}\\ \big(1 - \phi(\mathbf{B}(t, t - \Delta t))\big)\big(1 + \phi(\mathbf{B}(t, t - \Delta t))\big)\frac{d\mathbf{B}(t,t-\Delta t)}{dt}\\ \sigma\big(\mathbf{C}(t, t - \Delta t))(1 - \sigma(\mathbf{C}(t, t - \Delta t))\big)\frac{d\mathbf{C}(t,t-\Delta t)}{dt}\end{bmatrix}. \tag{10}$$

We note that the above definition becomes a DDE since $\zeta, \mathbf{A}, \mathbf{B}$, and $\mathbf{C}$ have internally $\mathbf{h}(t - \Delta t)$. In the perspective of implementations, $\mathbf{h}(t - \Delta t)$ can be approximated with $\mathbf{h}(t - s)$, where s is a small step size of ODE solvers. $\frac{d\mathbf{x}(t)}{dt}$ contained by the derivatives of $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$ can also be calculated since we use an interpolation method to construct $\mathbf{x}(t)$ (see Section 4.4 and Appendix E).

3.3 TRAINING METHOD

In Alg. 1, we show our training algorithm. Since our Cont-GRU can be used for various tasks, we show a brief pseudo-code of the training method in Alg. 1. For a more concrete example, suppose a time series classification task with $(\{(\mathbf{x}_i, t_i)\}_{i=0}^{N-1}, \mathbf{y})$, where $\mathbf{y}$ is the ground-truth class label of the discrete time series sample. For this, we first solve the following integral problem:

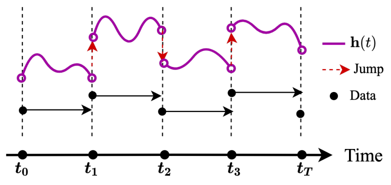
$$\begin{bmatrix}\mathbf{h}(t_{N-1})\\ \mathbf{r}(t_{N-1})\\ \mathbf{z}(t_{N-1})\\ \mathbf{g}(t_{N-1})\end{bmatrix} = \begin{bmatrix}\mathbf{h}(0)\\ \mathbf{r}(0)\\ \mathbf{z}(0)\\ \mathbf{g}(0)\end{bmatrix} + \int_0^{t_{N-1}} \frac{d}{dt}\begin{bmatrix}\mathbf{h}(t)\\ \mathbf{r}(t)\\ \mathbf{z}(t)\\ \mathbf{g}(t)\end{bmatrix}dt, \tag{11}$$

where $\mathbf{h}(0), \mathbf{r}(0), \mathbf{z}(0), \mathbf{g}(0)$ are set in the same way as the original discrete GRU.

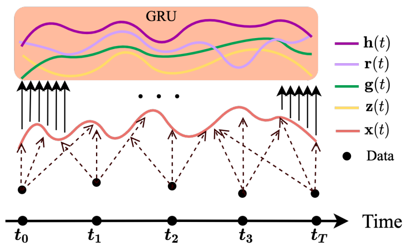
We then feed $\mathbf{h}(t_{N-1})$ into a following output layer with a softmax activation to predict its class label $\hat{\mathbf{y}}$, where the task loss L is a cross-entropy loss between the prediction $\hat{\mathbf{y}}$ and the ground-truth $\mathbf{y}$. During the process, one can easily calculate the gradients using either the standard backpropagation or the adjoint sensitivity method (Chen et al., 2018).

Well-posedness The well-posedness[1] of NODEs was already proved in (Lyons et al., 2004, Theorem 1.3) under the mild condition of the Lipschitz continuity. We show that our fully continuous GRUs are also well-posed. Almost all activations, such as ReLU, Leaky ReLU, SoftPlus, Tanh, Sigmoid, ArcTan, and Softsign, have a Lipschitz constant of 1. Other common neural network layers, such as dropout, batch normalization, and other pooling methods, have explicit Lipschitz constant values. Therefore, the Lipschitz continuity of $\frac{d\mathbf{h}(t)}{dt}$, $\frac{d\mathbf{r}(t)}{dt}$, $\frac{d\mathbf{z}(t)}{dt}$, and $\frac{d\mathbf{g}(t)}{dt}$ can be fulfilled in our case. Accordingly, it is a well-posed problem. Thus, its training process is stable in practice.

[1]A well-posed problem means i) its solution uniquely exists, and ii) its solution continuously changes as input data changes.



(a) GRU-ODE-Bayes



(b) Cont-GRU

Figure 2: Whereas GRU-ODE-Bayes piece-wise continuously generalizes GRUs, our proposed Cont-GRU fully continuously generalizes them.

3.4 Discussion

In Figure 2, we compare our proposed method with GRU-ODE-Bayes. First, our method does not need to use the jump mechanism since i) the time derivative of $\mathbf{x}(t)$ can be properly defined, and ii) our DDE definition keeps reading the time derivative. Second, our continuous generalization makes sense mathematically since we consider the time derivative terms of the reset gate, the update gate, and the update vector in conjunction with the time derivative term of the hidden state. In fact, calculating the time derivative of the hidden state requires the time derivatives of other gates, which were ignored in GRU-ODE-Bayes. In addition, our DDE-based fully continuous GRUs do not have the homeomorphic limitation of ODEs.

Owing to these facts of Cont-GRU, it shows more robust processing for irregular time series. In our experiments, we compare our method with existing methods in diverse environments.

4 Experiments

We describe our experimental environments and results. We evaluate our model on three datasets, considering the following three situations:

1. Forecast weather in various sequence lengths with USHCN.
2. Predict patient conditions with PhysioNet Sepsis.
3. Forecast volatile stock prices and volumes for a period, which includes the COVID-19 period, with Google Stock.

We repeat training and testing procedures with five different random seeds and report their mean and standard deviation scores. We will explain our 13 baselines in Appendix A.

4.1 Forecast weather in various sequence length

Ecosystems and other social systems have long been accustomed to predictable weather characteristics. Unexpected weather conditions, such as global warming or extreme weather, occur frequently from recently. Therefore, predicting these future weather conditions is very important to society (Salman et al., 2015; Grover et al., 2015). Due to the nature of weather data, it is challenging to predict long-distance weather conditions, but it is an important issue for our society.

In this paper, we forecast weather conditions with various sequence lengths. We use the United State Historical Climatology Network (USHCN) daily dataset (Menne & Williams Jr, 2009). USHCN data includes five climatic variables (daily temperatures, precipitation, snow, and so on) for 1,218 meteorological stations across the United States over 150 years. We use a subset of 1,114 meteorological stations over four years from 1996 to 2000 using the cleaning method proposed in (Brouwer et al., 2019). To evaluate various models, each model reads 100 sequences and forecasts the next 10, 20, or 30 sequences.

Experimental results Table 2 shows one of the most extensively used benchmark experiments, i.e., time series forecasting with the USHCN weather dataset. We conduct three sorts of experiments: forecasting the next 10, 20, or 30 sequences — GRU-ODE-Bayes forecasts up to the next 3 sequences

Table 2: USHCN

Model	Test MSE			Memory (MB)	Training Time (s)
	10 sequence	20 sequence	30 sequence		
NODE	0.23 ± 0.02	0.18 ± 0.03	0.25 ± 0.04	55.79	30.6
ODE-RNN	0.21 ± 0.03	0.17 ± 0.04	0.23 ± 0.05	565.1	9.14
GRU-Δt	0.28 ± 0.04	0.25 ± 0.02	0.30 ± 0.03	492.7	1.26
GRU-D	0.30 ± 0.02	0.25 ± 0.02	0.30 ± 0.04	528.9	1.50
GRU-ODE	0.27 ± 0.07	0.23 ± 0.09	0.31 ± 0.07	76.79	8.63
Latent-ODE	0.34 ± 0.00	0.32 ± 0.02	0.38 ± 0.00	210.3	15.2
Augmented-ODE	0.33 ± 0.01	0.33 ± 0.03	0.35 ± 0.02	197.2	46.5
ACE-NODE	0.35 ± 0.09	0.34 ± 0.07	0.38 ± 0.05	210.4	44.7
GRU-ODE-Bayes	0.39 ± 0.08	0.37 ± 0.04	0.42 ± 0.01	1,613	33.7
NJODE	0.37 ± 0.06	0.36 ± 0.01	0.39 ± 0.03	1,889	36.1
NCDE	0.24 ± 0.08	0.20 ± 0.00	0.41 ± 0.09	62.32	251
ANCDE	0.22 ± 0.04	0.18 ± 0.01	0.30 ± 0.07	121.7	267
EXIT	0.28 ± 0.01	0.24 ± 0.01	0.27 ± 0.00	182.3	271
Cont-GRU	**0.18 ± 0.00**	**0.14 ± 0.01**	**0.21 ± 0.01**	67.85	20.2

and our settings are much more challenging. For all cases, Cont-GRU shows the best accuracy. Jump-based models, i.e., GRU-ODE-Bayes and NJODE, show poor accuracy, which shows that their piece-wise continuous concepts do not effectively process weather data. Exceptionally, ODE-RNN, whose jump mechanism is based on RNN cells, shows good performance. Since GRU-based models are typically used for time series forecasting, some of them show reasonable results with small standard deviations. We visualize the prediction results of Cont-GRU and the top-2 baseline models for 20 sequences in Appendix B.

4.2 Predict patient condition with irregular missing data

Sepsis (Reyna et al., 2019; Reiter, 2005) is a life-threatening condition caused by bacteria or bacterial toxins in the blood. About 1.7 million people develop sepsis in the U.S., and 270,000 die from sepsis in a year. More than a third of people who die in U.S. hospitals have sepsis. Early sepsis prediction could potentially save lives, so this experiment is especially meaningful.

The dataset used in this paper consists of data from 40,335 patients in intensive care units (ICU). The data consists of 5 static variables that do not change over time, such as gender and age, and 34 non-static variables, such as the respiratory rate or partial pressure of carbon dioxide from arterial blood (PaCO2). This data can be described as an irregular time series dataset with 90% of values removed from the original full data to protect the privacy of patients. To classify the onset of sepsis, we consider the first 72 hours of the patient's hospitalization.

Table 3: PhysioNet Sepsis

Model	AUCROC	Memory (MB)	Training Time (s)
NODE	0.53 ± 0.04	177.8	51.8
ODE-RNN	0.87 ± 0.02	387.3	10.8
GRU-Δt	0.88 ± 0.01	352.5	2.03
GRU-D	0.87 ± 0.02	369.8	2.45
GRU-ODE	0.85 ± 0.01	178.8	15.5
Latent-ODE	0.79 ± 0.01	207.8	314
Augmented-ODE	0.83 ± 0.02	742.3	322
ACE-NODE	0.80 ± 0.01	184.5	212
GRU-ODE-Bayes	0.52 ± 0.01	456.3	104
NJODE	0.53 ± 0.01	401.4	113
NCDE	0.88 ± 0.01	204.4	191
ANCDE	0.90 ± 0.00	247.7	181
EXIT	0.91 ± 0.00	257.2	211
Cont-GRU	**0.93 ± 0.04**	177.9	49.2

Experimental results Table 3 shows our experimental results of the time series classification with PhysioNet Sepsis. We conduct the time series classification task with observation intensity (OI) as an additional variable, which was used in (Kidger et al., 2020). We report AUCROC rather than accuracy because the dataset is significantly imbalanced. Cont-GRU shows the best performance and the model size is small in comparison with other differential equation-based models. In this dataset, however, more than 90% of values are missing to protect the privacy of patients. For this reason, GRU-ODE-Bayes and NJODE do not show good performance. They can process irregular time series, but their accuracies are worse than others. We consider that this is because they piece-wise continuously generalize the hidden state only. However, all NCDE-based models, i.e., NCDE, ANCDE, and EXIT, show reasonable results since they fully continuously generalize the hidden state.

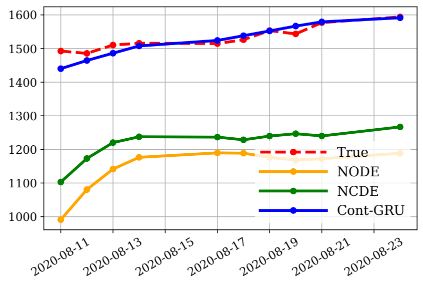
4.3 Forecast volatile stock prices and volumes

Stock prices are the results of the combination of social conditions and people's psychological factors (Andreassen, 1987; Wäneryd, 2001). Thus, accurate stock price forecasting is a very challenging

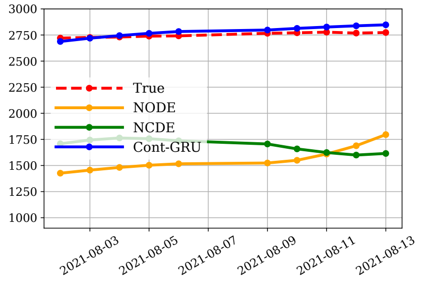
Table 4: Google Stock

Model	Test MSE			Memory	Training
	30% dropped	50% dropped	70% dropped	(MB)	Time (s)
NODE	0.057 ± 0.006	0.054 ± 0.005	0.052 ± 0.013	4.23	1.14
ODE-RNN	0.116 ± 0.018	0.145 ± 0.006	0.129 ± 0.011	5.95	0.35
GRU-Δt	0.145 ± 0.002	0.146 ± 0.001	0.145 ± 0.002	2.35	0.02
GRU-D	0.143 ± 0.002	0.145 ± 0.002	0.146 ± 0.002	3.33	0.02
GRU-ODE	0.064 ± 0.009	0.057 ± 0.003	0.059 ± 0.004	7.93	0.28
Latent-ODE	0.052 ± 0.005	0.053 ± 0.001	0.054 ± 0.007	17.3	4.17
Augmented-ODE	0.045 ± 0.004	0.051 ± 0.005	0.057 ± 0.002	21.3	4.26
ACE-NODE	0.044 ± 0.002	0.053 ± 0.008	0.056 ± 0.003	22.6	4.88
GRU-ODE-Bayes	0.175 ± 0.001	0.185 ± 0.022	0.197 ± 0.013	24.2	4.83
NJODE	0.185 ± 0.002	0.191 ± 0.012	0.181 ± 0.031	20.8	3.48
NCDE	0.056 ± 0.015	0.054 ± 0.002	0.056 ± 0.007	9.36	1.37
ANCDE	0.048 ± 0.012	0.047 ± 0.001	0.049 ± 0.004	10.3	3.42
EXIT	0.042 ± 0.020	0.045 ± 0.001	0.046 ± 0.002	14.7	4.01
Cont-GRU	**0.007 ± 0.001**	**0.007 ± 0.001**	**0.007 ± 0.002**	8.89	0.79

task. Particularly, forecasting stock prices, including the duration of COVID-19, makes our task more challenging and can properly evaluate time series forecasting models. We use the Google Stock data (Alphabet, 2021), which has six variables, i.e., the trading volumes of Google in conjunction with its high, low, open, close, and adjusted closing prices. We use the period from 2011 to 2021 of Google stock data, purposely including the COVID-19 period. The goal is, given the past 20 days of the time series values, to forecast the high, low, open, close, adjusted closing price, and volumes at the very next 10 days.



(a) Open price



(b) High price

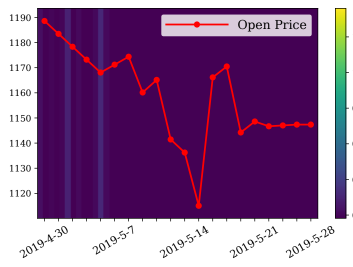
Figure 3: Forecasting visualization on Google Stock

Experimental results The experimental results on Google Stock are in Table 4. We conduct three sorts of experiments — randomly drop 30%, 50%, and 70% of observations in a time series sample. Overall, our model, Cont-GRU, shows the best accuracy. One impressive outcome of our method is that it is not greatly affected by the dropping ratio. ODE-based models, except ODE-RNN, show reasonable results and CDE-based models show better results than ODE-based models. Among CDE-based models EXIT shows the second-best performance in all the dropping ratio settings. Figure 3 visualizes prediction results. Figure 3(a) forecasts the open price from August 11, 2020 to August 23, 2020, and Figure 3(b) forecasts the high price from August 3, 2021 to August 13, 2021. Compared with the top-2 baseline models, NODE and NCDE, our model forecasts more accurately. More visualizations of the forecasting results are in Appendix C.

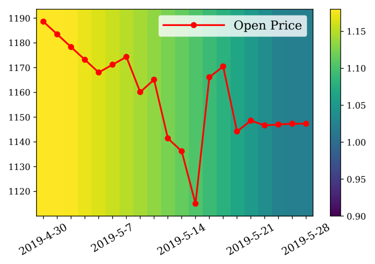
Figure 4 shows the difference between the reset gates of Cont-GRU and GRU-ODE-Bayes. The role of the reset gate is to determine how much of the previous hidden state is reflected. The red line in Figure 4 shows the stock market price for the 20-day period from April 30 to May 28, 2019. One can see that the reset gate of GRU-ODE-Bayes does not fluctuate much, but the reset gate of Cont-GRU captures meaningful information. More visualizations of values in the reset gate are in Appendix D.

4.4 Empirical study on interpolation methods

In this section, we further experiment with several interpolation methods to create a continuous path $\mathbf{x}(t)$ from $\{(\mathbf{x}_i, t_i)\}_{i=0}^{N-1}$ and compare their results. The results are in Table 5. We test with the natural cubic spline (McKinley & Levine, 1998), linear control (Martin et al., 1995), and cubic Hermite spline (De Boor et al., 1987) methods. We show that the interpolation method leads to the continuous derivative of $\frac{d\mathbf{x}(t)}{dt}$ in Appendix E, which enables our DDE-based *fully continuous* Cont-GRU.



(a) GRU-ODE-Bayes



(b) Cont-GRU

Figure 4: Values in the reset gate. Our method gives higher weights to recent observations whereas GRU-ODE-Bayes gives almost equal weights to all observations, which shows the correctness of our method.

Table 5: Interpolation methods

Interpolation Methods	USHCN (MSE)	Sepsis (AUCROC)	Google Stock (MSE)
Natural Cubic Spline	0.15 ± 0.03	0.89 ± 0.01	**0.007 ± 0.001**
Linear Control	0.16 ± 0.02	0.88 ± 0.04	0.008 ± 0.001
Cubic Hermite Spline	**0.14 ± 0.01**	**0.93 ± 0.04**	**0.007 ± 0.001**

Natural cubic spline The natural cubic spline method must have access to the entire time series data for this control signal before constructing a continuous path. Changes to one previous data point do not affect the overall structure. This method can be integrated numerically quickly because it is relatively smooth and changes slowly.

Linear control The linear control method generates the simplest and most natural control signal among the interpolation methods. An interpolated path is generated while applying the linear interpolation between observations. This linear control defines discrete online control paths for all observed data across all time, and so has the same online qualities as RNNs.

Cubic Hermite spline The linear control method is discrete, which can be a drawback. However, the cubic Hermite spline interpolation method smooths out discontinuities. This method achieves this by combining adjacent observations with a cubic spline. Comparing the cubic Hermite spline method with the natural cubic spline method, the main difference is that equations are solved independently at each point in time.

For USHCN, the cubic Hermite spline method shows the best performance. However, all interpolation methods show good results. In the case of PhysioNet Sepsis, the cubic Hermite spline method shows the best performance. Since it is highly irregular, the linear control method shows relatively poor results. In Google Stock, the linear control method and the cubic Hermite spline method show the best performance among the three interpolation methods. However, all three interpolation results are significantly better than existing baselines.

5 Conclusions and Limitations

We present the first fully continuous GRU model. The hidden state $\mathbf{h}(t)$ had been continuously generalized by existing methods. However, to our knowledge, Cont-GRU is the first model to successively generalize all parts (gates) of the GRU, including hidden states. To this end, we rely on interpolation methods to reconstruct a continuous path from a discrete time series sample. We then define a DDE-based model to interpret GRUs in a continuous manner. In our experiment with 3 real-world datasets and 13 baselines, our method consistently shows the best accuracy. Interestingly, other piece-wise continuous models generalizing the hidden state only do not work well in some cases where our fully continuous model works well. We consider that these experimental results well prove the efficacy of the fully continuous model. However, our model shows good performance, but there exists room for improvement. For example, in the USHCN dataset, our model performs well at forecasting sudden changes, but its absolute error scale is not always satisfactory.

6 Ethics Statement

In PhysioNet Sepsis, there are 40,335 patients' bio data recorded for 72 hours. We downloaded this data from PhysioNet. They intentionally deleted 90% of observations to protect the patients' personal information. Therefore, there are no concerns about their privacy leakage via our research.

Our method can be used for various purposes. One can use our method for forecasting stock prices and Bitcoin prices to abuse the forecast information. At the same time, however, others can use it for their business decision-making, e.g., controlling the risk of financial institutions. In total, we consider that our research will bring more positive benefits to our society than negative ones.

7 Reproducibility Statement

The source codes and data required to reproduce the reported results are available in the supplementary material. We enclose a ReadMe file describing all the detailed execution steps to reproduce our results. We also list all the GitHub links to our baselines and all the hyperparameter settings we used for our experiments.

References

Alphabet. Google Stock. `https://finance.yahoo.com/quote/GOOG/history?p=GOOG&guccounter=1`, 2021.

Paul B Andreassen. On the social psychology of the stock market: Aggregate attributional effects and the regressiveness of prediction. *Journal of Personality and Social Psychology*, 53(3):490, 1987.

Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. In *NeurIPS*, 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 T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Carl De Boor, Klaus Höllig, and Malcolm Sabin. High accuracy geometric hermite interpolation. *Computer Aided Geometric Design*, 4(4):269–278, 1987.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In *NeurIPS*, 2019.

Tak-chung Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

Aditya Grover, Ashish Kapoor, and Eric Horvitz. A deep hybrid model for weather forecasting. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 379–386, 2015.

Calypso Herrera, Florian Krach, and Josef Teichmann. Neural jump ordinary differential equations: Consistent continuous-time prediction and filtering. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=JFKR3WqwyXR`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Sheo Yon Jhin, Minju Jo, Taeyong Kong, Jinsung Jeon, and Noseong Park. Ace-node: Attentive co-evolving neural ordinary differential equations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 736–745, 2021a.

Sheo Yon Jhin, Heejoo Shin, Seoyoung Hong, Minju Jo, Solhee Park, Noseong Park, Seungbeom Lee, Hwiyoung Maeng, and Seungmin Jeon. Attentive neural controlled differential equations for time-series classification and forecasting. In *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 250–259. IEEE, 2021b.

Sheo Yon Jhin, Jaehoon Lee, Minju Jo, Seungji Kook, Jinsung Jeon, Jihyeon Hyeong, Jayoung Kim, and Noseong Park. Exit: Extrapolation and interpolation-based neural controlled differential equations for time-series classification and forecasting. In *Proceedings of the ACM Web Conference 2022*, pp. 3102–3112, 2022.

Ian D Jordan, Piotr Aleksander Sokół, and Il Memming Park. Gated recurrent units viewed through the lens of continuous time dynamical systems. *Frontiers in computational neuroscience*, pp. 67, 2021.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. In *NeurIPS*, 2020.

Mantas Lukoševičius and Arnas Uselis. Time-adaptive recurrent neural networks, 2022. URL `https://arxiv.org/abs/2204.05192`.

Terry Lyons, M. Caruana, and T. Lévy. *Differential Equations Driven by Rough Paths*. Springer, 2004. École D'Eté de Probabilités de Saint-Flour XXXIV - 2004.

Clyde Martin, Per Enqvist, John Tomlinson, and Zhimin Zhang. Linear control theory, splines and interpolation. In *Computation and Control iv*, pp. 269–287. Springer, 1995.

Sky McKinley and Megan Levine. Cubic spline interpolation. *College of the Redwoods*, 45(1): 1049–1060, 1998.

Matthew J Menne and Claude N Williams Jr. Homogenization of temperature series via pairwise comparisons. *Journal of Climate*, 22(7):1700–1717, 2009.

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

P. Jerome Reiter. Using cart to generate partially synthetic, public use microdata. *Journal of Official Statistics*, 21:441, 01 2005.

Matthew A Reyna, Chris Josef, Salman Seyedi, Russell Jeter, Supreeth P Shashikumar, M Brandon Westover, Ashish Sharma, Shamim Nemati, and Gari D Clifford. Early prediction of sepsis from clinical data: the physionet/computing in cardiology challenge 2019. In *CinC*, pp. Page 1–Page 4, 2019. doi: 10.23919/CinC49843.2019.9005736.

Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *NeurIPS*. 2019.

Afan Galih Salman, Bayu Kanigoro, and Yaya Heryadi. Weather forecasting using deep learning techniques. In *2015 international conference on advanced computer science and information systems (ICACSIS)*, pp. 281–285. Ieee, 2015.

Mona Schirmer, Mazin Eltayeb, Stefan Lessmann, and Maja Rudolph. Modeling irregular time series with continuous recurrent units. *CoRR*, abs/2111.11344, 2021. URL `https://arxiv.org/abs/2111.11344`.

Karl-Erik Wäneryd. *Stock-market psychology: How people value and trade stocks*. Edward Elgar Publishing, 2001.

Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

Qunxi Zhu, Yao Guo, and Wei Lin. Neural delay differential equations. *arXiv preprint arXiv:2102.10801*, 2021.

Qunxi Zhu, Yifei Shen, Dongsheng Li, and Wei Lin. Neural piecewise-constant delay differential equations. *arXiv preprint arXiv:2201.00960*, 2022.

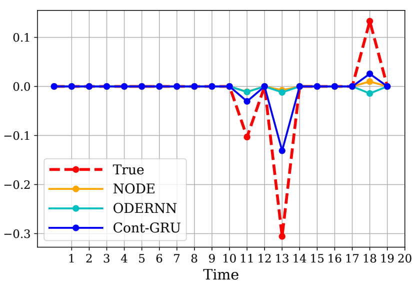
A Baselines

We test the following state-of-the-art baselines to compare our proposed Cont-GRU with:

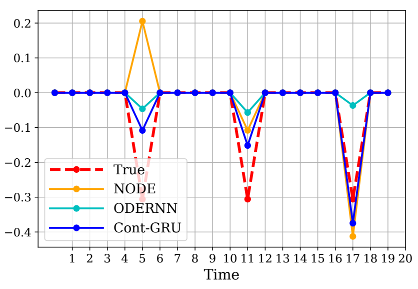
1. NODEs (Chen et al., 2018) are a continuous-time model that defines the hidden state $\mathbf{h}(t)$ with an initial value problem (IVP).
2. ODE-RNN is a combination of RNN and NODE. When a NODE evolves a hidden state $\mathbf{h}(t)$ continuously between observations, an RNN Cell causes that $\mathbf{h}(t)$ jumps to another hidden state when a new observation arrives.
3. GRU-Δt and GRU-D are advanced versions of GRU to process irregular time series. GRU-Δt is a GRU that additionally takes the time difference between observations as an input. GRU-D (Che et al., 2018) is a modified version of GRU-Δt with learnable exponential decay between observations.
4. GRU-ODE similar to Neural ODE. The only difference is that used GRU Cell as an ODE function, GRU-ODE is piece-wise continuous through all time.
5. Latent-ODE is a model that can explain the latent state with ODEs, and is a suitable model for time series prediction.
6. Augmented-ODE is the method proposed by (Dupont et al., 2019), which inserts zeros to the ODE state of Latent-ODE.
7. ACE-NODE (Jhin et al., 2021a) is an attention-based Neural ODE model.
8. GRU-ODE-Bayes (Brouwer et al., 2019) is a combination of GRU-ODE and GRU-Bayes. GRU-ODE calculates a hidden state $\mathbf{h}(t)$ in a continuous manner between observations, and GRU-Bayes is used to discretely jump $\mathbf{h}(t)$ to another state when a new observation arrives. GRU-ODE-Bayes is often used to predict sporadically observed data. However, those models that combine NODEs with jumps, such as ODE-RNN and GRU-ODE-Bayes, are not completely continuous but piece-wise continuous.
9. Neural jump ODEs (NJODEs) (Herrera et al., 2021), on the other hand, are a data-driven approach that continuously learns the conditional expectations of a probabilistic process.
10. Neural CDEs (NCDEs (Kidger et al., 2020)) are a conceptually enhanced model of NODEs based on the theory of controlled differential equations.
11. Attentive Neural CDE (ANCDE) (Jhin et al., 2021b) is an attention-based NCDE model.
12. Extrapolation and Interpolation-based Neural CDE (EXIT) (Jhin et al., 2022) use interpolation process of NCDEs, an encoder-decoder architecture corresponding to our neural network-based interpolation versus conventional explicit interpolation, to generate another latent continuous path and exploit its generative properties. In this paper, they can extrapolate beyond the temporal domain of the original data, if necessary. Thus, NCDEs design of EXIT can use both interpolated and extrapolated information for downstream machine learning tasks.

The best hyperparameters of our model and baselines are reported in Appendix G.

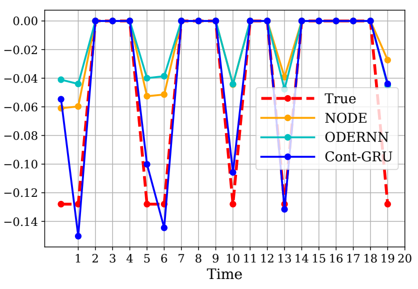
1. NODE : `https://github.com/rtqichen/torchdiffeq`
2. ODE-RNN, GRU-Δt, GRU-D, GRU-ODE and NCDE : `https://github.com/patrick-kidger/NeuralCDE`
3. Latent-ODE, Augmented-ODE : `https://github.com/YuliaRubanova/latent_ode`
4. ACE-NODE : `https://github.com/sheoyon-jhin/ACE-NODE`
5. GRU-ODE-Bayes : `https://github.com/edebrouwer/gru_ode_bayes`
6. NJODE : `https://github.com/HerreraKrachTeichmann/NJODE`
7. ANCDE : `https://github.com/sheoyon-jhin/ANCDE`
8. EXIT : `https://github.com/sheoyon-jhin/EXIT`



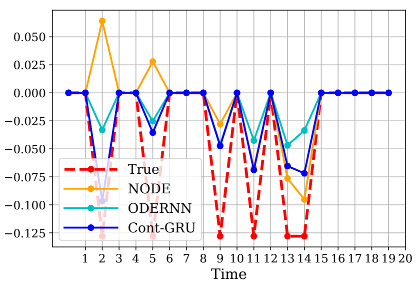
(a) SNOW



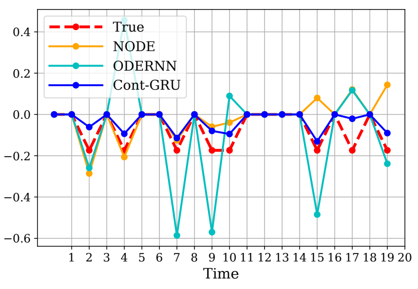
(b) SNOW



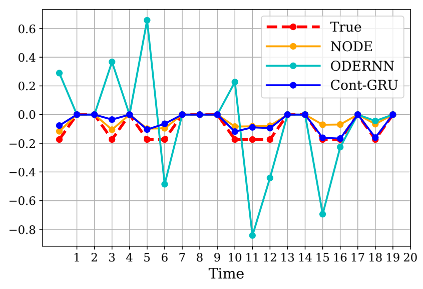
(c) PRCP



(d) PRCP



(e) SNWD



(f) SNWD

Figure 5: Forecasting visualization on USHCN

B Visualization on USHCN

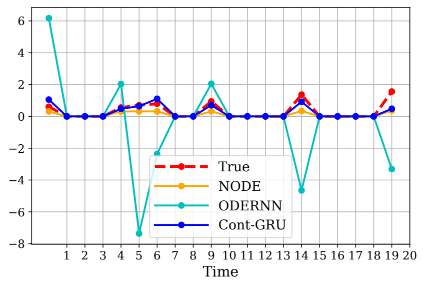
Figure 6 visualizes the prediction results for snowfall (SNOW), precipitation (PRCP), snow depth (SNWD), maximum temperature (TMAX), and minimum temperature (TMIN). It can be seen that our model in blue line predicts trends better than the top-2 baseline models, ODE-RNN and NODE.

C Visualization on Google Stock

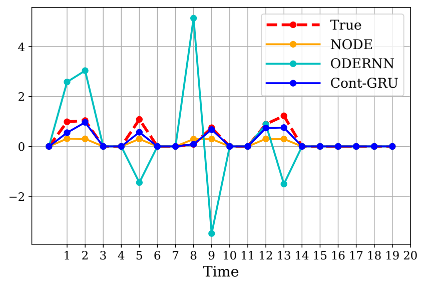
In addition to the period shown in Figure 3, we show the visualization of Google Stock for various periods and various variables in Figure 7 and 8.

D Values in the reset gate

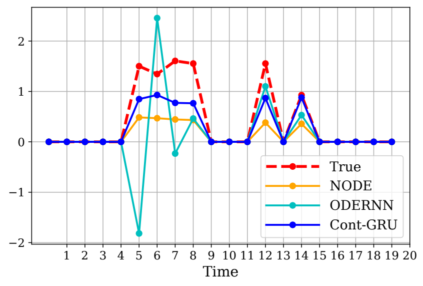
In Figure 9, we visualize the results of the role of the reset gate. We visualize for the 20-day period i) from November 13, 2017 to December 11, 2017, and ii) from May 7, 2021 to June, 4 2021 in Google Stock.



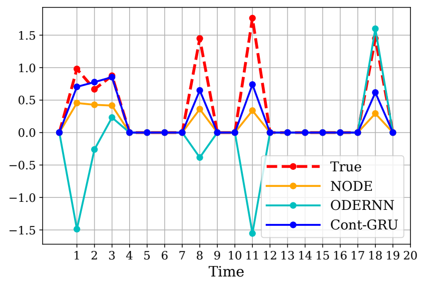
(a) TMAX



(b) TMAX



(c) TMIN



(d) TMIN

Figure 6: Forecasting visualization on USHCN

E Specific description of Interpolation Methods

A cubic spline is a spline of piece-wise cubic polynomials passing through m given control points. A second order derivative of zero at the endpoints is assumed to solve for a cubic spline. This provides boundary conditions that allow the two equations to be solved with respect to the m-2 equation. The system of equations for a cubic spline in one dimension can be written as :

$$S_i(x) = a_i + b_i * x + c_i * x^2 + d_i x^3 \tag{12}$$

For the function $y = \mathbf{f}(x)$, we take a set of points $[x_i, y_i]$ where $i \in \{0, 1, 2, ..., n\}$. It is a piece-wise continuous curve that passes through each value in the table for cubic spline interpolation. The prerequisites for the spline of degree $K = 3$ are as follows: first, the domain of s must lie inside the intervals of $[a, b]$; second, all three parameters must be continuous functions on $[a, b]$.

$$S(x) = \{S_i(x) | x \in [t_{i-1}, t_i], i \in [1, n)\} \tag{13}$$

$S_i(x)$ is defined as a cubic polynomial in the sub interval $[x_i, x_{i+1}]$. We need $4n$ parameters to solve the spline because there are n intervals and 4 coefficients in each equation. We may derive the $2n$ equation from the requirement that each cubic spline equation meet the value at both ends:

$$S_i(x_i) = y_i, \tag{14}$$

$$S_i(x_{i+1}) = y_{i+1} \tag{15}$$

In addition to being continuous and differentiable, the aforementioned cubic spline equations should also have defined first and second derivatives that are continuous on control points.

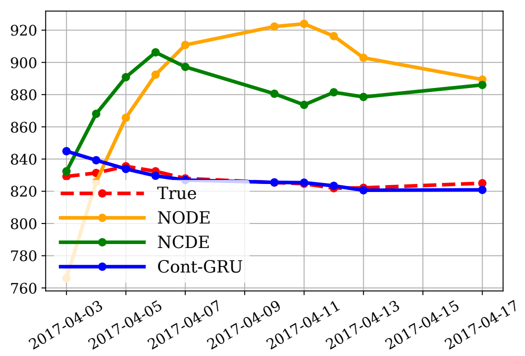
$$S_{i-1}^{'}(x_i) = S_i^{'}(x_i) \tag{16}$$

$$S_{i-1}^{'}(x_i) = S_i^{'}(x_i) \tag{17}$$

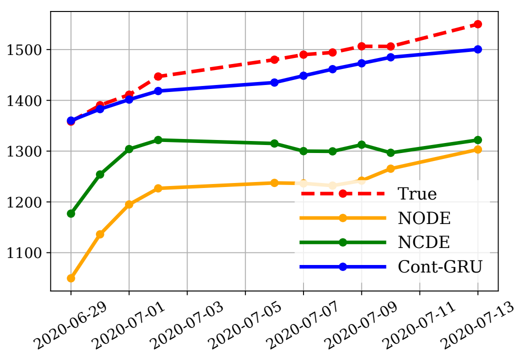
The $2n - 2$ equation restrictions are given for $\{1, 2, 3, ..., n - 1\}$. Thus, two additional equations are required to solve the above cubic spline. We'll employ some natural boundary conditions for it. We assume that the second derivative of the spline at boundary points is zero in the Natural Cubic Spline:

$$S^{''}(x_0) = 0 \tag{18}$$

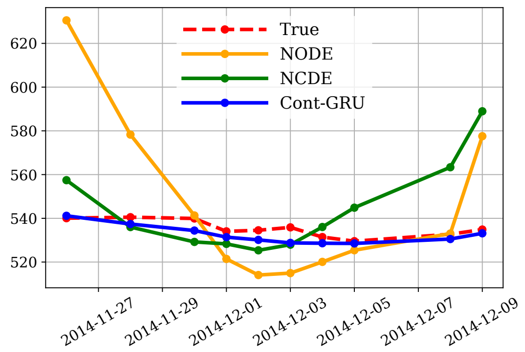
$$S^{''}(x_n) = 0 \tag{19}$$



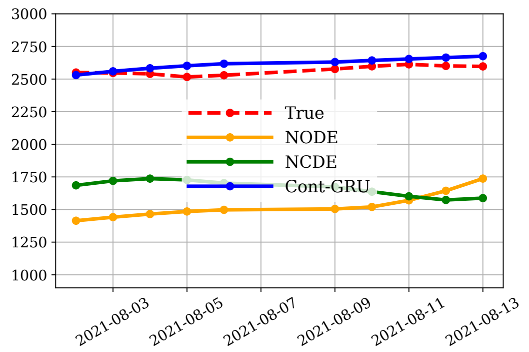
(a) Open price



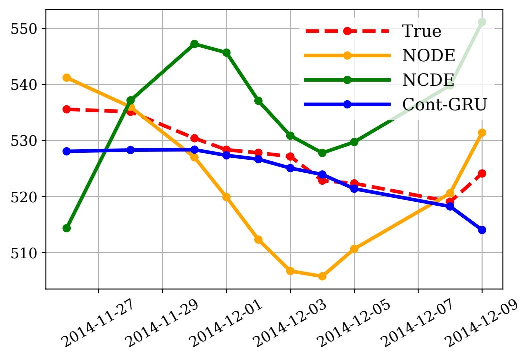
(b) Open price



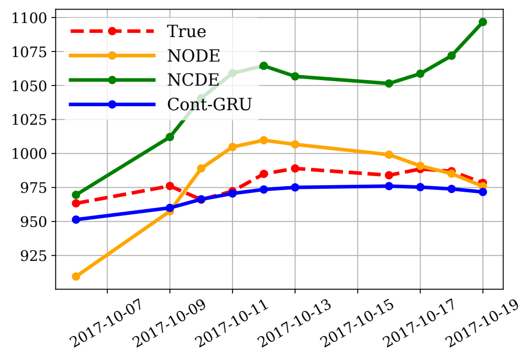
(c) High price



(d) High price

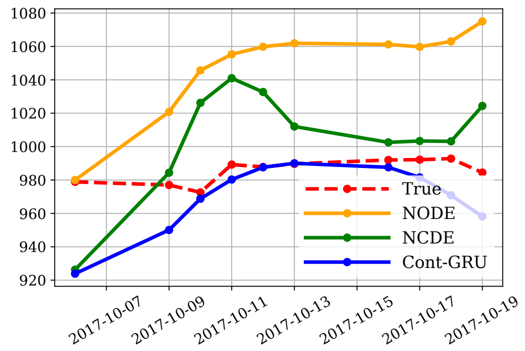


(e) Low price

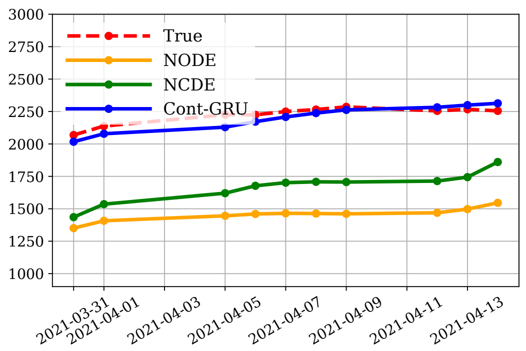


(f) Low price

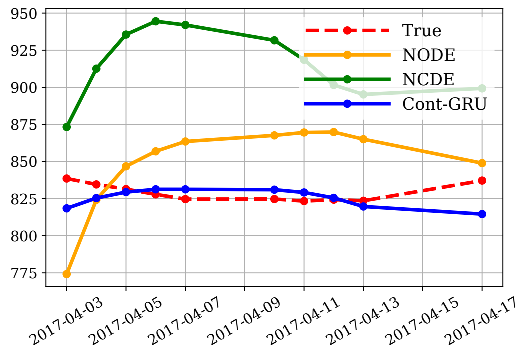
Figure 7: Forecasting visualization on Google Stock



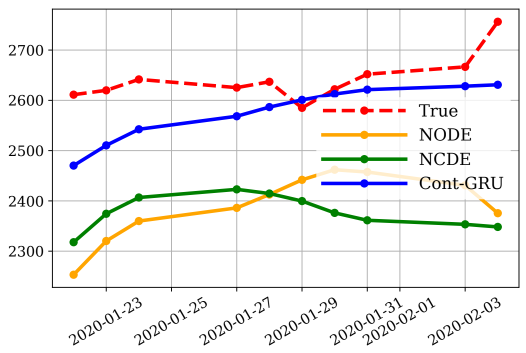
(a) Close price



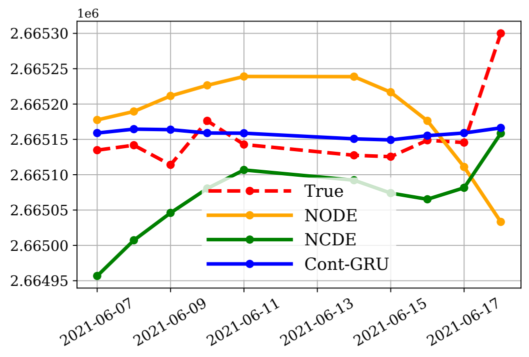
(b) Close price



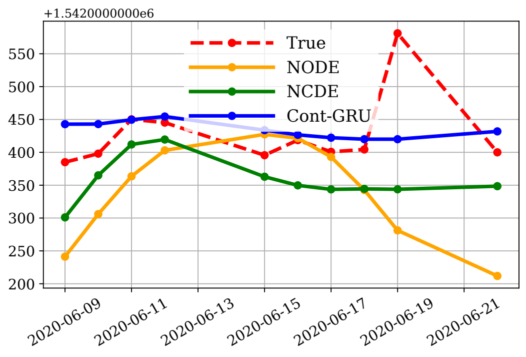
(c) Adjusted close price



(d) Adjusted close price

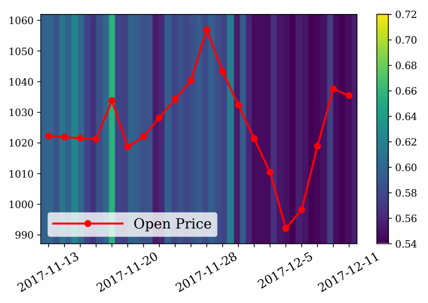


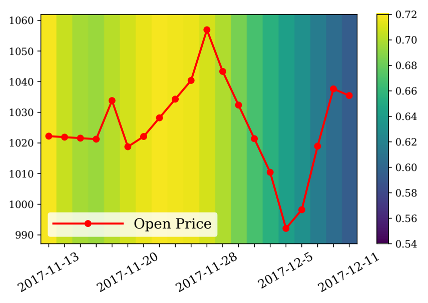
(e) Volume

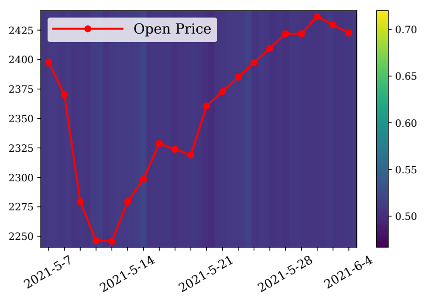


(f) Volume

Figure 8: Forecasting visualization on Google Stock


(a) GRU-ODE-Bayes


(b) Cont-GRU


(c) GRU-ODE-Bayes

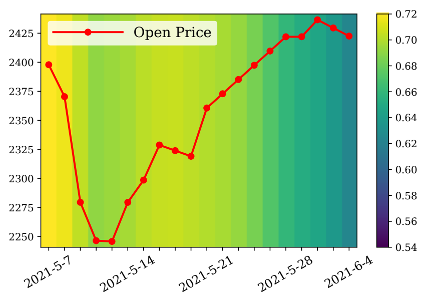

(d) Cont-GRU

Figure 9: Values in the reset gate

Now that we know that the $S(x)$ is a third-order polynomial, we may infer that the $S^{''}(x)$ is an interpolating linear spline. As a result, we first create $S^{''}(x)$ before twice integrating it to get $S(x)$. Now, let's assume that $z_i = S^{''}(x_i), i \in \{0, 1, 2..n\}$ and from the natural boundary condition $z_0 = z_n = 0$ that $t_i = x_i$ for $i \in \{0, 1, ...n\}$. A linear spline is produced by twice differentiating a cubic spline, and it has the following notation:

$$S_i^{''}(x) = z_i \frac{x - t_{i+1}}{t_i - t_{i+1}} + z_{i+1} \frac{x - t_i}{t_{i+1} - t_i} \tag{20}$$

where, $\Delta t_i = t_{i+1} - t_i; t \in \{0, 1, 2, ..., n\}$. The equation is written as : $S^{''}(x) = z_{i+1} \frac{x - t_i}{\Delta t_i} + z_i \frac{t_{i+1} - x}{\Delta t_i}$ Integrating this equation twice we get a cubic spline.

$$S(x) = \frac{z_{i+1}}{6\Delta t_i}(x - t_i)^3 + \frac{z_i}{6\Delta t_i}(t_{i+1} - x)^3 + C_i(x_i - t) + D_i(t_{i+1} - x) \tag{21}$$

where,

$$C_i = \frac{y_{i+1}}{\Delta t_i} - \frac{z_{i+1} * \Delta t_i}{6} \tag{22}$$

$$D_i = \frac{y_i}{\Delta t_i} - \frac{z_i * \Delta t_i}{6} \tag{23}$$

Now, verify the derivative at t_i is continuous. We must first locate the derivative and add the following condition:

$$S'(x) = \frac{z_{i+1}}{2\Delta t_i}(x - t_i)^2 - \frac{z_i}{2\Delta t_i}(t_{i+1} - x)^2 + \frac{1}{\Delta t_i}(y_{i+1} - y_i) - \frac{\Delta t_i}{6}(z_{i+1} - z_i) \tag{24}$$

where, $b_i = \frac{1}{\Delta t_i}(y_{i+1} - y_i)$. The following equation results from calculating the continuity equation mentioned above:

$$6(b_i - b_{i-1}) = \Delta t_{i-1} z_{i-1} + 2(\Delta t_{i-1} + \Delta t_i) z_i + \Delta t_i z_{i+1} \tag{25}$$

Since there are n parameters of our z, and n systems of equations we know, we can find z by solving the system of equations.

F Data preprocessing details

The datasets used in our experiments are publicly available and can be downloaded at the following locations:

1. USHCN : `https://cdiac.ess-dive.lbl.gov/ftp/ushcn_daily/`
2. PhysioNet Sepsis[2] : `https://physionet.org/content/challenge-2019/1.0.0/`
3. Google Stock : `https://finance.yahoo.com/quote/GOOG/history?p=GOOG`

We split the entire dataset into training/validating/testing parts. The first 70% of the data is used as training, 15% is used for validating, and the last 15% is used for testing.

F.1 USHCN

We follow the preprocessing process of GRU-ODE-Bayes (Brouwer et al., 2019). We look at 100 time sequences for training and forecast next 10,20,30 time sequences. And the stride interval is 50 time sequences.

F.2 PhysioNet Sepsis

We follow the preprocessing settings of NCDE (Kidger et al., 2020). NCDE used a new variable, called observation intensity (OI), for learning. The observation intensity (OI) is the frequency of observations.

F.3 Google Stock

We used the Google Stock data from 2011 to 2021. Since the scale of each variable is different, normalization was performed between 0 and 1. We look at 20 time sequences for training and forecast next 10 time sequences. And the stride interval is 5 time sequences.

G Hyperparameters

For reproducibility, we also report the best hyperparameters.

G.1 USHCN

In USHCN, we train for 150 epochs with a batch size of 256, and stop early if the training loss doesn't decrease for 50 epochs. A hidden size in $\{19, 29, 39, 49, 59, 69\}$ and a learning rate λ in $\{1.0e^{-3}, 5.0e^{-3}, 1.0e^{-2}, 5.0e^{-2}\}$ are used.

1. For baselines, we use $\lambda = 5.0e^{-3}$, hidden size = 49, and weight decay = $1.0e^{-3}$.
2. For Cont-GRU, we use $\lambda = 5.0e^{-3}$, hidden size = 19, and weight decay = $1.0e^{-4}$.

G.2 PhysioNet Sepsis

In PhysioNet Sepsis, we train for 100 epochs with a batch size of 1,024, and stop early if the training loss doesn't decrease for 50 epochs. A hidden vector size in $\{29, 39, 49, 59, 69\}$ and a learning rate λ in $\{1.0e^{-3}, 5.0e^{-3}, 1.0e^{-2}, 5.0e^{-2}\}$ are used.

1. For baselines, we use $\lambda = 1.0e^{-2}$, hidden size = 49, and weight decay = $1.0e^{-3}$.
2. For Cont-GRU, we use $\lambda = 1.0e^{-3}$, hidden size = 59, and weight decay = $1.0e^{-2}$.

[2]This dataset follows the license policy of CC-BY 4.0.

G.3 Google Stock

In Google Stock, we train for 200 epochs with a batch size of 256, and stop early if the training loss doesn't decrease for 100 epochs. We use a hidden vector size of $\{15, 25, 35, 45\}$, and a learning rate λ of $\{1.0e^{-4}, 5.0e^{-4}, 1.0e^{-3}, 5.0e^{-3}\}$.

1. For baselines, we use $\lambda = 1.0e^{-2}$, hidden size = 25, and weight decay = $1.0e^{-4}$.
2. For Cont-GRU, we use $\lambda = 1.0e^{-2}$, hidden size = 25, and weight decay = $1.0e^{-4}$.

H Computing infrastructures

In this section, we describe our software/hardware environments. All experiments were conducted in the following software and hardware environments: Ubuntu 18.04 LTS, Python 3.8, Numpy 1.21.5, Scipy 1.7.3, Matplotlib 3.3.1, PyTorch 1.7.1, CUDA 11.0, GeForce RTX 3090. We repeat the training and testing procedures with five different random seeds and report their mean and standard deviation accuracy.

I Why GRUs should be formulated as DDEs?

There are several attempts to interpret GRUs in a continuous manner for solving problems such as time series forecasting, classification, and so forth Mozer et al. (2017); Jordan et al. (2021). This shows how meaningful those attempts are in the field of time series. All these continuous interpretations of GRUs are robust to irregular time series processing — of course, then can process regular time series as well. Therefore, one can use a trained continuous model for real-world environments where time series observations can be missing from time to time (due to malfunctioning sensors and/or communication channels while collecting data).

In addition, our Cont-GRU, the first DDE-based continuous modeling of GRUs, has a couple of strong points in comparison with existing methods: i) Our DDE-based modeling does not have the homeomorphic limitation of ODEs. In all the tasks in our experiments, our method significantly outperforms existing methods. ii) Our method has smaller empirical memory usage and training time than those of existing continuous models (as reported in our experimental results). Owing to the increased model capability by DDEs, our model size is relatively smaller than others, resulting in lightweight empirical processing.

In the design philosophy of GRU cells, previous information is used to determine current output, i.e., long/short-term dependencies, and this concept is closely related to that of DDEs. Since DDEs are to model time-delay systems where long/shot-term past information influences current output, we believe that GRUs should be continuously interpreted in DDEs and our experimental results justify it.