Tensorized Timeline Alignment for Neural Dynamical Systems in Irregularly Sampled Time Series Prediction

Linxiao Qin[1], Tao Sun[1,2], Shuo Zhang[1], Xi-Ming Sun[1]

1. School of Control Science and Engineering, Dalian University of Technology, Dalian, 116024
E-mail: qinlinxiao@mail.dlut.edu.cn; shuozhang@dlut.edu.cn; sunxm@dlut.edu.cn

2. Department of Automation, Tsinghua University, Beijing, 100084
E-mail: suntao2022@tsinghua.org.cn

Abstract: For many real-world scenarios, time series prediction needs to be performed on irregularly sampled data. Neural ordinary differential equations have successfully introduced neural dynamical system (NDS) methods into such tasks. However, these models suffer from a large computational cost, and cannot effectively utilize the temporal information in mini-batches, due to mismatched timestamps of samples. In this work, a unified architecture for NDS is developed, with two novel implementations on discrete input sequences (named TDINDS) and continuous sequences (named TCINDS). The reuse of network modules effectively reduces the scale of parameters and time consumption, while maintaining accuracy. In addition, a tensorized timeline alignment method is proposed, which preserves the temporal information of each training sample via reversible mappings, improving prediction performance and reducing time complexity by at least one order of magnitude. Experiments are conducted on the GoogleStock dataset, where we evaluate model accuracy and computational cost by predicting irregular future sequences. The model robustness is tested using incomplete inputs. The results demonstrate the superior performance of TDINDS and TCINDS, with mean square error reduced by 44% and 50% respectively. Robustness evaluation shows that tensorized timeline alignment is also a key factor to enhance irregular time series prediction ability.

Key Words: Tensorized deep learning, neural ordinary differential equations, time series prediction, continuous neural networks

1 Introduction

Time series, also known as historical plural or dynamic sequence. It is to arrange the values of certain statistical indicators in chronological order to form a series of numbers. In particular, time series prediction refers to the analysis of the direction and degree of change based on the time series data arranged in chronological order, so as to speculate on the possible goals in the future. Thus, the time series prediction is essentially a regression forecasting and belongs to quantitative forecasting. In life and economic activities, the time interval between each two records could vary a lot, resulting irregularly sampled time series. For example, the spiking events of neurons, medical records and unscheduled production scheme changes. Time series prediction of such data is one of the most challenging problems in the context of regression forecasting [1].

In fact, time series prediction methods are mainly divided into traditional time series prediction methods and deep learning methods. The traditional time series method refers to the idea of predicting the trend development of future time series only according to the trend development of historical time series. Such methods fit the historical time trend curve by establishing appropriate mathematical model, and predict the trend curve of future time series according to the established model. Therefore, traditional time series models mainly include autoregressive integrated moving average model, autoregressive conditional heteroscedasticity model, etc. However, the traditional time series prediction method relies on relatively simple data, and only the historical time series trend curve can be used to design the model. In addition, the prediction method often faces the lag problem, that is, the predicted value is several time units later than the true value [2].

In order to further improve the prediction accuracy of time series, deep learning algorithm is generally introduced into time series prediction. Such methods are usually modeled on the features that may affect the predicted value according to specific application scenarios, and introduce these features into the artificial neural network of deep learning to predict. Neural ordinary differential equations (Neural ODEs) can capture the dynamical properties in the form of ODEs and solve them at arbitrary steps, making such model competitive for irregularly sampled time series prediction [1]. However, the computational burden of these models is catastrophically high. They also cannot utilize the temporal information of training data in mini-batches, due to different timestamps of each sample.

In this paper, a unified architecture for predictive neural dynamical system is developed, with two novel implementations on discrete input sequences (named TDINDS) and continuous input sequences (named TCINDS). The reuse of network modules effectively reduces the scale of parameters and time consumption, while maintaining accuracy. In addition, a tensorized timeline alignment method is proposed, which preserves the temporal information of each training sample via reversible mappings, improving prediction performance and reducing time complexity by at least one order of magnitude. The model accuracy and computational cost are evaluated on the GoogleStock dataset, via multivariate and irregular future sequences prediction. The model robustness is tested using incomplete input sequences. The results demonstrate the superior performance of TDINDS and TCINDS, with mean square error reduced by 44% and 50% respectively. Robustness evaluation shows that tensorized timeline alignment is also a key factor to enhance irregular time series prediction ability.

This work was supported in part by the National Science and Technology Major Project under Grant J2019-V-0010-0105, and in part by the China Postdoctoral Science Foundation Funded Project under Grant 2022TQ0179. (Corresponding authors: Shuo Zhang, Tao Sun)

2 Related Work and Preliminaries

2.1 Neural ODEs

Neural ODEs [1] was originally introduced as a continuous version of ResNet [3]. It regards the residual module (1) of ResNet as the discretization of ODEs (2):

$$\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \mathrm{Net}(\boldsymbol{x}_j \mid \boldsymbol{\theta}_j), \tag{1}$$

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \boldsymbol{F}(t, \boldsymbol{x}(t) \mid \boldsymbol{\theta}_{\boldsymbol{F}}). \tag{2}$$

Neural ODE aims to obtain the final state $\boldsymbol{x}(t_N)$ through a numerical ODE solver, and further maps it to the data domain label $\hat{\boldsymbol{y}}$ for tasks such as dynamical modeling and image classification [1]. Unfortunately, the model directly fits $\boldsymbol{x}$ to the actual data, which has defects in capturing latent dynamical characteristics.

Considering the problem introduced by direct fitting method in the above model, the variants of Neural ODEs use a set of hidden variables $\boldsymbol{h}$ to form (2), and input acquired sampling data $\boldsymbol{X}$ in the solver to obtain prior distribution of hidden parameters $\mathbb{P}(\boldsymbol{\theta} \mid \boldsymbol{X})$. In this way, over parameterization can be avoided in dynamical modeling, and a sequence is output in the process of ODE solving. Such models for sequence data need to consider estimation error of all sample points.

ODE-RNN [4] and Neural CDE [5] are the representatives of this class. In the implementation of ODE-RNN, recurrent neural network inputs sample data when solving the ODE up to sample timestamps, introducing additional time overhead due to the interruption of solution process. Neural CDE uses spline interpolation method to convert discrete sampling values into continuous input trajectories $\hat{\boldsymbol{x}}$:

$$\frac{\mathrm{d}\boldsymbol{h}(t)}{\mathrm{d}t} = \boldsymbol{F}(\boldsymbol{h}(t) \mid \boldsymbol{\theta}_{\boldsymbol{F}}) \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t}, \tag{3}$$

where $\boldsymbol{F} \in \mathbb{R}^{D_h \times (D_x+1)}, \boldsymbol{u}(\tau) = [\tau, \hat{\boldsymbol{x}}(\tau)^T]^T \in \mathbb{R}^{D_x+1}$.

However, the computational cost of Neural CDE is still very high. This is because it outputs a large-scale vector field $\boldsymbol{F}$. Besides, the tensor multiplication of $\boldsymbol{F}$ and $\mathrm{d}\boldsymbol{u}$ is computed, which also limits the ability of Neural CDE to deal with large-scale time series data.

2.2 Timeline Mismatch and Reparameterization

Mini-batch is a method dividing train dataset into several subsets for parallel training. These subsets are called batches, which are usually composed of a fixed number of samples (with batch size B). Each iteration will select one from the subsets and train B samples (with sequence length N) together. Mini-batch not only overcomes the difficulty to converge in single sample iteration, but also avoids the phenomenon for training all samples at the same time with huge resources, becoming an iconic method for deep learning.

However, different samples in mini-batches may have mismatched time stamps, while the mainstream ODE numerical solvers only support solving [1] on a unique time axis. If each sample is solved on its corresponding time axis, the time complexity increases from $\mathcal{O}(N)$ to $\mathcal{O}(BN)$, which is especially unacceptable for tasks that need to set large B.

Nickel et al. [6] provides a method to introduce original timestamps into the ODE solver, which is called reparameterization. It maps the timestamps $t \in [t_1, t_N]$ of the sample $i = 1, 2, \ldots, B$ to the same variable $s \in \mathcal{S}$ through a reversible function φ. For convenience, $\mathcal{S}$ is set to $[0, 1]$, and the mapping is a linear function:

$$t = \varphi(s) = s(t_N - t_1) + t_1, \; s = \frac{t - t_1}{t_N - t_1}. \tag{4}$$

Then the original ODE $\mathrm{d}\boldsymbol{h}/\mathrm{d}t = \boldsymbol{F}(\boldsymbol{h}(t), t)$ can convert the integral variable to s:

$$\tilde{\boldsymbol{h}}(s) = \boldsymbol{h}(s(t_N - t_1) + t_1), \tag{5}$$

$$\begin{aligned} \frac{\mathrm{d}\tilde{\boldsymbol{h}}(s)}{\mathrm{d}s} &= \frac{\mathrm{d}t}{\mathrm{d}s} \left. \frac{\mathrm{d}\boldsymbol{h}(t)}{\mathrm{d}t} \right|_{t=s(t_N-t_1)+t_1} \\ &= (t_N - t_1)\boldsymbol{F}(s(t_N - t_1) + t_1, \tilde{\boldsymbol{h}}(s)). \end{aligned} \tag{6}$$

However, this method is not applicable to the samples with nonlinearly changing timestamps. This is because the linear function only records the information of the interval endpoint $j = 1, N$, and for the timestamps $t_j, j = 2, \ldots, N-1$, φ cannot map them to the corresponding s, unless repeating reparameterization process for $N - 1$ times.

3 Method

3.1 Problem Statement

We define irregularly sampled time series prediction as the following supervised learning task. Given matrices for the sequence inputs $\boldsymbol{X}$ and predicting targets $\boldsymbol{Y}$:

$$\begin{aligned} \boldsymbol{X} &= [\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_N)] \in \mathbb{R}^{D_x \times N}, \\ \boldsymbol{Y} &= [\boldsymbol{y}(t'_1), \ldots, \boldsymbol{y}(t'_M)] \in \mathbb{R}^{D_y \times M}, \end{aligned} \tag{7}$$

and a loss function $\mathcal{L} : \mathbb{R}^{D_y \times M} \times \mathbb{R}^{D_y \times M} \to \mathbb{R}$. Find a function $F : \mathbb{R}^{D_x \times N} \to \mathbb{R}^{D_y \times M}$, called a model, that minimizes the expected loss:

$$\min \mathbb{E}\left(\mathcal{L}\left[\boldsymbol{Y}, \hat{\boldsymbol{Y}} = F(\boldsymbol{X})\right]\right). \tag{8}$$

3.2 Predictive Neural Dynamical Systems (NDS)

Unified Architecture. Based on the characteristics of Neural ODE models, we propose NDS (Fig. 1, upper) as a unified architecture for predictive continuous neural networks. First, for $t \in [t_1, t_N]$, the input signal flow $\boldsymbol{u}(t)$ is computed:

$$\boldsymbol{u}(t) = \boldsymbol{U}(t, \boldsymbol{X} \mid \boldsymbol{\theta}_{\boldsymbol{U}}), \tag{9}$$

where $\boldsymbol{\theta}_{\boldsymbol{U}}$ is the possible parameters.

Then, the model obtains the dynamic latent vector $\boldsymbol{h} \in \mathbb{R}^{D_h}$ at t_1 through an initialization method $\boldsymbol{F}_1$:

$$\boldsymbol{h}(t_1) = \boldsymbol{F}_1(t_1, \boldsymbol{u}(t_1) \mid \boldsymbol{\theta}_{\boldsymbol{F}_1}), \tag{10}$$

where $\boldsymbol{\theta}_{\boldsymbol{F}_1}$ denotes the possible network parameters. The dynamic trajectory of the latent variable is modeled by a network cell $\boldsymbol{F}$ with the parameters $\boldsymbol{\theta}_{\boldsymbol{F}}$ and gotten using a numerical solver:

$$\begin{aligned} \boldsymbol{h}(t) &= \boldsymbol{h}(t_1) + \int_{t_1}^{t} \boldsymbol{F}(\boldsymbol{u}(\tau), \boldsymbol{h}(\tau) \mid \boldsymbol{\theta}_{\boldsymbol{F}}) \, \mathrm{d}\tau \\ &= \mathrm{ODESolver}(\boldsymbol{F}, \boldsymbol{u}, \boldsymbol{h}(t_1), t_1, t), \end{aligned} \tag{11}$$

In the following models, the tsit5 numerical method is applied as ODESolver for each time step from t to $t + \Delta t$:

$$\boldsymbol{k}_1 = \Delta t \boldsymbol{F}(\boldsymbol{u}(t), \boldsymbol{h}(t) \mid \boldsymbol{\theta}_{\boldsymbol{F}}), \tag{12}$$

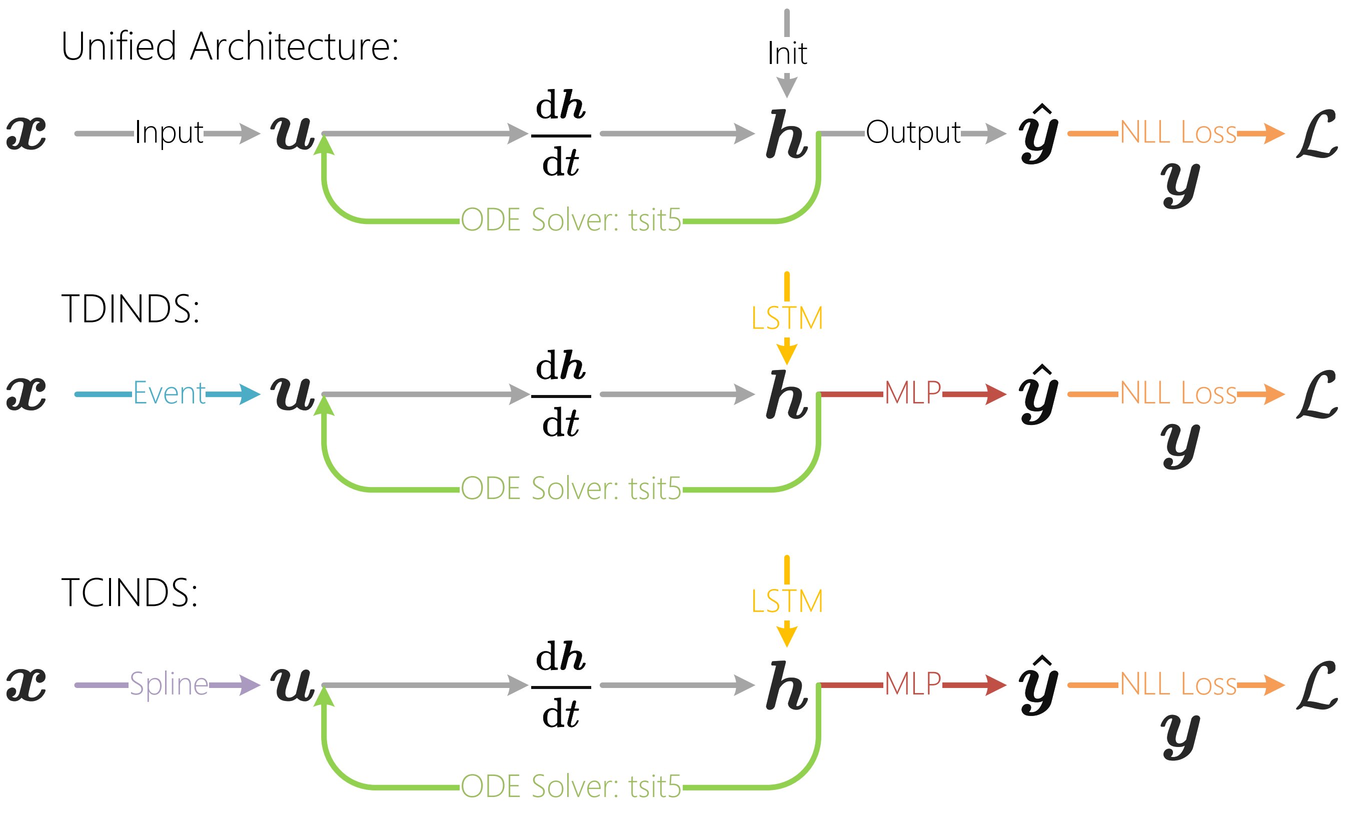

Fig. 1: The proposed unified architecture of predictive neural dynamical system, TDINDS and TCINDS.

$$\boldsymbol{k}_2 = \Delta t \boldsymbol{F}(\boldsymbol{u}(t+\Delta t/2), \boldsymbol{h}(t) + \boldsymbol{k}_1/2 \mid \boldsymbol{\theta_F}), \quad (13)$$

$$\boldsymbol{k}_3 = \Delta t \boldsymbol{F}(\boldsymbol{u}(t+\Delta t/2), \boldsymbol{h}(t) + \boldsymbol{k}_2/2 \mid \boldsymbol{\theta_F}), \quad (14)$$

$$\boldsymbol{k}_4 = \Delta t \boldsymbol{F}(\boldsymbol{u}(t+\Delta t), \boldsymbol{h}(t) + \boldsymbol{k}_3 \mid \boldsymbol{\theta_F}), \quad (15)$$

$$\boldsymbol{k}_5 = \boldsymbol{h}(t) + \begin{bmatrix} 1/6 \\ 1/3 \\ 1/3 \\ 1/6 \end{bmatrix}^T \begin{bmatrix} \boldsymbol{k}_1 \\ \boldsymbol{k}_2 \\ \boldsymbol{k}_3 \\ \boldsymbol{k}_4 \end{bmatrix}, \quad (16)$$

$$\boldsymbol{h}(t+\Delta t) = \boldsymbol{h}(t) + \begin{bmatrix} 5/48 \\ 1/3 \\ 23/48 \\ 1/3 \\ 5/48 \end{bmatrix}^T \begin{bmatrix} \boldsymbol{k}_1 \\ \boldsymbol{k}_2 \\ \boldsymbol{k}_3 \\ \boldsymbol{k}_4 \\ \boldsymbol{k}_5 \end{bmatrix}, \quad (17)$$

Finally, on the specified time step $t = t'_1, \dots, t'_M$, the corresponding prediction values are obtained by means of the output network $\boldsymbol{G}$:

$$\hat{\boldsymbol{y}}(t) = \boldsymbol{G}(\boldsymbol{h}(t) \mid \boldsymbol{\theta_G}), \quad (18)$$

where $\boldsymbol{\theta_G}$ is network parameters. The following NDS aim to minimize the negative log-likelihood (NLL) function as $\mathcal{L}$ in (8) by optimizing all network parameters $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} \sum_{\tau=t'_1}^{t'_M} -\log \mathbb{P}\left[\boldsymbol{y}(\tau) \mid \mathcal{N}(\hat{\boldsymbol{y}}(\tau), \sigma^2)\right], \quad (19)$$

where $\sigma = 0.02$ is the Gaussian standard deviation.

Discrete-Input NDS. The model is built on the assumption on a sequence of discrete events as the input signal, that is, when $t = t_j, j = 1, \dots, N$, the input $\boldsymbol{u}_j = \boldsymbol{x}(t_j) = \boldsymbol{x}_j$, otherwise $\boldsymbol{u} = \boldsymbol{0}$. For enabling the model to learn long-term dependencies, the long short-term memory (LSTM) module [7] is used to update the hidden variables and additional cell memory $\boldsymbol{c}$:

$$\boldsymbol{f}_{j+1} = \mathrm{sigmoid}(\boldsymbol{W}_f[\boldsymbol{h}_j; \boldsymbol{u}_{j+1}] + \boldsymbol{b}_f), \quad (20)$$

$$\boldsymbol{i}_{j+1} = \mathrm{sigmoid}(\boldsymbol{W}_i[\boldsymbol{h}_j; \boldsymbol{u}_{j+1}] + \boldsymbol{b}_i), \quad (21)$$

$$\boldsymbol{o}_{j+1} = \mathrm{sigmoid}(\boldsymbol{W}_o[\boldsymbol{h}_j; \boldsymbol{u}_{j+1}] + \boldsymbol{b}_o), \quad (22)$$

$$\bar{\boldsymbol{c}}_{j+1} = \tanh(\boldsymbol{W}_c[\boldsymbol{h}_j; \boldsymbol{u}_{j+1}] + \boldsymbol{b}_c), \quad (23)$$

$$\boldsymbol{c}_{j+1} = \boldsymbol{f}_{j+1} \circ \boldsymbol{c}_j + \boldsymbol{i}_{j+1} \circ \bar{\boldsymbol{c}}_{j+1}, \quad (24)$$

$$\boldsymbol{h}_{j+1} = \boldsymbol{o}_{j+1} \circ \tanh \boldsymbol{c}_{j+1}, \quad (25)$$

where $\circ$ is element-wise multiplication. $\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_o, \boldsymbol{b}_c \in \mathbb{R}^{D_h}, \boldsymbol{W}_f, \boldsymbol{W}_i, \boldsymbol{W}_o, \boldsymbol{W}_c \in \mathbb{R}^{D_h \times (D_h + D_x)}$ are parameters. This model utilizes the same LSTM cell (parameterized by $\boldsymbol{\theta}_{\mathrm{LSTM}}$) in initializing $\boldsymbol{F}_1$ and after solving the ODEs $\boldsymbol{F}$:

$$\begin{bmatrix} \boldsymbol{h}_1 \\ \boldsymbol{c}_1 \end{bmatrix} = \mathrm{LSTM}\left(\boldsymbol{u}_1, \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right), \quad (26)$$

$$\bar{\boldsymbol{h}}_{j+1} = \boldsymbol{h}_j + \int_{t_j}^{t_{j+1}} \boldsymbol{F}(\boldsymbol{h}(\tau) \mid \boldsymbol{\theta_F}) \, \mathrm{d}\tau, \quad (27)$$

$$\begin{bmatrix} \boldsymbol{h}_{j+1} \\ \boldsymbol{c}_{j+1} \end{bmatrix} = \mathrm{LSTM}\left(\boldsymbol{u}_{j+1}, \begin{bmatrix} \bar{\boldsymbol{h}}_{j+1} \\ \boldsymbol{c}_j \end{bmatrix}\right), \quad (28)$$

where $j = 1, \dots, N-1$. $\boldsymbol{F}$ is implemented by multilayer perceptron (MLP) cell with parameters $\boldsymbol{W}_1, \dots, \boldsymbol{W}_m$ and $\boldsymbol{b}_1, \dots, \boldsymbol{b}_m$:

$$\boldsymbol{F}(\boldsymbol{h} \mid \boldsymbol{\theta_F}) = \mathrm{MLP}(\boldsymbol{h}, m, \sigma)$$
$$= \mathrm{ReLU}(\underbrace{\boldsymbol{W}_m \sigma(\dots \sigma(\boldsymbol{W}_1}_{m \text{ layers}} \boldsymbol{h} + \boldsymbol{b}_1)) + \boldsymbol{b}_m), \quad (29)$$

where σ denotes the activation function and m is the number of layers. The selected values $\sigma = \text{sigmoid}$ and $m = 2$ can balance the computational cost and accuracy for the tasks in Section 4.

The output module enjoys another MLP cell with $m = 1$: $\boldsymbol{G}(\boldsymbol{h}(t) \mid \boldsymbol{\theta_G}) = \text{MLP}'(\boldsymbol{h}(t), 1, \text{sigmoid})$.

Continuous-Input NDS. The model assumes the inputs as continuous signals, and the sequence $\boldsymbol{X}$ is the sampling values for $t = t_1, \dots, t_N$. $\boldsymbol{u}(t)$ at timestamp $t \in [t_j, t_{j+1}), j = 1, 2, \dots, N-1$ is gotten by Hermite spline interpolation of the samples $\boldsymbol{x}$:

$$\boldsymbol{u}(t) = \begin{bmatrix} \boldsymbol{x}(t_j) \\ \boldsymbol{x}(t_{j+1}) \\ \delta_j \boldsymbol{z}_j \\ \delta_j \boldsymbol{z}_{j+1} \end{bmatrix}^T \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ p \\ p^2 \\ p^3 \end{bmatrix}, \tag{30}$$

where $\delta_j = t_{j+1} - t_j, p = (t - t_j)/\delta_j$ and

$$\boldsymbol{z}_j = \begin{cases} \boldsymbol{0}, & \text{if } j = 1, N, \\ \dfrac{\boldsymbol{x}(t_{j+1}) - \boldsymbol{x}(t_{j-1})}{2(t_{j+1} - t_{j-1})}, & \text{otherwise.} \end{cases} \tag{31}$$

The hidden variable initialization $\boldsymbol{F}_1$ is obtained using a LSTM network:

$$\begin{bmatrix} \boldsymbol{h}_1 \\ \boldsymbol{c}_1 \end{bmatrix} = \text{LSTM}\left(\boldsymbol{u}(t_1), \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}\right). \tag{32}$$

The ODEs are modeled utilizing a LSTM network. After solving them up to the prediction time $t \in t'_1, \dots, t'_M$, the outputs are obtained using an MLP module designed similarly to the discrete-input NDS:

$$\begin{bmatrix} \boldsymbol{h}(t) \\ \boldsymbol{c}(t) \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}_1 \\ \boldsymbol{c}_1 \end{bmatrix} + \int_{t_1}^{t} \text{LSTM}\left(\boldsymbol{u}(\tau), \begin{bmatrix} \boldsymbol{h}(\tau) \\ \boldsymbol{c}(\tau) \end{bmatrix}\right) \mathrm{d}\tau, \tag{33}$$

$$\hat{\boldsymbol{y}}(t) = \text{MLP}(\boldsymbol{h}(t), 1, \text{sigmoid}). \tag{34}$$

3.3 Tensorized Timeline Alignment

Aiming at the timeline mismatch problem in a mini-batch, tensorized timeline alignment is proposed to obtain the mapping $t = \varphi(s)$ for each sample. First, for any sample i, a reversible function φ_i must be determined to satisfy the following conditions in order to restore timestamp information t in the ODEs:

1) Monotonicity: Monotonicity of the function is a sufficient condition for reversibility, and it also ensures the advancing of time steps;
2) First-order continuity: Ensures that $\varphi'(s) = \mathrm{d}t/\mathrm{d}s$ always exists;
3) Interpolation: For each timestamp t_{ij} in the sample $i = 1, \dots, B$, there exists s_j such that $t_{ij} = \varphi_i(s_j)$, and all t_{ij} can be mapped to the same set of s via φ_i^{-1}.

Suppose the timestamp t_{ij} is stored in the tensor $\mathbf{T} \in \mathbb{R}^{B \times N \times 1}$. In theory, the final interpolation nodes can be arbitrarily set using a monotonic $\mathbf{S} \in \mathbb{R}^N$. However, since its elements will be used as divisors, it is best to set $\mathbf{S} = [0, 1, \dots, N-1]^T$ for numerical stability, where $s_j = j$. In this case, the tensor $\boldsymbol{\Delta}$ storing Δ_{ij} is simplified to the first-order difference of $\mathbf{T}$.

According to the optimal parameters proposed by [8] for such interpolation method, an initial value tensor $\mathbf{A}_0$ is defined as follows:

$$\mathbf{A}_0 = \begin{bmatrix} 1 & 5 & \cdots & 1 & 5 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 5 & \cdots & 1 & 5 \end{bmatrix}_{B \times N \times 1}, \tag{35}$$

Set the slice-wise ratio tensor $\boldsymbol{\Delta}_r = \boldsymbol{\Delta}[: -1]/\boldsymbol{\Delta}[1 :]$ and the iterative tensor:

$$\boldsymbol{\Delta}_{rj} = \begin{bmatrix} \mathbf{1}_{B \times j \times 1} & \boldsymbol{\Delta}_r[j/2 - 1] & \boldsymbol{\Delta}_r[j/2] & \cdots \end{bmatrix}, \tag{36}$$

where $[\cdot]$ denotes slicing operation on the second dimension of tensor. $j = 2, 4, \dots, P$. P is the biggest even number under $P < N$.

Then, calculate the element-wise product $\circ$ to obtain the tensor $\mathbf{A}$ that stores α_{ij} for $\lfloor (N-1)/2 \rfloor$ times:

$$\mathbf{A} = \mathbf{A}_0 \circ \boldsymbol{\Delta}_{r2} \circ \boldsymbol{\Delta}_{r4} \circ \cdots \circ \boldsymbol{\Delta}_{rP}, \tag{37}$$

where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x.

Since in $\mathbf{S}$, $s_j = j, s_{j+1} - s_j \equiv 1$, it follows that $\lambda_j = s - s_j = s - j$. Substitute $\alpha_{i0} = 1, \alpha_{i1} = 5$:

$$\varphi(s) = \frac{\mathbf{C}_0 \circ \mathbf{T}[j+1] - \mathbf{C}_0 \circ \mathbf{T}[j]}{\mathbf{C}_0 - \mathbf{C}_1}, \tag{38}$$

$$\varphi'(s) = \frac{5\boldsymbol{\Delta}[0]}{(\mathbf{C}_0 - \mathbf{C}_1)^2}, \tag{39}$$

where $\mathbf{C}_0 = (s - j)\mathbf{A}[j+1]$ and $\mathbf{C}_1 = (s - j - 1)\mathbf{A}[j]$.

Finally, use the obtained tensors $\mathbf{T}, \boldsymbol{\Delta}, \mathbf{A}$ to extract the corresponding components into (38) to obtain the mapping function φ and its derivative φ', and thus restore the time information in the corresponding ODE.

$$\frac{\mathrm{d}\tilde{\boldsymbol{h}}(s)}{\mathrm{d}s} = \varphi'(s)\boldsymbol{F}(\varphi(s), \tilde{\boldsymbol{h}}(s)). \tag{40}$$

Tensorized Timeline Alignment is utilized in the NDS models with discrete and continuous inputs, named TDINDS (Fig. 1, middle) and TCINDS respectively (Fig. 1, lower).

4 Experiments

4.1 Dataset

The GoogleStock dataset is used as the benchmark for model evaluation and comparison. It has four features: open prices, the highest prices, the lowest prices and close prices for the stock of Google company, daily from 2004 to 2019. The goal for time series prediction is set to predict each feature on the randomly selected 7 of the next 14 days, given past 30 days of data.

The data are cut by a sliding window and randomly recombined with a batch size B of 32. All mini-batches are then transformed into the range $[0, 1]$ by min-max normalization. 30% of the data (120 mini-batches) are partitioned to the test dataset. Besides, to verify the effectiveness of timeline alignment method for incomplete temporal information, the daily data are discarded randomly at the ratio of 10%, 20%, 30%, 40% and 50%, respectively. The model performance is evaluated with such irregularly sampled input sequences.

4.2 Experimental Designs

ODE-RNN [4] and Neural CDE [5] are introduced as the baseline models for evaluation. The implementation of TDINDS, TCINDS and them is based on PyTorch platform. For all models, the dimension D_h of latent variable vector $\boldsymbol{h}$ is set to 10. The adjustment of parameters $\boldsymbol{\theta}$ is carried out by Adamax optimizer, with a learning rate of 0.02. Xavier's initialization is used for these parameters [9]. The maximum number of training epochs is 200, with early stopping patience of 10 epochs.

Besides NLL as the loss function mentioned in Section 3.2, the mean square error (MSE) is also applied in model evaluation, which is defined as follows:

$$\text{MSE} = \frac{1}{BM} \sum_{i=1}^{B} \sum_{j=1}^{M} \left(y(t'_{ij}) - \hat{y}(t'_{ij})\right)^2 \tag{41}$$

where y is the true value of open price, the highest price, the lowest price or close price. $\hat{y}$ is the model prediction corresponding to it.

4.3 Evaluation Results

The results with regularly sampled input sequences. Fig. 2 shows the relative errors for 120 mini-batch predictions using ODE-RNN, Neural CDE, TDINDS and TCINDS, respectively. It can be found that the errors for TDINDS and TCINDS are restricted to the range $[-0.4, 0.4]$, while errors of the other models have exceeded this limit.

Table 1 lists the number of network parameters (Params), MSE, NLL and total training time for all models evaluated in this work. TDINDS and TCINDS reduce the MSE by 44% and 50% respectively, exhibiting high prediction performance. TCINDS achieves the smallest network scale and the best accuracy, while training time is slightly higher than TDINDS, the fastest trained model, due to the computation of the Hermite interpolation.

Table 1: Evaluation results for different models.

Models	Params	MSE	NLL	Training time
ODE-RNN	41.6k	0.2846	19.24	0.8h
Neural CDE	23.0k	0.2408	17.47	1.1h
TDINDS	16.9k	0.1355	12.89	**0.6h**
TCINDS	**11.4k**	**0.1212**	**11.03**	0.7h

As demonstrated in the evaluation results, tensorized timeline alignment could reduce the training time, by reducing time complexity of models from $\mathcal{O}(BN)$ to $\mathcal{O}(N)$ on the mini-batches of B sequences with length of N. After performing this operation, the accuracy of models is also improved, since the temporal information could be fully recovered by the mapping function φ. This experiment also proves that the proposed models have high fitting performance even if the number of parameters is limited.

The results with irregularly sampled input sequences. Fig. 3 shows the increase of MSE and NLL for all evaluated models, as the inputs are discarded under an ascending proportion. It can be found that all four models are resistant to irregularly sampled data, since the rise of MSE and NLL tends to be gentle. However, TDINDS and TCINDS can take advantage of the incomplete timestamps via tensorized timeline alignment, and are more robust to such data, as their error increase is lower than the baseline models. Since there is no need for TDINDS to recover the continuous input signals, getting rid of the estimation error in this process, its error increase is the lowest.

5 Conclusion

To reduce the computational burden of neural ordinary differential equations in irregularly sampled time series prediction, and make use of the timestamps contained in the samples of mini-batches, this paper proposes a unified architecture for predictive neural dynamical system, with two models implemented for discrete and continuous input sequences, respectively. In addition, a tensorized timeline alignment method is proposed to preserve temporal information of each training sample via reversible mappings. The evaluation results on the GoogleStock dataset demonstrate the superior performance of the two models. Robustness test shows that tensorized timeline alignment method is the key factor for enhancing irregular time series prediction ability.

References

[1] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018.

[2] Minhao Liu, Ailing Zeng, Zhijian Xu, Qiuxia Lai, and Qiang Xu. Time Series is a Special Sequence: Forecasting with Sample Convolution and Interaction. *ArXiv preprint*, abs/2106.09305, 2021.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[4] Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. *Latent ODEs for Irregularly-Sampled Time Series*, pages 1–13. Curran Associates Inc., Red Hook, NY, USA, 2019.

[5] Patrick Kidger, James Morrill, James Foster, and Terry J. Lyons. Neural controlled differential equations for irregular time series. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[6] Ricky T. Q. Chen, Brandon Amos, and Maximilian Nickel. Neural spatio-temporal point processes. In *International Conference on Learning Representations*, 2021.

[7] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[8] Ruzhong Tang, Qianjin Zhao, and Yuwu Zhang. Linear rational spline interpolation. *Journal of Anhui Institute of Architecture & Industry*, 18(1):76–78,82, 1 2010.

[9] 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*, pages 249–256, 2010.

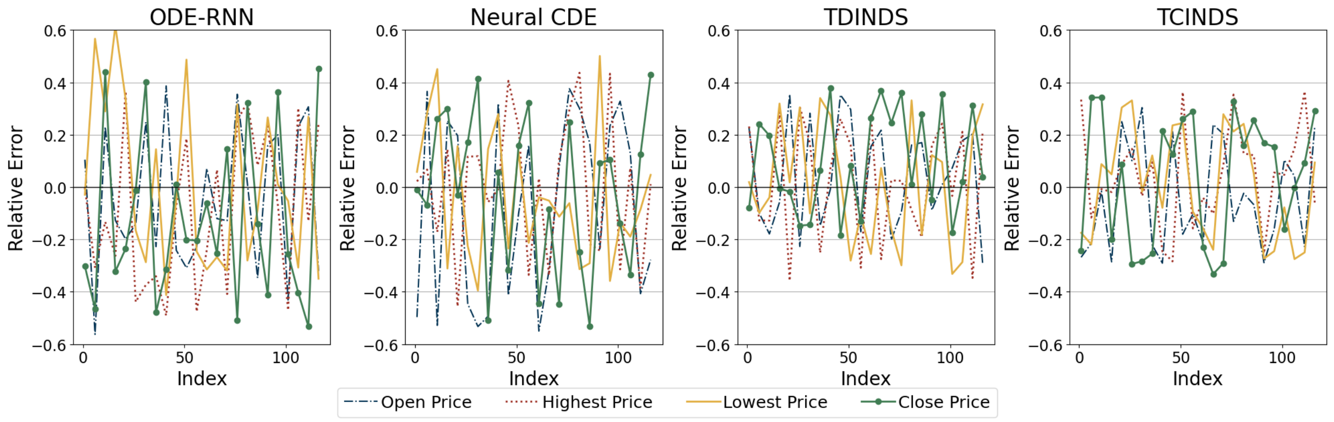

Fig. 2: The relative errors in GoogleStock dataset using ODE-RNN, Neural CDE, TDINDS and TCINDS.

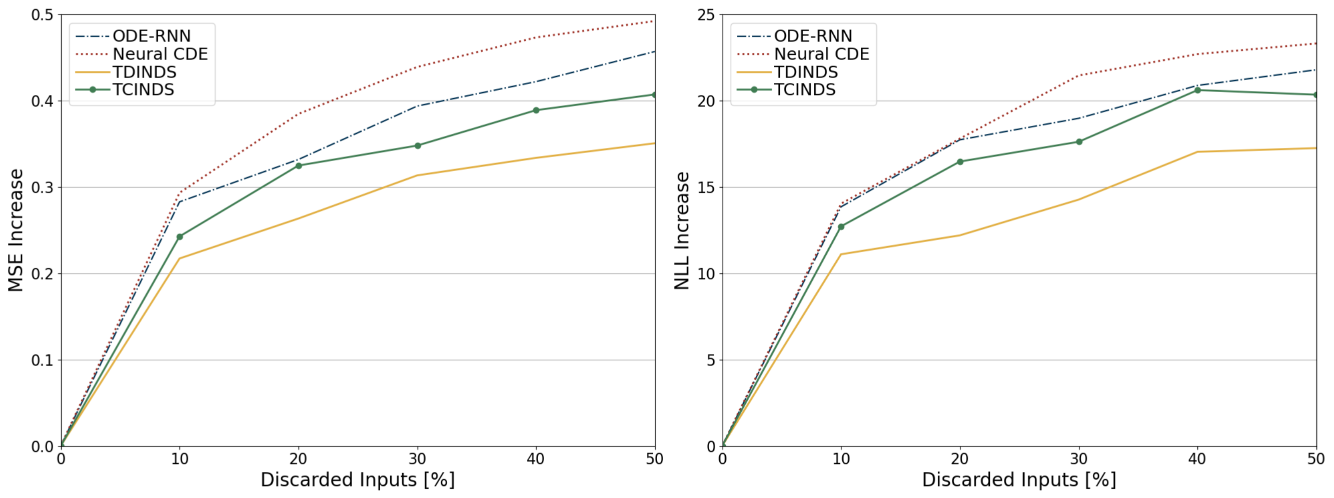

Fig. 3: The MSE and NLL increase with incomplete input sequences for each model.