Learnable Path in Neural Controlled Differential Equations

Sheo Yon Jhin[1*], **Minju Jo**[1*], **Seungji Kook**[1], **Noseong Park**[1]

[1] Yonsei University, Seoul, South Korea,
sheoyonj,alflsowl12,202132139,noseong@yonsei.ac.kr

Abstract

Neural controlled differential equations (NCDEs), which are *continuous* analogues to recurrent neural networks (RNNs), are a specialized model in (irregular) time-series processing. In comparison with similar models, e.g., neural ordinary differential equations (NODEs), the key distinctive characteristics of NCDEs are i) the adoption of the continuous path created by an interpolation algorithm from each raw discrete time-series sample and ii) the adoption of the Riemann–Stieltjes integral. It is the continuous path which makes NCDEs be analogues to continuous RNNs. However, NCDEs use existing interpolation algorithms to create the path, which is unclear whether they can create an optimal path. To this end, we present a method to generate another latent path (rather than relying on existing interpolation algorithms), which is identical to learning an appropriate interpolation method. We design an encoder-decoder module based on NCDEs and NODEs, and a special training method for it. Our method shows the best performance in both time-series classification and forecasting.

Introduction

Deep learning for time-series data is one of the most active research fields in machine learning since many real-world applications deal with time-series data. For instance, time-series forecasting is a long-standing research problem, ranging from stock price forecasting to climate and traffic forecasting (Reinsel 2003; Fu 2011; Yu, Yin, and Zhu 2018; Wu et al. 2019; Guo et al. 2019; Song et al. 2020; Huang et al. 2020; Bai et al. 2020; Chen, Segovia-Dominguez, and Gel 2021; Fang et al. 2021; Choi et al. 2022; Hwang et al. 2021). Time-series classification (Fawaz et al. 2019) and time-series anomaly detection (Zhang et al. 2019) are also popular. Recurrent neural networks (RNNs), such as LSTM (Hochreiter and Schmidhuber 1997), GRU (Chung et al. 2014) and so on, have been typically used for these purposes. Recently, however, researchers extended the basis of the time-series processing model design to differential equations (far beyond classical RNNs), e.g., neural ordinary differential equations (NODEs) and neural controlled differential equations (NCDEs). It is already well known that much scientific time-series data can be clearly described by differential equations (Chen et al. 2018; Rubanova, Chen, and Duvenaud 2019; Kidger et al. 2020). For instance, there exist various differential equation-based models for physical, social scientific, and financial phenomena and some of them received Nobel prizes, e.g., the Black–Scholes–Merton model (Black and Scholes 1973). In this regard, we consider that differential equation-based approaches are one of the most suitable strategies in designing time-series models, especially for real-world data. Neural controlled differential equations (NCDEs) (Kidger et al. 2020) are breakthrough concepts to interpret recurrent neural networks (RNNs) in a continuous manner. While the initial value determines the solution of differential equation in neural ordinary differential equations (NODEs) (Chen et al. 2018), NCDEs keep reading time-evolving values (as in RNNs) and their solutions are determined by the entire input. In this regard, NCDEs are a continuous analogue to RNNs and they show the state-of-the-art performance for many time-series datasets. We compare the solution $\mathbf{z}(T)$ of NODEs and NCDEs as follows:

1. For NODEs,

$$\mathbf{z}(T) = \mathbf{z}(0) + \int_0^T f(\mathbf{z}(t), t; \theta_f)dt; \tag{1}$$

2. For NCDEs,

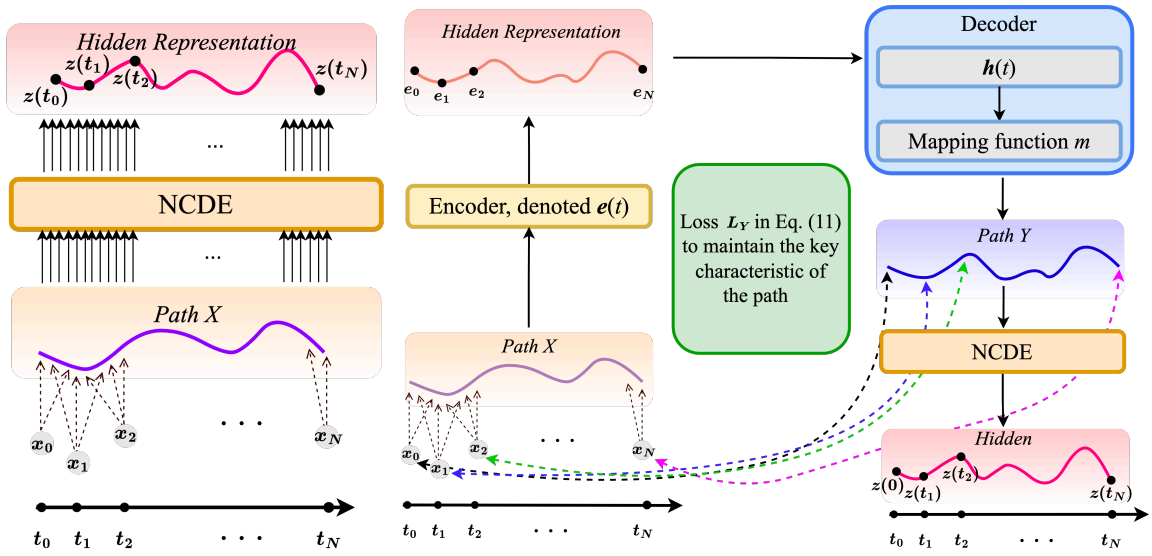
$$\mathbf{z}(T) = \mathbf{z}(0) + \int_0^T f(\mathbf{z}(t); \theta_f)dX(t) \tag{2}$$

$$= \mathbf{z}(0) + \int_0^T f(\mathbf{z}(t); \theta_f)\frac{dX(t)}{dt}dt, \tag{3}$$

where $X(t)$ is a continuous path created from a raw discrete time-series sample $\{(\mathbf{x}_i, t_i)\}_{i=0}^N$ by an interpolation algorithm, where t_i means the time-point of the observation $\mathbf{x}_i$, $t_0 = 0$, $t_N = T$ and $t_i < t_{i+1}$ (cf. Fig. 1(a)). Note that NCDEs keep reading the derivative of $X(t)$ over time, denoted $\dot{X}(t) \overset{\text{def}}{=} \frac{dX(t)}{dt}$, whereas NODEs do not. Given fixed θ_f (after training), $\mathbf{z}(0)$ solely determines the evolutionary trajectory from $\mathbf{z}(0)$ to $\mathbf{z}(T)$ in NODEs.

In NCDEs, the time-derivative of $X(t)$, $\frac{dX(t)}{dt}$, is considered to define the time-derivative of $\mathbf{z}(t)$, $\frac{d\mathbf{z}(t)}{dt}$, which means

[*]These authors contributed equally.



(a) The architecture of NCDE (b) The architecture of LEAP

Figure 1: (a) is the architecture of NCDE where the path X is created by interpolating $\{(\mathbf{x}_i, t_i)\}_{i=0}^{N}$. (b) is the proposed architecture of LEAP. Our proposed concept corresponds to learning an interpolation method to generate the path Y.

they keep reading values from $\frac{dX(t)}{dt}$. Thus, selecting the appropriate interpolation method for creating the continuous path X is crucial. The first work proposing the NCDE concept prefers the natural cubic spline algorithm for its suitable characteristic to be used in NCDEs, i.e., the path created by the natural cubic spline is twice differentiable and continuous. As reported in (Morrill et al. 2021), however, other interpolation algorithms can also be used. According to their experimental results, however, there does not exist a clear winning interpolation method that works always well although the natural cubic spline is, in general, a good choice.

To this end, we propose to learn an interpolation method optimized for a downstream task, i.e., **LEA**rnable **P**ath (LEAP). We insert an encoder-decoder module to generate a path Y, rather than relying on the path X created by an interpolation algorithm (cf. Fig. 1(b)). The additional module reads X to generate another fine-tuned path Y that will be used to define the hidden vector $\mathbf{z}$ over time t. One can consider that our method learns how to interpolate and generate the path Y.

We conducted time-series classification and forecasting experiments with four datasets and twelve baselines, which are all well-recognized standard benchmark environments. Our method shows the best performance in terms of various accuracy and error metrics. Our contributions can be summarized as follows:

1. We learn a path to be used to evolve $\mathbf{z}(t)$, which corresponds to learning an interpolation algorithm.

2. Our method outperforms existing baselines in all the cases in both time-series classification and forecasting.

Related Work and Preliminaries

We introduce our literature survey for recent deep learning work related to time-series processing. We also deliver base knowledge to understand our paper.

Neural Ordinary Differential Equations NODEs can process time-series data in a *continuous* manner, which means they can read and write values at any arbitrary time-point t by using the differential equation in Eq. (1).

$\mathbf{z}(t) \in \mathbb{R}^D$, where $t \in [0, T]$, means a D-dimensional vector — we use boldface to denote vectors and matrices. $\dot{\mathbf{z}}(t) \stackrel{\text{def}}{=} \frac{d\mathbf{z}(t)}{dt}$ is approximated by the neural network $f(\mathbf{z}(t), t; \theta_f)$, and we need to solve the initial value problem to derive the final value $\mathbf{z}(T)$ from the initial value $\mathbf{z}(0)$, which is basically an integral problem. To solve the problem, we typically rely on existing ODE solvers, such as the explicit Euler Method, the 4th order Runge–Kutta (RK4) method, the Dormand–Prince (DOPRI) method, and so forth. In particular, the explicit Euler method, denoted $\mathbf{z}(t+h) = \mathbf{z}(t) + hf(\mathbf{z}(t), t; \theta_f)$ where $h \in \mathbb{R}$ is a step-size parameter, is identical to the residual connection. Therefore, NODEs are considered as continuous analogues to residual networks.

There exist several popular time-series processing models based on NODEs, such as Latent-ODE, GRU-ODE, ACE-NODE, and so forth. Latent-ODE is an encoder-decoder architecture for processing time-series data. GRU-ODE showed that the GRU cell can be modeled by a differential equation. ACE-NODE is the first NODE-based model with an attention mechanism.

Neural Controlled Differential Equations NCDEs in Eq. (2) are technically more complicated than NODEs. NCDEs use the Riemann–Stieltjes integral, as shown in

Eq. (3), whereas NODEs use the Riemann integral. To solve Eq. (3), existing ODE solvers can also be used since $\dot{\mathbf{z}}(t) \stackrel{\text{def}}{=} \frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t);\theta_f)\frac{dX(t)}{dt}$ in NCDEs. Regardless of the integral problem type, existing ODE solvers can be used once $\dot{\mathbf{z}}(t)$ can be properly modeled and calculated. Additionally, NCDEs are considered as a *continuous* analogue to RNNs since they continuously read values $\dot{X}(t)$ over time.

In order to use NCDEs, however, we need to create a continuous path X from each raw discrete time-series sample, for which we typically use an interpolation algorithm. Among various methods, the natural cubic spline method is frequently used. As reported in (Morrill et al. 2021), the model accuracy of NCDEs is greatly influenced by how to interpolate discrete time-series data. To this end, we propose to let a neural network interpolate rather than relying on existing interpolation algorithms.

In (Jhin et al. 2021b), they proposed ANCDEs to insert an attention mechanism into NCDEs. Unlike our method, however, their goal is to pick useful information from the path X. Therefore, their method does not learn a new interpolation method whereas our proposed concept is equivalent to learning an interpolation method for a downstream task.

Proposed Method

We describe our proposed method that is to learn a path, which corresponds to learning an interpolation algorithm (rather than relying on existing algorithms). In other words, our proposed model generates another latent path Y from the path X for a downstream task.

Overall Design Fig. 1(b) shows the detailed design of our method, LEAP. The overall workflow is as follows:

1. The path X is created from a discrete time-series sample by an existing interpolation algorithm.
2. The encoder NCDE, the yellow box in Fig. 1(b), produces a hidden vector at time t, denoted $\mathbf{e}(t)$.
3. We use the hidden vector at t_N, i.e, $\mathbf{e}(t_N)$, to generate the hidden representation of the input time-series.
4. From the hidden representation, the NODE-based decoder, the blue box in Fig. 1(b), produces another latent path Y.
5. There is a loss to maintain the key characteristics of the path X while generating the path Y (cf. Eq. (11)). Therefore, the path Y is not simply a latent path but a latent path fine-tuned from X.
6. After that, there is one more NCDE based on Y for a downstream task, the orange box in Fig. 1(b).

We describe each part in detail, followed by a theoretical result that training the proposed model is well-posed.

Encoder-Decoder Module We first introduce our formulation to define the proposed LEAP. The entire module can be written, when we adopt the proposed encoder-decoder strategy to learn another path Y, as follows:

$$\mathbf{z}(T) = \mathbf{z}(0) + \int_0^T g(\mathbf{z}(t);\theta_g)\frac{dY(t)}{dt}dt, \tag{4}$$

$$Y(t) = m(\mathbf{h}(t);\theta_m), \tag{5}$$

$$\mathbf{h}(T) = \mathbf{h}(0) + \int_0^T f(\mathbf{h}(t),t;\theta_f)dt, \tag{6}$$

$$\mathbf{e}(T) = \mathbf{e}(0) + \int_0^T k(\mathbf{e}(t);\theta_k)\frac{dX(t)}{dt}dt, \tag{7}$$

where $\mathbf{h}(0) = \phi_{\mathbf{h}}(\mathbf{e}(t_N);\theta_{\phi_{\mathbf{h}}})$ for a raw discrete time-series sample $\{(\mathbf{x}_i,t_i)\}_{i=0}^N$, Y is a path created from X, $\mathbf{z}$ is controlled by Y, and $\phi_{\mathbf{h}}$ is a fully-connected layer-based transformation. As mentioned earlier, X is created by the natural cubic spline algorithm from raw discrete time-series observations. Even though the natural cubic spline algorithm is able to produce a path suitable for NCDEs, it is hard to say that the algorithm is always the best option (Morrill et al. 2021). Thus, we propose to learn another path Y which is optimized for a downstream machine learning task. Then, Y can be considered as a fine-tuned path from X for a downstream machine learning task. We also note that $\dim(Y(t)) = \dim(X(t))$. Since $\dot{Y}(t) \stackrel{\text{def}}{=} \frac{dY(t)}{dt} = \frac{dY(t)}{d\mathbf{h}(t)}\frac{d\mathbf{h}(t)}{dt}$ by the chain rule, Eq. (4) can be rewritten as follows:

$$\mathbf{z}(T) = \mathbf{z}(0) + \int_0^T g(\mathbf{z}(t);\theta_g)\frac{dY(t)}{d\mathbf{h}(t)}f(\mathbf{h}(t),t;\theta_f)dt, \tag{8}$$

where $\frac{dY(t)}{d\mathbf{h}(t)}$ is defined by the mapping function m and can be easily calculated by the automatic differentiation method. For instance, $\frac{dY(t)}{d\mathbf{h}(t)}$ will be simply $\mathbf{W}$ if $m(\mathbf{h}(t);\theta_m) = \mathbf{W}\mathbf{h}(t)$, i.e., m is a zero-biased fully connected layer where $\theta_m = \mathbf{W}$.

Therefore, our proposed concept can be rather succinctly described by Eq. (8). However, the key part in our method lies in training θ_f and θ_m to generate $\mathbf{h}(t)$ and $Y(t)$, i.e, the decoder part. We want that Y i) shares important characteristics with X to ensure the theoretical correctness of the proposed method and ii) at the same time produces a better path for a downstream machine learning task than X. To this end, we define the following augmented NCDE:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{z}(t)\\ \mathbf{h}(t)\end{bmatrix} = \begin{bmatrix} g(\mathbf{z}(t);\theta_g)\frac{d\mathbf{Y}(t)}{d\mathbf{h}(t)}f(\mathbf{h}(t),t;\theta_f)\\ f(\mathbf{h}(t),t;\theta_f)\end{bmatrix}, \tag{9}$$

where the initial values are defined as follows:

$$\begin{bmatrix}\mathbf{z}(0)\\ \mathbf{h}(0)\end{bmatrix} = \begin{bmatrix}\phi_{\mathbf{z}}(X(0));\theta_{\phi_{\mathbf{z}}})\\ \phi_{\mathbf{h}}(\mathbf{e}(t_N));\theta_{\phi_{\mathbf{h}}})\end{bmatrix}, \tag{10}$$

where $\phi_{\mathbf{z}}$ and $\phi_{\mathbf{h}}$ are fully-connected layer-based trainable transformation functions to generate the initial values.

In addition, let $\mathbf{z}(T)$ be the last hidden vector. We have an output layer with a typical construction based on $\mathbf{z}(T)$. The output layer is same as that in the original NCDE model. For forecasting (regression), we use a fully-connected layer whose output size is the same as the size of prediction.

Algorithm 1: How to train LEAP

Input: Training data D_{train}, Validating data D_{val}, Maximum iteration numbers max_iter

Initialize θ_f, θ_g, θ_k, θ_m, and other parameters, denoted θ_{others}, if any, e.g., the parameters of the output layer;
$i \leftarrow 0$;
while $i < max\_iter$ **do**
  Train θ_f, θ_g, θ_k, θ_m, and θ_{others} using the loss L;
  Validate and update the best parameters, θ_f^*, θ_g^*, θ_k^*, θ_m^*, and θ_{others}^*, with D_{val};
  $i \leftarrow i + 1$;
return θ_f^*, θ_g^*, θ_k^*, θ_m^*, and θ_{others}^*;

How To Train We use the following MSE and log-density loss definitions given training data D_{train} to define L_Y:

$$L_Y \stackrel{\text{def}}{=} \frac{\sum_{j=1}^{M}\sum_{i=1}^{N} \alpha \|Y(t_i^{(j)}) - \mathbf{x}_i^{(j)}\|_2^2 - \beta \log p(\hat{\mathbf{h}}(t_i^{(j)}))}{MN}, \tag{11}$$

where $M = |D_{train}|$ is the size of training data. The final loss L is the sum of L_Y and a task specific loss L_{task}, e.g., the cross-entropy loss for time-series classification or the mean squared error (MSE) loss for time-series forecasting. α and β are the coefficients of the two terms in Eq. (11). Algorithm 1 shows our training algorithm.

Given the j-th time-series sample in our training data, denoted $\{(\mathbf{x}_i^{(j)}, t_i^{(j)})\}_{i=0}^{N}$, $Y(t_i^{(j)}) = \mathbf{x}_i^{(j)}$ is preferred as in the natural cubic spline. We inject this by using the MSE loss term of L_Y in Eq. (11). In addition, we adopt the Hutchinson's unbiased estimator[1] to measure the log-density $\log p(\hat{\mathbf{h}}(t_i^{(j)}))$, where $Y(t_i^{(j)}) = m(\hat{\mathbf{h}}(t_i^{(j)}); \theta_m)$ and $\hat{\mathbf{h}}(t_i^{(j)}) = \mathbf{x}_{i-1}^{(j)} + \int_{t_{i-1}^{(j)}}^{t_i^{(j)}} f(\mathbf{h}(t); \theta_f) dt$ as in Eqs. (5) and (6). We use $\hat{\mathbf{h}}$, which is created directly from the real observation $\mathbf{x}_{i-1}^{(j)}$, to distinguish it from $\mathbf{h}$. The Hutchinson's estimator can be defined as follows in our case — refer to Appendix for the detailed explanation on the Hutchinson's estimator (**?**):

$$\log p(\hat{\mathbf{h}}(t_i^{(j)})) = \log p(\mathbf{x}_{i-1}^{(j)}) - \mathbb{E}_{p(\epsilon)}\Big[\int_{t_{i-1}^{(j)}}^{t_i^{(j)}} \epsilon^\intercal \frac{\partial f}{\partial \mathbf{h}(t)} \epsilon dt\Big], \tag{12}$$

where $p(\epsilon)$ is a standard Gaussian or Rademacher distribution (Hutchinson 1990). The time complexity to calculate the Hutchinson's estimator is slightly larger than that of evaluating f since the vector-Jacobian product $\epsilon^\intercal \frac{\partial f}{\partial \mathbf{h}(t)}$ has the same cost as that of evaluating f using the reverse-mode automatic differentiation. Since $\log p(\mathbf{x}_{i-1}^{(j)})$ is a constant, minimizing the negative log-density is the same as minimizing $\mathbb{E}_{p(\epsilon)}\Big[\int_{t_{i-1}^{(j)}}^{t_i^{(j)}} \epsilon^\intercal \frac{\partial f}{\partial \mathbf{h}(t)} \epsilon dt\Big]$ only.

In order to perform the density estimation via the change of variable theorem, we need invertible layers. Unfortunately, it is not theoretically guaranteed that NCDEs are invertible. However, NODEs are always invertible (more specifically, homeomorphic). Therefore, we do not perform the density estimation with NCDEs.

Rationale Behind Our Loss Definition For linear regression, it is known that the MSE training can achieve the maximum likelihood estimator (MLE) of model parameters. In general, however, the MSE training does not exactly return the MLE of parameters. Therefore, we mix the MLE training and the explicit likelihood training to yield the best outcome. The role of the MSE loss is to make $Y(t_i^{(j)})$ and $\mathbf{x}_i^{(j)}$ as close as possible and then, the negative log-density training enhances its likelihood since $Y(t_i^{(j)})$ is generated from $\hat{\mathbf{h}}(t_i^{(j)})$.

Well-posedness The well-posedness[2] of NODEs and NCDEs was already proved in (Lyons, Caruana, and Lévy 2004, Theorem 1.3) under the mild condition of the Lipschitz continuity. We show that our NODE/NCDE layers are also well-posed problems. Almost all activations, such as ReLU, Tanh, and Sigmoid, 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. (Chen et al. 2018) Therefore, the Lipschitz continuity of f, g, k can be fulfilled in our case. This makes our training problem well-posed. As a result, our training algorithm solves a well-posed problem so its training process is stable in practice.

Experiments

In this section, we describe our experimental environments and results. We conduct experiments with time-series classification and forecasting. Our software and hardware environments are as follows: UBUNTU 18.04 LTS, PYTHON 3.7.6, NUMPY 1.20.3, SCIPY 1.7, MATPLOTLIB 3.3.1, CUDA 11.0, and NVIDIA Driver 417.22, i9 CPU, and NVIDIA RTX TITAN. We repeat the training and testing procedures with five different random seeds and report their mean and standard deviation of evaluation metrics.

Experimental Environments

Hyperparameters We list all the hyperparameter settings in Appendix.

Baselines For time series forecasting and classification experiments, we compare our method with the following methods.

[1] Since NODEs are continuous and bijective, i.e., invertible, the change of variable theorem teaches us how to calculate the log-density — as a matter of fact, many other invertible neural networks, such as Glow, RealNVP, ans so on, rely on the same theory for their explicit likelihood training. However, this requires non-trivial computation. The Hutchinson's statistical method can reduce the complexity of measuring the log-density. On the top of that, better way to measure the log-density for NODEs than the Hutchinson's estimator is not known for now (Grathwohl et al. 2019).

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

1. RNN, LSTM (Hochreiter and Schmidhuber 1997), and GRU (Chung et al. 2014) are all recurrent neural network-based models that can process sequential data. LSTM is designed to learn long-term dependencies and GRU uses gating mechanisms to control the flow of information.
2. GRU-Δt is a GRU that additionally takes as input the time difference between observations. GRU-D (Che et al. 2016) is a modified version of GRU-Δt with learnable exponential decay between observations.
3. GRU-ODE (Brouwer et al. 2019; Jordan, Sokol, and Park 2019) is a NODE similar to GRU. This model is a continuous counterpart of GRU.
4. ODE-RNN (Rubanova, Chen, and Duvenaud 2019) is an extension of GRU-Δt to NODE.
5. Latent-ODE (Rubanova, Chen, and Duvenaud 2019) is a suitable model for time-series in which the latent state follows NODE. In this work, we use the recognition network of the existing Latent-ODE model as ODE-RNN.
6. Augmented-ODE is to augment the ODE state of Latent-ODE with the method proposed in (Dupont, Doucet, and Teh 2019).
7. ACE-NODE (Jhin et al. 2021a) is the state-of-the-art attention-based NODE model which has dual co-evolving NODEs.
8. NCDE (Kidger et al. 2020) is solved using controlled differential equations, which are well-understood mathematics.
9. ANCDE (Jhin et al. 2021b) is to insert an attention mechanism into NCDEs.

Time Series Classification Experimental Results

We introduce our experimental results for time-series classification with the following three datasets. Character trajectories, Speech Commands, and PhysioNet Sepsis are the datasets collected from real-world applications. Evaluating the performance of our model with these datasets proves the competence of our model in various fields. We use the accuracy for balanced datasets and AUROC for imbalanced datasets. (Rubanova, Chen, and Duvenaud 2019; Kidger et al. 2020).

Character Trajectories The Character Trajectories dataset from the UEA time-series classification archive (Bagnall et al. 2018) consists of values on the x-axis, y-axis and pen tip force of Latin alphabets as features. This dataset was collected while using a tablet with a sampling frequency of 200Hz, and there are 2,858 time-series character samples in total. The length of each data sample was truncated to 182 and each data has 3 dimensional vectors, i.e., x, y and pen tip force, and the length of each vector is 182. We used these features to classify alphabets into 20 classes ('a', 'b', 'c', 'd', 'e', 'g', 'h', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 'u', 'v', 'w', 'y', 'z'), while other 6 letters were excluded from this task.

Model	Test Accuracy		
	30% dropped	50% dropped	70% dropped
GRU-Δt	0.941 ± 0.018	0.928 ± 0.021	0.918 ± 0.017
GRU-D	0.938 ± 0.020	0.910 ± 0.048	0.916 ± 0.021
GRU-ODE	0.894 ± 0.012	0.886 ± 0.039	0.891 ± 0.029
ODE-RNN	0.946 ± 0.007	0.954 ± 0.003	0.949 ± 0.004
Latent-ODE	0.881 ± 0.021	0.878 ± 0.021	0.879 ± 0.048
Augmented-ODE	0.972 ± 0.012	0.948 ± 0.022	0.929 ± 0.023
ACE-NODE	0.881 ± 0.036	0.879 ± 0.025	0.913 ± 0.028
NCDE	0.988 ± 0.004	0.988 ± 0.002	0.985 ± 0.005
ANCDE	0.988 ± 0.001	0.986 ± 0.002	0.986 ± 0.004
LEAP	**0.992 ± 0.001**	**0.993 ± 0.002**	**0.991 ± 0.003**

Table 1: Accuracy (mean ± std, computed across five runs) on Irregular Character Trajectories

Model	Test Accuracy	Memory Usage (MB)
RNN	0.197 ± 0.006	1,905
LSTM	0.684 ± 0.034	4,080
GRU	0.747 ± 0.050	4,609
GRU-Δt	0.453 ± 0.313	1,612
GRU-D	0.346 ± 0.286	1,717
GRU-ODE	0.487 ± 0.018	171.3
ODE-RNN	0.678 ± 0.276	1,472
Latent-ODE	0.912 ± 0.006	2,668
Augmented-ODE	0.911 ± 0.008	2,626
ACE-NODE	0.911 ± 0.003	3,046
NCDE	0.898 ± 0.025	174.9
ANCDE	0.807 ± 0.075	179.8
LEAP	**0.922 ± 0.002**	391.1

Table 2: Accuracy on Speech Commands

Speech Commands The Speech Commands dataset is a one-second long audio data recorded with voice words such as 'left', 'right', 'cat', and 'dog' and noise heard in the background (Warden 2018). We use 'yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off', 'stop' to solve a balanced classification problem among all 35 labels, using a total of 34975 time-series samples. The length (resp. the dimensionality) of each time-series is 161 (resp. 20).

PhysioNet Sepsis Since Sepsis (Reyna et al. 2019; Reiter 2005) is a life-threatening illness leading many patients to death, early and correct diagnosis is important, which makes experiments on this dataset more meaningful. The PhysioNet 2019 challenge on sepsis prediction is originally a partially-observed dataset so that it fits to our irregular time-series classification experiments. Status of patients in ICU — both static and time-dependent features — is recorded in the dataset, and we only used 34 time-dependent features for our time-series classification. The goal for this classification is to predict the development of sepsis, which makes experiment binary classification. The dataset consists of 40,355 cases with variable time-series length, and about 90% of data is missing. Due to the data imbalance with only a sepsis positive rate of 5%, we evaluate our experiment using AUROC.

Experimental Results Table 1 summarizes the accuracy of Character Trajectories. In order to create challenging situations, we randomly selected 30%, 50%, and 70% of the

Model	Test AUROC	Memory Usage (MB)
GRU-Δt	0.861 ± 0.002	837
GRU-D	0.878 ± 0.019	889
GRU-ODE	0.852 ± 0.009	454
ODE-RNN	0.874 ± 0.012	696
Latent-ODE	0.782 ± 0.014	133
Augmented-ODE	0.842 ± 0.017	998
ACE-NODE	0.823 ± 0.012	194
NCDE	0.872 ± 0.003	244
ANCDE	0.886 ± 0.002	285
LEAP	**0.908 ± 0.004**	306

Table 3: AUROC on PhysioNet Sepsis

Model	Test MSE		
	30% dropped	50% dropped	70% dropped
GRU-Δt	0.186 ± 0.036	0.189 ± 0.015	0.187 ± 0.018
GRU-D	0.417 ± 0.032	0.421 ± 0.039	0.438 ± 0.042
GRU-ODE	0.826 ± 0.015	0.821 ± 0.015	0.681 ± 0.014
ODE-RNN	0.242 ± 0.213	0.240 ± 0.110	0.240 ± 0.116
Latent-ODE	0.048 ± 0.001	0.043 ± 0.004	0.056 ± 0.001
Augmented-ODE	0.042 ± 0.004	0.048 ± 0.005	0.052 ± 0.003
ACE-NODE	0.047 ± 0.007	0.047 ± 0.005	0.048 ± 0.005
NCDE	0.028 ± 0.000	0.029 ± 0.001	0.031 ± 0.004
ANCDE	0.035 ± 0.002	0.031 ± 0.003	0.033 ± 0.003
LEAP	**0.022 ± 0.001**	**0.022 ± 0.002**	**0.022 ± 0.001**

Table 4: MSE on Irregular MuJoCo

values from each sequence. Therefore, this is basically an irregular time-series classification, and many baselines show reasonable scores. The three GRU-based models are specialized in processing irregular time-series and outperform some other ODE-based models. However, CDE-based models, including LEAP, show the highest scores. Among them, LEAP is clearly the best. Our method maintains an accuracy larger than 0.99 across all the dropping settings.

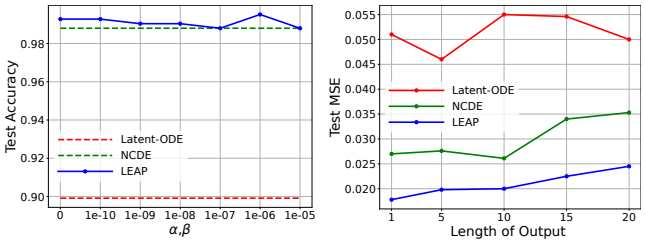
For the Speech Commands dataset, we summarize the results in Table 2. As summarized, all RNN/LSTM/GRU-based models are inferior to other differential equation-based models. We consider that this is because of the dataset characteristic. This dataset contains many audio signal samples and it is obvious that those physical phenomena can be well modeled as differential equations. Among many differential equation-based models, the two NCDE-based models, NCDE and LEAP, show the reasonable performance. However, LEAP significantly outperforms all others including NCDE. One more point is that our method requires much smaller GPU memory in comparison with many other baselines.

The time-series classification with PhysioNet Sepsis in Table 3 is one of the most widely used benchmark experiments. Our method, LEAP, shows the best AUROC and its GPU memory requirement is smaller than many other baselines. For this dataset, all CDE-based models show the highest performances.

Time Series Forecasting Experimental Results

We introduce our experimental results for time-series forecasting with the following dataset. We pick 1 dataset, MuJoCo, to evaluate the model's forecasting performance. Since MuJoCo is physics engine simulation data, using this data to predict future trajectories is an important study for mechanics and natural sciences. In addition, we use the MSE for our main metric, which is a metric generally used in time-series forecasting (Rubanova, Chen, and Duvenaud 2019; Brouwer et al. 2019; Kidger et al. 2020).

MuJoCo This dataset was generated from 10,000 simulations of the "Hopper" model using the DeepMind Control Suite. This physics engine supports research in robotics, machine learning, and other fields that require accurate simulation, such as biomechanics. The dataset is 14-dimensional, and 10,000 sequences of 100 regularly-sampled time points



(a) Character Trajectories (b) MuJoCo

Figure 2: Two sensitivity and ablation analyses. More figures are in Appendix.

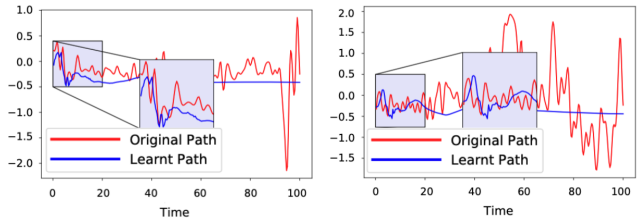
each (Tassa et al. 2018). The default training/testing horizon in MuJoCo is reading the first 50 observations in a sequence and forecasting the last 10 observations.

Experimental Results We drop random 30%, 50%, and 70% values, i.e., irregular time-series forecasting, to create challenging environments. In Table 4, our method, LEAP, clearly shows the best MSE for all dropping ratios. One outstanding point in our model is that the MSE is not greatly influenced by the dropping ratio but maintains its small error across all the dropping ratios.

Ablation and Sensitivity Studies

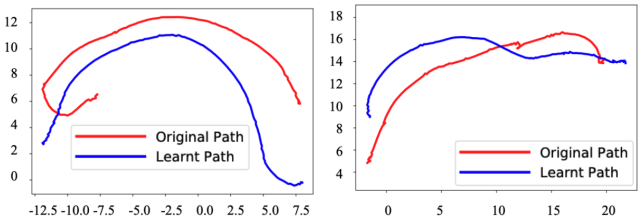
Sensitivity to α, β with Fixed Ratio 1. Fig. 2 (a) shows the sensitivity curve on Character Trajectories w.r.t. α, β, while setting α and β to same value, i.e. the ratio of α to β is 1, as in our original setting. With all the α, β settings, LEAP always shows better accuracy than those of the baselines, which shows the efficacy of our model. We also conduct experiments for the sensitivity to α, β with various ratios, and these results are in Appendix for space reasons.

Ablation on the Output Sequence Length We also compare our model with NCDE and Latent-ODE by varying the length of output (forecasting). After fixing the input length to 50, we variate the output length in $\{1, 5, 10, 15, 20\}$. As shown in Fig. 2(b), our proposed method consistently outperforms others. Moreover, MSE of LEAP's predicting 20-length sequence is lower than that of other models' predicting 1-length sequence. These results show that new latent paths made by LEAP skillfully represent the whole stream



(a) (b)

Figure 3: Examples of the original path X and learnt path Y in Speech Commands.



(a) (b)

Figure 4: Examples of the original path X and learnt path Y in MuJoCo using UMAP. More figures are in Appendix.

of data and capture parts which should be emphasized regardless of output sequence length.

Statistical Significance We test the statistical significance on MuJoCo using the paired t-test with a significance threshold of 0.01. Its null hypothesis and alternative hypothesis are set as follows:

$$H_0 : \mathcal{E}_{ours} \geq \mathcal{E}_{base}, \quad H_1 : \mathcal{E}_{ours} < \mathcal{E}_{base}, \tag{13}$$

where $\mathcal{E}$ is error between predicted values and true values measured in the L^2 vector norm. We do paired t-tests with every baseline on Table 4, and p-values are under 0.01 in every case, confirming our alternative hypothesis with high statistical significance, and rejecting the null hypothesis. The results of the paired t-test against GRU-Δt and NCDE are in Appendix, showing extremely low p-values. We report for each forecasting horizon — recall that we predict next 10 values in MuJoCo.

Visualization

We compare the original path X and the learnt (or fine-tuned) path Y in Fig. 3. According to Fig. 3, we select an element of a time-series sample in Speech Commands and visualize the original path and the learnt path on the selected element (dimension). Our MLE training to match them successfully ensures that they are well align when $t < 50$ for Fig. 3 (a). However, the learnt path in blue can be considered a smoothed (fine-tuned) version of X in the visualization. One remarkable point is that the learnt path becomes flat when $t \geq 50$. As shown in Eq. 4, the flat Y curve makes $\frac{dY(t)}{dt} \approx 0$ and $\mathbf{z}(t)$ is not much changed after $t = 50$. In case of Fig. 3 (b), same pattern is shown at $t < 60$.

Model	Datasets			
	Character Trajectories		MuJoCo	
	Test Accuracy	Memory	Test MSE	Memory
RNN	0.311 ± 0.038	52.2	0.051 ± 0.001	409.1
LSTM	0.791 ± 0.083	48.6	0.064 ± 0.001	411.2
GRU	0.844 ± 0.089	54.8	0.053 ± 0.000	439.5
GRU-Δt	0.834 ± 0.078	16.5	0.142 ± 0.020	532.9
GRU-D	0.896 ± 0.030	17.8	0.471 ± 0.038	569.1
GRU-ODE	0.878 ± 0.051	1.51	0.821 ± 0.027	146.2
ODE-RNN	0.827 ± 0.048	15.5	0.234 ± 0.211	146.3
Latent-ODE	0.942 ± 0.008	181	0.049 ± 0.007	314.9
Augmented-ODE	0.970 ± 0.009	186	0.045 ± 0.004	286.1
ACE-NODE	0.891 ± 0.001	113	0.046 ± 0.003	4,226
NCDE	0.981 ± 0.002	1.38	0.028 ± 0.002	52.08
ANCDE	0.986 ± 0.007	2.02	0.029 ± 0.003	79.22
LEAP	**0.992 ± 0.001**	9.24	**0.022 ± 0.002**	144.1

Table 5: Regular time-series prediction

Fig. 4 shows other visualization in our datasets. For this, we use a different visualization method, UMAP (Sainburg, McInnes, and Gentner 2020). This method is a lower-dimensional projection algorithm and each time-series sample is projected onto a 2-dim space. As shown in Fig. 4, our learnt paths are quite similar to the original paths but with little variation. From those facts, we can see that LEAP has different degrees of learning paths depending on datasets for enhancing the task performance.

Regular Time-series As Character Trajectories and MuJoCo provide complete data without missing values, i.e., regular time-series, we conduct the tasks with the full information and their results are summarized in Table 5. In both datasets, our method shows the best performance. One more point is that our method's regular and irregular forecasting results are the same for MuJoCo, which proves the strength of our method for irregular time-series. On top of that, accuracy of Character Trajectories with 50% missing values is even higher than that of full informed data, while many other baselines — ANCDE, ACE-NODE, Augmented-ODE, and so on — show lower performances compared to score with 50% missing data.

Conclusions

How to interpolate the input discrete time-series and create a continuous path is an important topic in NCDEs. In this work, we presented a method to learn how to interpolate from data (rather than relying on existing interpolation algorithms). To this end, we designed an encoder-decoder architecture and its special training method. We conducted a diverse set of experiments based on four datasets and twelve baselines, ranging from irregular/regular classification to forecasting. Our method, LEAP, clearly outperforms existing methods in almost all cases.

Acknowledgements

Noseong Park is the corresponding author. This work was supported by the Yonsei University Research Fund of 2021, and the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2020-0-01361, Artificial Intelligence Graduate School Program (Yonsei University), and (No.2022-0-00857, Development of AI/data-based financial/economic digital twin platform,10%) and (No.2022-0-00113, Developing a Sustainable Collaborative Multi-modal Lifelong Learning Framework, 45%),(2022-0-01032, Development of Collective Collaboration Intelligence Framework for Internet of Autonomous Things, 45%).

References

Bagnall, A.; Dau, H. A.; Lines, J.; Flynn, M.; Large, J.; Bostrom, A.; Southam, P.; and Keogh, E. 2018. The UEA multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*.

Bai, L.; Yao, L.; Li, C.; Wang, X.; and Wang, C. 2020. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *NeurIPS*, volume 33, 17804–17815.

Black, F.; and Scholes, M. 1973. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81(3): 637–654.

Brouwer, E. D.; Simm, J.; Arany, A.; and Moreau, Y. 2019. GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. In *NeurIPS*.

Che, Z.; Purushotham, S.; Cho, K.; Sontag, D.; and Liu, Y. 2016. Recurrent Neural Networks for Multivariate Time Series with Missing Values. arXiv:1606.01865.

Chen, R. T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural Ordinary Differential Equations. In *NeurIPS*.

Chen, Y.; Segovia-Dominguez, I.; and Gel, Y. R. 2021. Z-GCNETs: Time Zigzags at Graph Convolutional Networks for Time Series Forecasting. *arXiv preprint arXiv:2105.04100*.

Choi, J.; Choi, H.; Hwang, J.; and Park, N. 2022. Graph Neural Controlled Differential Equations for Traffic Forecasting. In *AAAI*.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Dupont, E.; Doucet, A.; and Teh, Y. W. 2019. Augmented Neural ODEs. In *NeurIPS*.

Fang, Z.; Long, Q.; Song, G.; and Xie, K. 2021. Spatial-Temporal Graph ODE Networks for Traffic Flow Forecasting. *arXiv preprint arXiv:2106.12931*.

Fawaz, H. I.; Forestier, G.; Weber, J.; Idoumghar, L.; and Muller, P.-A. 2019. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4): 917–963.

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

Grathwohl, W.; Chen, R. T. Q.; Bettencourt, J.; Sutskever, I.; and Duvenaud, D. 2019. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. In *ICLR*.

Guo, S.; Lin, Y.; Feng, N.; Song, C.; and Wan, H. 2019. Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting. *AAAI*, 33(01): 922–929.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-term Memory. *Neural computation*, 9: 1735–80.

Huang, R.; Huang, C.; Liu, Y.; Dai, G.; and Kong, W. 2020. LSGCN: Long Short-Term Traffic Prediction with Graph Convolutional Networks. In *IJCAI*, 2355–2361.

Hutchinson, M. 1990. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2).

Hwang, J.; Choi, J.; Choi, H.; Lee, K.; Lee, D.; and Park, N. 2021. Climate Modeling with Neural Diffusion Equations. In *ICDM*, 230–239.

Jhin, S. Y.; Jo, M.; Kong, T.; Jeon, J.; and Park, N. 2021a. ACE-NODE: Attentive Co-Evolving Neural Ordinary Differential Equations. In *KDD*.

Jhin, S. Y.; Shin, H.; Hong, S.; Jo, M.; Park, S.; and Park, N. 2021b. Attentive Neural Controlled Differential Equations for Time-series Classification and Forecasting. In *ICDM*.

Jordan, I. D.; Sokol, P. A.; and Park, I. M. 2019. Gated recurrent units viewed through the lens of continuous time dynamical systems. arXiv:1906.01005.

Kidger, P.; Morrill, J.; Foster, J.; and Lyons, T. 2020. Neural Controlled Differential Equations for Irregular Time Series. In *NeurIPS*.

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

Morrill, J.; Kidger, P.; Yang, L.; and Lyons, T. 2021. Neural Controlled Differential Equations for Online Prediction Tasks. *arXiv preprint arXiv:2106.11028*.

Reinsel, G. C. 2003. *Elements of multivariate time series analysis*. Springer Science & Business Media.

Reiter, P. J. 2005. Using CART to Generate Partially Synthetic, Public Use Microdata. *Journal of Official Statistics*, 21: 441.

Reyna, M. A.; Josef, C.; Seyedi, S.; Jeter, R.; Shashikumar, S. P.; Brandon Westover, M.; Sharma, A.; Nemati, S.; and Clifford, G. D. 2019. Early Prediction of Sepsis from Clinical Data: the PhysioNet/Computing in Cardiology Challenge 2019. In *CinC*, Page 1–Page 4.

Rubanova, Y.; Chen, R. T. Q.; and Duvenaud, D. K. 2019. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *NeurIPS*.

Sainburg, T.; McInnes, L.; and Gentner, T. Q. 2020. Parametric UMAP: learning embeddings with deep neural networks for representation and semi-supervised learning. *ArXiv e-prints*.

Song, C.; Lin, Y.; Guo, S.; and Wan, H. 2020. Spatial-Temporal Synchronous Graph Convolutional Networks: A New Framework for Spatial-Temporal Network Data Forecasting. *AAAI*, 34(01): 914–921.

Tassa, Y.; Doron, Y.; Muldal, A.; Erez, T.; Li, Y.; de Las Casas, D.; Budden, D.; Abdolmaleki, A.; Merel, J.; Lefrancq, A.; Lillicrap, T. P.; and Riedmiller, M. A. 2018. DeepMind Control Suite. *CoRR*, abs/1801.00690.

Warden, P. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *CoRR*, abs/1804.03209.

Wu, Z.; Pan, S.; Long, G.; Jiang, J.; and Zhang, C. 2019. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In *IJCAI*, 1907–1913.

Yu, B.; Yin, H.; and Zhu, Z. 2018. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *IJCAI*, 3634–3640.

Zhang, C.; Song, D.; Chen, Y.; Feng, X.; Lumezanu, C.; Cheng, W.; Ni, J.; Zong, B.; Chen, H.; and Chawla, N. V. 2019. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *AAAI*, volume 33, 1409–1416.