Graph Neural Ordinary Differential Equations

Michael Poli[*1], **Stefano Massaroli**[*2], **Junyoung Park**[*1]
Atsushi Yamashita[2], **Hajime Asama**[2], **Jinkyoo Park**[1]
[1]Department of Industrial & Systems Engineering, KAIST, Daejeon, South Korea
[2]Department of Precision Engineering, The University of Tokyo, Tokyo, Japan
{poli_m, junyoung, jinkyoo.park}@kaist.ac.kr,
{massaroli, yamashita, asama}@robot.t.u-tokyo.ac.jp[*]

Abstract

We extend the framework of *graph neural networks* (GNN) to continuous time. *Graph neural ordinary differential equations* (GDEs) are introduced as the counterpart to GNNs where the input–output relationship is determined by a *continuum* of GNN layers. The GDE framework is shown to be compatible with the majority of commonly used GNN models with minimal modification to the original formulations. We evaluate the effectiveness of GDEs on both static as well as dynamic datasets. Results prove their general applicability in cases where the data is not generated by continuous time dynamical network processes.

1 Introduction

Introducing appropriate inductive biases on deep learning models is a well known approach to improving sample efficiency and generalization performance (Battaglia et al. 2018). Graph neural networks (GNNs) represent a general computational framework for imposing such inductive biases when the target problem structure can be encoded as a graph or in settings where prior knowledge about relationships among input entities can itself be described as a graph (Li, Chen, and Koltun 2018; Gasse et al. 2019; Sanchez-Gonzalez et al. 2018). GNNs have shown remarkable results in various application areas such as node classification (Atwood and Towsley 2016), graph classification (Yan, Xiong, and Lin 2018) and forecasting (Li et al. 2017; Park and Park 2019) as well as generative tasks (Li et al. 2018).

A different but equally important class of inductive biases is concerned with the class of systems from which the data is collected. Although deep learning has traditionally been a field dominated by discrete models, recent advances propose a treatment of neural networks as models equipped with a continuum of layers (Chen et al. 2018). This view allows a reformulation of the forward pass as the solution of the initial value problem of an ordinary differential equation (ODE). Such approaches allow direct modeling of ODEs and enhance the performance of neural networks

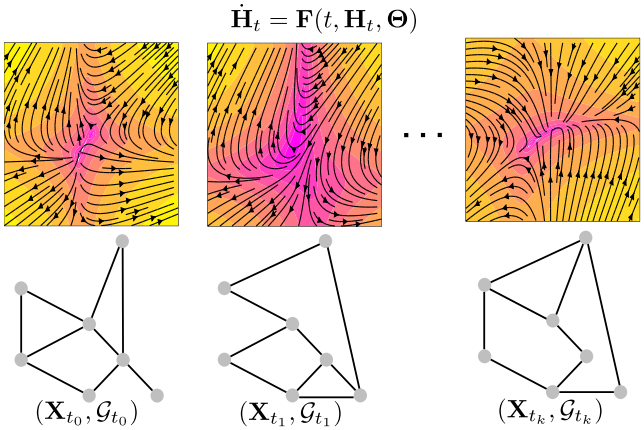

Figure 1: A *graph neural ordinary differential equation* (GDE) models vector fields defined on graph–structured data, both in cases when the structure is fixed or changes in time. This is achieved by equipping the model with a continuum of *graph neural network* (GNN) layers.

on tasks involving continuous time processes (Rubanova, Chen, and Duvenaud 2019). Our work is aimed at bridging the gap between geometric deep learning and continuous models. Graph neural ordinary differential equations (GDEs) cast common tasks on graph–structured data into a system–theoretic framework, as shown in Figure (1). GDEs provide flexibility due to their structure defined by a *continuum* of GNN layers and can therefore accommodate irregularly sampled sequential data. We show that GDEs offer improved performance in a traffic forecasting application due to their ability to track the underlying dynamics and adjust the prediction horizon according to timestamp information. In general, no assumptions on the data generating process are necessary in order for GDEs to be effective. Indeed, following recent work connecting different discretization schemes of ODEs (Lu et al. 2017) to previously known architectures such as FractalNets (Larsson, Maire, and Shakhnarovich 2016), we show that GDEs can equivalently be utilized as high–performance general purpose models. In the standard transductive semi–supervised node classification tasks on datasets Cora, Pubmed and Citeseer, GDEs are shown to be competitive with their state–of–

[*]These authors contributed equally to the work.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

the–art discrete counterpart, graph convolutional networks (GCNs) (Kipf and Welling 2016). Finally, we show that training GDEs with adaptive ODE solvers leads to deep GNN models without the need to specify the number of layers a–priori, sidestepping known depth–limitations of GNNs (Zhang 2019). We summarize our contributions as follows:

- We introduce *graph ordinary differential equation networks* (GDEs), continuous counterparts to graphical models. We show that the proposed framework includes most common GNN models and naturally extends to spatio-temporal and autoregressive models on graphs.
- We validate GDEs experimentally on a static semi–supervised node classification task on standard datasets Cora, Citeseer and Pubmed. GDEs are shown to be competitive with well known state–of–the–art models.
- We conduct an experiment on a forecasting task with severely undersampled data. We compare GDEs and an equivalent discrete GNN model, showing that GDEs offers improved performance in tasks involving dynamical systems thanks to their continuous nature.

2 Background

Notation Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{R}$ is the the one of reals. Scalars are indicated as lowercase letters, vectors as bold lowercase, matrices and tensors as bold uppercase and sets with calligraphic letters. Let $\mathcal{V}$ be a finite set with $|\,\mathcal{V}\,| = n$ whose element are called *nodes* and let $\mathcal{E}$ be a finite set of tuples of $\mathcal{V}$ elements. Its elements are called *edges* and are such that $\forall e_{ij} \in \mathcal{E},\ e_{ij} = (v_i, v_j)$ and $v_i, v_j \in \mathcal{V}$. A graph $\mathcal{G}$ is defined as the collection of nodes and edges, i.e. $\mathcal{G} := (\mathcal{V}, \mathcal{E})$. The *adjeciency* matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ of a graph is defined as

$$A_{ij} = \begin{cases} 1 & e_{ij} \in \mathcal{E} \\ 0 & e_{ij} \notin \mathcal{E} \end{cases}$$

If $\mathcal{G}$ is an *attributed graph*, the *feature vector* of each $v \in \mathcal{V}$ is $\mathbf{x}_v \in \mathbb{R}^d$. All the feature vectors are collected in a matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$. Note that often, the features of graphs exhibits temporal dependency, i.e. $\mathbf{X} := \mathbf{X}_t$.

Graph neural networks Current work on GNNs and ODEs is limited in scope and constrains the GNN architecture to the tracking of Hamiltonian functions (Sanchez-Gonzalez et al. 2019) or generative modeling (Deng et al. 2019). Our focus is developing general, continuous counterparts of the main variants of static GNNs: *graph convolutional networks* (GCNs) (Kipf and Welling 2016), diffusion graph convolution (DGC) (Atwood and Towsley 2016), *graph attention networks* (GATs) (Veličković et al. 2017) as well as recurrent models: *graph convolutional recurrent networks* (GCRNNs) (Cui et al. 2018) and *gated unit graph convolutional networks* (GCGRU) (Zhao, Chen, and Cho 2018).

Neural ordinary differential equations Continuous neural network architectures are built upon the observation that, for particular classes of discrete models such as ResNets (He et al. 2016), their inter–layer dynamics:

$$\mathbf{h}_{s+1} = \mathbf{h}_s + \mathbf{f}(\mathbf{h}_s, \boldsymbol{\theta}), \quad s \in \mathbb{N} \tag{1}$$

resemble Euler discretizations of an ordinary differential equation (ODE). The continuous counterpart of neural network layers with equal input and output dimensions can therefore be described by a first order ODE of the type:

$$\frac{d\mathbf{h}_s}{ds} = \mathbf{f}(s, \mathbf{h}_s, \boldsymbol{\theta}), \quad s \in \mathcal{S} \subset \mathbb{R} \tag{2}$$

It has been noted that the choice of discretization scheme of (2) can describe previously known discrete multi–step architectures (Lu et al. 2017). As a result, the *neural ordinary differential equation* (NODE) (Chen et al. 2018) framework is not limited to the modeling of differential equations and can guide discovery of novel general purpose models.

A motivating example Multi–agent systems permeate science in a variety of fields: from physics to robotics, game-theory, finance and molecular biology, among others. Based on the interplay between nonlinear dynamical systems and graphs, *dynamical network* theory has been developed as a widely applicable, classical approach to control and stabilization (Wang and Chen 2002; Li, Wang, and Chen 2004) of such systems.

Often, closed–form analytic formulations are not available and forecasting or decision making tasks have to rely on noisy, irregularly sampled observations. The primary purpose of GDEs is to offer a data–driven approach to the modeling of dynamical networks, particularly when the governing equations are highly nonlinear and therefore challenging to approach with classical or analytical methods.

3 Graph Neural Ordinary Differential Equations

Without any loss of generality, the inter–layer dynamics of a GNN node feature matrix can be represented in the form

$$\begin{cases} \mathbf{H}_{s+1} = \mathbf{H}_s + \mathbf{F}\left(s, \mathbf{H}_s, \boldsymbol{\Theta}_s\right) \\ \mathbf{H}_0 = \mathbf{X} \end{cases}, \quad s \in \mathbb{N}$$

where $\mathbf{F}$ is a matrix–valued nonlinear function conditioned on graph $\mathcal{G}$ and $\boldsymbol{\Theta}$ is the tensor of trainable parameters of the s-th layer. Note that the explicit dependence on s of the dynamics is justified in some graph architectures, such as diffusion graph convolutions (Atwood and Towsley 2016).

A *graph neural differential ordinary equation* (GDE) is defined as the following Cauchy problem:

$$\begin{cases} \dot{\mathbf{H}}_s = \mathbf{F}\left(s, \mathbf{H}_s, \boldsymbol{\Theta}\right) \\ \mathbf{H}_0 = \mathbf{X} \end{cases}, \quad s \in \mathcal{S} \subset \mathbb{R} \tag{3}$$

where $\mathbf{F} : \mathcal{S} \times \mathbb{R}^{n\times d} \times \mathbb{R}^p \to \mathbb{R}^{n\times d}$ is a depth–varying vector field defined on graph $\mathcal{G}$.

Well–posedness Let $\mathcal{S} := [0, 1]$. Under mild conditions on $\mathbf{F}$, namely Lipsichitz continuity with respect to $\mathbf{H}$ and uniform continuity with respect to s, for each initial condition (GDE input) $\mathbf{X}$, the ODE in (3) admits a unique solution $\mathbf{H}_s$ defined in the whole $\mathcal{S}$. Thus there is a mapping $\mathbf{\Psi}$ from $\mathbb{R}^{n\times d}$ to the space of absolutely continuous functions $\mathcal{S} \to \mathbb{R}^{n\times d}$ such that $\mathbf{H} := \mathbf{\Psi}(\mathbf{X})$ satisfies the ODE in (3). This implies the the output $\mathbf{Y}$ of the GDE satisfies

$$\mathbf{Y} = \mathbf{\Psi}(\mathbf{X})(1)$$

Symbolically, the output of the GDE is obtained by the following

$$\mathbf{Y} = \mathbf{X} + \int_{\mathcal{S}} \mathbf{F}(\tau, \mathbf{H}_\tau, \mathbf{\Theta}) d\tau$$

Integration domain We restrict the integration interval to $\mathcal{S} := [0, 1]$, given that any other integration time can be considered a rescaled version of $\mathcal{S}$. In application where $\mathcal{S}$ acquires a specific meaning (i.e forecasting with irregular timestamps) the integration domain can be appropriately tuned to evolve GDE dynamics between arrival times (Rubanova, Chen, and Duvenaud 2019) without assumptions on underlying vector field (Che et al. 2018).

GDE training GDEs can be trained with a variety of methods. Standard backpropagation through the computational graph, adjoint method for $\mathcal{O}(1)$ memory efficiency (Chen et al. 2018), or backpropagation through a relaxed spectral elements discretization (Quaglino et al. 2019). Numerical instability in the form of accumulating errors on the adjoint ODE during the backward pass of NODEs has been observed in (Gholami, Keutzer, and Biros 2019). A proposed solution is a hybrid checkpointing–adjoint scheme, where the adjoint trajectory is reset at predetermined points in order control the error dynamics.

Incorporating governing differential equation priors GDEs belong to the toolbox of scientific deep learning (Innes et al. 2019) along with Neural ODEs (Chen et al. 2018) and other continuous depth models. Scientific deep learning is concerned with merging prior, incomplete knowledge about governing equations with data-driven predictions. Within this framework GDEs can be extended to settings involving dynamical networks evolving according to different classes of differential equations, such as *stochastic differential equations* (SDEs):

$$\begin{cases} d\mathbf{H}_s = \mathbf{F}\left(s, \mathbf{H}_s\right) dt + \mathbf{G}\left(s, \mathbf{H}_s\right) d\mathbf{W}_t \\ \mathbf{H}_0 = \mathbf{X} \end{cases}, \quad s \in \mathcal{S} \qquad (4)$$

where $\mathbf{F}$ and $\mathbf{G}$ are GDEs that can be replaced by analytic terms when available and $\mathbf{W}$ is a standard multidimensional Wiener process. This extension enables a practical method to link dynamical network theory and deep learning with the objective of obtaining sample efficient, interpretable models.

3.1 Static Models

Graph convolution networks Based on graph spectral theory, *graph convolution network* (GCN) (Kipf and Welling 2016) layers are in the form:

$$\mathbf{H}_{s+1} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}_s\mathbf{W}_s)$$

where $\tilde{\mathbf{A}} := \mathbf{A} + \mathbb{I}_n$ and $\tilde{\mathbf{D}}$ is a diagonal matrix defined as $\tilde{\mathbf{D}}_{ii} := \sum_j \tilde{A}_{ij}$

In order to obtain the model in the continuous depth domain, a skip connection can be added as

$$\mathbf{H}_{s+1} = \mathbf{H}_s + \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}_s\mathbf{W}_s)$$

and the corresponding continuous counterpart becomes

$$\frac{d\mathbf{H}_s}{ds} = \mathbf{F}_{\texttt{GCN}}(\mathbf{H}, \mathbf{\Theta}) := \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}_s\mathbf{\Theta})$$

Diffusion graph convolution Consider a diffusion graph convolution (DGC) network (Atwood and Towsley 2016):

$$\mathbf{H}_{s+1} = \mathbf{H}_s + \sigma\left(\mathbf{P}^s\mathbf{X}\mathbf{W}_s\right) \qquad (5)$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function assumed to act component–wise, $\mathbf{P} \in \mathbb{R}^{n\times n}$ is a probability transition matrix, $\mathbf{P} := \mathbf{D}^{-1}\mathbf{A}$. The continuous counterpart of (5), DGCDEs, can be therefore derived as:

$$\frac{d\mathbf{H}}{ds} = \mathbf{F}_{\texttt{DGC}}(s, \mathbf{X}, \mathbf{\Theta}) := \sigma\left(\mathbf{P}^s\mathbf{X}\mathbf{\Theta}\right),$$

which consists in a depth–varying vector field constant with respect to the hidden node feature matrix.

Additional models and considerations We include additional derivation of continuous counterparts of common static GNN models such as *graph attention networks* (GAT) (Veličković et al. 2017) as supplementary material.

While the definition of GDE models is given with $\mathbf{F}$ made up by a single layer, in practice multi–layer architectures can also be used without any loss of generality. In these models, the vector field defined by $\mathbf{F}$ is computed by considering wider neighborhoods of each node.

3.2 Spatio–Temporal Continuous Graph Architectures

For settings involving a temporal component, the depth domain of GDEs coincides with the time domain $s \equiv t$ and can be adapted depending on the requirements. For example, given a time window Δt, the prediction performed by a GDE assumes the form:

$$\mathbf{H}_{t+\Delta t} = \mathbf{H}_t + \int_t^{t+\Delta t} \mathbf{F}\left(\tau, \mathbf{H}_\tau, \mathbf{\Theta}\right) d\tau$$

regardless of the specific GDE architecture employed. Here, GDEs represent a natural model class for autoregressive modeling of sequences of graphs $\{\mathcal{G}_t\}$ and directly fit into dynamical network theory. This line of reasoning naturally leads to an extension of classical spatio–temporal architectures in the form of *hybrid dynamical systems* (Van

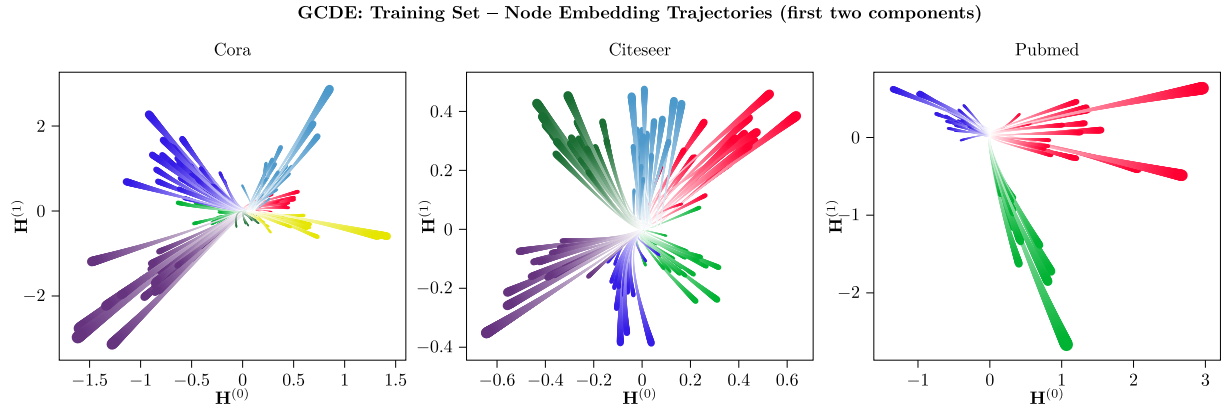

Figure 2: Node embedding trajectories defined by a forward pass of GCDE–dopri5 on Cora, Citeseer and Pubmed. Color differentiates between node labels.

Der Schaft and Schumacher 2000; Goebel, Sanfelice, and Teel 2009), i.e., systems characterized by interacting continuous and discrete –time dynamics. Let $(\mathcal{K}, >)$, $(\mathcal{T}, >)$ be linearly ordered sets; namely, $\mathcal{K} \subset \mathbb{N}$ and $\mathcal{T}$ is a set of time instants, $\mathcal{T} := \{t_k\}_{k\in\mathcal{K}}$. We suppose to be given a *state-graph data stream* which is a sequence in the form

$$\{(\mathbf{X}_t, \mathcal{G}_t)\}_{t\in\mathcal{T}}$$

Our aim is to build a continuous model predicting, at each $t_k \in \mathcal{T}$, the value of $\mathbf{X}_{t_{k+1}}$, given $(\mathbf{X}_t, \mathcal{G}_t)$. Let us also define a *hybrid time domain* as the set $\mathcal{I} := \bigcup_{k\in\mathcal{K}} ([t_k, t_{k+1}], k)$ and a *hybrid arc* on $\mathcal{I}$ as a function $\mathbf{\Phi}$ such that for each $k \in \mathcal{K}$, $t \mapsto \mathbf{\Phi}(t, k)$ is absolutely continuous in $\{t : (t, j) \in \operatorname{dom} \mathbf{\Phi}\}$.

The core idea is to have a GDE smoothly steering the latent node features between two time instants and then apply some discrete operator, resulting in a "jump" of $\mathbf{H}$ which is then processed by an output layer. Therefore, solutions of the proposed continuous spatio–temporal model are hybrid arcs.

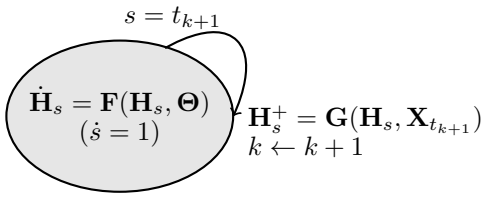

Figure 3: Schematic of autoregressive GDEs as hybrid automata.

Autoregressive GDEs The solution of a general autoregressive GDE model can be symbolically represented by:

$$\begin{cases} \dot{\mathbf{H}}_s & = \mathbf{F}(\mathbf{H}_s, \mathbf{\Theta}) \quad s \in [t_k, t_{k+1}] \\ \mathbf{H}_s^+ & = \mathbf{G}(\mathbf{H}_s, \mathbf{X}_{t_k}) \quad s = t_{k+1} \\ \mathbf{Y}_{t_{k+1}} & = \mathbf{K}(\mathbf{H}_s) \end{cases}, \quad k \in \mathcal{K} \quad (6)$$

Figure 4: Test loss and accuracy on the Cora experiments. The shaded area is the 95% confidence interval

where $\mathbf{F}, \mathbf{G}, \mathbf{K}$ are GNN–like operators or general neural network layers[1] and $\mathbf{H}^+$ represent the value of $\mathbf{H}$ after the discrete transition. The evolution of system (6) can be visualized by means of hybrid automata as shown in Fig. 3. Compared to standard recurrent models which are only equipped with discrete jumps, system (6) incorporates a continuous flow of latent node features $\mathbf{H}$ between jumps. This feature of autoregressive GDEs allows them to track dynamical systems from irregular observations.

Different combinations of $\mathbf{F}, \mathbf{G}, \mathbf{K}$ can yield continuous variants of most common spatio-temporal GNN models. It should be noted that the operators $\mathbf{F}, \mathbf{G}, \mathbf{K}$ can themselves have multi–layer structure.

Graph Differential Convolutional GRU We illustrate the generality of (6) by deriving the continuous version of GCGRUs (Zhao, Chen, and Cho 2018) as:

$$\begin{cases} \dot{\mathbf{H}}_s & = \mathbf{F}_{\texttt{GCN}}(\mathbf{H}_t) \quad s \in [t_k, t_{k+1}] \\ \mathbf{H}_s^+ & = \texttt{GCGRU}(\mathbf{H}_s, \mathbf{X}_{t_k}) \quad s = t_{k+1} \\ \mathbf{Y}_{t_{k+1}} & = \sigma(\mathbf{W}\mathbf{H}_s + \mathbf{b}) \end{cases}, \quad k \in \mathcal{K}$$

where $\mathbf{W}$ is a learnable weight matrix. The complete description of a GCGRU layer of computation is included as supplementary material.

4 Experiments

We evaluate GDEs on a transductive node classification task as well as a forecasting task. The code is available at https://github.com/Zymrael/gde.

4.1 Transductive Node Classification

Experimental setup The first task involves performing semi–supervised node classification on static graphs collected from baseline datasets Cora, Pubmed and Citeseer

[1]More formal definitions of the hybrid model in the form of *hybrid inclusions* can indeed be easily given. However, the technicalities involved are beyond the scope of this paper.

Model (depth)	Cora	Citeseer	Pubmed	Parameters
GCN(2)	$81.4 \pm 0.5\%$	$70.9 \pm 0.5\%$	$79.0 \pm 0.3\%$	184k/474k/64k
GAT (2)	$83.0 \pm 0.7\%$	$72.5 \pm 0.7\%$	$79.0 \pm 0.3\%$	93k/238k/39k
GCDE–rk4 (4)	**83.8** $\pm 0.5\%$	**72.5** $\pm 0.5\%$	**79.5** $\pm 0.3\%$	188k/478k/68k
GCDE–dpr5 **(108)**	$82.9 \pm 0.7\%$	$69.0 \pm 0.8\%$	$79.1 \pm 0.4\%$	188k/478k/68k

Table 1: Test results in percentages across 100 runs (mean and standard deviation). All GCN models have hidden dimension set to 64. For GAT and GCN64 results we refer to (Veličković et al. 2017).

(Sen et al. 2008). Main goal of these experiments is to show the usefulness of GDEs as general GNNs variants even when the data is not generated by continuous dynamical systems. We follow the standard experimental setup of (Kipf and Welling 2016) for a fair comparison. The models are optimized using Adam (Kingma and Ba 2014) with constant learning rate 0.01. L_2 weight penalty is set to 0.001 as a strong regularizer due to the small size of the training sets (Monti et al. 2017). All convolution–based models are equipped with a latent dimension of 64.

The performance of *graph convolutional differential equation* (GCDE) is assessed with both a fixed-step solver Runge–Kutta (Runge 1895; Kutta 1901) as well as an adaptive–step solver, Dormand–Prince (Dormand and Prince 1980). The resulting models are denoted as GCDE–rk4 and GCDE–dpr5 respectively. We utilize the `torchdiffeq` (Chen et al. 2018) PyTorch package to solve and backpropagate through ODEs via the adjoint method.

Discussion Following (Veličković et al. 2017), we report mean and standard deviation across 100 training runs. The best performing GCN in terms of layer depth is selected as a baseline. GCDE–rk4 outperform GCNs across all datasets; its structure offers a computationally efficient FractalNet–like (Larsson, Maire, and Shakhnarovich 2016) structure for GCNs that improves accuracy and training stability 4. GCDEs do not require more parameters than their discrete counterparts. Additionally, training GCDEs with adaptive step solvers naturally leads to deeper models than possible with vanilla GCNs, whose layer depth greatly reduces performance. We report the number of *function evaluations* (NFE) of the ODE function contained in the GCDE as a measure of its depth. It can be noted that the performance of GCDE–dpr5 is slightly worse compared to GCDE–rk4, since deeper models are penalized on these datasets by a lack of sufficient regularization (Kipf and Welling 2016).

4.2 Forecasting

Experimental setup To evaluate the effectiveness of autoregressive GDE models on forecasting tasks we perform a series of experiments on the established PeMS traffic dataset. We follow the setup of (Yu, Yin, and Zhu 2018) in which a subsampled version of PeMS, PeMS7(M), is obtained via selection of 228 sensor stations and aggregation of their historical speed data into regular 5 minute frequency time series. To simulate a challenging environment with missing data and irregular timestamps, we undersample the time series by performing independent Bernoulli trials on each data point with probability 0.7 of removal. In order to measure performance gains obtained by GDEs in settings with data generated by continuous time systems, we employ a GCDE–GRU as well as its discrete counterpart GCGRU (Zhao, Chen, and Cho 2018). To contextualize the results GRU performance on the task is included. For each model under consideration we collect *normalized RMSE* (NRMSE) and *mean absolute percentage error* (MAPE). More details about the chosen metrics and data are included as supplementary material.

Discussion The distribution of timestamp deltas (5 minute units) used to adjust the ODE integration domain of GCDE–GRU is shown in Figure 5. Non–constant differences between timestamps result in a challenging forecasting task for a single model since the average prediction horizon changes drastically over the course of training and testing. For a fair comparison between models we include delta timestamps information as an additional node feature for GCGNs and GRUs.

The main objective of these experiments is to measure the performance gain of GDEs when exploiting a correct assumption about the underlying data generating process. Traffic systems are intrinsically dynamic and continuous and therefore a model able to track continuous underlying dynamics is expected to offer improved performance. Since GCDE-GRUs and GCGRUs are designed to match exactly in structure and number of parameters we can measure this performance increase from the results shown in (2). GDEs offer an average improvement of 3% in NRSME and 7% in MAPE. A variety of other application areas with continuous dynamics and irregular datasets could similarly benefit from adopting GDEs as modeling tools: medicine, finance or distributed control systems, to name a few.

5 Conclusion

In this work we introduce the *graph neural ordinary differential equations* (GDE), the continuous counterpart to *graph neural networks* (GNN) where the inputs are propagated

Model (depth)	MAPE	NRMSE	Parameters
GRU	27.52 ± 0.00	1.47 ± 0.00	9658
GCGRU	24.80 ± 0.12	1.44 ± 0.00	9985
GDE-GRU	23.08 ± 0.11	1.40 ± 0.01	9955

Table 2: Forecasting test results across 5 runs (mean and standard deviation).

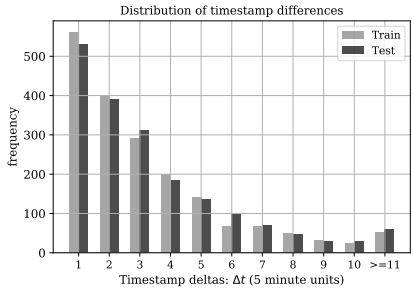

Figure 5: Distribution of deltas between timestamps $t_{k+1} - t_k$ in the undersampled dataset. The time scale of required predictions varies greatly during the task.

through a continuum of GNN layers. The GDE formulation is general, as it can be adapted to include any GNN architecture with minimal or no modifications. Additionally, GDEs represents a natural approach to sequential forecasting problems with irregularly sampled data since they are able to accommodate arbitrary timestamps. Finally, when GDEs are coupled with adaptive step solvers, as is the case for NODEs (Chen et al. 2018), they do not require apriori tuning of the number of layers and can naturally lead to deeper models, ultimately reducing the effort required on hyperparameter search for an effective deployment across application areas.

References

Atwood, J., and Towsley, D. 2016. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, 1993–2001.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Che, Z.; Purushotham, S.; Cho, K.; Sontag, D.; and Liu, Y. 2018. Recurrent neural networks for multivariate time series with missing values. *Scientific reports* 8(1):6085.

Chen, T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural ordinary differential equations. In *Advances in neural information processing systems*, 6571–6583.

Cui, Z.; Henrickson, K.; Ke, R.; and Wang, Y. 2018. Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *arXiv preprint arXiv:1802.07007*.

Deng, Z.; Nawhal, M.; Meng, L.; and Mori, G. 2019. Continuous graph flow.

Dormand, J. R., and Prince, P. J. 1980. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics* 6(1):19–26.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*.

Gholami, A.; Keutzer, K.; and Biros, G. 2019. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*.

Goebel, R.; Sanfelice, R. G.; and Teel, A. R. 2009. Hybrid dynamical systems. *IEEE Control Systems Magazine* 29(2):28–93.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Innes, M.; Edelman, A.; Fischer, K.; Rackauckus, C.; Saba, E.; Shah, V. B.; and Tebbutt, W. 2019. Zygote: A differentiable programming system to bridge machine learning and scientific computing. *arXiv preprint arXiv:1907.07587*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kutta, W. 1901. Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.* 46:435–453.

Larsson, G.; Maire, M.; and Shakhnarovich, G. 2016. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*.

Li, Y.; Yu, R.; Shahabi, C.; and Liu, Y. 2017. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*.

Li, Y.; Vinyals, O.; Dyer, C.; Pascanu, R.; and Battaglia, P. 2018. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, 539–548.

Li, X.; Wang, X.; and Chen, G. 2004. Pinning a complex dynamical network to its equilibrium. *IEEE Transactions on Circuits and Systems I: Regular Papers* 51(10):2074–2087.

Lu, Y.; Zhong, A.; Li, Q.; and Dong, B. 2017. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv preprint arXiv:1710.10121*.

Monti, F.; Boscaini, D.; Masci, J.; Rodola, E.; Svoboda, J.; and Bronstein, M. M. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5115–5124.

Park, J., and Park, J. 2019. Physics-induced graph neural network: An application to wind-farm power estimation. *Energy* 187:115883.

Quaglino, A.; Gallieri, M.; Masci, J.; and Koutník, J. 2019. Accelerating neural odes with spectral elements. *arXiv preprint arXiv:1906.07038*.

Rubanova, Y.; Chen, R. T.; and Duvenaud, D. 2019. Latent odes for irregularly-sampled time series. *arXiv preprint arXiv:1907.03907*.

Runge, C. 1895. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen* 46(2):167–178.

Sanchez-Gonzalez, A.; Heess, N.; Springenberg, J. T.; Merel, J.; Riedmiller, M.; Hadsell, R.; and Battaglia, P. 2018. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*.

Sanchez-Gonzalez, A.; Bapst, V.; Cranmer, K.; and Battaglia, P. 2019. Hamiltonian graph networks with ode integrators. *arXiv preprint arXiv:1909.12790*.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine* 29(3):93–93.

Van Der Schaft, A. J., and Schumacher, J. M. 2000. *An introduction to hybrid dynamical systems*, volume 251. Springer London.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, X. F., and Chen, G. 2002. Synchronization in scale-free dynamical networks: robustness and fragility. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 49(1):54–62.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

Yan, S.; Xiong, Y.; and Lin, D. 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Yu, B.; Yin, H.; and Zhu, Z. 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*.

Zhang, J. 2019. Gresnet: Graph residuals for reviving deep graph neural nets from suspended animation. *arXiv preprint arXiv:1909.05729*.

Zhao, X.; Chen, F.; and Cho, J.-H. 2018. Deep learning for predicting dynamic uncertain opinions in network data. In *2018 IEEE International Conference on Big Data (Big Data)*, 1150–1155. IEEE.

A Additional Static GDEs

Message passing neural networks Let us consider a single node $v \in \mathcal{V}$ and define the set of neighbors of v as $\mathcal{N} := \{u \in \mathcal{V} \ : \ (v,u) \in \mathcal{E} \vee (u,v) \in \mathcal{E}\}$. Message passing neural networks (MPNNs) perform a spatial–based convolution on the node v as

$$\mathbf{h}_{s+1}^{(v)} = \mathbf{u}_s \left(\mathbf{h}^{(v)}, \sum_{u \in \mathcal{N}(v)} \mathbf{m}_s \left(\mathbf{h}_s^{(v)}, \mathbf{h}_s^{(u)} \right) \right) \tag{7}$$

where, in general, $\mathbf{h}_0^v = \mathbf{x}_v$ while $\mathbf{u}$ and $\mathbf{m}$ are functions with trainable parameters. For clarity of exposition, let $\mathbf{u}_s(\mathbf{x}, \mathbf{y}) := \mathbf{x} + \mathbf{g}_s(\mathbf{y})$ where $\mathbf{g}_s$ is the actual parametrized function. The (7) becomes

$$\mathbf{h}_{s+1}^{(v)} = \mathbf{h}_s^{(v)} + \mathbf{g}_s \left(\sum_{u \in \mathcal{N}(v)} \mathbf{m}_s \left(\mathbf{h}_s^{(v)}, \mathbf{h}_s^{(u)} \right) \right) \tag{8}$$

and its continuous counterpart is

$$\dot{\mathbf{h}}_s^{(v)} = \mathbf{f}_{\texttt{MPNN}}^{(v)}(\mathbf{H}, \mathbf{\Theta}) := \mathbf{g}_s \left(\sum_{u \in \mathcal{N}(v)} \mathbf{m}_s \left(\mathbf{h}_s^{(v)}, \mathbf{h}_s^{(u)} \right) \right)$$

Graph Attention Networks Graph attention networks (GATs) (Veličković et al. 2017) perform convolution on the node v as

$$\mathbf{h}_{s+1}^{(v)} = \sigma \left(\sum_{u \in \mathcal{N}(v) \cup v} \alpha_{vu} \mathbf{W}_s \mathbf{h}_s^{(u)} \right) \tag{9}$$

Similarly, to GCNs, a *virtual* skip connection can be introduced allowing us to defined the GDE:

$$\dot{\mathbf{h}}_s^{(v)} = \mathbf{f}_{\texttt{GAT}}^{(v)}(\mathbf{H}, \mathbf{\Theta}) := \sigma \left(\sum_{u \in \mathcal{N}(v) \cup v} \alpha_{vu} \mathbf{\Theta} \mathbf{h}_s^{(u)} \right)$$

where α_{vu} are attention coefficient which can be computed following (Wu et al. 2019, eq. (23))

B Spatio Temporal GNNs

We include a complete description of GCGRUs to clarify the model used in our experiments.

B.1 GCGRU

GCGRUs (Zhao, Chen, and Cho 2018) perform the following computation on latents $\mathbf{H}$ and input features $\mathbf{X}$:

$$\begin{aligned}
\mathbf{Z} &= \sigma \left(\mathbf{W}_{xz} *_{\mathcal{G}} \mathbf{X}_t + \mathbf{W}_{hz} *_{\mathcal{G}} \mathbf{H}_{t-1} \right) \\
\mathbf{R} &= \sigma \left(\mathbf{W}_{xr} *_{\mathcal{G}} \mathbf{X}_t + \mathbf{W}_{hr} *_{\mathcal{G}} \mathbf{H}_{t-1} \right) \\
\tilde{\mathbf{H}} &= \tanh \left(\mathbf{W}_{xh} *_{\mathcal{G}} \mathbf{X}_t + \mathbf{W}_{hh} *_{\mathcal{G}} \left(\mathbf{R} \odot \mathbf{H}_{t-1} \right) \right) \\
\mathbf{H}_t &= \mathbf{Z} \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}) \odot \tilde{\mathbf{H}}
\end{aligned} \tag{10}$$

where the operator $*_{\mathcal{G}}$ denotes the standard graph convolution.

C Additional experimental details

Node classification The *hyperbolic tangent* (tanh) is used as activation for GDNs. Smooth activations have been observed to reduce stiffness (Chen et al. 2018) of the ODE and therefore the number of function evaluations (NFE) required for a solution that is within acceptable tolerances. All the other activation functions are *rectified linear units* (ReLU).

Forecasting PeMS7(M) contains a weighted adjacency matrix function of distances between stations. We threshold $\mathbf{A}$ using the 80th. Ratio of train/test data is set to 0.5 post–undersampling step. All models receive an input sequence of 5 graphs to perform prediction. We measured MAPE (mean absolute percentage error) and NRMSE (IQR normalized RMSE). These metrics are defined as follows:

$$\text{MAPE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{100\%}{pT} \left\| \sum_{t=1}^{T} (\mathbf{y}_t - \hat{\mathbf{y}}_t) \oslash \mathbf{y}_t \right\|_1 \tag{11}$$

where $\mathbf{y}, \text{and } \hat{\mathbf{y}} \in \mathbb{R}^p$ is the set of vectorized target and prediction of models respectively. $\oslash$ and $\|\cdot\|_1$ denotes Hadamard division and the 1-norm of vector.

$$\begin{aligned}
\mathbf{y}^{\text{IQR}} &= \mathbf{y}^{0.75} - \mathbf{y}^{0.25} \\
\text{NRMSE}(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{1}{p} \left\| \sqrt{\frac{1}{T} \sum_{t=1}^{T} (\mathbf{y}_t - \hat{\mathbf{y}}_t)^2} \oslash \mathbf{y}^{\text{IQR}} \right\|_1
\end{aligned}$$

where $\mathbf{y}^{0.75}$ and $\mathbf{y}^{0.25}$ denotes 75^{th} and 25^{th} percentiles of each station's measurement. $\mathbf{y}^{\text{IQR}}$ vector denotes the interquartile range of sensor measurements of the stations. $(\cdot)^2$ and $\sqrt{\cdot}$ denotes the element-wise squares and square root of the input vector. $\mathbf{y}_t$ and $\hat{\mathbf{y}}_t$ denote the target and prediction vector respectively.