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
[*]Equal contribution authors
{poli_m, junyoung, jinkyoo.park}@kaist.ac.kr,
{massaroli, yamashita, asama}@robot.t.u-tokyo.ac.jp

Abstract

We introduce the framework of continuous–depth *graph neural networks* (GNNs). *Graph neural ordinary differential equations* (GDEs) are formalized as the counterpart to GNNs where the input–output relationship is determined by a *continuum* of GNN layers, blending discrete topological structures and differential equations. The proposed framework is shown to be compatible with various static and autoregressive GNN models. Results prove general effectiveness of GDEs: in static settings they offer computational advantages by incorporating numerical methods in their forward pass; in dynamic settings, on the other hand, they are shown to improve performance by exploiting the geometry of the underlying dynamics.

1 Introduction

Introducing appropriate inductive biases on deep learning models is a well–known approach to improving sample efficiency and generalization performance [1]. Graph neural networks (GNNs) represent a general computational framework for imposing such inductive biases when the problem structure can be encoded as a graph or in settings where prior knowledge about entities composing a target system can itself be described as a graph [2, 3, 4]. GNNs have shown remarkable results in various application areas such as node classification [5, 6], graph classification [7] and forecasting [8, 9] as well as generative tasks [10, 11].

A different but equally important class of inductive biases is concerned with the type of temporal behavior of the systems from which the data is collected i.e., discrete or continuous dynamics. Although deep learning has traditionally been a field dominated by discrete models, recent advances propose a treatment of neural networks equipped with a continuum of layers [12, 13]. This view allows a reformulation of the forward and backward 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 on tasks involving continuous time processes [14].

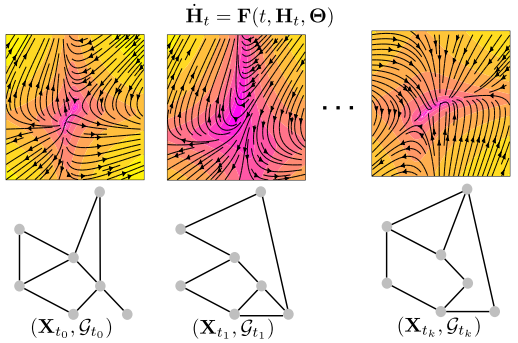

Figure 1: A *graph neural ordinary differential equation* (GDE) models vector fields defined on graphs, both in cases when the structure is fixed or changes in time, by utilizing a continuum of *graph neural network* (GNN) layers.

In this work we propose the system–theoretic framework of *graph neural ordinary differential equations* (GDEs) by defining ODEs parametrized by GNNs. GDEs are designed to

Preprint. Work in progress.

inherit the ability to impose relational inductive
biases of GNNs while retaining the dynamical system perspective of continuous–depth models. The structure–dependent vector field learned by GDEs offers a data–driven approach to the modeling of dynamical networked systems [15, 16], particularly when the governing equations are highly nonlinear and therefore challenging to approach with analytical methods. On tasks that explicitly involve dynamical systems, GDEs can adapt the prediction horizon by adjusting the integration interval of the ODE. This allows the model to track the evolution of the underlying system from irregular observations.

In general, no assumptions on the continuous nature of the data generating process are necessary in order for GDEs to be effective. Indeed, following recent work connecting different discretization schemes of ODEs [17] to previously known architectures such as FractalNets [18], we show that GDEs can equivalently be utilized as high–performance general purpose models. In this setting, GDEs offer a grounded approach to the embedding of black–box numerical schemes inside the forward pass of GNNs. Moreover, 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 [19].

We summarize our contributions as follows:

- We introduce *graph ordinary differential equation networks* (GDEs), continuous–depth counterparts to *graph neural networks* (GNNs). We show that the proposed framework is compatible with most common GNN models and allows for the use of additional inductive biases in the form of governing equations.
- We extend the GDE framework to the spatio–temporal setting and formalize a general autoregressive GDE model as a *hybrid dynamical system*.
- We validate GDEs experimentally on a static semi–supervised node classification task as well as spatio–temporal forecasting tasks. GDEs are shown to outperform their discrete GNN analogues: the different sources of performance improvement are identified and analyzed separately for static and dynamic settings.

2 Background

Notation Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{R}$ the set of reals. Scalars are indicated as lowercase letters, vectors as bold lowercase, matrices and tensors as bold uppercase and sets with calligraphic letters. Indices of arrays and matrices are reported as superscripts in round brackets.

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

$$\mathbf{A}^{(ij)} = \begin{cases} 1 & e_{ij} \in \mathcal{E} \\ 0 & e_{ij} \notin \mathcal{E} \end{cases}.$$

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 exhibit temporal dependency, i.e. $\mathbf{X} := \mathbf{X}_t$.

Neural ordinary differential equations Continuous–depth neural network architectures are built upon the observation that, for particular classes of discrete models such as ResNets [20], their inter–layer dynamics:

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

resembles the Euler discretization 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:

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

where $\mathbf{f}$ is in general a multi–layer neural network.

It has been noted that the choice of discretization scheme to solve (2) leads to previously known discrete multi–step architectures [17]. As a result, the *neural ordinary differential equation* (NODE) [13] framework is not limited to the modeling of differential equations and can guide the discovery of novel general purpose models. For the sake of compact notation, $\frac{d\mathbf{h}}{ds}$ will be denoted as $\dot{\mathbf{h}}$ throughout the paper.

Related work There exists a concurrent line of work [21] proposing a continuous variant of *graph convolution networks* (GCNs) [22] with a focus on static node classification tasks. Furthermore, [23] proposes using graph networks (GNs) [1] and ODEs to track Hamiltonian functions whereas [24] introduces a GNN version of continuous normalizing flows for generative modeling. Our goal is developing a unified system–theoretic framework for continuous–depth GNNs covering the main variants of static and spatio–temporal GNN models. We evaluate on both static as well as dynamic tasks with the primary aim of uncovering the sources of performance improvement of GDEs in each setting.

3 Graph Neural Ordinary Differential Equations

We begin by introducing the general formulation of *graph neural differential ordinary equations* (GDEs).

3.1 General Framework

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

$$\begin{cases} \mathbf{H}(s+1) = \mathbf{H}(s) + \mathbf{F}_{\mathcal{G}}\left(s, \mathbf{H}(s), \mathbf{\Theta}(s)\right) \\ \mathbf{H}(0) = \mathbf{X}_e \end{cases}, \quad s \in \mathbb{N},$$

where $\mathbf{X}_e \in \mathbb{R}^{n\times h}$ is an embedding of $\mathbf{X}$[1], $\mathbf{F}_{\mathcal{G}}$ is a matrix–valued nonlinear function conditioned on graph $\mathcal{G}$ and $\mathbf{\Theta}(s) \in \mathbb{R}^p$ 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 [25]. A *graph neural differential ordinary equation* (GDE) is defined as the following Cauchy problem:

$$\begin{cases} \dot{\mathbf{H}}(s) = \mathbf{F}_{\mathcal{G}}\left(s, \mathbf{H}(s), \mathbf{\Theta}\right) \\ \mathbf{H}(0) = \mathbf{X}_e \end{cases}, \quad s \in \mathcal{S} \subset \mathbb{R}, \tag{3}$$

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

To reduce complexity and stiffness of learned vector fields in order to alleviate the computational burden of adaptive ODE solvers, the node features can be augmented [26] by concatenating additional dimensions or prepending input layers to the GDE.

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 embedded input) $\mathbf{X}_e$, 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 $\mathbb{R}^{n\times h}$ to the space of absolutely continuous functions $\mathcal{S} \to \mathbb{R}^{n\times h}$ such that $\mathbf{H} := \mathbf{\Psi}(\mathbf{X}_e)$ satisfies the ODE in (3). This implies the the output $\mathbf{Y}$ of the GDE satisfies

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

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

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

Note that applying an output layer to $\mathbf{Y}$ before passing it to downstream applications is generally beneficial.

[1] $\mathbf{X}_e$ can be obtained from $\mathbf{X}$, e.g. with a single linear layer: $\mathbf{X}_e := \mathbf{X}\mathbf{W}$, $\mathbf{W} \in \mathbb{R}^{d\times h}$ or with another GNN layer.

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 $\mathcal{S}$. Following [13] we use the *number of function evaluations* (NFE) of the numerical solver utilized to solve (3) as a proxy for model depth. In applications 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* [14] without assumptions on the functional form of the underlying vector field, as is the case for example with exponential decay in GRU–D [27].

GDE training GDEs can be trained with a variety of methods. Standard backpropagation through the computational graph, adjoint sensitivity method [28] for $\mathcal{O}(1)$ memory efficiency [13], or backpropagation through a relaxed spectral elements discretization [29]. Numerical instability in the form of accumulating errors on the adjoint ODE during the backward pass of Neural ODEs has been observed in [30]. A proposed solution is a hybrid checkpointing–adjoint scheme commonly employed in scientific computing [31], 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 [32] along with Neural ODEs and other continuous depth models. Scientific deep learning is concerned with merging prior, incomplete knowledge about governing equations with data-driven models to enhance prediction performance, sample efficiency and interpretability [33]. Within this framework GDEs can be extended to settings involving dynamical networks evolving according to different classes of differential equations, such as *second–order* differential equation in the case of mechanics:

$$\begin{cases} \ddot{\mathbf{H}}(s) = \mathbf{F}_{\mathcal{G}}\left(s, \mathbf{H}(s)\right) \\ \left[\dot{\mathbf{H}}(0), \mathbf{H}(0)\right] = \mathbf{X}_e \end{cases}, \tag{4}$$

where the feature matrix contains the full state (i.e., position and velocity) of each node in the dynamical network. In this setting, GDEs enforce inductive biases on the "physics" of the data generating process, additionally to their intrinsic geometric structure. This approach can also be seen as an early conceptual extension of [34] to GNNs.

3.2 Static Models

Graph convolution differential equations Based on graph spectral theory [35, 36], the residual version of *graph convolution network* (GCN) [22] layers are in the form:

$$\mathbf{H}(s+1) = \mathbf{H}(s) + \sigma\left[\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}(s)\boldsymbol{\Theta}(s)\right],$$

where $\tilde{\mathbf{A}} := \mathbf{A} + \mathbb{I}_n$ and $\tilde{\mathbf{D}}$ is a diagonal matrix defined as $\tilde{\mathbf{D}}^{(ii)} := \sum_j \tilde{\mathbf{A}}^{(ij)}$ and $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation applied element–wise to its argument. The corresponding continuous counterpart, *graph convolution differential equation* (GCDE), is therefore defined as

$$\dot{\mathbf{H}}(s) = \mathbf{F}_{\texttt{GCN}}(\mathbf{H}(s), \boldsymbol{\Theta}) := \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}(s)\boldsymbol{\Theta}\right), \tag{5}$$

Note that other convolution filters can be applied as alternatives to the first–order approximation of the Chebychev one. See, e.g., [37, 38, 39, 5]. Diffusion–type convolution layers [8] are also compatible with the continuous–depth formulation.

Additional models and considerations We include additional derivation of continuous counterparts of common static GNN models such as *graph attention networks* (GAT) [40] and general message passing GNNs 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.

3.3 Spatio–Temporal Models

For settings involving a temporal component (i.e., modeling dynamical systems), the depth domain of GDEs coincides with the time domain $s \equiv t$ and can be adapted depending on the requirements.

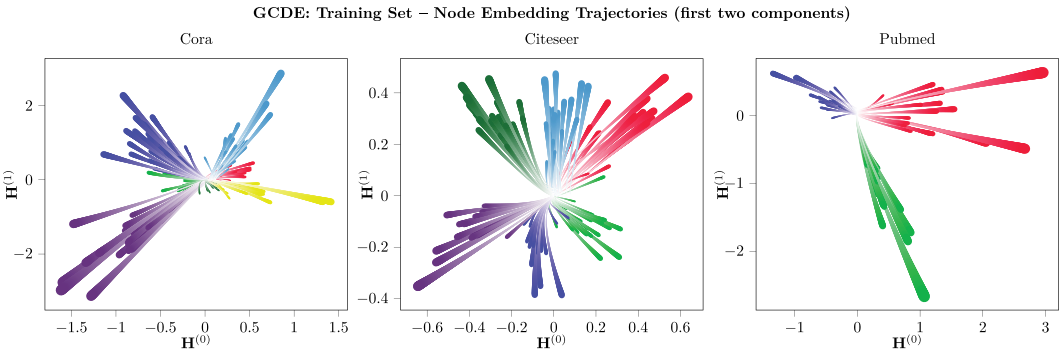

Figure 2: Node embedding trajectories defined by a forward pass of GCDE–dpr5 on Cora, Citeseer and Pubmed. Color differentiates between node classes.

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 seamlessly link to dynamical network theory.

Spatio–temporal GDEs as hybrid systems This line of reasoning naturally leads to an extension of classical spatio–temporal architectures in the form of *hybrid dynamical systems* [41, 42], i.e., systems characterized by interacting continuous and discrete –time dynamics. Let $(\mathcal{K}, >)$, $(\mathcal{T}, >)$ be linearly ordered sets; namely, $\mathcal{K} \subset \mathbb{N}\backslash\{0\}$ 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 $\{t : (t, j) \in \mathrm{dom}\, \mathbf{\Phi}\}$.

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.

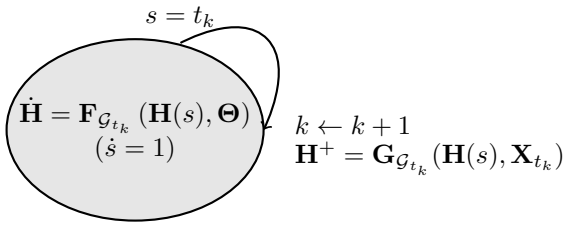

Figure 3: Schematic of autoregressive GDEs as hybrid automata.

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

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

where $\mathbf{F}, \mathbf{G}, \mathbf{K}$ are GNN–like operators or general neural network layers[2] and $\mathbf{H}^+$ represent the value of $\mathbf{H}$ after the discrete transition. The evolution of system (6) is indeed a sequence of hybrid

[2]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 (NFE)	Cora	Citeseer	Pubmed
GCN	$81.4 \pm 0.5\%$	$70.9 \pm 0.5\%$	$79.0 \pm 0.3\%$
GCN*	$82.8 \pm 0.3\%$	$71.2 \pm 0.4\%$	$79.5 \pm 0.4\%$
GCDE–rk2 (2)	$83.0 \pm 0.6\%$	$72.3 \pm 0.5\%$	**79.9** $\pm 0.3\%$
GCDE–rk4 (4)	**83.8** $\pm 0.5\%$	**72.5** $\pm 0.5\%$	$79.5 \pm 0.4\%$
GCDE–dpr5 (**158**)	$81.8 \pm 1.2\%$	$68.3 \pm 1.2\%$	$78.5 \pm 0.7\%$

Table 1: Test results in percentages across 100 runs (mean and standard deviation). All models have hidden dimension set to 64.

arcs defined on a hybrid time domain. A graphical representation of the overall system is given by the *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 the evolution of dynamical systems from observations with irregular time steps.

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.

GCDE–GRU We illustrate the generality of (6) by deriving the continuous–depth version of GCGRUs [43] as:

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

where $\mathbf{K}$ is a fully–connected neural network. The complete description of a GCGRU layer of computation is included as supplementary material. We refer to this model as GCDE–GRU. Similarly, GCDE–RNNs or GCDE–LSTMs can be obtained by replacing the GRU cell with other commonly used recurrent modules, such as vanilla RNNs or LSTMs [44].

4 Experiments

We evaluate GDEs on a suite of different tasks. The experiments and their primary objectives are summarized below:

- Semi–supervised node classification on static, standard benchmark datasets Cora, Citeseer, Pubmed [45]. We investigate the usefulness of the proposed method in a static setting via an ablation analysis that directly compares GCNs and analogue GCDEs solved with fixed–step and adaptive solvers.
- Trajectory extrapolation task on a synthetic multi–agent dynamical system. We compare Neural ODEs and GDEs while providing a motivating example for the framework of scientific deep learning in the form of second order models (4).
- Traffic forecasting on an undersampled version of PeMS [46] dataset. We measure the performance improvement obtained by a correct inductive bias on continuous dynamics and robustness to irregular timestamps.

The code will be open–sourced after the review phase and is included in the submission.

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 [45]. Main goal of these experiments is to perform an ablation study on the source of possible performance advantages of the GDE framework in settings that do not involve continuous dynamical systems.

The L_2 weight penalty is set to $5 \cdot 10^{-4}$ on Cora, Citeseer and 10^{-3} on Pubmed as a strong regularizer due to the small size of the training set [47]. We report mean and standard deviation across 100 training runs. Since our experimental setup follows [22] to allow for a fair comparison, other baselines present in recent GNN literature can be directly compared with Table 1.

Models and baselines All convolution–based models are equipped with a latent dimension of 64. We include results for best performing vanilla GCN baseline presented in [40]. To avoid flawed comparisons, we further evaluate an optimized version of GCN, GCN*, sharing exact architecture, as well as training and validation hyperparameters with the GCDE models. We experimented with different number of layers for GCN*: (2, 3, 4, 5, 6) and select 2, since it achieves the best results. The performance of *graph convolution differential equation* (GCDE) is assessed with both a fixed-step solver Runge–Kutta [48, 49] as well as an adaptive–step solver, Dormand–Prince [50]. The resulting models are denoted as GCDE–rk4 and GCDE–dpr5, respectively. We utilize the `torchdiffeq` [13] PyTorch package to solve and backpropagate through the ODE solver.

Continuous–depth models in static tasks The evaluation of continuous variants of GNNs in concurrent work [21] exclusively considers the static setting, where continuous–depth models do not have an inherent modeling advantage. As an example, ensuring a low error solution to the ODE parametrized by the model with adaptive–step solvers does not offer particular advantages in image classification tasks [13] compared to equivalent discrete models. While there is no reason to expect performance improvements solely from the transition away from discrete architectures, continuous–depth allows for the embedding of numerical ODE solvers in the forward pass. Multi–step architectures have previously been linked to ODE solver schemes [17] and routinely outperform their single–step counterparts [18, 17]. We investigate the performance gains by employing the GDE framework in static settings as a straightforward approach to the embedding of numerical ODE solvers in GNNs.

Results The variants of GCDEs solved with fixed–step schemes are shown to outperform or match GCN* across all datasets, with the margin of improvement being highest on Cora and Citeseer. Introducing GCDE–rk2 and GCDE–rk4 is observed to provide the most significant accuracy increases in more densely connected graphs or with larger training sets. In particular, GCDE–rk4 outperforming GCDE–rk2 indicates that, given equivalent network architectures, higher order ODE solvers are generally more effective, provided the graph is dense enough to benefit from the additional computation. Additionally, training GCDEs with adaptive step solvers naturally leads to deeper models than possible with vanilla GCNs, whose layer depth greatly reduces performance. However, the high *number of function evaluation* (NFEs) of GCDE–dpr5 necessary to stay within the ODE solver tolerances causes the model to overfit and therefore generalize poorly. We visualize the first two components of GCDE–dpr5 node embedding trajectories in Figure 2. The trajectories are divergent, suggesting a non–decreasing classification performance for GCDE models trained with longer integration times. We provide complete visualization of accuracy curves in Appendix C.

Resilience to integration time For each integration time $S \in [1, 5, 10]$, we train 100 GCDE-dpr5 models on Cora and report average metrics, along with 1 standard deviation confidence intervals in Figure 4. GCDEs are shown to be resilient to changes in S; however, GCDEs with longer integration times require more training epochs to achieve comparable accuracy. This result suggests that, indeed, GDEs are immune to node oversmoothing [51].

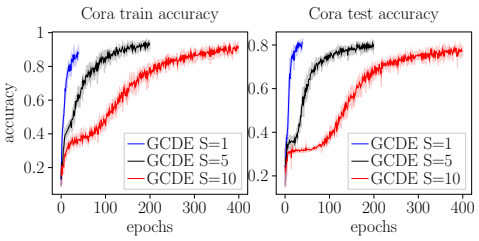

Figure 4: Cora accuracy of GCDE models with different integration times s. Higher values of S do not affect performance negatively but require a higher number of epochs.

4.2 Multi–Agent Trajectory Extrapolation

Experimental setup We evaluate GDEs and a collection of deep learning baselines on the task of extrapolating the dynamical behavior of a synthetic mechanical multi–particle system. Particles interact within a certain radius with a viscoelastic force. Outside the mutual interactions, captured by a time–varying adjacency matrix $\mathbf{A}_t$, the particles would follow a periodic motion. The adjacency matrix $\mathbf{A}_t$ is computed along the trajectory as:

$$\mathbf{A}_t^{(ij)} = \begin{cases} 1 & 2\|\mathbf{x}_i(t) - \mathbf{x}_j(t)\| \leq r \\ 0 & \text{otherwise} \end{cases},$$

where $\mathbf{x}_i(t)$ is the position of node i at time t. Therefore, $\mathbf{A}_t$ results to be symmetric, $\mathbf{A}_t = \mathbf{A}_t^\top$ and yields an undirected graph. The dataset is collected by integrating the system for $T = 5s$ with a fixed step–size of $dt = 1.95 \cdot 10^{-3}$ and is split evenly into a training and test set. We consider 10 particle systems. An example trajectory is shown in Figure 5. All models are optimized to minimize mean–squared–error (MSE) of 1–step predictions using Adam [52] with constant learning rate 0.01.

We measure test *mean average percentage error* (MAPE) of model predictions in different extrapolation settings. *Extrapolation steps* denotes the number of predictions each model Φ has to perform without access to the nominal trajectory. This is achieved by recursively letting inputs at time t be model predictions at time $t - \Delta t$ i.e $\hat{\mathbf{Y}}_{t+\Delta t} = \phi(\hat{\mathbf{Y}}_t)$ for a certain number of extrapolation steps, after which the model is fed the actual nominal state $\mathbf{X}$ and the cycle is repeated until the end of the test trajectory. For a robust comparison, we report mean and standard deviation across 10 seeded training and evaluation runs. Additional experimental details, including the analytical formulation of the dynamical system, are provided as supplementary material.

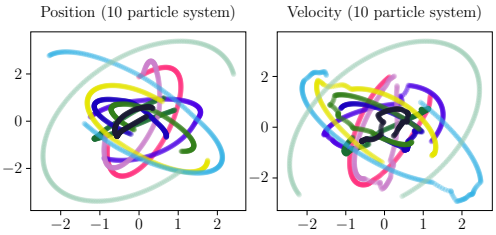

Figure 5: Example position and velocity trajectories of the multi–particle system.

Models and baselines As the vector field depends only on the state of the system, available in full during training, the baselines do not include recurrent modules. We consider the following models:

- A 3–layer fully-connected neural network, referred to as *Static*. No assumption on the dynamics
- A vanilla Neural ODE with the vector field parametrized by the same architecture as *Static*. ODE assumption on the dynamics.
- A 3–layer convolution GDE, GCDE. Dynamics assumed to be determined by a blend of graphs and ODEs
- A 3–layer, second order GCDE as described by (4) and referred to as *GCDE-II*. GCDE assumptions in addition to second order ODE dynamics.

A grid hyperparameter search on number of layers, ODE solver tolerances and learning rate is performed to optimize *Static* and Neural ODEs. We use the same hyperparameters for GDEs.

Results Figure 6 shows the growth rate of test MAPE error as the number of extrapolation steps is increased. *Static* fails to extrapolate beyond the 1–step setting seen during training. Neural ODEs overfit spurious particle interaction terms and their error rapidly grows as the number of extrapolation steps is increased. GCDEs, on the other hand, are able to effectively leverage relational information to track the system, as shown in Figure

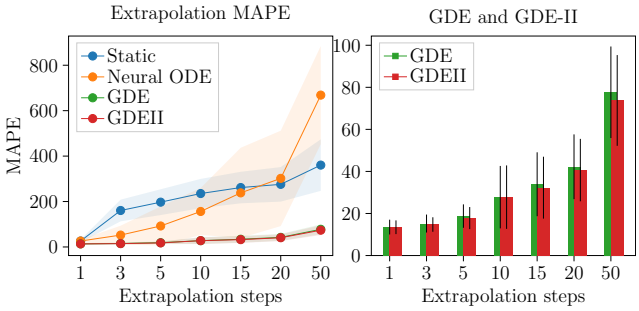

Figure 6: Test extrapolation MAPE averaged across 10 experiments. Shaded area and error bars indicate 1–standard deviation intervals.

7. Lastly, GCDE-IIs outperform first–order GCDEs as their structure inherently possesses crucial information about the relative relationship of positions and velocities that is accurate with repsect to the observed dynamical system. To further improve performance, additional information about the governing equations can be encoded in the model [33].

4.3 Traffic Forecasting

Experimental setup We evaluate the effectiveness of autoregressive GDE models on forecasting tasks by performing a series of experiments on the established PeMS traffic dataset. We follow the setup of [46] 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. We construct the adjacency matrix $\mathbf{A}$ by thresholding of the Euclidean distance between observation stations i.e. when two stations are closer than the threshold distance, an edge between them is included. The threshold is set to the 40^{th} percentile of the station distances distribution. 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. Results for 3 increasingly challenging experimental setups are provided: undersampling with 30%, 50% and 70% of removal. In order to provide a robust evaluation of model performance in regimes with irregular data, the testing is repeated 20 times per model, each with a different undersampled version of the test dataset. We collect *root mean square error* (RMSE) and MAPE. More details about the chosen metrics and data are included as supplementary material.

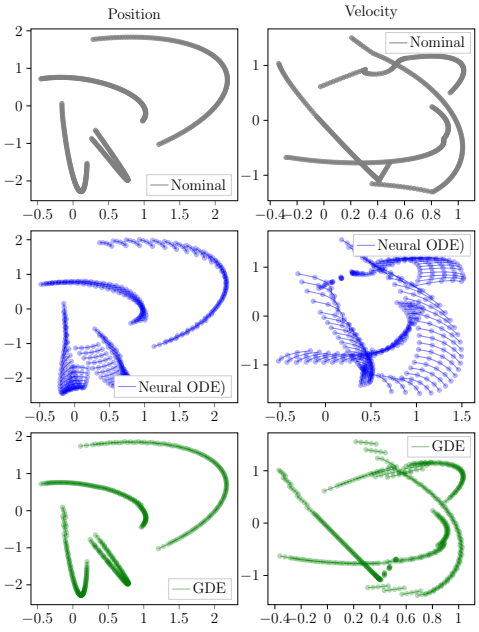

Figure 7: Test extrapolation, 5 steps. Trajectory predictions of Neural ODEs and GDEs. The extrapolation is terminated after 5 steps and the nominal state is fed to the model, as described in the experimental setup.

Models and baselines 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 [43]. To contextualize the effectiveness of introducing graph representations, we include the performance of GRUs since they do not directly utilize structural information of the system in predicting outputs. Apart from GCDE–GRU, both baseline models have no innate mechanism for handling timestamp information. For a fair comparison, we include timestamp differences between consecutive samples and *sine–encoded* [53] absolute time information as additional features. All models receive an input sequence of 5 graphs to perform the prediction.

Results 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.

Traffic systems are intrinsically dynamic and continuous in nature and, therefore, a model able to track continuous underlying dynamics is expected to offer improved performance. Since GCDE-

Model	$\text{MAPE}_{70\%}$	$\text{RMSE}_{70\%}$	$\text{MAPE}_{50\%}$	$\text{RMSE}_{50\%}$	$\text{MAPE}_{30\%}$	$\text{RMSE}_{30\%}$
GRU	27.14 ± 0.45	13.25 ± 0.11	27.08 ± 0.26	13.218 ± 0.065	27.24 ± 0.191	13.28 ± 0.05
GCGRU	23.60 ± 0.38	11.97 ± 0.03	22.86 ± 0.22	11.779 ± 0.056	21.324 ± 0.16	11.20 ± 0.04
GCDE-GRU	$\mathbf{22.95} \pm 0.37$	$\mathbf{11.67} \pm 0.10$	$\mathbf{21.25} \pm 0.21$	$\mathbf{11.04} \pm 0.05$	$\mathbf{20.94} \pm 0.14$	$\mathbf{10.95} \pm 0.04$

Table 2: Forecasting test results across 20 runs (mean and standard deviation). MAPE_i indicates $i\%$ undersampling of the test set.

GRUs and GCGRUs are designed to match in structure we can measure this performance increase from the results shown in Table 2. GCDE–GRUs outperform GCGRUs and GRUs in all undersampling regimes. Additional details and prediction visualizations are included in Appendix C.

5 Discussion

We discuss future extensions compatible with the GDE framework.

Unknown or dynamic topology Several lines of work concerned with learning the graph structure directly from data exist, either by inferring the adjacency matrix within a probabilistic framework [54] or using a soft–attention [55] mechanism [56, 57, 9]. In particular, the latter represents a commonly employed approach to the estimation of a dynamic adjacency matrix in spatio–temporal settings. Due to the algebraic nature of the relation between the attention operator and the node features, GDEs are compatible with its use inside the GNN layers parametrizing the vector field. Thus, if an optimal adaptive graph representation $\mathbf{S}(s, \mathbf{H})$ is computed through some attentive mechanism, standard convolution GDEs can be replaced by:

$$\dot{\mathbf{H}} = \sigma\left(\mathbf{S}\mathbf{H}\boldsymbol{\Theta}\right). \tag{7}$$

Introduction of control terms The GDE formulation allows for the introduction of control inputs:

$$\begin{cases} \dot{\mathbf{H}}(s) = \mathbf{F}_{\mathcal{G}}\left(s, \mathbf{H}(s), \boldsymbol{\Theta}\right) + \mathbf{U}\left(s\right) \\ \mathbf{H}(0) = \mathbf{X}_e \end{cases}, \quad s \in \mathcal{S}\,. \tag{8}$$

System (8) encompasses a variety of previously proposed approaches involving special residual connections [58] and multi–stage or jumping knowledge GNNs [59], unifying them under a control theory point of view. In particular, the control input can be parametrized by an additional neural network $\mathbf{U}\left(s\right) := \mathbf{U}_{\mathcal{G}}\left(s, \mathbf{X}\right)$.

6 Conclusion

In this work we introduce *graph neural ordinary differential equations* (GDE), the continuous–depth counterpart to *graph neural networks* (GNN) where the inputs are propagated through a continuum of GNN layers. The GDE formulation is general, as it can be adapted to include many static and autoregressive GNN models. GDEs are designed to offer a data–driven modeling approach for *dynamical networks*, whose dynamics are defined by a blend of discrete topological structures and differential equations. In sequential forecasting problems, GDEs can accommodate irregular timestamps and track the underlying continuous dynamics, whereas in static settings they offer computational advantages by allowing for the embedding of black–box numerical solvers in their forward pass. GDEs have been evaluated on both static and dynamic tasks and have been shown to outperform their discrete counterparts. Future directions include extending GDEs to other classes of differential equations as well as settings where the number of graph nodes evolves in time.

References

[1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[2] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pages 539–548, 2018.

[3] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*, 2019.

[4] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.

[5] Chenyi Zhuang and Qiang Ma. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*, pages 499–508. International World Wide Web Conferences Steering Committee, 2018.

[6] Claudio Gallicchio and Alessio Micheli. Fast and deep graph neural networks, 2019.

[7] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[8] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

[9] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. *arXiv preprint arXiv:1906.00121*, 2019.

[10] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[11] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

[12] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

[13] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[14] Yulia Rubanova, Ricky TQ Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *arXiv preprint arXiv:1907.03907*, 2019.

[15] Jinhu Lu and Guanrong Chen. A time-varying complex dynamical network model and its controlled synchronization criteria. *IEEE Transactions on Automatic Control*, 50(6):841–846, 2005.

[16] Martin Andreasson, Dimos V Dimarogonas, Henrik Sandberg, and Karl Henrik Johansson. Distributed control of networked dynamical systems: Static feedback, integral action and consensus. *IEEE Transactions on Automatic Control*, 59(7):1750–1764, 2014.

[17] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv preprint arXiv:1710.10121*, 2017.

[18] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[19] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9267–9276, 2019.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, and James S. Duncan. Ordinary differential equations on graph networks, 2020.

[22] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[23] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators. *arXiv preprint arXiv:1909.12790*, 2019.

[24] Zhiwei Deng, Megha Nawhal, Lili Meng, and Greg Mori. Continuous graph flow, 2019.

[25] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016.

[26] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. *arXiv preprint arXiv:1904.01681*, 2019.

[27] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):6085, 2018.

[28] Lev Semenovich Pontryagin, EF Mishchenko, VG Boltyanskii, and RV Gamkrelidze. The mathematical theory of optimal processes. 1962.

[29] Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. Accelerating neural odes with spectral elements. *arXiv preprint arXiv:1906.07038*, 2019.

[30] Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

[31] Qiqi Wang, Parviz Moin, and Gianluca Iaccarino. Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation. *SIAM Journal on Scientific Computing*, 31(4):2549–2567, 2009.

[32] Mike Innes, Alan Edelman, Keno Fischer, Chris Rackauckus, Elliot Saba, Viral B Shah, and Will Tebbutt. Zygote: A differentiable programming system to bridge machine learning and scientific computing. *arXiv preprint arXiv:1907.07587*, 2019.

[33] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, and Ali Ramadhan. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

[34] Çağatay Yıldız, Markus Heinonen, and Harri Lähdesmäki. Ode^{2}vae: Deep generative second order odes with bayesian neural networks. *arXiv preprint arXiv:1905.10994*, 2019.

[35] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

[36] Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *IEEE transactions on signal processing*, 61(7):1644–1656, 2013.

[37] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[38] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[39] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018.

[40] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[41] Arjan J Van Der Schaft and Johannes Maria Schumacher. *An introduction to hybrid dynamical systems*, volume 251. Springer London, 2000.

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

[43] Xujiang Zhao, Feng Chen, and Jin-Hee Cho. Deep learning for predicting dynamic uncertain opinions in network data. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1150–1155. IEEE, 2018.

[44] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[45] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[46] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[47] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

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

[49] Wilhelm Kutta. Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453, 1901.

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

[51] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification, 2019.

[52] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[53] Gábor Petneházi. Recurrent neural networks for time series forecasting. *arXiv preprint arXiv:1901.00069*, 2019.

[54] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[56] Edward Choi, Mohammad Taha Bahadori, Le Song, Walter F Stewart, and Jimeng Sun. Gram: graph-based attention model for healthcare representation learning. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 787–795. ACM, 2017.

[57] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

[59] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

[60] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

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 $\mathcal{N}(v) := \{u \in \mathcal{V} \ : \ (v,u) \in \mathcal{E} \vee (u,v) \in \mathcal{E}\}$. Message passing neural networks (MPNNs) perform a spatial–based convolution on the node v as

$$\mathbf{h}^{(v)}(s+1) = \mathbf{u}\left[\mathbf{h}^{(v)}(s), \sum_{u \in \mathcal{N}(v)} \mathbf{m}\left(\mathbf{h}^{(v)}(s), \mathbf{h}^{(u)}(s)\right)\right], \tag{9}$$

where, in general, $\mathbf{h}^{v}(0) = \mathbf{x}_v$ while $\mathbf{u}$ and $\mathbf{m}$ are functions with trainable parameters. For clarity of exposition, let $\mathbf{u}(\mathbf{x}, \mathbf{y}) := \mathbf{x} + \mathbf{g}(\mathbf{y})$ where $\mathbf{g}$ is the actual parametrized function. The (9) becomes

$$\mathbf{h}^{(v)}(s+1) = \mathbf{h}^{(v)}(s) + \mathbf{g}\left[\sum_{u \in \mathcal{N}(v)} \mathbf{m}\left(\mathbf{h}^{(v)}(s), \mathbf{h}^{(u)}(s)\right)\right], \tag{10}$$

and its continuous–depth counterpart, *graph message passing differential equation* (GMDE) is:

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

Graph Attention Networks Graph attention networks (GATs) [40] perform convolution on the node v as

$$\mathbf{h}^{(v)}(s+1) = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup v} \alpha_{vu} \mathbf{\Theta}(s) \mathbf{h}^{(u)}(s)\right). \tag{11}$$

Similarly, to GCNs, a *virtual* skip connection can be introduced allowing us to define the *graph attention differential equation* (GADE):

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

where α_{vu} are attention coefficient which can be computed following [40].

B Spatio–Temporal GDEs

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

B.1 GCGRU Jumps

Following GCGRUs [43], we perform a instantaneous change of $\mathbf{H}$ at each time t_k using the next input features $\mathbf{X}_{t_k}$. Let $\bar{\mathbf{A}}_{t_k} := \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{t_k} \tilde{\mathbf{D}}^{-\frac{1}{2}}$. Then, let

$$\begin{aligned}
\mathbf{Z} &:= \sigma\left(\bar{\mathbf{A}}_{t_k} \mathbf{X}_{t_k} \mathbf{\Theta}_{xz} + \bar{\mathbf{A}}_{t_k} \mathbf{H} \mathbf{\Theta}_{hz}\right), \\
\mathbf{R} &:= \sigma\left(\bar{\mathbf{A}}_{t_k} \mathbf{X}_{t_k} \mathbf{\Theta}_{xr} + \bar{\mathbf{A}}_{t_k} \mathbf{H} \mathbf{\Theta}_{hr}\right), \\
\tilde{\mathbf{H}} &:= \tanh\left(\bar{\mathbf{A}}_{t_k} \mathbf{X}_{t_k} \mathbf{\Theta}_{xh} + \bar{\mathbf{A}}_{t_k} \left(\mathbf{R} \odot \mathbf{H}\right) \mathbf{\Theta}_{hh}\right).
\end{aligned} \tag{12}$$

Finally,

$$\mathbf{H}^{+} = \texttt{GCGRU}(\mathbf{H}, \mathbf{X}_t) := \mathbf{Z} \odot \mathbf{H} + (\mathbb{1} - \mathbf{Z}), \odot \tilde{\mathbf{H}} \tag{13}$$

where $\mathbf{\Theta}_{xz}$, $\mathbf{\Theta}_{hz}$, $\mathbf{\Theta}_{xr}$, $\mathbf{\Theta}_{hr}$, $\mathbf{\Theta}_{xh}$, $\mathbf{\Theta}_{hh}$ are matrices of trainable parameters, σ is the standard sigmoid activation and $\mathbb{1}$ is all–ones matrix of suitable dimensions.

C Additional experimental details

Computational resources We carried out all experiments on a cluster of 4x12GB NVIDIA® Titan Xp GPUs and CUDA 10.1. The models were trained on GPU.

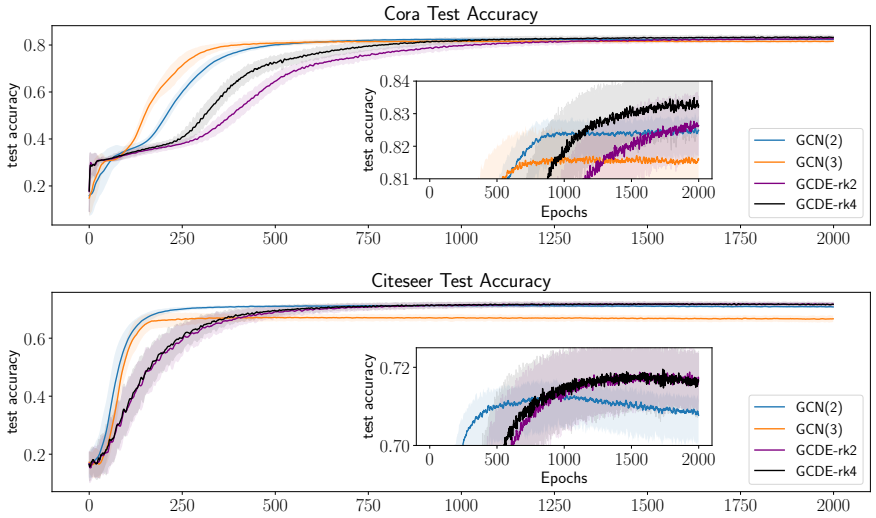

Figure 8: Test accuracy curves on Cora and Citeseer (100 experiments). Shaded area indicates the 1 standard deviation interval.

Layer	Input dim.	Output dim.	Activation
GCN–in	dim. in	64	ReLU
GDE–1 (GCN)	64	64	Softplus
GDE–2 (GCN)	64	64	None
GCN-out	64	dim. out	None

Table 3: General architecture for GCDEs on node classification tasks. GCDEs applied to different datasets share the same architecture. The vector field $\mathbf{F}$ is parameterized by two GCN layers. GCDEs–dopri5 shares the same structure without GDE–2 (GCN).

C.1 Node classification

Training hyperparameters All models are trained for 2000 epochs using Adam [52] with learning rate $lr = 10^{-3}$ on Cora, Citeseer and $lr = 10^{-2}$ on Pubmed due to its training set size. The reported results are obtained by selecting the lowest validation loss model after convergence (i.e. in the epoch range 1000 – 2000). Test metrics are not utilized in any way during the experimental setup. For the experiments to test resilience to integration time changes, we set a higher learning rate for all models i.e. $lr = 10^{-2}$ to reduce the number of epochs necessary to converge.

Architectural details *SoftPlus* is used as activation for GDEs. Smooth activations have been observed to reduce stiffness [13] 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). The exact input and output dimensions for the GCDE architectures are reported in Table 3. The vector field $\mathbf{F}$ of GCDEs–rk2 and GCDEs–rk4 is parameterized by two GCN layers. GCDEs–dopri5 shares the same structure without GDE–2 (GCN). Input GCN layers are set to dropout 0.6 whereas GCN layers parametrizing $\mathbf{F}$ are set to 0.9.

C.2 Multi–agent system dynamics

Dataset Let us consider a planar multi agent system with states $\mathbf{x}_i$ $(i = 1, \dots, n)$ and second–order dynamics:

$$\ddot{\mathbf{x}}_i = -\mathbf{x}_i - \sum_{j \in \mathcal{N}_i} \mathbf{f}_{ij}(\mathbf{x}_i, \mathbf{x}_j, \dot{\mathbf{x}}_i, \dot{\mathbf{x}}_j),$$

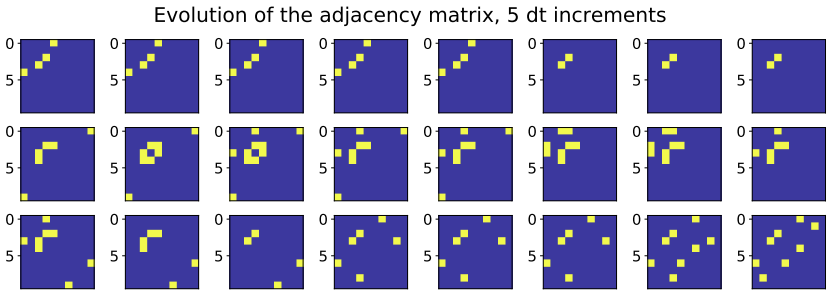

Figure 9: Snapshots of the evolution of adjacency matrix $\mathbf{A}_t$ throughout the dynamics of the multi–particle system. Yellow indicates the presence of an edge and therefore a reciprocal force acting on the two bodies

Model	MAPE_1	MAPE_3	MAPE_5	MAPE_{10}	MAPE_{15}	MAPE_{20}	MAPE_{50}
Static	26.12	160.56	197.20	235.21	261.56	275.60	360.39
Neural ODE	26.12	52.26	92.31	156.26	238.14	301.85	668.47
GDE	13.53	15.22	18.76	27.76	33.90	42.22	77.64
GDE–II	13.46	14.75	17.81	27.77	32.28	40.64	73.75

Table 4: Mean MAPE results across the 10 multi–particle dynamical system experiments. MAPE_i indicates results for i extrapolation steps on the full test trajectory.

where

$$\mathbf{f}_{ij} = -\left[\alpha\left(\|\mathbf{x}_i - \mathbf{x}_j\| - r\right) + \beta\frac{\langle\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j, \mathbf{x}_i - \mathbf{x}_j\rangle}{\|\mathbf{x}_i - \mathbf{x}_j\|}\right]\mathbf{n}_{ij},$$
$$\mathbf{n}_{ij} = \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}, \quad \alpha,\ \beta,\ r > 0,$$

and

$$\mathcal{N}_i := \{j : 2\|\mathbf{x}_i - \mathbf{x}_j\| \leq r \wedge j \neq i\}.$$

The force $\mathbf{f}_{ij}$ resembles the one of a spatial spring with drag interconnecting the two agents. The term $-\mathbf{x}_i$, is used instead to stabilize the trajectory and avoid the "explosion" of the phase–space. Note that $\mathbf{f}_{ij} = -\mathbf{f}_{ji}$. The adjaciency matrix $\mathbf{A}_t$ is computed along a trajectory

$$\mathbf{A}_t^{(ij)} = \begin{cases} 1 & 2\|\mathbf{x}_i(t) - \mathbf{x}_j(t)\| \leq r \\ 0 & \text{otherwise} \end{cases},$$

which indeed results to be symmetric, $\mathbf{A}_t = \mathbf{A}_t^\top$ and thus yields an undirected graph. Figure 9 visualizes an example trajectory of $\mathbf{A}_t$. We collect a single rollout with $T = 5$, $dt = 1.95 \cdot 10^{-3}$ and $n = 10$. The particle radius is set to $r = 1$.

Architectural details Node feature vectors are 4 dimensional, corresponding to the dimension of the state, i.e. position and velocity. Neural ODEs and *Static* share an architecture made up of 3 fully–connected layers: $4n$, $8n$, $8n$, $4n$ where $n = 10$ is the number of nodes. The last layer is linear. We evaluated different hidden layer dimensions: $8n$, $16n$, $32n$ and found $8n$ to be the most effective. Similarly, the architecture of first order GCDEs is composed of 3 GCN layers: 4, 16, 16, 4. Second order GCDEs, on the other hand, are augmented by 4 dimensions: 8, 32, 32, 8. We experimented with different ways of encoding the adjacency matrix $\mathbf{A}$ information into Neural ODEs and $Static$ but found that in all cases it lead to worse performance.

C.3 Traffic Forecasting

Dataset and metrics The timestamp differences between consecutive graphs in the sequence varies due to undersampling. The distribution of timestamp deltas (5 minute units) for the three different experiment setups (30%, 50%, 70% undersampling) is shown in Figure 11.

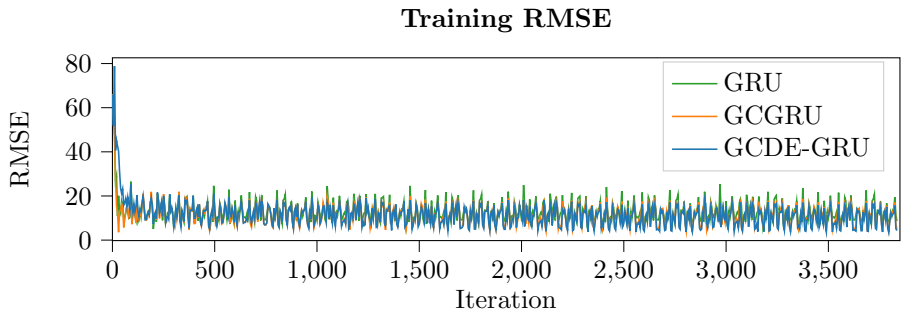

Figure 10: Traffic data training results of 50% undersampling.

As a result, GRU takes 230 dimensional vector inputs (228 sensor observations + 2 additional features) at each sequence step. Both GCGRU and GCDE–GRU graph inputs with and 3 dimensional node features (observation + 2 additional feature). The additional time features are excluded for the loss computations. We include MAPE and RMSE test measurements, defined as follows:

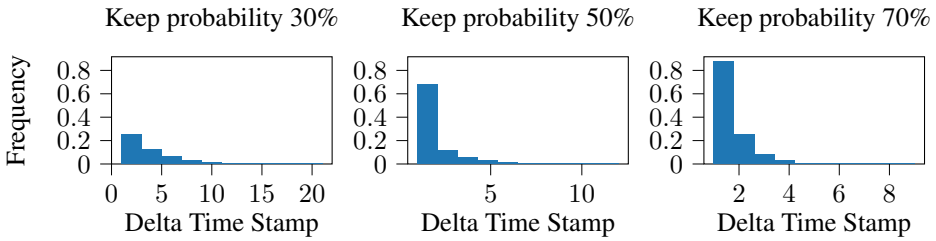

Figure 11: Distribution of deltas between timestamps $t_{k+1} - t_k$ in the undersampled dataset. The time scale of required predictions varies greatly during the task.

$$\text{MAPE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{100\%}{pT} \left\| \sum_{t=1}^{T} (\mathbf{y}_t - \hat{\mathbf{y}}_t) \oslash \mathbf{y}_t \right\|_1, \tag{14}$$

where $\mathbf{y}$, 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.

$$\text{RMSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{p} \left\| \sqrt{\frac{1}{T} \sum_{t=1}^{T} (\mathbf{y}_t - \hat{\mathbf{y}}_t)^2} \right\|_1,$$

where $(\cdot)^2$ and $\sqrt{\cdot}$ denotes the element-wise square and square root of the input vector, respectively. $\mathbf{y}_t$ and $\hat{\mathbf{y}}_t$ denote the target and prediction vector.

Architectural details We employed two baseline models for contextualizing the importance of key components of GCDE–GRU. *GRUs* architectures are equipped with 1 GRU layer with hidden dimension 50 and a 2 layer fully–connected head to map latents to predictions. *GCGRUs* employ a GCGRU layer with 46 hidden dimension and a 2 layer fully–connected head. Lastly, GCDE–GRU shares the same architecture GCGRU with the addition of the flow $\mathbf{F}$ tasked with evolving the hidden features between arrival times. $\mathbf{F}$ is parametrized by 2 GCN layers, one with tanh activation and the second without activation. ReLU is used as the general activation function.

Training hyperparameters All models are trained for 40 epochs using Adam[52] with $lr = 10^{-2}$. We schedule lr by using cosine annealing method [60] with $T_0 = 10$. The optimization is carried out by minimizing the *mean square error* (MSE) loss between predictions and corresponding targets.

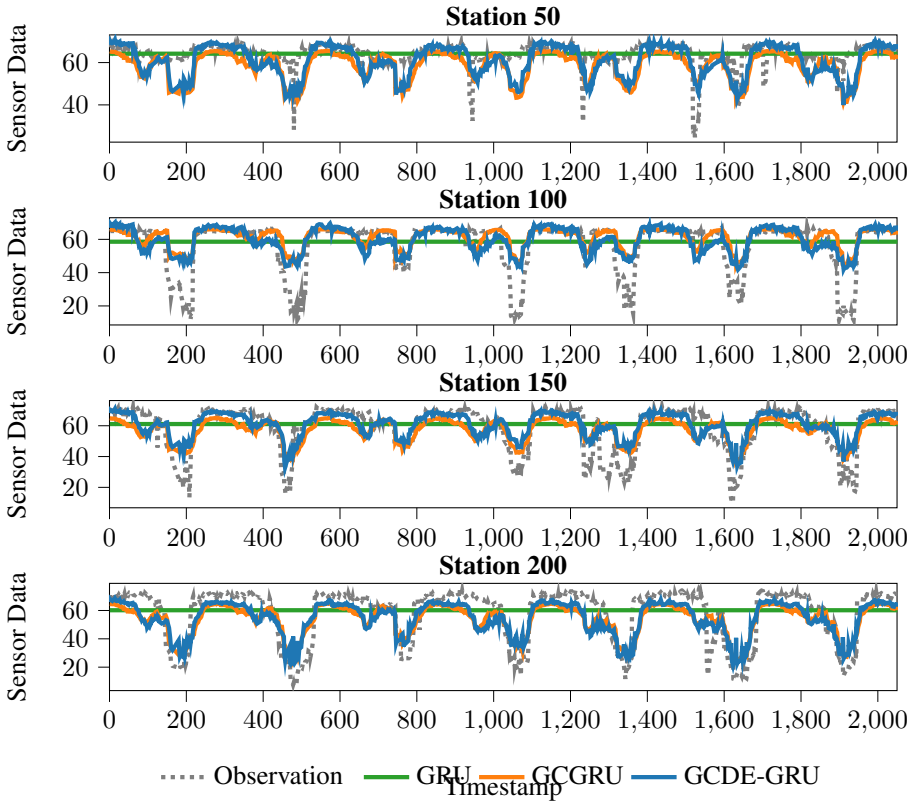

Figure 12: Traffic data prediction results of 50% undersampling. GCDE–GRUs are able to evolve the latents between timestamps and provide a more accurate fit.

Additional results Training curves of the models are presented in the Fig 10. All of models achieved nearly 13 in RMSE during training and fit the dataset. However, due to the lack of dedicated spatial modeling modules, GRUs were unable to generalize to the test set and resulted in a mean value prediction.