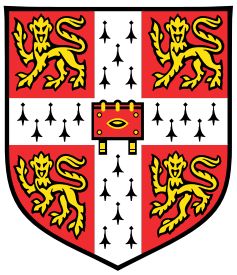
Graph Neural Stochastic Differential Equations



Richard Bergna

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

Clare Hall August 2023

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 15,000 (14,432) words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures. The code related to this work is available on GitHub and can be accessed via the reference Bergna (2023).

Richard Bergna
August 2023

Acknowledgements

First and foremost, my heartfelt gratitude goes to my supervisor, Jose Miguel Hernandez-Lobato. I deeply value Lobato's invaluable insights and amiable disposition, which consistently welcomed my queries and ideas. Each of our discussions was constructive and help the trajectory of my research. I'm also immensely grateful to Pietro Liò. Our interactions were always imbued with thought-provoking insights. Pietro's charismatic presence was not just a source of inspiration but also a beacon of positivity. Special recognition goes to Felix Opola. His presence, especially during the initial phase of my dissertation when the path was uncertain, proved invaluable. Felix's patience and willingness to explore intricate details about GNN and the papers we studied together immensely enhanced my understanding.

To my girlfriend, Maya Rao, thank you for your relentless support and understanding during this challenging yet exhilarating year. Your companionship, particularly during the intense months of this dissertation, made the entire experience more rewarding. I'd also like to acknowledge my gym buddy, Sergio Calvo Ordoñez. Our gym sessions and lighthearted conversations were not only refreshing breaks but also an inadvertent boost to my productivity. My heartfelt gratitude to Emma Prevot, Connall Garrod, and all my friends in Cambridge. Your company and support undoubtedly made my time more enjoyable and fulfilling. A special shout-out to my family: my twin, Philippe Bergna, for his keen interest in my dissertation and his insightful questions; my sister, Hazel, whose regular check-ins and concern for my well-being were immensely comforting; and my dad, Gilberto Bergna, who always encouraged me to be the best version of myself and took a genuine interest in my work.

Lastly, I extend my gratitude to Andrew Blake. The Clare Halls Blake's funds, which he generously provided, have been instrumental. Without this significant support, undertaking this Masters Program would not have been feasible.

To all who played a part in this journey, either directly or indirectly, I am forever thankful.

Abstract

In this dissertation, we introduce a novel framework, *Graph Neural Stochastic Differential Equations* (Graph Neural SDEs), which extends the capabilities of *Graph Neural Ordinary Differential Equations* (Graph Neural ODEs) by integrating randomness into data representation through Brownian motion. This advancement allows for the quantification of prediction uncertainty, a critical aspect often neglected in current methodologies. The primary focus revolves around the *Latent Graph Neural Stochastic Differential Equations* (Latent Graph Neural SDE). This approach is central to our research and has demonstrated superior capabilities in a multitude of contexts. In addition, we introduce a complementary model, the *Graph Neural GAN-SDE*, tailored for generating synthetic graph data in line with an SDE's dynamics. Empirical analyses underscore the exceptional performance of the Latent Graph Neural SDE, outpacing conventional contenders like Graph Convolutional Networks and Graph Neural ODEs across both static and spatio-temporal domains. Moreover, we evaluate the robustness of our model in out-of-distribution detection, uncertainty quantification, and active learning for data collection scenarios, where Graph Neural SDEs consistently outperform other tested models. Consequently, this research represents a substantial progression in the field, introducing a more robust and uncertainty-aware approach to graph neural networks.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Graph Neural Networks (GNNs) have profoundly influenced the landscape of graph-structured data interpretation (Bronstein et al., 2017; Hamilton et al., 2017; Kipf and Welling, 2016; Veličković et al., 2017), adeptly decoding challenges from social networks to intricate biological systems (Wu et al., 2020; Zitnik and Leskovec, 2017). Yet, for all their advantages, GNNs face a persistent limitation: they frequently grapple with the 'oversmoothing' issue, especially when multiple layers are employed (Li et al., 2020).

To advance the field, the authors in Poli et al. (2019) turned to Neural ODEs — continuous-time machine learning models rooted in the marriage of differential equations' theoretical rigor and neural networks' adaptability (Chen et al., 2018). These models provided a continuous counterpart to traditional discrete neural architectures, allowing for dynamic adaptations to data in its native temporal resolution. Building upon this foundation, Graph Neural ODEs emerged as a significant derivative. This fusion reaped two pivotal benefits. First, it adeptly addressed the oversmoothing issue, thereby enabling the construction of deeper graph neural architectures. Secondly, its inherent design effortlessly accommodates continuous-time data, a feature proving essential for irregularly sampled datasets (Poli et al., 2019).

However, both GNNs and Graph Neural ODEs share a glaring oversight: an inability to effectively quantify predictive uncertainties. Uncertainty quantification in Graph Neural Networks has profound real-world implications. Whether for robust decision-making in high-stakes scenarios like medical diagnosis, establishing model trustworthiness, guiding model improvement, detecting anomalies, enhancing data efficiency, or catering to risk-sensitive

applications, understanding uncertainty in predictions is pivotal (Gal and Ghahramani, 2016; Kendall and Gal, 2017; Lakshminarayanan et al., 2017; Leibig et al., 2017).

However, both GNNs and Graph Neural ODEs share a glaring oversight: an inability to effectively quantify predictive uncertainties. In the realm of machine learning, uncertainty quantification (UQ) is the process of determining the level of confidence a model has in its predictions (Gal and Ghahramani, 2016). Ideally, a model should recognize and candidly indicate when it makes predictions on unfamiliar or "out-of-distribution" data points. For instance, in classification tasks, if a model trained only on images of cats and dogs is suddenly presented with an image of a bird, it should ideally predict nearly equal probabilities for both classes, signaling its unfamiliarity with the input. This absence of overconfidence in uncertain scenarios is vital. Similarly, in regression tasks, as predictions deviate further from known training data points, the uncertainty or variance in predictions should grow. UQ is pivotal not only for determining the model's trustworthiness but also for offering valuable insights on areas that might benefit from further data collection or model refinement.

Uncertainty quantification in Graph Neural Networks has profound real-world implications. Whether for robust decision-making in high-stakes scenarios like medical diagnosis, establishing model trustworthiness, guiding model improvement, detecting anomalies, enhancing data efficiency, or catering to risk-sensitive applications, understanding uncertainty in predictions is pivotal (Gal and Ghahramani, 2016; Kendall and Gal, 2017; Lakshminarayanan et al., 2017; Leibig et al., 2017).

1.1 Contribution

Addressing this gap, our work introduces a hybrid approach that combines the strengths of Graph Neural ODEs with the innate ability of Stochastic Differential Equations (SDEs) to quantify uncertainty. This synthesis gives birth to Graph Neural SDEs. Crafting a Graph Neural SDE, however, introduces its own unique challenges, which we'll detail later in Chapter 3. To navigate these intricacies, we propose two innovative methodologies: the primary being the **Latent Graph Neural SDE**, which is for node, link, and graph prediction, and the supplementary **Graph SDE-GAN**, which demonstrates the capabilities of generating synthetic graph data with SDE-styled features.

Empirical results evidence the advantages of our approach over several counterparts like Graph Neural ODE, GCN, Bayesian GNN (Lamb and Paige, 2020), and Ensemble GNN

(Lin et al., 2022) in aspects such as robustness, out-of-distribution detection (emphasizing our effective uncertainty quantification), and accuracy. In essence, this work presents a novel methodology for continuous Graph Neural Networks, ensuring effective uncertainty quantification.

1.2 Dissertation Roadmap

Chapter 2: In this chapter, we lay the foundational concepts which form the backbone of the entire discourse. It begins with a dive into the realm of Neural Differential Equations, with its different flavors – Neural Ordinary Differential Equations, Neural Controlled Differential Equations, and Neural Stochastic Differential Equations. As we traverse this landscape, we also highlight the inherent advantages of using such equations. From there, the narrative shifts to Graph Neural Networks (GNNs). This segment offers a walkthrough of their inception, their prominent subtypes like Graph Convolutional Networks and Graph Attention Networks, and touches upon the persisting challenges, particularly the oversmoothing issue. The chapter concludes by introducing Graph Neural Ordinary Differential Equations, which serve as a conduit between GNNs and differential equations, showcasing how they are realized and their associated challenges.

Chapter 3: This chapter is devoted to the Graph Neural Stochastic Differential Equations (GN-SDEs), a primary contribution of this thesis. It initiates with an elucidation of the concept of GN-SDEs and delves into particular models, emphasizing the Latent Graph Neural SDEs and Graph SDE-GAN. The journey continues with a juxtaposition of Graph Neural ODEs and GN-SDEs, emphasizing their distinctive features and underlying connections. Further, the chapter offers a rigorous evaluation of the models, testing their mettle against various tasks and datasets. Moreover, the chapter draws connections to relevant works, providing a broader context.

Chapter 4: As we reach the culmination of our exploration, Chapter 4 offers a reflective analysis of the journey undertaken. It encapsulates the insights gathered, the implications of the models proposed, and the innovations introduced. Additionally, this chapter casts an eye towards the horizon, speculating about potential future endeavors in this realm and laying down a roadmap for subsequent research efforts.

Chapter 2

Background

In this chapter, we delve into the intricate background that forms the foundation of our exploration, weaving together strands from diverse areas. We begin by understanding the motivation behind employing differential equations within the realm of neural networks. This motivates our exploration into Neural Ordinary Differential Equations (ODEs), followed by the intricacies of Neural Control Differential Equations (CDEs) and Neural Stochastic Differential Equations (SDEs). With this mathematical foundation in place, our focus shifts to Graph Neural Networks, discussing popular architectures that are pivotal in our experimental studies. Finally, we introduce Graph Neural ODEs, demonstrating how the worlds of graph networks and differential equations intersect.

We anticipate readers to have a foundational grasp of differential equations. However, it's not essential to possess an exhaustive understanding. In this section, we endeavor to provide all necessary background details to ensure comprehensibility. For readers seeking a deeper or more specialized mathematical discourse, we direct them to our extended background section A. Throughout this chapter, we will reference this extended section when deemed appropriate.

2.1 Neural Differential Equations

Before the widespread use of neural networks and modern machine learning, differential equations were the gold standard for modeling diverse systems. These equations spanned

the realms of physics, biology, and social sciences (Buckdahn et al., 2011; Cardelli, 2008; Cvijovic et al., 2014; Hoops et al., 2016; Mandelzweig and Tabakin, 2001; Quach et al., 2007). Their prowess lay in capturing continuous changes over time and space, as evident from their aptitude in representing systems such as the motion of planets: Kepler's laws (Russell, 1964), the spread of diseases: SIR model (Kermack and McKendrick, 1927), and the behavior of electric circuits: Maxwell's equations (Maxwell, 1861).

As computational capabilities expanded and the volume of available data surged, the landscape of modeling and prediction shifted noticeably. Neural networks, once a burgeoning concept, came to the fore. They displayed an unprecedented ability to decode complex patterns and handle nonlinear relationships in data (Aggarwal et al., 2022; LeCun et al., 2015). This evolution in data analysis methods led to an essential pondering: Was it possible to merge the time-tested, robust modeling framework of differential equations with the newfound flexibility and depth offered by neural networks? Out of this introspection and the quest for a synthesis, Neural Differential Equations (NDEs) were conceptualized, embodying the best attributes of both traditional differential equations and modern neural networks.

2.1.1 Neural Ordinary Differential Equations

Neural Ordinary Differential Equations (Neural ODEs) were the first to combine neural network principles with differential equations (Chen et al., 2018). They offer a refined strategy to model dynamical systems by leveraging the foundational principles of neural networks. This new model, instead of representing data transformations as discrete layers in a traditional deep network, Neural ODEs describe them as continuous transformations parameterized by differential equations. This concept of continuous transformations within the neural network is often termed as "continuous-depth" or "continuously-deep", implying that instead of having distinct layers, the network smoothly transitions and evolves data through a continuum of depths.

These transformations specify how the state of a system, denoted as $z(t)$, evolves over time. The rate of change in the state of the system is determined by a function f, which is parameterized by neural network weights.

Definition: A Neural ODE is given by

$$\frac{dz}{dt} = f_\phi(z(t),t), \qquad z(0) = z_0.$$

where f_ϕ signifies the neural network with parameters ϕ.

This differential equation implies that the state of the system at any time t is determined by accumulating the effects of the function f from the initial state z_0 up to that time. This can be articulated more explicitly as

$$z(t) = z_0 + \int_{t_0}^{t_1} f(z(t),t)\,dt,$$

where t_0 and t_1 are the initial and final times, respectively, hence t lies within the interval $[t_0,t_1]$. The solution to this integral is usually determined using a numerical ODE solver.

Remark: Neural ODEs are completely differentiable, allowing for the standard back-propagation techniques to fine-tune the network's parameters ϕ in order to reduce the difference between observed and forecasted trajectories.

The core objective of the model is to ensure the predicted trajectory closely aligns with the actual trajectory. To accomplish this, the model undergoes a training process where it continuously adjusts its internal parameters to minimize the difference (or "loss") between these trajectories.

In many applications, Neural ODEs primarily serve to extract a latent representation or an embedding of data, denoted by z_t. Once this latent state is determined, it's passed through another neural network to generate a prediction $\hat{y}$, referencing the true label y. At their core, Neural ODEs aim to map x to y through the learned function f_ϕ and the associated linear transformations l^1_ϕ and l^2_ϕ. This can be formally captured as

$$y \approx l^1_\phi(z_t), \quad \text{where} \quad z_t = z_0 + \int_0^t f_\phi(z_s)\,ds, \quad \text{and} \quad z_0 = l^2_\phi(x).$$

In this formulation, z_t represents the system's state at time t, with $f_\phi(z_s)$ detailing the system's dynamics. The linear transformations, l^1_ϕ and l^2_ϕ, play distinctive roles: l^2_ϕ is employed to

determine the initial state z_0 from input x, and l_ϕ^1 is used to estimate the output y based on the terminal state z_t (Chen et al., 2018; Kidger, 2022).

Remark: Neural differential equations, including Neural ODEs, present a transformative approach to conceptualizing depth in deep learning models. Unlike traditional architectures where depth is defined by the discrete layers' count, here depth takes on a continuous form.

As mentioned, in the neural differential equations framework, "depth" is associated with an integration interval, typically represented as $[t_0, t_1]$. Here, t_0 and t_1 define the starting and ending points in a "time" or depth continuum. As the hidden state, $z(t)$, transforms according to the differential equation $\frac{dz}{dt} = f_\phi(z(t), t)$, its progression from t_0 to t_1 mirrors transformations found in traditional layer-based networks.

This continuous modeling eliminates the need for discrete layers. Instead, the neural ODE's capacity and intricacy arise from the governing differential equation and the states' continuous progression throughout the defined integration period. Within this framework, the interpretation of variables z and t varies with application.

Static datasets: $z(t)$ represents the data's current state at a specific "depth" or complexity, t (sometimes denoted as "s"), of the network's transformation. Here, t does not symbolize real-time but rather signifies the network's depth or transformation complexity.

Spatio-temporal datasets: $z(t)$ stands for the system's state at an actual time, t. The learned function f_ϕ reflects the system's temporal dynamics, moving beyond mere transformational mappings from inputs to outputs.

Neural ODEs as continuously-deep Residual Networks

In this section, we delve into the intimate connection between Neural Ordinary Differential Equations (ODEs) and Residual Networks (ResNets), drawing from insights presented by Chen et al. (2018); Kidger (2022).

Theorem: Neural ODEs are the continuously-deep analogs of ResNets.

Proof: To draw parallels between Neural ODEs and ResNets, let's first consider the discrete formulation of a ResNet, as detailed in Liu et al. (2021a)

$$y_{j+1} = y_j + f_\Theta(j, y_j), \tag{2.1}$$

where y_j represents the state or output at the j-th layer, and $f_\Theta(j, y_j)$ denotes the j-th residual block with parameters encapsulated in Θ.

Transitioning to the domain of Neural ODEs, the form is given by

$$\frac{dy}{dt}(t) = f_\Theta(t, y(t)), \tag{2.2}$$

where $y(t)$ signifies the state of the system at time t, and the function $f_\Theta(t, y(t))$ describes the rate of change of this state, parameterized by Θ.

Employing the explicit Euler method, and taking discrete time points t_j with a uniform difference of Δt, the above Neural ODE is discretized as

$$\frac{y(t_{j+1}) - y(t_j)}{\Delta t} \approx \frac{dy}{dt}(t_j) = f_\Theta(t_j, y(t_j)),$$

Simplifying, we get

$$y(t_{j+1}) = y(t_j) + \Delta t \cdot f_\Theta(t_j, y(t_j)).$$

On integrating the term Δt into f_Θ, we can observe that this formulation mirrors that of the ResNet in equation 2.1. This underlines the idea that Neural ODEs can be interpreted as a continuous-time rendition of ResNets.

Building on this, we can envision a Neural ODE as an infinite-layered ResNet. In such a representation, minute updates (akin to residuals) to its state occur incessantly. The culminating output emerges as an aggregation of these innumerable, minuscule updates, reminiscent of solving the ODE from its initial condition.

Remark: This bridge between ResNets and Neural ODEs not only offers a theoretical understanding but also practical benefits. While training a traditional ResNet mandates the

retention of layer-wise activations for backpropagation, a Neural ODE only demands the storage of its initial state. The subsequent states, at any desired time points, are computed on-the-fly by the ODE solver. This characteristic can result in significant memory economy, especially when the analogous ResNet has a large number of layers.

2.1.2 Neural Controlled Differential Equations

A alternative to Neural ODEs are Neural Controlled Differential Equations (Neural CDEs) proposed by Kidger et al. (2020). They are a recent advancement in deep learning that extend the concept of Neural ODEs to model temporal dynamics more effectively. They are particularly suitable for handling time-series data, as they incorporate a mechanism to adjust the system's trajectory based on subsequent observations.

Unlike Neural ODEs, which are determined solely by their initial conditions, Neural CDEs incorporate data that arrives later through the mathematics of controlled differential equations. This makes the Neural CDEs model applicable to the general setting of partially observed irregularly-sampled multivariate time series.

Definition: The Neural CDE model is defined as the solution of the CDE

$$z(t) = z(t_0) + \int_{t_0}^{t} f_\phi(z(s))dx(s) \quad \text{for} \quad t \in (t_0, t_n],$$

where '$dx(s)$' denotes a Riemann-Stieltjes integral, and '$f(z(s))dx(s)$' refers to a matrix-vector multiplication.

Remark: Neural CDEs can leverage the same techniques, including ODE solvers, traditionally employed for Neural ODEs.

2.1.3 Neural Stochastic Differential Equations

Building upon our understanding of Neural ODEs and Neural CDEs, we next delve into the fascinating realm of Stochastic Differential Equations (SDEs). SDEs have been widely employed to model real-world phenomena that exhibit randomness, such as physical systems,

financial markets, population dynamics, and genetic variations (Gontis and Kononovicius, 2014; Ricciardi, 2012; Schreiber et al., 2011). They generalize ODEs by modeling systems that evolve continuously over time, incorporating randomness. Informally, an SDE can be seen as an ODE that integrates a certain degree of noise

$$\frac{dz}{dt} = f(z(t),t) + \varepsilon(t).$$

Here, $\varepsilon(t)$ represents the time-dependent noise, modeled using diffusion models and Brownian motion.

Definition: An SDE is represented as:

$$dz(t) = \underbrace{f(t,z(t))}_{\text{drift}}dt + \underbrace{g(t,z(t))}_{\text{diffusion}}dW(t),$$

In this equation, the system state $z(t)$ at time t evolves due to two main components: the drift function f and the diffusion function g. The term $\mathrm{d}W(t)$ denotes the infinitesimal increment of a standard Brownian motion (or Wiener process) $W(t)$, with properties like $W(0) = 0$, independent increments, and $W(t) - W(\tau)$ being normally distributed with mean 0 and variance $t - \tau$ for $0 \leq \tau < t$. The strong solution for the SDE, denoted as $z(t)$, exists and is unique under conditions where f and g are Lipschitz and $E[z(0)^2] < \infty$[a].

[a]For a comprehensive and rigorous exploration of Stochastic Differential Equations, readers are referred to Khasminskii (2012) and Revuz and Yor (2013).

The **drift function** $f(t,z(t))$ represents the deterministic aspect of the system's evolution. It predicts the expected direction of the system's change at each point in time, based on the current state $z(t)$.

The **diffusion function** $g(t,z(t))$ represents the stochastic or random aspect of the system's evolution. It models the inherent randomness or uncertainty in the data by scaling the random noise introduced by the Wiener process $W(t)$, also known as Brownian motion.

Remark: Despite the promising capabilities of Neural SDEs, a significant challenge arises when the diffusion function g is a learnable parameter and is trained by maximizing likelihood, for instance, by directly minimizing cross-entropy or mean square error. In these scenarios, the diffusion function tends to converge towards zero, effectively trans-

forming the Neural SDE into a Neural ODE. Researchers have proposed diverse strategies to counteract this, including the minimization of Kullback-Leibler (KL) divergence or Wasserstein distance. These methodologies underpin advanced constructs like 'Latent SDEs' and 'SDE-Generative Adversarial Networks' (SDE-GANs) Kidger et al. (2021); Li et al. (2020).

Neural SDEs as continuous Recurrent Neural Networks

Similar to how Neural ODEs can be perceived as continuous residual neural networks, we will present an intuitive interpretation of Neural SDEs as continuously-deep, Recurrent Neural Networks (RNNs). A prominent analogy draws a parallel between numerically discretized neural stochastic differential equations and deep learning architectures, especially RNNs. For Neural SDEs, the RNN's input is akin to random noise or Brownian motion, while its output represents a generated sample Li et al. (2020).

Theorem: Neural SDEs are the continuously-deep of RNNs.

Proof: Consider an autonomous one-dimensional Stochastic Differential Equation represented as:

$$dy(t) = f(y(t))dt + \sigma(y(t))dw(t)$$

where $y(t)$, $f(y(t))$, $\sigma(y(t))$, and $w(t)$ belong to the set of real numbers, $\mathbb{R}$. The numerical Euler-Maruyama discretization of this SDE can be expressed as:

$$\frac{y(t_{j+1}) - y(t_j)}{\Delta t} \approx f(y(t_j)) + \frac{\sigma(y(t_j))\Delta w_j}{\Delta t}$$

Which simplifies to:

$$y_{j+1} = y_j + f(y_j)\Delta t + \sigma(y_j)\Delta w_j$$

Here, Δt denotes a fixed time step, and Δw_j follows a normal distribution with zero mean and variance Δt. This numerical discretization bears resemblance to a specific form of an RNN. As such, the Neural SDE can be viewed as the continuous-time limit of RNN, with the depth determined by the number of discretization steps.

2.1.4 Advantages of Neural Differential Equations

Having covered the essential background on NDFs, we now turn our attention to the benefits of these models. NDEs, combine the robust foundations of differential equations with the flexibility of modern neural networks. This integration offers several unique advantages.

Enhanced Representation: Through their continuous nature, NDEs excel at capturing the complexity of systems, often more efficiently in terms of parameters than discrete models Chen et al. (2018).

Adaptable Computation: NDEs dynamically alter their computational depth based on the complexity of the input data, optimizing the utilization of computational assets Li et al. (2020).

Robustness: Rooted in the theories of differential equations, NDEs maintain stability, proving resilient to challenges like adversarial interventions Sitzmann et al. (2020).

Versatility in Data Interpretation: NDEs can handle diverse datasets. For **static datasets**, $z(t)$ denotes the data's state at a certain depth or transformation complexity t within the neural framework. Here, t represents the network's depth or transformation complexity, reminiscent of layers in architectures like ResNet/RNN. In contrast, for **spatio-temporal datasets**, $z(t)$ (refer to 2.1.3) signals the system's state at an actual time instance t. In this context, the function f charts the system's time-based evolution rather than mere transformational mappings. This adaptability emphasizes the inherent interpretability of NDEs.

Summary

Throughout our journey, we have explored various neural differential equations, encompassing Neural ODEs, CDEs, and SDEs. These models can be interpreted as continuous representations of intricate neural architectures, such as RNNs and ResNet. Notably, these architectures rank among the most prevalent and transformative in modern deep learning. Additionally, we delved into the interpretative essence of these models in both static and spatio-temporal datasets, highlighting their multifaceted advantages. As we proceed, we will introduce Graph Neural Networks, discuss their challenges, and lay the groundwork for understanding Graph Neural SDEs.

2.2 Graph Neural Networks

Emerging as one of the primary tools for learning graph-structured data, Graph Neural Networks (GNNs) were first introduced in Scarselli et al. (2009). Drawing their inspiration from traditional neural networks, GNNs specialize in processing structured graph data. Graphs, foundational to myriad domains ranging from social networks to molecular structures, present unique challenges and opportunities for machine learning (Liu et al., 2021b; Min et al., 2021; Zhou et al., 2020).

Definition: A graph, represented as $\mathcal{G} = (V, E)$, is comprised of nodes (or vertices) V and edges E that connect these nodes. Each node in the graph is characterized by a d-dimensional feature vector x_v. All such feature vectors, numbering $n = |V|$, are assembled into an $n \times d$ matrix, expressed as $X = H(0)$. The set of edges E is captured within an $n \times n$ adjacency matrix A, with an entry A_{ij} being 1 if an edge $e = v_i \to v_j$ is present in E, and 0 otherwise. Each node $v \in V$ and edge $e \in E$ can be associated with respective feature vectors, denoted as x_v and x_e.

GNNs process the graph $\mathcal{G} = (V, E)$, accompanied by node features x_v and edge features x_e. They yield feature vectors for nodes (and potentially for edges) which cater to a range of tasks, such as node or graph classification/regression, and link prediction.

Example:

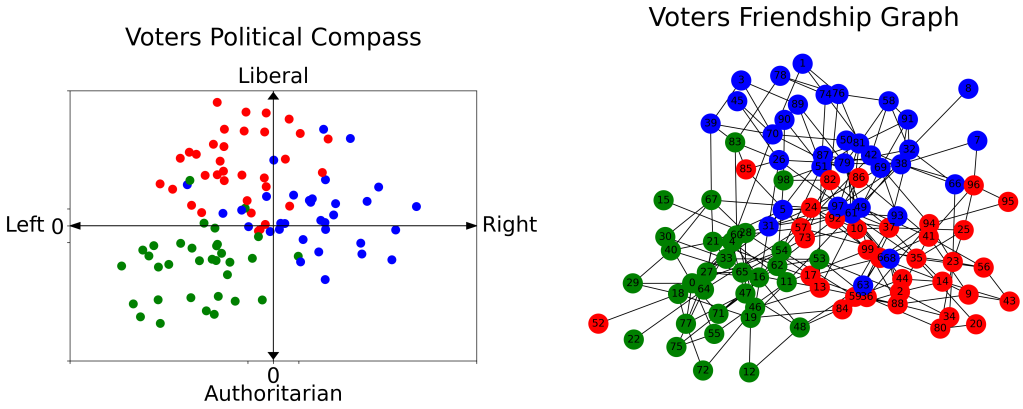


Fig. 2.1 On the left: a political compass of voters. On the right: their social circles. Node colors and features indicate voting preferences among three candidates: red, blue, and green.

The images above provide insights into the application of GNNs. The left depicts a political compass mapping voters, while the right presents their social circles. Nodes and their features signify voting preferences for three distinct candidates: red, blue, and green. The challenge lies in predicting individual voting patterns, harnessing their political and social affiliations as data. This exemplifies the node classification problem using GNNs.

Central to GNNs is the concept of layers, akin to traditional neural networks. Each GNN layer operates over the entire graph, generating latent feature vectors for each node.

Definition: The l-th GNN layer, $f^{(l)}$, receives the graph's latent feature matrix from the previous layer, $H^{(l-1)}$, and the adjacency matrix A. It subsequently outputs a new latent feature matrix $H(l)$ as:

$$H^{(l)} = f^{(l)}(H^{(l-1)}, A) \tag{2.3}$$

For a multi-layered GNN, the inaugural layer, where $l = 1$, accepts $H(0) = X$ as input. In contrast, subsequent layers ingest $H(l-1)$, which are latent features from the preceding GNN layer.

This layer-wise operation can be interpreted through the lens of message passing.

Definition: Message passing in GNNs refers to the mechanism by which nodes in a graph update their representations by aggregating and processing information from their neighbors. Given a node u and its representation in the l-th layer as h_u^l, the update is described by:

$$h_u^l = \phi\left(h_u^{l-1}, \sum_{v \in N_u} \psi(h_u^{l-1}, h_v^{l-1})\right)$$

Where ϕ is the message passing function, ψ is a readout or aggregation function, and N_u denotes the neighborhood of node u.

This process ensures that the final node representation is permutation-invariant with respect to its neighbors.

Specifically, for a given node u, the latent representation h_u^l in the l layer is computed using functions ϕ and ψ, which are typically chosen as Multi-Layer Perceptrons (MLPs), although their structures can vary.

The equation above succinctly captures the GNN's operational essence: to refine the representation of a node u, it contemplates both the node's immediate neighborhood and its current representation. This neighborhood information, transformed by ψ, is aggregated using permutation-invariant functions such as sum (but we can use other operator like max). The final representation is deduced by passing the node's current representation and the aggregated neighborhood data to ϕ. The permutation-invariant nature ensures that the final output remains agnostic to the ordering of nodes.

2.2.1 Graph Convolutional Networks

Graph Convolutional Networks (GCN) stand as a pivotal advancement in the domain of GNNs. Initially introduced by (Chen et al., 2020), GCNs are fundamentally rooted in spectral graph theory. They skillfully generalize the traditional convolutional operations—most commonly employed on grid-structured data like images—to the intricacies of graph-structured data (Bruna et al., 2013).

Definition: Graph Convolutional Layer (Node-centric formulation):

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{d_v d_u}} W^{(l)} h_u^{(l)}\right),$$

Here, $h_v^{(l)}$ characterizes the feature vector of node v at the l-th layer. $\mathcal{N}(v)$ represents the neighbors of node v, and d_v depicts the degree of node v. The transformation matrix $W^{(l)}$ undergoes learning, while σ instills a non-linearity, bolstering the network's expressive capability.

Another formulation of the GCN can be depicted in a more matrix-centric manner, capturing operations on the entire graph simultaneously:

Definition: Graph Convolutional Layer (matrix-centric formulation):

$$\mathbf{H}^{(s+1)} = \mathbf{H}^{(s)} + \sigma\left(\mathbf{L}_{\mathcal{G}} \mathbf{H}^{(s)} \mathbf{P}^{(s)}\right) \tag{2.4}$$

In this equation, $\mathbf{H}^{(s)}$ holds the features of all nodes in the graph at layer s. The matrix $\mathbf{L}_{\mathcal{G}}$ signifies the graph Laplacian of the graph $\mathcal{G}$, encapsulating its structural characteristics. Meanwhile, $\mathbf{P}^{(s)}$ functions as a parameter matrix for the s-th layer.

Remark: A foundational principle of GCNs is the recursive refinement of node representations by integrating and transmuting features from their local graph vicinity. By cascading this process, nodes assimilate information from increasingly distant parts of the graph, enriching their contextual embeddings and rendering them more descriptive.

Over time, GCNs have sown the seeds for a multitude of derivatives and enhancements, each crafted to surmount particular challenges or to fine-tune the approach for specialized applications. Whether by infusing attention mechanisms, accommodating diverse graph structures, or honing computational thriftiness, the seminal influence of GCNs is palpable in the wide array of GNN architectures that have since evolved.

2.2.2 Graph Attention Networks

Graph Attention Networks (GAT) infuse attention mechanisms, notably from transformer architectures in NLP, into GNNs (Vaswani et al., 2017; Veličković et al., 2017). Unlike

GCNs, which give equal weight to all neighbors, GATs use attention scores to assign different importance levels to neighbors.

Definition: Graph Attention Layer :

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu} W^{(l)} h_u^{(l)}\right), \tag{2.5}$$

with the attention coefficients α_{vu} defined as:

$$\alpha_{vu} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [Wh_v || Wh_u]\right)\right)}{\sum_{k \in \mathcal{N}(v)} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [Wh_v || Wh_k]\right)\right)},$$

This approach is especially valuable in graphs where node relationships are heterogeneous, and not every connection is of equal relevance.

Remark: By virtue of its attention mechanism, GATs discern and give prominence to more informative parts of the structure, achieving improved performance on certain benchmarks compared to GCNs.

2.2.3 Oversmoothing in GNNs

A recurrent issue faced by GNNs, especially when dealing with deep architectures, is the phenomenon of oversmoothing. Oversmoothing, also termed as over-smoothing or over-mixing, ensues when nodes' features become overly similar across iterations or layers in the GNN (Li et al., 2018a; Oono and Suzuki, 2019). This phenomenon is particularly evident during the aggregation phase, where the feature vectors of nodes are updated based on their neighbors.

Definition: Oversmoothing in a GNN is said to occur when, after several layers or iterations, the difference between nodes' latent feature vectors diminishes. Mathematically, for nodes v_i and v_j, oversmoothing is recognized when the feature vector difference $||H_{v_i}^{(l)} - H_{v_j}^{(l)}||$ becomes negligible for a sufficiently large l, even if nodes v_i and v_j are not structurally similar or closely situated in the original graph.

The implications of oversmoothing are twofold:

1. **Loss of Distinctiveness**: As node features converge to a uniform representation, the GNN's capacity to discern nodes based on their unique attributes and topological positioning dwindles.

2. **Performance Degradation**: Deep GNNs affected by oversmoothing often manifest plateaued or even deteriorated performance in downstream tasks such as node classification or graph clustering.

Example:

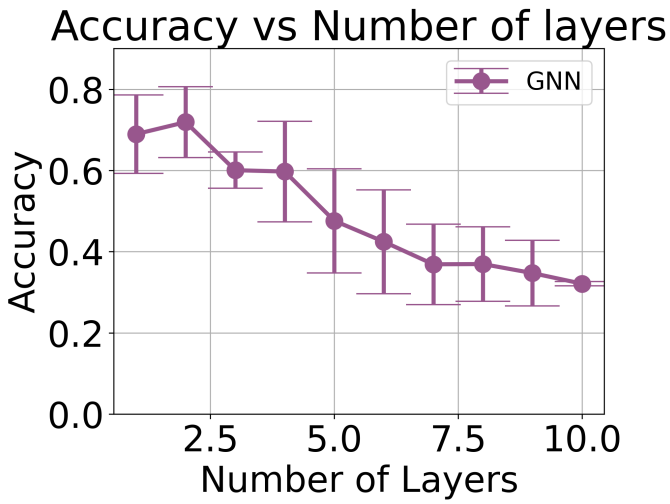


Fig. 2.2 Performance accuracy of a GCN across varying layer depths, showcasing the detrimental effects of oversmoothing.

Figure 2.2 depicts the performance accuracy of a GCN when varying its depth from 1 to 10 layers. The error bars represent the variance across 10 experiments, and the central points denote the mean accuracy for each layer configuration. A notable observation is the decline in performance beyond the depth of 2 layers. This deterioration is attributed to the phenomenon of oversmoothing. The dataset for this experiment originates from the earlier discussed political compass and friendship group example (see Example 2.2). The oversmoothing effect can lead to closely aligned latent representations for fundamentally different voters, such as staunch supporters of rival candidates, even if their relationship in the graph is distant. This convergence of node representations can result in erroneous classifications, thereby undermining the effectiveness of the GNN.

In response to these challenges, the GNN community has diligently pursued solutions to oversmoothing by delving into alternative aggregation approaches, regularization methods, and architectural modifications (Bouritsas et al., 2022; Li and Gupta, 2018). Additionally, we believe (and later show) that Neural Differential Equations on graphs could offer potential alleviations to this issue. The continuous nature of Neural Differential Equations allows for a dynamic adjustment of information flow through the graph, which might preserve node distinctiveness and prevent excessive blending of features—key factors contributing to the oversmoothing phenomenon. This inherent flexibility and continuous adaptability might serve as an effective countermeasure against the typical pitfalls observed with discrete architectures.

Summary

Thus far, we have delved into the intricacies of Neural Differential Equations, encompassing Neural ODEs, Neural CDEs, and Neural SDEs, and we have just covered GNNs, with their inherent limitations of GNNs, including the challenge of over-smoothing. We now transition to our final background chapter on Graph Neural ODEs. Subsequent to this, the forthcoming chapter will introduce our innovative approach: the Graph Neural SDEs.

2.3 Graph Neural Ordinary Differential Equations

Introduced by Poli et al. (2019), Graph Neural Ordinary Differential Equations (GN-ODEs) combine continuous-depth adaptability from the Neural ODEs with graph neural network structure. GN-ODEs meld the structured representation of graph data with the continuous model flexibility, providing a continuum of GNN layers. Compatible with both static and autoregressive GNN models, GDEs afford computational advantages in static contexts using the adjoint method (Chen et al., 2018) and enhance performance in dynamic situations by leveraging the geometry of the underlying dynamics.

2.3.1 Framework

At the heart of GN-ODEs is the representation of the dynamics between layers of GNN node feature matrices

Definition: Inter-layer Dynamics of GNN Node Feature Matrices

$$\mathbf{H}(s+1) = \mathbf{H}(s) + \mathbf{F}_{\mathscr{G}}(s, \mathbf{H}(s), \theta(s)),\ \mathbf{H}(0) = \mathbf{X}_e.$$

In this representation, $\mathbf{X}$ denotes the initial node features of the graph, and $\mathbf{X}_e$ is an embedding derived from various methods such as a single linear layer or another GNN layer. The function $\mathbf{F}_{\mathscr{G}}$ represents a matrix-valued nonlinear function conditioned on graph $\mathscr{G}$, and $\theta(s)$ is the tensor of parameters for the s-th layer.

The fundamental concept of a Graph Neural ODE can be simplified into the subsequent expression:

Definition: Graph Neural Differential Ordinary Equation:

$$\dot{\mathbf{H}}(s) = \mathbf{F}_{\mathscr{G}}(s, \mathbf{H}(s), \theta), \qquad \mathbf{H}(0) = \mathbf{X}_e,$$

where s belongs to a subset real numbers, $\mathbb{R}$, typically denoted as $[t_0, t_1]$.

In the context of GN-ODEs, $\mathbf{F}_{\mathscr{G}}$ functions as a field on graph $\mathscr{G}$ that varies with the depth or complexity of the model, which we refer to as "depth-varying". Depending on the context, the node features might be augmented to improve both computational efficiency and the model's performance, as indicated in prior studies (Poli et al., 2019).

2.3.2 Continuous GCN: Graph Convolution Differential Equations

Having introduced the concept of Graph Neural ODEs, we now turn our attention to continuous representations of some of the prevailing graph neural architectures. Among these, the discrete GCN stands out. As expressed in equation 2.4, it can be defined as

$$\mathbf{H}(s+1) = \mathbf{H}(s) + \sigma\left(\mathbf{L}_{\mathscr{G}}\mathbf{H}(s)\theta(s)\right) \tag{2.6}$$

with $\mathbf{L}_{\mathscr{G}}$ representing the graph Laplacian (Merris, 1995), and σ functioning as the activation. The operation of graph convolution is denoted by $\mathscr{C}_{\mathscr{G}}\,\mathbf{H}(s)$. Its continuous analogue, the *graph convolution differential equation* (GCDE), deploys $\mathbf{F}_{\mathscr{G}}$ to depict a continuous convolution.

Definition: Graph Convolution Differential Equations (GCDE)

$$\dot{\mathbf{H}}(s) = \mathbf{F}_{\texttt{GCN}}(\mathbf{H}(s), \theta) := \mathscr{C}^N_{\mathscr{G}} \circ \sigma \circ \mathscr{C}^{N-1}\mathscr{G} \circ \ldots \circ \sigma \circ \mathscr{C}^1 \mathscr{G}\mathbf{H}(s)$$

The Laplacian, $\mathbf{L}_{\mathscr{G}}$, can be deduced through various methods, and convolution layers of the diffusion type naturally align with this continuous framework.

2.3.3 Message Passing in Graph Neural Differential Equations

Let's delve into the workings of message passing in Graph NDEs. For a given node, represented by $v \in \mathscr{V}$, its neighboring node set can be defined as $\mathscr{N}(v) := \{u \in \mathscr{V} \ : \ (v,u) \in \mathscr{E} \vee (u,v) \in \mathscr{E}\}$. In this context, Message-passing neural networks (MPNNs) perform a spatial-based convolution on node v as follows

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

where typically $\mathbf{h}^v(0) = \mathbf{x}_v$, and both $\mathbf{u}$ and $\mathbf{m}$ are trainable parameter functions. For enhanced clarity, let $\mathbf{u}(\mathbf{x}, \mathbf{y}) := \mathbf{x} + \mathbf{g}(\mathbf{y})$, with $\mathbf{g}$ being the actual parameterized function. This results in

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

and its continuous counterpart can be defined as

Definition: Graph Message Passing Differential Equation

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

2.3.4 Continuous GAT: Graph Attention Differential Equation

We now transition to the continuous representation of GAT. In its discrete form, as outlined in equation 2.5, the GAT can be described as

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

Here, $is a learnable weight vector, and$ is a weight matrix associated with the layer.

The continuous analog of the GAT layer, which we can term as the Graph Attention Differential Equation (GADE), formulates the change in node representations over depth s as a function of their neighboring nodes, modulated by the attention weights. This can be defined as

Definition: Graph Attention Differential Equation (GADE) is defined as

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

with the attention coefficients $\alpha_{vu}(s)$ defined in a manner analogous to the discrete GAT layer.

GADEs leverage the continuous nature of depth to smoothly adjust the attention scores, enabling a more fine-grained extraction of patterns from the graph structure over depth. Furthermore, by using depth as a continuous parameter, GADEs can learn to assign varying importance to neighbors at different depths, possibly capturing multi-scale patterns in the graph data.

2.3.5 Graph Neural ODEs as Continuously-Deep Graph Residual Networks

Having established an understanding of graph neural ODEs, we are now equipped to demonstrate how graph neural ODE can be perceived as continuously-deep graph residual neural network. This notion is analogous to the connection between neural ODEs and ResNets,

where neural ODEs serve as the continuous-depth counterparts of ResNets (refer to Section 2.1.1). In this section, we extend this analogy to graph-structured data, delineating how a Graph Neural ODE can be conceptualized as a continuous-depth version of a Graph Residual Network.

Theorem: Graph neural ODEs operate as continuously-deep graph residual networks.

Proof: Considering the architecture of a residual graph network:

$$y_{j+1} = y_j + f_{\mathscr{G}}(j, y_j, \theta) \tag{2.7}$$

where $f_\Theta(j, y_j, \mathscr{G})$ represents the j-th residual block, with the parameters from all blocks being collectively represented by Θ.

In contrast, let's look at the Graph Neural ODE (abbreviated as GN-ODE):

$$\frac{dy}{dt}(t) = f_{\mathscr{G}}(j, y_j, \theta)$$

Discretizing this GN-ODE using the explicit Euler method at uniformly spaced time intervals t_j with a gap of Δt gives:

$$\frac{y(t_{j+1}) - y(t_j)}{\Delta t} \approx \frac{dy}{dt}(t_j) = f_{\mathscr{G}}(j, y_j, \theta)$$

Simplifying, we get:

$$y(t_{j+1}) = y(t_j) + \Delta t \cdot f_{\mathscr{G}}(j, y_j, \theta)$$

By integrating the factor of Δt into $f_{\mathscr{G}}$, this equation naturally aligns with the formulation in equation 2.7. Such a perspective underscores that neural ODEs can be seen as the continuous-time counterparts of residual networks.

This viewpoint casts the graph neural ODE as a continuously-deep Graph Residual Network. Here, the sequence of minor (residual) updates to its hidden states become both

infinitesimally small and infinitely frequent. The end output is the cumulative effect of these continuous updates, mirroring the solution to the ODE from its initial state.

Remark: As with ODEs, the adjoint method can be employed in the context of graph neural networks (refer to A.1.1). This brings a notable advantage in terms of memory efficiency when comparing Graph Neural ODEs to traditional Graph ResNets. Specifically, while Graph ResNet training necessitates storing activations from every layer for back-propagation, a neural ODE using the adjoint method demands only the storage of its initial state. Subsequently, the ODE solver computes the state at any desired point. This feature can lead to significant memory conservation, especially for ResNets with a high number of layers.

2.3.6 Comparison: Oversmoothing in GN-ODE vs. Standard GNN

This section delves into the robustness of GN-ODEs in combating oversmoothing, using a comparison with the standard GNN as a reference.

Example:

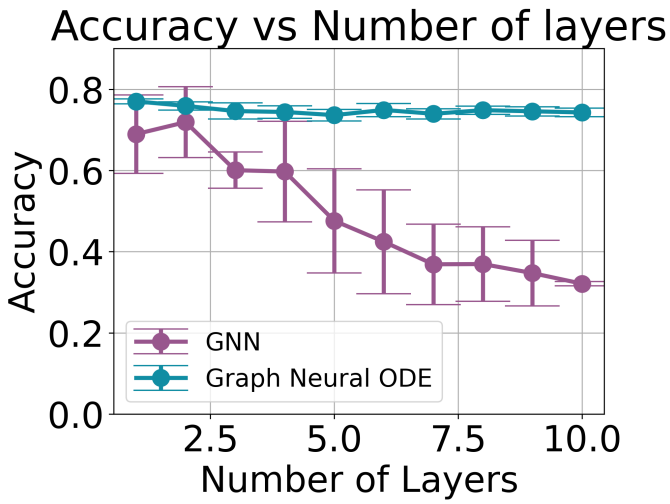


Fig. 2.3 Comparison of performance accuracy between a GCN and Graph Neural ODE over increasing layer depths, highlighting the impact of oversmoothing.

Figure 3.2 offers a side-by-side performance comparison of the standard GCN and the Graph Neural ODE. For this comparison, layer depths range from 1 to 10. Notably, in the context of the Graph Neural ODE, the layer depth corresponds to the number of steps taken within the ODE $H(s)$, effectively representing the depth of the Graph Neural ODE itself. Error bars in the graph represent the variance observed over 10 independent trials, while the central points indicate the mean accuracy for each depth. A standout observation is the Graph Neural ODE's robustness against oversmoothing, a pitfall to which the standard GCN is more vulnerable. The dataset underpinning this evaluation is derived from the previously detailed political compass and friendship group scenario (see Example 2.2). Such findings highlight the empirical benefits of weaving Neural Differential Equations into graph architectures.

Summary

In this journey, we introduced neural differential equations, encompassing Neural ODEs, Neural CDEs, and Neural SDEs. Subsequently, we delved into GNN and expounded on Graph Neural ODEs. Through this exploration, we demonstrated how to implement message passing with Graph Neural Differential Equations. We also translated popular GNN architectures, like CNN and GAT, into their continuous counterparts, resulting in GCDEs and GADEs. Armed with this knowledge, we are now prepared to embark on the premise of this work: our novel model, the Graph Neural SDEs.

Chapter 3

Graph Neural Stochastic Differential Equations

Pursuing the ambition to design a continuous GNN model capable of modeling uncertainty while also addressing the challenges of oversmoothing, we turned our gaze towards Graph Neural ODEs. These continuous models exhibit robustness against oversmoothing. However, they are not adept at capturing the uncertainty inherent in the model. This realization naturally transitioned our focus to Graph Neural Stochastic Differential Equations (GN-SDEs): a continuous model capable at both quantifying its uncertainty and warding off oversmoothing effects.

It is noteworthy that the exploration of uncertainty quantification in Graph Neural Networks has not been as extensive as conventional neural networks. A few seminal contributions in this space include the Bayesian Graph Neural Network by Hasanzadeh et al. (2020), robust ensemble methods (Lin et al., 2022), and Gaussian Processes on graphs (Borovitskiy et al., 2021; Lawrence and Jordan, 2004; Pérez-Cruz et al., 2013). In this work, we use the Bayesian GNN and the ensemble methods for GNN as benchmarks for quantifying uncertainty and out-of-distribution detection. These benchmarks are in conjunction with the previously introduced Graph Neural ODE and GCN, enabling a comprehensive evaluation landscape.

As defined in 2.1.3, a SDE is defined as

$$dz(t) = \underbrace{f(t,z(t))}_{\text{drift}}\,dt + \underbrace{g(t,z(t))}_{\text{diffusion}}\,dW(t).$$

It is important to distinguish between Graph Neural ODEs and Graph Neural SDEs. While the former seamlessly integrates the Neural ODE with a graph model, the latter contends with the inherent complexities of Neural SDEs, as elaborated in the background section 2.1.3. Central to these complexities is the task of properly handling the diffusion function (denoted as g). In Neural SDE models, if the diffusion function is treated as a trainable parameter (like a neural network), it will most definitely trend towards zero when directly trained using the negative log-likelihood (NLL) metrics like cross-entropy (Li et al., 2020). This effectively transforms our stochastic equation into a Neural ODE.

To navigate the challenges posed by uncertainty quantification in GN-SDEs, we introduce two strategies: the **Latent Graph Neural SDE** approach and the **Graph SDE-GAN** methodology. Our latent approach is inspired by the studies by Li et al. (2020). Conversely, Graph SDE-GAN is rooted in the foundational principles of the SDE-GAN paper (Arjovsky et al., 2017), adapting them to graph-centric applications. Within this paradigm, a Graph Neural SDE plays a generative model, while a GNN combined with Neural CDEs plays the role of the discriminative model. The primary application of Graph SDE-GAN is the generation of synthetic data.

In addition to constructing the GN-SDE model, we offer an in-depth theoretical analysis. We present a mathematical proof illustrating the equivalency between GN-SDE to a continuously-deep Graph Residual Neural Network. This relationship grants an intuitive grasp of GN-SDEs, bridging the gap between the continuous-time model and the more familiar, discrete-time deep learning architectures.

3.1 Graph Neural SDE

With the groundwork laid out, we now pivot to the specific implementations and strategies adopted for GN-SDE.

3.1.1 Latent Graph Neural SDEs

Our primary methodology in constructing the Graph Neural SDE revolves around the *Latent Graph Neural SDEs* approach. Latent Graph Neural SDEs learn an latent state $z(t)$ using Graph Neural SDEs to encapsulate the data's underlying patterns. Once determined, this state

is input into a projection network f_Ω to generate predictions $\hat{y}$. The model parameterizes an Ornstein–Uhlenbeck (OU) prior process and an approximate posterior, which is another OU process[1].

Definition:

$$d\tilde{z}(t) = f_\theta(z(t), t, \mathscr{G})dt + \sigma(\tilde{z}_t, t)dW_t$$

The Approximate posterior is expressed as:

$$dz(t) = f_\phi(z(t), t, \mathscr{G})dt + \sigma(z_t, t)dW_t$$

Here, f_ϕ is parameterized by a neural network, with ϕ representing the learned weights of the network.

Both the prior and posterior drift functions, f_θ and f_ϕ respectively, dictate the dynamics of the system. $z(t)$ denotes the system state at time t, and $\mathscr{G}$ symbolizes the graph structure. Notably, both the prior and posterior SDEs employ the same diffusion function σ but have distinct drift functions. Sharing the diffusion function ensures that the KL divergence between these processes remains finite, facilitating its estimation by sampling paths from the approximate posterior process (Li et al., 2020).

The evidence lower bound (ELBO) is given by

$$\mathscr{L}_{ELBO}(\phi) = \mathbb{E}_{z_t}\left[\log(p(x_{t_i}|z_{t_i}) - \int_{t_0}^{t_1} \frac{1}{2}||u(z_t, t, \phi, \theta, \mathscr{G})||_2^2 dt\right],$$

where x_{t_i} are the observations at time t (with i in $[t_0, t_1]$), and

$$u = g(z_t, t)^{-1}[f_\phi(z_t, t, \mathscr{G}) - f_\theta(z_t, t, \mathscr{G})].$$

The latent state z_t, is then passed into the projection layer, f_Ω, for further prediction. The posterior predictive is then:

$$p(y^*|t^*, \mathscr{G}, \mathscr{D}) = \int p\left(y^*|f_\Omega(z_t, t^*, \mathscr{G})\right) p(z|\mathscr{D})\, \mathrm{d}z.$$

[1]For an in-depth definition of the OU process, refer to A.2.4

This posterior predictive distribution is approximated using Monte Carlo sampling by drawing samples z_n from the posterior $p(z|\mathscr{D})$. Therefore, we can represent it as

$$p(y^*|t^*,\mathscr{G},\mathscr{D}) \approx \frac{1}{N}\sum_{n=1}^{N} p\left(y^*|f_\Omega(z_n^*,t^*,\mathscr{G})\right).$$

The variance, of the Monte Carlo mean estimation, is given by

$$\mathrm{Var}(y) = \frac{1}{N}\sum_{n=1}^{N}(y_n-\bar{y})^2.$$

3.1.2 Graph SDE-GAN

The second approach we explore is the Graph SDE-GAN. This technique extends the SDE-GAN methodology presented by Kidger et al. (2021) to work with graph-structured data.

We begin by considering the Graph Neural SDE, which serves as the generative function in the GAN. The differential equation for this process is given as

$$dy(t) = f_\phi(t,y(t))\,\mathrm{d}t + g_\theta(t,y(t))\,\mathrm{d}W(t).$$

Here, $y(s)$ is the predicted output for $y_{\text{true}}(s)$ at time s. Notably, we use y instead of z to emphasize that we are directly predicting the output from the input.

The goal of training this model is to ensure that the distribution of y closely resembles that of y_{true}. Specifically, we aim for y to approximate y_{true} in a precise sense—measured here by the Wasserstein distance, as detailed in (Arjovsky et al., 2017; Goodfellow et al., 2014).

In this setup, y is a random variable whose distribution is implicitly controlled by parameters θ. To effectively train the model, we must fit it to data, optimizing a suitable measure of distance between the probability distributions of y and y_{true}. We employ the Wasserstein distance for this purpose and train a discriminator adversarially. We denote this model by $F(Y)$, defined as follows

$$F(Y) = m_\Theta \cdot H(t),$$

where

$$H(0) = y(0),$$
$$dH(t) = f_\phi(t, H(t), \mathscr{G})\,\mathrm{d}t + g_\theta(t, H(t), \mathscr{G})\,\mathrm{d}W(t).$$

In this context, $\cdot$ denotes the dot product. The functions f_ϕ, g_θ, and q_τ are neural networks, and m_Θ is a vector. Each of these components—θ, ϕ, and Θ—are learnable parameters in our model.

The training objective can then be expressed as the following optimization problem:

$$\min_\theta \min_\phi \left(E_Y[F_\phi(Y)] - E_{Y_{\text{true}}}[Y_{\text{true}}] \right).$$

Note that, because we are not training the model via maximum likelihood, we are free to use separate neural networks for the drift function f_ϕ and the diffusion function g_θ without the diffusion function learn to become zero, although theoretically, it can. Once the Graph Neural SDE is trained, we can sample from this learned model to produce synthetic data. This model was not designed for predictive tasks but rather for the generation of synthetic data.

> **Remark:** With a comprehensive understanding of the Graph SDE-GAN approach, we are now equipped to employ this model for synthetic data generation. This capability enriches our data-driven efforts and enhances our ability to simulate real-world phenomena in controlled environments, thus paving the way for groundbreaking research and applications.

3.2 Graph Neural SDEs as Continuously-Deep Recurrent Graph Neural Networks

Having introduced the Graph Neural SDE, and given our prior discussions on Neural SDEs where we established that they serve as continuous-depth analogues of RNNs (refer to 2.1.3), we now venture further into this terrain. Specifically, our next logical step is to show this still holds for graph. In essence, we suggest that a Graph Neural SDE should be viewed

as a continuous-depth version of a Recurrent Graph Neural Network (RGNN). To draw a parallel, within the framework of Neural SDEs, the input for the RNN is typically interpreted as random noise or Brownian motion, and its output mirrors that of a generated sample.

Theorem: Graph Neural SDEs operate as continuously-deep RGNN.

Proof: Consider an autonomous one-dimensional Stochastic Differential Equation represented as

$$dy(t) = f(y(t),\mathscr{G})dt + \sigma(y(t))dw(t),$$

where $y(t)$, $f(y(t),\mathscr{G})$, $\sigma(y(t))$, and $w(t)$ belong to the set of real numbers, $\mathbb{R}$. The numerical Euler-Maruyama discretization of this SDE can be expressed as

$$\frac{y(t_{j+1}) - y(t_j)}{\Delta t} \approx f(y(t_j),\mathscr{G}) + \frac{\sigma(y(t_j))\Delta w_j}{\Delta t}.$$

Which simplifies to

$$y_{j+1} = y_j + f(y_j,\mathscr{G})\Delta t + \sigma(y_j)\Delta w_j.$$

Here, Δt represents a fixed time step and Δw_j is normally distributed with mean zero and variance Δt. This numerical discretization is reminiscent of an RNN with a specific form. Therefore, we can consider the Neural SDE as an infinitely deep RNN where the depth is defined by the number of discretization steps.

Remark: Deep learning designs, like the Graph Neural SDE, appear to have a strong resemblance to differential equations. At first glance, this may seem coincidental. However, considering the increasing popularity of this approach, there's an interesting link suggesting it's not all that surprising, especially when we acknowledge the longstanding use of differential equations in modeling. It makes sense that they influence new areas like deep learning. This shows how adept they are at describing intricate situations.

Summary

We have successfully introduced our novel model, the Graph Neural SDEs. Within this framework, we proposed two methodologies: the Latent Graph Neural SDE for standard

regression and classification tasks, and the Graph SDE-GAN for generating synthetic data (in a stochastic manner) from graph structures. Furthermore, we demonstrated that under the Euler-Maruyama discretization (an approximation method for SDEs), this model can be interpreted as continuously-deep RGNNs. With a foundational understanding of our innovative model in place, we will now transition to its evaluation on both static and spatio-temporal datasets. The next section will spotlight related work, serving as benchmarks for our model. Additionally, we will delve into the capacity of our continuous model to assess prediction uncertainties and its inherent resistance to over-smoothing.

3.3 Evaluation

To highlight the capabilities of the Graph Neural Stochastic Differential Equations (SDEs), we conducted a series of experiments on both static and spatio-temporal datasets. We initiated our evaluation with a basic static toy dataset, offering clear visualization and a tangible representation of our model's strengths. Following this, we delved into experiments on real-world datasets and explored the integration of active learning techniques. In each scenario, the performance of Graph Neural SDEs was contrasted against established benchmarks like Graph Neural ODEs and Graph Convolutional Neural Networks (GCNs). Additionally, we demonstrated the application potential of our Graph SDE-GAN model in generating synthetic SDE features for graphical data.

In the experiments involving both static and spatio-temporal datasets, we utilized the Latent Graph Neural SDE. For the sake of simplicity in presentation and discourse, we commonly referred to it as the Graph Neural SDE or GN-SDE.

3.3.1 Static

Consider the same scenario as in Example 2.2, where the objective is to predict the voting preferences of individuals for three candidates (red, blue, or green), based on their political compass (left, right, liberal, authoritarian) and their social circles (this is a node calcification problem). The right panel of Figure 3.1 depicts this problem, illustrating the voters along with their respective political compasses. The right panel of the figure shows the social groups or friend circles of these voters. It is important to note that there is inherent noise in the data. For instance, individuals often maintain friendships with those who have voted for different

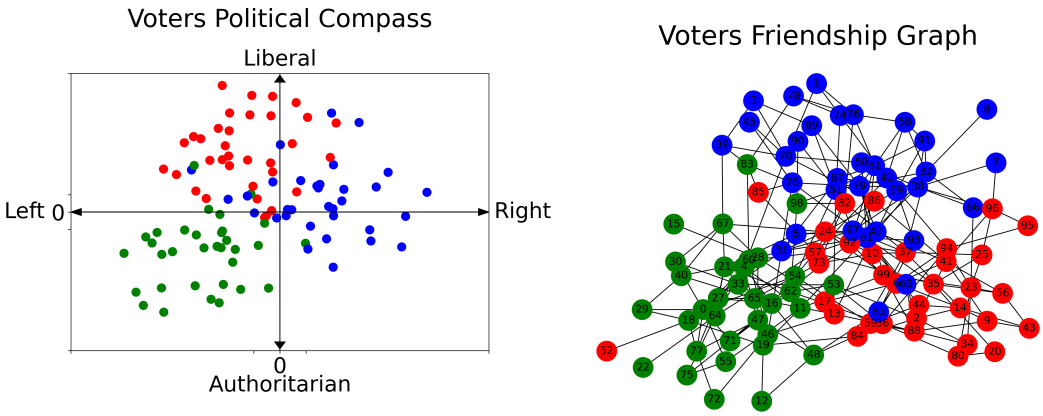


Fig. 3.1 The left image illustrates the political compass of voters while the right image presents their social circles, with colors indicating the candidates they voted for.

candidates. Similarly, the alignment of a voter's political compass does not always strictly dictate their voting preference, introducing an element of randomness.

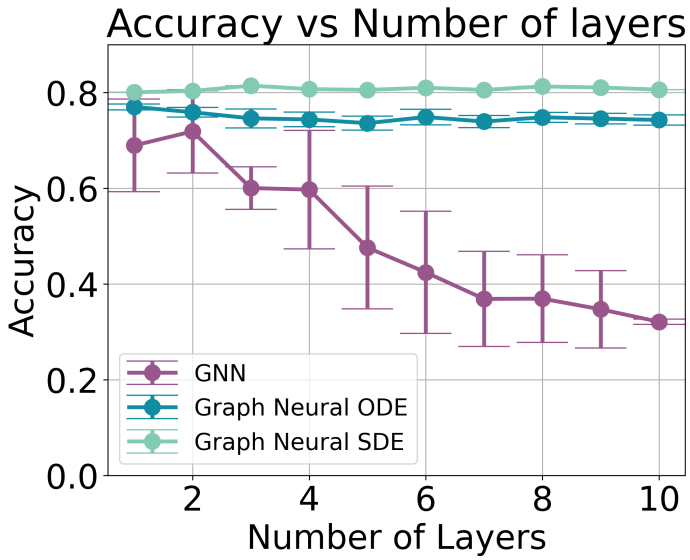
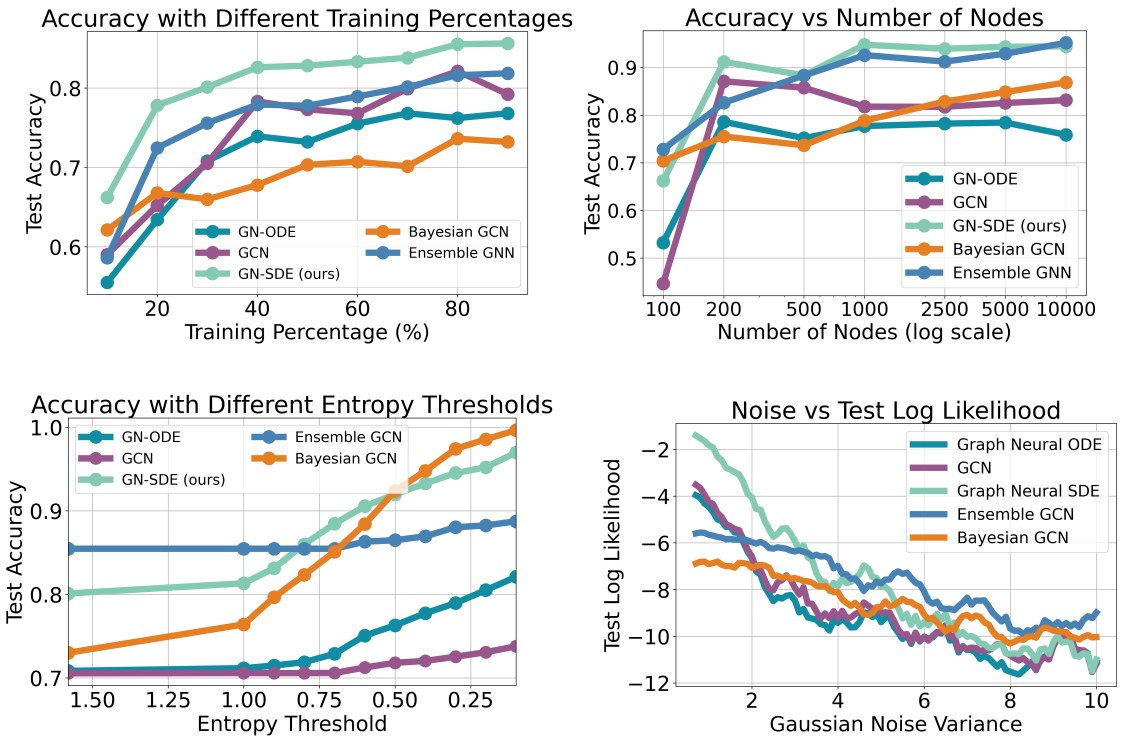


Fig. 3.2 Comparison of performance accuracy between a GCN, Graph Neural ODE, Graph Neural SDE over increasing layer depths, highlighting the impact of oversmoothing.

We first examine the resilience of the models to oversmoothing as the number of layers increases. Figure 3.2 provides a performance comparison between the standard GCN, Graph Neural ODE, and Graph Neural SDE across layer depths from 1 to 10. In the case of the Graph Neural ODE and SDE, "layer depth" denotes the number of steps taken within the differential equation to solve the integraph. This effectively captures the depth of these neural architectures. The error bars on the graph show variance over 10 independent trials,

with central points representing the mean accuracy at each depth. A key takeaway is the heightened resistance of the Graph Neural ODE and SDE to oversmoothing, a challenge that more prominently affects the standard GCN.



.

Fig. 3.3 A comprehensive comparison across Graph Neural ODE, GCN, Graph Neural SDE, Bayesian GCN, and Ensemble GCN models, illustrating performance dimensions across accuracy, scalability, prediction confidence, and resilience against noise.

Figure 3.3 offers a thorough analysis of five distinct models. These include the Graph Neural Ordinary Differential Equations (GN-ODE), GCN, our proposed Latent Graph Neural Stochastic Differential Equations (GN-SDE), Bayesian GCN, and Ensemble GCN — where the latter averages predictions from five individual GCNs. Four evaluation metrics inform this comparison, each presented in its corresponding sub-figure.

Accuracy in Relation to Training Data Proportions: We evaluated the model's accuracy by training it on different percentages of the dataset, ranging from 10% to 90%. The remaining data was allocated for testing. Across all these variations, the GN-SDE consistently surpassed the performance of other models. This underscores the data efficiency of our model.

Accuracy versus Number of Nodes: Next, we measured performance in relation to graph node count. The GN-SDE showed high performance even with fewer nodes, maintaining this lead as node numbers grew. This suggests the model's versatility in managing both small and large graphs. In the specific scenario with only 100 nodes, the Ensemble GCN briefly surpassed our model. Otherwise, the GN-SDE generally dominated.

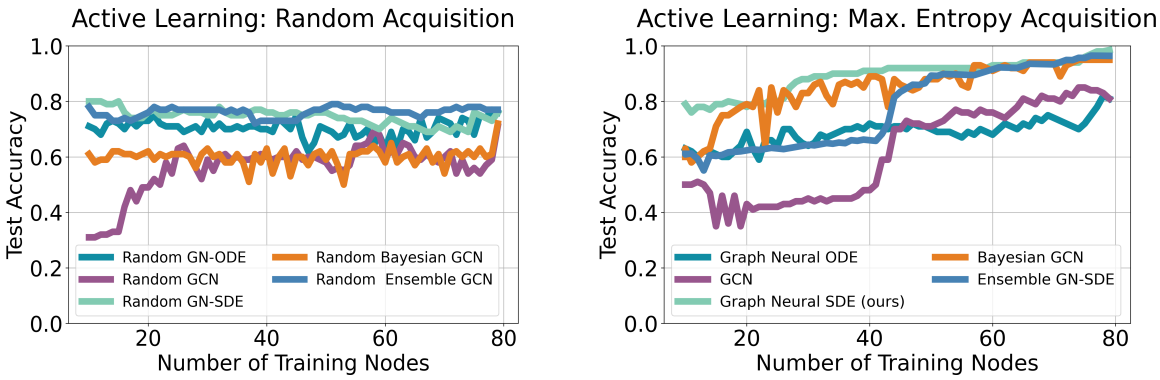
Accuracy versus Entropy Threshold: This metric probed the models' capacity for confident predictions by setting an entropy threshold. Only when prediction entropy falls below this threshold is the model allowed to predict on test data. Lower entropy indicates a higher prediction confidence. At an entropy of 1.60, the Ensemble model outdid ours, but as confidence requirements tightened (e.g., at entropy 0.9 onwards), our model beat the Ensembel GCN model, only surpassed by the Bayesian GCN after a threshold of 0.5. This signifies our model's adeptness at identifying out-of-distribution data and providing reliable uncertainty measures, bested only by the Bayesian GNC.

Noise versus Log-Likelihood: We define log-likelihood as the logarithm of the probability assigned to the correct class, and this metric was assessed as the noise level increased. It's important to note that a log-likelihood of 0 (indicating that the model assigns a probability of 0 to the correct class) would lead to negative infinity when taking the logarithm[2]. Our evaluations centered on the model's resilience to Gaussian noise introduced in the test data. As the noise escalated, our model's performance demonstrated a gradual and 'smooth degradation', especially when juxtaposed with the GCN and Graph Neural ODE. Although the Bayesian and Ensemble models fared better under extreme noise conditions, the GN-SDE model's performance degradation was the most graceful among all competitors, highlighting its robustness even under challenging circumstances.

It is worth emphasizing that our GN-SDE model consistently excelled over both the Graph Neural ODE and the GCN in every evaluation setting. Only on isolated occasions did the Bayesian GCN and the Ensemble model outshine ours. Overall, the GN-SDE demonstrated superior performance, particularly in terms of robustness, data efficiency, and uncertainty quantification. For a detailed examination of results pertaining to this dataset, refer to Appendix B.1.

Figure 3.4 illustrates an active learning experiment conducted on a dataset composed of 100 nodes. The experiment commenced with an initial training set of 10 nodes, with additional nodes incrementally incorporated one at a time. In the right figure, the node with the highest

[2]To prevent numerical issues associated with taking the logarithm of 0, we added a small constant 10^{-4} to all probabilities before computing the log-likelihood



(a) Node selection using a random acquisition function.

(b) Node selection using a maximum entropy acquisition function.

Fig. 3.4 The figure depicts an active learning experiment on a 100-node dataset, starting with 10 nodes and incrementally adding more until reaching 80. The left and right figures use random and max entropy acquisition functions respectively for node selection.

entropy was strategically selected, while in the left, node selection was based on a random acquisition function. After the inclusion of each new node, the model was retrained over a course of 5 epochs. This procedure was consistently followed until the training set expanded to encompass 80 nodes. Notably, our model achieved a remarkable 99% accuracy upon the addition of the 78th node, a performance closely paralleled by the Bayesian and Ensemble models. In stark contrast, the GCN and Graph Neural ODE models under examination could only attain a peak accuracy nearing 85%. This experiment illustrates the proficiency of our model in an active learning scenario, a context where it is crucial for a model to be able to select the next training example when dealing with limited and costly labeled datasets. This attribute is invaluable in situations where labeled training data is scarce and labeling new data is expensive. Additionally, compared this with the random acquisition method—which could not elevate the models' performance beyond 80% accuracy—shows the importance of employing uncertainty-aware acquisition strategies in amplifying the models' performance.

Real-Word Static Data sets

Shifting our focus to real-world datasets, Table 3.1 details the performance of the models GN-SDE, GN-ODE, GCN, Ensemble GCN, and Bayesian GCN on renowned datasets such as CORA (Sen et al., 2008), Pubmed (Sen et al., 2008), Citeseer (Giles et al., 1998), and OGB arXiv (Hu et al., 2020). The models were evaluated based on different entropy thresholds. This essentially means that a model is only permitted to make predictions if its confidence level, as

Dataset	Models	Entropy Thresholds											
		∞	1.6	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
CORA	**GN-SDE (ours)**	**0.817**	**0.834**	0.922	0.942	**0.957**	0.966	0.969	0.977	**0.991**	**1.0**	**1.0**	**1.0**
	GN-ODE	0.799	0.806	0.905	0.923	0.944	**0.969**	**0.976**	**0.984**	0.984	0.995	**1.0**	**1.0**
	GCN	0.717	0.717	0.720	0.720	0.723	0.734	0.756	0.761	0.771	0.780	0.786	0.824
	Ensemble GNN	0.777	0.802	**0.935**	**0.949**	0.954	0.958	0.962	0.972	0.983	**1.0**	**1.0**	**1.0**
	Bayesian GNN	0.709	0.719	0.800	0.834	0.871	0.893	0.917	0.925	0.930	0.948	0.972	0.981
Citeseer	**GN-SDE (ours)**	0.71	**0.753**	**0.879**	0.889	0.898	0.925	**0.929**	0.924	0.926	**0.947**	**0.972**	**1.0**
	GN-ODE	**0.712**	0.742	0.875	**0.891**	**0.905**	**0.931**	0.915	**0.936**	**0.930**	0.882	0.923	**1.0**
	GCN	0.516	0.516	0.524	0.530	0.544	0.557	0.583	0.599	0.626	0.651	0.678	0.725
	Bayesian GCN	0.61	0.619	0.729	0.746	0.769	0.791	0.815	0.821	0.854	0.863	0.867	0.88
	Ensemble GNN	0.527	0.532	0.743	0.780	0.821	0.834	0.859	0.858	0.920	0.939	0.953	0.933
Pubmed	**GN-SDE (ours)**	**0.791**	**0.791**	**0.794**	0.794	0.802	0.818	0.837	0.852	0.876	0.893	0.898	0.911
	GN-ODE	0.763	0.763	0.768	0.774	0.781	0.813	0.833	0.847	0.858	0.862	0.872	0.899
	GCN	0.78	0.78	0.784	0.785	0.789	0.795	0.809	0.821	0.823	0.835	0.847	0.864
	Ensemble GNN	0.786	0.786	0.796	**0.814**	**0.837**	**0.868**	**0.879**	**0.908**	**0.907**	**0.916**	**0.929**	**0.952**
	Bayesian GNN	0.715	0.715	0.719	0.730	0.736	0.752	0.815	0.850	0.864	0.882	0.904	0.918
OGB arXiv	**GN-SDE (ours)**	**0.531**	0.770	0.859	0.871	0.884	0.897	0.907	0.916	0.929	0.944	0.955	0.968
	GN-ODE	0.526	0.766	0.853	0.867	0.883	0.898	0.912	0.919	0.930	0.938	0.951	0.964
	GCN	0.470	**0.809**	**0.885**	**0.900**	0.909	0.917	0.931	**0.939**	**0.950**	**0.966**	**0.983**	0.976
	Ensemble GNN	0.512	0.785	0.699	0.800	**1.000**	**1.000**	**1.000**	-	-	-	-	-
	Bayesian GNN	0.433	0.828	0.880	0.884	0.893	0.897	0.900	0.919	0.916	0.935	0.954	**1.000**

Table 3.1 Accuracy scores for GN-SDE, GN-ODE, GCN, Ensemble GNN, and Bayesian GNN on CORA, Citeseer, Pubmed, and OGB arXiv datasets. Comparisons use varying entropy thresholds, with bold values indicating top performance. A '-' for accuracy indicates the model lacked sufficiently confident data points at that threshold.

measured by entropy, falls below a predefined threshold. Using entropy as a measure provides an understanding of a model's capacity for confident predictions (uncertainty quantification) and its ability in performing out-of-distribution detection.

To clarify further, a lower entropy value signifies higher model confidence; conversely, a higher value indicates lesser certainty. Importantly, an entropy of inf suggests the model's continuous prediction irrespective of its confidence level, implying it always outputs a prediction regardless of how uncertain it might be. In instances where the accuracy is denoted with a '-' in the table, this represents a unique scenario. It means that the model found all potential data points too ambiguous to confidently classify under the given entropy threshold. In other words, none of the data points satisfied the model's criteria for a confident prediction at that specific threshold, indicating a heightened level of uncertainty in the model's assessment of the data.

Upon examining the table, it becomes clear that our GN-SDE model consistently surpasses the other four models across most entropy thresholds, especially in the CORA and Pubmed datasets. This consistent top-tier performance further substantiates our model's

aptitude for detecting out-of-distribution data. Specifically, in the CORA dataset, GN-SDE shines as the top performer across nearly all entropy thresholds, achieving a perfect 100% accuracy at thresholds 0.3, 0.2, and 0.1. While the Ensemble GCN matched this feat at the same thresholds, GN-SDE maintains a higher accuracy at elevated entropy levels. Similarly, for the Pubmed dataset, GN-SDE holds its ground across the majority of thresholds, registering an impressive 91.1% accuracy at an entropy threshold of 0.1.

When examining the Citeseer and OGB arXiv datasets, the performance of our GN-SDE model seems more evenly matched with other models, rather than being the clear leader as observed previously. In the OGB arXiv dataset, the GN-SDE initiates with a strong performance at entropy inf. However, it faces stiff competition from the Bayesian GNN which achieves an impressive 100% accuracy at the entropy threshold of 0.1. Noteworthy is the Ensemble GNN's performance, which flaunts a flawless 100% accuracy at entropy thresholds of 0.8 and below, although it stops predicting beyond an entropy of 0.5. This underscores the model's high confidence in these thresholds. Interestingly, the GCN model consistently outstrips both GN-SDE and Graph Neural ODE across several entropy thresholds. Shifting to the Citeseer dataset, GN-SDE leads in performance across most thresholds. Nevertheless, GN-ODE emerges as a close second, occasionally surpassing our model at specific thresholds.

Overall, the GN-SDE model showcases commendable performance. Although there's variability across datasets and thresholds, the results affirm its robustness and exceptional ability in confident predictions, adeptly handling out-of-distribution data.

3.3.2 Spatio-Temporal

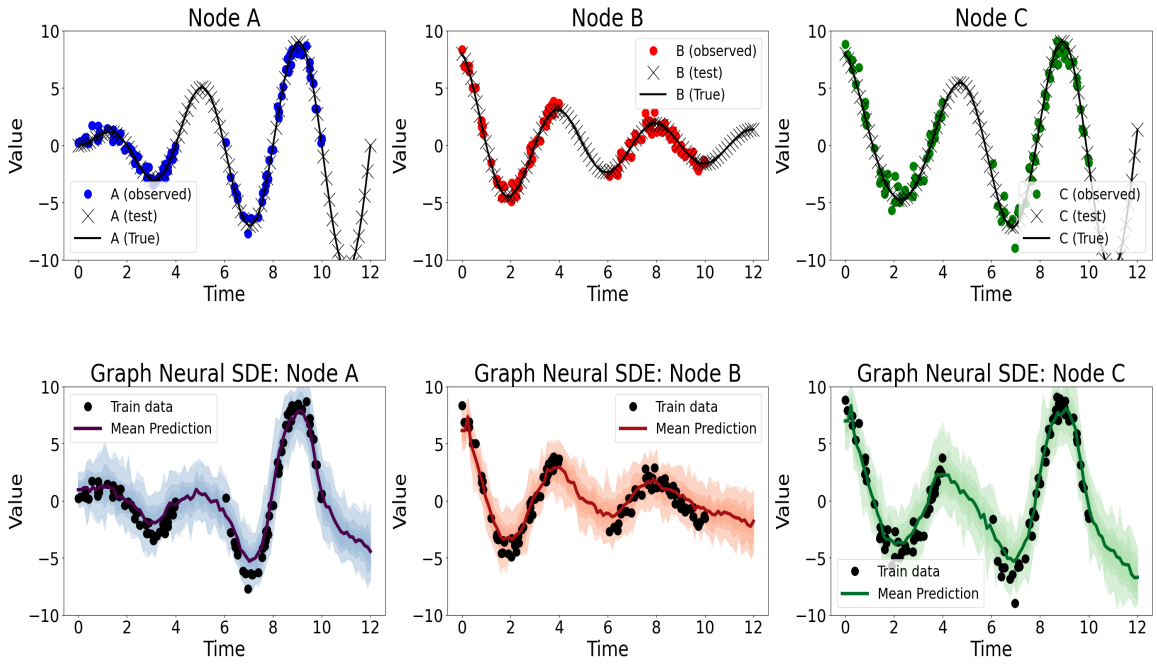


Fig. 3.5 Comparison of the true and predicted values for the three-node regression problem. The first row shows the true values of nodes A, B, and C over time. The second row presents the predictions made by the Graph Neural SDE model for nodes A, B, and C. The shaded regions represent the model's uncertainty quantification, demonstrating an increase in uncertainty during the interpolation and extrapolation phases.

Moving into a spatial-temporal dataset, let's consider a scenario involving a three-node regression problem with nodes A, B, and C. The objective is to predict the regression values of these nodes at various time points. As depicted in Figure 3.5 and 3.6, the observations are irregularly sampled, and the graph structure reveals that node C is connected to nodes A and B, while nodes A and B are independent, i.e., they do not share a direct link. The figure also delineates the training and testing data points, demonstrating both interpolation (particularly in the time frame between 4 and 6) and extrapolation (from time 10 to 12).

The underlying true distribution for each node is given by

$$A(t) = t * \sin\left(\frac{\pi t}{2}\right) + \varepsilon_A,$$
$$B(t) = \frac{4}{\frac{t}{5} + 0.5} * \cos\left(\frac{\pi t}{2}\right) + \varepsilon_B,$$
$$C(t) = A(t) + B(t) + \varepsilon_C,$$

where $\varepsilon_A, \varepsilon_B, \varepsilon_C \sim \mathcal{N}(0, 0.5^2)$. It is important to note that while nodes A and B are independent, node C is a function of both A and B, with an added Gaussian noise component.

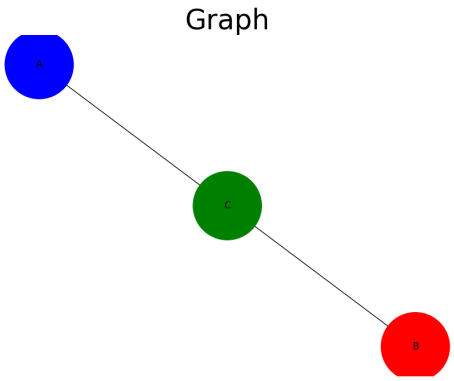


Fig. 3.6 Represents the graph structure, showing that node C is connected to nodes A and B, while nodes A and B are independent.

In this task, our objective is to incorporate uncertainty to filter out predictions during the testing phase. However, both Neural ODE and GCN models are deterministic. Unlike in classification tasks where entropy can be utilized as a measure of uncertainty, an equivalent metric is absent for regression tasks. To circumvent this, we employ Monte Carlo Dropout during both the training and testing phases for these two models. MC Dropout imparts stochasticity into the model, and it has been demonstrated that under certain conditions, neural networks with dropout converge to a Gaussian Process in the limit (Gal and Ghahramani, 2016). This approach enables us to estimate the model's uncertainty, thereby providing a mechanism to discard unreliable predictions. The uncertainty is quantified by predicting both a mean and a variance, with the magnitude of the variance serving as the measure of uncertainty. Consequently, a prediction is only made if the model's variance satisfies a predetermined threshold. It's worth noting that we do not need to use dropout for the Bayesian GCN and Ensemble methods, since they inherently quantify uncertainty: the Ensemble derives the mean and variance of regression predictions, while for the Bayesian GCN, we utilize Monte Carlo Sampling to ascertain the prediction's mean and variance, similar to our GN-SDE's approach.

The second row in figure 3.5 presents the performance of the Graph Neural SDE model in the node regression task. The figure highlights the model's ability to quantify the inherent uncertainty associated with the randomness of the model. Particularly, it demonstrates an increase in uncertainty during both the interpolation and extrapolation phases. A comparative plot, which includes the Dropout Graph Neural ODE model and the Dropout GCN model, is provided in the appendix (see Figure B.1). It is evident from this comparison that the

application of dropout does indeed facilitate effective uncertainty quantification, particularly during interpolation and extrapolation. However, it is worth noting that the models exhibit limited success in capturing the uncertainty associated with the training data points.

Metric	Models	Variance Thresholds					
		100	2.5	2	1.5	1.0	0.5
MAE	Dropout GCN	13.35	13.06	13.03	13.37	12.24	5.63
	GN-SDE (ours)	12.06	11.36	10.46	9.73	2.41	-
	Dropout GN-ODE	14.52	14.52	14.52	14.49	13.61	-
	Bayesian GCN	10.99	11.02	12.16	9.12	4.25	4.26
	Ensemble GCN	**2.67**	**2.68**	**2.74**	**2.72**	**2.62**	**2.74**
MAPE	Dropout GCN	9.34	9.39	**9.46**	14.08	**8.94**	3.06
	GN-SDE (ours)	**6.83**	**6.32**	13.08	**3.38**	13.55	-
	Dropout GN-ODE	8.91	8.91	8.91	10.67	9.96	-
	Bayesian GCN	7.50	7.59	**9.41**	7.70	7.24	**2.73**
	Ensemble GCN	15.44	141.20	25.15	54.49	18.31	8.61
MSE	Dropout GCN	13.35	13.06	13.03	13.37	12.24	5.63
	GN-SDE (ours)	12.06	11.36	**10.46**	**9.73**	**2.41**	-
	Dropout GN-ODE	14.52	14.52	14.52	14.49	13.61	-
	Bayesian GCN	**10.99**	**11.02**	12.16	9.12	4.25	**4.26**
	Ensemble GCN	12.73	12.72	13.27	13.28	12.55	13.47
NLL	Dropout GCN	22.55	25.07	25.29	26.74	26.66	19.81
	GN-SDE (ours)	**8.97**	**9.03**	**9.53**	**10.30**	8.44	-
	Dropout GN-ODE	40.19	40.19	40.19	42.04	43.97	-
	Bayesian GCN	13.28	13.35	12.28	12.01	**6.96**	**10.00**
	Ensemble GCN	51.38	85.56	81.54	89.95	104.68	205.54

Table 3.2 Performance comparison of Dropout GCN, GN-SDE, Dropout GN-ODE, and Bayesian GCN models on the three-node regression problem across different variance thresholds. Bold values indicate superior performance by the model. A '-' for accuracy indicates the model lacked sufficiently confident data points at that threshold.

Table 3.3 provides a comprehensive performance comparison of several models across different variance thresholds. Notably, the GN-SDE model demonstrates consistently strong results, underscoring its predictive accuracy and adeptness at uncertainty estimation.

The GN-SDE model frequently achieves the lowest Mean Absolute Error (MAE), Mean Squared Error (MSE), and Negative Log-Likelihood (NLL) across the higher variance thresholds. However, its performance in Mean Absolute Percentage Error (MAPE) demonstrates

some variability. For a more in-depth understanding of the metrics MAE, MSE, and MAPE, please refer to Appendix A.3.

For the NLL metric, calculated under the Gaussian-distributed predictions assumption

$$NLL = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\log(2\pi\sigma_i^2) + \frac{(y_i-\mu_i)^2}{2\sigma_i^2}\right),$$

where σ^2 is the predicted variance, μ is the observations, and $\hat{\mu}$ is the predicted mean.

Interestingly, as the variance threshold reduces, the Bayesian GCN begins to show competitive results, particularly evident at the stringent variance threshold of 0.5, where it even surpasses the GN-SDE in the MAE metric. The Ensemble GCN exhibits varied performance across the thresholds but remains a consistent contender in the comparison.

The GN-SDE model exhibits a significant decrease in MAE and MSE as the variance threshold decreases, reinforcing its ability to discard less reliable predictions. At the most restrictive threshold of 0.5, the Dropout GCN and Bayesian GCN are the only models providing results, with the Bayesian GCN showcasing impressive MAE and NLL scores.

To summarize, the GN-SDE model emerges as a potent tool in the three-node regression task, often outperforming its counterparts across multiple variance thresholds. The Bayesian GCN warrants a special mention for its robust results, especially at the lower variance thresholds. While the Dropout GCN displays resilience at the most stringent threshold, the Dropout GN-ODE model seems to face challenges at higher variance thresholds, indicating possible constraints in its uncertainty estimation capabilities. The Ensemble GCN, though consistent, occasionally exhibits suboptimal results, especially in the NLL metric.

Real World Datasets

Now we move to real-world spatio-temporal datasets. We focused our attention on two specific datasets: PEMS-BAY and METR-LA, as discussed in Li et al. (2018b).

The predictive modeling of traffic, a quintessential spatiotemporal phenomenon, generally takes the form of a time-series prediction problem. Here, the primary objective is to predict future traffic metrics (for instance, traffic speed or traffic flow) over the next H time steps given

a historical record of traffic observations over the previous M time steps. Mathematically, this prediction can be expressed as:

$$\hat{v}_{t+1}, \ldots, \hat{v}_{t+H} = \arg \max_{v_{t+1}, \ldots, v_{t+H}} \log P(v_{t+1}, \ldots, v_{t+H} | v_{t-M+1}, \ldots, v_t),$$

In this formulation, $v_t \in \mathbb{R}^n$ denotes an observation vector for n road segments at time step t, with each element of the vector representing historical observations for an individual road segment (Yu et al., 2017).

Among the datasets analyzed, the METR-LA traffic dataset is a widely utilized resource for traffic speed prediction. This dataset comprises traffic data gathered from loop detectors installed on highways across Los Angeles County. A total of 207 sensors were selected for this study, yielding a dataset that encompasses four months of data spanning from March 1st to June 30th, 2012. This rich dataset provides a comprehensive view of traffic dynamics over a considerable temporal and spatial scale.

In our experiments, we utilized Graph Attention Networks (GAT) to embed the input data, which consisted of the past six recordings. These inputs were embedded into 64-dimensional tensors. Subsequently, these embeddings were passed to our Latent Graph Stochastic Differential Equation (SDE) model. The Graph Latent SDE model employs a GCN for the drift function, while the diffusion function is held constant at a value of 1. The hidden state of the model is of size 64, which is then passed to a GCN projection layer for prediction.

The Graph Neural ODE model follows a similar structure but replaces the SDE with an ODE and omits the noise component. The GCN model also utilizes the same embedding and projection layers but bypasses the differential equations entirely. This setup allows for a fair comparison between the models, as they share the same embedding and projection layers, differing only in the differential equation component.

From the results presented in Table 3.3, the Graph Neural Stochastic Differential Equations (GN-SDE) model consistently outperforms the Dropout GAT-GCN, Dropout GN-ODE, Bayesian GCN, and Ensemble GCN models across all variance thresholds. For the variance thresholds of 100, 3, and 1, the GN-SDE model achieved the lowest Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Root Mean Squared Error (RMSE), and Root Negative Log-Likelihood (RNLL). This achievement indicates the GN-SDE model's superior predictive accuracy and capability in uncertainty estimation.

Metric	Models	Variance Thresholds				
		100	3	1	0.5	0.25
MAE	Dropout GAT-GCN	14.30	14.01	**9.38**	5.25	5.22
	GN-SDE (ours)	**14.13**	**12.42**	10.14	**5.40**	**4.82**
	Dropout GN-ODE	15.21	15.53	12.36	9.37	5.58
	Bayesian GCN	15.0	13.9	9.4	5.3	5.2
	Ensemble GCN	9.835	6.665	2.615	2.736	-
MAPE	Dropout GAT-GCN	**10.21**	10.59	20.42	21.01	20.87
	GN-SDE (ours)	10.63	**9.66**	**18.74**	**18.79**	**13.25**
	Dropout GN-ODE	10.27	13.92	21.86	17.14	13.70
	Bayesian GCN	10.5	10.6	20.5	20.9	20.8
	Ensemble GCN	18.15	15.44	18.31	8.61	-
RMSE	Dropout GAT-GCN	18.58	16.39	9.58	5.26	5.22
	GN-SDE (ours)	**15.38**	**13.17**	**10.12**	**5.22**	**4.78**
	Dropout GN-ODE	19.73	17.32	16.68	12.87	7.33
	Bayesian GCN	19.0	16.5	9.6	5.3	5.2
	Ensemble GCN	19.74	12.73	7.55	6.47	-
RNLL	Dropout GAT-GCN	7.03	13.28	15.12	11.56	14.52
	GN-SDE (ours)	**6.56**	**11.57**	**13.54**	**13.92**	**13.88**
	Dropout GN-ODE	32.33	75.10	31.03	92.42	213.09
	Bayesian GCN	7.2	13.4	15.3	12.0	14.6
	Ensemble GCN	39.63	51.38	104.68	205.54	-

Table 3.3 Comparative performance of various models on the METER-LA Dataset across different variance thresholds. Bold values indicate superior performance by the model.

As the variance threshold decreases further to 0.5 and 0.25, the GN-SDE model continues to maintain its leading performance, securing the lowest MAE, MAPE, RMSE, and RNLL scores. This persistence in performance suggests that the GN-SDE model's predictions align more closely with the observed data and that the model effectively quantifies the uncertainty of its predictions.

In contrast, the Dropout GN-ODE model consistently registers the highest RNLL across thresholds, pointing to a potentially poorer fit to the data and less effective uncertainty quantification. The severity of this gap is accentuated at the 0.25 variance threshold, where the RNLL of the Dropout GN-ODE model surpasses its counterparts by a substantial margin.

However, it's also essential to note the competitive performance of the Bayesian GCN. For most thresholds, BG Bayesian GCN maintains a performance closely trailing the GN-SDE model, reflecting its robust modeling and uncertainty estimation capabilities.

The Ensemble GCN, though showcasing some strength in lower variance thresholds, particularly in MAE, is inconsistent across metrics and thresholds. It often falls behind in the RNLL metric, which can be observed at the 100 and 3 variance thresholds.

In conclusion, while the GN-SDE model stands out for its robust performance across different variance thresholds, the Bayesian GCN model also showcases significant promise. Both Dropout GAT-GCN and Dropout GN-ODE face challenges in specific metrics, while the Ensemble GCN displays fluctuating performance.

3.3.3 Generating Graph SDE Data

In our study, we present a Generative Adversarial Network (GAN) model named the Graph Neural Stochastic Differential Equations GAN (Graph SDE-GAN). This model combines the generative abilities of GANs to produce graph-structured data with the dynamics of Stochastic Differential Equations (SDEs). This integration allows the Graph SDE-GAN to generate graph data with SDE-related features. As a result, the Graph SDE-GAN offers a mechanism for producing graph data with SDE characteristics, facilitating data analysis and modeling in various fields.

Additionally, the synthetic data created by the Graph SDE-GAN can be used to train a Latent Graph SDE model. In scenarios where the original dataset is inherently stochastic and noisy—attributes that align with a Graph SDE model—but is limited in quantity, the Graph SDE-GAN can supplement this dataset with synthetic data. This additional data can be utilized to train the Latent Graph SDE model, addressing challenges related to limited data availability.

In this graph regression challenge, we engage with a fully connected four-node graph. Every node carries a feature aligned to a temporal aspect 't'. The target is to deduce a regression outcome, illustrated by the functions $\cos(t)$ and $\sin(t)$ for the left and right images in Figure 3.7, respectively. The figures displays the synthetic data, shaped by the nuances of Stochastic Differential Equations (SDEs), generated by the Graph SDE-GAN. Remarkably,

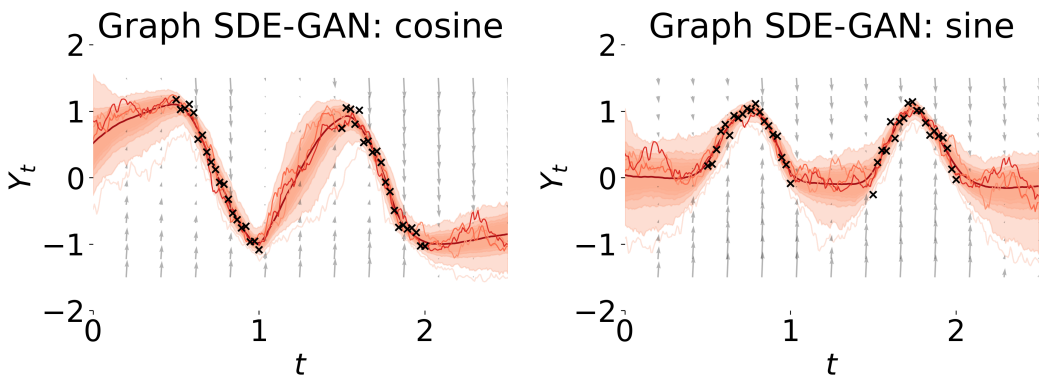


Fig. 3.7 Two visualizations of synthetic data from the Graph SDE-GAN, crafted for a four-node graph regression task. Here, the 'x' markers signify the original training data points.

the SDE-GAN exhibits proficiency in data modeling. Notably, as we move away from the training data's purview, the model's uncertainty escalates—aligning with our anticipations.

The Graph SDE-GAN, while promising, presents operational challenges, especially in the realm of hyperparameter tuning, a topic we delve into in the subsequent section. The potential and performance of the Graph SDE-GAN demand further examination, and a comparative study with other GNN generative models could yield valuable insights. However, we defer such in-depth investigations to future work.

3.4 Limitations

We have introduced the Graph Neural SDE and assessed its performance across a diverse set of datasets, covering both static and spatio-temporal data. In the majority of datasets tested, the Graph Neural SDE surpasses other methods, excelling in key metrics such as uncertainty quantification, out-of-distribution detection, robustness, and active learning.

Drawing upon the "No Free Lunch" theorem (Wolpert and Macready, 1997), we recognize that no single model is universally optimal across all scenarios. Consequently, our Graph Neural SDE does come with certain limitations. The primary drawback is its computational intensity. Compared to GNNs like GAT and GCN, and even Graph Neural ODEs, our model demands more computational resources. Its computational needs are on par with the Bayesian GCN, yet less than an ensemble of five GCNs. The root of this computational demand stems

from the inherent complexity of solving an SDE, which is intrinsically more involved than an ODE. Consequently, training times for the Graph Neural SDE tend to be longer.

From our empirical studies, we observed that Graph Neural SDE's training duration is approximately 1.5 times that of Graph Neural ODEs and about 3 times longer than basic GNNs. Nevertheless, this increased training time did not pose significant issues in our experiments.

When we delve deeper into the specific constructs of Graph Neural SDEs, distinct characteristics emerge between the Latent Graph SDE and the Graph SDE-GAN. While the Latent Graph SDE has consistently shown robust performance across tasks, the Graph SDE-GAN has thrown up certain implementation challenges. Specifically, the Graph SDE-GAN model demands meticulous tuning and optimization to deliver results effectively. Our testing has shown that this model showcases superior performance when gradient clipping is favored over the gradient penalty, and the Adadelta optimizer takes precedence over Adam.

These observations bring to the forefront the inherent trade-offs associated with Graph Neural SDEs. While they offer superior performance, robustness to noise, resistance to oversmoothing, continuous benefits for time-irregular datasets, a pronounced ability to quantify prediction uncertainty, and enhanced out-of-distribution detection, one must balance these advantages against the computational demands they impose and the intricate nuances of model optimization, especially for the Graph SDE-GAN.

3.5 Related Work

Uncertainty quantification in GNNs has been relatively less explored compared to traditional neural networks. Among the limited research in this domain, we have benchmarked our work against key contributions such as the Bayesian Graph Neural Network Lamb and Paige (2020) and robust ensemble methods (Lin et al., 2022). While there has been significant research into Gaussian Processes on graphs (Borovitskiy et al., 2021; Lawrence and Jordan, 2004; Pérez-Cruz et al., 2013), we did not include these in our evaluations.

Furthermore, the use of differential equations on graphs is a growing research area, with most work focused on Graph Neural ODEs (Poli et al., 2019) and Graph Control Differential Equations (Choi et al., 2022). Graph Neural ODEs have been applied to dynamic graph classification (Jin et al., 2022; Shi et al., 2023), traffic forecasting (Choi et al., 2022; Liu

et al., 2023), and protein interface prediction (Tan). Extensions include second-order and higher-order Graph ODEs (Luo et al., 2023; Zhang et al., 2022).

In the realm of synthesizing graph data using GNNs, a notable trajectory has been the integration with Generative Adversarial Networks (GANs). Works like GraphGAN (Wang et al., 2018) sought to enhance graph-based semi-supervised learning. Thereafter, GraphVAE (Simonovsky and Komodakis, 2018) applied Variational Autoencoders for graph generation. MolGAN (De Cao and Kipf, 2018), specifically targeted at molecular graphs, has displayed its niche utility in computational biology and chemistry. These endeavors signify the increasing importance of GNNs in synthetic data generation, akin to the advancements of GANs in image and sequence datasets.

While SDEs provide a promising avenue for better uncertainty quantification in differential equations, their application in graphs has been limited. They have primarily been employed in Graph Diffusion models for denoising purposes (Huang et al., 2022; Jo et al., 2022; Luo et al., 2022). In this context, the paper 'BroNet' (Bishnoi et al., 2023) is distinct for its claim to pioneer Graph Neural SDEs. However, its methodology deviates significantly from our approach as it merely employs a GNN to learn an SDE's scalar parameter, without truly modeling the graph as an SDE.

In summary, the integration of differential equations with stochastic dynamics in the context of graph neural networks is still nascent. Our work aims to be a trailblazer in this domain, setting the stage for future endeavors in Graph SDE-based techniques.

Summary

In this section, we introduced our innovative model, Graph Neural SDEs, detailing both its forms: the Latent Graph Neural SDE and the Graph SDE-GAN. We demonstrated that the (Latent) Graph Neural SDE outperforms in various metrics tested, including accuracy, robustness, uncertainty quantification, resistance to oversmoothing, and efficient data collection through active learning. We acknowledged the higher computational demands of our model compared to others and addressed the specific limitations associated with the Graph SDE-GAN, particularly concerning hyperparameter tuning. Additionally, we provided an overview of related work in the field. With these discussions, we conclude this chapter and transition to our final section to wrap up the dissertation.

Chapter 4

Conclusion

In this study, our primary contribution is the introduction of the **Latent Graph Stochastic Differential Equations (Latent Graph SDE)** model, tailored for advanced tasks such as node, graph, and link prediction. In addition to this primary model, we have also presented a novel approach for synthetic data generation — the **Graph Neural Stochastic Differential Equations Generative Adversarial Network (Graph SDE-GAN)**. By integrating the dynamics of Stochastic Differential Equations (SDEs) within a Generative Adversarial Network (GAN) framework, the Graph SDE-GAN emerges as a potent tool for producing graph-structured data with SDE characteristics. Conversely, the Latent Graph SDE model is tailored for tasks such as node, graph, and link prediction.

Our experimental results reveal that the (latent) Graph Neural SDEs consistently outperform contemporaneous models such as the Baysian GCN, Ensemble GCN, and Graph Neural ODEs and GCNs. The metrics where our model took the lead include uncertainty quantification, out-of-distribution detection, robustness, and active learning.

The unanticipated edge in accuracy displayed by our model compared to the Graph Neural ODE might be rooted in the inherent noise incorporated within the differential equations of our Graph Neural SDEs. This intrinsic noise, infused during each epoch or batch of training, can be analogized to the data augmentation techniques used in conventional machine learning. Consequently, the model regularly interacts with slightly altered data versions, which might bolster its robustness, leading to enhanced generalization during testing. This hypothesis warrants future exploration — perhaps by comparing a Graph Neural ODE trained on noise-augmented data and assessing the resulting accuracy differences.

While our findings underscore the prowess of Graph Neural SDEs compared to their counterparts like Graph Neural ODEs, this supremacy is not without its complications. The nature of SDEs is inherently intricate, necessitating more computational power. This complexity is largely due to the integration costs tied to SDEs, typically surpassing those of ODEs.

To conclude, our work with the Latent Graph SDE and the Graph SDE-GAN not only bridges a research gap but also paves the way for future endeavors in the integration of stochastic differential equations with graph neural networks, fostering advancements in both methodology and application domains. And provides a new Graph-based model which hable to sucessefuly qunatify the uncertianty of the predictions.

4.1 Future Work

The journey into Graph Neural SDEs has illuminated several promising avenues that beckon deeper exploration.

Comparison of Graph SDE-GAN with Contemporary Graph Generative Models: A direct comparison of our Graph SDE-GAN with state-of-the-art graph generative models would provide valuable insights into its standing within the synthetic data generation ecosystem. By pitting it against established models such as GraphVAE, GraphGAN, and MolGAN, we aim to identify its unique strengths, potential shortcomings, and areas for refinement.

Bayesian Neural Network SDE with Ornstein-Uhlenbeck Prior over the weights: A particularly promising direction is building upon the Bayesian Neural Network SDE utilizing an Ornstein-Uhlenbeck prior for its weights, as elucidated by Xu et al. (2022). There exists an enticing possibility to extend this paradigm to graph structures, creating Graph Bayesian Neural Network SDEs (Graph BNN-SDEs).

Adaptation of Advancements in PDEs: The recent innovations in Partial Differential Equations presented by Sun et al. (2020) offers an intriguing template. By harnessing a latent method akin to the one in our work, there lies the potential to craft Graph Stochastic Partial Differential Equations (Graph SPDEs) tailored for intricate graph contexts.

Integration of Higher-order SDEs: The fusion of higher-order stochastic differential equations with the Latent Graph Neural SDE we've pioneered could open new doors. Such integrations may serve as potent tools in the nuanced modeling of complex systems, capturing their dynamics with even greater fidelity.

Infusing Authentic Stochasticity in Deterministic Models: In our current exploration, we ventured into introducing stochasticity to deterministic models like the Graph Neural ODE and GNNs (e.g., GCN and GAT) through dropout at inference, inspired by Yarin Gal's methodology. While effective, this strategy might not capture the intricate randomness inherent in systems steered by Stochastic Differential Equations (SDEs). A compelling next step would be to integrate Gillespie's algorithm, famed for its stochastic simulations in biochemical realms, as a conduit to seed genuine stochastic dynamics into Neural ODEs. By emulating the organic noise observed in real-world scenarios, this fusion could usher in neural architectures that harmoniously blend the determinism's stability with the vibrancy of stochastic dynamics, refining their prowess in tasks where organic system noise takes center stage.

In wrapping up this dissertation and as we set our sights on future investigations, it's evident that the field is full of opportunities. We take pride in having contributed significantly to the realm of uncertainty-aware graph-based machine learning throughout this work.

References

K. Aggarwal, M. M. Mijwil, A.-H. Al-Mistarehi, S. Alomari, M. Gök, A. M. Z. Alaabdin, S. H. Abdulrhman, et al. Has the future started? the current growth of artificial intelligence, machine learning, and deep learning. *Iraqi Journal for Computer Science and Mathematics*, 3(1):115–123, 2022.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

R. Bergna. Graphneuralsde, 2023. URL https://github.com/Richard-Bergna/GraphNeuralSDE. GitHub repository.

S. Bishnoi, S. Ranu, N. Krishnan, et al. Graph neural stochastic differential equations for learning brownian dynamics. *arXiv preprint arXiv:2306.11435*, 2023.

V. Borovitskiy, I. Azangulov, A. Terenin, P. Mostowsky, M. Deisenroth, and N. Durrande. Matérn gaussian processes on graphs. In *International Conference on Artificial Intelligence and Statistics*, pages 2593–2601. PMLR, 2021.

G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668, 2022.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

R. Buckdahn, B. Djehiche, and J. Li. A general stochastic maximum principle for sdes of mean-field type. *Applied Mathematics & Optimization*, 64(2):197–216, 2011.

L. Cardelli. From processes to odes by chemistry. In *Fifth Ifip International Conference On Theoretical Computer Science–Tcs 2008*, pages 261–281. Springer, 2008.

M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020.

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

J. Choi, H. Choi, J. Hwang, and N. Park. Graph neural controlled differential equations for traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6367–6374, 2022.

M. Cvijovic, J. Almquist, J. Hagmar, S. Hohmann, H.-M. Kaltenbach, E. Klipp, M. Krantz, P. Mendes, S. Nelander, J. Nielsen, et al. Bridging the gaps in systems biology. *Molecular Genetics and Genomics*, 289:727–734, 2014.

N. De Cao and T. Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: An automatic citation indexing system. *Proceedings of the third ACM conference on Digital libraries*, pages 89–98, 1998.

V. Gontis and A. Kononovicius. Consentaneous agent-based and stochastic model of the financial markets. *PLoS One*, 9(7):e102201, 2014.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

A. Hasanzadeh, E. Hajiramezanali, S. Boluki, M. Zhou, N. Duffield, K. Narayanan, and X. Qian. Bayesian graph neural networks with adaptive connection sampling. In *International conference on machine learning*, pages 4094–4104. PMLR, 2020.

D. J. Higham. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM review*, 43(3):525–546, 2001.

S. Hoops, R. Hontecillas, V. Abedi, A. Leber, C. Philipson, A. Carbo, and J. Bassaganya-Riera. Ordinary differential equations (odes) based modeling. In *Computational Immunology*, pages 63–78. Elsevier, 2016.

W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

H. Huang, L. Sun, B. Du, Y. Fu, and W. Lv. Graphgdp: Generative diffusion processes for permutation invariant graph generation. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 201–210. IEEE, 2022.

K. Itô. 109. stochastic integral. *Proceedings of the Imperial Academy*, 20(8):519–524, 1944.

M. Jin, Y. Zheng, Y.-F. Li, S. Chen, B. Yang, and S. Pan. Multivariate time series forecasting with dynamic graph neural odes. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

J. Jo, S. Lee, and S. J. Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International Conference on Machine Learning*, pages 10362–10383. PMLR, 2022.

A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

W. O. Kermack and A. G. McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character*, 115(772):700–721, 1927.

R. Khasminskii. *Stochastic Stability of Differential Equations*. Springer, 2012.

P. Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

P. Kidger, J. Morrill, J. Foster, and T. Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

P. Kidger, J. Foster, X. C. Li, and T. Lyons. Efficient and accurate gradients for neural sdes. *Advances in Neural Information Processing Systems*, 34:18747–18761, 2021.

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

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

G. Lamb and B. Paige. Bayesian graph neural networks for molecular property prediction. *arXiv preprint arXiv:2012.02089*, 2020.

N. Lawrence and M. Jordan. Semi-supervised learning via gaussian processes. *Advances in neural information processing systems*, 17, 2004.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

C. Leibig, V. Allken, M. S. Ayhan, P. Berens, and S. Wahl. Leveraging uncertainty information from deep neural networks for disease detection. *Scientific reports*, 7(1):17816, 2017.

Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018a.

X. Li, T.-K. L. Wong, R. T. Chen, and D. Duvenaud. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pages 3870–3882. PMLR, 2020.

Y. Li and A. Gupta. Beyond grids: Learning graph representations for visual recognition. *Advances in neural information processing systems*, 31, 2018.

Y. Li, R. Yu, C. Shahabi, and Y. Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations (ICLR '18)*, 2018b.

Q. Lin, S. Yu, K. Sun, W. Zhao, O. Alfarraj, A. Tolba, and F. Xia. Robust graph neural networks via ensemble learning. *Mathematics*, 10(8):1300, 2022.

X. Liu, J. Ding, W. Jin, H. Xu, Y. Ma, Z. Liu, and J. Tang. Graph neural networks with adaptive residual. *Advances in Neural Information Processing Systems*, 34:9720–9733, 2021a.

Y. Liu, K. Zeng, H. Wang, X. Song, and B. Zhou. Content matters: A gnn-based model combined with text semantics for social network cascade prediction. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 728–740. Springer, 2021b.

Z. Liu, P. Shojaee, and C. K. Reddy. Graph-based multi-ode neural networks for spatio-temporal traffic forecasting. *arXiv preprint arXiv:2305.18687*, 2023.

T. Luo, Z. Mo, and S. J. Pan. Fast graph generative model via spectral diffusion. *arXiv preprint arXiv:2211.08892*, 2022.

X. Luo, J. Yuan, Z. Huang, H. Jiang, Y. Qin, W. Ju, M. Zhang, and Y. Sun. Hope: High-order graph ode for modeling interacting dynamics. 2023.

V. Mandelzweig and F. Tabakin. Quasilinearization approach to nonlinear problems in physics with application to nonlinear odes. *Computer Physics Communications*, 141(2): 268–281, 2001.

J. C. Maxwell. Li. on physical lines of force. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 21(141):338–348, 1861.

R. Merris. A survey of graph laplacians. *Linear and Multilinear Algebra*, 39(1-2):19–31, 1995.

S. Min, Z. Gao, J. Peng, L. Wang, K. Qin, and B. Fang. Stgsn—a spatial–temporal graph neural network framework for time-evolving social networks. *Knowledge-Based Systems*, 214:106746, 2021.

K. Oono and T. Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

F. Pérez-Cruz, S. Van Vaerenbergh, J. J. Murillo-Fuentes, M. Lázaro-Gredilla, and I. Santamaria. Gaussian processes for nonlinear signal processing: An overview of recent advances. *IEEE Signal Processing Magazine*, 30(4):40–50, 2013.

M. Poli, S. Massaroli, J. Park, A. Yamashita, H. Asama, and J. Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

M. Quach, N. Brunel, and F. d'Alché Buc. Estimating parameters and hidden variables in non-linear state-space models based on odes for biological networks inference. *Bioinformatics*, 23(23):3209–3216, 2007.

N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

D. Revuz and M. Yor. *Continuous martingales and Brownian motion*, volume 293. Springer Science & Business Media, 2013.

L. M. Ricciardi. *Biomathematics and related computational problems*. Springer Science & Business Media, 2012.

J. L. Russell. Kepler's laws of planetary motion: 1609–1666. *The British journal for the history of science*, 2(1):1–24, 1964.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

S. J. Schreiber, M. Benaïm, and K. A. Atchadé. Persistence in fluctuating environments. *Journal of Mathematical Biology*, 62:655–683, 2011.

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

G. Shi, D. Zhang, M. Jin, and S. Pan. Towards complex dynamic physics system simulation with graph neural odes. *arXiv preprint arXiv:2305.12334*, 2023.

M. Simonovsky and N. Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*, pages 412–422. Springer, 2018.

V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020.

R. Stratonovich. A new representation for stochastic integrals and equations. *SIAM Journal on Control*, 4(2):362–371, 1966.

Y. Sun, L. Zhang, and H. Schaeffer. Neupde: Neural network based ordinary and partial differential equations for modeling time-dependent data. In *Mathematical and Scientific Machine Learning*, pages 352–372. PMLR, 2020.

X. W. Tan. Fuzzy3dvecnet: Novel robust deep learning approach to protein-ligand interaction prediction.

G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

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

H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo. Graphgan: Graph representation learning with generative adversarial nets. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32 (1):4–24, 2020.

W. Xu, R. T. Chen, X. Li, and D. Duvenaud. Infinitely deep bayesian neural networks with stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pages 721–738. PMLR, 2022.

B. Yu, H. Yin, and Z. Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

Y. Zhang, S. Gao, J. Pei, and H. Huang. Improving social network embedding via new second-order continuous graph neural networks. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 2515–2523, 2022.

J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

M. Zitnik and J. Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.

Appendix A

Extended Backgorund

A.1 The Adjoint Method for Neural ODEs and Neural SDEs

A.1.1 Neural ODEs

Neural Ordinary Differential Equations (ODEs) provide a framework to represent continuous-time dynamics. An instance of a Neural ODE can be formulated as:

$$\frac{dz}{dt} = f(z(t), t, \theta) \tag{A.1}$$

In this formulation, $z(t)$ denotes the evolving state at time t, while θ encapsulates the parameters governing the function f.

Given an initial condition $z(t_0)$, the resultant state at t_1 is represented as $z(t_1)$. In the context of training Neural ODEs, a pertinent task involves the computation of the gradient:

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial z(t_1)} \frac{\partial z(t_1)}{\partial \theta} \tag{A.2}$$

Here, L stands for a designated loss function contingent upon $z(t_1)$.

To facilitate an efficient gradient calculation, we introduce an auxiliary concept termed

the adjoint state, represented as:

$$a(t) = \frac{\partial L}{\partial z(t)}$$

This adjoint state's temporal evolution is encapsulated by the relationship:

$$\frac{da(t)}{dt} = -a(t)\frac{\partial f}{\partial z} \tag{A.3}$$

Remarkably, this differential equation offers solutions through a backward time integration, spanning from t_1 to t_0.

The culmination of this methodology allows us to express the gradient concerning the parameters as:

$$\frac{dL}{d\theta} = \int_{t_0}^{t_1} a(t)\frac{\partial f}{\partial \theta}dt \tag{A.4}$$

Definition: The **Adjoint State** serves as an intermediary state, evolving inversely with time. It acts as the cornerstone for the efficacious computation of gradients within the realm of Neural ODEs.

Remark: The adjoint technique emerges as a memory-efficient strategy. It circumvents the need to persistently store intermediate activations, a prevailing bottleneck in deep learning architectures. Augmenting Neural ODEs with the adjoint methodology further bequeaths them with the prowess of adaptive depth, honing itself based on data intricacies.

A.1.2 Neural Stochastic Differential Equations

Embarking upon the domain of stochastic differential equations (as defined in Definition 2.1.3), we propose a neural stochastic differential equation (SDE) expressed as :

$$dZ_t = f(Z_t, t, \theta)\,dt + g(Z_t, t, \theta)\,dW_t \tag{A.5}$$

Within this portrayal, Z_t symbolizes the state at an arbitrary time t. Here, f operates as the drift function, g delineates the diffusion function, θ encompasses the model's parameters, and W_t denotes a canonical Wiener process.

The overarching objective revolves around deducing the gradient $\frac{dL}{d\theta}$ of a specified loss function L, relative to the parameters θ.

Derivation of the Adjoint Equations for Neural SDEs

A pivotal tool in our arsenal, Ito's lemma, facilitates discerning the dynamics of the gradient derivative $\frac{d}{dt}\left(\frac{dL}{dZ_t}\right)$ as:

$$\frac{d}{dt}\left(\frac{dL}{dZ_t}\right) = -\left(\frac{dL}{dZ_t}\right)\left(\frac{\partial f}{\partial Z_t}\right) - 0.5\left(\frac{dL}{dZ_t}\right)\left(\frac{\partial g}{\partial Z_t}\right)\left(\frac{\partial g}{\partial Z_t}\right)^T \tag{A.6}$$

The above dynamics, in tandem with the original forward SDE, embodies a symphony of coupled forward-backward SDEs.

Enlisting the aid of the chain rule, we extrapolate $\frac{dL}{d\theta}$ in terms of $\frac{dL}{dZ_t}$ and $\frac{dZ_t}{d\theta}$:

$$\frac{dL}{d\theta} = \left(\frac{dL}{dZ_t}\right)\left(\frac{dZ_t}{d\theta}\right) \tag{A.7}$$

Differentiating the rudimentary SDE concerning θ, we infer:

$$\frac{d}{dt}\left(\frac{dZ_t}{d\theta}\right) = \left(\frac{\partial f}{\partial Z_t}\right)^T \frac{dZ_t}{d\theta} + \frac{\partial f}{\partial \theta} + \left(\frac{\partial g}{\partial Z_t}\right)^T \frac{dZ_t}{d\theta}\frac{\partial g}{\partial \theta} \tag{A.8}$$

This expression epitomizes the adjoint SDE. Noteworthily, by integrating this system in a reverse temporal direction, one can compute the gradient $\frac{dL}{d\theta}$ with heightened efficiency.

> **Remark:** The adoption of the adjoint methodology for Neural SDEs is firmly anchored in its roots within stochastic calculus. This framework promises a methodologically robust and efficient means to extract gradients within neural stochastic differential systems. This is accomplished without incurring the computational overhead of exorbitant memory allocation.

A.2 Mathematics

A.2.1 Interpretations of Stochastic Integrals

When integrating stochastic processes, especially when dealing with Brownian motion, ambiguity arises concerning the interpretation of the integral due to the inherently non-smooth nature of such processes. Two of the most prominent interpretations in the literature are the Itō and Stratonovich interpretations.

A.2.2 Itō Interpretation

The Itō interpretation is arguably the more widespread and commonly used of the two, particularly in finance and related disciplines (Itô, 1944). It integrates with respect to the left-hand limit, inherently assuming a non-anticipative nature.

Definition: Given a stochastic process $X(t)$ defined by the SDE:

$$dX(t) = \alpha(t)dt + \beta(t)dW(t) \tag{A.9}$$

The Itō integral over the interval $[0,t]$ is defined as:

$$\int_0^t \beta(s)dW(s) \tag{A.10}$$

Where the integration essentially sums up all the tiny random increments of $W(s)$ weighted by $\beta(s)$ over the interval.

A.2.3 Stratonovich Interpretation

The Stratonovich interpretation finds its roots in physics and provides a more intuitive interpretation when considering physical systems (Stratonovich, 1966). Unlike Itō's interpretation, which is non-anticipative, the Stratonovich interpretation uses the midpoint for integration, inherently accounting for the anticipative nature of the system.

Definition: Given the same stochastic process $X(t)$ defined by the SDE:

$$dX(t) = \alpha(t)dt + \beta(t) \circ dW(t) \tag{A.11}$$

The Stratonovich integral over the interval $[0,t]$ is denoted by the "circle" notation ($\circ$) and defined similarly as:

$$\int_0^t \beta(s) \circ dW(s) \tag{A.12}$$

However, in this interpretation, the integrand $\beta(s)$ is evaluated at the midpoint of the interval.

Remark: The choice between Itō and Stratonovich integrals often depends on the problem at hand. While Itō's interpretation is non-anticipative and therefore often preferred in finance where anticipative strategies are inherently non-feasible, the Stratonovich interpretation, being more physically intuitive, is often chosen for problems in physics. Importantly, one can convert between the two forms using the so-called Stratonovich-Itō correction.

A.2.4 Ornstein–Uhlenbeck Process

The Ornstein–Uhlenbeck (OU) process stands as an integral pillar for comprehending specific types of Stochastic Differential Equations (SDEs) (Uhlenbeck and Ornstein, 1930). Frequently invoked to capture the dynamics of a Brownian particle under the sway of friction, the OU process functions as the continuous-time counterpart of the first-order autoregressive model, commonly denoted as AR(1).

Definition: The equation governing the Ornstein–Uhlenbeck process, $X(t)$, is represented by the following Stochastic Differential Equation (SDE):

$$dX(t) = \theta(\mu - X(t))dt + \sigma dW(t) \tag{A.13}$$

In this formulation, $\theta > 0$ denotes the rate at which the process reverts to its mean. The term μ signifies the long-term mean that the process gravitates towards. The parameter $\sigma > 0$ describes the volatility inherent in the process, and $W(t)$ is recognized as a Wiener process or Brownian motion.

Remark: Several crucial properties characterize the Ornstein–Uhlenbeck process. Foremost, as t advances towards infinity, the process inclines towards a mean μ, possessing a variance equal to $\frac{\sigma^2}{2\theta}$. Additionally, when t reaches infinity, the stationary distribution of the process assumes a Gaussian distribution. This distribution is distinguished by a mean μ and variance $\frac{\sigma^2}{2\theta}$. It's also worth noting that the Ornstein–Uhlenbeck process enjoys applications within the financial sector, particularly in rendering models for interest rates and exchange rate behaviors.

A.2.5 Lipschitz Continuity in Neural Networks and SDEs

The Lipschitz continuity plays an indispensable role in the study of Stochastic Differential Equations (SDEs) and neural networks (Higham, 2001; Rahaman et al., 2019). This continuity condition ensures the boundedness of the rate of change of functions, which, in the realm of SDEs, becomes pivotal for ascertaining the existence and uniqueness of solutions.

Definition: Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, it adheres to the Lipschitz condition within a domain D if there's a positive real constant L, termed the Lipschitz constant. For any pair of points x, y residing in D, the following relationship must be maintained:

$$\|f(x) - f(y)\| \leq L\|x - y\|$$

The implications of this condition are twofold. Firstly, it prevents the function f from exhibiting abrupt changes or exceedingly steep slopes between any two points. Secondly, while it embodies a stronger continuity requirement, the Lipschitz condition offers a systematic mechanism to control the variations a function may experience.

Remark: Consider a typical SDE expressed as:

$$dz(t) = f(t, z(t))dt + g(t, z(t))dW(t),$$

where f denotes the drift coefficient and g the diffusion coefficient. By ensuring Lipschitz conditions for both f and g with respect to z, we ensure that for a constant t, any alterations in z will not lead to rapid changes in the functions f and g. Such a guarantee is crucial as it asserts the existence of a unique strong solution for the SDE, implying that the solution

$z(t)$ correlates to the underlying Wiener process $W(t)$ in a manner that's deterministically unique.

However, it's important to highlight that while Lipschitz conditions facilitate an easy pathway to deduce the existence and uniqueness of solutions, they are not always imperative. There exist SDEs with coefficients that do not adhere to Lipschitz conditions, yet still possess unique solutions. Nevertheless, in a plethora of practical scenarios, the coefficients in focus usually respect the Lipschitz constraints.

A.3 Evaluation Metrics

In predictive modeling, it is essential to have reliable metrics to evaluate the performance of the models. This section provides a brief overview of three commonly used evaluation metrics: Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE).

A.3.1 Root Mean Square Error (RMSE)

The Root Mean Square Error (RMSE) is a frequently used measure of the differences between values predicted by a model and the values observed. It is especially useful when large errors are particularly undesirable. The RMSE of a model prediction with respect to the estimated variable X is defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X}_i)^2} \tag{A.14}$$

where X_i denotes the observed values, $\hat{X}_i$ denotes the predicted values, and n is the total number of observations.

A.3.2 Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) is a measure of errors between paired observations expressing the same phenomenon. It is less sensitive to outliers compared to RMSE. The MAE is given by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|X_i - \hat{X}_i| \tag{A.15}$$

where X_i denotes the observed values, $\hat{X}_i$ denotes the predicted values, and n is the total number of observations.

A.3.3 Mean Absolute Percentage Error (MAPE)

The Mean Absolute Percentage Error (MAPE) is a measure of prediction accuracy of a forecasting method in statistics. It usually expresses the accuracy as a percentage, and is defined by the formula:

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{X_i - \hat{X}_i}{X_i}\right| \tag{A.16}$$

where X_i denotes the observed values, $\hat{X}_i$ denotes the predicted values, and n is the total number of observations. Note that MAPE is undefined for $X_i = 0$ and can take on large values if some X_i are close to zero.

Appendix B

Extended Results

B.1 Static Toy Data set

Number of Nodes	Graph Neural ODE	GNN	Graph Neural SDE
100	Train: 0.7755 $\pm$ 0.138 Test: 0.532 $\pm$ 0.116	Train: 0.7865 $\pm$ 0.105 Test: 0.446 $\pm$ 0.103	Train: 0.8135 $\pm$ 0.077 Test: 0.662 $\pm$ 0.152
200	Train: 0.7525 $\pm$ 0.144 Test: 0.786 $\pm$ 0.161	Train: 0.8298 $\pm$ 0.110 Test: 0.871 $\pm$ 0.140	Train: 0.8157 $\pm$ 0.081 Test: 0.912 $\pm$ 0.116
500	Train: 0.7088 $\pm$ 0.140 Test: 0.7516 $\pm$ 0.154	Train: 0.8012 $\pm$ 0.114 Test: 0.8576 $\pm$ 0.138	Train: 0.8187 $\pm$ 0.097 Test: 0.8836 $\pm$ 0.098
1000	Train: 0.7417 $\pm$ 0.158 Test: 0.777 $\pm$ 0.178	Train: 0.7893 $\pm$ 0.144 Test: 0.8178 $\pm$ 0.162	Train: 0.8931 $\pm$ 0.096 Test: 0.9478 $\pm$ 0.107
2500	Train: 0.7629 $\pm$ 0.169 Test: 0.7822 $\pm$ 0.181	Train: 0.81 $\pm$ 0.152 Test: 0.8174 $\pm$ 0.172	Train: 0.9187 $\pm$ 0.113 Test: 0.9394 $\pm$ 0.126
5000	Train: 0.7729 $\pm$ 0.175 Test: 0.7843 $\pm$ 0.183	Train: 0.813 $\pm$ 0.158 Test: 0.8255 $\pm$ 0.164	Train: 0.9303 $\pm$ 0.121 Test: 0.9434 $\pm$ 0.122
10000	Train: 0.7534 $\pm$ 0.192 Test: 0.7587 $\pm$ 0.197	Train: 0.8219 $\pm$ 0.157 Test: 0.8314 $\pm$ 0.160	Train: 0.9346 $\pm$ 0.122 Test: 0.9446 $\pm$ 0.122

Table B.1 Training and Test Accuracy for Different Models and Number of Nodes, 80 percent traning

The table in B.1 provides a comparative analysis of the performance of three distinct models: GN-ODE, GNN , and GN_SDE, across various graph sizes determined by the number of nodes. The performance metrics utilized for the comparison are training and test accuracies, with results represented as mean values of 25 runs accompanied by standard deviations.

From the results, it becomes evident that the GN_SDE model consistently exhibits superior performance in test accuracy across all graph sizes. This superior performance becomes even more pronounced as the number of nodes in the graph increases. In the realm of training accuracies, while GN_SDE and GNN show comparable results for smaller graph sizes, GN_SDE manifests a discernible lead for larger graphs, specifically those with node counts exceeding 1000.

Furthermore, the GN_SDE model generally showcases smaller standard deviations, particularly in test accuracies, hinting at more consistent and reliable results. On the contrary, GN-ODE, although competitive in some scenarios, does not scale as effectively as its counterparts, especially for larger graphs.

In summation, the GN_SDE model emerges as the most robust and effective model, especially in test scenarios and larger graphs, showcasing both enhanced performance and consistency.

The table presented in B.2 showcases a systematic evaluation of three distinctive models, namely the Graph Neural ODE (GN-ODE), Graph Neural Network (GNN), and Graph Neural SDE (GN-SDE). The comparison is pivoted around performance metrics derived from varying training percentages from a dataset that comprises 200 training samples.

A close inspection of the results reveals a consistent trend: the GN-SDE model generally surpasses its counterparts in both training and test accuracies, especially at higher training percentages. While the GN-SDE and GNN exhibit closely aligned training accuracies, the GN-SDE demonstrates a clear advantage in test scenarios, indicating its better generalization capabilities. This advantage becomes progressively prominent as the training percentage escalates.

Moreover, the standard deviation values embedded in the results highlight the consistency of each model. GN-SDE often exhibits the smallest standard deviations in test accuracies, suggesting that its performance is not just superior but also more consistent across different data splits.

Training Percentage	Graph Neural ODE	GNN	Graph Neural SDE
10%	Train: 0.85 ± 0.091 Test: 0.555 ± 0.091	Train: 0.85 ± 0.009 Test: 0.590 ± 0.009	Train: 0.884 ± 0.087 Test: 0.662 ± 0.087
20%	Train: 0.752 ± 0.150 Test: 0.634 ± 0.150	Train: 0.743 ± 0.083 Test: 0.652 ± 0.083	Train: 0.77 ± 0.112 Test: 0.778 ± 0.112
30%	Train: 0.73 ± 0.123 Test: 0.708 ± 0.123	Train: 0.77 ± 0.109 Test: 0.705 ± 0.109	Train: 0.763 ± 0.114 Test: 0.801 ± 0.114
40%	Train: 0.73 ± 0.134 Test: 0.739 ± 0.134	Train: 0.799 ± 0.119 Test: 0.783 ± 0.119	Train: 0.765 ± 0.108 Test: 0.826 ± 0.108
50%	Train: 0.738 ± 0.137 Test: 0.732 ± 0.137	Train: 0.807 ± 0.115 Test: 0.773 ± 0.120	Train: 0.79 ± 0.086 Test: 0.828 ± 0.098
60%	Train: 0.735 ± 0.134 Test: 0.755 ± 0.137	Train: 0.769 ± 0.116 Test: 0.768 ± 0.144	Train: 0.791 ± 0.090 Test: 0.833 ± 0.110
70%	Train: 0.739 ± 0.137 Test: 0.768 ± 0.145	Train: 0.792 ± 0.115 Test: 0.799 ± 0.119	Train: 0.795 ± 0.089 Test: 0.838 ± 0.101
80%	Train: 0.741 ± 0.136 Test: 0.762 ± 0.146	Train: 0.79375 ± 0.112 Test: 0.821 ± 0.137	Train: 0.80375 ± 0.083 Test: 0.855 ± 0.108
90%	Train: 0.734 ± 0.136 Test: 0.768 ± 0.147	Train: 0.771 ± 0.125 Test: 0.792 ± 0.126	Train: 0.809 ± 0.088 Test: 0.856 ± 0.102

Table B.2 Training and Test Accuracy for Different Models and Training Percentages 200 traning dataset.

In conclusion, among the three models scrutinized, the GN-SDE consistently demonstrates a robust performance, with a particular inclination towards delivering higher test accuracies. This performance metric emphasizes its potential as the preferred model for tasks where training data can vary in size.

The table above presents the performance of three distinct models: GN-ODE, GNN, and GN-SDE, trained on varying percentages of data. The test accuracies are assessed based on different entropy thresholds, which serve as a confidence measure. Specifically, a smaller entropy value suggests a higher confidence in predictions, and only predictions with an entropy smaller than a given threshold are considered.

From the results, several insights can be extracted:

1. Across all training percentages, the *GN-SDE* consistently outperforms the *GN-ODE* and *GNN* models at nearly all entropy thresholds. This suggests that the *GN-SDE* model is more robust and confident in its predictions, making it preferable for tasks that require out-of-distribution detection.

Trained %	Models	Entropy Threshold										
		1.58	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
10%	GN-ODE	0.5549	0.5569	0.5585	0.5597	0.5627	0.5744	0.5811	0.5859	0.5937	0.6050	0.6242
	GNN	0.5898	0.5898	0.5898	0.5898	0.5898	0.5925	0.5928	0.5944	0.5925	0.5942	0.5966
	GN-SDE	0.6616	0.6672	0.6747	0.6880	0.7139	0.7419	0.7716	0.8028	0.8302	0.8474	0.8604
20%	GN-ODE	0.6335	0.6342	0.6344	0.6366	0.6429	0.6592	0.6696	0.6794	0.6926	0.7050	0.7277
	GNN	0.6517	0.6517	0.6517	0.6517	0.6517	0.6568	0.6608	0.6634	0.6664	0.6683	0.6726
	GN-SDE	0.7778	0.7865	0.7990	0.8598	0.8844	0.9054	0.9196	0.9326	0.9451	0.9519	0.9696
30%	GN-ODE	0.7083	0.7115	0.7147	0.7187	0.7287	0.7505	0.7628	0.7773	0.7894	0.8049	0.8208
	GNN	0.7054	0.7057	0.7056	0.7056	0.7058	0.7123	0.7180	0.7202	0.7253	0.7306	0.7377
	GN-SDE	0.8009	0.8131	0.8311	0.8598	0.8844	0.9054	0.9196	0.9326	0.9451	0.9519	0.9696
40%	GN-ODE	0.7387	0.7386	0.7400	0.7460	0.7542	0.7860	0.8051	0.8158	0.8230	0.8348	0.8534
	GNN	0.7833	0.7838	0.7849	0.7850	0.7856	0.7939	0.7983	0.8024	0.8078	0.8163	0.8279
	GN-SDE	0.8260	0.8412	0.8608	0.8837	0.9094	0.9297	0.9475	0.9623	0.9681	0.9727	0.9817
50%	GN-ODE	0.7324	0.7330	0.7368	0.7420	0.7514	0.7780	0.7938	0.8082	0.8291	0.8410	0.8567
	GNN	0.7728	0.7730	0.7733	0.7736	0.7736	0.7856	0.7900	0.7958	0.7996	0.8066	0.8186
	GN-SDE	0.8280	0.8337	0.8494	0.8753	0.9027	0.9235	0.9426	0.9551	0.9664	0.9742	0.9816
60%	GN-ODE	0.7550	0.7553	0.7576	0.7635	0.7702	0.7984	0.8150	0.8310	0.8423	0.8640	0.8897
	GNN	0.7680	0.7680	0.7678	0.7692	0.7693	0.7855	0.7908	0.7964	0.8031	0.8104	0.8148
	GN-SDE	0.8325	0.8438	0.8667	0.8911	0.9223	0.9422	0.9565	0.9682	0.9771	0.9753	0.9827
70%	GN-ODE	0.7680	0.7679	0.7679	0.7680	0.7752	0.8060	0.8237	0.8352	0.8514	0.8782	0.8812
	GNN	0.7993	0.7993	0.8011	0.8016	0.8020	0.8184	0.8223	0.8282	0.8330	0.8384	0.8457
	GN-SDE	0.8380	0.8527	0.8662	0.8934	0.9165	0.9349	0.9505	0.9701	0.9726	0.9791	0.9747
80%	GN-ODE	0.762	0.763	0.764	0.765	0.774	0.800	0.825	0.846	0.862	0.879	0.883
	GNN	0.821	0.821	0.821	0.821	0.821	0.834	0.843	0.851	0.855	0.859	0.866
	GN-SDE	0.855	0.856	0.861	0.877	0.892	0.914	0.930	0.939	0.952	0.971	0.974
90%	GN-ODE	0.768	0.770	0.773	0.771	0.782	0.804	0.818	0.828	0.833	0.850	0.865
	GNN	0.792	0.792	0.792	0.792	0.792	0.806	0.807	0.813	0.818	0.820	0.828
	GN-SDE	0.856	0.858	0.862	0.883	0.906	0.927	0.939	0.955	0.978	0.985	1.0

Table B.3 Test accuracies of GN-ODE, GNN, and GN-SDE models across varying training percentages and entropy thresholds. Lower entropy thresholds represent higher confidence levels in model predictions.

2. As the training percentage increases, there is a general trend of improvement in accuracy across all models and most entropy thresholds. This implies that with more training data, the models become more confident and accurate in their predictions.

3. When considering lower entropy thresholds (indicating higher confidence levels), the gap in performance between the models becomes more pronounced. Particularly, the *GN-SDE* model's superiority is more evident, suggesting that it is significantly more reliable when making highly confident predictions.

4. For certain models, especially the *GNN*, the accuracy tends to remain consistent across a range of entropy thresholds. This indicates that the model's predictions do not significantly change in confidence across these thresholds, potentially highlighting limitations in its ability to gauge its own uncertainty.

In conclusion, when evaluating models for out-of-distribution detection using entropy as a measure of confidence, the *GN-SDE* emerges as the most promising choice. Moreover,

increasing the training data percentage generally leads to improved performance, emphasizing the importance of training on extensive datasets for enhanced model confidence and accuracy.

B.1.1 Meter-LA Experiment Set Up

In our experiments, we utilized Graph Attention Networks (GAT) to embed the input data, which consisted of the past six recordings. These inputs were embedded into 64-dimensional tensors. Subsequently, these embeddings were passed to our Latent Graph Stochastic Differential Equation (SDE) model. The Graph Latent SDE model employs a GCN for the drift function, while the diffusion function is held constant at a value of 1. The hidden state of the model is of size 64, which is then passed to a GCN projection layer for prediction.

The Graph Neural ODE model follows a similar structure but replaces the SDE with an ODE and omits the noise component. The GCN model also utilizes the same embedding and projection layers but bypasses the differential equations entirely.

This setup allows for a fair comparison between the models, as they share the same embedding and projection layers, differing only in the differential equation component.

B.1.2 Spatial-Temporal Images

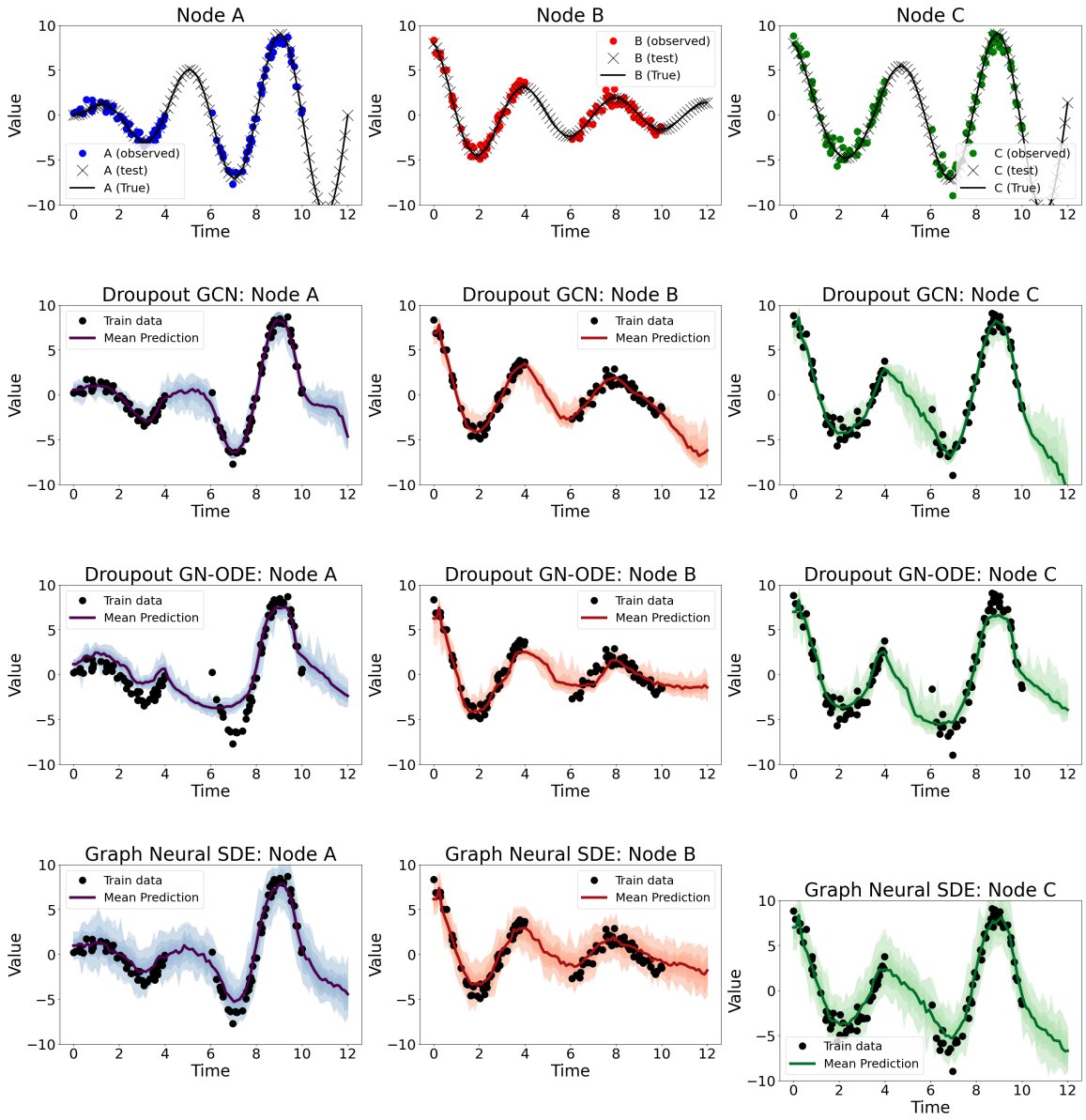


Fig. B.1 The figure displays 12 images organized. Each column corresponds to one of the 3 nodes, while each row represents a different model or dataset. The top row illustrates the training and testing datasets for each node, in the context of node regression. The aim is to predict the regression value for each node. The second row presents results from the GCN, the third row showcases those from the Graph Neural ODE, and the bottom row depicts our model, the Graph Neural SDE.