Connections between Graph Neural Networks and ODEs

Kaiwen Fu
MCAM

March 8, 2024

Abstract

Oartial Differential Equations (ODEs) play a fundamental role in describing physical phenomena across various scientific domains, from fluid dynamics to population growth. Meanwhile, Graph Neural Networks (GNNs) have emerged as powerful tools for learning and analyzing complex relationships within graph-structured data. This project explores the synergy between ODEs and GNNs, proposing a novel framework that seamlessly integrates these two domains for enhanced scientific computing.

1 Introduction

In our real world, nodes are a fundamental concept widely applicable across various fields, such as computer science, network analysis, biology, and social sciences, among others. They essentially represent points or positions within a network or system, which can be physical or abstract in nature. Nodes are interconnected by lines or links known as edges, forming structures known as graphs. These graphs can model relationships, paths, and networks in numerous contexts, providing a versatile tool for solving complex problems in the real world.

Graphs, consisting of nodes and edges, can be directed or undirected, implying the nature of the relationships they represent. In directed graphs, the connections have a direction, similar to one-way streets, indicating a one-sided relationship where you can move from one node to another but not necessarily back the same way. This can be used to model systems like web page links, where one page links to another without a reciprocal link, or traffic systems with one-way streets.

Undirected graphs, on the other hand, represent bidirectional relationships, similar to two-way streets, where there is no inherent directionality to the connections between nodes. This type can model undirected networks like social networks, where friendships are mutual, or physical networks like road maps where travel is possible in both directions between locations.

ODEs describe how the state of a system evolves over time. This characteristic is particularly useful in understanding the dynamics of complex systems, where the state at any time depends on its previous states. When applied to graphs, the propagation properties of ODEs can model how information (e.g., features of nodes or edges) diffuses across the graph. This is analogous to physical phenomena like heat diffusion or the spread of diseases in networks, where the change in state at any point (or node) is a function of its neighbors' states. Incorporating Ordinary Differential Equations (ODEs) with Graph Neural Networks (GNNs) for classification tasks leverages the propagation properties inherent in ODEs, providing a novel approach to machine learning models that handle graph-structured data.

The rest of the paper is organized as follows. We are going to introduce the idea and some background in Section 2, and some recent works in 3.Section 4 describes the Models we are going to use. The data will be presented in Section 5. The experiment will be implemented in Section 6. The results are reported in Section 7. A discussion concludes in Section 8.

2 Preliminary

2.1 Ordinary Differential Equations

Ordinary Differential Equations (ODEs) are commonly used to model the evolution of dynamic systems over time.

Now, We can simply consider the first order ODE:

$$\Big\{ u'(t) + au(t) = 0, t \in \mathbf{R}^+$$

Using Forward Difference method, we can get

$$\frac{U^{n+1} - U_n}{h} + aU^n = 0 \implies U^{n+1} = (1 - ah)U^n = (1 - ah)^n U^0$$

This means that the $n+1$ step could be represented by an operator $E = (1-ah)$ times the current step and $(1 - ah)^n$ times the initial condition. Each propagation is closely related to the previous situation. Connecting this feature with the Graph Neural Networks, which will be introduced in the next subsection, we can proposed a question. Can we apply ODEs into the Graph Neural Networks? The answer is YES! Due to the interconnected nature of nodes within a graph, an ideal ordinary differential equation (ODE) should consider the graph's structure and enable the propagation of information among various nodes. In each layer, the representation of each node is updated based on messages received from neighboring nodes.

In this project, we are only consider the ODE with the form

$$\frac{\partial u}{\partial t} = f(u(t), t)$$

Also, we can simply get the relation between two different stage with the first order ODE

$$u_{n+1} = u_n + f_n(u_n)$$

where t is a time variable which is continuous and from 0 to $+\infty$. Figure 1 shows the mind map of the continuous GNNs by Zhuang et al. [2020].

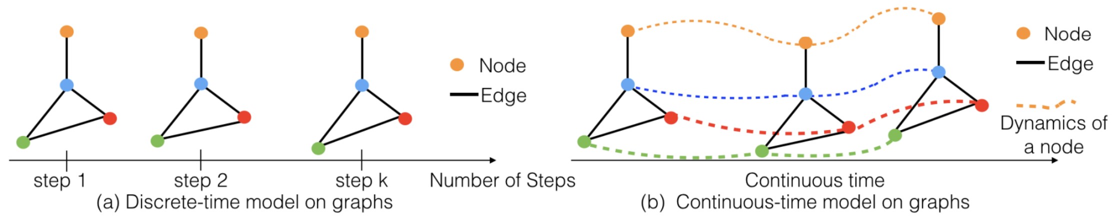

Figure 1: Idea of Continuous GNNs

2.2 Graph Neural Networks

From the class notes, a graph $\mathcal{G}$ is defined by a countable set V called the vertex set, an another set $E \subset V \times V$ called its edge set. we call G is undirected if

$$(v_1, v_2) \in E \iff (v_2, v_1) \in E$$

Graph neural networks (GNNs) are a class of neural networks designed to perform machine learning tasks on data represented as graphs. These models effectively process and interpret the complex structures and relationships inherent in graph data, enabling tasks such as node classification, link prediction, and graph classification. GNNs leverage the connectivity and features of nodes and edges within graphs to learn representations that capture both local and global structural information, offering a powerful approach to analyze data in domains where relationships are key.

The majority of Graph Neural Networks (GNNs)([Kipf and Welling, 2016]) operationalize their learning processes by extracting and incorporating structural information from graphs. This is accomplished by applying learned finite polynomial filters, denoted as g, to the eigenvalue Λ of the graph Laplacian (L). The graph Laplacian is a critical matrix in graph theory and spectral graph analysis, representing the structure of the graph. It is often expressed in its eigen-decomposition form as

$$L = P\Lambda P^{-1}$$

where P is the is the matrix of eigenvectors of L, and Λ is the diagonal matrix of eigenvalues.

The growing focus on graph-based machine learning reflects graphs' significant capability to encapsulate complex systems within various fields, such as social and natural sciences, including social networks, physical and biological systems, knowledge graphs, and beyond [Zhou et al., 2020]. This interest stems from graphs' ability to effectively represent intricate relationships and entities, demonstrating their potential in extracting deep insights across diverse research domains.

3 Recent Works

Poli et al. [2019] introduces Graph Ordinary Differential Equations (GODE), extending the idea of continuous-depth models from neural ordinary differential equations (NODE) to graph data. It proposes GODE to model the derivative of hidden node states with a graph neural network (GNN), solving the output states as an ODE solution. Two efficient training methods for GODE are presented: indirect back-propagation with the adjoint method and direct back-propagation through the ODE solver, with direct backprop showing superior performance. Additionally, a family of bijective blocks is introduced, enabling memory-efficient training. GODE's adaptability to various GNN structures and its performance improvement in semi-supervised node classification and graph classification tasks are validated.

Continuous Graph Neural Networks (CGNN) [Xhonneux et al., 2020] offer a novel and effective approach to learning graph representations by introducing continuous dynamics to the process. This advancement addresses common challenges in GNNs, such as over-smoothing and the difficulty in capturing long-range dependencies. Our project focuses on this method.

4 Model

In this section, we are going to introduce the model proposed in Continuous Graph Neural Networks (CGNN) by Xhonneux et al. [2020].

4.1 Continuous GNN without weights

Let $H_i \in \mathbf{R}^{|V|\times d}$ for node representations at different stage, and $E = H(0)$ where $H(0)$ is the initial value. Then, we can simply write the step-wise propagation equation:

$$H_{n+1} = AH_n + H_0$$

meaning that each node at stage $n + 1$ learns the node information from its neighbours through AH_n and remembers its original node features. And the explicit formula could be derive as:

$$H_n = (\sum_{i=0}^{\infty} A^i)H_0 = (A - I)^{-1}(A^{n+1} - I)H_0$$

and now, replacing all the step n with time $t \in \mathbf{R}^+$ so that we can get the continuous propagation. Specifically,

$$\frac{\partial H(t)}{\partial t} = \ln AH(t) + E$$

where $\ln A \approx (I - A)$ using the first order Taylor expansion

$$\frac{\partial H(t)}{\partial t} = (A - I)H(t) + E \implies H(t) = (I - A)^{-1}E$$

$$H(t) \approx (\sum_{i=0}^{\infty} A^i)E$$

4.2 Continuous GNNs with weights

Since ODE probably will not capture the correct dynamics of the graph, we can add a weight matrix $\in \mathbf{R}^{d\times d}$so that make the equation more powerful and be able to capture the data during the propagation:

$$H_{n+1} = AH_nW + H_0$$

Then, we can get that

$$\frac{\partial H(t)}{\partial t} = (A - I)H(t) + H(t)(W - I) + E$$

then, since A and W are symmetric, we can use eigenvalue decomposition, such that $A = P\Lambda P^{-1}$ and $W = Q\Phi Q^{-1}$, and denote $A - I$ and $W - I$ with eigenvalue decomposition $P\Lambda' P^{-1}$ and $Q\Phi Q^{-1}$ respectively, then we can derive the $H(t)$ as the form

$$H(t) = e^{(A-I)t}Ee^{(W-I)}t + PF(t)Q^{-1}$$

where

$$F_{ij}(t) = \frac{\tilde{E}}{\Lambda'_{ii} + \Phi'_{jj}} e^{t(\Lambda'_{ii}+\Phi'_{jj})} - \frac{\tilde{E}}{\Lambda'_{ii} + \Phi'_{jj}}$$

where $\tilde{E} = P^{-1}EQ$ When, $W = I$, then the equation will become the Continous GNNs without weights.

5 Data Description

The Cora dataset is a widely used benchmark in the field of graph machine learning [McCallum et al., 2000], particularly for tasks involving graph neural networks (GNNs). It is comprised of scientific publications classified into one of seven classes. These classes represent different areas of computer science, making the dataset suitable for evaluating the performance of models on node classification tasks. The key features of the Cora dataset include:

- Nodes: Each node in the graph represents a scientific publication. The dataset typically includes thousands of nodes, with each node being associated with a feature vector.
- Edges: The edges between nodes represent citation links, meaning there is an edge from publication A to publication B if publication A cites publication B. This citation information constructs the graph structure and allows the exploration of the relationships between different publications.
- Feature: The feature vector for each publication is derived from the word occurrences in the document, often represented as a binary or count-based vector indicating the presence or absence of certain words from a predefined dictionary.
- Classes: Each publication in the dataset is labeled with one of seven classes, which correspond to different areas of computer science research. The goal in many GNN tasks using the Cora dataset is to predict the class label of the nodes (publications) based on the structure of the graph and the features of the nodes.

Datasets	Number of Nodes	Number of Edges	Number of Features	Number of Classes
Cora	2,708	5,429	1,433	7

6 Experiment

We used the code provided by Xhonneux et al. [2020], which is the main theme of this project. For the experiment with Continuous Graph Neural Networks (CGNN) utilizing the specified number of epochs (50, 100, 200, 400) and a hidden dimension size of 16, we are going to investigate the effect of training duration on model performance, particularly focusing on test accuracy. Here is the structure to conducting the experiment:

- Model Initialization: Start by initializing the CGNN model with 16 hidden dimensions. This involves setting up the network architecture to ensure the correct propagation of continuous-time dynamics through the hidden layers. And We choose the hyper-parameter proposed by Bouthillier et al. [2022] using ORION with 50 entries.
- Training Regime: For each epoch configuration (50, 100, 200, 400), train the CGNN model separately. Monitor the loss on the training set and the accuracy on the validation set after each epoch.
- Evaluation: After training for the predefined number of epochs, evaluate the model on the test set to obtain the test accuracy and loss. It's crucial to maintain the same evaluation criteria for each training regime to ensure comparability.
- Comparison and Analysis: Compare the test accuracy and loss across different epoch settings. Analyze the results to determine the optimal number of epochs for the given model and dataset. This will give insights into how the model's ability to generalize improves or deteriorates over time with additional training.
- Overfitting Observation: Pay special attention to the possibility of overfitting, which may occur in longer training durations (such as 400 epochs). Overfitting can be identified if the validation accuracy starts to decrease while the training accuracy continues to improve.

- Results Interpretation: Discuss the findings and interpret the relationship between the number of epochs and test accuracy. It is expected that initially, with an increase in epochs, the model's performance will improve, but beyond a certain point, additional training may not yield better results or could even harm the model's ability to generalize.

7 Result

In this section, we are going to discuss the results generated by the classification process, and get the result about the test accuracy and loss.

7.1 Test Accuracy

After conducting trials with various epoch counts, we have obtained a range of accuracies, which are presented in the table below

Number of Epochs	Accuracy (%)
CGNN-50	82.9
WCGNN-50	82.8
CGNN-100	83.4
WCGNN-100	82.5
CGNN-200	83.4
WCGNN-200	83.0
CGNN-400	82.5
WCGNN-400	82.1

The data indicates that the accuracy hovers around 83%, suggesting that the models are consistently performing well across different epoch lengths. The fact that the accuracy does not deteriorate, even at 400 epochs, implies that overfitting is not occurring, as an overfit model would typically show a decrease in test accuracy due to memorizing the training data rather than learning generalizable patterns. This stability in accuracy, regardless of a high number of epochs, is indicative of robust models that generalize well to unseen data.

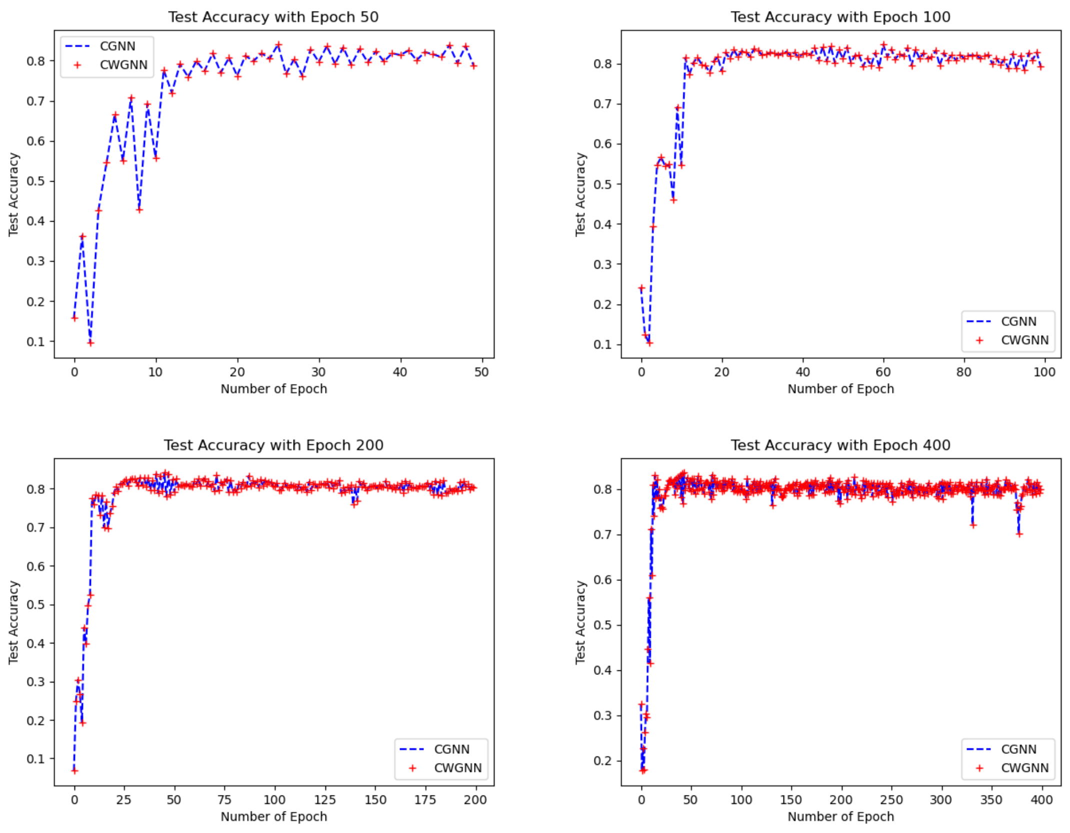

Figure 2: Test Accuracy with Different Epoches

The test accuracy from Figure 2 for both models stabilizes after a certain number of epochs, hovering around the 0.8 mark. Early epochs show significant improvements, which plateau relatively quickly. This rapid early improvement followed by stability suggests that the models can capture the patterns in the data efficiently and do not require extensive training to reach their peak performance. The lack of decline in test accuracy over many epochs suggests that both models are not overfitting significantly, even with extended training.

The comparison between CGNN and WCGNN does not show substantial differences in test accuracy across the number of epochs, indicating that both types of models have similar learning dynamics for the dataset in question.

To conclude, based on the data, increasing the number of epochs from 50 to 400 does not result in a significant increase in test accuracy for either model. This suggests that an optimal number of epochs for training these models, on this particular dataset, are not overfitting significantly.

7.2 Loss

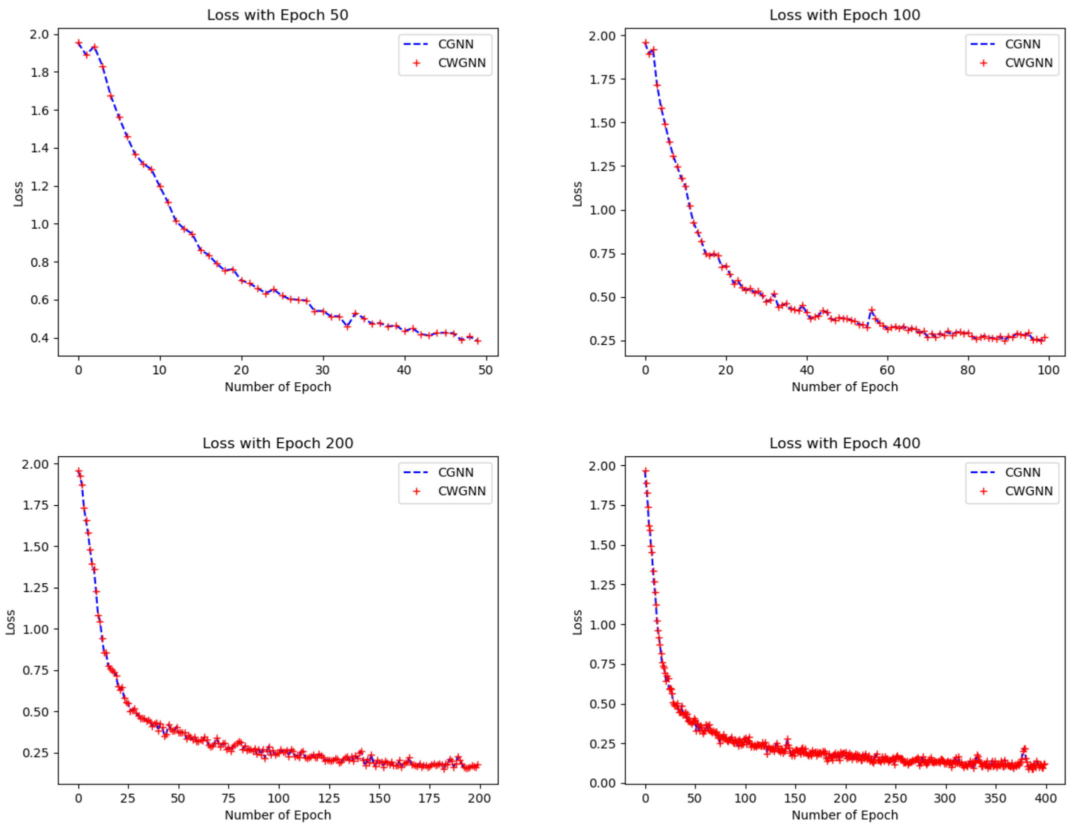

Figure 3: Loss with Different Epoches

In Figure 3, the loss metrics for both CGNN and CWGNN are presented over varying numbers of epochs (100, 200, and 400). Across all plots, both models exhibit a rapid decline in loss during the initial epochs, which suggests a quick learning of patterns in the data. The loss curves then level off, indicating that the models begin to converge. As the number of epochs increases, both models' loss metrics plateau, showing that additional training does not significantly change the loss value. This is particularly evident in the plots for 200 and 400 epochs where the loss remains nearly constant past a certain point.Both models follow a similar pattern of loss reduction, indicating that their learning behaviors are comparable. No model appears to outperform the other significantly in terms of loss reduction, as their curves almost overlap.The stability of the loss at lower values, even for higher epoch counts, shows that the models do not overfit the training data. If overfitting occurred, we would expect the loss to increase as the model begins to memorize the training data.It seems that most of the learning occurs in the initial epochs, after which improvements are marginal. This could suggest that a smaller number of epochs could be nearly as effective for training as a larger number, potentially saving computational resources.

8 Discussion

In this project, the primary focus was on evaluating Continuous Graph Neural Networks (CGNN) using the Cora dataset for training. While the initial findings are promising, several improvements could be implemented to enhance the robustness and depth of the project:

- Comparison with Existing Models: the results would be more informative if compared against a variety of established discrete GNN models. This comparative analysis would highlight CGNN's relative performance and potential advantages.
- Use more data to show the efficiency of the Model: Expanding the experiment to include additional datasets would validate the model's efficiency and generalizability. It would also reveal dataset-specific strengths or weaknesses of the CGNN.
- Repeat the trials several times: Conducting multiple runs of the experiment and averaging the results can account for variability due to random initialization and stochastic training processes, leading to more reliable conclusions.
- In-Depth Hyperparameter Optimization: Further tuning of hyperparameters, beyond the number of epochs and hidden dimensions, could potentially yield better performance and offer insights into the model's sensitivity to these parameters.

Also, since we used the data which have the features that have nodes and edges, Should we encounter more complex datasets lacking explicit node and edge features, is it still possible to apply CGNN and WCGNN models for classification tasks? Moreover, mathematically, ODE is the branch of PDE, what if we use PDEs to do the graph neural networks? What's the efficiency between using ODEs and PDEs, Is using PDEs structure GNN more efficient than ODE or vice versa?

9 Acknowledgement

I'm grateful to Professor Kondor for providing such engaging content throughout the course. The class has been a significant learning experience for me.

I would also like to extend my gratitude to the teaching assistants, Richard Xu and Andrew Hands, for their invaluable assistance with our homework assignments!

I wish to express my sincere appreciation to the authors Xhonneux, Louis-Pascal, Qu, Meng, and Tang, Jian for their innovative contributions, which have allowed me to bridge the gap between the concepts explored in this machine learning course and my prior knowledge.

References

Xavier Bouthillier, Christos Tsirigotis, François Corneau-Tremblay, Thomas Schweizer, Lin Dong, Pierre Delaunay, Fabrice Normandin, Mirko Bronzi, Dendi Suhubdy, Reyhane Askari, Michael Noukhovitch, Chao Xue, Satya Ortiz-Gagné, Olivier Breuleux, Arnaud Bergeron, Olexa Bilaniuk, Steven Bocco, Hadrien Bertrand, Guillaume Alain, Dmitriy Serdyuk, Peter Henderson, Pascal Lamblin, and Christopher Beckham. Epistimio/orion: Asynchronous Distributed Hyperparameter Optimization, 2022.

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

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning, 2000. URL https://doi.org/10.1023/A:1009953814988.

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. Continuous graph neural networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10432–10441. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/xhonneux20a.html.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, and James S. Duncan. Ordinary differential equations on graph networks, 2020. URL https://openreview.net/forum?id=SJg9z6VFDr.