Continuous Graph Neural Networks

Louis-Pascal A. C. Xhonneux [*1 2] **Meng Qu** [*1 2] **Jian Tang** [1 3 4]

Abstract

This paper builds on the connection between graph neural networks and traditional dynamical systems. We propose *continuous graph neural networks* (CGNN), which generalise existing graph neural networks with discrete dynamics in that they can be viewed as a specific discretisation scheme. The key idea is how to characterise the continuous dynamics of node representations, i.e. the derivatives of node representations, w.r.t. time. Inspired by existing diffusion-based methods on graphs (e.g. PageRank and epidemic models on social networks), we define the derivatives as a combination of the current node representations, the representations of neighbors, and the initial values of the nodes. We propose and analyse two possible dynamics on graphs—including each dimension of node representations (a.k.a. the feature channel) change independently or interact with each other—both with theoretical justification. The proposed continuous graph neural networks are robust to over-smoothing and hence allow us to build deeper networks, which in turn are able to capture the long-range dependencies between nodes. Experimental results on the task of node classification demonstrate the effectiveness of our proposed approach over competitive baselines.

1. Introduction

Graph neural networks (GNNs) have been attracting growing interest due to their simplicity and effectiveness in a variety of applications such as node classification (Kipf and Welling, 2016; Veličković et al., 2017), link prediction (Zhang and Chen, 2018), chemical properties prediction (Gilmer et al., 2017), and natural language understanding (Marcheggiani and Titov, 2017; Yao et al., 2019). The essential idea of GNNs is to design multiple graph propagation layers to iteratively update each node representation by aggregating the node representations from their neighbours and the representation of the node itself. In practice, a few layers (two or three) are usually sufficient for most tasks (Qu et al., 2019), and more layers may lead to inferior performance (Kipf and Welling, 2016; Zhou et al., 2018; Li et al., 2018b).

A key avenue to improving GNNs is being able to build deeper networks to learn more complex relationships between the data and the output labels. The GCN propagation layer smooths the node representations, i.e. nodes near each other in the graph become more similar (Kipf and Welling, 2016). This can lead to over-smoothing as we stack more and more layers, meaning that nodes representations converge to the same value leading to worse performance (Kipf and Welling, 2016; Zhou et al., 2018; Li et al., 2018b). Thus, it is important to alleviate the node over-smoothing effect, whereby node representations converge to the same value.

Furthermore, it is crucial to improve our theoretical understanding of GNNs to enable us to characterise what signals from the graph structure we can learn. Recent work on understanding GCN (Oono and Suzuki, 2020) has considered GCN as a discrete dynamical system defined by the discrete layers. In addition, Chen et al. (2018) demonstrated that the usage of discrete layers is not the only perspective to build neural networks. They pointed out that discrete layers with residual connections can be viewed as a discretisation of a continuous Ordinary Differential Equation (ODE). They showed that this approach is more memory-efficient and is able to model the dynamics of hidden layers more smoothly.

We use the continuous perspective inspired by diffusion based methods to propose a new propagation scheme, which we analyse using tools from ordinary differential equations (i.e. continuous dynamical systems). Indeed we are able to explain what representations our model learns as well as why it does not suffer from the common over-smoothing problem observed in GNNs. Allowing us to build 'deeper' networks in the sense that our model works well with large values of time. The key factor for the resilience to over-smoothing is the use of a restart distribution as originally

[*]Equal contribution with order determined by flipping a coin. [1]Mila - Quebec AI Institute, Montréal, Canada [2]University of Montréal, Montréal, Canada [3]HEC Montréal, Montréal, Canada [4]CIFAR AI Research Chair. Correspondence to: Jian Tang <jian.tang@hec.ca>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

proposed in Pagerank (Page et al., 1999) in a continuous setting. The intuition is that the restart distribution helps to not forget the information from low powers of the adjacency matrix, hence enabling the model to converge towards a meaningful stationary distribution.

The main contributions of this paper are:

1. We propose two continuous ODEs of increasing model capacity inspired by PageRank and diffusion based methods;
2. We theoretically analyse the representation learned by our layer and show that as $t \to \infty$ our method approaches a stable fixed point, which captures the graph structure together with the original node features. Because we are stable as $t \to \infty$ our network can have an infinite number of 'layers' and is able to learn long-range dependencies;
3. We demonstrate that our model is memory efficient and is robust to the choice of t. Further, we demonstrate on the node classification against competitive baselines that our model is able to outperform many existing state-of-the-art methods.

2. Preliminaries

Let a graph $G = (V, E)$[1] be defined by vertices V and edges $E \subseteq V \times V$ between vertices in V. It is common in graph machine learning to use an adjacency matrix $\boldsymbol{Adj}$ as an alternative characterisation. Given a node ordering π and a graph G, the elements of the adjacency matrix $\boldsymbol{Adj}^{|V|\times|V|}$ are defined by the edge set E:

$$\boldsymbol{Adj}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

As the degree of nodes can be very different, we typically normalize the adjacency matrix as $\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{Adj}\boldsymbol{D}^{-\frac{1}{2}}$, where $\boldsymbol{D}$ is the degree matrix of $\boldsymbol{Adj}$. Such a normalized adjacency matrix always has an eigenvalue decomposition, and the eigenvalues are in the interval $[-1, 1]$ (Chung and Graham, 1997). The negative eigenvalues can make graph learning algorithms unstable in practice, and hence we follow Kipf and Welling (2016) and leverage the following regularized matrix for characterizing graph structures:

$$\boldsymbol{A} \coloneqq \frac{\alpha}{2}\left(\boldsymbol{I} + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{Adj}\boldsymbol{D}^{-\frac{1}{2}}\right), \tag{2}$$

where $\alpha \in (0, 1)$ is a hyperparameter, and the eigenvalues of $\boldsymbol{A}$ are in the interval $[0, \alpha]$.

Given such a matrix $\boldsymbol{A} \in \mathbb{R}^{|V|\times|V|}$ to characterise the graph structure, and a node feature matrix $\boldsymbol{X} \in \mathbb{R}^{|V|\times|F|}$, with $|F|$ being the number of node features, our goal is to learn a matrix of node representations $\boldsymbol{H} \in \mathbb{R}^{|V|\times d}$, where d is the dimension of representations, and the k-th row of $\boldsymbol{H}$ is the representation of the k-th node in an ordering π.

A continuous ODE throughout this paper will refer to an equation of the following form:

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = f(\boldsymbol{x}(t), t), \tag{3}$$

where x may be scaler, vector valued, or matrix valued and f is function that we will parametrise to define the hidden dynamic. Chen et al. (2018) showed how we can backpropagate through such an ODE equation and hence use it as a building block for a neural network.

3. Related Work

Neural ODE Neural ODE (Chen et al., 2018) is an approach for modelling a continuous dynamics on hidden representation, where the dynamic is characterised through an ODE parameterised by a neural network. However, these methods can only deal with unstructured data, where different inputs are independent. Our approach extends this in a novel way to graph structured data.

GNNs. Graph neural networks (Kipf and Welling, 2016; Veličković et al., 2017) are an effective approach for learning node representations in graphs. Typically, GNNs model discrete dynamics of node representations with multiple propagation layers, where in each layer the representation of each node is updated according to messages from neighbouring nodes. The majority of GNNs learn the relevant information from the graph structure $\boldsymbol{A}$ by learning finite polynomial filters g to apply to the eigenvalues $\boldsymbol{\Lambda}$ of the graph Laplacian $\boldsymbol{L} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$ in each propagation layer (Kipf and Welling, 2016; Defferrard et al., 2016; Wijesinghe and Wang, 2019; Veličković et al., 2017). GCN (Kipf and Welling, 2016), for instance, uses a first-order Chebyshev polynomial. However, existing GNNs (e.g. GCN) have been shown (Li et al., 2018b; Zhou et al., 2018; Oono and Suzuki, 2020) to suffer from over-smoothing, i.e. node representations start to converge to the same value. Compared to these studies, we follow continuous dynamical systems to model the continuous dynamic on node representations. Moreover, our approach does not have the over-smoothing problem, because our model converges to a meaningful representation (fixed point) as $t \to \infty$.

A proposed solution in recent work is to either add a residual connection to the preceding layer (Avelar et al., 2019) or to use a concatenation of each layer (Wijesinghe and Wang, 2019; Luan et al., 2019; Xu et al., 2018; Dehmamy et al., 2019). The latter approach does not scale to very deep networks due to the growing size of the representation. Chen et al. (2018)'s approach of turning residual connec-

[1] Throughout the paper we will only consider simple graphs

tions from the preceding layer into a continuous ODE has gathered much attention (Deng et al., 2019; Avelar et al., 2019; Zhuang et al., 2020; Poli et al., 2019). In contrast to many of these works we are able to provide a theoretical justification for our ODE and our representation does not grow with depth. (Klicpera et al., 2019a) also uses PageRank as an inspiration, however, they try to build PageRank into a single GCN layer, they also stick to the discrete version and do not consider a propagation step with learnable weights.

A significant amount of work has focused on understanding the theoretical properties of GCN and related architectures better (Oono and Suzuki, 2020; Dehmamy et al., 2019; NT and Maehara, 2019). Dehmamy et al. (2019) argue that to learn the topology of a graph the network must learn the graph moments—an ensemble average of a polynomial of $\boldsymbol{Adj}$. They show that GCN can only learn graph moments of a specific power of $\boldsymbol{Adj}$. To remedy this they concatenate $[\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \ldots, \boldsymbol{h}^{(n)}]$ the feature maps (output) of each layer. Oono and Suzuki (2020) demonstrate that deeper GCNs exponentially lose expressive power by showing that in the limit as the number of layers goes to infinity the node representation is projected onto the nullspace of the Laplacian, which implies that two nodes with identical degree in the same connected component are will have the same representation. Instead we propose a continuous dynamical system which does not suffer from this problem and demonstrate so as $t \to \infty$. NT and Maehara (2019) focus on explaining which assumptions hold such that GCN (Kipf and Welling, 2016) and SGC (Wu et al., 2019) work on the common citation networks. Our work fits into the existing literature by using the Neural ODE framework to provide a novel intuition to what is needed to address loss of expressive power with depth. We do this by proposing and analysing our specific solution.

Concurrent work. Several concurrent works have developped similar ideas; (Poli et al., 2019) proposes an ODE based on treating the GCN-layer as a continuous vector field and combines discrete and continuous layers. Other concurrent works (Deng et al., 2019; Zhuang et al., 2020) use the Neural ODE framework and parametrise the derivative function using a 2- or 3-layer GNN directly, instead we develop a continuous message-passing layer and do not use a discrete deep neural network to parametrise the derivative. In contrast, we motivate our ODE from diffusion based methods and theoretically justify how our approach helps to address over-smoothing as well as long-range connections between nodes.

4. Model

In this section, we introduce our proposed approach. The goal is to learn informative node representations $\boldsymbol{H} \in \mathbb{R}^{|V| \times d}$ for node classification. We first employ a neural encoder to project each node into a latent space based on its features, i.e. $\boldsymbol{E} = \mathcal{E}(\boldsymbol{X})$. Afterwards, $\boldsymbol{E}$ is treated as the initial value $\boldsymbol{H}(0)$, and an ODE is designed to define the continuous dynamics on node representations, where the long-term dependency between nodes can be effectively modelled. Finally, the node representations obtained at ending time t_1 (i.e. $\boldsymbol{H}(t_1)$) can be used for downstream applications through a decoder $\mathcal{D}$. The overall model architecture is summarized in Fig. 1.

The key step of the framework is to design an effective ODE for defining the continuous dynamics on node representations, and thereby modelling the dependency of nodes. We design two such ODEs of increasing model capacity based on the intuition from existing diffusion-based methods on graphs. In the first ODE, each feature channel (i.e. dimension) of node representations evolves independently (see Sec. 4.1), whereas in the second ODE we also model the interaction of different feature channels (see Sec. 4.2).

4.1. Case 1: Independent Feature Channels

Since different nodes in a graph are interconnected, a desirable ODE should take the graph structure into consideration, and allow information to propagate among different nodes. Motivated by existing diffusion-based methods on graphs (e.g. PageRank (Page et al., 1999) and label propagation (Zhou et al., 2004)), an effective way for characterizing the propagation process is to use the following step-wise propagation equations:

$$\boldsymbol{H}_{n+1} = \boldsymbol{A}\boldsymbol{H}_n + \boldsymbol{H}_0, \tag{4}$$

where $\boldsymbol{H}_n \in \mathbb{R}^{|V| \times d}$ is the embedding matrix for all the nodes at step n, and $\boldsymbol{H}_0 = \boldsymbol{E} = \mathcal{E}(\boldsymbol{X})$ is the embedding matrix computed by the encoder $\mathcal{E}$. Intuitively, each node at stage $n+1$ learns the node information from its neighbours through $\boldsymbol{A}\boldsymbol{H}_n$ and remembers its original node features through $\boldsymbol{H}_0$. This allows us to learn the graph structure without forgetting the original node features. The explicit formula of Eq. 4 can be derived as follows:

$$\boldsymbol{H}_n = \left(\sum_{i=0}^{n} \boldsymbol{A}^i\right) \boldsymbol{H}_0 = (\boldsymbol{A} - \boldsymbol{I})^{-1}(\boldsymbol{A}^{n+1} - \boldsymbol{I})\boldsymbol{H}_0, \tag{5}$$

where we see that the representation $\boldsymbol{H}_n$ at step n incorporates all the information propagated up to n steps with the initial representation $\boldsymbol{H}_0$.

Because of the effectiveness of the discrete propagation process in Eq. (5), we aim to extend the process to continuous cases by replacing n with a continuous variable $t \in \mathbb{R}_0^+$, and further use an ODE to characterise such a continuous propagation dynamic. Intuitively, we view the summation in Eq. (5) as a Riemann sum of an integral from time 0 to time $t = n$, which allows us to naturally move from the discrete

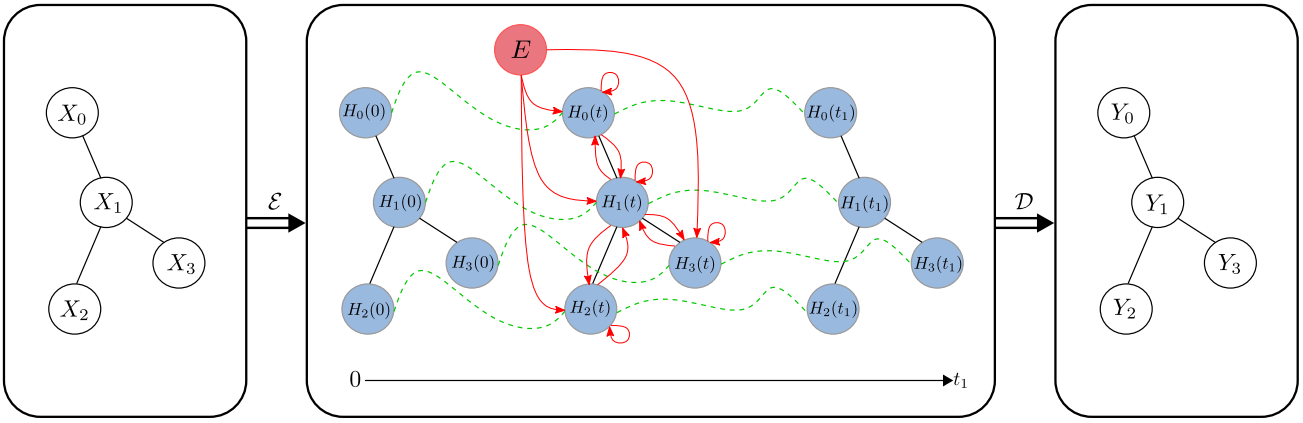


Figure 1: Architecture overview: The input to the model is a graph with node features, we initially encode these node features using a single neural network layer and ignoring the graph structure. Then we use a differential equation to change the representation over time, before projecting the representation using another single neural network layer and a softmax function to a one-hot encoding of the classes. The red lines represent the information transfer as defined by the ODE.

propagation process to the continuous case, as stated in the following proposition.

Proposition 1 *The discrete dynamic in Eq.* (5)*, where* $\boldsymbol{A}$ *has an eigenvalue decomposition, is a discretisation of the following ODE:*

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = \ln \boldsymbol{A}\boldsymbol{H}(t) + \boldsymbol{E}, \tag{6}$$

with the initial value $\boldsymbol{H}(0) = (\ln \boldsymbol{A})^{-1}(\boldsymbol{A} - \boldsymbol{I})\boldsymbol{E}$*, where* $\boldsymbol{E} = \mathcal{E}(\boldsymbol{X})$ *is the output of the encoder* $\mathcal{E}$*.*

We provide the proof in the Supplementary material. In practice, $\ln \boldsymbol{A}$ in Eq. (6) is intractable to compute, hence we approximate it using the first order of the Taylor expansion, i.e. $\ln \boldsymbol{A} \approx (\boldsymbol{A} - \boldsymbol{I})$, which gives us:

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = (\boldsymbol{A} - \boldsymbol{I})\boldsymbol{H}(t) + \boldsymbol{E}, \tag{7}$$

with the initial value being $\boldsymbol{H}(0) = \boldsymbol{E}$. This is the ODE we use in our model **CGNN**. The intuition behind the ODE defined in Eq. 7 can be understood from an epidemic modelling perspective. The epidemic model aims at studying the dynamics of infection in a population. Typically, the model assumes that the infection of people is affected by three factors, i.e. the infection from neighbours, the natural recovery, and the natural physique of people. Suppose that we treat the latent vectors $\boldsymbol{H}(t)$ as the infection conditions of a group of people at time t, then the three factors can be naturally captured by three terms: $\boldsymbol{A}\boldsymbol{H}(t)$ for the infection from neighbours, $-\boldsymbol{H}(t)$ for natural recovery, and $\boldsymbol{E}$ for the natural physique. Therefore, the infection dynamics in a population can be intuitively modelled by our first-order ODE, in Eq. (6), indicating that the intuition of our ODE agrees with the epidemic model.

The ODE we use can be understood theoretically. Specifically, the node representation matrix $\boldsymbol{H}(t)$ at time t has an analytical form, which is formally stated in the following proposition.

Proposition 2 *The analytical solution of the ODE defined in Eq.* (7) *is given by:*

$$\boldsymbol{H}(t) = (\boldsymbol{A} - \boldsymbol{I})^{-1}(e^{(\boldsymbol{A}-\boldsymbol{I})t} - \boldsymbol{I})\boldsymbol{E} + e^{(\boldsymbol{A}-\boldsymbol{I})t}\boldsymbol{E} \tag{8}$$

We prove the proposition in the Supplementary material. From the proposition, since the eigenvalues of $\boldsymbol{A} - \boldsymbol{I}$ are in the interval $[-1, 0)$, as we increase t to ∞, the exponential term $e^{(\boldsymbol{A}-\boldsymbol{I})t}$ will approach $\mathbf{0}$, i.e. $\lim_{t\to\infty} e^{(\boldsymbol{A}-\boldsymbol{I})t} = \mathbf{0}$. Therefore, for large enough t we can approximate $\boldsymbol{H}(t)$ as:

$$\boldsymbol{H}(t) \approx (\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{E} = \left(\sum_{i=0}^{\infty} \boldsymbol{A}^i\right)\boldsymbol{E}. \tag{9}$$

Thus, $\boldsymbol{H}(t)$ can be seen as the summation of all different orders of propagated information (i.e. $\{\boldsymbol{A}^i\boldsymbol{E}\}_{i=1}^{\infty}$). In this way, our approach essentially has an infinite number of discrete propagation layers, allowing us to model global dependencies of nodes more effectively than existing GNNs.

Implementation. In the node classification task, our decoder $\mathcal{D}$ to compute the node-label matrix $\boldsymbol{Y} = \mathcal{D}(\boldsymbol{H}(t_1))$ is a softmax classifier with the ReLU activation function (Nair and Hinton, 2010).

Note the parameter α in Eq. (2) decides the eigenvalues of $\boldsymbol{A}$, which thereby determines how quickly the higher order powers of $\boldsymbol{A}$ go to 0, this also means that by specifying α per node we can control how much of the neighbourhood each node gets to see as smaller values α imply that the powers of $\boldsymbol{A}$ vanish faster. In our final model we learn these parameters α.

4.2. Case 2: Modelling the Interaction of Feature Channels

The ODE so far models different feature channels (i.e. dimensions of hidden representations) independently, where different channels are not able to interact with each other, and thus the ODE may not capture the correct dynamics of the graph. To allow the interaction between different feature channels, we are inspired by the success of a linear variant of GCN (i.e. Simple GCN (Wu et al., 2019)) and consider a more powerful discrete dynamic:

$$\boldsymbol{H}_{n+1} = \boldsymbol{A}\boldsymbol{H}_n\boldsymbol{W} + \boldsymbol{H}_0, \tag{10}$$

where $\boldsymbol{W} \in \mathbb{R}^{d\times d}$ is a weight matrix. Essentially, with $\boldsymbol{W}$, we are able to model the interactions of different feature channels during propagation, which increases the model capacity and allows us to learn more effectively representations of nodes.

Similar to the previous section, we extend the discrete propagation process in Eq. (10) to continuous cases by viewing each $\boldsymbol{H}_n$ as a Riemann sum of an integral from time 0 to time $t = n$, which yields the following proposition:

Proposition 3 *Suppose that the eigenvalue decompositions of $\boldsymbol{A}, \boldsymbol{W}$ are $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$ and $\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Phi}\boldsymbol{Q}^{-1}$, respectively, then the discrete dynamic in Eq.* (10) *is a discretisation of the following ODE:*

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = \ln \boldsymbol{A}\boldsymbol{H}(t) + \boldsymbol{H}(t)\ln \boldsymbol{W} + \boldsymbol{E}, \tag{11}$$

where $\boldsymbol{E} = \mathcal{E}(\boldsymbol{X})$ is the output of the encoder $\mathcal{E}$ and with the initial value $\boldsymbol{H}(0) = \boldsymbol{P}\boldsymbol{F}\boldsymbol{Q}^{-1}$, where

$$F_{ij} = \frac{\Lambda_{ii}\widetilde{E}_{ij}\Phi_{jj} - \widetilde{E}_{ij}}{\ln \Lambda_{ii}\Phi_{jj}}, \tag{12}$$

where $\widetilde{\boldsymbol{E}} = \boldsymbol{P}^{-1}\boldsymbol{E}\boldsymbol{Q}$.

The proof is provided in the Supplementary material. By making a first-order Taylor approximation to get rid of the matrix logarithm, we further obtain:

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = (\boldsymbol{A} - \boldsymbol{I})\boldsymbol{H}(t) + \boldsymbol{H}(t)(\boldsymbol{W} - \boldsymbol{I}) + \boldsymbol{E}, \tag{13}$$

with the initial value being $\boldsymbol{H}(0) = \boldsymbol{E}$. This is the ODE we use in our model **CGNN** with weights. The ODEs of the form as in Eq. (13) have been studied in some detail in the control theory literature, where they are known as the Sylvester differential equation (Locatelli, 2001; Behr et al., 2019). Intuitively, $\boldsymbol{E}$ here would be the input into the system with the goal to get the system $\boldsymbol{H}$ into a desired state $\boldsymbol{H}(t)$. The matrices $\boldsymbol{A} - \boldsymbol{I}$ and $\boldsymbol{W} - \boldsymbol{I}$ describe the natural evolution of the system.

The ODE in Eq. (13) also has appealing theoretical properties. Specifically, $\boldsymbol{H}(t)$ has an analytical form as shown in the following proposition.

Proposition 4 *Suppose that the eigenvalue decompositions of $\boldsymbol{A} - \boldsymbol{I}, \boldsymbol{W} - \boldsymbol{I}$ are $\boldsymbol{A} - \boldsymbol{I} = \boldsymbol{P}\boldsymbol{\Lambda}'\boldsymbol{P}^{-1}$ and $\boldsymbol{W} - \boldsymbol{I} = \boldsymbol{Q}\boldsymbol{\Phi}'\boldsymbol{Q}^{-1}$, respectively, then the analytical soluton of the ODE in Eq.* (13) *is given by:*

$$\boldsymbol{H}(t) = e^{(\boldsymbol{A}-\boldsymbol{I})t}\boldsymbol{E}e^{(\boldsymbol{W}-\boldsymbol{I})t} + \boldsymbol{P}\boldsymbol{F}(t)\boldsymbol{Q}^{-1}, \tag{14}$$

where $\boldsymbol{F}(t) \in \mathbb{R}^{|V|\times d}$ with each element defined as follows:

$$F_{ij}(t) = \frac{\widetilde{E}_{ij}}{\Lambda'_{ii} + \Phi'_{jj}} e^{t(\Lambda'_{ii}+\Phi'_{jj})} - \frac{\widetilde{E}_{ij}}{\Lambda'_{ii} + \Phi'_{jj}} \tag{15}$$

where $\widetilde{\boldsymbol{E}} = \boldsymbol{P}^{-1}\boldsymbol{E}\boldsymbol{Q}$.

We prove the proposition in the Supplementary material. According to the definition of $\boldsymbol{A}$ and also our assumption about $\boldsymbol{W}$, the eigenvalues of $\boldsymbol{A} - \boldsymbol{I}$ and $\boldsymbol{W} - \boldsymbol{I}$ are in $(-1, 0)$, and therefore $\Lambda'_{i,i} < 0$ for every i and $\Phi'_{j,j} < 0$ for every j. Hence, as we increase t to ∞, the exponential terms will approach 0, and hence for large enough t we can approximate $\boldsymbol{H}(t)$ as:

$$\left(\boldsymbol{P}^{-1}\boldsymbol{H}(t)\boldsymbol{Q}\right)_{ij} \approx -\frac{\widetilde{E}_{ij}}{\Lambda'_{ii} + \Phi'_{jj}}. \tag{16}$$

Based on the above results, if $\boldsymbol{W} = \boldsymbol{I}$, then $\boldsymbol{H}(t)$ will converge to the same result as in Eq. (9), and hence the ODE defined in Eq. (6) is a special case of the ODE in Eq. (11).

Implementation. We use the same decoder as for the case when $W = I$.

In practice, to enforce $\boldsymbol{W}$ to be a diagonalisable matrix with all the eigenvalues less than 1, we parameterise $\boldsymbol{W}$ as $\boldsymbol{W} = \boldsymbol{U}\mathrm{diag}(\boldsymbol{M})\boldsymbol{U}^T$, where $\boldsymbol{U} \in \mathbb{R}^{d\times d}$ is a learnable orthogonal matrix and $\boldsymbol{M} \in \mathbb{R}^d$ is a learnable vector, characterising the eigenvalues of $\boldsymbol{W}$. During training, we clamp the values of $\boldsymbol{M}$ to guarantee they are between $(0, 1)$. To ensure $\boldsymbol{U}$ to be an orthogonal matrix, we follow previous work in (Cisse et al., 2017; Conneau et al., 2017) and perform the following secondary update of $\boldsymbol{U}$ after each main update during training:

$$\boldsymbol{U} \leftarrow (1 + \beta)\boldsymbol{U} - \beta(\boldsymbol{U}\boldsymbol{U}^T)\boldsymbol{U}, \tag{17}$$

where β is a hyperparameter, and the above secondary update enables $\boldsymbol{U}$ to be close to the manifold of orthogonal matrices after each training step.

Finally, to help stabilise training we use the idea from (Dupont et al., 2019) and add auxiliary dimensions to hidden representation only during the continuous propagation

Table 1: Statistics of datasets.

Dataset	# Nodes	# Edges	# Features	# Classes	Label Rate
Cora	2,708	5,429	1,433	7	0.036
Citeseer	3,327	4,732	3,703	6	0.052
Pubmed	19,717	44,338	500	3	0.003
NELL	65,755	266,144	5,414	210	0.001

process. Specifically, we double the latent representation initialising the second half of the initial representation with 0 and throwing the result away after solving the continuous ODE. This very slightly improves results, but importantly stabilises training significantly (see Supplementary material).

5. Discussion

Our continuous model for information propagation has several advantages over previous discrete GNNs such as GCN:

1. Robustness with time to over-smoothing;
2. Learning global dependencies in the graph;
3. α represents the "diffusion" constant, which is learned;
4. Weights entangle channels continuously over time;
5. Insight into the role of the restart distribution $\boldsymbol{H}_0$.

1. Robustness with time to over-smoothing: Despite the effectiveness of the discrete propagation process, it has been shown in (Li et al., 2018b) that the usage of discrete GCN layers can be tricky as the number n of layers (time in the continuous case) is a critical choice. Theoretical work in (Oono and Suzuki, 2020) further showed that on dense graphs as the number of GCN layers grow there is exponential information loss in the node representations. In contrast, our method is experimentally not very sensitive to the integration time chosen and theoretically does not suffer from information loss as time goes to infinity.

2. Global dependencies: Recent work (Klicpera et al., 2019b; Dehmamy et al., 2019; Xu et al., 2018) has shown that to improve on GNN it is necessary to build deeper networks to be able to learn long-range dependencies between nodes. Our work, thanks to the stability with time is able to learn global dependencies between nodes in the graph. Eq. (9) demonstrates that we propagate the information from all powers of the adjacency matrix, thus we are able to learn global dependencies.

3. Diffusion constant: The parameter α scales the matrix $\boldsymbol{A}$ (see Eq. (2)), i.e. it controls the rate of diffusion. Hence, α controls the rate at which higher-order powers of $\boldsymbol{A}$ vanish. Since each node has its own parameter α that is learned, our model is able to control the diffusion, i.e. the weight of higher-order powers, for each node independently.

4. Entangling channels during graph propagation: The ODE with weights (Eq. (11)) allows the model to entangle the information from different channels over time. In addition, we are able to explain how the eigenvalues of the weight matrix affect the learned representation (Eq. (14)).

5. Insight into the role of the restart distribution $\boldsymbol{H}_0$:

In both of our ODEs, Eq. (7) and (13), the derivative depends on $\boldsymbol{E}$, which equals to the initial value $\boldsymbol{H}(0)$. To intuitively understand the effect of the initial value in our ODEs, consider an ODE without $\boldsymbol{E}$, i.e. $\boldsymbol{H}'(t) = (\boldsymbol{A} - I)\boldsymbol{H}(t)$. The analytical solution to the ODE $\boldsymbol{H}'(t) = (\boldsymbol{A} - I)\boldsymbol{H}(t)$ is given by $\boldsymbol{H}(t) = \exp[(\boldsymbol{A} - \boldsymbol{I})t]\boldsymbol{H}(0)$. Remembering that $\boldsymbol{A} - \boldsymbol{I}$ is simply a first order approximation of $\ln \boldsymbol{A}$, we can see that the analytical solution we are trying to approximate is $\boldsymbol{H}(t) = \boldsymbol{A}^t\boldsymbol{H}(0)$. Thus, the end time of the ODE now determines, which power of the $\boldsymbol{Adj}$ we learn. Indeed in our experiments we show that removing the term $\boldsymbol{H}(0)$ causes us to become very sensitive to the end time chosen rather than just needing a sufficiently large value (see Fig. 2).

6. Experiment

In this section, we evaluate the performance of our proposed approach on the semi-supervised node classification task.

6.1. Datasets and Experiment Settings

In our experiment, we use four benchmark datasets for evaluation, including Cora, Citeseer, Pubmed, and NELL. Following existing studies (Yang et al., 2016; Kipf and Welling, 2016; Veličković et al., 2017), we use the standard data splits from (Yang et al., 2016) for Cora, Citeseer and Pubmed, where 20 nodes of each class are used for training and another 500 labeled nodes are used for validation. For the NELL dataset, as the data split used in (Yang et al., 2016) is not available, we create a new split for experiment. The results are in Table 2. We further run experiments with random splits on the same datasets in Table 3. The statistics of the datasets are summarized in Table 1. Accuracy is used as the evaluation metric.

Table 2: Node classification results on citation networks. $\star$ The values are taken from the original paper. $\star\star$ There is no standard test-validation-training split on this dataset, hence we generated a random one and used it across all experiments. $\star\star\star$ GAT ran out of memory on NELL.

Model	Cora	Citeseer	Pubmed	NELL**
GAT-GODE*	83.3 ± 0.3	72.1 ± 0.6	79.1 ± 0.5	-
GCN-GODE*	81.8 ± 0.3	72.4 ± 0.8	80.1 ± 0.3	-
GCN	81.8 ± 0.8	70.8 ± 0.8	80.0 ± 0.5	57.4 ± 0.7
GAT***	82.6 ± 0.7	71.5 ± 0.8	77.8 ± 0.6	-
CGNN discrete	81.8 ± 0.6	70.0 ± 0.5	81.0 ± 0.4	50.9 ± 3.9
CGNN	$\mathbf{84.2 \pm 1.0}$	72.6 ± 0.6	$\mathbf{82.5 \pm 0.4}$	65.4 ± 1.0
CGNN with weight	83.9 ± 0.7	$\mathbf{72.9 \pm 0.6}$	82.1 ± 0.5	$\mathbf{65.6 \pm 0.9}$

Table 3: Node classification results on citation networks over 15 random splits. $\star$ GAT ran out of memory on NELL.

Model	Cora	Citeseer	Pubmed	NELL
GCN	80.9 ± 1.1	72.0 ± 1.1	81.5 ± 1.4	64.8 ± 1.8
GAT*	78.2 ± 1.4	71.9 ± 1.4	79.8 ± 1.6	-
CGNN discrete	81.7 ± 1.5	70.3 ± 1.6	81.8 ± 1.7	69.5 ± 1.4
CGNN	$\mathbf{82.7 \pm 1.2}$	72.7 ± 0.9	$\mathbf{83.2 \pm 1.4}$	$\mathbf{73.4 \pm 1.0}$
CGNN with weight	82.1 ± 1.3	$\mathbf{72.9 \pm 0.9}$	82.7 ± 1.4	73.1 ± 0.9

6.2. Compared Algorithms

Discrete GNNs: For standard graph neural networks which model the discrete dynamic of node representations, we mainly compare with the Graph Convolutional Network (**GCN**) (Kipf and Welling, 2016) and the Graph Attention Network (**GAT**) (Veličković et al., 2017), which are the most representative methods.

Continuous GNNs: There is also a recent concurrent work (Zhuang et al., 2020) which learns node representations through modelling the continuous dynamics of node representations, where the ODE is parameterised by a graph neural network. We also compare with this method (**GODE**).

CGNN: For our proposed Continuous Graph Neural Network (CGNN), we consider a few variants. Specifically, **CGNN** leverages the ODE in Eq. (7) to define the continuous dynamic of node representations, where different feature channels are independent. **CGNN with weight** uses the ODE in Eq. (13), which allows different feature channels to interact with each other. We also compare with **CGNN discrete**, which uses the discrete propagation process defined in Eq. (4) for node representation learning (with $n = 50$).

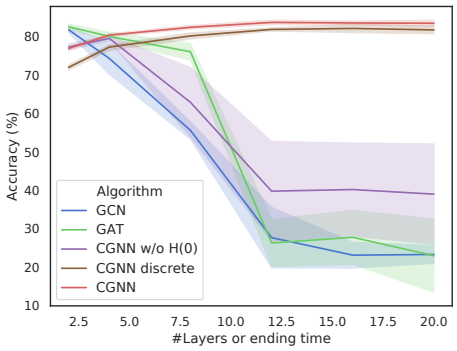
6.3. Parameter Settings

We do a random hyperparameter search using the ORION (Bouthillier et al., 2019; Li et al., 2018a) framework with 50 retries. The mean accuracy over 10 runs is reported for each dataset in Table 2.

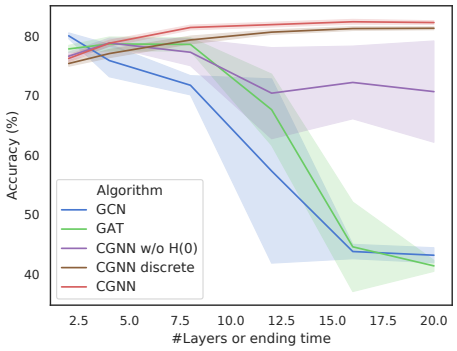
6.4. Results

1. Comparison with existing methods. The main results of different compared algorithms are summarized in Table 2. Compared with standard discrete graph neural networks, such as GCN and GAT, our approach achieves significantly better results in most cases. The reason is that our approach can better capture the long-term dependency of different nodes. Besides, our approach also outperforms the concurrent work GODE on Cora and Pubmed. This is because our ODEs are designed based on our prior knowledge about information propagation in graphs, whereas GODE parameterises the ODE by straightforwardly using an existing graph neural network (e.g. GCN or GAT), which may not effectively learn to propagate information in graphs. Overall, our approach achieves comparable results to state-of-the-art graph neural networks on node classification.

2. Comparison of CGNN and its variants. The ODEs in CGNN are inspired by the discrete propagation process in Eq. (4), which can be directly used for modelling the dynamic on node representations. Compared with this variant (CGNN discrete), CGNN achieves much better results on all the datasets, showing that modelling the dynamic on nodes continuously is more effective for node representation learning. Furthermore, comparing the ODEs with or without modelling the interactions of feature channels (CGNN with weight and CGNN respectively), we see that their results are close. A possible reason is that the datasets used in experiments are quite easy, and thus modelling the interactions of feature channels (CGNN with weight) does

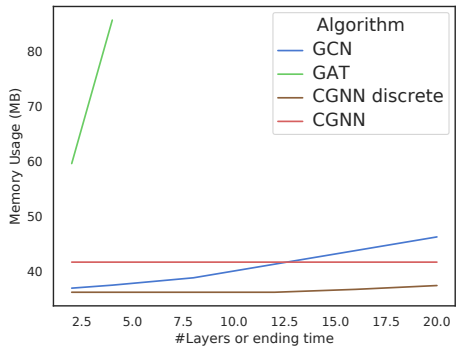


(a) Cora

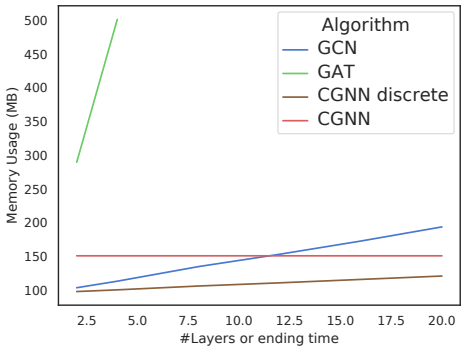


(b) Pubmed

Figure 2: Performance w.r.t. #layers or ending time. Note that the red line also has error bars, but they are very small.



(a) Cora



(b) Pubmed

Figure 3: Memory usage w.r.t. #layers or ending time.

not bring much gain in performance. We anticipate CGNN with weight could be more effective on more challenging graphs, and we leave it as future work to verify this.

3. Performance with respect to time steps. One major advantage of CGNN over existing methods is that it is robust to the over-smoothing problem. Next, we systematically justify this point by presenting the performance of different methods under different numbers of layers (e.g. GCN and GAT) or the ending time t (e.g. CGNN and its variants).

The results on Cora and Pubmed are presented in Fig. 2. For GCN and GAT, the optimal results are achieved when the number of layers is 2 or 3. If we stack more layers, the results drop significantly due to the over-smoothing problem. Therefore, GCN and GAT are only able to leverage information within 3 steps for each node to learn the representation. In contrast to them, the performance of CGNN is more stable and the optimal results are achieved when $t > 10$, which shows that CGNN is robust to over-smoothing and can effectively model long-term dependencies of nodes. To demonstrate this we use the ODE $\boldsymbol{H}'(t) = (\boldsymbol{A} - \boldsymbol{I})\boldsymbol{H}(0)$ with $\boldsymbol{H}(0) = \boldsymbol{E}$, which gets much worse results (CGNN w/o $\boldsymbol{H}(0)$), showing the importance of the initial value for modelling the continuous dynamic. Finally, CGNN also outperforms the variant which directly models the discrete dynamic in Eq. (4) (CGNN discrete), which demonstrates the advantage of our continuous approach.

4. Memory Efficiency. Finally, we compare the memory efficiency of different methods on Cora and Pubmed in Fig. 3. For all the methods modelling the discrete dynamic, i.e. GCN, GAT, and CGNN discrete, the memory cost is linear to the number of discrete propagation layers. In contrast, through using the adjoint method (Pontryagin, 2018) for optimization, CGNN has a constant memory cost and the cost is quite small, which is hence able to model long-term node dependency on large graphs.

7. Conclusion

In this paper, we build the connection between recent graph neural networks and traditional dynamic systems. Based on the connection, we further propose continuous graph neural networks (CGNNs), which generalise existing discrete graph neural networks to continuous cases through defining the evolution of node representations with ODEs. Our ODEs are motivated by existing diffusion-based methods on

graphs, where two different ways are considered, including different feature channels change independently or interact with each other. Extensive theoretical and empirical analysis prove the effectiveness of CGNN over many existing methods. Our current approach assumes that connected nodes are similar ('homophily' assumption), we leave it for future work to be able to learn more complex non-linear relationships such as can be found in molecules (Gilmer et al., 2017) or knowledge graphs (Sun et al., 2019).

ACKNOWLEDGEMENT

This project is supported by the Natural Sciences and Engineering Research Council (NSERC) Discovery Grant, the Canada CIFAR AI Chair Program, collaboration grants between Microsoft Research and Mila, Amazon Faculty Research Award, Tencent AI Lab Rhino-Bird Gift Fund and a NRC Collaborative R&D Project (AI4D-CORE-06).

We also would like to thank Joey Bose, Alexander Tong, Emma Rocheteau, and Andreea Deac for useful comments on the manuscript and Sékou-Oumar Kaba for pointing out a mistake in one equation.

References

Pedro H. C. Avelar, Anderson R. Tavares, Marco Gori, and Luis C. Lamb. Discrete and continuous deep residual learning over graphs, 2019.

Maximilian Behr, Peter Benner, and Jan Heiland. Solution formulas for differential sylvester and lyapunov equations. *Calcolo*, 56(4):51, 2019.

Xavier Bouthillier, Christos Tsirigotis, François Corneau-Tremblay, Pierre Delaunay, Michael Noukhovitch, Reyhane Askari, Peter Henderson, Dendi Suhubdy, Frédéric Bastien, and Pascal Lamblin. Oríon - Asynchronous Distributed Hyperparameter Optimization. `https://github.com/Epistimio/orion`, September 2019.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf`.

Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 854–863. JMLR. org, 2017.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*, 2017.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2016.

Nima Dehmamy, Albert-László Barabási, and Rose Yu. Understanding the representation power of graph neural networks in learning graph topology. In *Advances in Neural Information Processing Systems*, pages 15387–15397, 2019.

Zhiwei Deng, Megha Nawhal, Lili Meng, and Greg Mori. Continuous graph flow for flexible density estimation. *arXiv preprint arXiv:1908.02436*, 2019.

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

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

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

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019a. URL `https://openreview.net/forum?id=H1gL-2A9Ym`.

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning, 2019b.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning, 2018a. URL `https://openreview.net/forum?id=S1Y7OOlRZ`.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018b.

Arturo Locatelli. *Optimal Control. An Introduction*. Birkhauser Verlag, 2001.

Sitao Luan, Mingde Zhao, Xiao-Wen Chang, and Doina Precup. Break the ceiling: Stronger multi-scale deep graph convolutional networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 10943–10953. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9276-break-the-ceiling-stronger-multi-scale-deep-graph-convolutional-networks.pdf.

Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1506–1515, 2017.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters, 2019.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1ldO2EFPr.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

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

Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

Meng Qu, Yoshua Bengio, and Jian Tang. Gmnn: Graph markov neural networks. In *ICML*, 2019.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

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

W. O. K. Asiri Suranga Wijesinghe and Qing Wang. Dfnets: Spectral cnns for graphs with feedback-looped filters. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 6007–6018. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/8834-dfnets-spectral-cnns-for-graphs-with-feedback-looped-filters.pdf.

Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

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

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7370–7377, 2019.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328, 2004.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

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

A. Proof of Proposition 1 and 2

Proof of Proposition 1: The starting point is to see

$$\boldsymbol{H}_n = \left(\sum_{i=0}^{n} \boldsymbol{A}^i \right) \boldsymbol{H}_0 \tag{18}$$

as a Riemann sum, i.e.:

$$\sum_{i=1}^{n+1} \boldsymbol{A}^{0+(i-1)\cdot\Delta t} \boldsymbol{E} \Delta t, \tag{19}$$

where $\Delta t = \frac{t+1-0}{n+1}$ with $t = n$ and $\boldsymbol{E} = \boldsymbol{H}_0$ as before. So now letting $n \to \infty$ we get the following integral

$$\boldsymbol{H}(t) = \int_0^{t+1} \boldsymbol{A}^s \boldsymbol{E} \mathrm{d}s, \tag{20}$$

The derivative is then given by

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = \boldsymbol{A}^{t+1} \boldsymbol{E}. \tag{21}$$

However, $\boldsymbol{A}^{t+1}$ is intractable in practice to compute for non-integer t, hence we solve the ODE by considering the second order ODE and then integrating again.

$$\frac{\mathrm{d}^2\boldsymbol{H}(t)}{\mathrm{d}t^2} = \ln \boldsymbol{A} \boldsymbol{A}^{t+1} \boldsymbol{E} = \ln \boldsymbol{A} \frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} \tag{22}$$

Now, integrating again

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = \ln \boldsymbol{A} \boldsymbol{H}(t) + \mathit{const} \tag{23}$$

and solving for the constant using the fact that

$$\boldsymbol{H}(0) = \int_0^1 \boldsymbol{A}^s \boldsymbol{E} \mathrm{d}s = \frac{\boldsymbol{A} - \boldsymbol{I}}{\ln \boldsymbol{A}} \boldsymbol{E}, \tag{24}$$

we get that

$$\left. \frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} \right|_{t=0} = \boldsymbol{A}\boldsymbol{E} = \ln \boldsymbol{A} \boldsymbol{H}(0) + \mathit{const} \implies \mathit{const} = \boldsymbol{E}. \tag{25}$$

Thus, we get the final ODE

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = \ln \boldsymbol{A} \boldsymbol{H}(t) + \boldsymbol{E} \tag{26}$$

□

Proof of Proposition 2: To solve the ODE

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = (\boldsymbol{A} - \boldsymbol{I})\boldsymbol{H}(t) + \boldsymbol{E} \tag{27}$$

we will make use of a *Ansatz* and use the integrating factor $\exp(-(\boldsymbol{A} - \boldsymbol{I})t)$.

$$e^{-(\boldsymbol{A}-\boldsymbol{I})t} \frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = e^{-(\boldsymbol{A}-\boldsymbol{I})t} (\boldsymbol{A} - \boldsymbol{I}) \boldsymbol{H}(t) + e^{-(\boldsymbol{A}-\boldsymbol{I})t} \boldsymbol{E} \tag{28}$$

$$e^{-(\boldsymbol{A}-\boldsymbol{I})t} \frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} - e^{-(\boldsymbol{A}-\boldsymbol{I})t} (\boldsymbol{A} - \boldsymbol{I}) \boldsymbol{H}(t) = e^{-(\boldsymbol{A}-\boldsymbol{I})t} \boldsymbol{E} \tag{29}$$

$$e^{-(\boldsymbol{A}-\boldsymbol{I})t} \boldsymbol{H}(t) - \boldsymbol{E} = -(\boldsymbol{A} - \boldsymbol{I})^{-1} (e^{-(\boldsymbol{A}-\boldsymbol{I})t} - I) \boldsymbol{E} \tag{30}$$

$$\boldsymbol{H}(t) = (\boldsymbol{A} - \boldsymbol{I})^{-1} (e^{(\boldsymbol{A}-\boldsymbol{I})t} - I) \boldsymbol{E} + e^{(\boldsymbol{A}-\boldsymbol{I})t} \boldsymbol{E} \tag{31}$$

□

B. Proof of Proposition 3 and 4

To prove *Proposition* 3 and 4, we first prove the following Lemmata:

Lemma 1 *The analytical solution of the following ODE,*

$$\frac{\partial \boldsymbol{H}(t)}{\partial t} = \boldsymbol{B}\boldsymbol{H}(t) + \boldsymbol{H}(t)\boldsymbol{C} + \boldsymbol{D}, \tag{32}$$

where $\boldsymbol{B}$ and $\boldsymbol{C}$ have eigenvalue decompositions $\boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$ and $\boldsymbol{Q}\boldsymbol{\Phi}\boldsymbol{Q}^{-1}$ respectively, with initial value $\boldsymbol{H}(0)$ is:

$$\boldsymbol{H}(t) = e^{\boldsymbol{B}t}\boldsymbol{H}(0)e^{\boldsymbol{C}t} + \boldsymbol{P}\boldsymbol{F}(t)\boldsymbol{Q}^{-1}, \tag{33}$$

where

$$F_{ij}(t) = \frac{\widetilde{D}_{ij}}{\Lambda_{ii} + \Phi_{jj}} e^{t(\Lambda_{ii}+\Phi_{jj})} - \frac{\widetilde{D}_{ij}}{\Lambda_{ii} + \Phi_{jj}}. \tag{34}$$

with $\widetilde{\boldsymbol{D}} = \boldsymbol{P}^{-1}\boldsymbol{D}\boldsymbol{Q}$.

Proof: First note that the ODE in Eq. (32) is known as the Sylvester ODE (Behr et al., 2019) with analytical solution:

$$\boldsymbol{H}(t) = e^{\boldsymbol{B}t}\boldsymbol{H}(0)e^{\boldsymbol{C}t} + \int_0^t e^{\boldsymbol{B}(t-s)}\boldsymbol{D}e^{\boldsymbol{C}(t-s)}\mathrm{d}s. \tag{35}$$

Hence, to prove Lemma 1 it remains to solve the integral using the help our assumptions.

$$\int_0^t e^{\boldsymbol{B}(t-s)}\boldsymbol{D}e^{\boldsymbol{C}(t-s)}\mathrm{d}s = \int_0^t \boldsymbol{P}e^{\boldsymbol{\Lambda}(t-s)}\boldsymbol{P}^{-1}\boldsymbol{D}\boldsymbol{Q}e^{\boldsymbol{\Phi}(t-s)}\boldsymbol{Q}^{-1}\mathrm{d}s \tag{36}$$

Let $\widetilde{\boldsymbol{D}} = \boldsymbol{P}^{-1}\boldsymbol{D}\boldsymbol{Q}$,

$$= \boldsymbol{P}\left(\int_0^t e^{\boldsymbol{\Lambda}(t-s)}\widetilde{\boldsymbol{D}}e^{\boldsymbol{\Phi}(t-s)}\mathrm{d}s\right)\boldsymbol{Q}^{-1} \tag{37}$$

Considering the integral element-wise, we get

$$\int_0^t \left(e^{\boldsymbol{\Lambda}(t-s)}\widetilde{\boldsymbol{D}}e^{\boldsymbol{\Phi}(t-s)}\right)_{ij}\mathrm{d}s = \left[-\frac{1}{\Lambda_{ii}+\Phi_{jj}}e^{\Lambda_{ii}(t-s)}\widetilde{D}_{ij}e^{\Phi_{jj}(t-s)}\right]_0^t \tag{38}$$

$$= \frac{\widetilde{D}_{ij}}{\Lambda_{ii}+\Phi_{jj}}e^{t(\Lambda_{ii}+\Phi_{jj})} - \frac{\widetilde{D}_{ij}}{\Lambda_{ii}+\Phi_{jj}} \tag{39}$$

Hence, we get the required result

$$\boldsymbol{H}(t) = e^{\boldsymbol{B}t}\boldsymbol{H}(0)e^{\boldsymbol{C}t} + \boldsymbol{P}\boldsymbol{F}(t)\boldsymbol{Q}^{-1}, \tag{40}$$

where

$$F_{ij}(t) = \frac{\widetilde{D}_{ij}}{\Lambda_{ii}+\Phi_{jj}}e^{t(\Lambda_{ii}+\Phi_{jj})} - \frac{\widetilde{D}_{ij}}{\Lambda_{ii}+\Phi_{jj}}. \tag{41}$$

□

Lemma 2 *Assuming that $\boldsymbol{A}$ and $\boldsymbol{W}$ have eigenvalue decompositions $\boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$ and $\boldsymbol{Q}\boldsymbol{\Phi}\boldsymbol{Q}^{-1}$ respectively,*

$$\int_0^1 \boldsymbol{A}^s\boldsymbol{E}\boldsymbol{W}^s\mathrm{d}s = \boldsymbol{P}\boldsymbol{F}\boldsymbol{Q}^{-1}, \tag{42}$$

where

$$F_{ij} = \frac{\Lambda_{ii}\widetilde{E}_{ij}\Phi_{jj} - \widetilde{E}_{ij}}{\ln\Lambda_{ii} + \ln\Phi_{jj}}, \tag{43}$$

with $\widetilde{\boldsymbol{E}} = \boldsymbol{P}^{-1}\boldsymbol{E}\boldsymbol{Q}$.

Proof: We start by using the eigenvalue decompositions of $\boldsymbol{A}$ and $\boldsymbol{W}$

$$\int_0^1 \boldsymbol{A}^s \boldsymbol{E} \boldsymbol{W}^s \mathrm{d}s = \boldsymbol{P} \int_0^1 \boldsymbol{\Lambda}^s \widetilde{\boldsymbol{E}} \boldsymbol{\Phi}^s \mathrm{d}s \boldsymbol{Q}^{-1}, \tag{44}$$

where $\widetilde{\boldsymbol{E}} = \boldsymbol{P}^{-1}\boldsymbol{E}\boldsymbol{Q}$. Now we can consider the integral element-wise to get the required result:

$$\int_0^1 \left(\boldsymbol{\Lambda}^s \widetilde{\boldsymbol{E}} \boldsymbol{\Phi}^s\right)_{ij} \mathrm{d}s = \int_0^1 \Lambda_{ii}^s \widetilde{E}_{ij} \Phi_{jj}^s \mathrm{d}s \tag{45}$$

$$= \left[\frac{\Lambda_{ii}^s \widetilde{E}_{ij} \Phi_{jj}^s}{\ln \Lambda_{ii} + \ln \Phi_{jj}}\right]_0^1 \tag{46}$$

$$= \frac{\Lambda_{ii}\widetilde{E}_{ij}\Phi_{jj} - \widetilde{E}_{ij}}{\ln \Lambda_{ii} + \ln \Phi_{jj}} \tag{47}$$

□

B.1. Derivation of the ODE

Proof of Proposition 3: For the discrete dynamic defined

$$\boldsymbol{H}_{n+1} = \boldsymbol{A}\boldsymbol{H}_n\boldsymbol{W} + \boldsymbol{H}_0, \tag{48}$$

$\boldsymbol{H}_n$ can be rewritten as follows:

$$\boldsymbol{H}_n = \sum_{k=0}^{n} \boldsymbol{A}^k \boldsymbol{E} \boldsymbol{W}^k \tag{49}$$

Recall that as in the case where $\boldsymbol{W} = \boldsymbol{I}$ we move to a continuous setting by interpreting the equation as a Riemann integral:

$$\boldsymbol{H}(t) = \int_0^{t+1} \boldsymbol{A}^s \boldsymbol{E} \boldsymbol{W}^s \mathrm{d}s. \tag{50}$$

To derive a corresponding ODE, we consider the derivative of $\boldsymbol{H}(t)$ with respect to t, yielding the ODE below:

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = \boldsymbol{A}^{t+1}\boldsymbol{E}\boldsymbol{W}^{t+1}. \tag{51}$$

To get the ODE in a nicer form, we consider the second-derivative of $\boldsymbol{H}$:

$$\frac{\mathrm{d}^2\boldsymbol{H}(t)}{\mathrm{d}t^2}(t) = \ln \boldsymbol{A}\boldsymbol{A}^{t+1}\boldsymbol{E}\boldsymbol{W}^{t+1} + \boldsymbol{A}^{t+1}\boldsymbol{E}\boldsymbol{W}^{t+1}\ln \boldsymbol{W} = \ln \boldsymbol{A}\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} + \frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t}\ln \boldsymbol{W}. \tag{52}$$

By integrating over t in both sides of the above equation, we can obtain:

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t}(t) = \ln \boldsymbol{A}\boldsymbol{H}(t) + \boldsymbol{H}(t)\ln \boldsymbol{W} + c. \tag{53}$$

Note that the initial value of the ODE is $\boldsymbol{H}(0)$ by Lemma 2, we know that

$$\left(\boldsymbol{P}^{-1}\boldsymbol{H}(0)\boldsymbol{Q}\right)_{ij} = \frac{\Lambda_{ii}\widetilde{E}_{ij}\Phi_{jj} - \widetilde{E}_{ij}}{\ln \Lambda_{ii} + \ln \Phi_{jj}}, \tag{54}$$

where $\widetilde{\boldsymbol{E}} = \boldsymbol{P}^{-1}\boldsymbol{E}\boldsymbol{Q}$.

By setting t to 0, we can obtain:

$$\left.\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t}\right|_{t=0} = \boldsymbol{A}\boldsymbol{E}\boldsymbol{W} \implies \boldsymbol{A}\boldsymbol{E}\boldsymbol{W} - \ln \boldsymbol{A}\boldsymbol{H}(0) - \boldsymbol{H}(0)\ln \boldsymbol{W} = c. \tag{55}$$

We can simplify c as follows:

$$c = \boldsymbol{A}\boldsymbol{E}\boldsymbol{W} - \ln \boldsymbol{A}\boldsymbol{H}(0) - \boldsymbol{H}(0) \ln \boldsymbol{W} \tag{56}$$

$$\left(\boldsymbol{P}^{-1} c \boldsymbol{Q}\right)_{ij} = \Lambda_{ii} \widetilde{E}_{ij} \Phi_{jj} - \ln \Lambda_{ii} \frac{\Lambda_{ii} \widetilde{E}_{ij} \Phi_{jj} - \widetilde{E}_{ij}}{\ln \Lambda_{ii} + \ln \Phi_{jj}} - \frac{\Lambda_{ii} \widetilde{E}_{ij} \Phi_{jj} - \widetilde{E}_{ij}}{\ln \boldsymbol{\Lambda}_{ii} + \ln \Phi_{jj}} \ln \Phi_{jj} \tag{57}$$

$$c = \boldsymbol{P} \widetilde{\boldsymbol{E}} \boldsymbol{Q}^{-1} \tag{58}$$

$$= \boldsymbol{E} \tag{59}$$

Thus the ODE can be reformulated as:

$$\frac{\mathrm{d}\boldsymbol{H}(t)}{\mathrm{d}t} = \ln \boldsymbol{A}\boldsymbol{H}(t) + \boldsymbol{H}(t) \ln \boldsymbol{W} + \boldsymbol{E}. \tag{60}$$

□

Proof of Proposition 4: Proposition 4 follows trivially from Lemma 1.

□

C. Hyperparameters & training details

Across Cora, Citeseer, and Pubmed we use a hidden dimension of 16, input dropout of 0.5 in the encoder, and weight decay of 5×10^{-4}. For NELL we use a hidden dimension of 64, dropout of 0.1 (in both encoder and decoder), and weight decay of 1×10^{-5}.γ determines the weight of self-loops in the graph.

Table 4: Cora hyperparameters, using the `rmsprop` optimiser.

	fixed split					random split				
Algorithm	lr	t_1	α	γ	β	lr	t_1	α	γ	β
CGNN discrete	3.45×10^{-3}	12.0	0.950	0.309	—	2.08×10^{-3}	20.0	0.950	0.517	—
CGNN	4.70×10^{-3}	12.1	0.918	0.555	—	1.47×10^{-3}	23.9	0.885	0.595	—
CGNN with weights	2.11×10^{-3}	14.3	0.950	0.947	0.5	—	—	—	—	0.5

Table 5: Citeseer hyperparameters, using the `rmsprop` optimiser.

	fixed split					random split				
Algorithm	lr	t_1	α	γ	β	lr	t_1	α	γ	β
CGNN discrete	3.28×10^{-3}	20.0	0.950	0.693	—	2.59×10^{-3}	20.0	0.950	0.548	—
CGNN	5.48×10^{-3}	19.1	0.869	0.758	—	2.98×10^{-3}	17.1	0.936	0.459	—
CGNN with weights	5.70×10^{-3}	23.4	0.950	0.775	0.5	—	—	—	—	0.5

Table 6: Pubmed hyperparameters, using the `adam` optimiser.

	fixed split					random split				
Algorithm	lr	t_1	α	γ	β	lr	t_1	α	γ	β
CGNN discrete	5.57×10^{-3}	20.0	0.950	0.626	—	4.08×10^{-3}	20.0	0.950	0.933	—
CGNN	5.40×10^{-3}	16.2	0.960	0.644	—	5.51×10^{-3}	22.0	0.947	0.752	—
CGNN with weights	5.14×10^{-3}	20.3	0.950	0.668	0.5	—	—	—	—	0.5

In practice, we used the augmentation proposed in Dupont et al. (2019) to stabilise training, however, it had little no effect on the final performance. Writing down the augmentation mathematically yields the following matrix differential equation for the ODE

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} \boldsymbol{H}(t) & \boldsymbol{O}(t) \end{bmatrix} = (\boldsymbol{A} - \boldsymbol{I}) \begin{bmatrix} \boldsymbol{H}(t) & \boldsymbol{O}(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{H}(t) & \boldsymbol{O}(t) \end{bmatrix} (\boldsymbol{W} - \boldsymbol{I}) + \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \end{bmatrix}, \tag{61}$$

where $\boldsymbol{O}(0) = 0$. All the augmentation does is add some latent dimensions to allow the trajectory of the ODE (which cannot cross itself) a potentially simpler path.

Table 7: NELL hyperparameters, using the `adam` optimiser.

	fixed split					random split				
Algorithm	lr	t_1	α	γ	β	lr	t_1	α	γ	β
CGNN discrete	0.01	20.0	0.950	0.941	—	0.01	20.0	0.950	0.941	—
CGNN	0.01	20.0	0.950	0.941	—	0.01	20.0	0.950	0.941	—
CGNN with weights	0.01	20.0	0.950	0.956	0.5	—	—	—	—	0.5

D. Running time

In Table 8 we compare the running times between our algorithms and GCN as measured on the Cora dataset using the hyperparameters in Table 4. For GCN we used the hyperparameters quoted in (Kipf and Welling, 2016). For all models we used 400 epochs and measured the total running time. All results were collected on a single machine with the following specs: Intel Core i7-6700HQ CPU at 2.60GHz with 16GB of RAM with NVIDIA GeForce GTX 960M with 2GB of dedicated GPU memory.

Table 8: Running time per epoch on the Cora dataset.

Architecture	CGNN	CGNN with weights	GCN
Time per epoch (seconds)	2.18	2.65	0.039

E. Comparison to GCN with skip connections

We compare with GCN with residual links and with 2, 4, 8, and 16 layers respectively. The results on Cora, Citeseer, and PubMed in both the fixed data split setting and random split setting are given in Table 9.

	Cora		Citeseer		PubMed	
Number of Layers	Random splits	Fixed splits	Random splits	Fixed splits	Random splits	Fixed splits
2	77.2 ± 1.7	77.4 ± 1.9	68.0 ± 2.5	64.7 ± 1.5	80.7 ± 1.9	79.6 ± 0.6
4	76.5 ± 1.6	77.3 ± 2.0	65.6 ± 2.3	62.6 ± 2.0	80.5 ± 1.4	78.6 ± 0.6
8	76.8 ± 1.9	75.7 ± 1.3	64.2 ± 2.1	62.3 ± 1.3	80.0 ± 1.4	78.5 ± 0.9
16	77.6 ± 2.1	76.8 ± 1.9	64.4 ± 2.0	62.2 ± 1.9	80.4 ± 0.3	79.9 ± 1.3

Table 9: Performance of GCN with residual links and increasing depth on Cora, Citeseer, and PubMed.