Improving Social Network Embedding via New Second-Order Continuous Graph Neural Networks

Yanfu Zhang
Electrical and Computer Engineering
University of Pittsburgh
Pittsburgh, PA, USA
yaz91@pitt.edu

Shangqian Gao
Electrical and Computer Engineering
University of Pittsburgh
Pittsburgh, PA, USA
shg84@pitt.edu

Jian Pei
School of Computer Science
Simon Fraser University
Vancouver, BC, Canada
jpei@cs.sfu.ca

Heng Huang*
Electrical and Computer Engineering
University of Pittsburgh
Pittsburgh, PA, USA
henghuanghh@gmail.com

ABSTRACT

Graph neural networks (GNN) are powerful tools in many web research problems. However, existing GNNs are not fully suitable for many real-world web applications. For example, over-smoothing may affect personalized recommendations and the lack of an explanation for the GNN prediction hind the understanding of many business scenarios. To address these problems, in this paper, we propose a new second-order continuous GNN which naturally avoids over-smoothing and enjoys better interpretability. There is some research interest in continuous graph neural networks inspired by the recent success of neural ordinary differential equations (ODEs). However, there are some remaining problems *w.r.t.* the prevailing first-order continuous GNN frameworks. Firstly, augmenting node features is an essential, however heuristic step for the numerical stability of current frameworks; secondly, first-order methods characterize a diffusion process, in which the over-smoothing effect *w.r.t.* node representations are intrinsic; and thirdly, there are some difficulties to integrate the topology of graphs into the ODEs. Therefore, we propose a framework employing second-order graph neural networks, which usually learn a less stiff transformation than the first-order counterpart. Our method can also be viewed as a coupled first-order model, which is easy to implement. We propose a semi-model-agnostic method based on our model to enhance the prediction explanation using high-order information. We construct an analog between continuous GNNs and some famous partial differential equations and discuss some properties of the first and second-order models. Extensive experiments demonstrate the effectiveness of our proposed method, and the results outperform related baselines.

*This work was partially supported by NSF IIS 1845666, 1852606, 1838627, 1837956, 1956002, IIA 2040588.

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539415

CCS CONCEPTS

• **Computing methodologies** → **Learning latent representations**; • **Networks** → **Network algorithms**.

KEYWORDS

graph neural networks, ordinary differential equation, prediction explanation

ACM Reference Format:
Yanfu Zhang, Shangqian Gao, Jian Pei, and Heng Huang. 2022. 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 (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3534678.3539415

1 INTRODUCTION

Recently, there is growing interest in designing graph neural networks (GNN) to solve a variety of web research problems, such as social network analysis [20], recommendation systems [34], fraud detection [45], natural languages [16], brain connectome analysis [44], *etc.*. Graph neural networks are primitively motivated by graph convolution [4, 13] based on spectral graph theory. Consecutive researches [7, 9, 11, 36, 38, 40, 43] show GNNs are powerful in learning graph embeddings, and the simplicity and effectiveness of GNN can be boosted via generalizing the graph convolution operation to aggregate the information from the nodes of interest and their neighbors. Some representative variants include GraphSage [11], Message Passing Neural Network [38], Graph Attention Network [31] and Graph Isomorphism Network [36].

There is some discrepancy between the practical requirements for business scenarios and the existing research. For example, one usually wants to make a personalized recommendation targeting on individual users. Deeper networks usually are beneficial for learning more complex data distributions. But node representations may be over-smoothed [13, 22, 36] due to the low-frequency filtering property of many GNNs. As a result, the user difference is blurred, which becomes a hindrance if one wants to make personalized recommendation. Another example is that it is difficult to build business decision based on the black-box predictions of

GNNs [17, 39, 41]. In social networks, finding influential neighbors can significantly improve the understanding of the user behavior. As such, there is some interest in explaining the GNN predictions based on a small subgraph.

To address these problems, in this paper, we propose a novel second-order continuous graph neural network. There were some attempts to address over-smoothing problem [15, 36]. Inspired by an emerging research topic connecting the residual network and ordinary differential equations [2], continuous-depth GNNs was proposed based on the dynamics of hidden layers and show advantages than the discretized counterparts in better memory efficiency and superior performance. The first-order frameworks adopted by existing graph ODEs characterize the diffusion process which cannot fully address the over-smoothing problem because the heat kernel is usually the Gaussian. Besides, many methods [25, 35] use augmented features as an ad-hoc step for numerical stability. The selection of feature augmentation is thus discretionary and lacks interpretability. Our approach learns the second-order dynamics of GNNs and considers the effect of a force term and a damping term to avoid over-smoothing, which can be viewed as discretizing the variants of wave equations. Our model can also be viewed as a coupled first-order equation with interpretable augmented feature velocity. Previous GNN explanation methods are usually model-agnostic, which omits the second-order information in our model. We design a new semi-model-agnostic method that explicitly considers the high-order information but leaves the GNN structure untouched to enhance the prediction explanation.

Our contributions can be summarized as follows:

- We propose a second-order continuous GNN based on the message passing neural network framework. Our second-order model can be expressed as a coupled first-order equation via augmenting the node features with their gradients, and the implementation is feasible using the popular first-order framework.
- We construct an analog between continuous GNNs and partial differential equations because graph Laplacian can be viewed as discretized Laplacian operator on manifolds.
- We introduce a method to explain the GNN prediction via leveraging the high-order information with an independent edge embedding network. The explanation is agnostic to the specific GNN structure.
- We conduct extensive experiments on several graphs datasets of homophily, which is an important property of social networks [18, 23, 46]. The results show that our method outperforms related baselines in various downstream tasks, and the second-order information can boost the model explanation.

2 RELATED WORK

Neural ODEs: Residual network [12] is an important method to push neural networks to extreme depths. It is observed that the skip links in ResNet can be seen as an Euler discretization of a continuous transformation. Based on this connection, the research on neural ODEs evokes emerging interest. A popular framework, Neural ODEs [2], considers the continuous-depth neural networks via taking the limit of this discretization. The optimization *w.r.t.* the ODE solvers adopt the adjoint sensitivity method [26], which treats the ODE solve as a black box and solve a second augmented ODE backward. The continuous depths make neural ODEs suitable for modeling the dynamics of complex systems, particularly those that cannot be described analytically. Meanwhile, it is difficult for standard neural networks to learn symmetries and conservation laws while neural ODEs can address this issue. For example, Hamiltonian Neural Networks [10] and Lagrangian Neural Networks [3] can produce energy-conserving models for various tasks.

Augmented Neural ODEs In Neural ODEs, the hidden states evolves continuously according to a differential equation, whose velocity is described by a neural network. Once the dynamics are known, the gradients of some objective with regard to the model parameters can be computed through adjoint sensitivity method. It is shown that the feature mapping learned by Neural ODEs is a homeomorphism [6], which implies the existence of functions Neural ODEs cannot represent. Augmented Neural ODEs [6] is proposed to address these limitations via introducing additional features. A related variant of augmented neural ODEs is the Second Order Neural ODEs [21]. SONODE explicitly considers the velocities as augmented dimensions, and proposes to learn the acceleration. The inductive bias introduced in SONODE is beneficial in learning many dynamical systems governed by second order laws, such as Newton's equations of motion and oscillators. SONODE can be solved either using the adjoint state directly or using the coupled first order ODE. Of note, augmented neural ODEs can also learn high-order dynamics. However, the high order dynamics learned by ANODE are characterized by abstract functions and the augmented features typically are entangled, which leads to difficulties in interpretability.

Graph Representation Learning: Graph representation learning is featured by mining the topological structures of graphs and encoding nodes with low-dimensional embeddings. Representative works collect local patterns and learn mappings from graphs to vectors, including Word2Vec [19], DeepWalk [24], and LINE [30]. To simultaneously exploit the structural knowledge and the enriched side information in attribute graphs, increasing interests are paid to GNNs. GNN [13] is based on graph convolution, an extension of regular convolutions that can process structural data. Via designing a more efficient aggregation mechanism, GraphSage [11, 38, 40] and Message Passing Neural Network (MPNN) [9] broaden the application of GNNs to the analysis for large-scale graphs. Graph Attention Network (GAN) [31] introduces an attention mechanism into general network analysis, and self-attention is proposed for graph pooling [14] for graph classification tasks. Based on the relationship between GNNs and graph isomorphism problem, Graph Isomorphism Network (GIN) [36] and Deep Graph Convolutional Neural Network(DGCNN) [42] are designed. On the other hand, there is some attempt to learn interpretable graph representation [39], particularly for some applications, e.g., brain networks [28, 29].

Continuous-depth Graph Neural Networks The idea behind continuous-depth neural networks is also employed in graph neural networks to handle structural data. Graph neural ordinary Differential Equations (GDEs) [25] propose a continuum of GNN layers to characterize the input-output relationship. In detail, the static models and Spatio-temporal models are developed to handle different tasks. Alternatively, Continuous Graph Neural Networks

(CGNN) [35] proposes a propagation scheme inspired by diffusion-based methods. Specifically, the representations learned by CGNN have entangled *w.r.t.* the graph structure and the node features while the terminal time goes to infinite time. The continuous-depth formulation of GNNs also inspires some related methods. For example, the numerical gap between the discrete and continuous graph diffusion process can affect the model performance [32], and Simple Graph Convolution [33] can be boosted by decoupling the terminal time and the finite difference steps. Continuous Graph Flow (CGF) [5], as a graph generative model, generalizes the messaging passing mechanism to continuous time.

3 PROPOSED METHOD

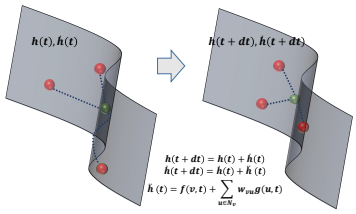


Figure 1: Learn the second order GNN.

In this section, we first review the Messaging Passing Neural Network (MPNN) and the first-order continuous GNN, then describe our second-order continuous GNNs. Figure. 1 illustrates the second-order dynamics of the node embeddings on a manifold, which will be detailed in the following. Via reformulating our model as coupled first-order neural ODEs, we give a simple implementation. We propose a method to explain the GNN prediction jointly using the first and the second order information. At last, motivated by the connection between graph Laplacian and the Laplace-Beltrami operator, we discuss some properties of continuous GNNs from the view of neural partial differential equations instead of ordinary differential equations.

3.1 Second-Order Continuous GNNs

3.1.1 Revisiting Messaging Passing Neural Networks and First-Order Continuous GNNs. Existing continuous GNNs are based on the continuous counterpart of residual structures in GNNs. In this part, we first describe the Messaging Passing Neural Network (MPNN), then recap the first-order continuous GNN under this framework.

We denote a graph $\mathcal{G}$ by its node set $\mathcal{V}$ and its edge set $\mathcal{E}$. For a single node $v \in \mathcal{V}$, we use $\mathcal{N}(v) \coloneqq \{u \in \mathcal{V} : (v,u) \in \mathcal{E} \vee (u,v) \in \mathcal{E}\}$ to denote its neighbors. A MPNN layer performs a spatial-based convolution on v, and the representation of v at layer $l+1$ is defined on the representations of v and its neighbor $\mathcal{N}(v)$ at layer l,

$$\boldsymbol{h}^{(v)}(l+1) = \boldsymbol{u}\left[\boldsymbol{h}^{(v)}(l), \sum_{u\in\mathcal{N}(v)} \boldsymbol{m}\left(\boldsymbol{h}^{(v)}(l), \boldsymbol{h}^{(u)}(l)\right)\right]. \quad (1)$$

Specifically, we have $\boldsymbol{h}^{(v)}(0) = \boldsymbol{x}_v$, which is the input to the MPNN. $\boldsymbol{x}_v$ can be node features or some representations learned by an encoder. $\boldsymbol{u}$ and $\boldsymbol{m}$ are functions with trainable parameters.

A general first-order continuous GNN can be defined by first setting $\boldsymbol{u}(\boldsymbol{x},\boldsymbol{y}) \coloneqq \boldsymbol{x} + \boldsymbol{g}(\boldsymbol{x},\boldsymbol{y})$ [25], where $\boldsymbol{g}\left(\boldsymbol{h}^{(v)}(l), \boldsymbol{h}^{(\mathcal{N}(v))}(l)\right) = \boldsymbol{g}\left(\sum_{u\in\mathcal{N}(v)} \boldsymbol{m}\left(\boldsymbol{h}^{(v)}(l), \boldsymbol{h}^{(u)}(l)\right)\right)$ is some function, (1) can be written as,

$$\boldsymbol{h}^{(v)}(l+1) - \boldsymbol{h}^{(v)}(l) = \boldsymbol{g}\left(\boldsymbol{h}^{(v)}(l), \boldsymbol{h}^{(\mathcal{N}(v))}(l)\right). \quad (2)$$

By interpreting the layer l as a discretezation step in time t, the continuous-depth counterpart of MPNN is defined on the continuity equation of the above difference equation,

$$\dot{\boldsymbol{h}}^{(v)}(t) = \boldsymbol{f}^{(v)}_{MPNN}(\boldsymbol{H},\boldsymbol{\theta}) \coloneqq \boldsymbol{g}\left(\boldsymbol{h}^{(v)}(t), \boldsymbol{h}^{(\mathcal{N}(v))}(t)\right). \quad (3)$$

Here $\boldsymbol{H}$ refers to all node feartures, and $\boldsymbol{\theta}$ refers to the model parameters, which defines the differential function. Formally, given input node features, the continuous MPNNs compute the node representations by solving the Cauchy problem,

$$\dot{\boldsymbol{h}}^{(v)}(t) = \boldsymbol{g}\left(\boldsymbol{h}^{(v)}(t), \boldsymbol{h}^{(\mathcal{N}(v))}(t)\right), \quad \boldsymbol{h}^{(v)}(0) = \boldsymbol{x}^{(v)}_0. \quad (4)$$

Similar to neural ODEs, the forward pass is to solve the problem numerically using ODE solvers, and the optimization of parameters uses the adjoint sensitivity method.

3.1.2 Second-Order Continuous Graph Neural Networks. In this paper, we consider a second-order dynamics for GCN. Specifically, we alter the definition of $\boldsymbol{u}$ and the second-order model is defined as,

$$\ddot{\boldsymbol{h}}^{(v)}(t) = \boldsymbol{f}^{(v)}_{MPNN}(\boldsymbol{H},\boldsymbol{\theta}) \coloneqq \boldsymbol{g}\left(\boldsymbol{h}^{(v)}(t), \boldsymbol{h}^{(\mathcal{N}(v))}(t)\right). \quad (5)$$

$\boldsymbol{g}(\cdot)$ considers the interaction the neighbors have on the center node v. More generally, we consider two additional terms for (5), a term for the velocity decay and a term only related to the center node. Finally, our second-order continuous GNNs solves the following Cauchy problem,

$$\ddot{\boldsymbol{h}}^{(v)}(t) + \alpha\dot{\boldsymbol{h}}^{(v)}(t) = \boldsymbol{f}(\boldsymbol{h}^{(v)}(t)) + \boldsymbol{g}\left(\boldsymbol{h}^{(v)}(t), \boldsymbol{h}^{(\mathcal{N}(v))}(t)\right), \quad (6)$$

$$\boldsymbol{h}^{(v)}(0) = \boldsymbol{x}^{(v)}_0, \quad \dot{\boldsymbol{h}}^{(v)}(0) = 0. \quad (7)$$

Here $\alpha > 0$ is a small positive number, $\boldsymbol{f}(\boldsymbol{h}^{(v)}(t))$ is the force term. The initial state for $\dot{\boldsymbol{h}}^{(v)}(0)$ is set to 0 for simplicity. The differential and the node representations at time t is then given by $\dot{\boldsymbol{h}}^{(v)}(t) = \int_0^t \ddot{\boldsymbol{h}}^{(v)}(t)dt$ and $\boldsymbol{h}^{(v)}(t) = \boldsymbol{x}^{(v)}_0 + \int_0^t \dot{\boldsymbol{h}}^{(v)}(t)dt$. $\alpha\dot{\boldsymbol{h}}^{(v)}(t)$ is usually called a damping term.

Our model has a straightforward physical explanation. Let every node in a graph represent a ball. The features describe the spatial position of the balls. The edges denote the existence of the interaction between balls, for example, we can think that two linked nodes are two balls connected by a spring. The damping term can be viewed as friction. The force term is associate with the medium a ball lies in, which is not dependent on the other balls. For example, the balls are placed on a curved surface and are under the influence of a component of weight. An one-dimensional example is illustrated in Fig. 2.

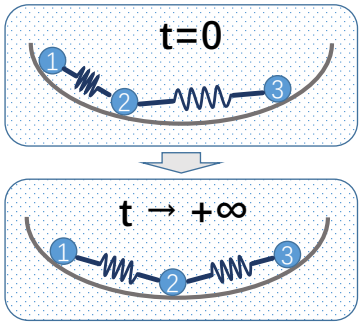


Figure 2: A ball-spring system under the second order dynamics (elastic force, gravity, and friction).

3.2 Implementation of Second-Order Continuous GNN

The forward pass of our model can be accomplished via off-the-shelf ODE solvers. Similar to the first-order scenario, the backpropagation can be performed via the adjoint sensitivity method [26], which treats ODE solvers as black-boxes and has a low memory cost. The adjoint state of (8) can be computed using Lagrangian methods.

Proposition 1. *The adjoint state of* (8) *follows the second order ODE* $\ddot{r} = r^\intercal \frac{\partial \ddot{\boldsymbol{h}}}{\partial \boldsymbol{h}(v)} - \alpha \dot{r}^\intercal$. *The update of the model parameters is the integral* $\frac{dL}{d\theta} = -\int_t^0 r^\intercal \frac{\partial x}{\partial \theta} ds$.

Our model is a special case of Proposition 3.1 in Norcliffe et al. [21], and the above result is obtained by substituting the damping $\ddot{\boldsymbol{h}}$. The second-order differential equations in our approach can also be viewed as first-order coupled differential equations [21]. This relation gives an alternative model of our second-order CGNN as first-order augmented Neural ODEs. More specifically, we augment the node features to include their differentials $z^{(v)} = \left[\boldsymbol{h}^{(v)}(t), \dot{\boldsymbol{h}}^{(v)}(t)\right]$, then our second-order continuous GNN can then be represented as a first-order Cauchy problem,

$$z^{(v)} = \left[\boldsymbol{h}^{(v)}(t), \dot{\boldsymbol{h}}^{(v)}(t)\right], \dot{z}^{(v)} = \left[\dot{\boldsymbol{h}}^{(v)}(t), \ddot{\boldsymbol{h}}^{(v)}(t)\right], z_0^{(v)} = \left[\boldsymbol{x}_0^{(v)}, 0\right]. \tag{8}$$

As pointed out in Norcliffe et al. [21], the adjoint state of the second-order model is equivalent to the adjoint state of the coupled first-order model. The disentangled representation in second-order models may involve more computation than the entangled augmented NODE. Due to this reason, we use the first-order optimization for simplicity in our implementation. In detail, we maintain $z^{(v)}$ in our continuous GNNs, and in the forward pass $\dot{z}^{(v)}$ is obtained by concatenating $\dot{\boldsymbol{h}}^{(v)}$ from the stale z and the newly computed $\ddot{\boldsymbol{h}}^{(v)}$. No modification is required for the backward propagation.

3.3 Enhanced Explanation for Graph Neural Networks Using Second-Order Information

In the representation learning for graph data, the high-order part of z is discarded. However, it can provide some information in explaining the prediction of GNNs, which indicates that our second-order continuous GNN enjoys better interpretability compared to first-order methods. More specifically, in this section we propose a new method for example-level explanations via employing the first-order and the second-order information jointly.

3.3.1 Problem Formulation. We employ the definition of example-level explanation from [39]. Assume we have a trained GNN $\hat{y} = f(\mathcal{G}_c, x_c)$. Here $\mathcal{G}_c(v)$ is a computation graph spanned from $v \in \mathcal{G}$, which is the induced subgraph on the full $\mathcal{G}$ involved in computing the prediction for v. The associated adjacency matrix is $A_c(v)$. $x_c(v)$ is the associated feature set $\{x_j | v_j \in \mathcal{G}_c(v)\}$. Using the computation graph instead of the full graph can greatly reduce the computational burden. Given a node v, its explanation is $(\mathcal{G}_s, x_s(v))$. Here $\mathcal{G}_s$ is a small subgraph on the computation graph $\mathcal{G}_c(v)$ and $x_s(v)$ is a small subset of node features, masked out by $\{x_j | v_j \in \mathcal{G}_s(v)\}$. $(\mathcal{G}_s, x_s(v))$ are important the prediction $\hat{y}_v$. The importance is evaluated using mutual information I,

$$\max_{\mathcal{G}_s} I(y, (\mathcal{G}_s, x_s)) = H(Y) - H(Y|\mathcal{G} = \mathcal{G}_s, x = x_S), \tag{9}$$

here H is the entropy. Typical explanation-generation methods are model-agnostic by fixing the GNN $f(\mathcal{G}_c, x_c)$, therefore, $H(Y)$ is constant. The second term can be bounded,

$$H(Y|\mathcal{G} = \mathcal{G}_s, x = x_S) = -\mathbb{E}_{Y|\mathcal{G}_S, x_S}[\log P_f(Y|\mathcal{G} = \mathcal{G}_S, X = X_S)], \tag{10}$$

We use $A_s(v)$ to represent the adjacency matrix of $\mathcal{G}_S$. Of note, model-agnostic methods fail to make use the second-order information in our model. To address this problem, we provide a semi-model-aware explanation-generation method in the following.

3.3.2 Semi-model-aware GNN Explainer. The explanation adjacency matrix can be expressed by $A_s(v) = A_c(v) \odot M$, where M is a mask matrix, and its entries are binary. Using this expression, directly optimizing (10) boils down to an integer problem with respect to M, and the size of feasible set is $2^{|\mathcal{E}|}$. To make the problem tractable, we resort to a relaxation of M using mean-field variational approximation, and using Monte Carlo method to compute the explanation. As illustrated in Fig. 3, our method sample subgraphs by two steps. First, we decompose the distribution of $\mathcal{G}$ into a multivariate Bernoulli distribution as $P(\mathcal{G}) = \Pi_{(i,j)\in\mathcal{E}} P(e_{ij})$, here $e_{ij} \in [0, 1]$ is the relaxed entry of M, denoting the probability that edge (v_i, v_j) is selected. We compute e_{ij} using an edge embedding network based on the node embeddings. Second, the subgraph is sampled from $P(\mathcal{G})$ using a reparameterization trick. Since the sampling is not differentiable, we use straight through estimator (STE) to enable the back-propagation. We feed the sampled subgraphs into the GNN with weights frozen, and update the edge embedding network with respect to (10).

In detail, we compute the embedding for edge (v_i, v_j) by a deep neural network. Since our GNN works on undirected graphs, we anonymize the edge direction by defining,

$$\boldsymbol{\mu}_{ij}, \Sigma_{ij} = \frac{1}{2}\left(\phi(\boldsymbol{z}_{v_i}, \boldsymbol{z}_{v_j}) + \phi(\boldsymbol{z}_{v_j}, \boldsymbol{z}_{v_i})\right). \tag{11}$$

Similar to variaitional autoencoder, we learn a mean and a variance for the edge embeddings. In the sampling step, we use the reparameterization trick from the invertible Gaussian family [27],

$$e_{ij} = g(\boldsymbol{\mu}_{ij} + \epsilon * \Sigma_{ij}, \tau), \tag{12}$$

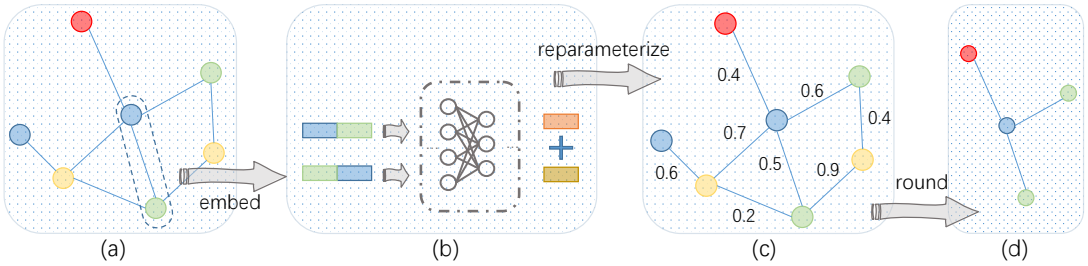


Figure 3: The pipeline for subgraph sampling. We compute the edge embeddings using a deep neural network. We use the invertible Gaussian reparameterization trick to sample the subgraph, which allows the straight-through gradients estimation for the back-propagation. We update the embedding network using Monte Carlo method to optimize the mutual information between the explanation and the computation graph.

here $\epsilon \sim \mathcal{N}(0, \mathbf{1})$ is a Gaussian noise of the same size $|\mathcal{E}_c|$, τ is a temperature parameter, $g(\cdot, \tau)$ is an invertible smooth function. Specifically, we let,

$$g(y, \tau) = softmax_{++}(w, \tau) = \frac{\exp(y_k/\tau)}{\sum^{K-1} \exp(y_k/\tau) + \delta}, \tag{13}$$

here $\delta > 0$ ensures the invertibility. $\lim_{\tau \leftarrow 0} softmax_{++}(y, \tau)$ is one-hot [27], which allows the straight through gradient estimation. Compared to Gumbel-Softmax, the invertible Gaussian reparameterization is more flexible.

To pursue a compact explanation, we also consider a regularization with respect to the size of the sampled subgraphs. To sum up, we update $\phi(\cdot)$ with respect to the sampled subgraphs using the following objectives,

$$\min_{\phi} \mathcal{T} = -\mathbb{E}_{Y|\mathcal{G}_S, x_S}\left[\log P_f\left(Y|\mathcal{G} = \mathcal{G}_S, X = X_S\right)\right] + \lambda\left(ReLU\left(\sum e_{ij} - K_m\right)\right)^2. \tag{14}$$

here K_m is a predetermined positive integer limiting the node numbers in subgraphs.

3.4 First-Order v.s. Second-Order

Existing works usually let

$$\boldsymbol{g}\left(\sum_{u \in \mathcal{N}(v)} \boldsymbol{m}\left(\boldsymbol{h}^{(v)}(t), \boldsymbol{h}^{(u)}(t)\right)\right) = \sigma\left(\sum_{u \in \mathcal{N}(v)} \boldsymbol{w}_u(\boldsymbol{h}^{(v)}(t) - \boldsymbol{h}^{(u)}(t))\right), \tag{15}$$

which can be viewed as applying some generalized graph Laplacian to the features. Graph Laplacian matrix associated to a point cloud converges to the Laplace-Beltrami operator on the underlying data manifold [1]. As such, to better capture the data distribution, we should consider both temporal and spatial continuity in GNNs, which corresponds to the depth and the data distributions respectively. We insert the continuous Laplacian operator to (3) and abuse v to represent the hidden state, the first order continuous GNNs can be written as a standard heat equation,

$$\dot{\boldsymbol{h}}(v, t) = \Delta \boldsymbol{h}(v, t), \quad \boldsymbol{h}(v, 0) = \boldsymbol{h}_0. \tag{16}$$

Similarly, the second-order continuous GNNs can be written as an inhomogeneous wave equations,

$$\ddot{\boldsymbol{h}}(v, t) + \alpha \dot{\boldsymbol{h}}(v, t) = \Delta \boldsymbol{h}(v, t) + \boldsymbol{f}(v, t), \tag{17}$$

$$\boldsymbol{h}(v, 0) = \boldsymbol{h}_0, \quad \frac{\partial}{\partial t}\boldsymbol{h}(v, 0) = \mathbf{0}. \tag{18}$$

Compared to standard Neural ODEs, the properties of continuous GNNs can be better characterized by the associated partial differential equations. For simplicity, we omnit the weights in GNNS. In the rest of the section, we will use these simplified models to tentatively show that the over-smoothing effect is intrinsic in first-order continuous GNNs, and discuss some pros and cons to use the second-order methods instead of the first-order method.

Over-Smoothing Effect as Intrinsic Property of First-Order Continuous Graph Neural Networks: First-order continuous graph neural networks are intimately related to the heat equation, which characterizes the diffusion of heat on a manifold. More specifically, existing works define a Cauchy problem for the homogeneous heat equation. However, the steady-state solution of the heat equation implies that the over-smoothing is intrinsic for first-order continuous graph neural networks. To see this, we have the following proposition:

Proposition 2. *Let $\boldsymbol{h}_0 \in L^1(\mathbb{R}^n)$, namely $\int_{\mathbb{R}^n} |\boldsymbol{h}_0| dx < \infty$. For any $x \in \mathbb{R}^k$, we have $\lim_{t \to +\infty} |\boldsymbol{h}(x, t)| = 0$.*

Proof. This is a direct result by applying the fundamental solution of heat equation [8]. The fundamental solution is,

$$\boldsymbol{h}(x, t) = \int_{\mathbb{R}^n} \Phi(x - y, t)\boldsymbol{h}_0(y) dy, \tag{19}$$

$$\Phi(x, t) := \frac{1}{(4\pi t)^{n/2}} \exp(-\frac{|x|^2}{4t}), \tag{20}$$

The heat kernel $\Phi(x, t)$ is a Gaussian, and we have

$$|\boldsymbol{h}(x, t)| \leq \frac{1}{(4\pi t)^{n/2}} \int_{\mathbb{R}^n} \exp(-\frac{|x - y|^2}{4t})\boldsymbol{h}_0(y) dy \leq \frac{1}{(4\pi t)^{n/2}} \int_{\mathbb{R}^n} |\boldsymbol{h}_0(y)| dy. \tag{21}$$

Then we have $\lim_{t \to +\infty} |\boldsymbol{h}(x, t)| = \lim_{t \to +\infty} \frac{1}{(4\pi t)^{n/2}} \int_{\mathbb{R}^n} |\boldsymbol{h}_0(y)| dy = 0$. □

Using the above result we immediate have $\lim_{t\to+\infty}|\boldsymbol{h}(v_1,t) - \boldsymbol{h}(v_2,t)| = 0$ for any two nodes v_1 and v_2. In the context of continuous graph neural networks, it means that node features are eventually blurred when the terminal time goes to infinity. More specifically, the steady-state solution have exponential decay rate which is not related to the specific form of the original feature distribution $\boldsymbol{h}_0$. This result also implies that the learned data representations are more dispersed to compensate for the exponential decay *w.r.t.* the terminal time if the model is trained using a longer integral time.

The solution of the second-order CGNN: The solution of the second-order CGNN is dependent on the initial values. For simplicity, we consider a forced wave equation via setting $\alpha = 0$ in (17) and assume the feature dimension n is even. First we obtain a homogeneous problem by letting $\boldsymbol{f}(v,t) = 0$. Using the spherical means Evans [8], the solution is,

$$\boldsymbol{h}(x,t) = \frac{1}{n!!}\left[\left(\frac{\partial}{\partial t}\right)\left(\frac{1}{t}\frac{\partial}{\partial t}\right)^{\frac{n-2}{2}}\left(\frac{1}{(n+1)\alpha(n+1)}\cdot \int_{B(x,t)}\frac{\boldsymbol{h}_0(y)}{(t^2-|y-x|^2)^{1/2}}dy\right)\right], \tag{22}$$

Here $B(x,t)$ is a ball, $\alpha(n+1)$ is the volume of n-dimension unit ball. Next we insert $\boldsymbol{f}(v,t)$ and let $\boldsymbol{h}_0(v) = 0$. We have the following nonhomogeneous problem,

$$\frac{\partial^2}{\partial t^2}\boldsymbol{h}(v,t) = \Delta\boldsymbol{h}(v,t) + \boldsymbol{f}(v,t), \boldsymbol{h}(v,0) = 0, \frac{\partial}{\partial t}\boldsymbol{h}(v,0) = 0. \tag{23}$$

Define $\boldsymbol{h}(v,t;s)$ to be the solution of a homogeneous problem with starting time s and initial value $\boldsymbol{h}(v,s) = \boldsymbol{f}(v,t)$. Duhamel's principle asserts that $\boldsymbol{h}_n(v,t) := \int_0^t \boldsymbol{h}(v,t;s)ds$ is the solution to (22). The full solution to the forced wave equation is then the sum of (23) and $\boldsymbol{h}(v,s)$. The outputs of the second-order model are always related to the initial values since $\boldsymbol{h}_0(y)$ is contained in the formulae, which avoids the zero value problem in the first-order case. The solution takes a different form when the feature dimension is odd, but the dependency on initial states are still valid.

More Discussions: In this part, we discuss the pros and cons of using second-order continuous GNN instead of the first-order continuous GNN.

Second-order continuous GNN usually learn a smoother transformation than the first-order, which may alleviate over-fitting. The first-order and the second-order continuous GNN both attempt to compute a depth-varying vector field via solving Cauchy problems. However, a severe issue with the first-order Cauchy problem is the stiffness of the learned vector fields. More specifically, the feature mapping g is a homeomorphism, so the features of Neural ODEs preserve the topology of the input space [6]. Instead, the second-order neural ODEs are not limited to homeomorphic transformations [21].

The first-order augmented ODE implementation of our approach avoids the ad-hoc feature-augmentation in existing first-order graph ODEs and allows better interpretability. To address this stiffness of learned representations, Augmented Neural ODEs (ANODE) [6] proposes mapping the node features to higher dimensions, which is a widely used trick in Graph-based Neural ODEs. Although augmented neural ODEs can learn high-order dynamics [21], the learned augmented features are difficult to be interpreted, because the high-order behavior is entangled in these features. Our model explicitly defines the augmented features as the gradients of the node features, which have fixed size and clear interpretability.

Sometimes, the second-order models may not be advantageous compared to the augmented first-order models. For example, our definition of the second-order model may not be the minimal augmentation [21]. On the other hand, sometimes, there exist eminent higher-order dynamics behind the data transformation, which can be learned by augmented first-order models. In this case, the second-order models may lose their interpretability and performance superiority.

In general, although it is difficult to claim there exists a performance gap between the optimal potential first-order and the second-order continuous GNN, second-order model has better interpretability, has a simpler form, and usually learns a less stiff data distribution. As such, second-order continuous GNN is more likely to give better performance in practice.

4 EXPERIMENTAL RESULTS

This section evaluates the performance of our approach on the semi-supervised node and graph classification tasks.

4.1 Datasets and Experiment Settings

Semi-supervised Node Classification For this task, three benchmark datasets are used, including Cora, Citeseer, and Pubmed. We use the standard data splits for the benchmark datasets, where 20 nodes of each class are used for training and another 500 labeled nodes are used for validation. We also include the experiments with random splits. The statistics of datasets are described in Table 1.

Table 1: Statistics of datasets for node classification.

Dataset	# Nodes	# Edges	# Features	# Classes	# Label Rate
Cora	2708	5429	1433	7	0.036
Citeseer	3327	4732	3703	6	0.052
Pubmed	19,717	44,338	500	3	0.003

Graph Classification For this task, we predict the labels for graphs. We consider five datasets [37], including IMDB-Binary (IMDB-B), IMDB-Multi (IMDB-M), COLLAB, Reddit Binary (RDT-B), and Reddit-Multi5k (RDT-M). The statistics of datasets are described in Table 2.

Table 2: Statistics of datasets for graph classification.

Dataset	IMDB-B	IMDB-M	COLLAB	RDT-B	RDT-M
# graphs	1000	1500	5000	2000	5000
# classes	2	3	3	2	5
avg.# nodes	19.8	13.0	74.5	429.6	508.5

Explanation for Graph Neural Network In this task, we follow the setting in GNNExplainer [39] and PGExplainer [17] and construct four kinds of node classification datasets. (1) *BA-Shapes* is a single graph without node features. We first generate a base

Barabasi-Albert (BA) graph with 300 nodes. Then we attach 80 "house"-structured graph motifs to the nodes in the base graph randomly. We add 0.1 random edges to perturb the graph. Nodes in the base BA graph is labelled with 0. The top, middle, and bottom nodes of the houses are labelled with 1, 2 and 3 respectively. (2) *BA-Community* is a union of two BA-Shapes graphs with node features. We assign eight classes to the nodes based on the BA-Shapes graph community and the structural roles. We generate node features using two Gaussian distributions for the two BA-Shapes respectively. (3) *Tree-Cycles* is also a base-motif graph. We use an 8-level balance binary tree as the base graph, and attach 80 six-node circle graph motifs to the base graph. (4) *Tree-Grids* uses the same base graph as Tree-Cycles but 3-by-3 grid graph motifs. Table 2 illustrates the synthesized datasets.

Experimental Settings: For the classification tasks, we consider three variants of our second-order continuous GNNs: *proposed*, where neither the damping nor the force term is used; *proposed**, which uses the force term; *proposed***, in which both the damping and the force term are used. In all models we use one linear layer, one continuous GNN layer, and one linear prediction layer. For *proposed*, we use a standard graph convolution layer to approximate the dynamics. For *proposed**, the force term is approximated by one linear linear with sigmoid function. For *proposed***, the damping term α is set to 0.95. The terminal time for the continuous layer is set to 15. The hidden dimension is 16. For the prediction layer, we use a dropout rate of 0.2. Cora and Citeseer are trained for 100 epochs using rmsprop optimizer with learning rate 0.005. The rest of the involved datasets are train for 200 epochs using Adam optimizer with learning rate 0.005. All results were collected using single NVIDIA P40 with 24GB GPU memory.

For the explanation task, we follow the quantitative evaluation settings in GNNExplainer [39] and PGExplainer [17]. The explanation is assessed as a binary classification of edges. Specifically, the edges inside motifs are regarded as positive, and negative otherwise. e_{ij} is viewed the prediction score. We report the average AUC scores and the standard deviations based on ten repeats of the experiments.

4.2 Baseline Methods

We include representative discrete GNN and continuous GNN methods as the baselines. Given the relation between our second-order method and augmented neural ODEs, we also include related methods. The following results are obtained by run the official implementation of the baselines. When the codes are not available, we cite the values from the original papers.

Discrete GNNs: For standard discrete GNNs we consider the Graph Convolutional Network (GCN) [13] and the Graph Attention Network (GAT) [31]. Both are the most representative GNN methods, and are conidered as widely used baselines for related continuous GNNs methods [25, 35].

Continuous GNNs: We include three state-of-the-art continuous GNN models as the related baselines, Graph Neural Ordinary Differential Equations (GDE) [25], Ordinary differential equations on graph networks (GODE) [47], and continuous GNN (CGNN) [35]. We include three variants for GDE with different ODE solvers. For CGNN, we also include two variants, CGNN with weights (WCGNN) and diffusion CGNN.

Augmented Neural ODEs: We include Augmented Neural ODEs (ANODE) and Second Order Neural ODEs (SONODE) as baselines. Both ANODE [6] and SONODE [21] are designed for non-structural data, so we only use the node features and drops the graph for these two methods.

4.3 Comparison Results

4.3.1 Semi-Supervised Node Classification. : The results for semi-supervised node classification on standard test-train splits are summarized in Table 3 . In most cases, continuous methods outperforms discrete GNNs. For non-graph methods ANODE and SONODE, the performance is significantly lower than GNNs. The results demonstrates that our approach has superior performance compared to the first-order continuous GNNs in most cases. Particularly, damping term can help improve the performance. Meanwhile, the huge gap between non-graph ANODEs and our approach show that the structural information is critical in node classification, and partial differential equations are of potential. Table 3 summarizes the results for both the predefined and the random test-train splits. Similar observations can be confirmed. It is also observed that GAT is less stable considering the data splits, compared to the continuous methods.

4.3.2 Graph Classification. : The results for graph classification are summarized in Table 4. Compare to the standard baselines, continuous GNNs are generally superior in peformance. Among all continuous methods, our approach with a damping term performs the best. We also notice that the average size of graphs has a large influence on the performance of continuous GNNs. For example, the improvement of continuous methods are much more prominent for RDT-B and RDT-M than for IMDB-B and IMDB-M.

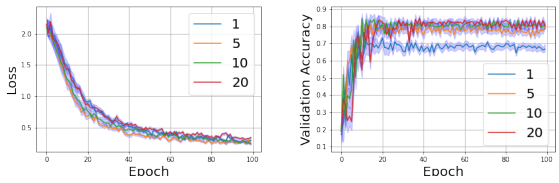


Figure 4: Training loss (left) and validation accuracy (right) under different integration time.

4.3.3 Prediction Explanation. : The results for prediction explanation are summarized in Table 5. The explanation accuracy is computed based on the ground truth node labels for the synthetic datasets. A better explainability method has higher prediction scores for edges that are in the ground-truth explanation. We compare the explainability of our method to two first-order continuous GNNs. Proposed First-order omit the high order information, which serves as an ablation study. Proposed Second-order is our full method. The quantitative results show that different first order models show similar explainability in most cases. Our second-order method shows some improvement compared to the first order methods.

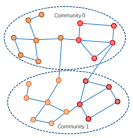
Table 3: Node classification results on citation networks. The Cora, Citeseer, and Pubmed columns summarize the results using standard test-training split. The corresponding columns with superscript * summarize the results over 10 random splits test-training split. The values for GCN-GODE and GAT-GODE are taken from the original paper. Best-performing methods are bold faced, and the runner-ups are underlined. For the random split, The results of GCN-GODE and GAT-GODE are not available and are denoted by –.

Dataset	Cora	Citeseer	Pubmed	Cora*	Citeseer*	Pubmed*
GCN [13]	81.8 ± 0.8	70.8 ± 0.8	80.0 ± 0.5	80.5 ± 1.1	72.3 ± 0.9	79.4 ± 1.5
GAT [31]	82.6 ± 0.7	71.5 ± 0.8	79.7 ± 0.4	80.1 ± 1.5	72.4 ± 1.2	78.4 ± 1.6
GDE-rk2 [25]	82.9 ± 0.5	72.4 ± 0.5	79.8 ± 0.4	81.4 ± 0.5	72.1 ± 0.6	78.9 ± 0.7
GDE-rk4 [25]	83.8 ± 0.5	72.4 ± 0.5	79.7 ± 0.4	81.6 ± 0.4	71.8 ± 0.5	79.5 ± 0.6
GDE-dpr5 [25]	81.9 ± 1.1	69.0 ± 1.1	78.3 ± 0.7	79.3 ± 1.3	68.1 ± 0.9	75.1 ± 0.9
GCN-GODE [47]	81.8 ± 0.3	72.4 ± 0.8	80.1 ± 0.3	–	–	–
GAT-GODE [47]	83.3 ± 0.3	72.1 ± 0.6	79.1 ± 0.5	–	–	–
CGNN [35]	83.8 ± 1.1	72.7 ± 0.6	**82.2** ± 0.5	82.5 ± 1.0	72.2 ± 1.0	80.4 ± 1.3
WCGNN [35]	83.6 ± 0.7	72.8 ± 0.7	82.1 ± 0.5	82.0 ± 1.0	72.2 ± 0.9	79.8 ± 0.9
ANODE [6]	58.4 ± 1.5	61.6 ± 0.8	69.8 ± 0.5	59.6 ± 1.3	59.9 ± 1.1	70.7 ± 0.5
SONODE [21]	59.9 ± 1.3	61.5 ± 0.9	70.1 ± 0.4	60.1 ± 1.4	60.3 ± 0.9	71.0 ± 0.6
Proposed	83.3 ± 0.9	72.9 ± 0.8	82.0 ± 0.7	82.5 ± 0.8	**72.7** ± 0.6	79.4 ± 1.0
Proposed*	83.5 ± 0.6	72.5 ± 1.0	81.5 ± 0.5	82.6 ± 0.6	71.6 ± 0.8	79.9 ± 0.8
Proposed**	**84.3** ± 0.8	**73.2** ± 0.9	82.1 ± 0.5	**83.5** ± 1.0	72.1 ± 0.8	**80.9** ± 0.9

Table 4: Graph Classification results.

Dataset	IMDB-B	IMDB-M	COLLAB	RDT-B	RDT-M
GCN [13]	70.9 ± 4.6	**49.4** ± 3.2	73.3 ± 3.1	82.9 ± 2.8	52.4 ± 2.1
GDE-rk2 [25]	70.4 ± 3.5	48.5 ± 3.6	75.6 ± 1.6	87.9 ± 1.7	54.1 ± 1.6
GDE-rk4 [25]	71.5 ± 3.8	48.1 ± 2.9	74.9 ± 1.3	86.4 ± 1.5	54.3 ± 1.4
GDE-dpr5 [25]	69.1 ± 4.1	48.2 ± 3.0	71.0 ± 2.5	83.1 ± 3.4	52.0 ± 2.2
CGNN [35]	71.4 ± 3.3	47.4 ± 3.6	75.4 ± 1.9	87.7 ± 2.2	55.3 ± 1.5
WCGNN [35]	71.8 ± 2.9	48.9 ± 2.3	75.7 ± 2.1	88.2 ± 2.3	54.9 ± 1.2
Proposed	71.6 ± 3.1	**49.4** ± 3.0	75.2 ± 2.2	88.0 ± 2.0	53.7 ± 1.5
Proposed*	71.9 ± 3.8	48.5 ± 3.4	76.0 ± 1.7	87.5 ± 2.1	53.2 ± 1.4
Proposed**	**72.5** ± 3.6	48.8 ± 2.9	**76.3** ± 2.5	**89.6** ± 1.8	**56.0** ± 1.7

Table 5: Explanation AUC results.

	BA-Shapes	BA-Community	Tree-Cycles	Tree-Grid
Base				
Motifs				
GDE-rk2 [25]	0.934 ± 0.013	0.877 ± 0.022	0.914 ± 0.010	0.873 ± 0.016
CGNN [35]	0.926 ± 0.017	0.849 ± 0.016	0.955 ± 0.015	0.881 ± 0.007
Proposed First-order	0.930 ± 0.015	0.911 ± 0.024	0.949 ± 0.008	0.887 ± 0.015
Proposed Second-order	**0.954** ± 0.011	**0.955** ± 0.021	**0.975** ± 0.011	**0.912** ± 0.013

4.4 Analysis of Model Parameters

4.4.1 The model performance v.s. Integration Time. : Figure 4 describes the semi-supervised node classification performance of our method when the model is trained using different integration time. It can be observed that the predicting power is robust to the different integration time. For the first order method, heuristic augmented features are used to achieve a similar phenomenon. Our method suggest a more explicit explanation by defining the high-order behaviors.

4.4.2 The Effect of Damping Term. : We notice that the damped version of our approach usually yield the best performance, in

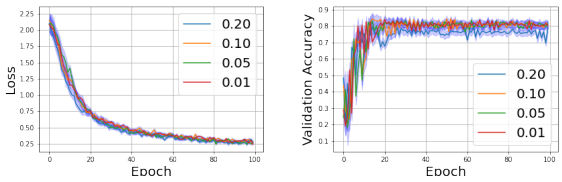


Figure 5: Training loss (left) and validation accuracy (right) with different damping terms.

which α is a tunable hyperparamater. Figure. 5 describes the semi-supervised node classification performance of our method when the model is trained using different α. It can be observed that our approach is insensitive to α for a wide range of $[0.01, 0.10]$. When α is larger (for example, 0.2), the gradients vanishes quickly and the model performance is witnessed to decrease slightly.

5 CONCLUSION

In this paper we propose a second order Continuous GNN. Compared to existing methods, our approach employs a second order dynamics, which avoids the over-smoothing and provide additional information to understand the prediction. Our method also can be viewed as a continuous GNN model with interpretable augmented features. Extensive experiments demonstrate that our approach outperforms related baselines for social network applications and has better interpretability.

REFERENCES

[1] Mikhail Belkin et al. 2008. Towards a theoretical foundation for Laplacian-based manifold methods. *J. Comput. System Sci.* 74, 8 (2008), 1289–1308.
[2] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366* (2018).
[3] Miles Cranmer, Sam Greydanus, et al. 2020. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630* (2020).
[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.
[5] Zhiwei Deng, Megha Nawhal, Lili Meng, and Greg Mori. 2019. Continuous graph flow. *arXiv preprint arXiv:1908.02436* (2019).
[6] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. 2019. Augmented neural odes. *arXiv preprint arXiv:1904.01681* (2019).
[7] David K Duvenaud, Dougal Maclaurin, et al. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc., 2224–2232.
[8] Lawrence C Evans. 1998. Partial differential equations. *Graduate studies in mathematics* 19, 2 (1998).
[9] Justin Gilmer, Samuel S Schoenholz, et al. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*. PMLR, 1263–1272.
[10] Sam Greydanus, Misko Dzamba, and Jason Yosinski. 2019. Hamiltonian neural networks. *arXiv preprint arXiv:1906.01563* (2019).
[11] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216* (2017).
[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[13] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[14] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. In *International conference on machine learning*. PMLR, 3734–3743.
[15] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9267–9276.
[16] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. 2018. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. *arXiv preprint arXiv:1808.09602* (2018).
[17] Dongsheng Luo, Wei Cheng, et al. 2020. Parameterized explainer for graph neural network. *arXiv preprint arXiv:2011.04573* (2020).
[18] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology* 27, 1 (2001), 415–444.
[19] Tomas Mikolov et al. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[20] Seth A Myers, Aneesh Sharma, Pankaj Gupta, and Jimmy Lin. 2014. Information network or social network? The structure of the Twitter follow graph. In *Proceedings of the 23rd International Conference on World Wide Web*. 493–498.
[21] Alexander Norcliffe, Cristian Bodnar, et al. 2020. On second order behaviour in augmented neural odes. *arXiv preprint arXiv:2006.07220* (2020).
[22] Kenta Oono and Taiji Suzuki. 2019. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947* (2019).
[23] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287* (2020).
[24] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.
[25] Michael Poli, Stefano Massaroli, et al. 2019. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532* (2019).
[26] Lev Semenovich Pontryagin. 2018. *Mathematical theory of optimal processes*. Routledge.
[27] Andres Potapczynski, Gabriel Loaiza-Ganem, and John P Cunningham. 2019. Invertible gaussian reparameterization: Revisiting the gumbel-softmax. *arXiv preprint arXiv:1912.09588* (2019).
[28] Haoteng Tang, Lei Guo, et al. 2022. Hierarchical Brain Embedding Using Explainable Graph Learning. In *2022 IEEE 19th International Symposium on Biomedical Imaging (ISBI)*. IEEE, 1–5.
[29] Haoteng Tang, Guixiang Ma, et al. 2021. CommPOOL: An interpretable graph pooling framework for hierarchical graph representation learning. *Neural Networks* 143 (2021), 669–677.
[30] Jian Tang, Meng Qu, et al. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.
[31] Petar Veličković, Guillem Cucurull, et al. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).
[32] Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. 2021. Dissecting the Diffusion Process in Linear Graph Convolutional Networks. *arXiv preprint arXiv:2102.10739* (2021).
[33] Felix Wu, Amauri Souza, et al. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.
[34] Shu Wu, Yuyuan Tang, et al. 2019. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 346–353.
[35] Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. 2020. Continuous graph neural networks. In *International Conference on Machine Learning*. PMLR, 10432–10441.
[36] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).
[37] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 1365–1374.
[38] Rex Ying et al. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.
[39] Rex Ying, Dylan Bourgeois, et al. 2019. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems* 32 (2019), 9240.
[40] Zhitao Ying et al. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*. 4805–4815.
[41] Hao Yuan, Jiliang Tang, et al. 2020. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 430–438.
[42] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
[43] Yanfu Zhang, Hongchang Gao, et al. 2022. Robust Self-Supervised Structural Graph Neural Network for Social Network Prediction. In *Proceedings of the ACM Web Conference 2022*. 1352–1361.
[44] Yanfu Zhang and Heng Huang. 2019. New graph-blind convolutional network for brain connectome data analysis. In *International Conference on Information Processing in Medical Imaging*. Springer, 669–681.
[45] Da Zheng, Minjie Wang, et al. 2020. Learning graph neural networks with deep graph library. In *Companion Proceedings of the Web Conference 2020*. 305–306.
[46] Jiong Zhu, Yujun Yan, et al. 2020. Generalizing graph neural networks beyond homophily. *arXiv preprint arXiv:2006.11468* (2020).
[47] Juntang Zhuang, Nicha Dvornek, et al. 2019. Ordinary differential equations on graph networks. (2019).