GRAND++: GRAPH NEURAL DIFFUSION WITH A SOURCE TERM

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose GRAph Neural Diffusion with a source term (GRAND++) for graph deep learning with a limited number of labeled nodes, i.e., low-labeling rate. GRAND++ is a class of continuous-depth graph deep learning architectures whose theoretical underpinning is the diffusion process on graphs with a source term. The source term guarantees two interesting theoretical properties of GRAND++: (i) the representation of graph nodes, under the dynamics of GRAND++, will not converge to a constant vector over all nodes even as the time goes to infinity, which mitigates the over-smoothing issue of graph neural networks and enables graph learning in very deep architectures. (ii) GRAND++ can provide accurate classification even when the model is trained with a very limited number of labeled training data. We experimentally verify the above two advantages on various graph deep learning benchmark tasks, showing a significant improvement over many existing graph neural networks.

1 INTRODUCTION

Graph neural networks (GNNs) are the backbone for deep learning on graphs. Recent GNN architectures include graph convolutional networks (GCNs) [30], ChebyNet [16], GraphSAGE [29], neural graph fingerprints [20], message passing neural networks [28], and graph attention networks (GATs) [52]. These graph deep networks have achieved success in many applications, including computational physics and computational chemistry [20, 28, 3], recommender systems [39, 58], and social networks [59, 45]. Hyperbolic GNNs have also been proposed to enable certain kinds of data embedding with much smaller distortion [11, 35]. See [6] for some recent advances of GNN algorithm development and applications.

A well-known problem of GNNs is that increasing the depth of GNNs often results in a significant drop in performance on various graph learning tasks. This performance degradation has been widely interpreted as the *over-smoothing* issue of GNNs [33, 42, 12]. Intuitively, GNN layers update the node representation by taking a weighted average of its neighbors' features, making representations for neighboring nodes to be similar. As the GNN architecture gets deeper, all nodes' representation will become indistinguishable resulting in over-smoothing. In Sec. 2, we briefly show that certain GNNs have a diffusive nature which makes over-smoothing inevitable. Another interesting interpretation of the GNN performance degradation is via a *bottleneck* [1], since a GNN tends to represent exponentially growing information from neighbors with fixed-size vectors. Several algorithms have been proposed to mitigate the over-smoothing of GNNs, including skip connection and dilated convolution [32], Jumping Knowledge [56], DropEdge [47], PairNorm [60], graph neural diffusion (GRAND) [10], and wave equation motivated GNNs [21]. Nevertheless, developing deep GNN architectures is still in its infancy compared to the development of other deep networks.

Besides suffering from over-smoothing, we notice that the accuracy of existing GNNs drops severely when they are trained with a limited labeled data. As illustrated in Fig. 1, the test accuracy of several celebrated GNN architectures, including GCN, GAT, and GraphSage, drops rapidly when they are trained with fewer labeled data. Moreover, the variance of classification accuracy grows significantly as number of labeled nodes drops. Indeed, semi-supervised graph learning with very low-labeling rates has been studied in the Laplace learning and graph deep learning settings, see, e.g., [34, 9, 23]; one question is *can we develop new GNN architectures to improve the performance of graph deep learning in low-labeling rate regimes?*

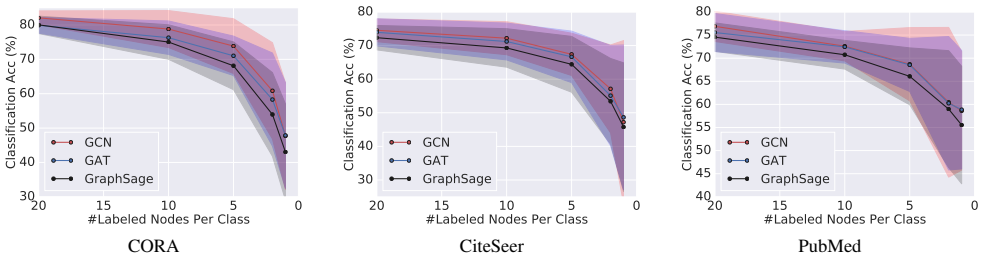


Figure 1: Test accuracy of GCN, GAT, and GraphSage vs. the number of labeled training data per class. All networks have two layers, and each experiment is run with 100 splits and 20 random seeds following [10]. The accuracy drops rapidly with fewer labeled data for training. CORA, CiteSeer, and PubMed have 2485, 2120, and 19717 nodes in total respectively; Appendix D.1 has more details. Results on more benchmark GNN architectures are in Appendix D.4.

1.1 Our contribution

With the above GNN problems in mind, we focus on developing new continuous-depth GNNs to overcome over-smoothing and boost the accuracy of GNNs with a limited number of labeled data. We first present a random walk interpretation of the GRAND model [10], revealing a potentially inevitable over-smoothing phenomenon when GRAND is implicitly very deep. Based on the random walk viewpoint of GRAND, we then propose graph neural diffusion with a source term (GRAND++) that corrects the bias arising from the diffusion process, see Sec. 4 for details. GRAND++ theoretically guarantees that: (i) under GRAND++ dynamics, the graph node features do not converge to a constant vector over all nodes even as the time goes to infinity, and (ii) GRAND++ can provide accurate prediction even when it is trained with a limited number of labeled nodes. Moreover, these theoretical results resonate with the practical advantages of GRAND++. We summarize the major practical advantages of GRAND++ below.

- GRAND++ can effectively overcome the over-smoothing issue; it is remarkably more accurate than existing GNNs when the architecture is very deep.
- GRAND++ is suitable for graph deep learning when only a few nodes are labeled as training data. Moreover, in the low-labeling rates, GRAND++ can be more accurate when the network is deeper.
- GRAND++ inherits the continuous-depth merit from GRAND, which defines the network depth implicitly and enables memory-efficient training by using the adjoint method.

1.2 Related work

Diffusion on graphs and continuous-depth graph neural networks. Diffusion has been defined on graphs, see, e.g., [25, 24], and used in various applications, including data clustering and dimension reduction [15, 4], image processing [27, 22, 17, 36], and semi-supervised graph nodes classification [63, 61]. From the numerical viewpoint, fast algorithms have been proposed for using diffusion on graphs to solve penalized graph cut problems [26]. The connection between GNNs and diffusion on graphs has been studied substantially. For instance, GNN has been interpreted as a diffusion process on graphs, which performs low-pass filtering on the input features [41]. Moreover, insights from the diffusion process on graphs have been used to improve the performance of GNNs, see, e.g., [2, 34, 31, 54].

Leveraging neural ordinary differential equations (ODEs) [13], continuous-depth GNNs have been proposed, see, e.g., [43, 55, 64]. One recent work is GRAND [10], which parameterizes the diffusion equation on graphs with a neural network. See Sec. 3 for a brief review of GRAND.

Neural ODEs. Neural ODEs [13] are a class of continuous-depth neural networks whose depth is defined implicitly. Training neural ODEs using the adjoint method [44] is more memory efficient than training other neural networks using backpropagation. We provide a brief review of neural ODEs and the adjoint method in Appendix C. GRANDs [10] are a class of neural partial differential equations (PDEs) on graphs that can also be considered as a coupled system of neural ODEs. Furthermore, GRANDs are also trained by using the adjoint method.

Laplace learning and Poisson learning. Laplace learning has been used for semi-supervised data classification [63, 61, 53], image processing [7, 27], etc. Direct application of Laplace learning with Gaussian weights [5] or locally linear embedding weights [48] for the above tasks may cause inference inconsistency when only a limited number of graph nodes are labeled. Several algorithms address the inference inconsistency at low labeling rate. They include up-weighting the weights of the labeled data [50] and the p-Laplacian [8, 46, 62]. In [9], the authors have proposed Poisson learning for improving Laplace learning at extremely low-labeling rate regimes. Poisson learning augments Laplace learning with a Green's function at each labeled data, enabling accurate node classification when only a few labeled data are available.

1.3 NOTATION

We denote scalars by lower- or upper-case letters and vectors and matrices by lower- and upper-case boldface letters, respectively. For a matrix $\boldsymbol{A}$, we denote its transpose as $\boldsymbol{A}^\top$ and its Hadamard product with another matrix $\boldsymbol{B}$ as $\boldsymbol{A} \odot \boldsymbol{B}$. We write the set $\{1, 2, \cdots, n\}$ as $[n]$. We denote the probability and expectation of a given random variable $\boldsymbol{x}$ as $\mathbb{P}(\boldsymbol{x})$ and $\mathbb{E}[\boldsymbol{x}]$, respectively. The meaning of other notations can be inferred from the context.

1.4 ORGANIZATION

The paper is organized as follows: In Sec. 2, we review diffusion equation on graphs and its connection to GNNs. In Sec. 3, we briefly review and present a random walk interpretation of GRAND. Leveraging the random walk viewpoint of GRAND, we propose GRAND++ for deep graph learning with theoretical guarantees in Sec. 4. We numerically verify the effectiveness of GRAND++ in Sec. 5. Technical proofs are provided in the appendix.

2 BACKGROUND

Diffusion equation on graphs. Let $G = (\boldsymbol{X}, \boldsymbol{W})$ represent an undirected graph with n nodes, where $\boldsymbol{X} = \big([\boldsymbol{x}^{(1)}]^\top, \cdots, [\boldsymbol{x}^{(n)}]^\top\big)^\top \in \mathbb{R}^{n\times d}$ with each row $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ a feature vector and $\boldsymbol{W} := \big(W_{ij}\big)$ a $n \times n$ matrix with W_{ij} representing the similarity (edge weight) between the ith and j^{th} feature vectors, and we assume $W_{ij} = W_{ji}$. Consider the following diffusion process that evolves the feature matrix $\boldsymbol{X}$ on the graph (see Appendix A for a brief review of calculus on graphs):

$$\frac{\partial \boldsymbol{X}(t)}{\partial t} = \operatorname{div}\big(\boldsymbol{G}(\boldsymbol{X}(t), t) \odot \nabla \boldsymbol{X}(t)\big), \tag{1}$$

where $\boldsymbol{X}(t) = \big([\boldsymbol{x}^{(1)}(t)]^\top, \cdots, [\boldsymbol{x}^{(n)}(t)]^\top\big)^\top \in \mathbb{R}^{n\times d}$ with $\boldsymbol{x}^{(i)}(0) = \boldsymbol{x}^{(i)}$. The matrix $\boldsymbol{G}(\boldsymbol{X}(t), t)$ is chosen such that $\boldsymbol{W} \odot \boldsymbol{G}$ is right-stochastic, i.e., each row of $\boldsymbol{W} \odot \boldsymbol{G}$ summing to 1. In the machine learning setting, we can parameterize $\boldsymbol{G}$ with learnable parameters θ which we denote by $\boldsymbol{G}(\boldsymbol{X}(t), t, \theta)$. The initial features are evolved under the diffusion dynamics (1) from $t = 0$ to T to learn the final representation $\boldsymbol{X}(T)$ for further machine learning tasks.

In the simplest case when $\boldsymbol{G}(\boldsymbol{X}(t), t)$ is only dependent on the initial node features $\boldsymbol{X}$, i.e., $\boldsymbol{G}$ is time-independent, right-stochasticity implies the condition $\sum_j W_{ij}G_{ij} = 1$ for all i, and so we pick $G_{ij} = 1/d_i$ with $d_i = \sum_{j=1}^n W_{ij}$. In this case the right-hand side of the diffusion equation (1) reduces to the negative of the random-walk Laplacian applied to $\boldsymbol{X}(t)$ and (1) becomes

$$\frac{\partial \boldsymbol{X}(t)}{\partial t} = \operatorname{div}\big(\boldsymbol{G}(\boldsymbol{X}(t), t) \odot \nabla \boldsymbol{X}(t)\big) = -\boldsymbol{L}\boldsymbol{X}(t), \tag{2}$$

where $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{W} := \boldsymbol{I} - \boldsymbol{A}$ is the random walk Laplacian and $\boldsymbol{D}$ is diagonal with $\boldsymbol{D}_{ii} = d_i$. More details of random walk Laplacian and diffusion on graphs can be found in, e.g., [14, 25, 24].

Graph neural networks. Applying a forward Euler discretization, with step size δ_t, of (2) gives

$$\boldsymbol{X}(k\delta_t) = \boldsymbol{X}((k-1)\delta_t) - \delta_t \boldsymbol{L}\boldsymbol{X}((k-1)\delta_t) := \tilde{\boldsymbol{L}}\boldsymbol{X}((k-1)\delta_t), \quad \text{for } k = 1, 2, \cdots, K, \tag{3}$$

$T = K\delta_t$, and $\boldsymbol{X}(0) = \boldsymbol{X}$. Note that (3) is a prototype for motivating GNNs: by introducing weights $\boldsymbol{W}^{(k)} \in \mathbb{R}^{d\times d}$ and a nonlinearity σ, e.g., ReLU, into (3), we have

$$\boldsymbol{X}((k+1)\delta_t) = \sigma\big(\tilde{\boldsymbol{L}}\boldsymbol{X}(k\delta_t)\boldsymbol{W}^{(k)}\big). \tag{4}$$

The model in (4) is similar to the well-established GCN architecture proposed in [30]. The diffusive nature of the GNN architecture in (4) further explains the over-smoothing issue of training deep GNNs; the deeper the network architecture is, the more the node features diffuse. Eventually, all nodes share similar features and become indistinguishable. See Sec. 4 for a detailed analysis.

3 GRAND AND ITS RANDOM WALK INTERPRETATION

GRAND is a new continuous-depth GNN proposed in [10]. It integrates a learnable encoder function ϕ and a learnable decoder function ψ with the neural network parameterized graph diffusion process, resulting in the prediction $\boldsymbol{Y} = \psi(\boldsymbol{X}(T))$, where $\boldsymbol{X}(T)$ is computed as

$$\boldsymbol{X}(T) = \boldsymbol{X}(0) + \int_0^T \frac{\partial \boldsymbol{X}(t)}{\partial t} dt, \quad \text{with} \quad \boldsymbol{X}(0) = \phi(\boldsymbol{X}), \tag{5}$$

where $\partial \boldsymbol{X}(t)/\partial t$ is given by the graph diffusion equation (2). From the neural ODE perspective, we can perform forward propagation of GRAND, i.e., we solve (5), using numerical ODE solvers.

In the simplest case, when $\boldsymbol{G}$ is only dependent on the initial node features, we can rewrite (2) as

$$\frac{\partial \boldsymbol{X}(t)}{\partial t} = \big(\boldsymbol{A}(\boldsymbol{X}) - \boldsymbol{I}\big)\boldsymbol{X}(t), \tag{6}$$

GRAND models the diffusivity $\boldsymbol{A}(\boldsymbol{X})$ in (6) by the multi-head self-attention mechanism; potential choices of the attention function include the ones proposed in [51, 52]. More precisely, in GRAND $\boldsymbol{A}(\boldsymbol{X}) = \frac{1}{h}\sum_h \boldsymbol{A}^h(\boldsymbol{X})$ with h being the number of heads and the attention matrix $\boldsymbol{A}^l(\boldsymbol{X}) = (a^l(\boldsymbol{x}_i, \boldsymbol{x}_j))$, for $l = 1, \cdots, h$, is computed as follows:

$$a^l(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\exp\big(\mathrm{LeakyReLU}(\boldsymbol{a}^{l\top}[\boldsymbol{W}^l\boldsymbol{x}_i \| \boldsymbol{W}^l\boldsymbol{x}_j])\big)}{\sum_{k\in\mathcal{N}_i}\exp\big(\mathrm{LeakyReLU}(\boldsymbol{a}^{l\top}[\boldsymbol{W}^l\boldsymbol{x}_i \| \boldsymbol{W}^l\boldsymbol{x}_k])\big)}, \tag{7}$$

where $\boldsymbol{W}^l$ and $\boldsymbol{a}^l$ are learned, $\|$ is the concatenation operator, and $\mathcal{N}_i$ is the index set of the nodes that are connected to the ith node in the graph. GRAND with the attention in (7) is called GRAND-l. Time-dependent attention and graph rewiring can be integrated into GRAND, resulting in GRAND-nl and GRAND-nl-rw, respectively [10]. From the ODE viewpoint, GRAND and its variants are a class of coupled neural ODEs defined on an unweighted graph. Their merits include continuous-depth and memory-efficient training using the adjoint method [44, 13].

3.1 RANDOM WALK VIEWPOINT OF GRAND

In this section, we present a random walk interpretation of GRAND. The connection between graph random walks and the diffusion equation has been extensively studied, but we recap the key idea here to motivate the new GRAND with a source term architecture. Let $\{\boldsymbol{B}^{(i)}(k)\}_{k\in\mathbb{N}}$ be the random walk on $\{\boldsymbol{x}^{(j)}(0)\}_{j=1}^n$ defined by

$$\begin{aligned} &\boldsymbol{B}^{(i)}(0) = \boldsymbol{x}^{(i)}(0) \\ &\mathbb{P}\big(\boldsymbol{B}^{(i)}(k+1) = \boldsymbol{x}^{(\ell)} | \boldsymbol{B}^{(i)}(k) = \boldsymbol{x}^{(j)}\big) = \begin{cases} 1-\delta_t & \text{if } \ell = j \\ \frac{\delta_t W_{j\ell}}{d_j} & \text{if } \ell \neq j \end{cases} \end{aligned} \tag{8}$$

where $d_j = \sum_{\ell=1}^n W_{j\ell}$ (assume $W_{\ell\ell} = 0$ for all ℓ). Proposition 1 below is well-known, see [63]. We provide the proof of Proposition 1 and all the subsequent theoretical results in Appendix B.

Proposition 1 *Let $\boldsymbol{X}$ solve* (3) *and $\boldsymbol{B}^{(i)}$ be the random walk determined by* (8)*. Then*

$$\boldsymbol{x}^{(i)}(\delta_t k) = \mathbb{E}\big[\boldsymbol{B}^{(i)}(k)\big].$$

Proposition 2 below gives the stationary distribution of the random walk $\{\boldsymbol{B}^{(i)}(k)\}_{k\in\mathbb{N}}$.

Proposition 2 *Assume the graph $G = (\boldsymbol{X}, \boldsymbol{W})$ is connected. Then, the stationary distribution of $\{\boldsymbol{B}^{(i)}(k)\}_{k\in\mathbb{N}}$ is*

$$\pi = \left(\frac{d_1}{\sum_{j=1}^n d_j}, \ldots, \frac{d_n}{\sum_{j=1}^n d_j}\right), \tag{9}$$

which is independent of the starting position $\boldsymbol{x}^{(i)}$.

Furthermore, we have the following theoretical result on the asymptotic behavior of graph node features under the GRAND dynamics given by (3).

Proposition 3 *Assume the graph $G = (\boldsymbol{X}, \boldsymbol{W})$ is connected. Then for all $i = 1, \cdots, n$, we have*

$$\boldsymbol{x}^{(i)}(k\delta_t) \to \overline{\boldsymbol{x}} := \sum_{j=1}^{n} \boldsymbol{x}^{(j)} \pi_j, \quad as \quad k \to \infty.$$

Hence, for the case of (3), i.e., GRAND-l, we expect the output to be approximately independent of the input, due to over-smoothing. Of course, once we reintroduce the $\boldsymbol{X}(t)$ dependence back into $\boldsymbol{G}$ in (1) and (2) or into the operator $\boldsymbol{A}$ in (6) then the above arguments no longer hold. Nevertheless, the GRAND architectures are built on a principle that is ill-suited to deep networks. In the next section we introduce a source term and perform a similar random walk analysis that illustrates how the new architecture can be better suited for deep GNN architectures.

4 GRAND++: Graph Neural Diffusion with A Source Term

4.1 Algorithm and formulation

At the core of GRAND++ is the introduction of a source term into GRAND, leveraging the random walk viewpoint of the diffusion process. We take a small subset of feature vectors, indexed by $\mathcal{I} \subseteq [n]$, believed to be "trustworthy" for use as a source term. In particular, we use the features of labeled data. The GRAND++ dynamics are defined by a diffusion equation with a source term

$$\frac{\partial \boldsymbol{z}^{(i)}(t)}{\partial t} = \operatorname{div}\left[\boldsymbol{G}(\boldsymbol{Z}(t), t) \odot \nabla \boldsymbol{Z}(t)\right]^{(i)} + \sum_{j \in \mathcal{I}} \delta_{ij} C_j \tag{10}$$

where C_j is the source at feature vector of node j. Below we motivate a particular choice of C_j.

The key idea is to first characterise the bias that arises from the diffusion and use that to propose a correction via the choice of source terms C_j. Following the simplifications in (2), our diffusion equation (without the source term), for $i \in \mathcal{I}$, follows the approximate dynamics when $t \gg 1$

$$\frac{\partial \boldsymbol{x}^{(i)}(t)}{\partial t} = -\left[\boldsymbol{L}\boldsymbol{X}(t)\right]^{(i)} = -\boldsymbol{x}^{(i)}(t) + \frac{1}{d_i} \sum_{j=1}^{n} W_{ij} \underbrace{\boldsymbol{x}^{(j)}(t)}_{\approx \sum_{j=1}^{n} \boldsymbol{x}^{(j)}(0)\pi_j =: \widetilde{\boldsymbol{x}}} \approx -\boldsymbol{x}^{(i)}(t) + \widetilde{\boldsymbol{x}}.$$

For $i \in \mathcal{I}$ we choose c such that $\frac{\partial \boldsymbol{z}^{(i)}(t)}{\partial t} \approx 0$ so that our trustworthy feature vectors are approximately fixed. This implies that we choose $C_i = \boldsymbol{x}^{(i)} - \widetilde{\boldsymbol{x}}$.

One can in fact choose $\widetilde{\boldsymbol{x}}$ with a certain degree of freedom. If we initialise $\boldsymbol{X}(0) = \boldsymbol{X}$ then we obtain $\widetilde{\boldsymbol{x}} = \overline{\boldsymbol{x}}$ as in the previous section. However, as the similarities are encoded in the graph weights, and the diffusion dynamics will drive it towards a non-trivial state, we can choose a different initialization than $\boldsymbol{X}(0) = \boldsymbol{X}$. Through connections with random walks, we motivate the choice of $\widetilde{\boldsymbol{x}} = \frac{1}{|\mathcal{I}|} \sum_{j \in \mathcal{I}} \boldsymbol{x}^{(j)}$ in the next subsection.

Our final model is therefore based on the following diffusion equation with a source term

$$\frac{\partial \boldsymbol{z}^{(i)}(t)}{\partial t} = \operatorname{div}\left[\boldsymbol{G}(\boldsymbol{Z}(t), t) \odot \nabla \boldsymbol{Z}(t)\right]^{(i)} + \sum_{j \in \mathcal{I}} \delta_{ij} \left(\boldsymbol{x}^{(i)} - \widetilde{\boldsymbol{x}}\right). \tag{11}$$

For the initial condition we use any initialisation satisfying

$$\sum_{i=1}^{n} \boldsymbol{z}^{(i)}(0) = \sum_{i \in \mathcal{I}} \frac{\boldsymbol{x}^{(i)} - \widetilde{\boldsymbol{x}}}{d_i}, \quad \text{where } \widetilde{\boldsymbol{x}} = \frac{1}{|\mathcal{I}|} \sum_{j \in \mathcal{I}} \boldsymbol{x}^{(j)}. \tag{12}$$

For example, we could choose

$$\boldsymbol{z}^{(i)}(0) = \begin{cases} \frac{1}{d_i}\left(\boldsymbol{x}^{(i)} - \widetilde{\boldsymbol{x}}\right) & \text{if } i \in \mathcal{I} \\ \boldsymbol{0} & \text{otherwise,} \end{cases} \quad \text{or} \quad \boldsymbol{z}^{(i)}(0) = \boldsymbol{x}^{(i)} - c,$$

where $c = \frac{1}{n}\left(\sum_{i=1}^{n} \boldsymbol{x}^{(i)} - \sum_{j\in\mathcal{I}} \frac{\boldsymbol{x}^{(j)}-\widetilde{\boldsymbol{x}}}{d_j}\right)$ is chosen such that (12) holds. We do not believe that the constant c (that shifts by a constant) is particularly important but it is included to provide a random walk interpretation which helps to understand the deep architecture (when T is big) behaviour of the model GRAND++. The justification for this choice will be made in Sec. 4.2. To summarize, the GRAND++ model in (11), with initial condition satisfying (12), simply adds a source term to the original GRAND model and uses a different initial condition. Therefore, the nonlinear diffusivity and graph rewiring tricks used by GRAND can be easily integrated into GRAND++. In terms of implementation, since GRAND++ merely changes the right-hand side of GRAND, which again can be regarded as a system of coupled first-order neural ODEs; we can leverage neural ODE training, testing, and inference for GRAND++ similar to GRAND.

In the next subsection we explore the random walk connection of the above model, suggesting that building a graph neural network based on the diffusion with source model does not suffer from the same degeneracy as we observed in Sec. 3 and is therefore better suited to build deep GNNs. In particular, we can write the diffusion with source model as the short time expected behaviour of a random walk and therefore we do not have the issue of reaching the stationary state (in other words passing the mixing time). Our experiments in Sec. 5 suggest the formal motivation holds and we are able to design deep GNNs.

4.2 THE RANDOM WALK PERSPECTIVE OF GRAND++

Let us continue to consider the simplified model in the previous subsection, i.e., assume the dynamics are governed by

$$\frac{\partial \boldsymbol{z}^{(i)}(t)}{\partial t} = -\left[\boldsymbol{L}\boldsymbol{X}(t)\right]^{(i)} + \sum_{j\in\mathcal{I}} \delta_{ij}\left(\boldsymbol{x}^{(i)} - \widetilde{\boldsymbol{x}}\right) \tag{13}$$

where the initial condition satisfies (12). Using the forward Euler discretisation of the above dynamics we have

$$\boldsymbol{z}^{(i)}(\delta_t k) = \boldsymbol{z}^{(i)}(\delta_t(k-1)) - \delta_t\left[\boldsymbol{L}\boldsymbol{Z}(\delta_t(k-1))\right]^{(i)} + \delta_t \sum_{j\in\mathcal{I}} \delta_{ij}\left(\boldsymbol{x}^{(i)} - \widetilde{\boldsymbol{x}}\right), \tag{14}$$

for $k = 1, 2, \ldots, K$ where again $T = K\delta_t$.

We use the same random walk as that introduced in Sec. 3, i.e. the random walk defined by (8), but we will now only consider random walks that are initialised on the nodes indexed by $\mathcal{I}$.

Proposition 4 *Let $\boldsymbol{Z}$ solve* (14) *with the initial condition satisfying* (12)*, and let $\boldsymbol{B}^{(i)}$ be the random walk determined by* (8)*. Then,*

$$\left|\boldsymbol{z}^{(i)}(k\delta_t) - \mathbb{E}\left[\sum_{s=0}^{k} \frac{1}{d_i} \sum_{j\in\mathcal{I}} \left(\boldsymbol{x}^{(j)} - \widetilde{\boldsymbol{x}}\right) \mathbb{1}_{\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(i)}}\right]\right| \to 0 \quad \text{as } k \to \infty.$$

Remark 1 *In the limit $k \to \infty$ the term*

$$\mathbb{E}\left[\sum_{s=0}^{k} \frac{1}{d_i} \sum_{j\in\mathcal{I}} \boldsymbol{x}^{(j)} \mathbb{1}_{\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(i)}}\right]$$

is formally a function of the random walk at all times. Whilst if k is very large (i.e. in comparison to the mixing time) we still have that

$$\mathbb{E}\left[\frac{1}{d_i} \sum_{j\in\mathcal{I}} \boldsymbol{x}^{(j)} \mathbb{1}_{\boldsymbol{B}^{(j)}(k)=\boldsymbol{x}^{(i)}}\right] = \frac{1}{d_i} \sum_{j\in\mathcal{I}} \underbrace{\mathbb{P}\left(\boldsymbol{B}^{(j)}(k) = \boldsymbol{x}^{(j)}\right)}_{\approx \pi_i} \approx \frac{\pi_i}{d_i} \sum_{j\in\mathcal{I}} \boldsymbol{x}^{(j)} \tag{15}$$

and on the other hand

$$\mathbb{E}\left[\frac{1}{d_i} \sum_{j\in\mathcal{I}} \widetilde{\boldsymbol{x}} \mathbb{1}_{\boldsymbol{B}^{(j)}(k)=\boldsymbol{x}^{(i)}}\right] = \frac{\widetilde{\boldsymbol{x}}}{d_i} \sum_{j\in\mathcal{I}} \underbrace{\mathbb{P}\left(\boldsymbol{B}^{(j)}(k) = \boldsymbol{x}^{(j)}\right)}_{\approx \pi_i} \approx \frac{\pi_i |\mathcal{I}| \widetilde{\boldsymbol{x}}}{d_i}. \tag{16}$$

From the definition of $\widetilde{\boldsymbol{x}}$ we see that (15) *and* (16) *are approximately equal. And therefore we can understand $\mathbb{E}[\frac{1}{d_i}\sum_{j\in\mathcal{I}}\widetilde{\boldsymbol{x}}\mathbb{1}_{\boldsymbol{B}^{(j)}(k)=\boldsymbol{x}^{(i)}}]$ as the long time behaviour of $\mathbb{E}[\frac{1}{d_i}\sum_{j\in\mathcal{I}}\boldsymbol{x}^{(j)}\mathbb{1}_{\boldsymbol{B}^{(j)}(k)=\boldsymbol{x}^{(i)}}]$ Very formally we can see that subtracting the long-time behaviour from the all-time behaviour leaves us with the short time behaviour. This provides one explanation as to why we do not expect the deep layers to be determined by the stationary state of the random walk (at which point there is little dependence on the initial layers, causing the deep layers to be approximately constant).*

Remark 2 *The random walk interpretation*

$$\mathbb{E}\left[\sum_{s=0}^{k}\frac{1}{d_i}\sum_{j\in\mathcal{I}}\left(\boldsymbol{x}^{(j)}-\widetilde{\boldsymbol{x}}\right)\mathbb{1}_{\boldsymbol{B}^{(j)}=\boldsymbol{x}^{(i)}}\right] \tag{17}$$

can be considered to be dual to the random walk interpretation in Sec. 3.1: in Sec. 3.1 we released the random walker from the node of interest, whilst now we release the random walkers from nodes indexed by $\mathcal{I}$ and see how many of them hit the node of interest. We note also that we do not require a lower bound on the size of the set $\mathcal{I}$. Indeed, if $|\mathcal{I}|$ is fixed whilst one takes the number of feature vectors $n\to\infty$ we still expect many properties of GRAND++, in particular Proposition 5 below, to hold. This is due to the asymptotic well-posedness of the dual random walk in low labeling rates [9].

Proposition 3 reveals that in the simple setting of (2), GRAND converges to a constant when its depth goes to infinity. However, this is not true for GRAND++ since the graph node features will not converge to a constant vector driven by the GRAND++, as shown in Proposition 5 below.

Proposition 5 *Assume the graph $G=(\boldsymbol{X},\boldsymbol{W})$ is connected. Then $\boldsymbol{z}^{(i)}(k\delta_t)$ does not converge to a constant vector, for all i, as $k\to\infty$.*

Remark 3 *Proposition 5 guarantees GRAND++ is less likely to suffer from over-smoothing than GRAND, and in particular it shows that we have a non-constant deep layer limit, i.e., as $t\to\infty$. Analysing the limit is beyond the scope of the paper but we have seen one characterisation in Proposition 4. By construction we have $\partial\boldsymbol{z}^{(i)}(t)/\partial t\approx\mathbf{0}$ for $i\in\mathcal{I}$ so one should expect that the deep layer limit is (close to) a smooth interpolation of the feature vectors labeled by $\mathcal{I}$.*

The continuous time model (10) is, in the special case of (13), the mean field limit of the probabilistic formulation (17). Our proposed algorithm is formulated from the mean field limit.

5 EXPERIMENTS

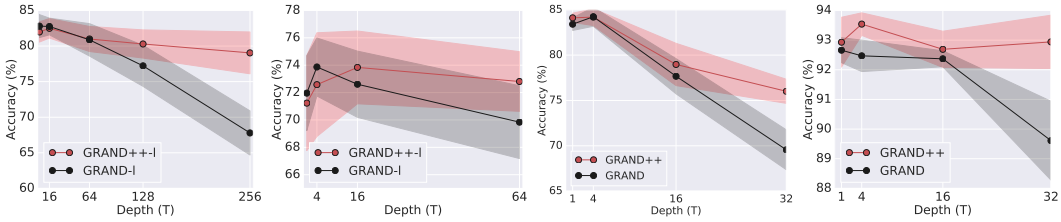
In this section, we compare the performance of GRAND++ with GRAND and several other popular GNNs on various graph node classification tasks. We aim to show the practical advantages of GRAND++ in learning with limited labeled data and using deep architectures. Without mentioning clearly, we use the same hyperparameters that that used for GRAND in [10] for GRAND++. We provide detailed descriptions of experimental settings and datasets that are omitted in the main text in Appendix D.1. For all experiments, we run 100 splits for each dataset with 20 random seeds for each split, which are conducted on a server with four NVIDIA RTX 3090 graphics cards.

We compare the performance of GRAND++ and its nonlinear and graph rewiring variants with several popular GNNs on various graph node classification benchmarks. Except for the integration time, which measures the implicit depth of GRAND and GRAND++, we adopt the experimental settings of GRAND in [10] for GRAND++ include numerical differential equation solvers. Following [10], we study seven graph node classification datasets, namely CORA, CiteSeer, PubMed, CoauthorCS, Computer, Photo, and ogb-arxiv; we describe these datasets in Appendix D.1.

5.1 GRAND++ IS MORE RESILIENT TO DEEP ARCHITECTURES

We first show that our introduced source term in (11) can improve the accuracy of GRAND-l when the architecture is deep, i.e., the integration time T in (5) is big. We denote GRAND-l with the source term as GRAND++-l. For each node classification task, we train all models using the same number of labeled nodes as in [10]. Figure 2 contrasts the performance of GRAND-l and GRAND++-l

with different depths, or T, on CORA, CiteSeer, Computer, and Photo datasets. We provide the detailed results on PubMed and CoauthorCS, together with more comparisons of GRAND++-l with GRAND-l and several other celebrated GNNs include GCN, GAT, and GraphSage in Appendix D.2. The results in Fig. 2 and Appendix D.2 confirm that GRAND-l suffers less from over-smoothing compared to GCN, GAT, and GraphSage. Moreover, GRAND++-l performs on par with GRAND-l when the depth (T) of the network is small, but GRAND++-l significantly outperforms GRAND-l when T is large. As T increases, the margin becomes wider, indicating that GRAND++-l can overcome over-smoothing much more effectively than GRAND-l. Note that we did not use uniform depth for GRAND-l and GRAND++-l on all datasets because the adaptive step-size ODE solver fails when T is large for some tasks.



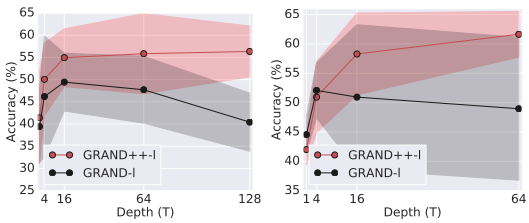
CORA CiteSeer Computer Photo

Figure 2: Test accuracy vs. the "depth" (T in (5)) of GRAND-l and GRAND++-l on the four graph node classification tasks. We see that GRAND++-l is much more resilient to deep architectures than GRAND-l.

Next, we compare GRAND-l and GRAND++-l on the ogb-arxiv node classification task, which is a much larger scale task than the above benchmarks. We train two models using labeling rates of 3.0% and 5.0%, respectively; the corresponding test accuracy for GRAND-l/GRAND++-l are 65.26%/66.64% and 67.42%/67.77%, respectively. GRAND++-l outperforms GRAND-l in both labeling rates.

5.2 GRAND++ IS MORE ACCURATE WITH LIMITED LABELED TRAINING DATA

Besides helping to overcome over-smoothing, our theory shows that the source term can boost the accuracy of GRAND-l with limited labeled training data. We verify this advantage of GRAND++-l here. Table 1 compares the accuracy of GRAND++-l with GRAND-l, GCN, GAT, GraphSage, and MoNet, trained with different numbers of labeled data. Here, we use the default depth T from [10] for both GRAND++-l and GRAND-l. We see that with few labeled data, in most tasks GRAND++-l is significantly more accurate than the other GNNs include GRAND-l, confirming our theoretical insight. For CoauthorCS task, both GRAND-l and GRAND++-l are worse than GCN and GraphSage. Moreover, increasing the depth of GRAND++-l can improve the classification accuracy with limited training data, but this is not the case for GRAND-l, see Fig. 3.



CORA CiteSeer

Figure 3: Classification accuracy of GRAND++-l and GRAND-l for CORA and CiteSeer tasks, where both models, with different depth (T), are train with 1 labeled node per class.

5.3 TIME-DEPENDENT ATTENTION AND GRAPH REWIRING

The previous experimental results show that GRAND++-l enhances the accuracy of GRAND-l in the cases when the labeled training data is limited and when the network is deep. Here, we explore the same strategy for GRAND-nl and GRAND-nl-rw; we name the corresponding models with the new source term GRAND++-nl and GRAND++-nl-rw, respectively. Table 2 compares GRAND-nl and GRAND-nl-rw with the corresponding model with a source term. We see that overall GRAND++-nl (GRAND++-nl-rw) outperforms GRAND-nl (GRAND-nl-rw) when the network is deep, i.e., T is big. We further study the low-labeling rate regimes in Appendix D.3.

Model	#per class	CORA	CiteSeer	PubMed	CoauthorCS	Computer	Photo
GRAND++-l (ours)	1	**54.94 ± 16.09**	**58.95 ± 9.59**	**65.94 ± 4.87**	58.36 ± 2.61	**68.95 ± 0.39**	**83.12 ± 0.78**
	2	**66.92 ± 10.04**	**64.98 ± 8.31**	**69.31 ± 4.87**	77.23 ± 2.08	74.23 ± 1.42	**83.71 ± 0.90**
	5	**77.80 ± 4.46**	**70.03 ± 3.63**	71.99 ± 1.91	85.69 ± 1.15	81.29 ± 0.61	88.33 ± 1.21
	10	**80.86 ± 2.99**	**72.34 ± 2.42**	75.13 ± 3.88	87.28 ± 1.14	81.36 ± 1.05	90.65 ± 1.19
	20	**82.95 ± 1.37**	73.53 ± 3.31	**79.16 ± 1.37**	90.24 ± 0.30	84.19 ± 0.72	**93.55 ± 0.38**
GRAND-l [10]	1	52.53 ± 16.40	50.06 ± 17.98	62.11 ± 10.58	59.15 ± 5.73	48.67 ± 1.66	81.25 ± 2.50
	2	64.82 ± 11.16	59.55 ± 10.89	69.00 ± 7.55	73.83 ± 5.58	74.77 ± 1.85	82.13 ± 3.27
	5	76.07 ± 5.08	68.37 ± 5.00	**73.98 ± 5.08**	85.29 ± 2.19	80.72 ± 1.09	88.27 ± 1.94
	10	80.25 ± 3.40	71.90 ± 7.66	**76.33 ± 3.41**	87.81 ± 1.36	82.42 ± 1.10	**90.98 ± 0.93**
	20	82.86 ± 2.39	73.02 ± 5.89	78.76 ± 1.69	91.03 ± 0.47	**84.54 ± 0.90**	93.53 ± 0.47
GCN [30]	1	47.72 ± 15.33	48.94 ± 10.24	58.61 ± 12.83	**65.22 ± 2.25**	49.46 ± 1.65	82.94 ± 2.17
	2	60.85 ± 14.01	58.06 ± 9.76	60.45 ± 16.20	**83.61 ± 1.49**	**76.90 ± 1.49**	83.61 ± 0.71
	5	73.86 ± 7.97	67.24 ± 4.19	68.69 ± 7.93	86.66 ± 0.43	**82.47 ± 0.97**	**88.86 ± 1.56**
	10	78.82 ± 5.38	72.18 ± 3.47	72.59 ± 3.19	88.60 ± 0.50	**82.53 ± 0.74**	90.41 ± 0.35
	20	82.07 ± 2.03	**74.21 ± 2.90**	76.89 ± 3.27	91.09 ± 0.35	82.94 ± 1.54	91.95 ± 0.11
GAT [52]	1	47.86 ± 15.38	50.31 ± 14.27	58.84 ± 12.81	51.13 ± 5.24	37.14 ± 7.81	73.58 ± 8.15
	2	58.30 ± 13.55	55.55 ± 9.19	60.24 ± 14.44	63.12 ± 6.09	65.07 ± 8.86	76.89 ± 4.89
	5	71.04 ± 5.74	67.37 ± 5.08	68.54 ± 5.75	71.65 ± 4.53	71.43 ± 7.34	83.01 ± 3.64
	10	76.31 ± 4.87	71.35 ± 4.92	72.44 ± 3.50	74.71 ± 3.35	76.04 ± 0.35	87.42 ± 2.38
	20	79.92 ± 2.28	73.22 ± 2.90	75.55 ± 4.11	79.95 ± 2.88	80.05 ± 1.81	89.38 ± 2.48
GraphSage [29]	1	43.04 ± 14.01	48.81 ± 11.45	55.53 ± 12.71	61.35 ± 1.35	27.65 ± 2.39	45.36 ± 7.13
	2	53.96 ± 12.18	54.39 ± 11.37	58.97 ± 12.65	76.51 ± 1.31	42.63 ± 4.29	51.93 ± 4.21
	5	68.14 ± 6.95	64.79 ± 5.16	66.07 ± 6.16	**89.06 ± 0.69**	64.83 ± 1.62	78.26 ± 1.93
	10	75.04 ± 5.03	68.90 ± 5.08	70.74 ± 3.11	**89.68 ± 0.39**	74.66 ± 1.29	84.38 ± 1.75
	20	80.04 ± 2.54	72.02 ± 2.82	74.55 ± 3.09	**91.33 ± 0.36**	79.98 ± 0.96	91.29 ± 0.67
MoNet [38]	1	47.72 ± 15.53	39.13 ± 11.37	56.47 ± 4.67	58.99 ± 5.17	23.78 ± 7.57	34.72 ± 8.18
	2	60.85 ± 14.01	48.52 ± 9.52	61.03 ± 6.93	76.57 ± 4.06	38.19 ± 3.72	43.03 ± 8.22
	5	73.86 ± 7.97	61.66 ± 6.61	67.92 ± 2.50	87.02 ± 1.67	59.38 ± 4.73	71.80 ± 5.02
	10	78.82 ± 5.38	68.08 ± 6.29	71.24 ± 1.54	88.76 ± 0.49	68.66 ± 3.30	78.66 ± 3.17
	20	82.07 ± 2.03	71.52 ± 4.11	76.49 ± 1.75	90.31 ± 0.41	73.66 ± 2.87	88.61 ± 1.18

Table 1: Classification accuracy of different GNNs trained with different number of labeled data per class (#per class) on six benchmark graph node classification tasks. The highest accuracy is highlighted in bold for each number of labeled data per class. (Unit: %)

Model	Depth (T)	GRAND-nl [10]	GRAND-nl-rw [10]	GRAND++-nl (ours)	GRAND++-nl-rw (ours)
CORA	1	**79.70 ± 1.88**	79.07 ± 3.05	79.24 ± 1.48	79.24 ± 1.48
	4	82.31 ± 0.91	82.47 ± 1.32	**82.64 ± 0.89**	82.23 ± 1.14
	16	82.11 ± 1.42	82.05 ± 1.31	**83.24 ± 0.20**	81.48 ± 1.07
	32	79.42 ± 0.64	81.01 ± 0.81	81.21 ± 0.37	**82.20 ± 1.15**
CiteSeer	1	71.84 ± 2.98	**71.84 ± 2.66**	70.45 ± 2.12	71.74 ± 1.37
	16	72.65 ± 2.42	73.06 ± 2.98	72.48 ± 1.10	**73.29 ± 1.37**
	64	70.29 ± 2.58	69.65 ± 2.50	72.64 ± 0.93	**73.38 ± 0.95**
	128	65.19 ± 6.77	65.45 ± 7.18	**74.24 ± 0.70**	74.23 ± 0.70
PubMed	1	77.93 ± 1.27	77.93 ± 1.26	**78.01 ± 0.68**	78.01 ± 0.68
	4	77.95 ± 1.28	78.02 ± 1.14	**78.41 ± 0.88**	78.17 ± 0.93
	16	76.51 ± 2.73	76.88 ± 2.57	**78.43 ± 0.78**	78.12 ± 0.87

Table 2: Classification accuracy of GRAND and GRAND++ variants of different depth trained 20 labeled data per class. The highest accuracy is highlighted in bold for each of the depths $T = 1, 4, 16, 32, 64$, and 128. We test T only up to 16 for PubMed and up to 32 for 32 since the neural ODE solver failed for larger T. (Unit: %)

6 Concluding Remarks

We propose GRAND++, which augments graph neural diffusion with a source term. We present some theory that connects the model to a random walk formulation on graphs. GRAND++ outperforms many existing GNNs for graph deep learning with very deep architectures and when the number of labeled data is limited. GRAND++ can be regarded as coupled ODE system in which each ODE has an external force term. As such, it is natural to consider if advanced techniques in accelerating training, test, and inference of neural ODEs can be leveraged to improve the efficiency and accuracy of GRAND++, in particular high-order neural ODEs [19, 57, 40]. It is interesting to note that the second-order neural ODE can be connected to the wave equation in the graph setting, which can automatically bypass over-smoothing. We leave studying the second-order neural ODE on graphs as future work.

REFERENCES

[1] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

[2] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in neural information processing systems*, pages 1993–2001, 2016.

[3] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.

[4] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[5] Mikhail Belkin and Partha Niyogi. Semi-supervised learning on Riemannian manifolds. *Machine learning*, 56(1-3):209–239, 2004.

[6] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[7] Antoni Buades, Bartomeu Coll, and Jean M. Morel. Neighborhood filters and PDE's. *Numerische Mathematik*, 105(1):1–34, 2006.

[8] Jeff Calder. The game theoretic p-Laplacian and semi-supervised learning with few labels. *Nonlinearity*, 32(1), 2018.

[9] Jeff Calder, Brendan Cook, Matthew Thorpe, and Dejan Slepcev. Poisson learning: Graph based semi-supervised learning at very low label rates. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1306–1316. PMLR, 13–18 Jul 2020.

[10] Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. GRAND: Graph neural diffusion. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1407–1418. PMLR, 18–24 Jul 2021.

[11] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. *Advances in neural information processing systems*, 32:4868–4879, 2019.

[12] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020.

[13] Ricky T. Q. 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*, volume 31. Curran Associates, Inc., 2018.

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

[15] Ronald R Coifman, Stephane Lafon, Ann B Lee, Mauro Maggioni, Boaz Nadler, Frederick Warner, and Steven W Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences*, 102(21):7426–7431, 2005.

[16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29:3844–3852, 2016.

[17] Xavier Desquesnes, Abderrahim Elmoataz, and Lézoray, Olivier. Eikonal equation adaptation on weighted graphs: fast geometric diffusion process for local and non-local image and data processing. *Journal of Mathematical Imaging and Vision*, 46(2):238–257, 2013.

[18] John R Dormand and Peter J Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

[19] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[20] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[21] Moshe Eliasof, Eldad Haber, and Eran Treister. PDE-GCN: Novel architectures for graph neural networks motivated by partial differential equations. *arXiv preprint arXiv:2108.01938*, 2021.

[22] Abderrahim Elmoataz, Olivier Lezoray, and Sébastien Bougleux. Nonlocal discrete regularization on weighted graphs: A framework for image and manifold processing. *IEEE Transactions on Image Processing*, 17(7):1047–1060, 2008.

[23] Francesco Farina and Emma Slade. Data efficiency in graph networks through equivariance. *arXiv preprint arXiv:2106.13786*, 2021.

[24] Mark Freidlin and Shuenn-Jyi Sheu. Diffusion processes on graphs: stochastic differential equations, large deviation principle. *Probability theory and related fields*, 116(2):181–220, 2000.

[25] Mark I. Freidlin and Alexander D. Wentzell. Diffusion Processes on Graphs and the Averaging Principle. *The Annals of Probability*, 21(4):2215 – 2245, 1993.

[26] Cristina Garcia-Cardona, Ekaterina Merkurjev, Andrea L. Bertozzi, Arjuna Flenner, and Allon G. Percus. Multiclass data segmentation using diffuse interface methods on graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1600–1613, 2014.

[27] Guy Gilboa and Stanley Osher. Nonlocal operators with applications to image processing. *Multiscale Modeling & Simulation*, 7(3):1005–1028, 2008.

[28] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

[29] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[30] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017.

[31] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *Advances in Neural Information Processing Systems*, 32:13354–13366, 2019.

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

[33] 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*, 2018.

[34] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. In *International Conference on Learning Representations*, 2019.

[35] Qi Liu, Maximilian Nickel, and Douwe Kiela. Hyperbolic graph neural networks. *Advances in Neural Information Processing Systems*, 32:8230–8241, 2019.

[36] François Lozes, Abderrahim Elmoataz, and Olivier Lézoray. Partial difference operators on weighted graphs for image processing on surfaces and point clouds. *IEEE Transactions on Image Processing*, 23(9):3896–3909, 2014.

[37] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3952–3963. Curran Associates, Inc., 2020.

[38] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, Los Alamitos, CA, USA, jul 2017. IEEE Computer Society.

[39] Federico Monti, Michael M Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3700–3710, 2017.

[40] Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Lió. On second order behaviour in augmented neural odes. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5911–5921. Curran Associates, Inc., 2020.

[41] Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

[42] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

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

[44] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987.

[45] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2110–2119, 2018.

[46] Mauricio Flores Rios, Jeff Calder, and Gilad Lerman. Algorithms for lp-based semi-supervised learning on graphs. *arXiv preprint arXiv:1901.05031*, 2019.

[47] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

[48] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[49] Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[50] Zuoqiang Shi, Stanley J. Osher, and Wei. Zhu. Weighted nonlocal Laplacian on interpolation from sparse data. *Journal of Scientific Computing*, 73(2-3):1164–1177, 2017.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[52] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[53] Fei Wang, Changshui Zhang, Helen C Shen, and Jingdong Wang. Semi-supervised classification using linear neighborhood propagation. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 160–167. IEEE, 2006.

[54] Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. Dissecting the diffusion process in linear graph convolutional networks. *arXiv preprint arXiv:2102.10739*, 2021.

[55] Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. Continuous graph neural networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10432–10441. PMLR, 13–18 Jul 2020.

[56] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, 2018.

[57] Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. ODE2VAE: deep generative second order odes with Bayesian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[58] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[59] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5171–5181, 2018.

[60] Lingxiao Zhao and Leman Akoglu. PairNorm: Tackling oversmoothing in GNNs. In *International Conference on Learning Representations*, 2020.

[61] 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.

[62] Dengyong Zhou and Bernhard Schölkopf. Regularization on discrete spaces. In *27th DAGM Conference on Pattern Recognition*, pages 361–368, 2005.

[63] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine learning*, pages 912–919, 2003.

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

A BACKGROUND ON GRAPH DIFFERENTIAL OPERATORS

Let $(\boldsymbol{X}, \boldsymbol{W})$ represent a graph where $\boldsymbol{X} = ([\boldsymbol{x}^{(1)}]^\top, \dots, [\boldsymbol{x}^{(n)}]^\top)^\top \in \mathbb{R}^{n\times d}$ is the matrix where each row $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ is a feature vector and $\boldsymbol{W} = (W_{ij})_{i,j=1}^n$ is a $n \times n$ matrix with W_{ij} representing the similarity (edge weight) between the ith and jth feature vector. We assume that we are dealing with an undirected graph, i.e., $W_{ij} = W_{ji}$. A $\mathbb{R}^{k_1}$-valued function on the nodes of the graph can be represented as a matrix $\boldsymbol{U} \in \mathbb{R}^{n\times k_1}$ by $\boldsymbol{U} = ([\boldsymbol{u}^{(1)}]^\top, \dots, [\boldsymbol{u}^{(n)}]^\top)^\top$ and we define the inner product

$$\langle \boldsymbol{U}, \boldsymbol{V} \rangle = \sum_{i=1}^n \boldsymbol{u}^{(i)} \cdot \boldsymbol{v}^{(i)}.$$

Similarly, a $\mathbb{R}^{k_2}$-valued function on the edges can be represented as a third-order tensor $\mathcal{U} \in \mathbb{R}^{n\times n\times k_2}$ which we write as

$$\mathcal{U} = \begin{pmatrix} \mathcal{U}^{(1,1)} & \dots & \mathcal{U}^{(1,n)} \\ \vdots & \ddots & \vdots \\ \mathcal{U}^{(n,1)} & \dots & \mathcal{U}^{(n,n)} \end{pmatrix}$$

and $\mathcal{U}^{(i,j)} \in \mathbb{R}^{k_2}$. On edge functions we use the inner product

$$\langle \mathcal{U}, \mathcal{V} \rangle = \frac{1}{2} \sum_{i,j=1}^n W_{ij} \mathcal{U}^{(i,j)} \cdot \mathcal{V}^{(i,j)}.$$

Multiplication between a matrix and an edge function is usually defined pointwise and we use the notation $[\boldsymbol{A} \odot \mathcal{U}]^{(i,j)} = A_{ij}\mathcal{U}^{(i,j)} \in \mathbb{R}^{k_2}$, for a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and an edge function $\mathcal{U} \in \mathbb{R}^{n\times n\times k_2}$, to make this clear. Similarly, pointwise multiplication between two matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ is defined by $[\boldsymbol{A} \odot \boldsymbol{B}]_{ij} = A_{ij}B_{ij} \in \mathbb{R}$. When a matrix is acting as a linear operator on a node function we use the usual matrix-vector notation and write $[\boldsymbol{A}\boldsymbol{U}]^{(i)} = \sum_{j=1}^n A_{ij}\boldsymbol{u}^{(j)} \in \mathbb{R}^{k_1}$ for a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and node function $\boldsymbol{U} = ([\boldsymbol{u}^{(1)}]^\top, \dots, [\boldsymbol{u}^{(n)}]^\top)^\top \in \mathbb{R}^{n\times k_1}$. In the sequel we will have $k_1 = k_2 = d$.

The gradient of a node-function $\boldsymbol{U} = ([\boldsymbol{u}^{(1)}]^\top, \dots, [\boldsymbol{u}^{(n)}]^\top)^\top \in \mathbb{R}^{n\times d}$ is defined as the edge-function $\nabla \boldsymbol{U} \in \mathbb{R}^{n\times n\times d}$ with $[\nabla \boldsymbol{U}]^{(i,j)} = \boldsymbol{u}^{(j)} - \boldsymbol{u}^{(i)} \in \mathbb{R}^d$. The divergence $\mathrm{div}\mathcal{V} = ([[\mathrm{div}\mathcal{V}]^{(1)}]^\top, \dots, [[\mathrm{div}\mathcal{V}]^{(n)}]^\top)^\top \in \mathbb{R}^{n\times d}$ of an edge-function $\mathcal{V} \in \mathbb{R}^{n\times n\times d}$ is defined as

$$[\mathrm{div}\mathcal{V}]^{(i)} = \sum_{j=1}^n W_{ij}\mathcal{V}^{(i,j)}$$

for all $i = 1, \dots, n$. For anti-symmetric edge functions, i.e. $\mathcal{V}^{(i,j)} = -\mathcal{V}^{(j,i)}$ for all i, j, we have that the divergence is the negative adjoint to the gradient, i.e.

$$\langle \mathrm{div}\mathcal{V}, \boldsymbol{U} \rangle = -\langle \mathcal{V}, \nabla \boldsymbol{U} \rangle.$$

B TECHNICAL PROOFS

Proof: [Proof of Proposition 1] Clearly

$$\mathbb{E}\left[\boldsymbol{B}^{(i)}(0)\right] = \boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i)}(0)$$

for all $i = 1, \dots, n$. Assume that

$$\mathbb{E}\left[\boldsymbol{B}^{(i)}(k)\right] = \boldsymbol{x}^{(i)}(\delta_t k)$$

for all $i = 1, \ldots, n$. Then,

$$
\begin{aligned}
&\mathbb{E}\left[\boldsymbol{B}^{(i)}(k+1)\right] \\
&= \sum_{j=1}^{n} \boldsymbol{x}^{(j)} \mathbb{P}\left(\boldsymbol{B}^{(i)}(k+1) = \boldsymbol{x}^{(j)}\right) \\
&= \sum_{j=1}^{n}\sum_{\ell=1}^{n} \boldsymbol{x}^{(j)} \mathbb{P}\left(\boldsymbol{B}^{(\ell)}(k) = \boldsymbol{x}^{(j)} | \boldsymbol{B}^{(i)}(1) = \boldsymbol{x}^{(\ell)}\right) \mathbb{P}\left(\boldsymbol{B}^{(i)}(1) = \boldsymbol{x}^{(\ell)}\right) \\
&= \sum_{j=1}^{n}\sum_{\ell=1}^{n} \boldsymbol{x}^{(j)} \left((1-\delta_t)\mathbb{1}_{i=\ell} + \frac{\delta_t W_{i\ell}}{d_i}\right) \mathbb{P}\left(\boldsymbol{B}^{(i)}(1) = \boldsymbol{x}^{(\ell)}\right) \\
&= (1-\delta_t)\sum_{j=1}^{n} \boldsymbol{x}^{(j)} \mathbb{P}\left(\boldsymbol{B}^{(i)}(1) = \boldsymbol{x}^{(\ell)}\right) + \frac{\delta_t}{d_i}\sum_{\ell=1}^{n} W_{i\ell} \sum_{j=1}^{n} \boldsymbol{x}^{(j)} \mathbb{P}\left(\boldsymbol{B}^{(\ell)}(k) = \boldsymbol{x}^{(j)}\right) \\
&= (1-\delta_t)\mathbb{E}\left[\boldsymbol{B}^{(i)}(k)\right] + \frac{\delta_t}{d_i}\sum_{\ell=1}^{n} W_{i\ell}\mathbb{E}\left[\boldsymbol{B}^{(\ell)}(k)\right] \\
&= (1-\delta_t)\boldsymbol{x}^{(i)}(\delta_t k) + \frac{\delta_t}{d_i}\sum_{\ell=1}^{n} W_{i\ell}\boldsymbol{x}^{(\ell)}(\delta_t k) \\
&= \boldsymbol{x}^{(i)}(\delta_t k) + \frac{\delta_t}{d_i}\sum_{\ell=1}^{n} W_{i\ell}W_{i\ell}\left(\boldsymbol{x}^{(\ell)}(\delta_t k) - \boldsymbol{x}^{(i)}(\delta_t k)\right) \\
&= \boldsymbol{x}^{(i)}(\delta_t k) - \delta_t\left[\boldsymbol{L}\boldsymbol{X}(\delta_t k)\right]^{(i)} \\
&= \boldsymbol{x}^{(i)}(\delta_t(k+1)),
\end{aligned}
$$

as required. □

Proof: [Proof of Proposition 2] Let $\boldsymbol{P} = \left(P_{ij}\right) \in \mathbb{R}^{n\times n}$ be the probability transition kernel, so

$$
P_{ij} = \begin{cases} 1-\delta_t & \text{if } i = j, \\ \frac{\delta_t W_{ij}}{d_i} & \text{if } i \neq j. \end{cases}
$$

We have

$$
\begin{aligned}
\sum_{i=1}^{n} \pi_i P_{ij} &= \sum_{i=1}^{n} \frac{d_i}{\sum_{k=1}^{n} d_k}\left((1-\delta_t\mathbb{1}_{i=j} + \frac{\delta_t W_{ij}}{d_i}\right) \\
&= \frac{d_j(1-\delta_t)}{\sum_{k=1}^{n} d_k} + \frac{\delta_t \sum_{i=1}^{n} W_{ij}}{\sum_{k=1}^{n} d_k} \\
&= \frac{d_j}{\sum_{k=1}^{n} d_k} \\
&= \pi_j,
\end{aligned}
$$

as required. □

Proof: [Proof of Proposition 3] The proof follows from a simple application of Propositions 1 and 2. Namely, for any $i \in \{1, \ldots, n\}$

$$
\boldsymbol{x}^{(i)}(k\delta_t) = \mathbb{E}\big[\boldsymbol{B}^{(i)}(k)\big] = \sum_{j=1}^{n} \boldsymbol{x}^{(j)} \mathbb{P}\big(\boldsymbol{B}^{(i)}(k) = \boldsymbol{x}^{(j)}\big) \to \sum_{j=1}^{n} \boldsymbol{x}^{(j)}\pi_j = \overline{\boldsymbol{x}}
$$

as $k \to \infty$. □

Proof: [Proof of Proposition 4] Let

$$
\boldsymbol{y}^{(i)}(k) = \mathbb{E}\left[\sum_{s=0}^{k} \frac{1}{d_i}\sum_{j\in\mathcal{I}}\left(\boldsymbol{x}^{(j)} - \widetilde{\boldsymbol{x}}\right)\mathbb{1}_{\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(i)}}\right].
$$

Notice that

$$
\begin{aligned}
&\mathbb{E}\left[\sum_{s=0}^{k} \mathbb{1}_{\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(i)}}\right] \\
&= \sum_{s=0}^{k} \mathbb{P}\left(\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(i)}\right) \\
&= \underbrace{\mathbb{P}\left(\boldsymbol{B}^{(j)}(0)=\boldsymbol{x}^{(i)}\right)}_{\delta_{ij}} + \sum_{s=1}^{k} \mathbb{P}\left(\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(i)}\right) \\
&= \delta_{ij} + \sum_{s=1}^{k}\sum_{\ell=1}^{n} \mathbb{P}\left(\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(i)} | \boldsymbol{B}^{(j)}(s-1)=\boldsymbol{x}^{(\ell)}\right) \mathbb{P}\left(\boldsymbol{B}^{(j)}(s-1)=\boldsymbol{x}^{(\ell)}\right) \\
&= \delta_{ij} + \sum_{s=1}^{k}\sum_{\ell=1}^{n} \left((1-\delta_t)\delta_{\ell i} + \frac{\delta_t W_{\ell i}}{d_\ell}\right) \mathbb{P}\left(\boldsymbol{B}^{(j)}(s-1)=\boldsymbol{x}^{(\ell)}\right) \\
&= \delta_{ij} + (1-\delta_t)\sum_{s=1}^{k} \mathbb{P}\left(\boldsymbol{B}^{(j)}(s-1)=\boldsymbol{x}^{(i)}\right) + \delta_t \sum_{\ell=1}^{n} \frac{W_{\ell i}}{d_\ell} \sum_{s=1}^{k} \mathbb{P}\left(\boldsymbol{B}^{(j)}(s-1)=\boldsymbol{x}^{(\ell)}\right) \\
&= \delta_{ij} + (1-\delta_t)\sum_{s=0}^{k-1} \mathbb{P}\left(\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(i)}\right) + \delta_t \sum_{\ell=1}^{n} \frac{W_{\ell i}}{d_\ell} \sum_{s=0}^{k-1} \mathbb{P}\left(\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(\ell)}\right) \\
&= \delta_{ij} + (1-\delta_t)\mathbb{E}\left[\sum_{s=0}^{k-1} \mathbb{1}_{\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(i)}}\right] + \delta_t \sum_{\ell=1}^{n} \frac{W_{\ell i}}{d_\ell} \mathbb{E}\left[\sum_{s=0}^{k-1} \mathbb{1}_{\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(\ell)}}\right].
\end{aligned}
$$

From the definition of $\boldsymbol{Y}$ and the above recursive relationship we have

$$
\begin{aligned}
\boldsymbol{y}^{(i)}(k) &= \frac{1}{d_i}\sum_{j\in\mathcal{I}} \left(\boldsymbol{x}^{(j)} - \widetilde{\boldsymbol{x}}\right) \mathbb{E}\left[\sum_{s=0}^{k} \mathbb{1}_{\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(i)}}\right] \\
&= \frac{1}{d_i}\sum_{j\in\mathcal{I}} \left(\boldsymbol{x}^{(j)} - \widetilde{\boldsymbol{x}}\right) \delta_{ij} + (1-\delta_t)\frac{1}{d_i}\sum_{j\in\mathcal{I}} \left(\boldsymbol{x}^{(j)} - \widetilde{\boldsymbol{x}}\right) \mathbb{E}\left[\sum_{s=0}^{k-1} \mathbb{1}_{\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(\ell)}}\right] \\
&\qquad + \frac{\delta_t}{d_i}\sum_{\ell=1}^{n} \frac{W_{\ell i}}{d_\ell} \sum_{j\in\mathcal{I}} \left(\boldsymbol{x}^{(j)} - \widetilde{\boldsymbol{x}}\right) \mathbb{E}\left[\sum_{s=0}^{k-1} \mathbb{1}_{\boldsymbol{B}^{(j)}(s)=\boldsymbol{x}^{(\ell)}}\right] \\
&= \frac{1}{d_i}\sum_{j\in\mathcal{I}} \left(\boldsymbol{x}^{(j)} - \widetilde{\boldsymbol{x}}\right) \delta_{ij} + (1-\delta_t)\boldsymbol{y}^{(i)}(k-1) + \frac{\delta_t}{d_i}\sum_{\ell=1}^{n} W_{\ell i}\boldsymbol{y}^{(\ell)}(k-1) \\
&= \boldsymbol{y}^{(i)}(k-1) + \frac{1}{d_i}\sum_{j\in\mathcal{I}} \left(\boldsymbol{x}^{(j)} - \widetilde{\boldsymbol{x}}\right) \delta_{ij} - \delta_t \left[\boldsymbol{L}\boldsymbol{Y}(k-1)\right]^{(i)}.
\end{aligned}
$$

Now we let

$$
\boldsymbol{w}^{(i)}(k) = d_i\left(\boldsymbol{z}^{(i)}(k\delta_t) - \boldsymbol{y}^{(i)}(k)\right)
$$

so that $\boldsymbol{W}$ satisfies

$$
\boldsymbol{w}^{(i)}(k) = \boldsymbol{w}^{(i)}(k-1) - \delta_t\left[\boldsymbol{L}\boldsymbol{W}(k-1)\right]^{(i)} = \left[\boldsymbol{P}\boldsymbol{W}(k-1)\right]^{(i)}.
$$

Hence, $\boldsymbol{W}(k) = \boldsymbol{P}^k\boldsymbol{W}(0)$. Since the stationary distribution of the random walk with transition kernel is π, we have $\lim_{k\to\infty}\boldsymbol{P}^k = \mathbb{1}\pi^\top$. Hence, as $k\to\infty$, $\boldsymbol{w}^{(i)}(k) \to \sum_{j=1}^{n}\pi_j\boldsymbol{w}^{(j)}(0) = 0$ since $\boldsymbol{w}^{(j)}(0) = 0$ by the choice in the initialisation of $\boldsymbol{Z}$. □

Proof: [Proof of Proposition 5] We prove the result by contradiction. Assume that there exists $\overline{\boldsymbol{z}}$ such that $\boldsymbol{z}^{(i)}(k\delta_t) \to \overline{\boldsymbol{z}}$ for all $i = 1, \ldots, n$ as $k \to \infty$. Then $\boldsymbol{LZ}(k\delta_t) \to 0$. Since we can write

$$\begin{aligned}\boldsymbol{z}^{(i)}(k\delta_t) - \boldsymbol{z}^{(j)}(k\delta_t) = \boldsymbol{z}^{(i)}((k-1)\delta_t) - \boldsymbol{z}^{(j)}((k-1)\delta_t) \\ - \delta_t \left([\boldsymbol{LZ}((k-1)\delta_t)]^{(i)} - [\boldsymbol{LZ}((k-1)\delta_t)]^{(j)} \right) \\ + \delta_t \sum_{\ell \in \mathcal{I}} \Big(\delta_{i\ell}(\boldsymbol{x}^{(i)} - \widetilde{\boldsymbol{x}}) - \delta_{j\ell}(\boldsymbol{x}^{(j)} - \widetilde{\boldsymbol{x}}) \Big),\end{aligned}$$

then taking the limit $k \to \infty$ implies

$$\sum_{\ell \in \mathcal{I}} \delta_{i\ell}(\boldsymbol{x}^{(i)} - \widetilde{\boldsymbol{x}}) = \sum_{\ell \in \mathcal{I}} \delta_{j\ell}(\boldsymbol{x}^{(j)} - \widetilde{\boldsymbol{x}})$$

for all i, j which is clearly not true. □

C Neural ODEs and Training Neural ODEs with Adjoint Method

Neural ODEs [13] are a class of continuous-depth (-time) neural networks that are particularly suitable for learning complex dynamics from irregularly sampled sequential data, see, e.g., [13, 49, 19, 37, 40]. Mathematically, a neural ODE is the first-order ODE:

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta), \tag{18}$$

where $f(\boldsymbol{h}(t), t, \theta) \in \mathbb{R}^d$ is specified by a neural network parameterised by θ, e.g., a two-layer feed-forward neural network. Starting from the input $\boldsymbol{h}(0)$, neural ODEs learn the representation and perform prediction by solving (18) from $t = 0$ to T using a numerical integrator with a given error tolerance, often with an adaptive step size solver (or adaptive solver for short) [18]. Solving (18) from $t = 0$ to T in a single pass with an adaptive solver requires evaluating $f(\boldsymbol{h}(t), t, \theta)$ at various timesteps, with computational complexity counted by the number of forward function evaluations (forward NFEs) [13].

The adjoint sensitivity method (or adjoint method) [44], is a memory-efficient method for training neural ODEs. We regard the output $\boldsymbol{h}(T)$ as the prediction and denote the loss between $\boldsymbol{h}(T)$ and the ground truth as $\mathcal{L}$. Let $\boldsymbol{a}(t) := \partial\mathcal{L}/\partial\boldsymbol{h}(t)$ be the adjoint state, then we have (see [13, 44] for details)

$$\frac{d\mathcal{L}}{d\theta} = \int_0^T \boldsymbol{a}(t)^\top \frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \theta} dt, \tag{19}$$

with $\boldsymbol{a}(t)$ satisfying the following adjoint ODE

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t)^\top \frac{\partial}{\partial \boldsymbol{h}} f(\boldsymbol{h}(t), t, \theta), \tag{20}$$

which is solved numerically from $t = T$ to 0 and also requires the evaluation of the right-hand side of (20) at various timestamps, and the backward NFEs measure the computational complexity.

D Experimental Details and More Experimental Results

D.1 Datasets and experimental settings

Graph node classification dataset. Following [10], we consider the largest connected component of seven graph node classification datasets, including CORA, CiteSeer, PubMed, coauthor graph CoauthorCS, and Amazon co-purchasing graphs Computer and Photo, and a large scale ogb-arxiv dataset. For completeness, we list the number of classes, the number of features, and the number of nodes and edges of each dataset in Table 3. More detailed information can be found in [10].

Dataset	Classes	Features	#Nodes	#Edges
CORA	7	1433	2485	5069
CiteSeer	6	3703	2120	3679
PubMed	3	500	19717	44324
CoauthorCS	15	6805	18333	81894
Computer	10	767	13381	245778
Photo	8	745	7487	119043
ogb-arxiv	40	128	169343	1166243

Table 3: Summary of the graph node classification datasets.

D.2 CLASSIFICATION ACCURACY OF GNNS WITH DIFFERENT DEPTHS

In this subsection, we provide detailed numbers that correspond to Fig. 1. We further compare GRAND++-l with several other GNN architectures, including GCN, GAT, and GraphSage, with different depths on a few benchmark datasets. Table 4 lists the classification accuracy of GRAND++, GRAND, and three benchmark GNNs with different depths on six graph node classification tasks. Again, we see that GRAND++ is better than the other GNN models when the networks are deep.

Model	depth	CORA	CiteSeer	PubMed	CoauthorCS	Computer	Photo
GRAND++-l (ours)	1	77.48 ± 1.43	71.23 ± 3.47	78.11 ± 1.47	90.42 ± 0.76	84.11 ± 0.51	92.93 ± 0.84
	4	81.98 ± 1.42	72.58 ± 3.79	79.20 ± 0.74	90.89 ± 0.36	84.19 ± 0.93	93.54 ± 0.38
	16	82.49 ± 1.37	73.84 ± 2.66	79.49 ± 0.84	90.24 ± 0.30	78.97 ± 2.33	92.69 ± 0.61
	32	82.48 ± 0.71	73.29 ± 1.29	79.81 ± 1.61	NA	76.01 ± 1.33	92.94 ± 0.90
	64	80.99 ± 1.76	72.81 ± 2.18	NA	NA	NA	NA
	128	80.29 ± 1.98	NA	NA	NA	NA	NA
	256	79.04 ± 2.94	NA	NA	NA	NA	NA
GRAND-l [10]	1	78.59 ± 1.17	71.96 ± 2.74	77.93 ± 1.26	90.79 ± 0.93	83.41 ± 0.69	92.66 ± 0.42
	4	82.80 ± 1.62	73.87 ± 2.12	78.71 ± 1.19	90.94 ± 0.21	84.23 ± 1.05	92.47 ± 0.53
	16	82.75 ± 1.17	72.61 ± 2.42	78.79 ± 0.93	87.66 ± 1.70	77.67 ± 1.94	92.37 ± 0.27
	32	82.19 ± 1.73	72.65 ± 3.15	78.70 ± 1.08	NA	69.56 ± 2.20	89.61 ± 1.33
	64	80.87 ± 2.28	69.84 ± 2.66	NA	NA	NA	NA
	128	77.22 ± 2.88	NA	NA	NA	NA	NA
	256	67.79 ± 3.10	NA	NA	NA	NA	NA
GCN [30]	1	76.92 ± 0.56	72.80 ± 1.69	72.78 ± 1.80	91.53 ± 0.45	81.44 ± 0.24	91.31 ± 0.19
	4	81.35 ± 1.27	70.54 ± 6.61	77.15 ± 3.00	87.84 ± 0.96	75.73 ± 1.02	90.11 ± 0.66
	16	19.70 ± 7.06	24.78 ± 1.45	41.36 ± 1.77	14.49 ± 0.91	12.86 ± 2.39	23.11 ± 1.76
	32	21.86 ± 6.09	24.23 ± 1.65	40.66 ± 1.86	12.14 ± 1.64	21.15 ± 13.10	24.30 ± 0.73
GAT [52]	1	72.49 ± 2.03	71.83 ± 1.53	77.24 ± 0.72	79.22 ± 0.60	73.97 ± 1.20	87.08 ± 0.37
	4	80.95 ± 2.28	72.31 ± 2.82	77.37 ± 1.32	78.05 ± 1.10	76.67 ± 2.79	87.95 ± 1.76
	16	29.14 ± 1.02	24.84 ± 1.45	39.21 ± 0.43	24.20 ± 2.22	37.07 ± 2.99	29.97 ± 3.68
	32	29.75 ± 1.57	24.83 ± 1.45	39.02 ± 0.12	22.73 ± 2.08	32.53 ± 3.09	25.57 ± 4.03
GraphSage [29]	1	73.47 ± 1.98	71.94 ± 1.45	72.42 ± 0.61	91.74 ± 0.26	75.95 ± 0.70	88.10 ± 0.87
	4	79.83 ± 2.43	50.00 ± 14.27	76.01 ± 2.35	87.94 ± 0.23	75.62 ± 2.85	90.68 ± 2.11
	16	25.52 ± 6.45	24.84 ± 1.45	37.55 ± 3.92	10.12 ± 2.21	22.79 ± 10.77	25.57 ± 3.31
	32	29.14 ± 1.02	28.38 ± 2.54	39.21 ± 4.39	7.91 ± 3.15	37.07 ± 13.22	20.09 ± 5.67

Table 4: Classification accuracy of different GNN models with different depths on six benchmark graph node classification tasks. NA: neural ODE solver failed. (Unit: %)

D.3 MORE RESULTS ON TIME-DEPENDENT ATTENTION AND GRAPH REWIRING

We further explore the effects of the source term for GRAND-nl and GRAND-nl-rw in the low-labeling rate regimes. Table 5 compares GRAND-nl and GRAND-nl-rw with the corresponding model with a source term. We see that GRAND-nl and GRAND-nl-rw are almost always worse than the vanilla GRAND-l, consistent with the results reported in [10]. GRAND++-nl and GRAND++-nl-rw cannot help learning at low labeling rates anymore. However, when the labeling rates are not low, GRAND++-nl or GRAND++-nl-rw can outperform GRAND-nl and GRAND-nl-rw, even outperform GRAND++.

D.4 CLASSIFICATION ACCURACY OF GNNS WITH FEWER NUMBER OF TRAINING DATA

Besides the results shown in Fig. 1, we further test the classification accuracy of more benchmark GNN architectures trained with fewer numbers of labeled data per class. Tables 6-8 list the classification accuracy, on the test set, of different benchmark GNN models when they are trained with different numbers of labeled nodes per class.

Model	#per class	GRAND-nl [10]	GRAND-nl-rw [10]	GRAND++-nl (ours)	GRAND++-nl-rw (ours)
CORA	1	50.55 ± 15.68	50.63 ± 17.71	48.89 ± 11.51	47.94 ± 11.06
	2	65.06 ± 9.35	61.24 ± 16.19	59.96 ± 7.90	58.25 ± 11.97
	5	76.93 ± 3.10	76.50 ± 3.91	74.01 ± 1.73	74.25 ± 1.99
	10	79.60 ± 2.69	79.38 ± 3.25	80.14 ± 0.69	80.18 ± 0.40
	20	82.22 ± 1.93	82.14 ± 2.49	83.24 ± 0.20	81.48 ± 1.07
CiteSeer	1	50.25 ± 17.66	50.20 ± 17.90	49.65 ± 5.45	53.10 ± 5.51
	2	59.87 ± 10.89	59.95 ± 10.48	59.16 ± 8.13	60.26 ± 5.10
	5	68.21 ± 5.08	68.05 ± 5.48	66.13 ± 2.09	67.81 ± 1.97
	10	71.88 ± 6.94	71.92 ± 7.34	68.84 ± 2.84	71.45 ± 1.64
	20	72.84 ± 6.61	72.72 ± 6.85	72.52 ± 1.24	73.87 ± 1.35
PubMed	1	66.97 ± 10.07	67.69 ± 7.89	63.85 ± 4.86	67.45 ± 3.88
	2	69.17 ± 2.46	69.42 ± 2.13	66.98 ± 5.30	69.11 ± 1.80
	5	72.56 ± 3.36	72.68 ± 2.52	71.49 ± 1.53	72.05 ± 3.67
	10	76.03 ± 3.73	75.32 ± 3.45	74.94 ± 2.15	75.09 ± 2.88
	20	78.55 ± 1.59	78.30 ± 1.43	78.41 ± 0.99	79.44 ± 0.56

Table 5: Classification accuracy of the variants of GRAND and GRAND++ models trained with different numbers of labeled data per class (#per class) on graph node classification tasks. (Unit: %)

#labeled nodes per class	1	2	5	10	20
GCN [30]	47.72 ± 15.53	60.85 ± 14.01	73.86 ± 7.97	78.82 ± 5.38	82.07 ± 2.03
GAT [52]	47.86 ± 15.38	58.30 ± 13.55	71.04 ± 5.74	76.31 ± 4.87	79.92 ± 2.28
GraphSage [29]	43.04 ± 14.01	53.96 ± 12.18	68.14 ± 6.95	75.04 ± 5.03	80.04 ± 2.54
MoNet [38]	47.72 ± 15.53	60.85 ± 14.01	73.86 ± 7.97	78.82 ± 5.38	82.07 ± 2.03
Lanczos [34]	47.41 ± 11.82	60.94 ± 4.00	74.28 ± 3.07	76.12 ± 0.93	79.85 ± 1.82
AdaLanczos [34]	48.23 ± 11.82	61.46 ± 4.96	74.24 ± 3.25	77.61 ± 1.36	81.03 ± 1.56
GCNN [55]	43.31 ± 11.95	60.28 ± 12.89	72.75 ± 4.21	78.92 ± 1.32	81.89 ± 1.12
GRAND-l [10]	52.53 ± 16.40	64.82 ± 11.16	76.07 ± 5.08	80.25 ± 3.40	82.86 ± 2.39
GRAND-nl [10]	40.97 ± 14.87	50.59 ± 13.25	65.13 ± 9.14	72.55 ± 6.65	77.76 ± 4.21
GRAND-nl-rw (gdc) [10]	52.68 ± 12.48	65.54 ± 10.01	74.94 ± 7.04	80.64 ± 6.19	82.47 ± 1.93
GRAND-nl-rw (two-hop) [10]	53.79 ± 17.72	64.50 ± 11.88	74.33 ± 6.28	79.61 ± 4.47	82.37 ± 1.98

Table 6: Classification accuracy of different GNNs trained with different numbers of labeled nodes per class. Dataset: CORA.

#labeled nodes per class	1	2	5	10	20
GCN [30]	48.94 ± 10.24	58.06 ± 9.76	67.24 ± 4.19	72.18 ± 3.47	74.21 ± 2.90
GAT [52]	50.31 ± 14.27	55.55 ± 9.19	67.37 ± 5.08	71.35 ± 4.92	73.22 ± 2.90
GraphSage [29]	48.81 ± 11.45	54.39 ± 11.37	64.79 ± 5.16	68.90 ± 5.08	72.02 ± 2.82
MoNet [38]	39.13 ± 11.37	48.52 ± 9.52	61.66 ± 6.61	68.08 ± 6.29	71.52 ± 4.11
Lanczos [34]	49.16 ± 3.63	57.65 ± 7.60	66.72 ± 9.38	71.01 ± 4.90	72.14 ± 2.00
AdaLanczos [34]	50.32 ± 7.42	58.35 ± 7.97	67.39 ± 8.20	72.15 ± 4.85	74.33 ± 2.83
GCNN [55]	40.58 ± 15.32	51.71 ± 13.87	63.16 ± 12.26	67.06 ± 5.65	69.84 ± 1.77
GRAND-l [10]	50.06 ± 17.98	59.55 ± 10.89	68.37 ± 5.00	71.90 ± 7.66	73.02 ± 5.89
GRAND-nl [10]	49.96 ± 18.62	59.57 ± 11.03	68.21 ± 7.08	71.88 ± 6.94	72.84 ± 6.61
GRAND-nl-rw (gdc) [10]	50.35 ± 17.74	59.98 ± 10.32	68.39 ± 5.81	71.83 ± 7.26	72.81 ± 6.94
GRAND-nl-rw (two-hop) [10]	50.20 ± 17.90	59.95 ± 10.48	68.05 ± 5.49	71.92 ± 7.34	72.72 ± 6.85

Table 7: Classification accuracy of different GNNs trained with different numbers of labeled nodes per class. Dataset: CiteSeer.

#labeled nodes per class	1	2	5	10	20
GCN [30]	58.61 ± 12.83	60.45 ± 16.20	68.69 ± 7.93	72.59 ± 3.19	76.89 ± 3.27
GAT [52]	58.84 ± 12.81	60.24 ± 14.44	68.54 ± 5.75	72.44 ± 3.50	75.55 ± 4.11
GraphSage [29]	55.53 ± 12.71	58.97 ± 12.65	66.07 ± 6.16	70.74 ± 3.11	74.55 ± 3.09
MoNet [38]	56.47 ± 4.67	61.03 ± 6.93	67.92 ± 2.50	71.24 ± 1.54	76.49 ± 1.75
Lanczos [34]	60.12 ± 6.37	63.65 ± 6.97	70.61 ± 4.50	73.01 ± 3.27	78.35 ± 1.84
AdaLanczos [34]	61.07 ± 5.16	64.11 ± 6.88	69.05 ± 3.00	72.79 ± 2.74	78.10 ± 1.91
GCNN [55]	60.78 ± 20.64	65.14 ± 19.45	72.72 ± 10.82	76.47 ± 6.03	79.24 ± 3.45
GRAND-l [10]	62.11 ± 10.58	69.00 ± 7.55	73.98 ± 5.08	76.33 ± 3.41	78.76 ± 1.69
GRAND-nl [10]	61.75 ± 11.12	69.16 ± 8.46	72.35 ± 5.35	76.03 ± 3.72	78.55 ± 1.59
GRAND-nl-rw (gdc) [10]	61.70 ± 10.74	69.42 ± 8.21	72.39 ± 5.25	75.32 ± 3.45	78.30 ± 1.43
GRAND-nl-rw (two-hop) [10]	61.65 ± 12.09	68.49 ± 8.99	72.68 ± 5.92	75.72 ± 3.50	78.77 ± 1.88

Table 8: Classification accuracy of different GNNs trained with different numbers of labeled nodes per class. Dataset: PubMed.