GRAND++: GRAPH NEURAL DIFFUSION WITH A SOURCE TERM

Matthew Thorpe*, Hedi Xia*, Tan Nguyen*, Thomas Strohmer
Andrea L. Bertozzi², Stanley J. Osher² & Bao Wang*

¹Department of Mathematics, University of Manchester, Manchester M13 9PL, UK
²Department of Mathematics, UCLA, Los Angeles, CA, 90095, USA
³Department of Mathematics, UC Davis, Davis, CA 95616, USA
⁴Department of Mathematics and Scientific Computing and Imaging (SCI) Institute
University of Utah, Salt Lake City, UT, 84102, USA

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) [54]. These graph deep networks have achieved success in many applications, including computational physics and computational chemistry [20, 28, 3], recommender systems [41, 62], and social networks [63, 47]. Hyperbolic GNNs have also been proposed to enable certain kinds of data embedding with much smaller distortion [11, 37]. 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 [35, 44, 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 [33], Jumping Knowledge [60], DropEdge [49], PairNorm [64], 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., [36, 9, 23]; one question is can we develop new GNN architectures to improve the performance of graph deep learning in low-labeling rate regimes?

![Graph Learning Accuracy](image1)

Figure 1: Test accuracy of GCN, GAT, and GraphSage vs. the number of labeled nodes per class. All networks have 2 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. 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. 5 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, 38], and semi-supervised graph nodes classification [67, 65]. 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 [43]. Moreover, insights from the diffusion process on graphs have been used to improve the performance of GNNs, see, e.g., [2, 35, 31, 57].

Leveraging neural ordinary differential equations (ODEs) [13], continuous-depth GNNs have been proposed, see, e.g., [43, 58, 68]. 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 [46] 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 [67, 65, 50], image processing [7, 27], etc. Direct application of Laplace learning with Gaussian weights [5] or locally linear embedding weights [50] for the above tasks may cause inference inconsistency when only a limited number of graph nodes are labeled, resulting in poor performance. Several algorithms address the inference inconsistency at low labeling rate. They include up-weighting the weights of the labeled data [52] and the p-Laplacian [8, 48, 66]. 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. Compared to Laplace learning, Poisson learning adds Green’s function to the label of each labeled node and then performs label propagation to predict the label for unlabeled graph nodes. GRAND and GRAND++ both learn graph node representations and perform prediction by activating the node representations, which are fundamentally different from Laplace and Poisson learning.

1.3 Notation
We denote scalars by lower- or upper-case letters and vectors by lower- and upper-case boldface letters, respectively. For a matrix $A$, we denote its transpose as $A^\top$ and its Hadamard product with another matrix $B$ as $A \odot B$, i.e., the entrywise multiplication of $A$ and $B$. We write the set $\{1, 2, \cdots, n\}$ as $[n]$. We denote the probability and expectation of a given random variable $x$ as $\mathbb{P}(x)$ and $\mathbb{E}[x]$, 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 Secs. 3 and 4 we briefly review GRAND and present a random walk interpretation of GRAND, respectively. Leveraging the random walk viewpoint of GRAND, we propose GRAND++ for deep graph learning with theoretical guarantees in Sec. 5. We verify the efficacy of GRAND++ in Sec. 6. Technical proofs and more validations are provided in the appendix.

2 Background

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

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

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

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

$$\frac{\partial X(t)}{\partial t} = \text{div}(G(X(t), t) \odot \nabla X(t)) = -LX(t), \tag{2}$$

where $L = I - D^{-1}W := I - A (A := A(X))$ is the random walk Laplacian and $D$ is diagonal with $D_{ii} = d_i$. See [14, 23, 24] for more about random walk Laplacian and diffusion on graphs.

Graph neural networks. Applying forward Euler discretization, with step size $\delta_t < 1$, of (2) gives

$$X(k\delta_t) = X((k-1)\delta_t) - \delta_t LX((k-1)\delta_t) := \tilde{L}X((k-1)\delta_t), \quad \text{for} \quad k = 1, 2, \cdots, K, \tag{3}$$
\( T = K \delta_t, \) and \( X(0) = X. \) Note that the matrix \( \hat{L} \) is the discretization of the diffusion operator, which is a special low-pass filter. Equation (3) is a prototype for motivating GNNs: by introducing weights \( W^{(k)} \in \mathbb{R}^{d\times d} \) and a nonlinearity \( \sigma, \) e.g., \( \text{ReLU}, \) into (3), we have
\[
X((k+1)\delta_t) = \sigma(\hat{L}X(k\delta_t)W^{(k)}).
\]
(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. 5 for a detailed analysis.

3 A Brief Review of GRAND

GRAND is a new continuous-depth GNN proposed in [10]. It integrates a learnable encoder function \( \phi \) and a learnable decoder function \( \psi \) with the neural network parameterized graph diffusion process, resulting in the prediction \( Y = \psi(X(T)), \) where \( X(T) \) is computed as
\[
X(T) = X(0) + \int_0^T \frac{\partial X(t)}{\partial t} dt, \text{ with } X(0) = \phi(X),
\]
(5)
where \( \frac{\partial X(t)}{\partial t} \) 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 \( G \) is only dependent on the initial node features, we can rewrite (1) as
\[
\frac{\partial X(t)}{\partial t} = (A_X - I)X(t),
\]
(6)
GRAND models the diffusivity \( A_X \) in (6) by the multi-head self-attention mechanism; potential choices of the attention function include the ones proposed in [53, 54]. More precisely, in GRAND \( A_X(x) = \frac{1}{h} \sum_{l=1}^h A^l(X) \) with \( h \) being the number of heads and the attention matrix \( A^l(X) = (a^l(x_i, x_j)), \) for \( l = 1, \ldots, h, \) is computed as follows:
\[
a^l(x_i, x_j) = \frac{\exp(\text{LeakyReLU}(a^l\top[W^l x_i \| W^l x_j]))}{\sum_{k \in N_i} \exp(\text{LeakyReLU}(a^l\top[W^l x_i \| W^l x_k]))},
\]
(7)
where \( W^l \) and \( a^l \) are learned, \( \| \) is the concatenation operator, and \( N_i \) is the index set of the nodes that are connected to the \( i^{th} \) node in the graph. GRAND with the attention in (7) is called GRAND-l, that is, GRAND-l is a special case of GRAND when the diffusivity is dependent only on the initial graph node features. 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 [46, 13].

4 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 \( \{B^{(i)}(k)\}_{k \in \mathbb{N}} \) be the random walk on \( \{x^{(j)}(0)\}_{j=1} \) defined by, for \( \delta_t \in [0, 1], \)
\[
B^{(i)}(0) = x^{(i)}(0),
\]
\[
\mathbb{P}(B^{(i)}(k+1) = x^{(j)}(0) | B^{(i)}(k) = x^{(j)}(0)) = \begin{cases} 1 - \delta_t & \text{if } \ell = j \\ \frac{\delta_t W_{i\ell}}{d_i} & \text{if } \ell \neq j \end{cases}
\]
(8)
where \( d_i = \sum_{\ell=1}^n W_{i\ell} \) (assume \( W_{i\ell} = 0 \) for all \( \ell \). Proposition 1 below is well-known, see [67]. We provide the proof of Proposition 1 and all the subsequent theoretical results in Appendix B.

Proposition 1 Let \( X \) solve (3) and \( B^{(i)} \) be the random walk determined by (8) where \( \delta_t \in [0, 1]. \) Then
\[
x^{(i)}(\delta_t k) = \mathbb{E}[B^{(i)}(k)].
\]
Proposition 2 below gives the stationary distribution of the random walk \( \{ \mathbf{B}^{(i)}(k) \}_{k \in \mathbb{N}} \).

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

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

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

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 \( G = (\mathbf{X}, \mathbf{W}) \) is connected. Then for all \( i = 1, \ldots, n \), we have

\[
x^{(i)}(k\delta_i) \to \bar{x} := \sum_{j=1}^{n} x^{(j)}(0)\pi_j, \quad \text{as} \quad k \to \infty.
\]

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 \( \mathbf{X}(t) \) dependence back into \( G \) in (1) and (2) or into the operator \( \mathbf{A} \) 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.

## 5 GRAND++: Graph Neural Diffusion with A Source Term

### 5.1 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 (we use the variable \( z \) for GRAND++-related dynamics and \( x \) for GRAND dynamics)

\[
\frac{\partial x^{(i)}(t)}{\partial t} = \text{div} \left( \mathbf{G}(\mathbf{Z}(t), t) \circ \nabla \mathbf{Z}(t)^{(i)} \right) + \sum_{j \in \mathcal{I}} \delta_{ij} C_j
\]

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) follows the approximate dynamics when \( t \gg 1 \)

\[
\frac{\partial x^{(i)}(t)}{\partial t} = -[\mathbf{LX}(t)]^{(i)} = -\bar{x}^{(i)}(t) + \frac{1}{d_i} \sum_{j = 1}^{n} W_{ij} x^{(j)}(t) \approx 0.
\]

For \( i \in \mathcal{I} \), it transpires that choosing \( C_i = x^{(i)} - \bar{x} \) (where \( \bar{x} \) is defined below) gives rise to a random walk interpretation that allows us to prove that the oversmoothing seen in the GRAND model is avoided.

One can in fact choose \( \bar{x} \) with a certain degree of freedom. If we initialise \( \mathbf{X}(0) = \mathbf{X} \) then we obtain \( \bar{x} = \sum_{i = 1}^{n} x^{(j)} \pi_j \) (as is usual in the GRAND model). 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 \( \mathbf{X}(0) = \mathbf{X} \). Through connections with random walks we, in the next subsection, motivate an alternative initialisation

\[
\sum_{i = 1}^{n} x^{(i)}(0) = \sum_{i \in \mathcal{I}} \frac{x^{(i)} - \bar{x}}{d_i}, \quad \text{where} \quad \bar{x} = \frac{1}{|\mathcal{I}|} \sum_{j \in \mathcal{I}} x^{(j)}
\]
with the dynamics
\[
\frac{\partial z^{(i)}(t)}{\partial t} = \text{div} \left[ G(Z(t), t) \odot \nabla Z(t)^{(i)} \right] + \sum_{j \in I} \delta_{ij} \left( x^{(i)} - \hat{x} \right).
\]
(12)

For example, we could choose
\[
z^{(i)}(0) = \begin{cases} \frac{1}{d_i} (x^{(i)} - \hat{x}) & \text{if } i \in I \\ 0 & \text{otherwise,} \end{cases} \quad \text{or } z^{(i)}(0) = x^{(i)} - c,
\]
where \(c = \frac{1}{n} \left( \sum_{i=1}^{n} x^{(i)} - \sum_{j \in I} x^{(j)} - \hat{x} \right) \) is chosen such that (11) 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. 5.2. To summarize, the GRAND++ model in (12), with initial condition satisfying (11), 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. 4 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. 6 suggest the formal motivation holds and we are able to design deep GNNs.

5.2 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
\[
\frac{\partial z^{(i)}(t)}{\partial t} = - [LZ(t)]^{(i)} + \sum_{j \in I} \delta_{ij} \left( x^{(i)} - \hat{x} \right)
\]
(13)
where the initial condition satisfies (11). Using the forward Euler discretisation of the above dynamics we have
\[
z^{(i)}(k \delta_t) = z^{(i)}(k \delta_t - 1) - \delta_t [LZ(\delta_t)]^{(i)} + \delta_t \sum_{j \in I} \delta_{ij} \left( x^{(i)} - \hat{x} \right),
\]
(14)
for \(k = 1, 2, \ldots, K\) where again \(T = K \delta_t\).

We use the same random walk as that introduced in Sec. 4 i.e. the random walk defined by (8), but we will now only consider random walks that are initialised on the nodes indexed by \(I\).

**Proposition 4** Let \(Z\) solve (14) with the initial condition satisfying (11), and let \(B^{(i)}\) be the random walk determined by (8). Then,
\[
\left| z^{(i)}(k \delta_t) - \mathbb{E} \left[ \sum_{s=0}^{k} \frac{1}{d_i} \sum_{j \in I} \left( x^{(j)} - \hat{x} \right) 1_{B^{(j)}(s) = x^{(i)}} \right] \right| \rightarrow 0 \quad \text{as } k \rightarrow \infty.
\]

**Remark 1** In the limit \(k \rightarrow \infty\) the term
\[
\mathbb{E} \left[ \sum_{s=0}^{k} \frac{1}{d_i} \sum_{j \in I} x^{(j)} 1_{B^{(j)}(s) = x^{(i)}} \right]
\]
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
\[
\mathbb{E} \left[ \frac{1}{d_i} \sum_{j \in I} x^{(j)} 1_{B^{(j)}(k) = x^{(i)}} \right] = \frac{1}{d_i} \sum_{j \in I} x^{(j)} P(B^{(j)}(k) = x^{(j)}) \approx \frac{\pi_i}{d_i} \sum_{j \in I} x^{(j)}
\]
(15)
Proposition 5 guarantees GRAND++ is less likely to suffer from over-smoothing than will not become the same across graph nodes under the GRAND++ dynamics. Therefore we can understand \( \mathbb{E} \left[ \frac{1}{d_i} \sum_{j \in I} \hat{x} \mathbb{I}_{B^{(j)}(k) = \varpi^{(i)}} \right] \) as the long time behaviour of \( \mathbb{E} \left[ \frac{1}{d_i} \sum_{j \in I} x^{(j)} \mathbb{I}_{B^{(j)}(k) = \varpi^{(i)}} \right] \) as a function of \( k \to \infty \). That is, the node features will not converge to a constant vector driven by the GRAND++, as shown in Proposition 5 below, to hold. This is due to the asymptotic well-posedness of the dual random walk in low labeling rates \([9]\).

**Remark 2** The random walk interpretation

\[
\mathbb{E} \left[ \frac{1}{d_i} \sum_{j \in I} \left( x^{(j)} - \hat{x} \right) \mathbb{I}_{B^{(j)} = \varpi^{(i)}} \right] 
\]

can be considered to be dual to the random walk interpretation in Sec. 4. Now we release the random walkers from nodes indexed by \( I \) 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 \( I \). Indeed, if \( |I| \) is fixed whilst one takes the number of feature vectors \( n \to \infty \) we will 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 \( G = (X, W) \) is connected. Then \( z^{(i)}(k\delta_t) \) that was defined in \( (14) \) does not converge to a constant vector as a function of \( i \) as \( k \to \infty \). That is, the node features will not become the same across graph nodes under the GRAND++ dynamics.

**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 \( t \to \infty \). The limit is beyond the scope of the paper but we have seen one characterisation in Proposition 4. By construction we have \( \partial z^{(i)}(t)/\partial t \approx 0 \) for \( i \in I \) so one should expect that the deep layer limit is (close to) a smooth interpolation of the feature vectors labeled by \( I \).

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.

# 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 ogbn-arxiv; we describe these datasets in Appendix D.1.
6.1 GRAND++ is more resilient to deep architectures

We first show that our introduced source term in (12) 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++. 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 including GCN, GAT, and GraphSage in Table 5 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.

Next, we compare GRAND-l and GRAND++-l on the ogbn-arxiv node classification task, which is a large-scale benchmark. 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 88.35%/89.89%, respectively. GRAND++-l outperforms GRAND-l in both labeling rates. We further compare GRAND and GRAND++ with different depth on the ogbn-arxiv task in Appendix D.6.

6.2 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 low-labeling rates. 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 slightly tune $T$ for GRAND++ based on the optimal value for GRAND, see Table 4 in the Appendix for their values. 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. We perform the t-test in Appendix D.5 to confirm the statistical significance of the accuracy gain of GRAND++ over GRAND in Table 1.

6.3 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 outperforms 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 [1,3].

<table>
<thead>
<tr>
<th>Model</th>
<th>#per class</th>
<th>CORA</th>
<th>CiteSeer</th>
<th>PubMed</th>
<th>CoauthorCS</th>
<th>Computer</th>
<th>Photo</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAND++-l</td>
<td>1</td>
<td>54.94 ± 16.09</td>
<td>58.95 ± 9.59</td>
<td>65.94 ± 4.87</td>
<td>60.30 ± 1.50</td>
<td>67.65 ± 0.37</td>
<td>83.12 ± 0.78</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>66.92 ± 10.04</td>
<td>64.98 ± 8.31</td>
<td>69.31 ± 4.87</td>
<td>76.37 ± 1.85</td>
<td>83.71 ± 0.90</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>77.80 ± 4.46</td>
<td>70.03 ± 3.63</td>
<td>71.99 ± 1.91</td>
<td>84.83 ± 0.84</td>
<td>82.64 ± 0.56</td>
<td>88.33 ± 1.21</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>80.86 ± 2.99</td>
<td>72.34 ± 2.42</td>
<td>75.13 ± 3.88</td>
<td>86.94 ± 0.46</td>
<td>82.99 ± 0.81</td>
<td>90.65 ± 1.19</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>82.95 ± 1.37</td>
<td>73.53 ± 3.31</td>
<td>78.53 ± 0.59</td>
<td>90.80 ± 0.34</td>
<td>93.55 ± 0.38</td>
<td></td>
</tr>
</tbody>
</table>

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. These results show that GRAND++ is more effective in learning with low-labeling rates than GRAND. (Unit: %)

7 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, 61, 42, 59] and noise injection [55]. 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.
8 ACKNOWLEDGEMENT

This material is based on research sponsored by NSF grants DMS-1924935, DMS-1952339, DMS-2027248 and NSF CCF-1934568, DOE grant DE-SC0021142, and ONR grant N00014-18-1-2527 and the MURI grant N00014-20-1-2787. MT would like to thank the Isaac Newton Institute for Mathematical Sciences for support and hospitality during the programme *Mathematics of Deep Learning* when work on this paper was undertaken (EPSRC grant number EP/R014604/1) and acknowledge support from the European Union Horizon 2020 research and innovation programmes under the Marie Skłodowska-Curie grant agreement No. 777826 (NoMADS). MT also holds a Turing Fellowship at the Alan Turing Institute.

REFERENCES


A  BACKGROUND ON GRAPH DIFFERENTIAL OPERATORS

Let \((X, W)\) represent a graph where \(X = ([x^{(1)}]^T, \ldots, [x^{(n)}]^T)^T \in \mathbb{R}^{n \times d}\) is the matrix where each row \(x^{(i)} \in \mathbb{R}^d\) is a feature vector and \(W = (W_{ij})_{n \times n}^1\) is a \(n \times n\) matrix with \(W_{ij}\) representing the similarity (edge weight) between the \(i\)th and \(j\)th 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 \(U \in \mathbb{R}^{n \times k_1}\) by \(U = ([u^{(1)}]^T, \ldots, [u^{(n)}]^T)^T\) and we define the inner product

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

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

\[
U = \begin{pmatrix}
U^{(1,1)} & \cdots & U^{(1,n)} \\
\vdots & \ddots & \vdots \\
U^{(n,1)} & \cdots & U^{(n,n)}
\end{pmatrix}
\]

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

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

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

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

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

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

\[
\langle \text{div} V, U \rangle = -\langle V, \nabla U \rangle.
\]

B  TECHNICAL PROOFS

Proof: [Proof of Proposition 1] For notational convenience let us assume that \(x^{(i)}(0) = x^{(i)}\). Clearly

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

for all \(i = 1, \ldots, n\). Assume that

\[
E \left[ B^{(i)}(k) \right] = x^{(i)}(\delta_i k)
\]
for all \( i = 1, \ldots, n \). Then,

\[
\mathbb{E} \left[ B^{(i)}(k + 1) \right] = \sum_{j=1}^{n} x^{(j)} \mathbb{P} \left( B^{(i)}(k + 1) = x^{(j)} \right)
\]

\[
= \sum_{j=1}^{n} \sum_{l=1}^{n} x^{(j)} \mathbb{P} \left( B^{(i)}(k) = x^{(j)} | B^{(i)}(1) = x^{(l)} \right) \mathbb{P} \left( B^{(i)}(1) = x^{(l)} \right)
\]

\[
= \sum_{j=1}^{n} \sum_{l=1}^{n} x^{(j)} \left( 1 - \delta_{i} \right) \mathbb{I}_{i=l} + \frac{\delta_{i} W_{il}}{d_{i}} \mathbb{P} \left( B^{(i)}(1) = x^{(l)} \right)
\]

\[
= (1 - \delta_{i}) \sum_{j=1}^{n} x^{(j)} \mathbb{P} \left( B^{(i)}(1) = x^{(l)} \right) + \frac{\delta_{i}}{d_{i}} \sum_{l=1}^{n} W_{il} \sum_{j=1}^{n} x^{(j)} \mathbb{P} \left( B^{(i)}(k) = x^{(j)} \right)
\]

\[
= (1 - \delta_{i}) \mathbb{E} \left[ B^{(i)}(k) \right] + \frac{\delta_{i}}{d_{i}} \sum_{l=1}^{n} W_{il} \mathbb{E} \left[ B^{(l)}(k) \right]
\]

\[
= (1 - \delta_{i}) x^{(i)}(\delta_{i}k) + \frac{\delta_{i}}{d_{i}} \sum_{l=1}^{n} W_{il} x^{(l)}(\delta_{i}k)
\]

\[
= x^{(i)}(\delta_{i}k) + \frac{\delta_{i}}{d_{i}} \sum_{l=1}^{n} W_{il} x^{(l)}(\delta_{i}k) - x^{(i)}(\delta_{i}k)
\]

\[
= x^{(i)}(\delta_{i}k) - \delta_{i} LX(\delta_{i}k)^{(i)}
\]

\[
= x^{(i)}(\delta_{i}(k + 1)),
\]

as required. \( \square \)

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

\[
P_{ij} = \begin{cases} 
1 - \delta_{i} & \text{if } i = j, \\
\frac{\delta_{i} W_{ij}}{d_{i}} & \text{if } i \neq j.
\end{cases}
\]

We have

\[
\sum_{i=1}^{n} \pi_{i} P_{i} = \sum_{i=1}^{n} \frac{d_{i}}{\sum_{k=1}^{n} d_{k}} \left( (1 - \delta_{i}) \mathbb{I}_{i=j} + \frac{\delta_{i} W_{ij}}{d_{i}} \right)
\]

\[
= \frac{d_{j} \left( 1 - \delta_{j} \right)}{\sum_{k=1}^{n} d_{k}} + \frac{\delta_{j} \sum_{i=1}^{n} W_{ij}}{\sum_{k=1}^{n} d_{k}}
\]

\[
= \frac{d_{j}}{\sum_{k=1}^{n} d_{k}}
\]

\[
= \pi_{j},
\]

as required. \( \square \)

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

\[
x^{(i)}(k \delta_{i}) = \mathbb{E} \left[ B^{(i)}(k) \right] = \sum_{j=1}^{n} x^{(j)}(0) \mathbb{P} \left( B^{(i)}(k) = x^{(j)} \right) \rightarrow \mathbb{E} \sum_{j=1}^{n} x^{(j)}(0) \pi_{j} = \bar{x}
\]

as \( k \rightarrow \infty \). \( \square \)

**Proof:** [Proof of Proposition 4] Let

\[
y^{(i)}(k) = \mathbb{E} \left[ \sum_{s=0}^{k} \frac{1}{d_{i}} \sum_{j \in \mathcal{I}} \left( x^{(j)} - \bar{x} \right) \mathbb{I}_{B^{(j)}(s) = x^{(i)}} \right].
\]
Notice that

\[
\mathbb{E} \left[ \sum_{s=0}^{k} \mathbb{1}_{B^{(j)}(s) = x^{(i)}} \right] \\
= \sum_{s=0}^{k} \mathbb{P} \left( B^{(j)}(s) = x^{(i)} \right) \\
= \underbrace{\mathbb{P} \left( B^{(j)}(0) = x^{(i)} \right)}_{\delta_{ij}} + \sum_{s=1}^{k} \mathbb{P} \left( B^{(j)}(s) = x^{(i)} \right)
\]

\[
\delta_{ij} + \sum_{s=1}^{k} \sum_{\ell=1}^{n} \mathbb{P} \left( B^{(j)}(s) = x^{(i)} | B^{(j)}(s-1) = x^{(l)} \right) \mathbb{P} \left( B^{(j)}(s-1) = x^{(l)} \right)
\]

\[
= \delta_{ij} + \sum_{s=1}^{k} \sum_{\ell=1}^{n} \left( 1 - \delta_{i} \right) \delta_{\ell i} + \delta_{i} \sum_{\ell=1}^{n} \frac{W_{\ell i}}{d_{\ell}} \sum_{s=1}^{k} \mathbb{P} \left( B^{(j)}(s-1) = x^{(l)} \right)
\]

\[
= \delta_{ij} + \left( 1 - \delta_{i} \right) \sum_{s=1}^{k} \mathbb{P} \left( B^{(j)}(s) = x^{(l)} \right) + \delta_{i} \sum_{\ell=1}^{n} \frac{W_{\ell i}}{d_{\ell}} \sum_{s=0}^{k} \mathbb{P} \left( B^{(j)}(s) = x^{(l)} \right)
\]

\[
= \delta_{ij} + \left( 1 - \delta_{i} \right) \mathbb{E} \left[ \sum_{s=0}^{k} \mathbb{1}_{B^{(j)}(s) = x^{(i)}} \right] + \delta_{i} \sum_{\ell=1}^{n} \frac{W_{\ell i}}{d_{\ell}} \mathbb{E} \left[ \sum_{s=0}^{k-1} \mathbb{1}_{B^{(j)}(s) = x^{(i)}} \right].
\]

From the definition of \( Y \) and the above recursive relationship we have

\[
y^{(i)}(k) = \frac{1}{d_{i}} \sum_{j \in I} \left( x^{(j)} - \hat{x} \right) \mathbb{E} \left[ \sum_{s=0}^{k} \mathbb{1}_{B^{(j)}(s) = x^{(i)}} \right]
\]

\[
= \frac{1}{d_{i}} \sum_{j \in I} \left( x^{(j)} - \hat{x} \right) \delta_{ij} + \left( 1 - \delta_{i} \right) \frac{1}{d_{i}} \sum_{j \in I} \left( x^{(j)} - \hat{x} \right) \mathbb{E} \left[ \sum_{s=0}^{k-1} \mathbb{1}_{B^{(j)}(s) = x^{(i)}} \right] + \delta_{i} \sum_{\ell=1}^{n} \frac{W_{\ell i}}{d_{\ell}} \sum_{j \in I} \left( x^{(j)} - \hat{x} \right) \mathbb{E} \left[ \sum_{s=0}^{k-1} \mathbb{1}_{B^{(j)}(s) = x^{(i)}} \right]
\]

\[
= \frac{1}{d_{i}} \sum_{j \in I} \left( x^{(j)} - \hat{x} \right) \delta_{ij} + \left( 1 - \delta_{i} \right) y^{(i)}(k-1) + \delta_{i} \sum_{\ell=1}^{n} W_{\ell i} y^{(i)}(k-1)
\]

\[
y^{(i)}(k-1) + \frac{1}{d_{i}} \sum_{j \in I} \left( x^{(j)} - \hat{x} \right) \delta_{ij} - \delta_{i} \left[ L Y(k-1) \right]^{(i)}.
\]

Now we let

\[
w^{(i)}(k) = d_{i} \left( z^{(i)}(k \delta_{i}) - y^{(i)}(k) \right)
\]

so that \( W \) satisfies

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

Hence, \( W(k) = P^{k} W(0) \). Since the stationary distribution of the random walk with transition kernel is \( \pi \), we have \( \lim_{k \to \infty} P^{k} = \mathbf{1} \pi' \). Hence, as \( k \to \infty \), \( w^{(i)}(k) \to \sum j=1 \pi_{j} w^{(j)}(0) = 0 \) since \( w^{(j)}(0) = 0 \) by the choice in the initialisation of \( Z \). \qed
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., [29]. Mathematically, a neural ODE is the first-order ODE:

\[
dh(t) = f(h(t), t, \theta),
\]

where \( f(h(t), t, \theta) \in \mathbb{R}^d \) is specified by a neural network parameterised by \( \theta \), e.g., a two-layer feed-forward neural network. Starting from the input \( h(0) \), 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 \( f(h(t), t, \theta) \) at various timesteps, with computational complexity counted by the number of forward function evaluations (forward NFEs) [13].

The adjoint sensitivity method (or adjoint method) [46], is a memory-efficient method for training neural ODEs. We regard the output \( h(T) \) as the prediction and denote the loss between \( h(T) \) and the ground truth as \( \hat{L} \). Let \( a(t) := \partial L / \partial h(t) \) be the adjoint state, then we have (see [13, 46] for details)

\[
\frac{dL}{d\theta} = \int_0^T a(t)^\top \frac{\partial f(h(t), t, \theta)}{\partial \theta} dt,
\]

with \( a(t) \) satisfying the following adjoint ODE

\[
\frac{da(t)}{dt} = -a(t)^\top \frac{\partial}{\partial h} f(h(t), t, \theta),
\]

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 ogbn-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].

**Depth of GRAND and GRAND++ for the results in Table 1.** Table 4 lists the fine-tuned \( T \) for the results in Table 1. Due to the limited time, we only search around the value of optimal \( T \) for GRAND with grid spacing 0.1.
Again, we see that GRAND++ is better than the other GNN models when the networks are deep. Table 4 lists the classification accuracy of GRAND++ with several other GNN architectures, including GCN, GAT, and GraphSage, with different depths on a few benchmark datasets. In this subsection, we provide detailed numbers that correspond to Fig. 1. We further compare NA: neural ODE solver failed. These results show that GRAND++ is better suited for classification tasks. Table 5 lists the classification accuracy of GRAND++-l with several other GNN architectures, including GCN, GAT, and GraphSage, with different depths on a few benchmark datasets. Table 5 shows that GRAND++ is better suited for classification tasks. NA: neural ODE solver failed. These results show that GRAND++ is better suited for learning with a very deep architecture than GRAND. Unit: %.

### 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 suited for learning with a very deep architecture than GRAND. 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 6 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-1, 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++.
different numbers of labeled nodes per class. We evaluate the classification accuracy, on the test set, of different benchmark GNN models when they are trained with different GNN architectures trained with fewer numbers of labeled data per class. Tables 7-9 list the classification accuracy of different GNNs trained with different numbers of labeled nodes per class.

Classification accuracy of different GNNs trained with different numbers of labeled nodes per class.

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 7-9 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.

### Table 6: 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: %)

<table>
<thead>
<tr>
<th>Model</th>
<th>#labeled nodes per class</th>
<th>GRAND-nl [10]</th>
<th>GRAND-nl-rw [10]</th>
<th>GRAND++-nl (ours)</th>
<th>GRAND++-nl-rw (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORA</td>
<td>1</td>
<td>50.55 ± 15.68</td>
<td>50.63 ± 17.71</td>
<td>48.89 ± 11.51</td>
<td>47.94 ± 11.06</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>65.06 ± 9.35</td>
<td>61.24 ± 16.19</td>
<td>59.96 ± 7.90</td>
<td>58.25 ± 11.97</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>76.93 ± 3.10</td>
<td>76.50 ± 3.91</td>
<td>78.01 ± 1.73</td>
<td>74.25 ± 1.99</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>79.60 ± 2.69</td>
<td>79.38 ± 3.25</td>
<td>80.14 ± 0.69</td>
<td>80.18 ± 0.40</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>82.22 ± 1.93</td>
<td>82.14 ± 2.49</td>
<td>83.24 ± 0.20</td>
<td>81.48 ± 1.07</td>
</tr>
<tr>
<td>CiteSeer</td>
<td>1</td>
<td>50.25 ± 17.66</td>
<td>50.20 ± 17.90</td>
<td>49.65 ± 5.45</td>
<td>53.10 ± 5.51</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>59.87 ± 10.89</td>
<td>59.95 ± 10.48</td>
<td>59.16 ± 8.13</td>
<td>60.26 ± 5.10</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>68.21 ± 5.08</td>
<td>68.05 ± 5.48</td>
<td>66.13 ± 2.09</td>
<td>67.81 ± 1.97</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>71.88 ± 6.94</td>
<td>71.92 ± 7.34</td>
<td>68.84 ± 2.84</td>
<td>71.45 ± 1.64</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>72.84 ± 6.61</td>
<td>72.73 ± 6.85</td>
<td>72.52 ± 1.24</td>
<td>73.87 ± 1.35</td>
</tr>
</tbody>
</table>

### Table 7: Classification accuracy of different GNNs trained with different numbers of labeled nodes per class. (Unit: %)

To confirm the statistical significance of the accuracy improvement of GRAND++ over GRAND in Table 6, in this subsection, we conduct t-test experiments at 0.95 confidence to compare GRAND and GRAND++ on six different benchmark graph node classification tasks. We first perform unpaired t-tests to show the improvement of GRAND++ over GRAND on low labeled datasets using the following t-score:

$$t\text{-score} = \frac{\bar{\mu}_{\text{GRAND++}} - \bar{\mu}_{\text{GRAND}}}{\sqrt{\frac{\sigma_{\text{GRAND++}}^2}{n} + \frac{\sigma_{\text{GRAND}}^2}{n}}}$$  \(2\)
Table 9: Classification accuracy of different GNNs trained with different numbers of labeled nodes per class. Dataset: PubMed.

<table>
<thead>
<tr>
<th>#labeled nodes per class</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN [30]</td>
<td>58.61 ± 12.83</td>
<td>60.45 ± 16.20</td>
<td>68.69 ± 7.93</td>
<td>72.59 ± 3.19</td>
<td>76.89 ± 3.27</td>
</tr>
<tr>
<td>GAT [54]</td>
<td>58.84 ± 12.81</td>
<td>60.24 ± 14.44</td>
<td>68.54 ± 5.75</td>
<td>72.44 ± 3.50</td>
<td>75.55 ± 4.11</td>
</tr>
<tr>
<td>GraphSage [29]</td>
<td>55.93 ± 12.71</td>
<td>58.97 ± 12.65</td>
<td>66.07 ± 6.16</td>
<td>70.74 ± 3.11</td>
<td>74.55 ± 3.09</td>
</tr>
<tr>
<td>MoNet [60]</td>
<td>56.47 ± 4.67</td>
<td>61.03 ± 6.93</td>
<td>67.92 ± 2.50</td>
<td>71.24 ± 1.54</td>
<td>76.49 ± 1.75</td>
</tr>
<tr>
<td>Lanczos [26]</td>
<td>60.12 ± 5.16</td>
<td>64.11 ± 6.88</td>
<td>69.05 ± 3.00</td>
<td>72.79 ± 2.74</td>
<td>78.10 ± 1.91</td>
</tr>
<tr>
<td>Adalanczos [36]</td>
<td>61.07 ± 5.16</td>
<td>64.11 ± 6.88</td>
<td>69.05 ± 3.00</td>
<td>72.79 ± 2.74</td>
<td>78.10 ± 1.91</td>
</tr>
<tr>
<td>GCNN [55]</td>
<td>60.78 ± 20.64</td>
<td>65.14 ± 19.45</td>
<td>72.72 ± 10.82</td>
<td>76.47 ± 6.03</td>
<td>79.24 ± 3.45</td>
</tr>
<tr>
<td>GRAND-l [10]</td>
<td>62.11 ± 10.58</td>
<td>69.00 ± 7.55</td>
<td>73.98 ± 5.08</td>
<td>76.33 ± 3.41</td>
<td>78.76 ± 1.84</td>
</tr>
<tr>
<td>GRAND-nl [10]</td>
<td>61.75 ± 11.12</td>
<td>69.16 ± 8.46</td>
<td>72.35 ± 5.35</td>
<td>76.03 ± 3.72</td>
<td>78.55 ± 1.59</td>
</tr>
<tr>
<td>GRAND-nl-rw (gdc) [10]</td>
<td>61.70 ± 10.74</td>
<td>69.42 ± 8.21</td>
<td>72.39 ± 5.25</td>
<td>75.32 ± 3.45</td>
<td>78.30 ± 1.43</td>
</tr>
<tr>
<td>GRAND-nl-rw (two-hop) [10]</td>
<td>61.65 ± 12.09</td>
<td>68.49 ± 8.99</td>
<td>72.68 ± 5.92</td>
<td>75.72 ± 3.50</td>
<td>78.77 ± 1.88</td>
</tr>
</tbody>
</table>

where $\mu$ and $\sigma^2$ are the mean and variance of the performances of each model, and $n$ is the number of runs for each model. The t-test score are shown in Table 10.

Table 10: Unpaired t-test scores of GRAND++ v.s. GRAND on six different benchmark graph node classification tasks. With $n = 100$, over 0.95 confidence is equivalent to exceed roughly 1.66 t-test scores. Highlighted are the ones passing the test.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#per class</th>
<th>Accuracy</th>
<th>Difference</th>
<th># splits</th>
<th>t-score</th>
<th>p-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORA</td>
<td>1</td>
<td>1.05</td>
<td>4.36</td>
<td>3.28</td>
<td>1.95</td>
<td>111.60</td>
</tr>
<tr>
<td>CiteSeer</td>
<td>2</td>
<td>1.39</td>
<td>3.96</td>
<td>0.34</td>
<td>4.59</td>
<td>7.18</td>
</tr>
<tr>
<td>PubMed</td>
<td>5</td>
<td>2.55</td>
<td>2.68</td>
<td>-3.67</td>
<td>-1.96</td>
<td>15.67</td>
</tr>
</tbody>
</table>

For some entries in Table 10 that are not significant enough, we further conduct paired t-test between GRAND++ and GRAND on these specific datasets as shown in Table 11. Since a large portion of variance comes from splitting of the datasets, we pair up tests of GRAND and GRAND++ with the same splitting in this experiment. In this case, a sample of difference of size $n$ is computed, and t-test score can be computed using the equation

$$t\text{-score} = \frac{\mu_{\text{diff}} - 0}{\sigma_{\text{diff}} / \sqrt{n}}$$

(22)

Table 11: Paired t-test scores of GRAND++ v.s. GRAND on datasets where unpaired t-test scores are not significant enough.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#per class</th>
<th>Accuracy</th>
<th>Difference</th>
<th># splits</th>
<th>t-score</th>
<th>p-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORA</td>
<td>1</td>
<td>1.06 ± 6.24</td>
<td>100</td>
<td>1.80</td>
<td>0.044</td>
<td></td>
</tr>
<tr>
<td>CiteSeer</td>
<td>2</td>
<td>1.45 ± 5.23</td>
<td>100</td>
<td>2.78</td>
<td>0.003</td>
<td></td>
</tr>
</tbody>
</table>

D.6 TASKS FOR FURTHER EVALUATING DEEP GRAND AND GRAND++

Open graph benchmark with paper citation network (ogbn-arxiv). Ogbn-arxiv consists of 169,343 nodes and 1,166,243 directed edges. Each node is an arxiv paper represented by a 128-dimensional features and each directed edge indicates the citation direction. This dataset is used for node property prediction and has been a popular benchmark to test the advantage of deep graph neural networks over shallow graph neural networks [34, 32]. Compared to the GRAND model used in [10], we reduce the hidden dimension from 162 to 81 to fit the model into the GPU in our lab.

<table>
<thead>
<tr>
<th>Model</th>
<th>depth ($T$)</th>
<th>GRAND-l [10]</th>
<th>GRAND++-l (ours)</th>
<th>Improvement from GRAND++-l (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORA</td>
<td>1</td>
<td>68.50 ± 0.76</td>
<td>68.79 ± 0.35</td>
<td>0.29</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>69.53 ± 0.21</td>
<td>69.68 ± 0.38</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>69.46 ± 0.43</td>
<td>69.71 ± 0.24</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>69.46 ± 0.30</td>
<td>69.61 ± 0.28</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>67.44 ± 0.59</td>
<td>69.64 ± 0.53</td>
<td>1.97</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>67.44 ± 0.59</td>
<td>69.64 ± 0.53</td>
<td>1.97</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>63.47 ± 0.28</td>
<td>68.05 ± 0.73</td>
<td>4.58</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>55.95 ± 1.24</td>
<td>67.26 ± 0.61</td>
<td>11.31</td>
</tr>
</tbody>
</table>

Table 12: Classification accuracy of the linear GRAND and GRAND++ models trained with different depth on the OGBN-arxiv graph node classification task. Compared to the GRAND model used in [10], we reduce the hidden dimension from 162 to 81 to fit the model into the GPU in our lab. (Unit: %)