Honours Project
Graph Neural Diffusion: An Analysis of Stability in Graph Neural Networks

Concordia University
Department of Mathematics and Statistics

Étienne Dyer
Supervised by Simone Brugiapaglia and Giuseppe Alessio D'Inverno

August 27, 2025

Abstract

GRAph Neural Diffusion (GRAND) is a framework for graph neural networks developed by Chamberlain et al. [1], a team of researchers at Twitter, whose main innovation lies in treating the propagation of information throughout a Graph Neural Network (GNN) as a continuous diffusion process, where the state of the graph is solved for at each step with various numerical methods for Ordinary Differential Equations (ODEs). This project explores the GRAND model from the perspective of numerical analysis (understanding its stability properties for different solving schemes) and machine learning (understanding its strengths and weaknesses compared to other GNNs, specifically Graph Convolutional Networks (GCNs) and Graph ATtention networks (GATs)). Finally, we explore its numerical performance on different graph data benchmarks, and analyze the authors' claim that GRAND is resistant to oversmoothing.

1 Introduction

Graph neural networks (GNNs) are a class of neural networks that operate on graphs, structured data that consists of a set of nodes connected by edges. Graphs are used to represent a variety of data, such as social networks, protein interactions, and communication networks [6]. GNNs have been successfully applied to many tasks, such as predicting node labels (e.g., classifying users in a social network) or graph properties (e.g., predicting the bio-activity of a molecule [7]). While there has been significant interest in this topic in recent years, much of the focus has been on practical applications, with comparatively little attention paid to their theoretical properties.

GRAND is a node classification model that suggests a new paradigm for GNNs, where the propagation of information throughout a graph is treated as a continuous diffusion process.

This allows us to model the process using differential equations, offering a mathematically rigorous way to understand GNNs, potentially offering better performance at tasks such as graph classification and clustering.

Motivation. Discrete-layer GNNs such as GCN [2] and GAT [3] iteratively propagate information through distinct layers. When the number of layers, or depth, is increased, the node representations on which the models operate tends to get very similar (a process known as oversmoothing) and can exhibit issues with optimization and numerical stability. [1]. GRAND addresses these issues by modeling propagation as a continuous-time diffusion, allowing us to represent it as a differential equation, which is integrated numerically. This work investigates the numerical properties and practical performance of GRAND relative to these baselines.

Outline. The remainder of the report is organized as follows:

- Background: numerical methods for ODEs and a short history of GNNs to motivate GRAND.
- Graph Neural Diffusion: GRAND architecture (encoder — diffusion — decoder), diffusion equation, and stability results (explicit / implicit Euler) following [1].
- Numerical Results: classification benchmark on the Cora dataset and oversmoothing analysis on Cora comparing GCN, GAT, and GRAND.
- Conclusion: main takeaways and directions for improved solvers and broader datasets.

2 Background

In this section, we will explore the topics necessary to understanding Graph Neural Diffusion. To appreciate the innovations of GRAND, one must first be familiar with two distinct fields: the numerical analysis of differential equations and the recent history of graph neural networks. This section provides a self-contained overview of both, establishing the context in which GRAND was developed and the problems it aims to solve.

2.1 Numerical Methods for Differential Equations

Numerical methods is the study of approximation of solutions to problems that are difficult or impossible to solve analytically. In the context of GRAND, we are concerned with solving ODEs, which describe the rate of change of a system over time. Since the diffusion process at the heart of GRAND is modeled by an ODE, understanding how to approximate its solution is fundamental to implementing and analyzing the model.

2.1.1 Euler Methods

The first numerical method we will analyze is the forward Euler method, a first-order method for solving ODEs. Given an ordinary differential equation of the form $\frac{dx}{dt} = f(t, x)$, with initial condition $x(0) = x_0$, the forward Euler method uses a finite-difference approximation

of the time derivative to estimate the solution at discrete time steps $t_k = k\tau$, where τ is the step size. The update rule is given by

$$\begin{aligned} x_{k+1} &= x_k + \tau f(t_k, x_k),\ k \in \mathbb{N},\ k \geq 0, \\ x_0 &= x(0). \end{aligned} \tag{1}$$

Heuristically, this tells us that the solution at time $t = k+1$ is equal to the solution at time $t = k$, plus a small increment τ times the value of our original function at time $t = k+1$. This is also known as an explicit method because the new state x_{k+1} is given as an explicit function of the previous state x_k. This is contrasted by the backwards, or implicit Euler method, whose name points to the fact that the equation must be solved implicitly. The update rule for the implicit Euler method is given by:

$$\begin{aligned} x_{k+1} &= x_k + \tau f(t_{k+1}, x_{k+1}), \\ x_0 &= x(0). \end{aligned} \tag{2}$$

Notice that the new state x_{k+1} appears on both sides of the equation. This means we cannot simply compute it directly; we must rearrange and solve for it. When applied to GRAND's graph diffusion equation,

$$\frac{\partial \boldsymbol{x}(t)}{\partial t} = (\boldsymbol{A} - \boldsymbol{I})\boldsymbol{x}, \tag{3}$$

which we will motivate in section 3.2, this yields:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \tau(\boldsymbol{A} - \boldsymbol{I})\boldsymbol{x}_{k+1}. \tag{4}$$

Rearranging the terms to solve for $\boldsymbol{x}_{k+1}$, assuming the matrix is invertible, we get:

$$\begin{aligned} (\boldsymbol{I} - \tau(\boldsymbol{A} - \boldsymbol{I}))\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k \\ \boldsymbol{x}_{k+1} &= (\boldsymbol{I} - \tau(\boldsymbol{A} - \boldsymbol{I}))^{-1}\boldsymbol{x}_k. \end{aligned} \tag{5}$$

This is a system of linear equations that must be solved at each time step to find the next state. While computationally more expensive than the explicit method, its significant advantage is that it is unconditionally stable. We explore what this means in the following section, 2.1.2.

2.1.2 Numerical Stability

Numerical stability is a fundamental concept in the analysis of differential equations that determines whether small perturbations in the initial conditions or parameters lead to bounded errors in the computed solution. A numerical method is said to be stable if the accumulated error remains bounded as the computation progresses, which is often formalized by requiring the amplification factor $|g(\tau\lambda)| \leq 1$, where $g(x)$ is a function that dictates how errors accumulate with each time step. Methods can be conditionally or unconditionally stable, meaning that stability may be contingent upon the value of a parameter. In section 3.2, we find that one discretization is unconditionally stable (backward Euler), and one is stable given a sufficiently small time step (forward Euler).

In the context of GRAND, stability analysis is particularly important because the diffusion equation we solve is inherently stable (it tends toward a steady state), but our numerical discretization might not preserve this property. The choice of solver affects both the computational cost and the quality of the learned representations.

2.2 A Brief History of GNNs

The concept of applying neural networks to graph-structured data predates the recent surge in deep learning. The original Graph Neural Network model, proposed by Scarselli et al. [4] in 2009, introduced a recurrent architecture capable of processing graphs by iteratively updating node states until a stable equilibrium was reached. This foundational work established the core principles of message passing and state updates that underpin modern GNNs. In this section, we briefly review two of the most influential discrete-layer models, GCN and GAT, to provide context for the continuous-time approach taken by GRAND.

2.2.1 GCN

Graph Convolutional Networks (GCN) were introduced by Kipf and Welling in 2017 [2]. They introduced the core idea of convolution on graphs, but treated all neighbors equally. However, it is not always the case that all neighbors are equally important. This is especially true in contexts like social networks, where the importance of a node's neighbors can vary greatly depending on the context.

The propagation rule for a GCN layer is given by:

$$H^{(l+1)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}). \tag{6}$$

where $\hat{A} = A + I$ is the adjacency matrix, $\hat{D}$ is the diagonal degree matrix of $\hat{A}$, $H^{(l)}$ is the matrix of node features at layer l, $W^{(l)}$ is a learnable weight matrix, and σ is a non-linear activation function like the Rectified Linear Unit (ReLU).

2.2.2 GAT

Graph Attention Networks (GAT) [3] improved upon GCN by introducing attention mechanisms, allowing the model to learn the importance of different neighbors dynamically. This addresses one of GCN's key limitations: treating all neighbors equally regardless of their relevance to the current task.

The attention mechanism in GAT computes attention coefficients between node i and its neighbor j as:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j]))}{\sum_{k\in\mathcal{N}_i}\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_k]))}, \tag{7}$$

where $\mathbf{W}$ is a learnable weight matrix, $\mathbf{a}$ is a learnable attention vector, $\mathbf{h}_n$ is the feature vector of node n, and $\|$ denotes column-wise concatenation. The updated node representation is then given by

$$\mathbf{h}'_i = \sigma(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\mathbf{h}_j), \tag{8}$$

where $\mathcal{N}_\rangle$ represents the set of nodes neighboring node i. While GAT represents a significant improvement over GCN, it still suffers from the fundamental limitation of discrete layer updates, which can lead to oversmoothing in deep architectures [1].

3 Graph Neural Diffusion

Building on top of GAT, GRAND gives this process a rigorous mathematical foundation by modeling diffusion as a PDE, and brings about practical improvements like reducing oversmoothing and increased efficiency through parameter sharing.

The authors base their model on the heat diffusion equation

$$\frac{\partial x(u,t)}{\partial t} = \operatorname{div}\left[g(u,t)\nabla x(u,t)\right], \tag{9}$$

where $x(u,t)$ is the temperature at position u and time t, $g(u,t)$ is the diffusivity coefficient, ∇ is the gradient operator, and div is the divergence operator.

3.1 Architecture

In practice, the fact we are considering diffusion over a graph allows us to discretize space: each point in continuous space u instead becomes a fixed node in the graph, which allows us to express our problem as a system of ODEs with respect to time. The GRAND model has three main components: an encoder, the diffusion block, and a decoder.

The encoder ϕ is a single linear layer which takes the initial node features $\boldsymbol{x}_{\text{in}}$ and projects them into a lower-dimensional latent space. This produces the initial state for the diffusion process, $\boldsymbol{x}(0) = \phi(\boldsymbol{x}_{\text{in}})$.

The diffusion block is the core of the GRAND architecture. Instead of using a discrete stack of layers, GRAND treats depth as a continuous variable, i.e., time. It takes the initial latent state $\boldsymbol{x}(0)$ and solves the graph diffusion equation over a time interval $[0, T]$. The output of this block is the final state of the node features after the diffusion process.

Finally, we have the decoder, which is a linear layer that takes the final latent state $\boldsymbol{x}(T)$ and projects it into the desired output dimension.

This structure allows the model to learn an optimal transformation of node features by solving a continuous differential equation, where the integration time T is analogous to the depth of a traditional neural network.

3.2 Graph Diffusion Equation

Formally, a graph $G(V, \mathcal{E})$ is a pair of vertices V and edges $\mathcal{E}$. The graph operator div is defined component-wise as

$$(\operatorname{div}(\mathbf{K}))_i = \sum_{j:(i,j)\in\mathcal{E}} \mathbf{K}_{ij} = \sum_{j=1}^{n} w_{ij}\mathbf{K}_{ij}, \tag{10}$$

where $\mathbf{K}$ is the feature matrix of G, and w is its adjacency matrix. We also define an arbitrary attention function $a : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^+$, which computes a scalar attention weight between two nodes' feature vectors at a given time. This allows us to define a time-dependent attention matrix $\boldsymbol{A}(t)$ where each entry is given by:

$$A_{ij}(t) = a(\boldsymbol{x}_i(t), \boldsymbol{x}_j(t), t). \tag{11}$$

Note that we must also normalize $\boldsymbol{A}(t)$'s rows, in order to ensure it is right-stochastic (meaning row entries sum to 1), a property we will use in our proofs of numerical stability. The diffusion process is governed by the diagonal matrix $\boldsymbol{G}(t)$, an $m \times m$ matrix where $m = |\mathcal{E}|$ is the number of edges. For each edge $(i, j) \in \mathcal{E}$, the corresponding diagonal entry in $\boldsymbol{G}(t)$ is the attention weight $A_{ij}(t)$. We consider the following diffusion equation on the graph:

$$\frac{\partial \boldsymbol{x}(t)}{\partial t} = \mathrm{div}[\boldsymbol{G}(t)\nabla \boldsymbol{x}(t)]. \tag{12}$$

Using Equations (10) and the definition of $\boldsymbol{G}$ we obtain:

$$\begin{aligned}\left(\frac{\partial \boldsymbol{x}(t)}{\partial t}\right)_{ij} &= \mathrm{div}[a(x_i, x_j)(x_j - x_i)] \\ &= \mathrm{div}[a_{ij}x_j - a_{ij}x_i].\end{aligned}$$

Applying the given definition for div yields

$$\left(\frac{\partial \boldsymbol{x}(t)}{\partial t}\right)_i = \sum_j a_{ij}x_j - x_i \sum_j a_{ij}. \tag{13}$$

Since $\boldsymbol{A}$ is right-stochastic, $\sum_j a_{ij} = 1$. Therefore, looking at the i-th component we have

$$\left(\frac{\partial \boldsymbol{x}(t)}{\partial t}\right)_i = (\boldsymbol{A}\boldsymbol{x})_i - (\boldsymbol{I}\boldsymbol{x})_i. \tag{14}$$

Or, in matrix form

$$\frac{\partial \boldsymbol{x}(t)}{\partial t} = (\boldsymbol{A} - \boldsymbol{I})\boldsymbol{x}. \tag{15}$$

We will now proceed with the demonstration of numerical stability for the forward and backward Euler schemes. Our proofs are based on the ones published in the original GRAND paper [1].

Theorem 1. *The explicit Euler scheme* $\mathbf{x}^{(t+1)} = (\mathbf{I} + \tau\bar{\mathbf{A}}(\mathbf{x}^{(t)}))\mathbf{x}^{(t)} := Q^{(t)}\mathbf{x}^{(t)}$ *is stable for* $0 < \tau < 1$*, when* $\boldsymbol{A}(t)$ *is right-stochastic.*

Proof. For this scheme to be numerically stable, we require that $\|Q\| \leq 1$. A matrix being right-stochastic implies its spectral radius = 1 ([9]), so we simply need that every row sum is ≤ 1. We're given that A is right-stochastic, so for $0 < \tau < 1$, we have

$$\max_i \sum_j |q_{ij}| = \max_i \sum_j |I_{ij} + \tau(a_{ij} - I_{ij})|.$$

Since $\tau < 1$, and $a_{ij} > 0$ for all i, j the summand is nonnegative, so we can remove the absolute value and distribute the sum over j, yielding

$$\max_i \sum_j |q_{ij}| = \max_i \left(1 + \tau \sum_j (a_{ij} - I_{ij})\right).$$

Finally, the max for each term over any given row is 1, so we obtain

$$\begin{aligned}\max_i \sum_j |q_{ij}| &= 1 + \tau(1-1) \\ &= 1,\end{aligned}$$

completing our proof that $\|Q\| \leq 1$, hence that the forward Euler scheme is numerically stable. □

Theorem 2. *The implicit scheme* $\left(I - \tau \bar{A}(\boldsymbol{x}^{(k)})\right)^{-1} \boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)}$ *is unconditionally stable for any* $\tau > 0$.

Proof. Similarly to the proof of Theorem 1, we want to show that $Q = (I - \tau\bar{A})^{-1} := B^{-1}$ has a spectral radius bounded by 1, which follows from it being right-stochastic. However, this proof is slightly more involved, as we're dealing with the inverse of a matrix. Note that B is invertible as it is diagonally dominant.
We start by showing that the row sums of B are all 1:

$$B = I - \tau(A - I) = (1+\tau)I - \tau A$$

So the sum of B's i-th row is

$$\sum_j B_{ij} = (1+\tau) - \tau \sum_j a_{ij}.$$

Since A is right-stochastic, this reduces to

$$\sum_j B_{ij} = 1.$$

Then, given $w = (1, 1, ..., 1)$, we have $Bw = w$. Multiplying each side by B^{-1}, we obtain $w = B^{-1}w \implies w = Qw$, so Q also has rows summing to 1.

The other condition Q needs to fulfill in order to be right-stochastic is to have all its entries be nonnegative. This follows from *Corollary 3.20* in R. S. Varga's *Matrix Iterative Analysis*, [5], which we state without proof:

Theorem 3. *If* $A = [a_{i,j}]$ *is a real, irreducibly diagonally dominant* $n \times n$ *matrix with* $a_{i,j} \leq 0$ *for all* $i \neq j$, *and* $a_{i,j} > 0$ *for all* $1 \leq i \leq n$, *then* $A^{-1} > 0$.

B being diagonally dominant with with positive diagonal entries, so its inverse, Q has nonnegative entries. Therefore, Q is right stochastic, so its spectral radius is bounded above by 1, so the implicit Euler scheme is unconditionally stable □

4 Numerical Results

To evaluate the performance and stability of GRAND, we conducted a series of experiments on the Cora citation network. We first benchmarked our implementation of GRAND against GCN and GAT on a standard node classification task to establish a performance

baseline. We then conducted a novel experiment to analyze the evolution of node representations as a function of network depth (or time, in the case of GRAND). This analysis was performed on trained models to understand how the learning process interacts with the inherent architectural properties of each model, providing a more nuanced view of the oversmoothing phenomenon and revealing crucial differences in numerical stability. All our implementations use PyTorch, and the code is publicly available on GitHub.

Figure 1: Visualization of Cora dataset. Taken from *PKGCN: prior knowledge enhanced graph convolutional network for graph-based semi-supervised learning* by Yu et al. [8]

4.1 Cora Classification Benchmark

For a quantitative evaluation, we benchmarked our model on the Cora dataset, a standard citation network where the task is to classify academic papers into one of seven subjects. We compare our result against the baseline GCN and GAT models, as well as the official GRAND-l performance reported by Chamberlain et al. [1].

As shown in Table 1, our implementation achieves a test accuracy of 79% over 10 seeds on the Cora dataset. While this clearly demonstrates the algorithm is working, it underperforms both the standard GCN and GAT baselines, as well as the 84.7% reported for the authors' GRAND-l model. The performance gap between our implementation and the official one can be attributed to several key architectural differences. The official implementation is a more robust and feature-rich framework that leverages the specialized "torchdiffeq" library for solving the ODE, allowing for the use of more advanced and adaptive numerical integrators. Furthermore, it incorporates additional architectural components that enhance performance,

Model	Cora Test Accuracy
GCN [2]	81.9% ± 0.8
GAT [3]	82.8% ± 0.5
Our GRAND-l Implementation	**79.0%** ± 0.9
Official GRAND-l [1]	84.7% ± 0.6

Table 1: Performance comparison on the Cora dataset. Baseline results for GCN and GAT are taken from the official GRAND paper [1] for consistency.

such as a learnable parameter 'alpha' that scales the diffusion term, optional MLP layers after the encoder, and the use of Batch Normalization. Our implementation sought to remain faithful to what was described in the paper (GRAND-l with a manual forward Euler solver).

4.2 Oversmoothing and Stability Analysis

One of the most important claims made by the authors is that GRAND is more resistant to the problem of oversmoothing. Oversmoothing is a well-known limitation in deep GNNs where, as the number of layers increases, the representations of all nodes converge to a homogeneous, uninformative feature. This washes out the unique features of individual nodes, making it impossible for the model to distinguish between them and leading to a sharp decline in performance. We sought to measure this phenomenon directly by analyzing the distinctiveness of node representations as a function of model depth.

Our initial hypothesis was that we would observe this phenomenon directly by measuring the collapse of node representations in untrained models. However, we found this to be highly sensitive to the random weight initialization. To create a more meaningful and robust analysis, we redesigned the experiment to analyze the oversmoothing properties of models that have been trained on a node classification task.

4.2.1 Experimental Design

To measure oversmoothing and stability, we first trained each model (GCN, GAT, and GRAND) on a standard node classification task using the Cora dataset. The dataset represents 2,708 scientific publications, each classified into one of seven categories, with the 5,429 edges representing a citation. Each publication is described by a 1,433-dimensional feature vector representing what words are present or absent from the paper. The task is to classify academic papers into one of seven subjects. Each model was trained for 200 epochs using an Adam optimizer and cross-entropy loss. To ensure the robustness of our findings, we repeated this process for 10 runs with different random seeds.

We used the mean pairwise Euclidean distance between all node representations as our primary metric. A decrease in this distance signifies smoothing (a collapse towards a single point), while an increase signifies that the model is actively separating the nodes in the embedding space. We use the formula (final distance − initial distance)/(initial distance) to calculate distance change, where a negative value indicates smoothing and a positive value indicates separation.

For each model, we performed a forward pass of the initial node features through the trained network. At each layer (for GCN and GAT) or sampled time step (for GRAND),

we extracted the latent node representations and calculated the mean pairwise Euclidean distance. The results were then aggregated across the 10 runs to compute the mean and standard deviation for our metrics.

4.2.2 Code implementation

Each model was implemented in PyTorch. The GCN and GAT models were built with 5 layers to create a sufficiently deep network to observe oversmoothing. The GRAND model's diffusion function was adjusted to store node states at different time intervals between $t = 0$ and $t = 5$.

4.2.3 Results and discussion

The results of the experiment offer a more nuanced perspective than the simple narrative of oversmoothing. While our initial hypothesis was that we would observe a collapse in node distances, the experiment on trained models revealed the opposite: a significant *increase* in mean pairwise distance for all architectures. This suggests that the training objective, which pushes representations of different classes apart, is a stronger force than the inherent smoothing tendency of the graph convolutions.

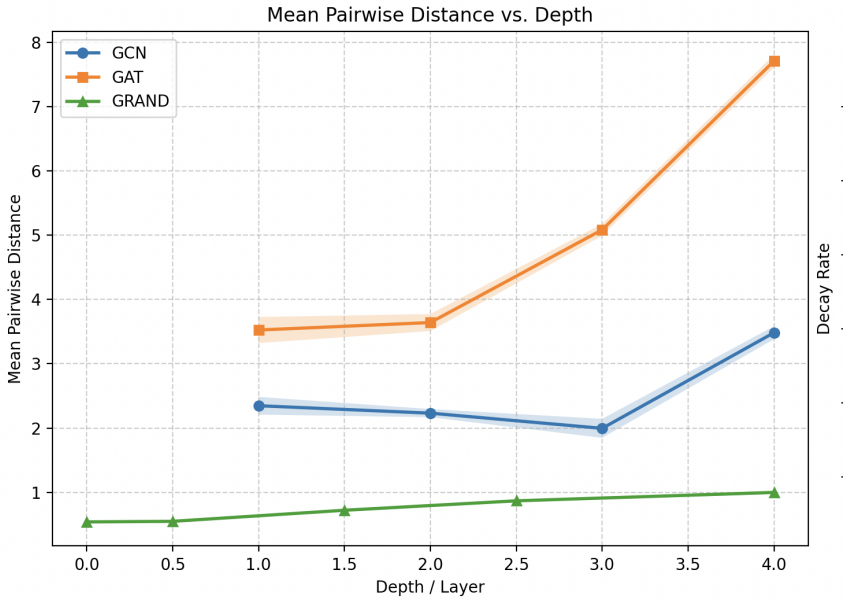

Figure 2: Mean pairwise distances vs. depth for GCN, GAT, and GRAND models on the Cora dataset, aggregated over 10 runs. The shaded regions represent ±1 standard deviation. Contrary to simple oversmoothing, all trained models learn to increase node separation with depth to aid classification. GCN exhibits a non-monotonic trend, while GRAND shows a stable, controlled increase in distance.

GAT exhibited the most aggressive separation, more than doubling the mean distance between nodes, but as we will discuss, this came at the cost of stability. GCN showed

a curious non-monotonic behavior, with an initial slight smoothing followed by a sharp increase in distance at the final layer. GRAND demonstrated a strong and steady increase in node separation, showing that it effectively learns to distinguish nodes while maintaining a more controlled evolution of representations.

Model	Initial Distance	Final Distance	Relative Change
GCN	2.3472 ± 0.1	3.4844 ± 0.1	+48.80% ± 6.8%
GAT	3.5255 ± 0.2	7.7130 ± 0.1	+119.50% ± 12.9%
GRAND	0.5389 ± 0.01	0.9976 ± 0.02	+85.1% ± 4.5%

Table 2: Oversmoothing and stability analysis results on the Cora dataset, aggregated over 10 runs. All models learn to separate node representations. The relative change indicates the percentage increase in mean pairwise distance from the first to the last layer/time step.

One interesting result from this experiment is the difference in numerical stability between the models. Stability can be measured by the variance in outcomes across multiple runs with different random initializations. As shown by the standard deviation values in Table 2, GRAND is exceptionally stable. Its standard deviation in relative change (4.59%) is significantly lower than that of GCN (6.84%) and almost three times lower than GAT's (12.86%). This indicates that GRAND's continuous, gradual update process makes it highly robust and its performance far less sensitive to initialization, which is a highly desirable property in deep learning models.

While we did not observe the expected oversmoothing phenomenon, this experiment reveals a crucial insight: for trained models on a sufficiently complex task, the primary challenge is not avoiding collapse, but managing the expansion of the representation space in a stable and predictable manner. In this regard, GRAND's architecture proves to be markedly superior.

4.2.4 Why is GRAND More Stable?

The superior numerical stability of GRAND, as evidenced by the lower variance across runs, can be attributed to its continuous-time formulation and the principle of parameter sharing. Interpreting the mechanics of deep neural networks is notoriously difficult, but there does exist a natural explanation of the phenomenon we've observed. In discrete models like GCN and GAT, each layer introduces a distinct set of learnable parameters ($W^{(l)}$). During training, a small change in the weights of an early layer can be amplified through subsequent layers, leading to a large and potentially chaotic shift in the final output. This effect, known as the exploding gradient problem in deep networks, makes the model highly sensitive to the random initialization of its weights. Different initializations can set the model on vastly different optimization paths, resulting in high variance in the final learned representations.

GRAND, by contrast, uses the same underlying ODE function, $f(t, \boldsymbol{x}) = (\boldsymbol{A} - \boldsymbol{I})\boldsymbol{x}$, at every point in time. The layers are replaced by the steps of a numerical ODE solver, but the parameters governing the dynamics (the weights in the attention matrix $\boldsymbol{A}$) are shared across the entire integration interval $[0, T]$. This parameter sharing imposes a strong regularization on the model, ensuring a smoother and more constrained evolution of the node features. The transformation applied at time t is very similar to the one at $t + \Delta t$, preventing the kind of abrupt changes that can occur between discrete layers. As a result,

the optimization landscape is smoother, and the model is far less sensitive to its initial parameters, leading to the highly consistent and stable results observed in our experiments.

5 Conclusion

This project set out to implement and analyze the Graph Neural Diffusion (GRAND) model, with a specific focus on empirically testing the authors' claims regarding its resistance to oversmoothing. Our investigation led to a more nuanced conclusion than anticipated. The experiment on trained models using the Cora dataset did not demonstrate the expected oversmoothing (a collapse in node representations), but instead revealed a process of "over-spreading," where all models learned to actively increase the distance between node representations to aid classification.

The most significant finding of this work is the clear evidence of GRAND's superior numerical stability compared to discrete-layer models like GCN and GAT. Across multiple runs with different initializations, GRAND produced remarkably consistent results, with significantly lower variance in its final representations and performance. This robustness suggests that its continuous-time formulation provides a more stable and reliable framework for learning on graphs.

Future work could explore this interplay between training and oversmoothing on a wider range of datasets and tasks. Investigating whether oversmoothing becomes dominant in much deeper, trained architectures or on different graph structures remains an open question. However, our results strongly support the idea that continuous-time approaches like GRAND represent a promising direction for building more robust and predictable graph neural networks.

References

[1] B. P. Chamberlain, J. Rowbottom, D. Eynard, F. Di Giovanni, F. Monti, and M. M. Bronstein. Grand: Graph neural diffusion. In *International Conference on Machine learning*, pages 1531–1542, 2021.

[2] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[3] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[4] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[5] R. S. Varga. Matrix Iterative Analysis. *Englewood Cliffs, N.J.: Prentice-Hall.*, 1962.

[6] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[7] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, 30, 2017.

[8] J. Yu, C. Li, C. Yang, M. Zhu, and J. Han. Prior knowledge enhanced graph convolutional network for graph-based semi-supervised learning. In *Proceedings of The Web Conference 2020*, 2020.

[9] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2012.