ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240667

Resilient Graph Neural Networks: A Coupled Dynamical Systems Approach

Moshe Eliasof[a,*,1], **Davide Murari**[b,**,1], **Ferdia Sherry**[a,***,1] **and Carola-Bibiane Schönlieb**[a,****]

[a]Department of Applied Mathematics and Theoretical Physics, University of Cambridge
[b]Department of Mathematical Sciences, Norwegian University of Science and Technology

Abstract. Graph Neural Networks (GNNs) have established themselves as a key component in addressing diverse graph-based tasks. Despite their notable successes, GNNs remain susceptible to input perturbations in the form of adversarial attacks. This paper introduces an innovative approach to fortify GNNs against adversarial perturbations through the lens of coupled dynamical systems. Our method introduces graph neural layers based on differential equations with contractive properties, which, as we show, improve the robustness of GNNs. A distinctive feature of the proposed approach is the simultaneous learned evolution of both the node features and the adjacency matrix, yielding an intrinsic enhancement of model robustness to perturbations in the input features and the connectivity of the graph. We mathematically derive the underpinnings of our novel architecture and provide theoretical insights to reason about its expected behavior. We demonstrate the efficacy of our method through numerous real-world benchmarks, reading on par or improved performance compared to existing methods.

1 Introduction

In recent years, the emergence of Graph Neural Networks (GNNs) has revolutionized the field of graph machine learning, offering remarkable capabilities for modeling and analyzing complex graph-structured data. These networks have found applications in diverse domains and applications, from Network Analysis [12, 31] and recommendation systems, Bioinformatics [11], Computer Vision [44], and more. However, the increasing prevalence of GNNs in critical decision-making processes has also exposed them to new challenges, particularly in terms of vulnerability to adversarial attacks.

In particular, it has been shown that one can design small adversarial perturbations of the input graph and its node features, that result in vastly different GNN predictions [45, 53]. Adversarial attacks received extensive attention in the context of Convolutional Neural Networks (CNNs) [24], but graph data has an added degree of freedom compared to data on a regular grid: the connectivity of the graph can be altered by adding or removing edges. Also, in natural settings, such as social network graphs, connectivity perturbations may be more realistically implementable by a potential adversary, rather than perturbations of the node features. This gives rise to hard discrete optimization problems, which necessitates the study of adversarial robustness specialized to graph data and GNNs [25].

In this paper we propose a GNN architecture that jointly processes the adversarially attacked adjacency matrix and node features by a learnable neural dynamical system. Our approach extends the active research front that aims to design neural architectures that enjoy inherent properties and behavior, drawing inspiration from dynamical systems with similar properties [26, 46, 8, 7, 41, 14, 6]. This approach has also been used to improve the robustness of CNNs, see SM:G. Specifically, the flow map of the coupled dynamical system under consideration in this work draws inspiration from the theory of non-Euclidean contractive systems developed in [4] to offer an adversarially robust GNN. We name our method CSGNN, standing for Coupled dynamical Systems GNN. Notably, because adjacency matrices are not arbitrary matrices, their learnable neural dynamical system needs to be carefully crafted to ensure it is node-permutation equivariant, and that it yields a symmetric adjacency matrix. To the best of our knowledge, this is the first attempt at learning coupled dynamical systems that evolve both the node features and the adjacency matrix.

Main contributions. This paper offers the following advances in adversarial defense against poisoning attacks in GNNs: (i) A novel architecture, CSGNN, that jointly evolves the node features and the adjacency matrix in a data-driven manner to improve GNN robustness to input perturbations, (ii) A theoretical analysis of our CSGNN, addressing the relevance of the architecture based on the theory of contractive dynamical systems and, (iii) Improved performance on various graph adversarial defense benchmarks.

Outline of the paper. Section 2 presents a literature overview. Section 3 introduces the problem of adversarial robustness. Section 4 presents and theoretically analyzes our methodology based on coupled contractive dynamical systems to improve the network's robustness. Section 5 includes a thorough experimental analysis of our CSGNN and compares its performance with previous works. Section 6 summarizes and discusses the obtained results. Proofs and further results are provided in the supplementary material that can be found in the full manuscript [15][2].

* Email: me532@cam.ac.uk.
** Email: davide.murari@ntnu.no.
*** Email: fs436@cam.ac.uk.
**** Email: cbs31@cam.ac.uk.
[1] Equal contribution.

[2] When referring to Appendix M, for example, we write SM:M.

2 Related Work

Graph Neural Networks as Dynamical Systems. Drawing inspiration from dynamical systems models that admit desired behaviors, various GNN architectures have been proposed to take advantage of such characteristic properties. In particular, building on the interpretation of CNNs as discretizations of PDEs [8, 38], there have been multiple works that view GNN layers as time integration steps of learnable non-linear diffusion equations. Such an approach allowed exploiting this connection to understand and improve GNNs [7, 14], for example by including energy-conserving terms alongside diffusion [14, 37] or using reaction-diffusion equations [10, 17], advection and convection systems [50, 16], and higher-order ODEs [18]. These approaches to designing GNNs have been shown to be of significant benefit when trying to overcome common issues such as over-smoothing [36, 5] and over-squashing [1]. Recently, it was shown that neural diffusion GNNs are robust to graph attacks [40]. The design of GNNs via contractive dynamical systems was also considered in [39, 21], but not in the context of adversarial robustness.

We note that deep learning architectures are also often harnessed to numerically solve ODEs and PDEs or discover such dynamical systems from data [33, 3]. However, in this paper, we focus on drawing links between GNNs and contractive dynamical systems to improve GNN robustness to adversarial attacks. Additionally, we remark that the use of coupled dynamical systems updating both the adjacency matrix and the feature matrix jointly has also been used in [28]. However, this paper focuses on dynamic graphs for data-driven modeling purposes, while the focus of this paper it on adversarial robustness. Additionally, the parameterization of the adjacency matrix updates differs considerably between our CSGNN and [28] both in structure and the number of required parameters.

Adversarial Defense in Graph Neural Networks. Various adversarial attack algorithms have been designed for graph data, notably including nettack [54], which makes local changes to targeted nodes' features and connectivity, and metattack [53], which uses a meta-learning approach with a surrogate model, usually a graph convolutional network (GCN) [31], to generate a non-targeted global graph attack and, recently [9] proposed a novel method to create graph injection attacks.

In response to these developments, significant efforts were made to design methods that improve GNN robustness. The majority of these approaches focus on perturbations of the graph connectivity, as those are more likely and practical in social network graph datasets. Several of these methods preprocess the graph based on underlying assumptions or heuristics, for example, dropping edges where node features are not similar enough, under the assumption that the true, non-attacked, graph is homophilic [47]. Another approach, in [19], suggests truncating the singular value decomposition of the adjacency matrix, effectively eliminating its high-frequency components, based on the assumption that adversarial attacks add high-frequency perturbations to the true adjacency matrix. The aforementioned approaches are unsupervised, and are typically added to existing GNN architectures while training them for a specific downstream task, such as node classification. Additionally, there are defenses that clean the attacked graph in a supervised manner, such as Pro-GNN [30], which solves a joint optimization problem for the GNN's learnable parameters, as well as for the adjacency matrix, with sparsity and low-rank regularization.

Besides methods for cleaning attacked adjacency matrices, there are also methods that aim to design robust GNN *architectures*. An example of this is given in [27], where the GCN architecture is modified to use a mid-pass filter, resulting in increased robustness. In this context, it is also natural to consider the use of Lipschitz constraints: given an upper bound on the Lipschitz constant of a classifier and a lower bound on its margin, we can issue robustness certificates [42]. This has been studied to some extent in the context of GNNs [29], although, in this case, and in contrast to our work, the Lipschitz continuity is studied only with respect to the node features. In our work, we also consider the Lipschitz continuity with respect to the adjacency matrix. It is worth noting that the development of a defense mechanism should ideally be done in tandem with the development of an adaptive attack, although designing an appropriate adaptive attack is not generally a straightforward task [35]. In [35], a set of attacked graphs are provided as "unit tests", which have been generated using adaptive attacks for various defenses. Therefore, in our experiments, we consider both standard, long-standing benchmarks, as well as recently proposed attacks in [35]. In recent works like [40, 51], the robustness of GNNs was studied through a *node feature* dynamical systems point of view. However, in our CSGNN, we propose to learn a coupled dynamical system that involves *node features as well as the adjacency matrix*.

3 Preliminaries

Notations. Let $G = (V, E)$ be a graph with n nodes V and m edges E, also associated with the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, such that $\mathbf{A}_{i,j} = 1$ if $(i, j) \in E$ and 0 otherwise, and let $\mathbf{f}_i \in \mathbb{R}^{c_{\text{in}}}$ be the input feature vector of the node $\mathbf{v}_i \in V$. In this paper, we focus on *poisoning* attacks, and we assume two types of possible attacks (perturbations) of the true data before training the GNN: (i) The features $\mathbf{f}_i$ are perturbed to $(\mathbf{f}_*)_i$, and, (ii) the adjacency matrix $\mathbf{A}$ of the graph is perturbed by adding or removing edges, inducing a perturbed adjacency matrix $\mathbf{A}_* \in \mathbb{R}^{n\times n}$. We denote by $G_* = (V_*, E_*)$ the attacked graph with the same vertices, i.e. $V = V_*$, by $\mathbf{A}_* \in \mathbb{R}^{n\times n}$ the perturbed adjacency matrix, and the perturbed node features are denoted by $(\mathbf{f}_*)_i$. We also denote by $\mathbf{F}, \mathbf{F}_* \in \mathbb{R}^{n\times c_{\text{in}}}$ the matrices collecting, as rows, the individual node features $\mathbf{f}_i, (\mathbf{f}_*)_i$. A table with these symbols and the rest of the notation is provided in SM:I.

Measuring graph attacks. To quantify the robustness of a GNN with respect to an adversarial attack, it is necessary to measure the impact of the attack. For node features, it is common to consider the Frobenius norm $\|\cdot\|_F$ to quantify the difference between the perturbed features $\mathbf{F}_*$ from the clean ones $\mathbf{F}$. However, the Frobenius norm is not a natural metric for adjacency attacks, see [2, 25] for example. Instead, it is common to measure the ℓ^0 distance between the true and attacked adjacency matrices, as follows:

$$\ell^0(\mathbf{A}, \mathbf{A}_*) = |\mathcal{I}(\mathbf{A}, \mathbf{A}_*)|, \tag{1}$$

where $\mathcal{I}(\mathbf{A}, \mathbf{A}_*) = \{i, j \in \{1, \dots, n\} : \mathbf{A}_{ij} \neq (\mathbf{A}_*)_{ij}\}$. For brevity, we refer to $\mathcal{I}(\mathbf{A}, \mathbf{A}_*)$ as $\mathcal{I}$, and by $|\mathcal{I}|$ we refer to the cardinality of $\mathcal{I}(\mathbf{A}, \mathbf{A}_*)$, as in Equation (1). The ℓ^0 distance measures how many entries of $\mathbf{A}$ need to be modified to obtain $\mathbf{A}_*$, and is typically used to measure budget constraints in studies of adversarial robustness of GNNs [25, 35]. For binary matrices, the ℓ^0 norm coincides with the ℓ^1 norm of the flattened version of the matrix $\text{vec}(\mathbf{A})$. We thus work with the ℓ^1 norm since it allows us to build contractive networks based on dynamical systems, and present some additional remarks on the connection it has with ℓ^0 in SM:C. Throughout this paper, we denote the perturbed node features by $\mathbf{F}_* = \mathbf{F} + \delta\mathbf{F}$, and the perturbed adjacency matrix by $\mathbf{A}_* = \mathbf{A} + \delta\mathbf{A}$, where $\|\delta\mathbf{F}\|_F \leq \varepsilon_1$, and $\|\text{vec}(\delta\mathbf{A})\|_1 \leq \varepsilon_2$.

In adversarial defense, the goal is to design a mechanism such that the output of the neural network is stable with respect to the perturbations $\delta\mathbf{F}$ and $\delta\mathbf{A}$, as formally described in the following definition.

Definition 3.1 ($(\varepsilon_1,\varepsilon_2)-$robust GNN)**.** Let $\mathcal{N}:\mathbb{R}^{n\times c_{\text{in}}}\times\mathbb{R}^{n\times n}\to\mathbb{R}^{n\times c_{\text{out}}}$ be a GNN. We say $\mathcal{N}$ to be $(\varepsilon_1,\varepsilon_2)-$robust if for every $j=1,\cdots,n$

$$\underset{i=1,\ldots,c_{\text{out}}}{\arg\max}\left(\mathcal{N}(\mathbf{F}+\delta\mathbf{F},\mathbf{A}+\delta\mathbf{A})\right)_j=\underset{i=1,\ldots,c_{\text{out}}}{\arg\max}\left(\mathcal{N}(\mathbf{F},\mathbf{A})\right)_j,$$

for every $(\delta\mathbf{F},\delta\mathbf{A})\in\mathbb{R}^{n\times c_{\text{in}}}\times\mathbb{R}^{n\times n}$ with $\|\delta\mathbf{F}\|_F\le\varepsilon_1$ and $\|\text{vec}\,(\delta\mathbf{A})\,\|_1\le\varepsilon_2$, where $\left(\mathcal{N}(\mathbf{F},\mathbf{A})\right)_j\in\mathbb{R}^{c_{\text{out}}}$ is the $j-$th row of $\mathcal{N}(\mathbf{F},\mathbf{A})$.

As discussed in Section 2, this goal is typically met either by modified architectures, training schemes, as well as their combinations. In Section 4, we present CSGNN - a defense mechanism based on a dynamical system perspective. This approach aims to reduce the sensitivity to input perturbations of the neural network and is based on the theory of contractive dynamical systems [4]. We will refer to a map as contractive with respect to a norm $\|\cdot\|$ if it is $1-$Lipschitz in such norm. Furthermore, we define contractive dynamical systems as those whose solution map is contractive with respect to the initial conditions. For completeness, in SM:A, we mathematically define and discuss contractive systems.

We start from the assumption that the best training accuracy on a given task corresponds to the clean inputs $(\mathbf{A},\mathbf{F})$*. The main idea of CSGNN is to jointly evolve the features* $\mathbf{F}_*$ *and the adjacency matrix* $\mathbf{A}_*$*, so that even if their clean versions* $\mathbf{F}$ *and* $\mathbf{A}$ *are not known, the network would output a vector measurably similar to the one corresponding to* $(\mathbf{A},\mathbf{F})$*, as we formulate in the following section. Instead of using heuristics to clean or modify the adjacency matrix, hence needing more knowledge on the attack, we process both the matrices in a data-driven manner.*

4 Method

4.1 Graph Neural Networks Inspired by Coupled Contractive Systems

We now present our CSGNN, focused on the task of robust node classification, where we wish to predict the class of each node in the graph, given attacked input data $(\mathbf{F}_*,\mathbf{A}_*)$. The goal is therefore to design and learn a map $\mathcal{D}:\mathbb{R}^{n\times c}\times\mathbb{R}^{n\times n}\to\mathbb{R}^{n\times c}\times\mathbb{R}^{n\times n}$, that evolves the node features, as well as the adjacency matrix. To the best of our knowledge, this is the first attempt at learning a *coupled dynamical system* that considers both the node features and the adjacency matrix in the context of graph-node classification. As discussed in Section 2, utilizing dynamical systems–perspective in GNNs was shown to provide strong inductive bias and more predictable behavior. However, existing defence methods often limit this interpretation to *node feature updates*, while using heuristics to pre-process the adjacency matrix, if desired. These heuristics assume further knowledge of the attack, hence motivating the use of a data-driven approach to design these updates, as we present in this section. Here, we advocate for the coupling of *node features and adjacency matrix* updates, in a principled, data-driven and dynamical system–based fashion. We focus on undirected graphs, but the proposed methodology can be adapted to directed ones with some modifications that we discuss throughout.

We implement the map $\mathcal{D}$ as a composition of learnable dynamical systems inspired by contractivity theory, that simultaneously update $\mathbf{F}_*$ and $\mathbf{A}_*$. Specifically, we model $\mathcal{D}$ as an approximation of the solution, at the final time T, of the continuous dynamical system:

$$\begin{cases}\dot{F}(t)=X(t,F(t),A(t))\in\mathbb{R}^{n\times c}\\ \dot{A}(t)=Y(t,A(t))\in\mathbb{R}^{n\times n},\\ (F(0),A(0))=(\mathcal{K}(\mathbf{F}_*),\mathbf{A}_*),\end{cases}\tag{2}$$

where $\dot{F}=\mathrm{d}F/\mathrm{d}t$ denotes the first order derivative in time, and $\mathcal{K}:\mathbb{R}^{c_{\text{in}}}\to\mathbb{R}^c$ is a linear embedding layer. Similarly to [26, 14, 7], we assume both X and Y to be piecewise constant in time, i.e., that on a given time interval $[0,T]$, there is a partition $0=\tau_0<\tau_1<\ldots<\tau_L=T$, $h_l=\tau_l-\tau_{l-1}$ for $l=1,\ldots,L$, such that $X(t,\mathbf{F},\mathbf{A})=X_l(\mathbf{F},\mathbf{A})$, $Y(t,\mathbf{A})=Y_l(\mathbf{A})$, $\mathbf{F}\in\mathbb{R}^{n\times c}$, $\mathbf{A}\in\mathbb{R}^{n\times n}$, $t\in[\tau_{l-1},\tau_l)$, for a pair of functions $X_l:\mathbb{R}^{n\times c}\times\mathbb{R}^{n\times n}\to\mathbb{R}^{n\times c}$, $Y_l:\mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}$. When referring to the approximation of $(F(\tau_l),A(\tau_l))$, we use $(\mathbf{F}^{(l)},\mathbf{A}^{(l)})$ when we start with the clean pair, and $(\mathbf{F}_*^{(l)},\mathbf{A}_*^{(l)})$ with the perturbed one. To obtain a neural network, we consider the solution of Equation (2) at time T, which is approximated using the explicit Euler method. More explicitly, we compose L explicit Euler layers each defined as $D_l((\mathbf{F},\mathbf{A}))\coloneqq(\Psi_{X_l}^{h_l}(\mathbf{F},\mathbf{A}),\Psi_{Y_l}^{h_l}(\mathbf{A}))$, $l=1,\ldots,L$, where $\Psi_{X_l}^{h_l}(\mathbf{F},\mathbf{A})\coloneqq\mathbf{F}+h_lX_l(\mathbf{F},\mathbf{A})$, and $\Psi_{Y_l}^{h_l}(\mathbf{A})\coloneqq\mathbf{A}+h_lY_l(\mathbf{A})$ are the explicit Euler steps for X_l and Y_l, respectively. The map $\mathcal{D}$ is then defined as the composition of L layers:

$$\mathcal{D}\coloneqq D_L\circ\ldots\circ D_1.\tag{3}$$

The coupled dynamical system encapsulated in $\mathcal{D}$ evolves both the hidden node features and the adjacency matrix for L layers. We denote the output of $\mathcal{D}$ by $(\mathbf{F}_*^{(L)},\mathbf{A}_*^{(L)})=\mathcal{D}((\mathcal{K}(\mathbf{F}_*),\mathbf{A}_*))$. To obtain node-wise predictions from the network to solve the downstream task, we feed the final GNN node features $\mathbf{F}_*^{(L)}$ to a classifier $\mathcal{P}:\mathbb{R}^c\to\mathbb{R}^{c_{\text{out}}}$, which is implemented by a linear layer. To better explain the structure of CSGNN, we provide an illustration in Figure 1 and a detailed feed-forward description in SM:J.

In what follows, we describe how to characterize the functions $\Psi_{X_l}^{h_l}$ and $\Psi_{Y_l}^{h_l}$ from Equation (3). First, in Section 4.2, we derive the node feature dynamical system governed by X. We show, that under mild conditions, contractivity can be achieved, allowing us to derive a bound on the influence of the attacked node features $\mathbf{F}_*$ on the GNN output. Second, in Section 4.3, we develop and propose a novel contractive dynamical system for the adjacency matrix, which is guided by Y.

Our motivation in designing such a coupled system stems from the nature of our considered adversarial settings. That is, we assume, that the adjacency matrix is perturbed. We note, that the adjacency matrix controls the propagation of node features. Therefore, leaving the input attacked adjacency unchanged may result in sub-par performance, as we show experimentally in SM:M. While some methods employ a pre-processing step of the attacked matrix $\mathbf{A}_*$ [19, 47], it has been shown that joint optimization of the node features and the adjacency matrix can lead to improved performance [30]. Therefore, we develop and study novel, coupled dynamical systems that evolve both the node features and the adjacency matrix and are learned in a data-driven manner. This perspective allows to obtain favorable properties such as adjacency matrix contractivity, thereby reducing the sensitivity to adversarial adjacency matrix attacks.

4.2 Contractive Node Feature Dynamical System

We now describe the learnable functions $\Psi_{X_l}^{h_l}$, $l=1,\ldots,L$, that determine the node feature dynamics of our CSGNN. We build upon

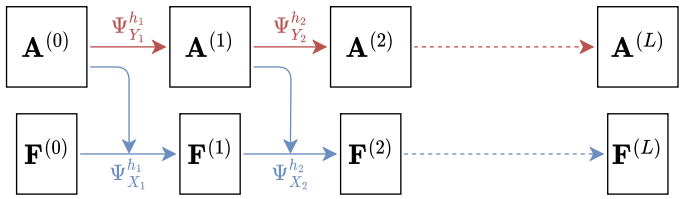


Figure 1: The coupled dynamical system $\mathcal{D}$ in CSGNN, as formulated in Equation (3).

a diffusion-based GNN layer, [7, 14], that is known to be stable and, under certain assumptions, is contractive. More explicitly, our proposed $\Psi^{h_l}_{X_l}$ is characterized as follows:

$$\begin{aligned}&\Psi^{h_l}_{X_l}(\mathbf{F}^{(l-1)}, \mathbf{A}^{(l-1)}) := \mathbf{F}^{(l)} \\ &:= \mathbf{F}^{(l-1)} + h_l X_l(\mathbf{F}^{(l-1)}, \mathbf{A}^{(l-1)}) \\ &= \mathbf{F}^{(l-1)} - h_l \left(\mathcal{G}^{(l-1)}\right)^\top \sigma\left(\mathcal{G}^{(l-1)}\mathbf{F}^{(l-1)}\mathbf{W}_l\right)\mathbf{W}_l^\top \tilde{\mathbf{K}}_l,\end{aligned} \tag{4}$$

where $\mathcal{G}^{(l-1)} := \mathcal{G}(\mathbf{A}^{(l-1)})$, while $\mathbf{W}_l \in \mathbb{R}^{c\times c}$ and $\tilde{\mathbf{K}}_l = (\mathbf{K}_l + \mathbf{K}_l^\top)/2 \in \mathbb{R}^{c\times c}$ are learnable parameters which allows a gradient flow interpretation of our system, as in [23]. The time steps $h_1, ..., h_L > 0$ are hyperparameters that are suitably constrained to ensure the network's contractivity according to our theoretical analysis. For each layer, we use the same step-size for the feature and adjacency updates. Also, as in [14], the map $\mathcal{G}(\mathbf{A}^{(l-1)}) : \mathbb{R}^{n\times c} \to \mathbb{R}^{n\times n\times c}$ is the gradient operator of $\mathbf{A}^{(l-1)}$ defined in SM:D, and we set $\sigma = \mathrm{LeakyReLU}$.

Theorem 4.1 (Equation (4) can induce stable node dynamics)**.** *Assume σ is a monotonically increasing 1-Lipschitz non-linear function. There are choices of $(\mathbf{W}_l, \mathbf{K}_l) \in \mathbb{R}^{c\times c}\times\mathbb{R}^{c\times c}$, for which the explicit Euler step in Equation (4) is stable for a small enough $h_l > 0$, i.e. there is a convex energy $\mathcal{E}_\mathbf{A}$ for which*

$$\mathcal{E}_\mathbf{A}(\Psi^{h_l}_{X_l}(\mathbf{F}^{(l-1)}, \mathbf{A})) \le \mathcal{E}_\mathbf{A}(\mathbf{F}^{(l-1)}),\ l = 1, \dots, L. \tag{5}$$

Theorem 4.2 (Equation (4) induces contractive node dynamics)**.** *Assume σ is a monotonically increasing 1-Lipschitz non-linear function. There are choices of $(\mathbf{W}_l, \mathbf{K}_l) \in \mathbb{R}^{c\times c}\times\mathbb{R}^{c\times c}$, for which the explicit Euler step in Equation (4) is contractive for a small enough $h_l > 0$, i.e. given any $\delta\mathbf{F} \in \mathbb{R}^{n\times c}$,*

$$\|\Psi^{h_l}_{X_l}(\mathbf{F} + \delta\mathbf{F}, \mathbf{A}) - \Psi^{h_l}_{X_l}(\mathbf{F}, \mathbf{A})\|_F \le \|\delta\mathbf{F}\|_F. \tag{6}$$

In SM:D we prove Theorems 4.1 and 4.2 for various parameterizations. One parameterization for which both theorems are satisfied corresponds to $\mathbf{K}_l = I_c$. We have experimented with this configuration, learning only $\mathbf{W}_l \in \mathbb{R}^{c\times c}$. We found that this configuration improves several baseline results, showing the benefit of contractive node feature dynamics. We report those results in SM:M. We also found, following recent interpretations of dissipative and expanding GNNs [23], that choosing the parameterization as $\mathbf{W}_l = I_c$, and training $\mathbf{K}_l \in \mathbb{R}^{c\times c}$ leads to further improved results, as we show in our experiments in Section 5 and SM:M. We note that this parameterization admits stable dynamical systems, in the sense of Theorem 4.1, as discussed in SM:D.

4.3 Contractive Adjacency Matrix Dynamical System

As previously discussed, our CSGNN learns both node features and adjacency matrix dynamical systems to defend against adversarial attacks. We now elaborate on the latter, aiming to design and learn dynamical systems with explicit Euler approximation of the solution $\Psi^{h_l}_{Y_l} : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$, $l = 1, \dots, L$, such that:

$$\begin{aligned}&\|\mathrm{vec}(\Psi^{h_l}_{Y_l}(\mathbf{A}^{(l-1)})) - \mathrm{vec}(\Psi^{h_l}_{Y_l}(\mathbf{A}^{(l-1)}_*))\|_1 \\ &\le \|\mathrm{vec}(\mathbf{A}^{(l-1)}) - \mathrm{vec}(\mathbf{A}^{(l-1)}_*)\|_1,\end{aligned} \tag{7}$$

where

$$\Psi^{h_l}_{Y_l}(\mathbf{A}^{(l-1)}) = \mathbf{A}^{(l)} = \mathbf{A}^{(l-1)} + h_l Y_l(\mathbf{A}^{(l-1)}). \tag{8}$$

In other words, we wish to learn maps $\Psi^{h_l}_{Y_l}$ that *decrease* the vectorized ℓ^1 distance between the true and attacked adjacency matrices, thereby reducing the effect of the adjacency matrix attack.

Since we are concerned with adjacency matrices, we need to pay attention to the structure of the designed map Y_l. Specifically, we demand that (i) the learned map Y_l are *node-permutation-equivariant*. That is, relabelling (change of order) of the graph nodes should not influence the dynamical system $\Psi^{h^l}_{Y_l}$ output up to its order, and, (ii) if the input graph is symmetric, then the updated adjacency matrix $\mathbf{A}^{(l)}$ should remain symmetric. Formally, the requirement (i) demands that:

$$\Psi^{h_l}_{Y_l}(\mathbf{P}\mathbf{A}\mathbf{P}^\top) = \mathbf{P}\Psi^{h_l}_{Y_l}(\mathbf{A})\mathbf{P}^\top \tag{9}$$

should hold for every permutation matrix $\mathbf{P} \in \{0, 1\}^{n\times n}$. The symmetry condition (ii) implies that we want $(\Psi^{h_l}_{Y_l}(\mathbf{A}))^\top = \Psi^{h_l}_{Y_l}(\mathbf{A})$ if $\mathbf{A}^\top = \mathbf{A}$. To this end, we adopt the derivations provided in [34, Appendix A], that show that in order to make the map $\Psi^{h_l}_{Y_l}$ permutation-equivariant and also symmetry preserving, we can set $Y_l(\mathbf{A}) = \sigma(M(\mathbf{A}))$ in Equation (8), where $\sigma : \mathbb{R} \to \mathbb{R}$ is any non-linear activation function applied componentwise, and $M : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ is a suitably designed linear map depending on a learnable vector $\mathbf{k} = (k_1, \cdots, k_9) \in \mathbb{R}^9$ and defined in SM:B. The design of Y_l allows us not only to design adjacency matrix updates with the inductive bias of coming from a dynamical system, but also respecting the expected symmetry and invariance of the resulting matrix. In case of directed graphs, one could adapt the procedure by removing the symmetry assumption, hence adding some more free parameters to the map M, as we comment on in SM:B. We now provide a theorem that validates the contractivity of the proposed adjacency matrix dynamical system, with its proof in SM:E.

Theorem 4.3 (Equation (8) defines contractive adjacency dynamics)**.** *Let $\alpha \le 0$, $\sigma : \mathbb{R} \to \mathbb{R}$ be a Lipschitz continuous function, with $\sigma'(s) \in [0, 1]$ almost everywhere. If $0 \le h_l \le 2/(2\sum_{i=2}^9 |k_i| - \alpha)$, then the explicit Euler step*

$$\Psi^{h_l}_{Y_l}(\mathbf{A}^{(l-1)}) := \mathbf{A}^{(l-1)} + h_l\sigma\left(M(\mathbf{A}^{(l-1)})\right), \tag{10}$$

where M is as in SM:B and $k_1 = \left(\alpha - \sum_{i=2}^9 |k_i|\right)$, is contractive in the vectorized ℓ^1 norm.

In our experiments, $\alpha \leq 0$ is a non-positive hyperparameter of the network. In this Section, we presented the contractive node and adjacency updates in our CSGNN that allow for reduced sensitivity to adversarial perturbed inputs in a learnable fashion with respect to the downstream performance of the task at hand, which is node classification in this paper. Notably, it is important to distinguish between extreme cases of contractivity, such as in the case of multi-layer perceptron (MLP), which holds no sensitivity to the adjacency matrix by design, and a network with *learnable* sensitivity through a contractive behavior, as presented in our CSGNN.

5 Experiments

We now study the effectiveness of CSGNN against different graph adversarial attacks. In Section 5.1 we discuss our experimental settings. Our choice of datasets and attacks comes from the interest in comparing our proposed methodology with previous papers presenting similar procedures to improve the network's robustness. The experimental analysis we propose focuses on poisoning based on modifying the true structure of the graph, by adding/removing edges between existent nodes or perturbing their node features. The presented mathematical setup is not limited to this class of attacks, but we focus on them so as to compare our performances to similar techniques for improving the network robustness. In Section 5.2, we report our experimental results and observations on several benchmarks, with additional results and an ablation study in SM:M. The results presented in Section 5.2 focuses on adjacency matrix attacks, which are the most popular in the literature [48]. To provide a comprehensive evaluation of our CSGNN, we also perform a set of experiments that include attacked node features with nettack. Since we follow the attacks evaluated in the literature, which utilize different training/validation/test splits for different types of attacks, the reported results for perturbation rate 0 can be different under different attacks.

5.1 Experimental settings

Datasets. Following [54, 53], we validate the proposed approach on four benchmark datasets, including three citation graphs, i.e., Cora, Citeseer, Pubmed, and one blog graph, Polblogs . The statistics of the datasets are shown in SM:K. Note that in the Polblogs graph, node features are not available. In this case, we follow Pro-GNN [30] and set the input node features to a $n \times n$ identity matrix.

Baselines. We demonstrate the efficacy of CSGNN by comparing it with popular GNNs and defense models, as follows: **GCN** [31]: Is one of the most commonly used GNN architectures, consisting of feature propagation according to the symmetric normalized Laplacian and channel mixing steps. **GAT** [43]: Graph Attention Networks (GAT) employ an attention mechanism to learn edge weights for the feature propagation step. **RGCN** [52]: RGCN models node features as samples from Gaussian distributions, and modifies GCN to propagate both the mean and the variance. In the neighborhood aggregation operation, high-variance features are down-weighted to improve robustness. **GCN-Jaccard** [47]: This is an unsupervised pre-processing method that relies on binary input node features, based on the assumption that the true graph is homophilic. Edges between nodes with features whose Jaccard similarity is below a certain threshold are removed. **GCN-SVD** [19]: GCN-SVD is also an unsupervised pre-processing step. Based on the observation that nettack tends to generate high-rank perturbations to the adjacency matrix, it is suggested to truncate the SVD of the adjacency matrix before it is used to train a GNN. **Pro-GNN** [30]: Pro-GNN attempts to jointly optimize GCN weights and a corrected adjacency matrix using a loss function consisting of a downstream supervised task-related loss function and low-rank and sparsity regularization. In Pro-GNN-fs, an additional feature smoothing regularization is used. **Mid-GCN** [27]: Mid-GCN modifies the standard GCN architecture to utilize a mid-pass filter, unlike the typical low-pass filter in GCN. **GNNGuard** [49]: GNNGuard modifies message-passing GNNs to include layer-dependent neighbor importance weights in the aggregation step. The neighbor importance weights are designed to favor edges between nodes with similar features, encoding an assumption of homophily. **GRAND** [20]: In this method, multiple random graph data augmentations are generated, which are then propagated through the GNN. The GNN is trained using a task-related loss and a consistency regularization that encourages similar outputs for the different augmented graphs. **Soft-Median-GDC** [22]: This approach first preprocesses the adjacency matrix using graph diffusion convolution [32], after which a GNN that uses soft median neighborhood aggregation function is trained. **GARNET** [13]: This method suggests wiring the graph using weighted spectral embeddings, which are shown to be related to the original, clean graph. **HANG** [51]: This approach is based on conservative Hamiltonian neural flows, used to process node features and for improved robustness. The comparisons with GNNGuard, GARNET and HANG are reported in SM:M.

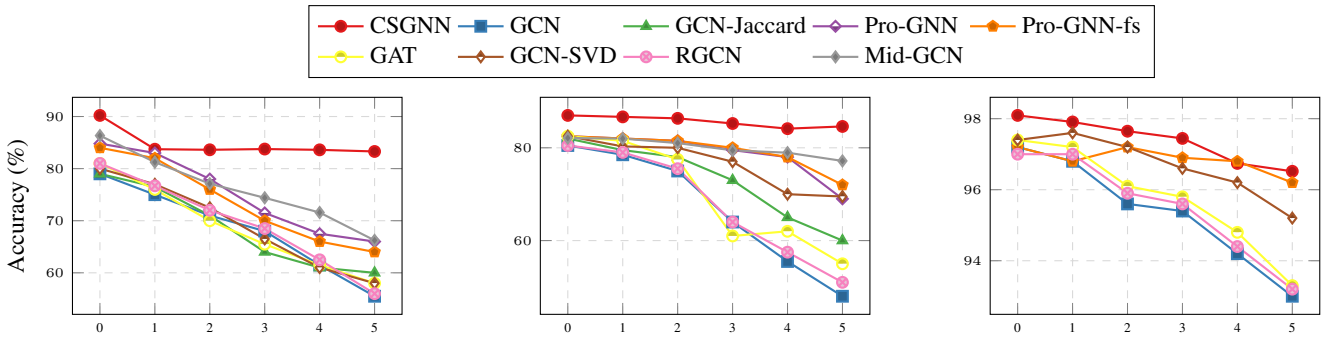
Training and Evaluation. We follow the same experimental settings as in [30]. Put precisely, and unless otherwise specified, for each dataset, we randomly choose 10% of the nodes for training, 10% of the nodes for validation, and the remaining 80% nodes for testing. For each experiment, we report the average node classification accuracy of 10 runs. The hyperparameters of all the models are tuned based on the validation set accuracy. In all experiments, the objective function to be minimized is the cross-entropy loss, using the Adam optimizer. Note that another benefit of our CSGNN is the use of downstream loss only, compared to other methods that utilize multiple losses to learn adjacency matrix updates. In SM:L, we discuss the hyperparameters of CSGNN. A complexity and runtime discussion is given in SM:N.

5.2 Adversarial Defense Performance

We evaluate the node classification performance of CSGNN against four types of poisoning attacks: (i) non-targeted attack, (ii) targeted attack, (iii) random attack, and, (iv) unit tests. Below we elaborate on the results obtained on each type of attack.

Robustness to Non-Targeted Adversarial Attacks. We evaluate the node classification accuracy of our CSGNN and compare it with the baseline methods after using the non-targeted adversarial attack metattack [53]. We follow the publicly available attacks and splits in [30]. We experiment with varying perturbation rates, i.e., the ratio of changed edges, from 0 to 25% with a step size of 5%. We report the average accuracy, as well as the obtained standard deviation over 10 runs in Table 1. The best-performing method is highlighted in bold. We can see that except for a few cases, our CSGNN consistently improves or offers on-par performance with other methods.

Robustness to Targeted Adversarial Attacks. In this experiment, we use nettack [54] as a targeted attack. Following [52], we vary the number of perturbations made on every targeted node from 1 to 5 with a step size of 1. The nodes in the test set with degree larger than 10 are set as target nodes. Here, we also use the publicly available splits in [30]. The node classification accuracy on target nodes is shown in Figure 2. From the figure, we can observe that when the number



(a) Cora (b) Citeseer (c) Polblogs

Figure 2: Node classification accuracy (%) under nettack. The horizontal axis describes the number of perturbations per node.

Table 1: Node classification performance (accuracy±std) under a non-targeted attack (metattack) with varying perturbation rates.

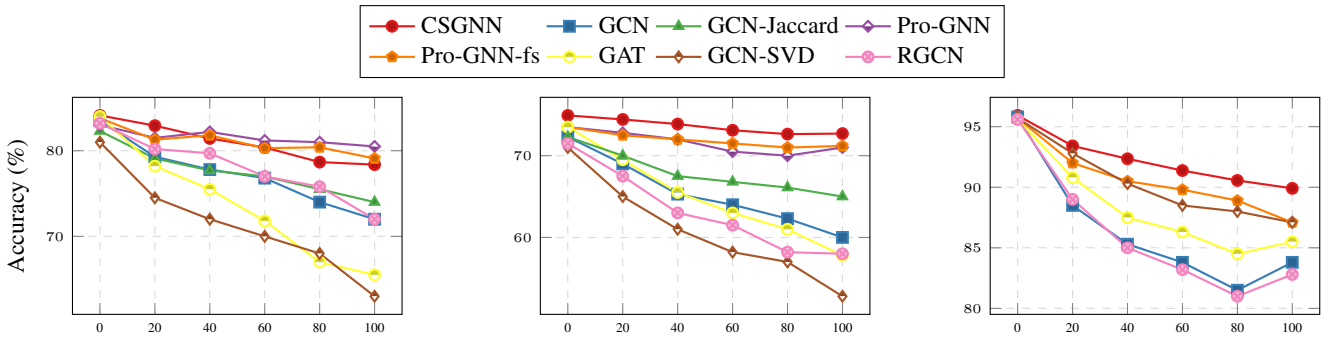
Dataset	Ptb Rate (%)	0	5	10	15	20	25
Cora	GCN	83.50±0.44	76.55±0.79	70.39±1.28	65.10±0.71	59.56±2.72	47.53±1.96
	GAT	83.97±0.65	80.44±0.74	75.61±0.59	69.78±1.28	59.94±0.92	54.78±0.74
	RGCN	83.09±0.44	77.42±0.39	72.22±0.38	66.82±0.39	59.27±0.37	50.51±0.78
	GCN-Jaccard	82.05±0.51	79.13±0.59	75.16±0.76	71.03±0.64	65.71±0.89	60.82±1.08
	GCN-SVD	80.63±0.45	78.39±0.54	71.47±0.83	66.69±1.18	58.94±1.13	52.06±1.19
	Pro-GNN-fs	83.42±0.52	82.78±0.39	77.91±0.86	76.01±1.12	68.78±5.84	56.54±2.58
	Pro-GNN	82.98±0.23	82.27±0.45	79.03±0.59	76.40±1.27	73.32±1.56	69.72±1.69
	Mid-GCN	**84.61±0.46**	**82.94±0.59**	80.14±0.86	77.77±0.75	76.58±0.29	72.89±0.81
	CSGNN	84.12±0.31	82.20±0.65	**80.43±0.74**	**79.32±1.04**	**77.47±1.22**	**74.46±0.99**
Citeseer	GCN	71.96±0.55	70.88±0.62	67.55±0.89	64.52±1.11	62.03±3.49	56.94±2.09
	GAT	73.26±0.83	72.89±0.83	70.63±0.48	69.02±1.09	61.04±1.52	61.85±1.12
	RGCN	71.20±0.83	70.50±0.43	67.71±0.30	65.69±0.37	62.49±1.22	55.35±0.66
	GCN-Jaccard	72.10±0.63	70.51±0.97	69.54±0.56	65.95±0.94	59.30±1.40	59.89±1.47
	GCN-SVD	70.65±0.32	68.84±0.72	68.87±0.62	63.26±0.96	58.55±1.09	57.18±1.87
	Pro-GNN-fs	73.26±0.38	73.09±0.34	72.43±0.52	70.82±0.87	66.19±2.38	66.40±2.57
	Pro-GNN	73.28±0.69	72.93±0.57	72.51±0.75	72.03±1.11	70.02±2.28	68.95±2.78
	Mid-GCN	74.17±0.28	74.31±0.42	73.59±0.29	73.69±0.29	71.51±0.83	69.12±0.72
	CSGNN	**74.93±0.52**	**74.91±0.33**	**73.95±0.35**	**73.82±0.61**	**73.01±0.77**	**72.94±0.56**
Polblogs	GCN	95.69±0.38	73.07±0.80	70.72±1.13	64.96±1.91	51.27±1.23	49.23±1.36
	GAT	95.35±0.20	83.69±1.45	76.32±0.85	68.80±1.14	51.50±1.63	51.19±1.49
	RGCN	95.22±0.14	74.34±0.19	71.04±0.34	67.28±0.38	59.89±0.34	56.02±0.56
	GCN-SVD	95.31±0.18	89.09±0.22	81.24±0.49	68.10±3.73	57.33±3.15	48.66±9.93
	Pro-GNN-fs	93.20±0.64	93.29±0.18	89.42±1.09	86.04±2.21	79.56±5.68	63.18±4.40
	CSGNN	**95.87±0.26**	**95.79±0.15**	**93.21±0.16**	**92.08±0.39**	**90.10±0.37**	**87.37±0.66**

of perturbations increases, the performance of CSGNN is better than other methods on the attacked target nodes in most cases.

Robustness to targeted attacks to node features and adjacency matrix. We now provide additional experiments, where not only the connectivity structure of the graph is attacked, but also the node features, demonstrated on the Cora and Citeseer datasets. To generate the attacked versions of these datasets, we follow the same protocol as in Pro-GNN [30]. The attacks are based on nettack [54], which applies a targeted attack to the test nodes of the clean graph having a degree larger than 10. To attack all these nodes, we iterate through the target nodes and iteratively update the feature and adjacency matrices attacking the previously obtained one at the next target node. We work with different attack intensities, applying 1 to 5 perturbations per targeted node, with a step of 1. The results are reported in Figure 5, where we compare the performance of CSGNN, with those of GCN, GCN-SVD, and Pro-GNN. As we can see, CSGNN outperforms all of the compared models on this task.

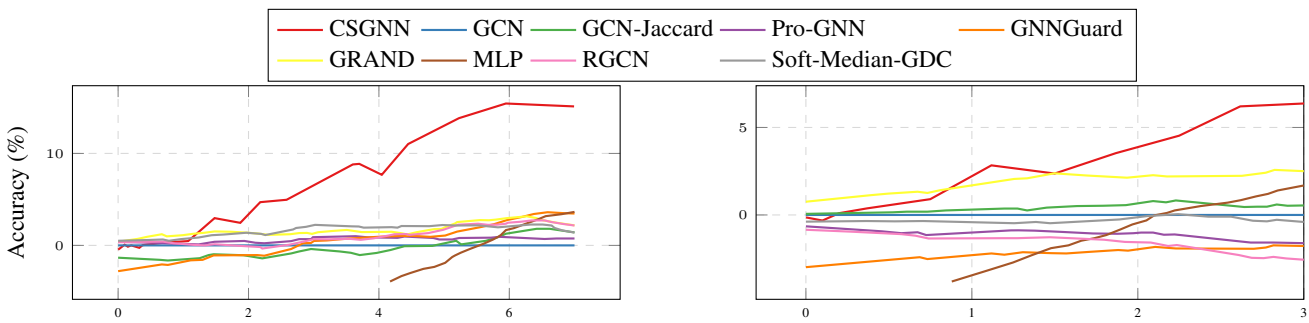
Robustness to Random Attacks. In this experimental setting, we evaluate the performance of CSGNN when the adjacency matrix is attacked by adding random fake edges, from 0% to 100% of the number of edges in the true adjacency matrix, with a step size of 20%. The results are reported in Figure 3. It can be seen, that CSGNN is on par with or better than the considered baselines.

Unit tests. We utilize the recently suggested *unit tests* from [35]. This is a set of perturbed citation datasets, which are notable for the fact that the perturbations were not generated using standard attack generation procedures that focus only on attacks like nettack or metattack. Instead, 8 adversarial defense methods were studied. Then, bespoke, adaptive attack methods were designed for each of them. These attack methods were applied to the citation datasets to generate the "unit tests". We experiment with those attacks as they offer a challenging benchmark, that further highlights the contribution of our CSGNN. We present the results in Figure 4, showing the relative performance of CSGNN and other baselines compared to GCN. We see that our CSGNN performs better than other considered models.



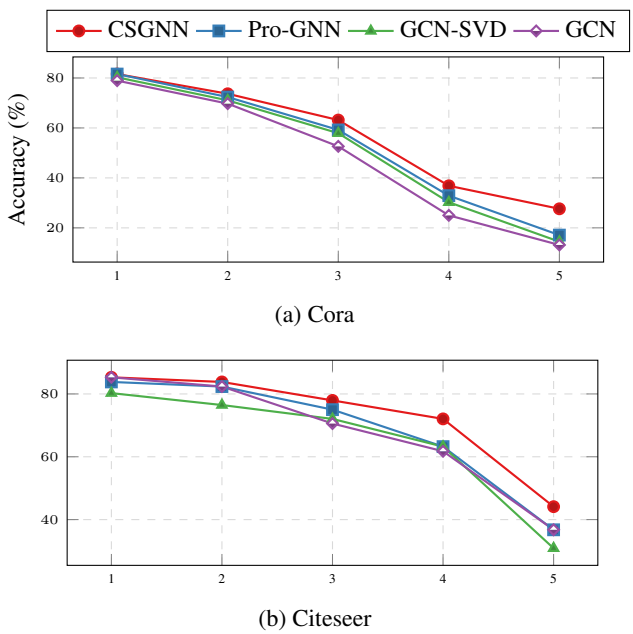
(a) Cora (b) Citeseer (c) Polblogs

Figure 3: Node classification accuracy (%) under a random adjacency matrix attack. The horizontal axis describes the attack percentage.



(a) Cora-ML (b) Citeseer

Figure 4: Node classification accuracy (%) using unit-tests from [35] relative to a baseline GCN. The horizontal axis shows the attack budget (%).



(a) Cora

(b) Citeseer

Figure 5: Node classification accuracy (%) under targeted attack with nettack to both node features and adjacency matrix. The horizontal axis describes the number of perturbations per node.

This result further highlights the robustness of CSGNN under different adversarial attack scenarios, on several datasets. In Figure 8 of SM:M, we also provide absolute performance results.

6 Summary and Discussion

In this paper, we present CSGNN, a novel GNN architecture inspired by contractive dynamical systems for graph adversarial defense. Our CSGNN learns a coupled dynamical system that updates both the node features as well as the adjacency matrix to reduce the impact of input perturbations, thereby defending against graph adversarial attacks. We provide a theoretical analysis of our CSGNN, to gain insights into its characteristics and expected behavior. Our profound experimental study of CSGNN reveals the importance of employing the proposed coupled dynamical system to reduce attack influence on the model's accuracy. Namely, our results verify both the efficacy compared to existing methods, as well as the necessity of each of the dynamical systems in CSGNN. Since our approach presents a novel way to model both the node features and adjacency matrix through the lens of dynamical systems, we believe that our findings and developments will find further use in graph adversarial defense and attacks, as well as other applications of GNNs where being stable to input perturbations is relevant.

References

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

[2] A. Bojchevski and S. Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *International Conference on Machine Learning*, pages 695–704. PMLR, 2019.

[3] J. Brandstetter, D. E. Worrall, and M. Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022.

[4] F. Bullo. *Contraction Theory for Dynamical Systems*. Kindle Direct Publishing, 1.1 edition, 2023. ISBN 979-8836646806.

[5] C. Cai and Y. Wang. A note on over-smoothing for graph neural networks. *arXiv:2006.13318*, 2020.

[6] E. Celledoni, D. Murari, B. Owren, C.-B. Schönlieb, and F. Sherry. Dynamical Systems–Based Neural Networks. *SIAM Journal on Scientific Computing*, 45(6):A3071–A3094, 2023.

[7] B. P. Chamberlain, J. Rowbottom, M. Gorinova, S. Webb, E. Rossi, and M. M. Bronstein. GRAND: Graph neural diffusion. In *International Conference on Machine Learning (ICML)*, pages 1407–1418, 2021.

[8] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems 31: NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018.

[9] Y. Chen, H. Yang, Y. Zhang, K. Ma, T. Liu, B. Han, and J. Cheng. Understanding and improving graph injection attack by promoting unnoticeability. *arXiv:2202.08057*, 2022.

[10] J. Choi, S. Hong, N. Park, and S.-B. Cho. GREAD: Graph neural reaction-diffusion equations. *arXiv:2211.14208*, 2022.

[11] G. Corso, H. Stärk, B. Jing, R. Barzilay, and T. S. Jaakkola. Diffdock: Diffusion steps, twists, and turns for molecular docking. In *The Eleventh International Conference on Learning Representations*, 2023.

[12] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[13] C. Deng, X. Li, Z. Feng, and Z. Zhang. Garnet: Reduced-rank topology learning for robust and scalable graph neural networks. In *Learning on Graphs Conference*, pages 3–1. PMLR, 2022.

[14] M. Eliasof, E. Haber, and E. Treister. PDE-GCN: Novel architectures for graph neural networks motivated by partial differential equations. *Advances in Neural Information Processing Systems*, 34:3836–3849, 2021.

[15] M. Eliasof, D. Murari, F. Sherry, and C.-B. Schönlieb. Contractive Systems Improve Graph Neural Networks Against Adversarial Attacks. *arXiv:2311.06942*, 2023.

[16] M. Eliasof, E. Haber, and E. Treister. Feature transportation improves graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11874–11882, 2024.

[17] M. Eliasof, E. Haber, and E. Treister. Graph neural reaction diffusion models. *arXiv:2406.10871*, 2024.

[18] M. Eliasof, E. Haber, E. Treister, and C.-B. B. Schönlieb. On the temporal domain of differential equation inspired graph neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 1792–1800. PMLR, 2024.

[19] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis. All you need is low (rank): defending against adversarial attacks on graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 169–177, 2020.

[20] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang. Graph random neural networks for semi-supervised learning on graphs. *Advances in neural information processing systems*, 33:22092–22103, 2020.

[21] C. Gallicchio and A. Micheli. Graph Echo State Networks. In *The 2010 international joint conference on neural networks*, pages 1–8. IEEE.

[22] S. Geisler, T. Schmidt, H. Sirin, D. Zügner, A. Bojchevski, and S. Günnemann. Robustness of graph neural networks at scale. In *Advances in Neural Information Processing Systems 34: NeurIPS 2021, December 6-14, 2021, virtual*, pages 7637–7649, 2021.

[23] F. D. Giovanni, J. Rowbottom, B. P. Chamberlain, T. Markovich, and M. M. Bronstein. Understanding convolution on graphs via energies. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

[24] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015*, 2015.

[25] S. Günnemann. Graph Neural Networks: Adversarial Robustness. In L. Wu, P. Cui, J. Pei, and L. Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 149–176. Springer Nature Singapore, Singapore, 2022.

[26] E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, Dec. 2017.

[27] J. Huang, L. Du, X. Chen, Q. Fu, S. Han, and D. Zhang. Robust mid-pass filtering graph convolutional networks. In *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 328–338. ACM, 2023.

[28] Z. Huang, Y. Sun, and W. Wang. Coupled graph ode for learning interacting system dynamics. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2021.

[29] Y. Jia, D. Zou, H. Wang, and H. Jin. Enhancing node-level adversarial defenses by Lipschitz regularization of graph neural networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, page 951–963, 2023.

[30] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 66–74, 2020.

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

[32] J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems 32: NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13333–13345, 2019.

[33] Z. Long, Y. Lu, X. Ma, and B. Dong. PDE-net: Learning PDEs from data, 2018.

[34] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman. Invariant and equivariant graph networks. *ICLR*, 2018.

[35] F. Mujkanovic, S. Geisler, S. Günnemann, and A. Bojchevski. Are Defenses for Graph Neural Networks Robust? *Advances in Neural Information Processing Systems*, 35:8954–8968, Dec. 2022.

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

[37] T. K. Rusch, B. Chamberlain, J. Rowbottom, S. Mishra, and M. Bronstein. Graph-coupled oscillator networks. In *International Conference on Machine Learning*, pages 18888–18909. PMLR, 2022.

[38] L. Ruthotto and E. Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62: 352–364, 2020.

[39] 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, 2008.

[40] Y. Song, Q. Kang, S. Wang, K. Zhao, and W. P. Tay. On the robustness of graph neural diffusion to topology perturbations. *Advances in Neural Information Processing Systems*, 35:6384–6396, 2022.

[41] M. Thorpe, T. M. Nguyen, H. Xia, T. Strohmer, A. Bertozzi, S. Osher, and B. Wang. GRAND++: Graph neural diffusion with a source term. In *International Conference on Learning Representations*, 2022.

[42] Y. Tsuzuku, I. Sato, and M. Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. *Advances in neural information processing systems*, 31, 2018.

[43] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[44] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph CNN for learning on point clouds. *arXiv:1801.07829*, 2018.

[45] M. Waniek, T. P. Michalak, M. J. Wooldridge, and T. Rahwan. Hiding individuals and communities in a social network. *Nature Human Behaviour*, 2(2):139–147, 2018.

[46] E. Weinan. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11, Mar. 2017.

[47] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu. Adversarial examples for graph data: Deep insights into attack and defense. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4816–4823, 7 2019.

[48] Y. Xu, M. Lanier, A. Sarkar, and Y. Vorobeychik. Attacks on node attributes in graph neural networks. *arXiv:2402.12426*, 2024.

[49] X. Zhang and M. Zitnik. GNNGuard: Defending graph neural networks against adversarial attacks. In *Advances in Neural Information Processing Systems 33: NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[50] K. Zhao, Q. Kang, Y. Song, R. She, S. Wang, and W. P. Tay. Graph neural convection-diffusion with heterophily. *arXiv:2305.16780*, 2023.

[51] K. Zhao, Q. Kang, Y. Song, R. She, S. Wang, and W. P. Tay. Adversarial robustness in graph neural networks: A Hamiltonian approach. *Advances in Neural Information Processing Systems*, 36, 2024.

[52] D. Zhu, Z. Zhang, P. Cui, and W. Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1399–1407, 2019.

[53] D. Zügner and S. Günnemann. Adversarial attacks on graph neural networks via meta learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9*, 2019.

[54] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2847–2856, 2018.