AI Open 5 (2024) 216–223

Contents lists available at ScienceDirect

AI Open

journal homepage: www.keaipublishing.com/en/journals/ai-open

Full Length Article

Generating graph perturbations to enhance the generalization of GNNs

Sofiane Ennadir [a,*], Giannis Nikolentzos [b], Michalis Vazirgiannis [a,c], Henrik Boström [a]

[a] EECS, KTH Royal Institute of Technology, Stockholm, Sweden
[b] University of Peloponnese, Tripoli, Greece
[c] LIX, École Polytechnique, Paris, France

ARTICLE INFO

Keywords:
Generalization
Graph neural networks
Data augmentation

ABSTRACT

Graph neural networks (GNNs) have become the standard approach for performing machine learning on graphs. Such models need large amounts of training data, however, in several graph classification and regression tasks, only limited training data is available. Unfortunately, due to the complex nature of graphs, common augmentation strategies employed in other settings, such as computer vision, do not apply to graphs. This work aims to improve the generalization ability of GNNs by increasing the size of the training set of a given problem. The new samples are generated using an iterative contrastive learning procedure that augments the dataset during the training, in a task-relevant approach, by manipulating the graph topology. The proposed approach is general, assumes no knowledge about the underlying architecture, and can thus be applied to any GNN. We provided a theoretical analysis regarding the equivalence of the proposed approach to a regularization technique. We demonstrate instances of our framework on popular GNNs, and evaluate them on several real-world benchmark graph classification datasets. The experimental results show that the proposed approach, in several cases, enhances the generalization of the underlying prediction models reaching in some datasets state-of-the-art performance.

1. Introduction

Graph-structured data is ubiquitous in many domains such as in chemoinformatics, bioinformatics and social network analysis. The multitude of machine learning applications in these domains motivated the development of neural network models that can operate on graphs, known as Graph Neural Networks (GNNs). In the past years, GNNs have emerged as a powerful tool for learning both node and graph representations. These models have been applied to several different problems including drug design (Kearnes et al., 2016) and session-based recommendation (Wu et al., 2019).

Similarly to other Neural Networks, GNNs are known to require significant amounts of training data to achieve high-performance levels. However, in many application domains, such as bioinformatics, only a limited amount of training data is available, where for instance only a small number of protein functions may be known. Such scenarios pose significant challenges to GNNs as they may suffer from overfitting, leading to poor generalization to new, unseen data. Additionally, GNNs are trained under the assumption that both the training and test data originate from the same distribution, which is not always true in practice. Consequently, GNNs may fail to generalize to out-of-distribution samples, further exacerbating the challenge of poor generalization. Therefore, it is essential to develop techniques that can enhance the generalization capacity of GNNs, especially in scenarios with limited training data.

Such issues have already been identified and addressed in the field of computer vision. An effective possible approach to enhancing the generalization capacity of the model is to capitalize on data augmentation techniques, which are very common in computer vision (Shorten and Khoshgoftaar, 2019). The main idea is to artificially inflate the training set with label preserving transformations. In the case of image data, increasing the size of the training set is rather straightforward since one can apply geometric and color-based augmentations such as translation, reflection and rotation, or can change the image's color palette. Generalizing these techniques to other types of data such as text in the context of natural language processing (Feng et al., 2021) and graphs remains an important open challenge. Data augmentation techniques for GNNs are limited. Traditionally, one can produce new graph data by applying perturbations to existing graphs such as adding and/or dropping nodes and/or edges (Rong et al., 2020). Moreover, in some cases, where the nodes and edges are annotated with multidimensional feature vector, it is possible to produce new samples by also editing the features of the nodes and edges. However, the selection of the appropriate nodes and edges to modify or remove remains a significant

* Corresponding author.
E-mail address: ennadir@kth.se (S. Ennadir).

https://doi.org/10.1016/j.aiopen.2024.10.001
Received 15 August 2023; Received in revised form 9 August 2024; Accepted 9 October 2024
Available online 24 October 2024

challenge for the success of these techniques, and the validity of the augmented graph.

Recently, and following the advancements in understanding adversarial examples (Goodfellow et al., 2015), adversarial data augmentation (Ganin et al., 2016; Sinha et al., 2017) has emerged as a promising approach to generating artificial examples. The approach capitalizes on the adversarial loss and aims to produce perturbations that will lead to misbehavior of a model, such as misclassification of the output. While these perturbations have been mainly used to improve the model's robustness via re-training, they can also be seen as a data augmentation to enhance generalization (Volpi et al., 2018; Zhao et al., 2020).

In this study, we present a framework for enhancing the generalization and robustness of GNNs in graph classification settings. Specifically, we leverage an adversarial approach to generate additional training samples by perturbing existing ones. We demonstrate that our approach is equivalent to incorporating a Lipschitz-like regularization, and provide theoretical analysis to support our claim. Our framework is agnostic to the choice of GNN architecture, and can be easily applied on top of any existing model. We evaluate our method on three different GNNs, and demonstrate its effectiveness on a range of standard graph classification datasets. Our results indicate that the proposed framework consistently outperforms both baseline GNN models and other existing augmentation techniques in terms of classification accuracy. Our main contributions are summarized as follows:

- We propose a general framework for increasing the generalization of GNNs which can be seen as a data augmentation approach. We optimize the proposed framework in an adversarial setting, where we alternately train a GNN model and a generator component which generates new samples by injecting perturbations to existing samples. We establish a theoretical foundation for our approach by demonstrating its equivalence to a regularization technique.
- The framework assumes no knowledge about the underlying architecture, thus allowing us to demonstrate it on three popular GNN models.
- We evaluate the proposed framework on several benchmark datasets from chemoinformatics, bioinformatics and social networks. In most cases, the proposed framework improves the generalization performance of the base GNN models.

2. Related work

Enhancing the generalization of Graph Neural Networks through data augmentation have been studied recently in the literature. Edge perturbation methods, such as DropEdge (Rong et al., 2020) and AdaEdge (Chen et al., 2019), have been proposed to randomly remove or iteratively add/remove edges based on label predictions. While randomly adding/removing edges have had some success, recently, the work (Zhao et al., 2021) proposed to substitute the random edit in the previously Citepd methods by using an edge predictor to select the appropriate edges to be edited such as to maximize an introduced reward. The mixup paradigm, which has demonstrated success in enhancing neural network generalization in computer vision tasks, has also been adapted for GNNs. GraphMix (Wang et al., 2021) proposed various mixup techniques for both node and graph classification that generate new samples by interpolating diverse graphs in the semantic space or by interpolating the generator of different classes of graphs. Similarly, G-Mixup (Han et al., 2022) has proposed an improved mixup paradigm for GNNs that leverages graphons to generate synthetic graphs through sampling. The approach, instead of directly manipulating graphs, interpolates graphons of different classes in the Euclidean space allowing therefore the possibility to leverage multiple input graphs.

Our proposed framework generates new samples by injecting perturbations to the graph structure, which relates it closely to the literature on adversarial attacks on graph-structured data (Dai et al., 2018; Zügner et al., 2018; Bojchevski and Günnemann, 2019; Zügner and Günnemann, 2019). Adversarial attack settings have been defined in recent years based on the attacker's goal and knowledge, and our approach is more geared towards a gray-box setting where the internal structure of the model being attacked is not known. One relevant approach in the literature is Nettack, which generates perturbations to the graph structure and nodes' features with the goal of causing misclassification of specific nodes. Another approach is Mettack, which perturbs the input graph as a bilevel optimization problem to compromise node classification performance, using meta-gradients for optimization. This work have been expanded and enhanced to the black-box setting, where Zhan and Pei (2021) proposed a gradient-based black-box attacking algorithm to overcome a number of identified limitation in Mettack. Reinforcement learning has also been used in adversarial attacking algorithms, as in Dai et al. (2018), where a reinforcement learning-based approach learns to modify the graph structure (adding or dropping edges) with only the feedback from the target classifier.

Recently, injecting perturbations as a data augmentation technique to enhance generalization has gained attention, particularly for node features. Recently, the work (Kong et al., 2020) proposed FLAG, an iterative approach that uses gradient-based adversarial perturbations to augment node features during training. However, our work differs from this approach as we do not target node features but rather we aim to augment in the semantic space by generating new graphs from scratch. Finally, it is worth noting that unlike the proposed mixup techniques including GraphMix (Wang et al., 2021) and G-Mixup (Han et al., 2022), which merges two graphs, our proposed method is more general and unlimited since it creates a new graph at each training step that is semantically similar to an existing one in the training set but has a different prediction.

3. Preliminaries

Before continuing with our contribution, we begin by introducing notation and some fundamental concepts.

Notation and Problem Setup. Let $G = (V, E)$ be a graph where V is its set of vertices and E its set of edges. We will denote by $n = |V|$ and $m = |E|$ the number of vertices and number of edges, respectively. Let $\mathcal{N}(v)$ denote the set of neighbors of a node $v \in V$, i.e., $\mathcal{N}(v) = \{u : (v, u) \in E\}$. The degree of a node is equal to its number of neighbors, i.e., equal to $|\mathcal{N}(v)|$ for a node $v \in V$. A graph is commonly represented by its adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ which encodes edge information. The (i, j)th element of the adjacency matrix is equal to the weight of the edge between the ith and jth node of the graph and a weight of 0 in case the edge does not exist. In some settings, the nodes of a graph might be annotated with feature vectors. We use $\mathbf{X} \in \mathbb{R}^{n \times D}$ to denote the node features where D is the feature dimensionality. The feature of the ith node of the graph corresponds to the ith row of $\mathbf{X}$.

GNNs. A GNN model consists of a series of neighborhood aggregation layers which use the graph structure and the nodes' feature vectors from the previous layer to generate new representations for the nodes. Specifically, GNNs update nodes' feature vectors by aggregating local neighborhood information. Suppose we have a GNN model that contains T neighborhood aggregation layers. Let also $\mathbf{h}_v^{(0)}$ denote the initial feature vector of node v, i.e., the row of matrix $\mathbf{X}$ that corresponds to node v. At each iteration ($t > 0$), the hidden state $\mathbf{h}_v^{(t)}$ of a node v is updated as follows:

$$\mathbf{a}_v^{(t)} = \text{AGGREGATE}^{(t)}\left(\left\{\mathbf{h}_u^{(t-1)} : u \in \mathcal{N}(v)\right\}\right)$$

$$\mathbf{h}_v^{(t)} = \text{COMBINE}^{(t)}\left(\mathbf{h}_v^{(t-1)}, \mathbf{a}_v^{(t)}\right)$$

where AGGREGATE is a permutation invariant function that maps the feature vectors of the neighbors of a node v to an aggregated vector. This aggregated vector is passed along with the previous representation of v (i.e., $\mathbf{h}_v^{(t-1)}$) to the COMBINE function which combines those two vectors and produces the new representation of v.

After T iterations of neighborhood aggregation, to produce a graph-level representation, GNNs apply a permutation invariant readout function (e.g., sum operator, mean operator) to the feature vectors of all nodes of the graph as follows:

$$\mathbf{h}_G = \text{READOUT}\Big(\{\mathbf{h}_v^{(T)} : v \in V\}\Big)$$

Adversarial attacks. Let $G \in \mathcal{G}$ be a graph that is given as input to a GNN classifier $f : \mathcal{G} \to \mathcal{Y}$. Suppose that the attacker has limited knowledge of f (i.e., black-box attack). The goal of an adversarial attack is to find a graph $\tilde{G} = (\tilde{V}, \tilde{E})$ slightly different from the original graph with its predicted class being different from the predicted class of G. This could be formulated as finding $\tilde{G}$ subject to $d(G, \tilde{G}) \leq \epsilon$ and $f(G) \neq f(\tilde{G})$, where d is a function that measures distance of graphs such as some matrix norm of the difference of their aligned adjacency matrices $|| \mathbf{A} - \tilde{\mathbf{A}} ||$ (e.g., l_∞, l_2 or l_0).

4. Graph perturbations for GNN robustness

In this section, we propose a new framework for improving the generalization of GNNs. The proposed framework is a data augmentation technique which increases the size of the training set by learning graph perturbations.

4.1. Uncertainty-based augmentation

Given three measurable spaces with a defined norm over each space $(\mathcal{G}, \|.\|_{\mathcal{G}})$, $(\mathcal{X}, \|.\|_{\mathcal{X}})$ and $(\mathcal{Y}, \|.\|_{\mathcal{Y}})$. Let $\{(A_1, X_1, y_1), \ldots, (A_N, X_N, y_N)\} \in (\mathcal{G}, \mathcal{X}, \mathcal{Y})^N$ be a sampled set from an underlying probability distribution $\mathcal{D}$ defined on $(\mathcal{G}, \mathcal{X}, \mathcal{Y})$. Graph classification aims to train a graph-based classifier $f : (\mathcal{G}, \mathcal{X}) \to \mathcal{Y}$ minimizing the risk of f w.r.t $\mathcal{D}$ as follows:

$$\min_{\theta \in \Theta} \mathbb{E}_{(A,X,y)\sim\mathcal{D}}[\mathcal{L}(A, X, y; \theta)], \tag{1}$$

where θ represents the model's parameters and $\mathcal{L}$ is the classification loss function (e.g., cross-entropy loss). Given that we usually do not have access to the underlying distribution $\mathcal{D}$, the training paradigm consists therefore of minimizing an unbiased estimator of the risk quantity :

$$\min_{\theta \in \Theta} \hat{R}[f_\theta] := \min_{\theta \in \Theta} \frac{1}{N} \sum_{i=0}^{N} \mathcal{L}(A_i, X_i, y_i ; \theta)$$

It has been shown that there exists a link between adversarial robustness and standard generalization performance (Tsipras et al., 2019). Specifically, adversarial training on bounded perturbations can enhance a model's predictive performance. We are interested in investigating whether the above scenario also applies to the task of graph classification by generating perturbations for which the model is less certain in its predictions. As described in Section 3, an adversarial attack involves identifying a slightly modified graph that is semantically similar to the input graph but results in a different output from the model. To this end, to quantify the semantic similarity, we define the following graph distance in our input metric spaces:

$$d_{\alpha,\beta}([A, X], [\tilde{A}, \tilde{X}]) = \min_{P \in \Pi} \{\alpha \| A - P\tilde{A}P^T \|_2 + \beta \| X - P\tilde{X} \|_2\},$$

where Π is the set of permutation matrices and α, β are hyper-parameters. Notably, for un-attributed graphs, this distance corresponds to the widely used concept of edit distance on graphs, which measures the similarity between two graphs by computing the minimum number of edge edits required to transform one graph into another, while considering graph isomorphism. We note that while we are considering the l_2 in our graph distance formulation, any other defined measure on the input space can be used.

In this work, we propose to augment the training dataset using adversarially-crafted samples in the structural space. The process is equivalent to editing the training procedure to the following:

$$\min_{\theta \in \Theta} \hat{R}_A[f_\theta] := \min_{\theta \in \Theta} \frac{1}{N} \sum_{i=0}^{N} \max_{[\tilde{A}_i, X_i] \in B([A_i, X_i]; \epsilon)} \mathcal{L}(\tilde{A}_i, X_i, y_i; \theta),$$

with $B([A_i, X_i]; \epsilon) = \{(\tilde{A}_i, X_i) \mid d_{\alpha,0}([A_i, X_i], [\tilde{A}_i, X_i]) < \epsilon\}$ for any budget $\epsilon \geq 0$. We consider that this introduced risk quantity is equivalent to the classical one. The following theorem shows that our approach is similar to introducing a gradient-norm regularization without explicitly calculating the gradient.

Theorem 1. *Giving a loss function $\mathcal{L}$ that is second-order differentiable with respect to A. By considering structural-based augmentation, we have the following:*

$$\hat{R}_A[f_\theta] = \hat{R}[f_\theta] + \frac{c_\epsilon}{N} \sum_{i=0}^{N} \|(\nabla_{A_i} \mathcal{L}\{A_i, X_i, y_i\})\|_{1,1} + \mathcal{O}(\epsilon^2),$$

with c_ϵ being a constant dominated by ϵ, ie. $c_\epsilon = \Theta(\epsilon)$.

The obtained result confirms that our proposed adaptation approach can be viewed as an additional Lipschitz-like regularization of the gradient, which encourages smoothness in the loss function with respect to the input (Zhao et al., 2022). Specifically, penalizing the gradient norm of the loss function promotes a smaller Lipschitz constant in local regions. A lower Lipschitz constant suggests that the loss function landscape is flatter, potentially leading to improved model generalization. We additionally note that the weight of the regularization is controlled by the parameter ϵ, which corresponds to the size of the considered neighborhood, based on the chosen distance metric. It is worth noting that while our focus is on structural-based perturbations, the same principle and proof can be extended to node feature-based augmentations, providing a theoretical understanding of the recently proposed techniques FLAG (Kong et al., 2020). The proof of the theorem is provided in Appendix A.1.

4.2. Proposed architecture

Based on the previously introduced adaptation, we propose PERTURB, a graph data augmentation based on the following assumption:

Assumption 1. Let G be a graph and let y be the class label associated to G. Let $\mathbf{A}$ denote the adjacency matrix of G. If we apply a small perturbation to G to obtain graph $\tilde{G}$, i.e., $\tilde{\mathbf{A}} = \mathbf{A} + \boldsymbol{\Delta}$ where $\tilde{\mathbf{A}}$ is the adjacency matrix of $\tilde{G}$ and $\|\boldsymbol{\Delta}\| < \epsilon$ for some small ϵ, then $\tilde{G}$ belongs to the same class as G, i.e., $\tilde{y} = y$ where $\tilde{y}$ is the class label of $\tilde{G}$.

Although the above assumption is not always true, in most real-world scenarios it is valid. For instance, if the graph representations of two proteins are very similar to each other, it is very likely that the two proteins have the same function.

Given a trained GNN model, the proposed approach aims to generate a set of perturbed graphs $\{\tilde{G}_1, \ldots, \tilde{G}_M\}$ which emerge from the set of training graphs that are correctly classified by the GNN. Those graphs are produced by applying a set of perturbation matrices $\{\boldsymbol{\Delta}_1, \boldsymbol{\Delta}_2, \ldots, \boldsymbol{\Delta}_M\}$ to the correctly classified graphs. The norm of each perturbation matrix $\boldsymbol{\Delta}_i$ is smaller than ϵ, and therefore, based on our assumption above, the class label of the associated graph is identical to the one from which it emerged. The generated perturbed graphs lead the already trained GNN model to misbehavior, misclassification of the output in our setting, i.e., $f(\tilde{G}) \neq f(G)$ where G is the graph from which $\tilde{G}$ emerged and f is a model that classifies graphs.

Based on the above, it is clear that the proposed framework consists of two components: (1) a classifier, i.e., a GNN model; (2) a graph generator which produces new graphs by injecting perturbations to existing graphs. For the second component, we devise an encoder–decoder-like architecture. Fig. 1 illustrates the proposed framework, while in what follows, we give more details about the two components of the framework.

(1) **Classifier.** This is an instance of a GNN model following the general framework presented in the Preliminaries section. The main

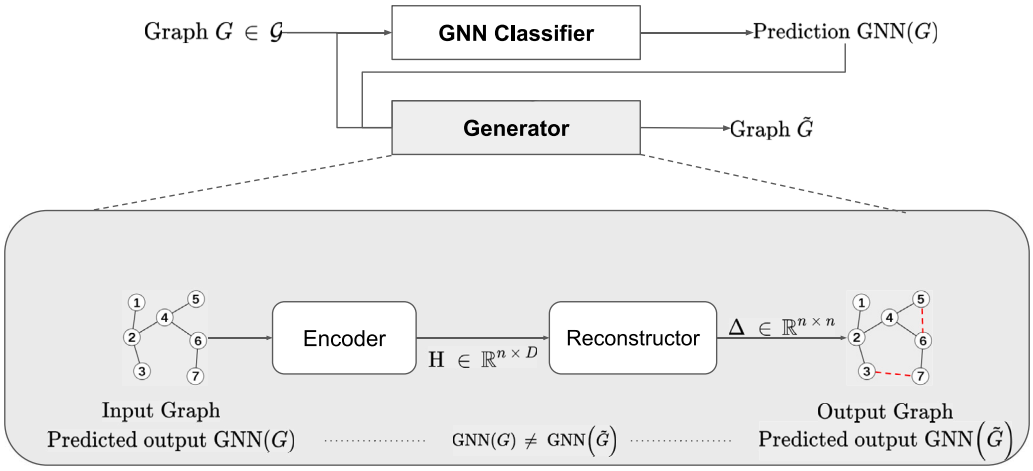

Fig. 1. Illustration of the proposed framework for increasing the robustness of GNNs. The framework consists of two main components : (1) A GNN model; (2) A generator, consisting of an encoder and a reconstructor, which is responsible for producing perturbation matrices that will result into new samples. Once the new samples are generated, the GNN is re-trained on the new dataset.

objective of the proposed approach is to increase the robustness of this model. This is performed in an iterative fashion. At each step, the model is trained on a different training set using a problem-specific loss function $\mathcal{L}$ (e.g., cross-entropy loss).

(2) **Generator.** This module is a graph auto-encoder-like architecture which takes a graph G as input and produces a perturbation matrix. This perturbation matrix is applied to G and results in the generation of a new graph $\tilde{G}$. More specifically, the generation of new samples is a two-step approach. First, the encoder part takes the adjacency matrix $\mathbf{A}$ and the matrix of features $\mathbf{X}$ of the input graph G, and embedds the nodes of G into a D-dimensional vector space. The encoder could be an instance of the general family of GNN models presented in the Preliminaries Section 3, which produces a matrix $\mathbf{H}^{(T)} \in \mathbb{R}^{n \times D}$ where T denotes the number of neighborhood aggregation steps and the rows of $\mathbf{H}^{(T)}$ store the representations of the different nodes of G (we will omit the superscript (T) henceforth). Then, the second part of the generator produces a perturbation matrix $\boldsymbol{\Delta}$ by computing the inner product between the embeddings of the nodes, i.e., $\boldsymbol{\Delta} = \mathbf{H}\,\mathbf{H}^{\top}$. Finally, to generate the new sample, the perturbation matrix $\boldsymbol{\Delta}$ is added to the adjacency matrix of the input graph G, and the ReLU function is applied to the resulting matrix, thus we have $\tilde{\mathbf{A}} = ReLU(\mathbf{A} + \boldsymbol{\Delta})$ where $\tilde{\mathbf{A}}$ is the adjacency matrix of the new graph $\tilde{G}$. Since the ReLU function is applied to the above sum, $\tilde{\mathbf{A}}$ contains no negative values, and it corresponds to the adjacency matrix of a weighted graph (edge weights can take any positive value). Furthermore, since $\boldsymbol{\Delta}$ is a symmetric matrix, $\tilde{G}$ is an undirected graph in case G is undirected. In case of directed graphs, two different representations could be learned for each node, one for the inward and one for the outward links.

4.3. Training

The proposed approach, PERTURB, iteratively increases the size of the training set based on the above idea. Let K denote the total number of iterations. Let also $\mathcal{G}^{(0)}$ denote the initial training set and $\tilde{\mathcal{G}}^{(k)} = \{\tilde{G}_1, \tilde{G}_2, \ldots, \tilde{G}_{M_k}\}$ denote the set of perturbed graphs generated at the kth iteration of the algorithm. Then, for the $(k+1)$th iteration, the size of the training set is increased by adding the new graphs that were generated at the kth iteration and the new training set is defined as $\mathcal{G}^{(k+1)} = \mathcal{G}^{(k)} \cup \tilde{\mathcal{G}}^{(k)}$. The classifier (i.e., GNN model) is re-trained on the new training set. Interestingly, the generator and the classifier can be seen as two players that compete with each other in a game. The generator is responsible for generating new examples that the discriminator (the classifier) will potentially misclassify, while the classifier becomes more robust and increases its task-related classification ability by being trained to correctly classify the generated graphs. The two models are trained together, and after K iterations, this game will allow the GNN model to be adversarially robust and to deal successfully with samples coming from distributions different from those of the training samples. The main steps of the proposed approach are illustrated in Algorithm 1.

Algorithm 1 Framework to Enhance the Robustness of GNNs

1: **Input:** Training set $(\mathcal{G}^{(0)}, \mathcal{Y})$ where $\mathcal{G}^{(0)} = \{G_1, \ldots, G_N\}$; pre-trained classifier GNN$^{(0)}$; number of iterations K
2: **Output:** Classifier GNN$^{(K)}$
3: **for** $k = 1$ **to** K **do**
4: **for** $G_i \in \mathcal{G}^{(0)}$ **do**
5: Generate a perturbed graph $\tilde{G}_i = \text{GEN}(G_i)$ based on the current model GNN$^{(k-1)}$
6: **end for**
7: Increment the training set $\mathcal{G}^{(k)} = \mathcal{G}^{(k-1)} \cup \{\tilde{G}_i : i \in N\}$
8: Re-train the model on $\mathcal{G}^{(k)}$ to obtain GNN$^{(k)}$
9: **end for**

Given a graph, the generator's objective is to learn useful node representations that will allow it to generate a new graph that is similar to the input graph and for which the classification model is not very confident in its prediction. Given a GNN model already trained in a classification setting, let GNN$(\mathbf{A}, \mathbf{X})$ denote the output of the neural network (probability that the input graph G represented by its adjacency matrix $\mathbf{A}$ and matrix of node features $\mathbf{X}$ belongs to the different classes). Let also $\boldsymbol{\Delta} = \text{GEN}(\mathbf{A}, \mathbf{X})$ denote the perturbation matrix produced by the generator.

To train the generator, we minimize the following objective function:

$$\min_{\theta} \rho\Big(\text{GNN}\big(\mathbf{A}, \mathbf{X}\big), \text{GNN}\big(\tilde{\mathbf{A}}, \mathbf{X}\big)\Big) + \lambda \|\boldsymbol{\Delta}\|_1, \tag{2}$$

where θ denotes the parameters of the generator and $\tilde{\mathbf{A}} = ReLU(\mathbf{A} + \boldsymbol{\Delta})$. The main objective behind the second term of the loss function is to control the amount of perturbation the generator is allowed to add, and therefore, to control the similarity of the generated graph and the input graph where λ is a regularization coefficient. We use the l_1 norm to promote sparsity in the produced perturbations, i.e., to limit the number of modified edges. With regards to the first term of the loss function, it aims to make the generator produce graphs for which the classification model is less certain in its predictions. For example, in the case of binary classification, if graph G (that belongs to class 1) is classified as positive, we would like the generated graph $\tilde{G}$ to be

classified as negative. Therefore, $\rho : \mathbb{R}^c \times \mathbb{R}^c \to \mathbb{R}$ can be any comparison function satisfying this objective where c is the number of classes. For instance, in the case of binary classification tasks, a potential function could be:

$$\rho\Big(\mathrm{GNN}(\mathbf{A},\mathbf{X}),\ \mathrm{GNN}(\tilde{\mathbf{A}},\mathbf{X})\Big) = 1 - y - \Big[\mathrm{GNN}(\tilde{\mathbf{A}},\mathbf{X})\Big]_{1-y},$$

where y is the class label of the correctly classified graph G, and $\Big[\mathrm{GNN}(\tilde{\mathbf{A}},\mathbf{X})\Big]_{1-y}$ denotes the $(1-y)$th component of the vector of probabilities, i.e., the predicted probability of class $1-y$.

The previous function can be extended to the multi-class classification setting by increasing the model's output of the second most probable class. This leads to the following formulation:

$$\rho\Big(\mathrm{GNN}(\mathbf{A},\mathbf{X}),\ \mathrm{GNN}(\tilde{\mathbf{A}},\mathbf{X})\Big) = 1 - \Big[\mathrm{GNN}(\tilde{\mathbf{A}},\mathbf{X})\Big]_{h},$$

where $h = \arg\max_{i \neq y}\Big[\mathrm{GNN}(\mathbf{A},\mathbf{X})\Big]_i$ is the class with second highest probability.

5. Experimental evaluation

In this section, we first give details about the experimental settings. We next describe the empirical evaluation applied on real-world datasets. We afterwards report on the performance of the approach in comparison to other available data augmentation methods. We last analyzed the performance of our proposed architecture in generating relevant adversarial augmentation.

5.1. Experimental setup

We used a Graph Convolutional Network (GCN) (Kipf and Welling, 2017) for our generator's encoder model where the number of message passing layers and hidden dimensions have been arranged differently from the model that we are attempting to improve (especially in the GCN case) since we assume no knowledge about the inner architecture of the underlying model.

We demonstrate instances of the proposed framework on two main GNNs: (1) Graph Convolutional Network (GCN) (Kipf and Welling, 2017) and (2) Graph Isomorphism Network (GIN) (Xu et al., 2019) on real benchmark datasets. We experiment the proposed approach, PERTURB, on standard graph classification datasets derived from bioinformatics and chemoinformatics (MUTAG, PROTEINS, NCI1, D&D), and from social networks (IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, COLLAB) (Morris et al., 2020). Note that the social network graphs are unlabeled, while all other graph datasets come with either node labels or node attributes. We take those labels/attributes into account when available. For the unlabeled datasets, we considered the node degrees as node features. To evaluate the different models, we use the framework proposed by Errica et al. (2020). Thus, we perform 10-fold cross-validation to obtain an estimate of the generalization performance of each method, while within each fold a model is selected based on a 90%/10% split of the training set. We used the same folds as provided by the original work. For all models, we used the sum operator as the readout function to produce graph-level representations. Furthermore, we train the different models by minimizing the cross-entropy loss function with the Adam optimizer and an initial learning rate of 10^{-3}.

We first compare the instances of the proposed approach against the GNN models from which they were derived, where we aim to show the framework's ability to enhance their robustness. We take into account other state-of-the-art available GNNs: (1) DGCNN (Zhan and Pei, 2021); (2) DiffPool (Ying et al., 2018); (3) GraphSAGE (Hamilton et al., 2017); (4) ECC (Simonovsky and Komodakis, 2017).

We conducted a comparison between our proposed approach and existing benchmark graph data augmentation techniques aimed at improving the generalization of Graph Neural Networks (GNNs). We evaluated our approach against DropEdge (Rong et al., 2020), a method that randomly removes a certain ratio of edges from the input graph to prevent over-fitting and over-smoothing. We also considered DropNode (Do et al., 2021; You et al., 2020), a similar technique that randomly removes a certain portion of nodes and their connections as a form of regularization. Additionally, we compared our method to two mixup-based techniques: Manifold-mixup paradigm (Wang et al., 2021), which interpolates graph-level embeddings after the Readout function, and G-mixup (Han et al., 2022), which interpolates graphons of different classes in the Euclidean space instead of manipulating graphs directly. We used the same hyper-parameters as the original works for each method to ensure a fair comparison, and further details about the implementation and experimental setup are provided in Appendix C.

Table 1
Classification accuracy (±standard deviation) of the different approaches on the 8 graph classification datasets. OOR means Out of Resources, either time (>72 h for a single training) or GPU memory. Best performance per dataset in **bold**. Best augmentation method for each model is highlighted using the underline.

Dataset	MUTAG	D&D	NCI1	PROTEINS
DGCNN	84.0 (±6.7)	76.6 (±4.3)	76.4 (±1.7)	72.9 (±3.5)
DiffPool	79.8 (±7.1)	75.0 (±3.5)	76.9 (±1.9)	73.7 (±3.5)
ECC	75.4 (±6.2)	72.6 (±4.1)	76.2 (±1.4)	72.3 (±3.4)
GraphSAGE	83.6 (±1.7)	72.9 (±2.0)	76.0 (±1.8)	73.0 (±4.5)
GCN	82.2 (±6.3)	75.4 (±3.1)	75.7 (±2.3)	73.3 (±3.6)
GCN + DropEdge	82.4 (±6.7)	75.8 (±3.4)	75.9 (±2.0)	73.8 (±3.1)
GCN + DropNode	83.6 (±5.9)	76.1 (±3.6)	76.1 (±2.3)	74.1 (±3.6)
GCN + MixUp	82.6 (±5.8)	76.3 (±2.8)	76.5 (±2.1)	74.3 (±3.5)
GCN + G-MixUp	84.2 (±6.1)	75.8 (±3.5)	76.1 (±2.8)	74.6 (±4.2)
GCN + PERTURB	84.6 (±6.4)	**77.6** (±3.8)	75.9 (±2.5)	**75.4** (±3.5)
GIN	84.3 (±5.4)	74.9 (±4.8)	77.8 (±1.6)	72.2 (±4.1)
GIN + DropEdge	84.5 (±8.3)	75.0 (±5.3)	77.4 (±2.2)	72.6 (±4.0)
GIN + DropNode	84.2 (±6.9)	75.3 (±6.3)	77.1 (±2.3)	72.1 (±3.3)
GIN + MixUp	84.9 (±6.0)	76.2 (±2.9)	**78.5** (±2.0)	72.9 (±3.5)
GIN + G-MixUp	85.2 (±5.8)	76.8 (±2.3)	78.1 (±1.8)	72.7 (±2.9)
GIN + PETURB	**85.3** (±4.6)	76.2 (±4.3)	77.6 (±1.4)	72.5 (±4.5)
Dataset	IMDB-B	IMDB-M	REDDIT-B	COLLAB
DGCNN	69.2 (±3.0)	45.6 (±3.4)	87.8 (±2.5)	71.2 (±1.9)
DiffPool	68.4 (±3.3)	45.6 (±3.4)	**89.1** (±1.6)	68.9 (±2.0)
ECC	67.7 (±2.8)	43.5 (±3.1)	OOR	OOR
GraphSAGE	68.8 (±4.5)	47.6 (±3.5)	84.3 (±1.9)	73.9 (±1.7)
GCN	70.1 (±1.6)	48.0 (±3.1)	83.5 (±2.8)	66.4 (±1.4)
GCN + DropEdge	70.5 (±2.6)	48.3 (±3.7)	84.2 (±4.6)	67.1 (±1.8)
GCN + DropNode	70.0 (±3.4)	47.7 (±2.9)	83.9 (±3.8)	67.3 (±2.1)
GCN + MixUp	70.6 (±1.8)	48.8 (±3.5)	84.9 (±3.4)	67.8 (±1.4)
GCN + G-MixUp	70.9 (±3.8)	**50.3** (±2.2)	86.5 (±4.7)	67.7 (±2.2)
GCN + PERTURB	71.2 (±2.4)	49.6 (±2.8)	86.9 (±2.2)	67.5 (±0.7)
GIN	70.6 (±3.6)	47.1 (±3.2)	79.1 (±4.3)	74.8 (±1.6)
GIN + DropEdge	71.2 (±4.8)	47.2 (±2.8)	79.9 (±5.4)	74.8 (±2.3)
GIN + DropNode	71.0 (±3.2)	46.9 (±3.5)	79.5 (±3.1)	75.1 (±2.8)
GIN + MixUp	70.8 (±3.2)	48.2 (±3.2)	80.2 (±5.1)	**75.9** (±2.2)
GIN + G-MixUp	70.9 (±2.1)	48.3 (±3.5)	80.7 (±3.8)	75.6 (±1.9)
GIN + PERTURB	**71.4** (±2.4)	47.9 (±3.8)	81.3 (±2.8)	75.3 (±2.9)

5.2. Results

We start by evaluating the effectiveness of our proposed generator architecture by generating graphs using the used datasets and compare them to the original graphs. Fig. 2 illustrates examples of generated graphs from the PROTEINS datasets. As desired, the generated graphs share a number of similar semantic proprieties with the original graphs while successfully being able to achieve our adversarial aim. Hence, we can consider that the proposed generator architecture is successfully achieving its generation aim.

Table 1 illustrates average prediction accuracies and the corresponding standard deviations. We observe that the proposed framework can enhance the performance of the underlying model in the majority of cases. Therefore, the experimental results gives a first insight to verify our claim that the proposed framework can increase the generalization of GNNs. Additionally, the experimental results clearly

Table 2
Classification accuracy (±standard deviation) of GCN, the gradient-based framework (GM) and the proposed framework (PERTURB) on the D&D, PROTEINS, NCI1 and MUTAG datasets. Best performance per dataset in **bold**.

Dataset	D&D	PROTEINS	NCI1	MUTAG
GCN	75.4 (±3.1)	73.3 (±3.6)	75.7 (±2.3)	82.2 (±6.3)
GCN + GM	76.1 (±3.4)	74.3 (±3.2)	75.4 (±2.9)	82.8 (±8.3)
GCN + PERTURB	**77.6 (±3.8)**	**75.4 (±3.5)**	**75.9 (±2.5)**	**84.6 (±6.4)**

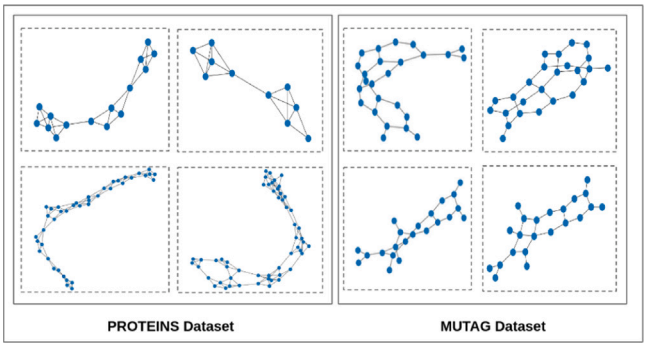

Fig. 2. *(left)* Examples of generated graphs from the PROTEINS dataset *(right)* Examples of generated graphs from the MUTAG dataset. These examples are generated by perturbing existing graphs in the dataset and for which the output classification is different from the original output ($f(G) \neq f(\tilde{G})$).

demonstrate that PERTURB outperforms both DropEdge and DropNode methods, especially in the case of non-attributed graphs. The improvements achieved by PERTURB over these random-based baseline methods are significant, highlighting the effectiveness of our approach. Furthermore, our experimental evaluation shows that PERTURB performs better or closely to both considered mixup-based approaches. Overall, our proposed framework performs the best results in the majority of cases, achieving superior performance in eight out of sixteen cases. These findings suggest that PERTURB is not only effective compared to existing baselines but also has certain advantages over approaches based on the mixup paradigm.

We would like to point out that for these considered real-world datasets, the test samples are considered to be sampled from the same distribution as the training samples. As a result, the potential impact of our proposed approach may be more pronounced for other types of datasets. To further show the effectiveness of our framework for this potential type of datasets, we conducted out-of-distribution experiments using synthetically generated datasets, where the assumption of train/test samples drawn from the same distribution is not met (detailed results are presented in Appendix C.3). Moreover, we can see that the increase in accuracy is significant on some datasets (e.g., D&D, PROTEINS, REDDIT-BINARY), while only minor or no improvements were observed for others. Intuitively, we believe this is due to the fact that on those datasets node attributes might be more important for the classification task than the graph structure itself. In fact, this observation have been empirically verified in the findings reported in previous studies (Errica et al., 2020). Furthermore, we observed that our proposed framework provides greater improvements to the GCN model than to the GIN model, and achieves state-of-the-art results on 5 out of 8 datasets.

5.3. Additional analysis

In this section, our aim is to assess the effectiveness of our proposed architecture in generating adversarial-based augmentation compared to other available attack methods. The majority of the previous work (discussed in the related work section), treats the adversarial setting as an optimization problem using some gradient-based method or other greedy approaches. However, these approaches are not the most appropriate for the graph classification task since they do not consider the structural aspect of graphs. On the other hand, the generator component of the proposed framework is well-suited to this setting since it can extract useful information from both the graph's structure and the node features. Next, we compare the proposed framework against a gradient-based method (GM) similar to the work of Zhan and Pei (2021), where we use the proximal gradient descent (PGD) algorithm to generate the perturbations. We use the same objective function and the same value for hyperparameter λ as in the case of the proposed approach. For the PGD algorithm, we computed the process for a sufficient number of epochs to ensure reaching the minimum and hence a useful perturbation. Table 2 reports the average prediction accuracies and standard deviations of the two approaches and the underlying model. We observe that our method outperforms the gradient-based method in terms of accuracy.

6. Conclusion

In this paper, we proposed a framework for improving the generalization ability and robustness of GNNs. This is achieved by increasing the size of the training set of a given task. The new samples are generated by applying perturbations to existing graphs. Those perturbations are not randomly produced, but the proposed approach learns how to generate them such that the emerging graphs can be useful for increasing the model's generalization ability. The proposed approach was applied to two popular GNNs, and its effectiveness was empirically tested on several standard graph classification datasets. The proposed framework showed improved performance in most cases compared to the baseline GNN models and other benchmark graph augmentation techniques. One direction for future research is to introduce a data augmentation technique for graphs combining both structural perturbations and node feature perturbations.

CRediT authorship contribution statement

Sofiane Ennadir: Conceptualization, Data curation, Methodology, Software, Writing – original draft. **Giannis Nikolentzos:** Methodology, Supervision, Writing – review & editing. **Michalis Vazirgiannis:** Supervision. **Henrik Boström:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) (Grant No. 71563 (ECS-JHK)) funded by the Knut and Alice Wallenberg Foundation, Sweden.

Appendix A

A.1. Proof of Theorem 1

Theorem 1. *Giving a loss function $\mathcal{L}$ that is second-order differentiable with respect to $G \in \mathcal{G}$. By considering structural-based augmentation, we have the following:*

$$\hat{R}_A[f_\theta] = \hat{R}[f_\theta] + \frac{c_\epsilon}{N} \sum_{i=0}^{N} \|(\nabla_{A_i} \mathcal{L}\{A_i, X_i, y_i\})\|_{1,1} + \mathcal{O}(\epsilon^2),$$

with c_ϵ being a constant dominated by ϵ, ie. $c_\epsilon = \Theta(\epsilon)$.

Table 3
Mean training time analysis (in s) of the GCN+PERTURB and GIN+PERTURB in comparison to the GCN and GIN instances.

Dataset	GCN	GCN+PERTURB
DD	38.81	112.73
NCI1	52.34	119.93
PROTEINS	23.52	86.43
Dataset	GIN	GIN+PERTURB
DD	43.18	119.33
NCI1	54.82	124.79
PROTEINS	22.35	81.96

Proof. Let us consider the introduced risk quantity $\hat{R}_A$. We have the following:

$$\hat{R}_A[f_\theta] = \frac{1}{N}\sum_{i=0}^{N} \max_{G_i' \in B(G_i,\epsilon)} \mathcal{L}\{\tilde{A}_i, X_i, y_i\}$$
$$= \frac{1}{N}\sum_{i=0}^{N} \mathcal{L}\{A_i + \Delta_i, X_i, y_i\}$$
$$= \frac{1}{N}\sum_{i=0}^{N} \mathcal{L}\{A_i + \Delta_i, X_i, y_i\} + \mathcal{L}\{A_i, X_i, y_i\} - \mathcal{L}\{A_i, X_i, y_i\}$$
$$= \frac{1}{N}\sum_{i=0}^{N} \mathcal{L}\{A_i, X_i, y_i\} + \frac{1}{N}\sum_{i=0}^{N} \mathcal{L}\{A_i + \Delta_i, X_i, y_i\} - \mathcal{L}\{A_i, X_i, y_i\}$$
$$= \hat{R}[f_\theta] + \frac{1}{N}\sum_{i=0}^{N} \mathcal{L}\{A_i + \Delta_i, X_i, y_i\} - \mathcal{L}\{A_i, X_i, y_i\}$$

Giving the loss $\mathcal{L}$ being differentiable, we have:

$$\mathcal{L}\{A_i + \Delta_i, X_i, y_i\}$$
$$= \mathcal{L}\{A_i, X_i, y_i\} + Tr((\nabla_{A_i}\mathcal{L}\{A_i, X_i, y_i\})^T \Delta_i) + \mathcal{O}(\epsilon^2)$$

Consequently, we have the following result:

$$\hat{R}_A[f_\theta] = \hat{R}[f_\theta] + \frac{c_\epsilon}{N}\sum_{i=0}^{N} \|(\nabla_{A_i}\mathcal{L}\{A_i, X_i, y_i\})\|_{1,1} + \mathcal{O}(\epsilon^2),$$

with c_ϵ being a constant dominated by ϵ, ie. $c_\epsilon = \Theta(\epsilon)$.

Appendix B. Additional experiments

B.1. On the complexity of PERTURB

The primary complexity of the proposed method lies in the generation of augmented graphs. This generation process can be repeated multiple times, denoted by K in the provided algorithm. For the results presented in this paper, we have fixed $K = 1$, meaning only a single batch of augmentations is computed for the model. Importantly, the graph generation process is independent across inputs, which makes it highly amenable to parallelization. Consequently, significant reductions in training time can be achieved.

To illustrate the potential time complexity of our method, we conducted training time comparisons between our proposed method and the original models. Specifically, Table 3 presents the mean training times (along with their corresponding standard deviations) for our proposed augmentation applied to both GCN and GIN, compared to their non-augmented counterparts. As anticipated, PERTURB requires significantly more time to train. It should be noted that these experiments were conducted on a single Tesla V100 GPU without parallelizing the generation process. With minimal adjustments, such as parallelizing this process, the training time could be considerably reduced.

B.2. On the validity of the generated graphs

In contrast to images, where small amounts of added noise within a certain threshold can still result in a valid image, in the graph domain, even minor modifications like adding or deleting an edge can significantly compromise the validity of the final graph. This is particularly crucial in datasets related to proteins or other bioinformatics domains, where the structural validity of the graph is essential.

Motivated by this concern, our paper proposes a generation scheme that leverages access to the real training dataset, enabling the generation of 'semantically' valid graphs. This approach stands in contrast to methods that rely on randomly dropping edges, which can lead to invalid or less meaningful graphs. To validate the quality of the graphs generated by our method, we evaluate them using various graph metrics using the Maximum Mean Discrepancy (MMD) measures, as described in You et al. (2018). Specifically, we employ the RBF kernel to assess both the degree and clustering coefficient distributions. The results, presented in Table 4, include the mean values and corresponding standard deviations for the DD, NCI1, and PROTEINS datasets.

Table 4
Results from the MMD (±standard deviation) of the generated samples using the PERTURB framework.

Metric	MMD metric	
	Deg.	Clus.
DD	0.14 ± 0.01	0.03 ± 0.02
NCI1	0.13 ± 0.01	0.05 ± 0.01
PROTEINS	0.16 ± 0.02	0.06 ± 0.03

Table 5
Statistics of the graph classification datasets used in our experiments.

Dataset	#Graphs	#Nodes	#Edges	#Classes
DD	1178	284.32	715.66	2
NCI1	4110	29.87	32.30	2
PROTEINS	1113	39.06	72.82	2
COLLAB	5000	74.49	2457.78	3
IMDB-BINARY	1000	19.77	96.53	2
IMDB-MULTI	1500	13.00	65.94	3
REDDIT-BINARY	2000	429.63	497.75	2

Appendix C. Details - Experimental setup

C.1. Datasets

We empirically evaluated our proposed method, PERTURB, on benchmark datasets derived from bioinformatics and chemoinformatics (Morris et al., 2020). The framework proposed by Errica et al. (2020) was used to evaluate the performance of the models on this task. We therefore performed a 10-fold cross-validation using the same folds as provided by the paper to obtain an estimate of the generalization performance of each method. Characteristics and information about the datasets utilized in the graph classification empirical analysis are presented in Table 5.

C.2. Implementation details

Our implementation is available as part of the submission (and will be publicly available afterwards). It is built using the open-source library *PyTorch Geometric* (PyG) under the MIT license (Fey and Lenssen, 2019).

We leveraged the publicly available implementation of the G-MixUp from their official Github. We additionally leveraged the implementation of the mixup paradigm from Wang et al. (2021) by adapting their implementation to take into account graph classification. For DropNode and DropEdge, we used directly their available implementation.

C.3. Synthetic datasets

As previously stated, the goal of our proposed method is to enhance the generalization and robustness of a given model. The aim of this

Table 6
Classification accuracy (±standard deviation) of GCN, GIN and GAT and the proposed framework (PERTURB) on the synthetic dataset.

Dataset	Accuracy on synthetic data
GCN	56.8 (±2.8)
GCN + PERTURB	67.2 (±1.2)
GIN	53.6 (±1.2)
GIN + PERTURB	59.1 (±2.1)
GAT	62.4 (±3.1)
GAT + PERTURB	68.3 (±2.6)

experiment is to empirically evaluate our method's ability to increase the generalization of a model to unseen and out-of-sample data. We created a dataset, each featuring 1000 synthetically generated Erdös-Rényi graphs with a 70%/30% training/test ratio. The train set is formed with graphs whose density is $d = 0.9$ and $d = 0.1$ while the test set is composed of graphs whose density is $d = 0.6$ and $d = 0.4$. The task is transformed into a binary classification of detecting the densest graph by taking a threshold $t = 0.5$ and each experiment is repeated 3 times. By construction, train and test data are drawn from the different distribution. Table 6 reports the resulting average prediction accuracy and the corresponding standard deviation of each experiment. These results illustrates the ability of the proposed methods to enhance the generalization ability of a number of models (GCN, GIN and GAT). We observe that our proposed framework yields notable improvement for the three underlying models.

References

Bojchevski, A., Günnemann, S., 2019. Adversarial attacks on node embeddings via graph poisoning. In: Proceedings of the 36th International Conference on Machine Learning. pp. 695–704.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., Sun, X., 2019. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. arXiv:1909.03211.

Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., Song, L., 2018. Adversarial attack on graph structured data. In: Proceedings of the 35th International Conference on Machine Learning. pp. 1115–1124.

Do, T.H., Nguyen, D.M., Bekoulis, G., Munteanu, A., Deligiannis, N., 2021. Graph convolutional neural networks with node transition probability-based message passing and DropNode regularization. Expert Syst. Appl. 174, 114711.

Errica, F., Podda, M., Bacciu, D., Micheli, A., 2020. A fair comparison of graph neural networks for graph classification. In: 8th International Conference on Learning Representations.

Feng, S.Y., Gangal, V., Wei, J., Chandar, S., Vosoughi, S., Mitamura, T., Hovy, E., 2021. A survey of data augmentation approaches for NLP. arXiv preprint arXiv:2105.03075.

Fey, M., Lenssen, J.E., 2019. Fast graph representation learning with pytorch geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., Lempitsky, V., 2016. Domain-adversarial training of neural networks. J. Mach. Learn. Res. 17 (59), 1–35, 2096–2030.

Goodfellow, I.J., Shlens, J., Szegedy, C., 2015. Explaining and harnessing adversarial examples. In: 3rd International Conference on Learning Representations.

Hamilton, W.L., Ying, R., Leskovec, J., 2017. Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 1025–1035.

Han, X., Jiang, Z., Liu, N., Hu, X., 2022. G-mixup: Graph data augmentation for graph classification. arXiv:2202.07179.

Kearnes, S., McCloskey, K., Berndl, M., Pande, V., Riley, P., 2016. Molecular graph convolutions: moving beyond fingerprints. J. Comput.-Aided Mol. Des. 30 (8), 595–608.

Kipf, T.N., Welling, M., 2017. Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations.

Kong, K., Li, G., Ding, M., Wu, Z., Zhu, C., Ghanem, B., Taylor, G., Goldstein, T., 2020. FLAG: Adversarial data augmentation for graph neural networks. arXiv preprint arXiv:2010.09891.

Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M., 2020. Tudataset: A collection of benchmark datasets for learning with graphs. arXiv preprint arXiv:2007.08663.

Rong, Y., Huang, W., Xu, T., Huang, J., 2020. Dropedge: Towards deep graph convolutional networks on node classification. arXiv:1907.10903.

Shorten, C., Khoshgoftaar, T.M., 2019. A survey on image data augmentation for deep learning. J. Big Data 6 (1), 1–48.

Simonovsky, M., Komodakis, N., 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3693–3702.

Sinha, A., Namkoong, H., Volpi, R., Duchi, J., 2017. Certifying some distributional robustness with principled adversarial training. arXiv preprint arXiv:1710.10571.

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., Madry, A., 2019. Robustness may be at odds with accuracy. In: 7th International Conference on Learning Representations.

Volpi, R., Namkoong, H., Sener, O., Duchi, J., Murino, V., Savarese, S., 2018. Generalizing to unseen domains via adversarial data augmentation. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. pp. 5339–5349.

Wang, Y., Wang, W., Liang, Y., Cai, Y., Hooi, B., 2021. Mixup for node and graph classification. In: Proceedings of the Web Conference 2021. WWW '21, Association for Computing Machinery, New York, NY, USA, pp. 3663–3674. http://dx.doi.org/10.1145/3442381.3449796.

Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., Tan, T., 2019. Session-based recommendation with graph neural networks. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence. pp. 346–353.

Xu, K., Hu, W., Leskovec, J., Jegelka, S., 2019. How powerful are graph neural networks?. In: 7th International Conference on Learning Representations.

Ying, R., You, J., Morris, C., Ren, X., Hamilton, W.L., Leskovec, J., 2018. Hierarchical graph representation learning with differentiable pooling In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. pp. 4805–4815.

You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., Shen, Y., 2020. Graph contrastive learning with augmentations In: Proceedings of the 33rd International Conference on Neural Information Processing Systems. pp. 5812–5823.

You, J., Ying, R., Ren, X., Hamilton, W.L., Leskovec, J., 2018. Graphrnn: generating realistic graphs with deep auto-regressive models http://dx.doi.org/10.48550/ARXIV.1802.08773, URL: https://arxiv.org/abs/1802.08773.

Zhan, H., Pei, X., 2021. Black-box gradient attack on graph neural networks: Deeper insights in graph-based attack and defense arXiv preprint arXiv:2104.15061.

Zhao, T., Liu, Y., Neves, L., Woodford, O., Jiang, M., Shah, N., 2021. Data augmentation for graph neural networks In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 35, pp. 11015–11023.

Zhao, L., Liu, T., Peng, X., Metaxas, D., 2020. Maximum-entropy adversarial data augmentation for improved generalization and robustness In: Proceedings of the 34th International Conference on Neural Information Processing Systems.

Zhao, Y., Zhang, H., Hu, X., 2022. Penalizing gradient norm for efficiently improving generalization in deep learning arXiv:2202.03599.

Zügner, D., Akbarnejad, A., Günnemann, S., 2018. Adversarial attacks on neural networks for graph data In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 2847–2856.

Zügner, D., Günnemann, S., 2019. Adversarial attacks on graph neural networks via meta learning In: 7th International Conference on Learning Representations.