GNN Predictions on k-hop Egonets Boosts Adversarial Robustness

Jian Vora
Department of Computer Science, Stanford University
jianv@cs.stanford.edu

Abstract

Like many other deep learning models, Graph Neural Networks (GNNs) have been shown to be susceptible to adversarial attacks, i.e., the addition of crafted imperceptible noise to input data changes the model predictions drastically. Most of these attacks perturb the graph edge structure and are known to connect very dissimilar parts of the network. We show that a very simple method, k-HOP-PURIFY, which makes node predictions using a k-hop Egonet centered at the node instead of the entire graph which boosts adversarial accuracies. This could be used both as i) a post-processing step after applying popular defenses or ii) as a standalone defense method that is comparable to many other competitors. The method is extremely lightweight and scalable (takes 4 lines of code to implement) unlike many other defense methods which are computationally expensive or rely on heuristics. We show performance gains by extensive experimentation across various types of attacks (poison/evasion, targetted/untargeted), perturbation rates, and defenses implemented in the DeepRobust Library [1].

1 Introduction

Graph Neural Networks (GNNs) have served as powerful representation models for graph data and have received attention across various disciplines [1, 2, 3]. The success of these models is attributed mainly to the message-passing scheme [4] where neural features are propagated along graph edges and at each layer, a node aggregates messages received from neighbors in the graph. Such a message-passing and aggregation scheme has led to great success in the representation power of GNNs with applications in social networks [5], particle interactions [6], disease pathways [7], etc. However on the other hand, like many deep learning models in domains such as images, text, etc., GNNs are susceptible to adversarial attacks which are designed to degrade GNN predictions by making imperceptible changes to the input graph [8, 9].

Adversarial attacks on graphs change the node features or the graph topology. While most attacks in the image or text domain inject adversarial noise in the data features [10, 11], most graph attacks inject carefully crafted edge perturbations and attack the graph structure. These attacks typically join nodes with dissimilar features/labels that were not connected in the clean graph [12]. The presence of such adversarial edges interferes with neural message passing which in turn makes the GNN make wrong predictions. The lack of GNN robustness is a critical issue in many application areas, which hinders the use of GNNs especially in many safety-critical applications [13, 14]. We deal with the task of node classification but the same ideas can be extended to other downstream tasks.

Due to the pressing need for developing GNNs to be robust to adversarial attacks, a lot of work has been pushed into developing defense methods for these models. Most existing defense methods for graphs either involve adversarial training (i.e., training the graph for the worst-possible perturbation)

[1] https://github.com/DSE-MSU/DeepRobust

NeurIPS 2023 New Frontiers in Graph Learning Workshop (NeurIPS GLFrontiers 2023).

or some graph processing that involves identifying adversarial edges. Many of these defense methods are computationally expensive [15] and rely on heuristics to prune off edges [16] which also destroys the original structure of the graph which hampers performance.

GNN predictions are done by passing the entire graph in the model which provides logits for all nodes at once which are used to provide the class to the node. For images, some very simple inference time processing techniques have been shown to mitigate adversarial robustness, such as random resizing and padding [17], quantization of input image pixels [18], etc. All of these were based on a simple principle: the image looks almost the same to the human eye so the model performance should not degrade a lot on performing these transformations. On the other hand, performing such transformations helps reduce the effect of any attack. We ask a similar question for graphs:

Is there a simple test-time transformation that helps in mitigating adversarial noise for graphs?

We answer the question affirmatively: using k-hop egonets (subgraphs centered at a node) around the node to be predicted helps a lot in mitigating the effect of an adversary. Thus, instead of using the entire graph to predict node labels, just construct a local neighborhood for each test node, and using these (much) smaller subgraphs helps mitigate the effects of a lot of common adversaries. The intuition is similar – showing the model only the most relevant information avoids distraction which generally leads to wildly incorrect features.

2 Background and Preliminaries

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ with n nodes and m edges, we deal with the task of transductive node classification, i.e., given some labeled nodes in the graph, the task is to predict the class of all the unlabelled nodes with node labels $\mathbf{y} \in \{1, 2, \ldots, C\}^n$, where C is the total number of classes. $\mathbf{X} \in \mathbb{R}^{n \times d} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$ are the node features where d is the dimensionality of the node features. The graph adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$ is the denotes the node connectivity, i.e., $\mathbf{A}_{ij} = 1$ iff nodes i and j are connected by an edge. We have a GNN model $f_\theta(\mathbf{A}, \mathbf{X})$, that takes in the graph features and connectivity and outputs logits $\hat{Y} \in \mathbb{R}^{n \times C}$ which are then used to make predictions. In the adversarial setting, an attacker perturbs the graph $\mathcal{G}$ to $\mathcal{G}' = (\mathcal{V}, \mathcal{E}', \mathbf{X}')$ which degrades the performance of the classifiers f_θ. Note that we are only focussing on attacks that do not perturb the vertex set of the graph but can change the connectivity and node features. In the subsequent subsections, we shall go over some preliminaries that shall serve as an overview of common graph attack and defense techniques.

2.1 Background on Graph Neural Networks (GNNs)

Graph neural networks (GNNs) learn a low-dimensional representation vector for each node in the graph which can be used for downstream predictions [19, 20]. These representations are learned using both the node features $\mathbf{X}$ provided in the graph and features of the neighborhood of the node using the graph connectivity $\mathbf{A}$. A GNN is made by stacking L layers where node features are modified at each layer. Let $\mathbf{h}_u^k \in \mathbb{R}^d$, be the a feature of node u at layer k. Using this notation, we have $\mathbf{h}_u^0 = \mathbf{x}_u$, the original node features that one starts off with. The GNN f_θ comprises a triplet (MSG, AGG, UPD) which governs how the node embeddings evolve during a GNN forward pass [21]. A node u receives messages $\mathbf{m}_{uv}^k$ from each of it's neighbors $v \in \mathcal{N}(u)$, where the message $\mathbf{m}_{uv}^k$ is computed using $\text{MSG}(\mathbf{h}_u^k, \mathbf{h}_v^k, \mathbf{A}_{uv})$. The node u after receiving messages from all its neighbours aggregates those into $\hat{\mathbf{m}}_u^k = \text{AGG}(\mathbf{m}_{uv}^k; v \in \mathcal{N}(u))$. The node then finally updates its features using the UPD function, i.e., $\mathbf{h}_u^{k+1} = \text{UPD}(\hat{\mathbf{m}}_u^k, \mathbf{h}_u^k)$. The same process is continued for L layers to assign the nodes their final features which is equal to $\mathbf{h}_u^L$. This final feature can then be used for any downstream task such as node classification. Note that GNNs are both invariant to node permutation and the size of the input graph, i.e., the same GNN can be tested on a graph of variable size as the learned weights just update pairwise node connections and are invariant to the global connectivity [22].

2.2 Background on Adversarial Attacks

Deep learning models used for domains such as images and texts mainly modify the data features (image pixels or word embeddings) [8, 11]. On the other hand, attacks on graphs have many design choices which we shall be going over in this subsection. For this work, we mainly concern ourselves

with only white-box attacks given that they are the strongest and it makes sense to study defense methods for these worst-case attacks. Most of these attacks aim to add some (bounded) perturbation to the input. The attacks are designed to be imperceptible (say, from anomaly detection methods) yet are capable of degrading model performance which requires crafting these attacks carefully. Attacks on graphs can be classified into one of the following categories:

1. **Feature vs Structure Perturbation.** An attack can either add adversarial noise in the node features or change the structure of the graph (node/edge addition or deletion) in order to attack the message-passing algorithm which in turn would lead to spurious node features. Most attacks on graphs are known to connect nodes with dissimilar features/labels which destroys the homophily and has been shown to degrade the GNN performance [23].
2. **Global vs Local Attacks.** *Local* (or targetted) on a particular node or *targetted* (or global) to perturb the performance on the entire test set. Performance for local attacks is measured by success rate, i.e., how many times the attacker is able to misclassify the target node on applying the attack vs global attacks are measured by overall test-set accuracies.
3. **Evasion vs Poisoning Attacks.** Evasion (test-time) attacks involve having an already trained GNN f_θ model on a clean graph $\mathcal{G}$, and then perturbing it into a new graph $\mathcal{G}'$ where the model would fail. Poisoing attacks (train-time), on the other hand do not have access to the GNN model and first change the graph $\mathcal{G}$ to $\mathcal{G}'$. The GNN model is then finally trained on the adversarial graph $\mathcal{G}'$. Poisoning attacks are typically harder to design than evasion attacks due to lack of the model information and the need to *see through* the training process to analyze the effects of the made changes, both through training and inference [24].

Some common attacks which we use in our experiments are briefly described below:

Projected Randomized Block Coordinate Descent (PRBCD) [15]. This is an evasion attack that scales to millions of nodes and performs PGD on only a subset of promising variables at a time which improves its scalability. This is the only known attack to scale to larger graphs such as ogbn-arxiv.

Delete Internally, Connect Externally (DICE) [25]. This is a simple baseline global attack where, for each perturbation, we randomly choose whether to insert or remove an edge. Edges are only removed between nodes from the same classes and inserted between nodes from different classes.

Nettack [9]. This is a targetted attack that selects a set of nodes within the graph that are most influential and then alternately updates the node features and edge structure.

Mettack [26]. This is typically used as a strong global poisoning attack which uses meta-gradients using the to-be-attacked graph as a hyperparameter to optimise.

2.3 Background on Adversarial Defenses

In this subsection, we shall go over some common defenses against structural graph attacks. On a high level, most of these methods try to identify discriminating features between a clean and attacked graph. The next step is essentially to try to circumvent the identified anomaly in some form which subsequently improves the model performance. As described in [24], these methods can be classified into one of the following three methods.

Improving the Graph. Attacks typically affect the higher-order singular values of the graph adjacency matrix, and hence GCNSVD [27] uses a low-rank approximation of the adjacency matrix before a pass through the GNN. GNN-Jaccard [16] prunes edges between nodes whose similarity is below a certain pre-determined threshold. Pro-GNN [28] is a supervised defense method that alternatingly optimizes the parameters of the GNN and the adjacency matrix (which leads to a cleaner graph).

Improving the Training. For having more robust models, one idea is adversarial training which involves training the model on adversarial samples, i.e., minimizing the loss for a worst-case perturbation. GRAND [29] performs random feature augmentations of the node and its neighbors.

Improving the Architecture. This typically involves changing some part of message passing, such as using a median aggregation function [30, 31] that is more robust or filtering potentially adversarial edges during message passing. Some other methods like RGCN [32] also try to induce robustness by using ideas of distributions over node features as opposed to point estimates.

3 Method

As described in the previous section, most literature on adversarial graph learning can be pinned down as the following: *attacks* add edges between dissimilar nodes and removes edges from similar nodes; *defenses* that try to identify such perturbations and try it "purify" the graph at different stages which would try to ensure that model performance would not degrade. Inference on graph neural networks proceeds by passing the entire graph through the model which provides logits (and hence labels) for all the nodes in a single forward pass. The issue with passing the entire graph is that adversarial edges joining very unrelated parts of the network affect message propagation leading to misclassification. We propose that during inference time, instead of passing the entire graph, we only pass a k-hop subgraph centered at the test node in the GNN model. In code, the change is extremely minor as shown below:

```
data: PyG data object
model: GNN model
node_idx: Node to be classified

# Earlier Inference:

output = model(data.x, data.edge_index)
assigned_label = output[node_idx].argmax(-1)

# k-Hop-Purify Inference:

from torch_geometric.utils.subgraph import k_hop_subgraph
subset, edge_index_subgraph, _, _ = k_hop_subgraph(node_idx, k, data.edge_index)
output = model(data.x[subset], edge_index_subgraph)
assigned_label = output[node_idx].argmax(-1)
```

Note that we are generating a new subgraph for the test node which increases computational cost, however, this step can be easily parallelized and all the obtained subgraphs can be passed as a batch to the model. The idea is to *show* the model only the most important part of the graph around the node to be predicted and surprisingly, just doing this circumvents messages received along spurious edges. This inference mechanism can be used in general as a post-processing method after applying any defense method, i.e., `model` could be either a vanilla model or an already defended model. Similarly, this method can be used for both evasion and poison attacks. In the latter case, the graph passed for inference would be the edge index of the poisoned graph. We show that in our experiments, this way of performing inference can also be used as a post-processing step to any popular defense to significantly improve their adversarial accuracy.

4 Experiments

We perform extensive experimentation across a variety of attacks and defenses using the DeepRobust Library. For all the experiments, the value of k was chosen as a hyperparameter which maximized the validation accuracy. The values of k which worked the best for all datasets were in the range of $2-4$. Results indicate test-accuracies averaged over 5 runs along with one standard deviation. All other hyperparameters related to both the model training and attacks are kept the same as in the library unless mentioned explicitly. Δ indicates the perturbation rate, denoting that a total of $\Delta \times m$ edges could be perturbed by the attacker (written in parentheses next to the attack name). Table 1 describes the results of performing different types and strengths of global evasion attacks on different datasets using a vanilla GCN model only. The number in parentheses is the model accuracy on that dataset without any attack. The attacked model column indicates the model accuracy after performing the mentioned attack and the final column just changes the inference method as described above without any additional training/processing. We see that for all these scenarios, using the k-Hop-Purify inference method does significantly better in terms of test-set accuracies.

Next, in Table 2, we observe that our method can be used as a post-processing step over commonly used defenses to give a $4-5\%$ boost in adversarial accuracies. We also compare with other commonly used GNN architectures such as AirGNN [33], SGC [34], and APPNP [35]. The above two tables

Dataset	Attack Name (Δ)	Model Name	Attacked Model	Attacked Model + k-Hop-Purify
Cora	PRBCD (0.1)	GCN (77.16)	65.13 ± 1.34	**70.72** ± 1.61
Cora	PRBCD (0.2)	GCN (77.16)	56.84 ± 1.53	**68.49** ± 1.28
Cora	PRBCD (0.3)	GCN (77.16)	46.93 ± 1.02	**63.73** ± 1.31
Cora	DICE (0.1)	GCN (77.16)	67.22 ± 1.03	**70.53** ± 0.94
Cora	DICE (0.2)	GCN (77.16)	57.55 ± 0.86	**63.17** ± 0.72
Cora	DICE (0.3)	GCN (77.16)	50.05 ± 0.85	**58.35** ± 0.94
Cora	METTACK (0.1)	GCN (77.16)	61.34 ± 0.56	**68.32** ± 1.14
Cora	METTACK (0.2)	GCN (77.16)	52.13 ± 1.34	**59.73** ± 0.86
Cora	METTACK (0.3)	GCN (77.16)	41.03 ± 0.73	**55.01** ± 0.92
Pubmed	PRBCD (0.1)	GCN (81.61)	71.33 ± 1.01	**75.91** ± 1.29
Pubmed	PRBCD (0.2)	GCN (81.61)	64.41 ± 1.62	**69.01** ± 1.11
Pubmed	PRBCD (0.3)	GCN (81.61)	57.94 ± 1.29	**64.27** ± 1.83
Pubmed	DICE (0.1)	GCN (81.61)	**73.39** ± 0.84	**74.86** ± 1.07
Pubmed	DICE (0.2)	GCN (81.61)	66.63 ± 1.12	**69.06** ± 1.32
Pubmed	DICE (0.3)	GCN (81.61)	57.91 ± 1.04	**64.39** ± 0.87
Pubmed	METTACK (0.1)	GCN (81.61)	**67.34** ± 1.03	**69.04** ± 0.83
Pubmed	METTACK (0.2)	GCN (81.61)	62.91 ± 1.27	**67.31** ± 1.02
Pubmed	METTACK (0.3)	GCN (81.61)	55.13 ± 1.14	**62.87** ± 0.91
ogbn-arxiv	PRBCD (0.1)	GCN (70.65)	49.12 ± 1.14	**62.06** ± 1.71
ogbn-arxiv	PRBCD (0.2)	GCN (70.65)	31.79 ± 1.01	**53.92** ± 1.42
ogbn-arxiv	PRBCD (0.3)	GCN (70.65)	15.43 ± 1.86	**43.73** ± 1.37

Table 1: Global Evasion: Using k-Hop-Purify as a standalone defense method

Dataset	Model Name	Attacked Model	Attacked Model + k-Hop-Purify
Cora	AirGNN (83.2)	70.88 ± 0.75	**75.88** ± 0.45
Cora	MedianGCN (85.7)	75.88 ± 0.63	**79.88** ± 0.71
Cora	SGC (79.13)	73.88 ± 1.15	**77.19** ± 0.67
Cora	APPNP (81.2)	**74.14** ± 1.13	**75.39** ± 1.05
Cora	GCNSVD (83.1)	70.14 ± 0.65	**77.21** ± 0.82
Cora	GCNJaccard (80.3)	76.34 ± 1.34	**79.12** ± 0.75
Cora	RCGN (78.7)	73.24 ± 1.03	**76.19** ± 0.73
ogbn-arxiv	AirGNN (72.3)	52.98 ± 0.81	**61.12** ± 0.53
ogbn-arxiv	MedianGCN (75.8)	61.39 ± 0.52	**65.13** ± 1.10
ogbn-arxiv	SGC (74.6)	55.32 ± 0.81	**62.34** ± 1.32
ogbn-arxiv	APPNP (73.4)	**52.35** ± 0.69	**55.54** ± 0.82
ogbn-arxiv	GCNSVD (70.7)	51.35 ± 1.54	**59.35** ± 1.15
ogbn-arxiv	GCNJaccard (72.9)	56.89 ± 0.75	**62.97** ± 0.77
ogbn-arxiv	RCGN (74.6)	59.38 ± 1.42	**63.94** ± 0.65

Table 2: Global Evasion: Using k-Hop-Purify as a post-processing inference after applying a defense model. The attack was a PRBCD attack with $\Delta = 0.2$.

suggest the utility of k-Hop-Purify for global evasion attacks. We also investigate whether similar trends are observed in global poisoning attacks, where the graph is perturbed first and then a model is trained on the adversarial graph. Poisoning is a harder attack and most successful attacks use some variation of meta-learning. For our experiments, we use already perturbed graphs using Mettack with varying perturbation rates as provided here [2]. Table 3 indicates that the proposed inference method performs well for poisoning attacks as well, albeit with lower improvements indicating the hardness of defending these attacks over evasion attacks.

We then look at targeted attacks where the goal of the adversary is to misclassify a provided target node instead of overall test accuracies as in evasion attacks. We instead measure the attack success rate, i.e., how many fraction of times the attack succeeds in misclassifying the target node. Nettack is a popular local adversarial graph attack which is what we used in our experiments with perturbation rate $\Delta = 0.2$. Table 4 shows the results where the model to be attacked was a GCN model. The number indicates the rate of correct classification of the target node (higher is better) and it indicates that just passing a k-hop neighborhood to the GNN also benefits local attacks.

[2] https://github.com/ChandlerBang/Pro-GNN

Dataset	Model Name	Attacked Model	Attacked Model + k-Hop-Purify
Cora	GCN	44.67 ± 1.34	**55.38** ± 0.79
Cora	MedianGCN	56.23 ± 1.07	**63.93** ± 0.92
Cora	GCNSVD	54.32 ± 1.94	**60.38** ± 1.73
Cora	GCNJaccard	55.43 ± 0.64	**58.12** ± 1.41
Cora	RCGN	53.98 ± 0.71	**57.03** ± 1.52
Pubmed	GCN	75.46 ± 1.14	**77.13** ± 1.01
Pubmed	MedianGCN	**77.23** ± 0.92	**78.14** ± 1.43
Pubmed	GCNSVD	76.32 ± 0.71	**78.04** ± 0.89
Pubmed	GCNJaccard	**78.13** ± 0.62	**79.15** ± 1.16
Pubmed	RCGN	77.74 ± 0.95	**78.84** ± 0.54

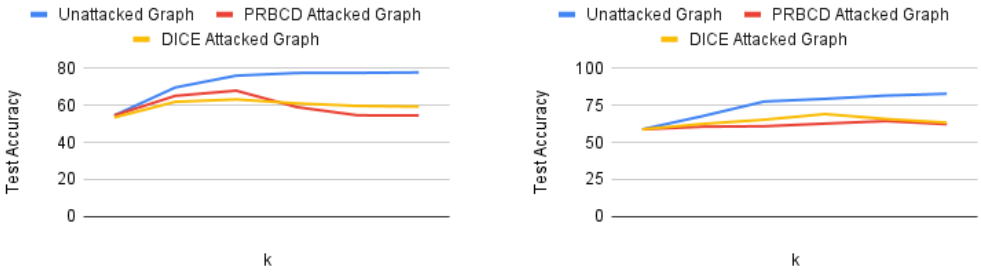
Table 3: Global Poison: Using k-Hop-Purify as a post-processing inference after applying a defense model. The attack was a Mettack attack with $\Delta = 0.2$.

Dataset	Attacked Model	Attacked Model + k-Hop-Purify
Cora	2%	**18**%
Pubmed	3%	**26**%
Citeseer	1%	**14**%

Table 4: Local Nettack with $\Delta = 0.2$.

Performance Change with Varying k

One important question one might ask is how the performance varies with changing k, the number of hops to sample the subgraph from $0 - 5$. $k = 0$ indicates just using the node features of prediction, and increasing k takes more of the neighborhood into account. As shown in Fig 1, we see that performance typically peaks by taking an intermediate value of k and worsens on both sides of the spectrum. This indicates that some information of neighborhood helps in prediction but as we start getting too far, the effect of adversarial edges kicks in. Another question is the following: does the approach to perform $k-$hop inference help only adversarial graphs or even help in usual GNN inference to improve performance? As we see in Fig 1, for unattacked graphs, it makes sense to pass as much context as possible which can be seen with monotonically increasing performance with increasing k. This points to the fact that performing k-hop subgraph inference is something special for adversarial graphs and isn't a general recipe for GNN inference.



(a) Cora Dataset

(b) Pubmed Dataset

Figure 1: Effect of changing k on the method for both attacked and unattacked graphs

5 Conclusion

In this work, we demonstrated that a simple method of using k-hop subgraphs instead of an entire graph boosts adversarial robustness for a variety of different attacks. This raises some very interesting questions in studying the nature of attacks on graphs in more detail, and whether simply pruning edges that connect *far-away* nodes in the graph help in robustness. The existence of such an inference method calls for more sophisticated attacks for which even local neighborhoods are not reliable for making a prediction. Some avenues of future research involve analyzing k-Hop-Purify theoretically and looking at the overlap between the edges removed and adversarial edges introduced by the attacker.

References

[1] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[2] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):249–270, 2020.

[3] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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

[5] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426, 2019.

[6] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020.

[7] Neal Ravindra, Arijit Sehanobish, Jenna L Pappalardo, David A Hafler, and David van Dijk. Disease state prediction from single-cell data using graph attention networks. In *Proceedings of the ACM conference on health, inference, and learning*, pages 121–130, 2020.

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[9] Daniel Zügner, Amir Akbarnejad, and Stephan 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.

[10] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[11] Siddhant Garg and Goutham Ramakrishnan. Bae: Bert-based adversarial examples for text classification. *arXiv preprint arXiv:2004.01970*, 2020.

[12] Enyan Dai, Tianxiang Zhao, Huaisheng Zhu, Junjie Xu, Zhimeng Guo, Hui Liu, Jiliang Tang, and Suhang Wang. A comprehensive survey on trustworthy graph neural networks: Privacy, robustness, fairness, and explainability. *arXiv preprint arXiv:2204.08570*, 2022.

[13] Walt Woods, Jack Chen, and Christof Teuscher. Adversarial explanations for understanding image classification decisions and improved neural network robustness. *Nature Machine Intelligence*, 1(11):508–516, 2019.

[14] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

[15] Simon Geisler, Tobias Schmidt, Hakan Şirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. Robustness of graph neural networks at scale. *Advances in Neural Information Processing Systems*, 34:7637–7649, 2021.

[16] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. *arXiv preprint arXiv:1903.01610*, 2019.

[17] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[18] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. *arXiv preprint arXiv:1904.08444*, 2019.

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

[20] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[21] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33:17009–17021, 2020.

[22] Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[23] Xiang Zhang and Marinka Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. *Advances in neural information processing systems*, 33:9263–9275, 2020.

[24] Felix Mujkanovic, Simon Geisler, Stephan Günnemann, and Aleksandar Bojchevski. Are defenses for graph neural networks robust? *Advances in Neural Information Processing Systems*, 35:8954–8968, 2022.

[25] Marcin Waniek, Tomasz P Michalak, Michael J Wooldridge, and Talal Rahwan. Hiding individuals and communities in a social network. *Nature Human Behaviour*, 2(2):139–147, 2018.

[26] Daniel Zügner, Oliver Borchert, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on graph neural networks: Perturbations and their patterns. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(5):1–31, 2020.

[27] Shaowen Peng, Kazunari Sugiyama, and Tsunenori Mine. Svd-gcn: A simplified graph convolution paradigm for recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 1625–1634, 2022.

[28] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang 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.

[29] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. *Advances in neural information processing systems*, 33:22092–22103, 2020.

[30] Liang Chen, Jintang Li, Qibiao Peng, Yang Liu, Zibin Zheng, and Carl Yang. Understanding structural vulnerability in graph convolutional networks. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2249–2255. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[31] Simon Geisler, Daniel Zügner, and Stephan Günnemann. Reliable graph neural networks via robust aggregation. *Advances in Neural Information Processing Systems*, 33:13272–13284, 2020.

[32] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019.

[33] Zhan Gao and Deniz Gunduz. Airgnn: Graph neural network over the air. *arXiv preprint arXiv:2302.08447*, 2023.

[34] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[35] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.