2024 IEEE 3rd International Conference on Computing and Machine Intelligence (ICMI) | 979-8-3503-7297-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICMI60790.2024.10586025

Adversarial Attack and Training for Graph Convolutional Networks using Focal Loss-Projected Momentum

Mohammed Aburidi
Department of Applied Mathematics
University of California Merced
Merced, USA
maburidi@ucmerced.edu

Roummel F. Marcia
Department of Applied Mathematics
University of California Merced
Merced, USA
rmarcia@ucmerced.edu

***Abstract*—Graph Neural Networks (GNNs) have exhibited remarkable success in various applications such as social and telecommunication networks, yet their vulnerability to adversarial attacks poses significant risks in security-sensitive domains. Imperceptible perturbations in graphs can lead to severe performance degradation, necessitating robust GNN models for safety and privacy in critical applications. We address this challenge by proposing optimization-based attacks on GNNs, specifically focusing on modifying graph structures. Our approach leverages convex relaxation and projected momentum optimization. Introducing the focal loss as an attack criterion, we generate perturbations by minimizing a constrained optimization problem. Evaluating on node classification tasks, our attacks outperform state-of-the-art methods under the same perturbation budget, highlighting the effectiveness of our approach. This work contributes to enhancing the robustness of GNNs against adversarial manipulations in real-world scenarios.**

***Index Terms*—Graph Neural Networks, Adversarial Attack, Adversarial Training, Defense, Focal Loss.**

I. Introduction

Graph-structured data is a fundamental component in various AI applications, serving as a versatile representation for modeling datasets across diverse domains such as molecules, social networks, and interlinked documents with citations [1]–[5]. The application of Graph Neural Networks (GNNs) to graph-structured data has demonstrated remarkable success in tasks ranging from node classification to link prediction and recommender systems [1], [2], [6]. Despite these achievements, recent studies reveal a critical challenge: the vulnerability of GNNs to adversarial attacks [7]–[11]. Even imperceptible perturbations in graphs can lead to substantial performance degradation, posing severe risks in security-sensitive domains such as blockchain and communication networks. Notably, GNNs, following a message-passing scheme [12], are susceptible to adversarial manipulations primarily focused on modifying graph structures, including edge additions, deletions, and rewirings [13], [14]. In this context, we address the pressing need for robust GNN models capable of withstanding adversarial attacks on graph structure, given their potential implications for safety and privacy in critical applications.

Attacking graph structures effectively poses a challenging problem. In contrast to images, where data is continuous, graphs are discrete. Additionally, the combinatorial nature of graph structures introduces increased complexity compared to text and images. Motivated by recent progress in generating adversarial attacks on audios or images using first-order optimization methods [15]–[18], we delve into the task of crafting attacks through convex relaxation and projected momentum-based optimization. We propose the use of the focal loss as an attack loss, and we generate perturbations by minimizing a focal loss-based constrained optimization problem, leveraging momentum and projections onto the feasible set.

In the evaluation of node classification tasks with GNNs, our optimization-based attacks demonstrate outperformance compared to existing state-of-the-art attacks under the same perturbation budget. This showcases the efficiency of our attack generation approach utilizing convex relaxation and first-order projected momentum optimization, resulting in substantially enhanced robustness against greedy attacks.

II. Problem Statement

A. Preliminaries on GNNs

Recent studies have demonstrated the effectiveness of Graph Neural Networks (GNN) in transductive learning, e.g., node classification within graph data [1]–[3]. This means that when provided with a network topology, node features, and a known subset of node labels, GNNs can efficiently predict the classes of unlabeled nodes. Prior to defining GNN, let's establish some graph notations. Consider $\mathscr{G} = (\mathscr{V}, \mathscr{E}, X)$ as an undirected and unweighted graph. Here, V represents the set of N vertices (or nodes), $\mathscr{E} \in (\mathscr{V} \times \mathscr{V})$ represents the set of edges. The edges describe the relations between nodes and can also be represented by a binary adjacency matrix $A \in \mathbb{R}^{N\times N}$, where A_{ij} denotes the relation between nodes v_i and v_j, and $A_{ij} = 0$ if $(i, j) \notin \mathscr{E}$. Here $X = [x_1, x_2, ..., x_N] \in \mathbb{R}^{N\times d_0}$ denotes the node feature matrix where $x_i \in \mathbb{R}^{d_0}$ is the feature vector of the node

This research is partially supported by NSF Grant IIS 1741490 and DMS 1840265.

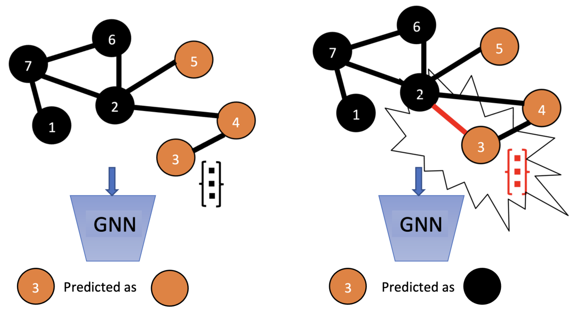


Fig. 1. A demonstration of an adversarial attack on graph data. The objective of the Graph Neural Network is to predict the color of the nodes. In this scenario, node 3 is the specific target. The attacker intends to alter the GNN's prediction for node 3 by manipulating either the edges or features or both.

v_i. Following the common node classification setting, only a part of nodes $\mathscr{V}_{train} = [v_1, v_2, \ldots, v_{N_{train}}]$ are associated with corresponding labels $Y_{train} = [y_1, y_2, \ldots, y_{N_{train}}]$.

Given a graph $\mathscr{G} = (\mathscr{V}, \mathscr{E}, X)$ and the partial labels Y_{train}, the goal of GNN is to learn a function $f_W : \mathscr{V}_{train} \rightarrow \mathscr{Y}_{train}$ that maps the nodes to the set of labels so that f_W can predict labels of unlabeled nodes.

In GNNs, the representation $h_i^{(k)} \in \mathbb{R}^{1\times d_k}$ of node v_i at the k^{th} layer is recursively calculated by aggregating the messages (information) propagated from its neighbors. The update of the representation of node v_i at the k^{th} is given by following generic propagation rule

$$\mathrm{h}_i^{(k+1)} = g^{k+1}\left(\mathrm{Aggr}_{j\in\mathscr{N}(i)}\left(\phi^{k+1}(\mathrm{h}_j^{(k)})\right)\right) \tag{1}$$

where $\mathrm{h}_i^{(k+1)} \in \mathbb{R}^{d_{k+1}}$ denotes the feature vector of node v_i at layer $k+1$, $\mathrm{h}_i^{(0)} = x_i$ is the input feature vector of node v_i. $g(\cdot)$ is a mapping (activation) function (e.g., ReLU). The function $\mathrm{Aggr}(\cdot)$ is a permutation-invariant aggregator such as the mean pooling, and $\phi(\cdot)^{(k+1)}$ is differential function such as one defined by an MLP, and $\mathscr{N}(i)$ denotes the node v_i's neighbors together with itself $(j = i)$ [1], [2].

A fundamental form of GNNs is the Graph Convolutional Network (GCN) proposed by Kipf and Welling in 2016 [1]. GCNs leverage the spectral graph theory and convolutional operations to learn node representations. The key idea behind GCNs is to aggregate information from neighboring nodes, allowing each node to refine its representation based on the local graph structure. In GCNs, the propagation rule in (1) becomes

$$\mathrm{h}_i^{(k+1)} = \sigma\Big(\sum_{j\in\mathscr{N}(i)} \frac{1}{\sqrt{\mathrm{d}_i\mathrm{d}_j}}\mathrm{h}_j^{(k)}\mathrm{W}^{(k)}\Big) \tag{2}$$

where $\sigma(\cdot) = max(0,\cdot)$ is the ReLU function. $\mathrm{W}^{(k)} \in \mathbb{R}^{d_k\times d_{k+1}}$ is the weight matrix for layer k. The scalars d_i and d_j are the degrees of nodes v_i and v_j, respectively. Using the layer-wise notations, the propagation rule will take the form

$$\mathrm{H}^{(k+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\mathrm{H}^{(k)}\mathrm{W}^{(k)}\right) \tag{3}$$

where $\mathrm{H}^{(k)} \in \mathbb{R}^{N\times d_k} = [(h_1^{(k)})^T; (h_2^{(k)})^T; \cdots; (h_N^{(k)})^T]^T$; $\tilde{A} = A + I_N$ is the normalized adjacency matrix with added self-connections, where I_N is the identity matrix; and $\tilde{D}$ is a diagonal matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. For further details, see [1].

B. Attack Generation Through Edge Perturbation

A topological perturbed graph is represented by a perturbed adjacency matrix, denoted as $A' = A + \delta$, where the matrix δ represents the changes to the edges added to the original matrix A, and it is given by the element-wise product of two $N\times N$ matrices, C and M:

$$\delta = C \odot M, \tag{4}$$

where $\odot$ denotes the element-wise product and $M \in \{0,1\}^{N\times N}$ is a Boolean symmetric matrix that encodes whether an edge in $\mathscr{G}$ is modified. The matrix C is given by $C = \mathbb{1}_{N\times N} - I_N - 2A$, where $\mathbb{1}_{N\times N}$ is the $N\times N$ matrix of ones. The positive entries of C denote the edges that can be added to the graph A while its negative entries denote edges that can be removed. Here we assume that nodes are not self-connected. Our goal is to find the minimum edge perturbations encoded by the matrix M to mislead GNNs. We call our attack generation method Focal Loss-Projected Momentum (FL-PM) Attack, and we begin by defining our proposed attack loss function.

C. Focal Loss Function

Let $P(M,W;A,X)$ denote the prediction probability of a GNN specified by A. Here, we propose to use *focal loss* as a per-node attack loss. Introduced in [19] as a training loss, it surpasses the traditional cross-entropy loss in scenarios where class imbalance is prevalent by introducing a dynamic modulating factor that downweights the impact of well-classified examples. This factor is controlled by a focusing parameter, mitigating the dominance of the majority class, allowing the model to prioritize learning from challenging instances. Previous studies have employed various types of loss functions, such as cross-entropy [20], [21], as attack losses. We, here, propose to use focal loss-type attacks for attacking graphs for the first time, which is given by

$$FL_i(M,\mathrm{W};A,X,y_i) = -\frac{1}{K}\sum_{j=1}^{K}\bar{y}_{ij}\cdot(1-P_{ij})^{\gamma}\cdot\log(P_{ij}) \tag{5}$$

where γ is the focusing parameter which controls the rate at which easy examples are down-weighted. K are the number of classes and $\bar{y}_{ij}$ is a binary indicator of whether class j is the correct classification for instance v_i (1 if true, 0 otherwise).

To formulate our attack strategy, we seek M in (4) that minimizes the per-node FL-type attack loss given a finite budget of edge perturbations $\Delta \in \mathbb{N}$. We consider a model in which we target a pre-defined GNN with a known weight matrix W. Given a fixed W, the problem of generating attacks can be framed as follows

$$\begin{aligned} \underset{\mathbf{m}\in\mathbb{R}^n}{\text{minimize}} \quad & \sum_{i\in\mathscr{V}} FL_i(M,\mathrm{W};A,X,y_i) \\ \text{subject to} \quad & \mathbf{1}_n^T\mathbf{m} \le \Delta \\ & \mathbf{m} \in \{0,1\}^n, \end{aligned} \tag{6}$$

where we replace the symmetric matrix variable M with its vector form $\mathbf{m}$ that consists of $n = N(N-1)/2$ unique variables in M and $\mathbf{1}_n$ is the vector of ones with length n. The second constraint enforces the resulting M to be Boolean.

D. Projected Momentum-Based Attack

The optimization problem (6) presents a combinatorial optimization challenge due to the inclusion of Boolean variables. To simplify the optimization process, we choose to relax the constraint on $\mathbf{m}$ to its convex hull $\mathbf{m} \in [0,1]^n$, which makes our optimization problem continuous:

$$\begin{aligned} &\underset{\mathbf{m}\in\mathbb{R}^n}{\text{minimize}} \quad FL(\mathbf{m}) := \sum_{i\in\mathscr{V}} FL_i(M, \mathrm{W}; A, X, y_i) \\ &\text{subject to} \quad \mathbf{m} \in \mathscr{M} \end{aligned} \tag{7}$$

where $\mathscr{M} = \{\mathbf{m} | \mathbf{1}^T\mathbf{m} \le \Delta, \mathbf{m} \in [0,1]^n\}$. We solve the optimization problem (7) by projected momentum:

$$\begin{aligned} \mathbf{v}_{t+1} &= \beta \mathbf{v}_t + (1-\beta)\nabla FL(\mathbf{m}^t) \\ \mathbf{m}^{t+1} &= \mathrm{Proj}_{\mathscr{M}}\left[\mathbf{m}^t - \eta \mathbf{v}_{t+1}\right] \end{aligned} \tag{8}$$

where t denotes the iteration index of projected momentum, η is the learning rate, β is the momentum decay factor, $\mathbf{v}_t$ is the momentum, $\nabla FL(\mathbf{m}^t)$ is the gradient of the attack loss at $\mathbf{m}^t$, and $\mathrm{Proj}_{\mathscr{M}}[\cdot]$ is a projection mapping onto the set $\mathscr{M}$. Momentum is an optimization technique that accelerates convergence and enhances the stability of gradient-based methods [22]. By incorporating a momentum term that accumulates information from past gradients, this helps maintain a consistent direction of optimization, enabling faster movement along relevant dimensions and dampening oscillations. This smoothing effect makes our solver effective in escaping local minima and navigating complex optimization landscapes.

Using the approach in [20] and by letting $\mathbf{z} = \mathbf{m}^t - \eta \mathbf{v}_{t+1}$, the projection $\mathrm{Proj}_{\mathscr{M}}[\mathbf{z}]$ is computed by solving a minimization problem given by

$$\begin{aligned} &\underset{\mathbf{m}\in\mathbb{R}^n}{\text{minimize}} \quad \tfrac{1}{2}\|\mathbf{m}-\mathbf{z}\|_2^2 + \mathscr{R}_{[0,1]^n}(\mathbf{m}) \\ &\text{subject to} \quad \mathbf{1}_n^T\mathbf{m} \le \Delta, \end{aligned} \tag{9}$$

where $\mathscr{R}_{[0,1]^n}(\mathbf{m})$ is the indicator function such that

$$\mathscr{R}_{[0,1]^n}(\mathbf{m}) = \begin{cases} 0 & \text{if } \mathbf{m} \in [0,1]^n \\ \infty & \text{otherwise} \end{cases}.$$

The unconstrained minimizer $\mathbf{m}_u$ of (9) is

$$\mathbf{m}_u = \mathrm{Proj}_{[0,1]^n}(\mathbf{z}). \tag{10}$$

This projection is component-wise and can be calculated easily since each component $\mathrm{Proj}_{[0,1]}(\mathbf{z}_i)$ takes the middle value among $0, 1$, and $\mathbf{z}_i$. If $\mathbf{1}_n^T\mathbf{m}_u \le \Delta$, then $\mathbf{m}_u$ is the minimizer of (9) and $\mathbf{m}^{t+1} = \mathbf{m}_u$. Now suppose $\mathbf{m}_u$ is not feasible, i.e., $\mathbf{1}_n^T\mathbf{m}_u > \Delta$. Consider the Lagrangian function associated with (9), which is given by

$$\begin{aligned} \mathscr{L}(\mathbf{m},\lambda) &= \tfrac{1}{2}\|\mathbf{m}-\mathbf{z}\|_2^2 + \mathscr{R}_{[0,1]^n}(\mathbf{m}) + \lambda(\mathbf{1}_n^T\mathbf{m} - \Delta) \\ &= \left\{\sum_{i=1}^n \left(\tfrac{1}{2}(m_i - z_i)^2 + \mathscr{R}_{[0,1]}(m_i) + \lambda m_i\right)\right\} - \lambda\Delta \end{aligned} \tag{11}$$

where $\lambda \in \mathbb{R}$ is the Lagrange multiplier associated with the constraint. Setting the partial derivatives of the Lagrangian with respect to $\mathbf{m}$ and λ equal to zero, we see that the optimal solution is given by

$$\mathbf{m} = \mathrm{Proj}_{[0,1]^n}(\mathbf{z} - \lambda\mathbf{1}_n) \tag{12}$$

(see [20] for details). Note that as λ tends to $z_{\max} = \max\{\mathbf{z}_i\}$, $\mathbf{m}$ tends to $\mathbf{0}$, which is feasible with respect to the inequality constraint in (9). Similarly, as λ tends to 0, $\mathbf{m}$ tends to $\mathbf{m}_u$, which is not feasible by assumption. To compute the optimal λ, we use the corresponding Karush-Kuhn-Tucker (KKT) conditions:

$$\lambda(\mathbf{1}_n^T\mathbf{m} - \Delta) = 0 \tag{13}$$

$$\lambda \ge 0 \tag{14}$$

$$\mathbf{1}_n^T\mathbf{m} \le \Delta \tag{15}$$

(see [23] for details). Since $\mathbf{m}_u$ is not feasible, $\lambda \ne 0$. Letting

$$f(\lambda) = \mathbf{1}_n^T \mathrm{Proj}_{[0,1]^n}(\mathbf{z} - \lambda\mathbf{1}_n) - \Delta, \tag{16}$$

we see that (13) implies $f(\lambda) = 0$ at the optimal λ. To find the root of $f(\lambda)$ in (16), we use the bisection method, which applies since $f(0) > 0$ (because $\mathbf{1}_n^T\mathbf{m}_u > \Delta$ by assumption) and $f(z_{\max}) < 0$ (see [20] for further details).

Finally, since the variable $\mathbf{m}$ can be interpreted as a probabilistic vector. We recover a binary solution from $\mathbf{m}$ using random sampling [8], [20]. Details are given in Algorithm 1.

Algorithm 1: Random sampling of a perturbation vector

```
Input: probabilistic vector $\mathbf{m}$, J is # of random trials
Output: binary perturbation vector $\mathbf{m}^*$
1 for $j=1$ to $J$ do
2     draw binary vector $\mathbf{r}^{(j)}$ following
3         $r_i^{(j)} = \begin{cases} 1 & \text{with probability } m_i \\ 0 & \text{with probability } 1 - m_i \end{cases}$
4 Select a vector $\mathbf{m}^*$ from $\{\mathbf{r}^{(j)}\}$ that yields the smallest
    attack loss $FL(\mathbf{m}^*)$
```

We summarize the PM attack in Algorithm 2. We note that the key differences between our proposed method and those from [20] is (i) the use of the focal loss as an attack loss function and (ii) the use of momentum in the optimization algorithm to minimize the loss.

Our defense strategy is done by leveraging our proposed robust training of a graph convolutional networks (GCN) against PM attacks (the resulted perturbed adjacency matrix A'). We present our experimental results for both PM attack and adversarial training defense on a GCN.

III. Experiments

In this section, we evaluate the effectiveness of FL-PM against different graph adversarial attacks. Before presenting our empirical results and findings, we first introduce the experimental settings.

Algorithm 2: Focal Loss-Projected Momentum (FL-PM) Attack on GNN

```
Input: $\mathbf{m}^{(0)}$, learning rate $\eta$, and iterations $T$
Output: Perturbed adjacency matrix $A'$
1 for $t$=1 to $T$ do
2     Momentum:
3         $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1-\beta)\nabla FL(\mathbf{m})$
4         $\mathbf{m}^{t+1} = \mathbf{m}^t - \eta \mathbf{v}_{t+1}$
5     Projection using (10) or (16):
6         $\mathrm{Proj}_{\mathscr{M}}\left[\mathbf{m}^{t+1}\right]$
7 Random sampling using Algorithm 1 to return m*
8 Compute $\delta$ using (4) and add it to $A$ to return $A'$
```

A. Experimental settings

1) Datasets and model design: Following [11], [23], we validate the proposed methodology using two benchmark datasets: citation graphs Cora and Citeseer [25]. Both datasets consist of unweighted edges, represented by a symmetric adjacency matrix, denoted as A, and sparse feature vectors. These feature vectors are utilized as the input for GCN. Our GCN comprises two layers, each with 16 hidden units. The Rectified Linear Unit (ReLU) activation function is applied between the layers, and the Softmax function is employed to produce soft labels as the output. All training is done using 200 epochs with learning rate 0.01. For each dataset, we randomly choose 10% of nodes for training, 10% of nodes for validation and the remaining 80% for test. We repeat each experiment for 5 times and report the average performance. The iteration number of random sampling is set as $K = 20$.

2) Baselines: To assess the effectiveness of our approach, we conduct a comprehensive evaluation by comparing our proposed attack model, Focal Loss-Projected Momentum (FL-PM), with state-of-the-art GNN attacks and defense models using the adversarial attack repository DeepRobust [26]. Specifically, we compare it with CE-PGD, CW-PGD, CE-min-max, and CW-min-max [20]. These are optimization-based attacks where CE-PGD generates node perturbations by minimizing cross-entropy loss through projected gradient descent. Additionally, we compare our method with DICE ('delete edges internally, connect externally') [27], Meta-Self attack [28], and a greedy attack – a variant of Meta-Self attack without weight re-training for GCN.

TABLE I
Misclassification rates (%) under 10% perturbed edges

Attack	**Cora**	**Citeseer**
Clean	16.8±0.11	25.5±0.08
DICE	18.9±0.13	26.9±0.07
Greedy	20.3±0.20	27.1±0.05
Meta-Self	19.7±0.14	26.1±0.06
CE-PGD	21.2±0.15	27.4±0.03
CW-PGD	17.4±0.12	26.0±0.05
CE-min-max	21.4±0.16	26.3±0.05
CW-min-max	20.8±0.13	26.1±0.06
FL-PM (ours)	**22.9±0.13**	**27.7±0.02**

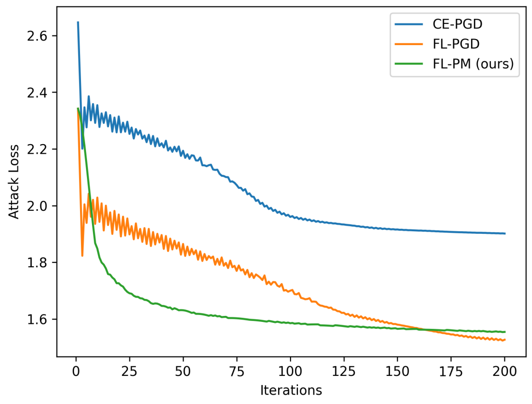


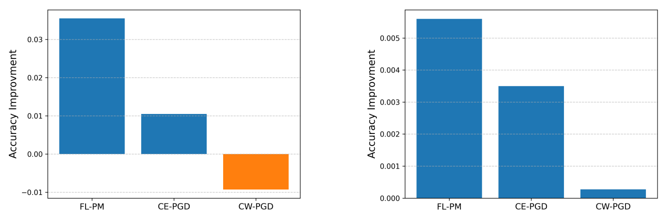
Fig. 2. CE-PGD, FL-PGD, and FL-PM attack losses convergence on Cora. The perturbation rate is 10%.

B. Attack Performance

In Table I, we showcase the misclassification rates ($(1 - \text{Accuracy}) \times 100\%$) of various attack methods when tested against the natural model (GCN trained on clean data, i.e., A). The natural model is evaluated on perturbed data generated by different attacks. The effectiveness of each attack is measured by its ability to deceive the natural model, leading to an increased misclassification rate. As demonstrated in the table, our attack proves to be the most effective, successfully deceiving the model with a misclassification rate that reaches approximately 23% on the Cora dataset. The closest competitor, CE-PGD, as anticipated, achieved a misclassification rate of approximately 21%. Surprisingly, the performance of CW-PGD was notably lower compared to the other methods. We additionally visualize the convergence of the attack loss function for our method, CE-PGD, and focal loss function with the PGD solver. Figure 2 illustrates the relative monotonic convergence behavior of our method, characterized by damped oscillations and faster convergence to a local minima, demonstrating the effectiveness of our approach.

C. Defense Performance

We also trained the GCN on perturbed data generated by our model and compared the performance on a testing perturbed data with different models. We computed the improvement on accuracy which is the difference between a accuracy of a GCN trained on clean data (natural model) and tested on perturbed data (i.e., attacked model) and a GCN trained and tested on perturbed data (robust model). Experiments are done on both datasets. Figure 3 shows the improvements on accuracy obtained after applying our defense strategy (i.e., adversarial training) using perturbations obtained from our model and different models. Notably, we observed a decrease in performance when the GCN was trained on perturbed data



(a) Cora (b) Citeseer

Fig. 3. Improvements on accuracy after training GCN on different attacks for both datasets.

TABLE II
Average of five runs classification accuracy for different perturbation rates of robust GCN trained and tested on perturbed data generated from the methods in mentioned in columns. 0% mean that the model is trained on clean data.

Dataset	Pert. Ratio	CE-PGD	CW-PGD	FL-PM
Cora	0%	0.8320	0.8320	0.8320
	5%	0.8123	0.8024	0.8209
	10%	0.7871	0.8069	0.8033
	15%	0.7746	0.7434	0.7706
	20%	0.7555	0.7645	0.7691
	25%	0.7414	0.7245	0.7434
Citeseer	0%	0.7446	0.7446	0.7446
	5%	0.7399	0.7452	0.7386
	10%	0.7292	0.7332	0.7384
	15%	0.7055	0.7452	0.7340
	20%	0.7020	0.7305	0.7273
	25%	0.6889	0.7133	0.7181

generated by CW-PGD. This implies that the natural model exhibits better behavior compared to a robust model trained and tested on this perturbed data. To assess the behavior of a robust model under varying rates of perturbations, we calculated the accuracy at different perturbation rates and compared it to models trained and tested on CE-PGD and CW-PGD. Overall, our model exhibited outperforms robust models subjected to these attacks. Results are shown in Table II.

IV. Conclusion

In conclusion, our study addresses the pressing vulnerability of Graph Neural Networks (GNNs) to adversarial attacks on graph structures, crucial in diverse AI applications. The proposed optimization-based attacks, utilizing projected momentum optimization, specifically target graph structure modifications. By incorporating the focal loss as an attack criterion and minimizing a constrained optimization problem, our approach generates perturbations that outperform state-of-the-art methods in node classification tasks under the same perturbation budget. The complexities inherent in attacking discrete graph structures are effectively tackled through our approach, drawing inspiration from recent advancements in adversarial attacks on continuous data. The demonstrated improvements of our attacks underscores their efficiency in fortifying GNNs against adversarial manipulations, providing valuable insights for enhancing the security of graph-structured data in real-world applications where safety and privacy are paramount.

References

[1] T. N. Kipf and M. Welling, ‘Semi-Supervised Classification with Graph Convolutional Networks’, CoRR, vol. abs/1609.02907, 2016.
[2] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, ‘Graph Attention Networks’, arXiv [stat.ML]. 2018.
[3] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, ‘How Powerful are Graph Neural Networks?’, CoRR, vol. abs/1810.00826, 2018.
[4] M. Aburidi and R. Marcia, ‘Wasserstein Distance-Based Graph Kernel for Enhancing Drug Safety and Efficacy Prediction’, in 2024 IEEE First IEEE International Conference on AI for Medicine, Health, and Care (AIMHC), 2024.
[5] M. Aburidi and R. F. Marcia, ‘Optimal Transport-Based Graph Kernels for Drug Property Prediction’, Scientific reports, 2024.
[6] W. L. Hamilton, R. Ying, and J. Leskovec, ‘Inductive Representation Learning on Large Graphs’, CoRR, vol. abs/1706.02216, 2017.
[7] H. Dai et al., ‘Adversarial Attack on Graph Structured Data’, CoRR, vol. abs/1806.02371, 2018.
[8] S. Liu, S. P. Chepuri, M. Fardad, E. Maşazade, G. Leus, and P. K. Varshney, ‘Sensor Selection for Estimation with Correlated Measurement Noise’, IEEE Transactions on Signal Processing, vol. 64, no. 13, pp. 3509–3522, 2016.
[9] H. Liang, E. He, Y. Zhao, Z. Jia, and H. Li, ‘Adversarial Attack and Defense: A Survey’, Electronics, vol. 11, no. 8, 2022.
[10] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, ‘The Vulnerabilities of Graph Convolutional Networks: Stronger Attacks and Defensive Techniques’, CoRR, vol. abs/1903.01610, 2019.
[11] D. Zügner and S. Günnemann, ‘Adversarial Attacks on Graph Neural Networks via Meta Learning’, arXiv [cs.LG]. 2019.
[12] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, ‘Neural Message Passing for Quantum Chemistry’, CoRR, vol. abs/1704.01212, 2017.
[13] H. Xu et al., ‘Adversarial Attacks and Defenses in Images, Graphs and Text: A Review’, CoRR, vol. abs/1909.08072, 2019.
[14] M. Aburidi and R. Marcia, ‘CLOT: Contrastive Learning-Driven and Optimal Transport-Based Training for Simultaneous Clustering’, in 2023 IEEE International Conference on Image Processing (ICIP), 2023, pp. 1515–1519.
[15] N. Carlini and D. A. Wagner, ‘Audio Adversarial Examples: Targeted Attacks on Speech-to-Text’, CoRR, vol. abs/1801.01944, 2018.
[16] K. Xu et al., ‘Interpreting Adversarial Examples by Activation Promotion and Suppression’, CoRR, vol. abs/1904.02057, 2019.
[17] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, ‘EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples’, arXiv [stat.ML]. 2018.
[18] H. Chen, H. Zhang, P.-Y. Chen, J. Yi, and C.-J. Hsieh, ‘Show-and-Fool: Crafting Adversarial Examples for Neural Image Captioning’, CoRR, vol. abs/1712.02051, 2017.
[19] T.-Y. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, ‘Focal Loss for Dense Object Detection’, CoRR, vol. abs/1708.02002, 2017.
[20] K. Xu et al., ‘Topology Attack and Defense for Graph Neural Networks: An Optimization Perspective’, CoRR, vol. abs/1906.04214, 2019.
[21] I. J. Goodfellow, J. Shlens, and C. Szegedy, ‘Explaining and Harnessing Adversarial Examples’, arXiv [stat.ML]. 2015.
[22] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, ‘Improving neural networks by preventing co-adaptation of feature detectors’, CoRR, vol. abs/1207.0580, 2012.
[23] S. P. Boyd and L. Vandenberghe, Convex optimization. Cambridge university press, 2004.
[24] D. Zügner and S. Günnemann, ‘Adversarial Attacks on Graph Neural Networks via Meta Learning’, arXiv [cs.LG]. 2019.
[25] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, ‘Collective classification in network data’, AI magazine, vol. 29, no. 3, pp. 93–93, 2008.
[26] Y. Li, W. Jin, H. Xu, and J. Tang, ‘DeepRobust: A PyTorch Library for Adversarial Attacks and Defenses’, CoRR, vol. abs/2005.06149, 2020.
[27] M. Waniek, T. P. Michalak, T. Rahwan, and M. J. Wooldridge, ‘Hiding Individuals and Communities in a Social Network’, CoRR, vol. abs/1608.00375, 2016.
[28] K. Xu et al., ‘Structured Adversarial Attack: Towards General Implementation and Better Interpretability’, CoRR, vol. abs/1808.01664, 2018.