

Adversarial Attacks on Neural Networks for Graph Data

Author: Daniel Zugner

Technical University of Munich

Presenter: Faizan Ahmad

<https://qdata.github.io/deep2Read>

Outline

- 1 Introduction
- 2 Graph Convolutional Networks Overview
- 3 Related Work
- 4 Attack Model
- 5 Generating Adversarial Graphs
- 6 Evaluation and Results
- 7 Takeaways
- 8 Discussion

Introduction

- What are Adversarial Attacks?
- Success of Adversarial Attacks
 - Images
 - Text
 - **Graphs?** Adversaries in graph-based domains.
- Adversarial Attacks on Node Classification by changing:
 - Graph Structure
 - Node Features

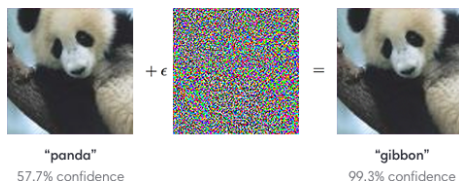


Figure: Adversarial Attack Example. Carefully crafted image is added to the input to make the model misclassify it.

Basic Idea

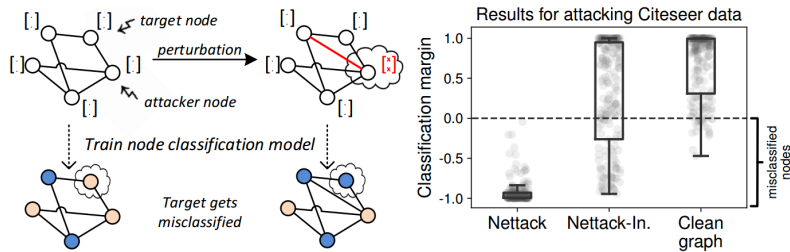


Figure: Small perturbations of the graph structure or node features lead to misclassification of the target

Attacker Node: Manipulated nodes

Target Node: Nodes that we want to misclassify

Why Attacks on Graphs?

- Large body of work on adversarial attacks for Images and Text
 - DeepFool (Moosavi et al. [1])
 - Adversarial Examples in Physical World (Kurakin et al. [2])
 - Hotflip (Ebrahimi et al. [3])
 - Plenty more..
- No work on adversarial attacks for graphs
- Many new challenges
 - Discrete Domain
 - Transductive Learning Setting
 - Network Effects

Task Setup

Node Classification

- Semi-supervised Node Classification via Binary Node Features
- Attributed Graph $G = (A, X)$
 - Adjacency Matrix representing edges $A \in \{1, 0\}^{N \times N}$
 - Feature Matrix representing D dimensional node features $X \in \{0, 1\}^{N \times D}$
 - Node-ids $V = \{1, 2, \dots, N\}$
 - Feature-ids $F = \{1, 2, \dots, D\}$
- Given: A subset $V_L \subseteq V$ of labeled nodes with labels $C = \{1, \dots, c_k\}$
- Task: Learn $g : V \rightarrow C$

Graph Convolutional Networks Overview (GCN) [4]

- Node classification is done via Kipf et al. [4]
 - Learns node-level output features
 - Graph level features can be obtained via aggregating
- A hidden layer $l + 1$ is defined as $H^{(l+1)} = f(H^{(l)}, A)$ where $H^{(0)} = X$
- Simple GCN propagation rule is $f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$
 - A is not normalized
 - No self-loops
- Proposed fixes in [4]:

$$f(H^{(l)}, A) = \sigma(\hat{D}^{-1/2} \hat{A} \hat{D}^{1/2} H^{(l)} W^{(l)}) \quad (1)$$

$$\hat{A} = A + I \quad (2)$$

$$Z = f_{\theta}(A, X) = \text{softmax}(\tilde{A} \sigma(\tilde{A} X W^{(1)}) W^{(2)}) \quad (3)$$

- Graph Neural Networks
 - Node Embeddings/Classification [5]
 - Graph Classification [6]
- Adversarial Attacks
 - Evasion Attacks [1]
 - Poisoning Attacks [7]
- Adversarial Attacks on Graphs
 - Using Greedy Approximation (This Work)
 - Via Reinforcement Learning [8]

Attack Model

Terminologies

- Original Graph $G^{(0)} = (A^{(0)}, X^{(0)})$
- Perturbed Graph $G^* = (A^*, X^*)$
- Two type of attacks:
 - Structure Attacks \rightarrow Changes to $A^{(0)}$
 - Feature Attacks \rightarrow Changes to $X^{(0)}$
- Two types of nodes:
 - Target Node $v_0 \rightarrow$ Node we aim to misclassify \rightarrow Direct Attack
 - Attacker Nodes $\mathcal{A} \rightarrow$ Nodes we aim to manipulate \rightarrow Influencer Attack

Attack Model

Problem Statement

$$\operatorname{argmax}_{(A^*, X^*) \in \mathbf{P}} \max_{c \neq c_{old}} (\ln Z_{v_0, c} - \ln Z_{v_0, c_{old}})$$

subject to $Z = f_{\theta^*}(A^*, X^*)$ with $\theta^* = \operatorname{argmin}_{\theta} L(\theta, A^*, X^*)$

- **Bi**-level optimization problem.
- \mathbf{P} is the set of graphs with changes made under a budget Δ

$$\mathbf{P} \in \sum_u \sum_i |X_{ui} - X_{ui}^*| + \sum_{u < v} |A_{uv} - A_{uv}^*| \leq \Delta \quad (4)$$

Attack Model

Graph Structure Preserving Perturbations

- Degree distribution
- Easy to tell the difference with different degree distributions
- Generate perturbations that *preserve* power-law $p(x) \propto x^{-\alpha}$ behavior of degree distribution
- Likelihood (Eq. 6) ratio test for degree distribution by *approximately* estimating α

$$\alpha_G \approx 1 + |D_G| \cdot \left[\sum_{d_i \in D_G} \log(d_i) - \log(d_{min} - 0.5) \right]^{-1} \quad (5)$$

$$l(D_x) = |D_x| \cdot \log(\alpha_x) + |D_x| \cdot \alpha_x \cdot \log d_{min} + (\alpha_x + 1) \sum_{d_i \in D_x} \log d_i \quad (6)$$

Attack Model

Feature Statistics Preserving Perturbations

- Main Question: Which features can be set to 1?
 - Why don't we care about 0?
- Test based on feature co-occurrence
- Co-occurrence graph $C = (F, E)$.
 - F is set of features.
 - $E \subseteq F \times F$ denote feature co-occurrence
- Probabilistic Random Walk on C
- Only add feature i if for $S_u = \{j | X_{uj} \neq 0\}$

$$p(i|S_u) = \frac{1}{|S_u|} \sum_{j \in S_u} (1/d_j) \cdot E_{ij} > \sigma \quad (7)$$

Generating Adversarial Graphs

- Bi-level optimization for discrete case is hard
- Introduce a **surrogate model** with linearization
 - $Z = \text{softmax}(A^* XW)$
- Surrogate loss

$$L_s(A, X; W, v_0) = \max_{c \neq c_{old}} [\hat{A}^2 XW]_{v_0 c} - [\hat{A}^2 XW]_{v_0 c_{old}} \quad (8)$$

- Loss maximization to obtain adversarial graph

$$\operatorname{argmax}_{(A^*, X^*) \in P} L_s(A^*, X^*; W, v_0) \quad (9)$$

This is still **intractable** due to discreteness

Generating Adversarial Graphs

Greedy Approximation

- Define scoring functions for greedy approximation of Eq. 9
- For $A^* = A \pm e$ and $X^* = X \pm f$

$$s_{\text{struc}}(e, G, v_0) = L_s(A^*, X; W, v_0) \quad (10)$$

$$s_{\text{feat}}(f, G, v_0) = L_s(A, X^*; W, v_0) \quad (11)$$

- Fast computation for the scores

Generating Adversarial Graphs

Greedy Approximation Pseudo Code

Algorithm 1: NETTACK: Adversarial attacks on graphs

Input: Graph $G^{(0)} \leftarrow (A^{(0)}, X^{(0)})$, target node v_0 ,
attacker nodes \mathcal{A} , modification budget Δ

Output: Modified Graph $G' = (A', X')$

Train surrogate model on $G^{(0)}$ to obtain W // Eq. 13;

$t \leftarrow 0$;

while $|A^{(t)} - A^{(0)}| + |X^{(t)} - X^{(0)}| < \Delta$ **do**

$C_{struct} \leftarrow \text{candidate_edge_perturbations}(A^{(t)}, \mathcal{A})$;

$e^* = (u^*, v^*) \leftarrow \arg \max_{e \in C_{struct}} s_{struct}(e; G^{(t)}, v_0)$;

$C_{feat} \leftarrow \text{candidate_feature_perturbations}(X^{(t)}, \mathcal{A})$;

$f^* = (u^*, i^*) \leftarrow \arg \max_{f \in C_{feat}} s_{feat}(f; G^{(t)}, v_0)$;

if $s_{struct}(e^*; G^{(t)}, v_0) > s_{feat}(f^*; G^{(t)}, v_0)$ **then**

$G^{(t+1)} \leftarrow G^{(t)} \pm e^*$;

else $G^{(t+1)} \leftarrow G^{(t)} \pm f^*$;

$t \leftarrow t + 1$;

return $G^{(t)}$

// Train final graph model on the corrupted graph $G^{(t)}$;

Experimental Setup

- Three Graph data sets; CORA-ML, CITESEET, POL. BLOGS
- 20% labeled nodes, 80% unlabeled
- For adversarial graph generation
 - Train on uncorrupted data
 - Pick 40 correctly classified nodes as *target nodes*
 - Pick 5 random neighbors as *attacker nodes*
- Two corruption methods
 - NETTACK (Direct Attack)
 - NETTACK-IN (Influence Attack)
- Comparison Methods
 - Fast Gradient Sign Method (FGSM)
 - Random Modification

Results

Attacks on Surrogate Model

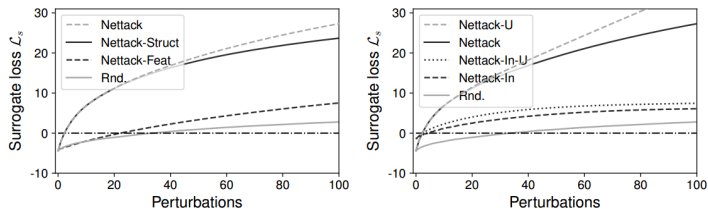


Figure: Average surrogate loss for increasing number of perturbations. Different variants of our method on the Cora data. Larger is better

Results

Attacks on Surrogate Model

Class: neural networks				Class: theory				Class: probabilistic models			
constrained		unconstrained		constrained		unconstrained		constrained		unconstrained	
probabilistic	25	efforts	2	driven	3	designer	0	difference	2	calls	1
probability	38	david	0	increase	8	assist	0	solve	3	chemical	0
bayesian	28	averages	2	heuristic	4	disjunctive	7	previously	12	unseen	1
inference	27	accomplished	3	approach	56	interface	1	control	16	corporation	3
probabilities	20	generality	1	describes	20	driven	3	reported	1	fourier	1
observations	9	expectation	10	performing	7	refinement	0	represents	8	expressed	2
estimation	35	specifications	0	allow	11	refines	0	steps	5	robots	0
distributions	21	family	10	functional	2	starts	1	allowing	7	achieving	0
independence	5	uncertain	3	11	3	restrict	0	task	17	difference	2
variant	9	observations	9	acquisition	1	management	0	expressed	2	requirement	1

Figure: Top-10 **feature** perturbations per class on Cora

Results

Transferability of Attacks

- Generate adversarial graphs on surrogate model.
- Test on benchmark Graph Neural Networks
 - Evasion Attack: Train on clean data, keep parameters fixed, attack.
 - Poisoning Attack: Train on adversarial data and measure accuracy.

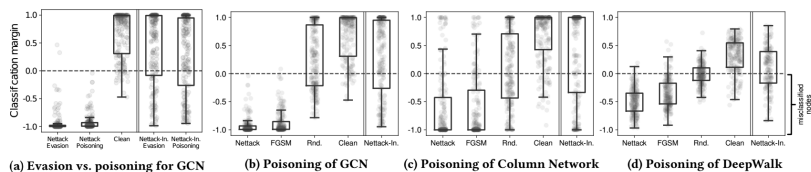


Figure: Results on Cora data using different attack algorithms. Clean indicates the original data. Lower scores are better

Results

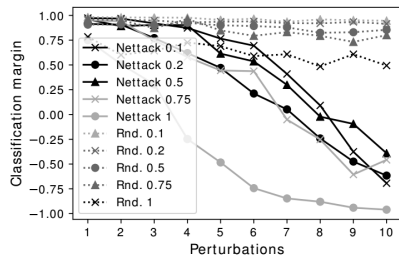
Transferability of Poisoning Attacks

<i>Attack method</i>	Cora			Citeseer			Polblogs		
	GCN	CLN	DW	GCN	CLN	DW	GCN	CLN	DW
Clean	0.90	0.84	0.82	0.88	0.76	0.71	0.93	0.92	0.63
NETTACK	0.01	0.17	0.02	0.02	0.20	0.01	0.06	0.47	0.06
FGSM	0.03	0.18	0.10	0.07	0.23	0.05	0.41	0.55	0.37
RND	0.61	0.52	0.46	0.60	0.52	0.38	0.36	0.56	0.30
NETTACK-IN	0.67	0.68	0.59	0.62	0.54	0.48	0.86	0.62	0.91

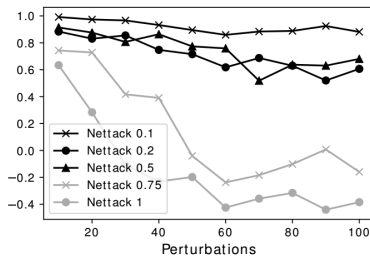
Figure: Overview of results. Smaller is better

Results

Transferability of Poisoning Attacks



(a) Direct attack



(b) Influence attack




Figure: Attacks with limited knowledge about the data

Takeaways

- Adversarial attacks on graphs are more challenging
- Attacks on linear systems transfer well to non-linear systems
- Limited knowledge of graph is sometimes enough to mount attacks
- Greedy approaches usually work amazingly!
- Black box attacks are successful too.

- How to handle continuous features?
- How to handle weighted edges?
- What happens after adversarial training of graph neural networks?
- What happens if we apply the greedy approximation on non-linear model?

-  S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: a simple and accurate method to fool deep neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2574–2582, 2016.
-  A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *arXiv preprint arXiv:1607.02533*, 2016.
-  J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, “Hotflip: White-box adversarial examples for text classification,” *arXiv preprint arXiv:1712.06751*, 2017.
-  T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
-  B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, ACM, 2014.

-  Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated graph sequence neural networks,” *arXiv preprint arXiv:1511.05493*, 2015.
-  B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, “Data poisoning attacks on factorization-based collaborative filtering,” in *Advances in neural information processing systems*, pp. 1885–1893, 2016.
-  H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, “Adversarial attack on graph structured data,” *arXiv preprint arXiv:1806.02371*, 2018.