An Efficient Adversarial Attack on Graph Structured Data

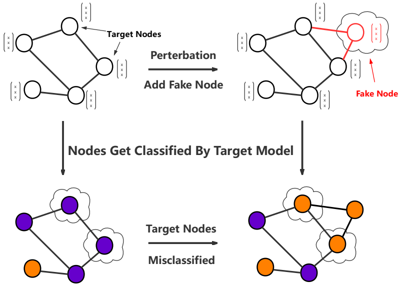
Zhengyi Wang, Hang Su

Tsinghua University

Jan. 7th, 2021

Graph Adversarial Attack

- Graph Adversarial Attack = Graph Neural Network + Adversarial Attack.
- Add *fake nodes* and extra connections.
- Lower the performance of node classification.



Problem Formulation

- Given $G = (A, X)$, where $A \in \{0,1\}^{N \times N}$ and $X \in \{0,1\}^{N \times D}$
- Target node set $V_{target} \subseteq V$.
- Add N_{fake} fake nodes, leading to $G^+ = (A^+, X^+)$. where $A^+ = \begin{bmatrix} A & B^T \\ B & A_{fake} \end{bmatrix}$ and $X^+ = \begin{bmatrix} X \\ X_{fake} \end{bmatrix}$. (Mild perturbation.)
- Minimize the adversarial loss $\min_{A_{fake}, B, X_{fake}} \mathbb{L}(G^+, V_{target}) \quad s.t. ||B||_0 \leq \Delta_{edge}$

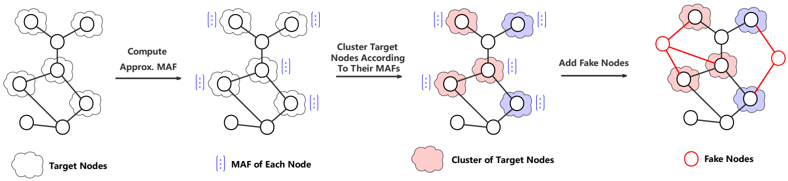
Challenge

$$\min_{A_{fake}, B, X_{fake}} \mathbb{L}(G^+, V_{target}) \quad s.t. ||B||_0 \leq \Delta_{edge}$$

- Discrete optimization. Gradient method is prohibitive.
- Relax $A^+ \in \{0,1\}^{(N+N_{fake})\times(N+N_{fake})}$ to $A^+ \in [0,1]^{(N+N_{fake})\times(N+N_{fake})}$ to use PGD (Xu et al., 2019) $\Rightarrow$ loss of accuracy.
- RL-based method (Sun et al., 2019)/Greedy search (Wang et al., 2018) $\Rightarrow$ time consuming.

Algorithm

- Motivation - divide target nodes into several clusters to gain better performance and lower the complexity.
- Target nodes in same cluster are supposed share a certain kind of similarity.
- How to denote this similarity? $\Rightarrow$ *Most Adversarial Feature*.
- Features of fake nodes derive from cluster centers of Most Adversarial Feature.



Experiment Results

表 1: Success rate of targeted attack adding 4 fake nodes. T denotes number of target nodes.

Method	Cora $T=3$	Cora $T=5$	Cora $T=7$	Cora $T=10$	Citeseer $T=3$	Citeseer $T=5$	Citeseer $T=7$	Citeseer $T=10$
Random	0.07	0.08	0.04	0.05	0.04	0.02	0.03	0.03
NETTACK	0.61	0.57	0.55	0.53	0.75	0.71	0.66	0.61
Sequential Greedy	0.68	0.73	0.72	0.70	0.76	0.74	0.72	0.67
Cluster Attack	**0.99**	**0.93**	**0.84**	**0.72**	**1.00**	**0.89**	**0.80**	**0.70**

- Generally, our algorithm outperforms existing baselines.

Thanks!