Adversarial Attacks on Graphs by Adding Fake Nodes

Yu Chen, Zhiling Luo, Sha Zhao, Ying Li*, Jianwei Yin
College of Computer Science and Technology, Zhejiang University, Hangzhou, China
{cheny_17, luozhiling, szhao, cnliying, zjuyjw}@zju.edu.cn

Abstract

Deep neural network models for graph structures have achieved great performance on many tasks, e.g. node classification and link prediction. Despite the success on performance, deep neural network models' robustness has attracted much attention in recent years. Previous researches have shown that these models can be attacked by manipulating existing edges and features. Changing the edges and features, however, is hard in practice due to the restrict security policies. We witness that adding fake nodes is a more accessible way to attack the models. In this work, we study the adversarial attacks on graph structures by adding fake nodes to change the classification result on the target node. We propose a reinforcement learning method, *g-Faker*, for black-box attack, and a gradient-based greedy method, *GradAtt*, for white-box attack, both of which are designed to add fake nodes to graphs. The experimental results on multiple datasets demonstrate the effectiveness and the transferability of our methods.

Introduction

As an important and widely used data representation, graph has been used in a wide range of real-world application scenarios, e.g. diffusion graph in social networks (Faliszewski et al. 2018), user preference graph in e-commerce (LE, Lauw, and Fang 2017), and proteins structure graph in biology (LI and Yu 2016). Modeling the data as a graph structure, a series of graph analyzing techniques are leveraged, e.g. graph clustering, and node embedding (Perozzi, Al-Rfou, and Skiena 2014; Dai, Dai, and Song 2016). Among them, node classification is one of the most important tasks on graphs — given a large (attributed) graph with partially labeled nodes, the goal is to predict the labels of the rest nodes (Bhagat, Cormode, and Muthukrishnan 2011). For example, by taking a person's interests in movies as the labels, one can infer the interests of the target people in social networks, and recommend relevant movies to them accordingly (Zhao et al. 2018). Like most classification problems, graph node classification can be solved by various deep neural network models. In particular, Graph Convolutional Networks (GCNs) have achieved a state-of-the-art performance (Kipf and Welling 2016).

However, many deep neural network models have been proved to be vulnerable — with unnoticeable perturbations of an instance, the predictions can be misguided, which is known as *adversarial attack* (Goodfellow, Shlens, and Szegedy). Recently adversarial attacks on graph structure data have drawn more and more attention. The work (Dai et al. 2018; Zügner, Akbarnejad, and Günnemann 2018; Zügner and Günnemann 2019) show that changing a small number of edges and features can make deep neural network models easily fooled/attacked. In many real-world problems, however, it may be difficult to alter the edges and features in a graph. For example, in a graph structure for an online social network, in order to maliciously change the edges, an attacker needs to hack different accounts in the network websites to tamper with the content. It is difficult to obtain the login access to so many different accounts. Compared with changing edges in a graph structure, adding fake nodes is more easily in practice, which just needs to create new accounts and take them to build connections in the graph.

One key challenge to attack the deep neural networks by adding fake nodes to the graph is that, it is hard to evaluate the effectiveness. The added fake nodes are required to connect to the nodes in the original graph and it will lead to cascading effects, which makes results unpredictable. In order to overcome this challenge, we propose a reinforcement learning approach, which contains a value function to evaluate the effectiveness of adding a fake node.

In this paper, we propose a reinforcement learning based model named *g-Faker* for black-box attack and a white-box attack method *GradAtt*, respectively, to attack the state-of-the-art deep learning models for graphs by adding fake nodes. Specifically, we focus on the models based on graph convolutions, such as GCN (Kipf and Welling 2016). We assume attackers have full knowledge of the graph structure. Moreover, our experiments conducted in terms of transferability show that our attack policies can generalize to other models as well. Over all, our contributions are summarized below:

- We propose a reinforcement learning based method named *g-Faker* for black-box attack and a gradient based

greedy method *GradAtt* for white-box attack, respectively, to design fake nodes for adversarial attacks on node classification.

- The experimental results show that our methods effectively misguide the prediction of the target nodes to the target label with only few changes to the graph. The attack policy learned on one deep neural network model can work well on others.

Related Work

In this section, we briefly review the literature about adversarial attacks on graphs.

Several studies have performed an adversarial attack on graph neural networks. (Dai et al. 2018) attacks graph neural networks (GNNs) by altering a few edges in the graph in a reinforcement learning way. They also employ genetic algorithm and gradient-based methods. (Zügner, Akbarnejad, and Günnemann 2018) scores the perturbations based on the surrogate model classification margin and select the top one. Different from the adversarial attack on images, the unnoticeable manipulation can be hard to define, since the graph structure has been modified. Both (Dai et al. 2018) and (Zügner, Akbarnejad, and Günnemann 2018) limit the cost of changing graphs, while Zugner further tries to preserve the distribution of degrees and the co-occurrence of features to make the perturbations unnoticeable. In (Zügner and Günnemann 2019), they use meta-learning to solve the bilevel optimization problem in the discrete graph data and perform training-time attack. What's more, adversarial attacks on node embeddings are investigated by (Bojchevski and Günnemann 2019). However, all these work focus on perturbing existing nodes and their relations.

In this work, we propose a reinforcement learning based algorithm and a gradient based method to perform the attacks on GCN by adding fake nodes. We focus on targeted attacks, which is more difficult than the non-targeted attacks.

Preliminary

We consider the task of node classification in a large (attributed) graph in this work. Formally, the graph is denoted by $G = (A, X)$, where $A \in \{0,1\}^{N\times N}$ is the adjacency matrix of the graph and $X \in \{0,1\}^{N\times D}$ is the features for each node, where D is the dimension of features. Each node $v_i \in V$ with feature $x_i \in \{0,1\}^D$ is associated with a label $y_i \in \mathcal{Y}$, where $\mathcal{Y}$ is the label set.

Given a subset of nodes $V_T \subseteq V$ and corresponding labels from $\mathcal{Y}$, the goal of node classification is to train a classifier $f(\cdot; G) : V \longmapsto \mathcal{Y}$ that maps each node $v \in V$ to a class in $\mathcal{Y}$ and minimizes the following loss

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} L(f(v_i; G), y_i)$$

where L is the distance defined in $\mathcal{Y}$, like $l2$ norm.

In this work, we set the method GCN with two convolutional layers proposed by (Kipf and Welling 2016) as our attack target, which is formulated as:

$$Z = f_\Theta(A, X) = softmax(\hat{A}\sigma(\hat{A}XW^0)W^1)$$

where $\hat{A} = \tilde{D}^{-\frac{1}{2}}(I_N + A)\tilde{D}^{-\frac{1}{2}}$, $\tilde{D}_{ii} = \sum_j A_{ij} + 1$, $\sigma(\cdot)$ is an activation function and $\Theta = \{W^0, W^1\}$ is the parameter set of the classifier. Each row of output $Z \in \mathbb{R}^{N\times|\mathcal{Y}|}$ represents the classification probabilities of the node over labels.

Attack Model

Given the GCN model, our goal is to insert m fake nodes on the graph $G = (A, X)$, leading to the graph $G' = (A', X')$, where $A' = \begin{bmatrix} A & B^T \\ B & C \end{bmatrix}$ and $X' = \begin{bmatrix} X \\ P \end{bmatrix}$, such that the node classification performance drops. More specifically, we aim to change the prediction label of a correctly classified target node v^*. It calls **targeted attacks** if the target label is specified, **non-targeted attacks** otherwise. And we focus on targeted attacks only in this work. In our settings, attackers are allowed to create new fake nodes connecting to the target node v^* and design the features and neighbors of the fake nodes, in order to affect the classification of the target node.

To make sure the fake nodes are not too obvious and change the whole graph structure, we further limit the number of modifications by a budget Δ. Given above settings, we can formalize this as follows:

$$\begin{aligned} &\arg\max_{B,P} \; Z^{v^*,y^*} - max_{y\neq y^*} Z^{v^*,y} \\ &s.t. \sum_i \sum_j |B_{ij}| + \sum_i \sum_d |P_{id}| \leq \Delta \end{aligned} \tag{1}$$

where $Z = f_\Theta(A', X')$ and $Z^{v,y}$ is the v-th row and y-th column of Z.

We set $m = 1$ by default, C and P are initialized to zeros. The degree of the fake node is up to $k + 1$, which means we only add one fake node and perform the structural attack. It is noteworthy that the method can extend the methods to add more than one fake node and design features, but it is not the main point in this paper.

Attack as Reinforcement Learning

Given a target classifier f, a graph G and a target node v^* with a target label y^*, we define the attack procedure as a Markov Decision Process (MDP (Howard 1960)) $M(f, G, v^*, y^*)$:

- **State** The state s_t at time step t is a tuple (G'_t, v^*, y^*), where G'_t is a partially modified graph.
- **Action** The action a_t at time step t is to select a node and establish a link with the node we add. So the action set $|\mathcal{A}| = |V|$.
- **Reward** Our goal is to change the classification result of the target node v^*. So the reward is defined as:

$$R(G', v^*) = \begin{cases} 1 & f(G', v^*) \neq y^* \\ -1 & f(G', v^*) = y^* \end{cases} \tag{2}$$

Note that non-zero reward is received when the process reaches the terminal state. If the classification probabilities Z over labels is available, the reward can be defined as $R(G'_t, v^*) = Z_t^{v^*,y^*} - Z_{t-1}^{v^*,y^*}$ as well, where $Z^{v,y}$ denotes the v-th row and y-th column of Z.

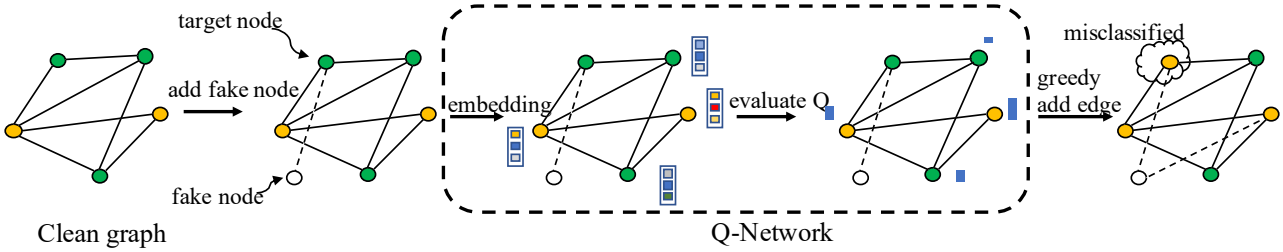

Figure 1: Illustration of the framework of g-Faker by adding a fake node to influence the classification of the target. Different colors of the nodes represent the predicted labels of the nodes. For a clean graph, a fake node is first added and create a link with the target node. At each time step, the Q-values of each actions is evaluated by a Q-network and then we greedily select the action with highest Q-value to get the next graph structure.

- **Terminal** Once the k edges have been modified or the label of target has been changed successfully, the process terminates.

In this work, we adopt Q-learning to solve the MDP, since the action space is discrete and varies at different steps to avoid duplicate actions. We will introduce the setting of our Q-learning below.

Deep Q-Network (DQN) uses a deep neural network as a Q-function approximator with parameter θ. It iteratively improves the estimate $Q(s_t, a_t|\theta)$ of expected greedy return which is achieved after taking action a_t at state s_t. The following mean squared loss is minimized in DQN:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}[(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'|\theta^-) - Q(s, a|\theta))^2] \tag{3}$$

where parameters θ^- are a stale copy of θ that helps prevent oscillations or divergence of Q.

In our settings, at each time step t is defined as $s_t = (G'_t, v^*, y^*)$, which means we expect that function Q can take the current graph structure, target node, and target label into account. Intuitively, Q should summarize the state of each node and figure out a value for candidate nodes. In order to represent the state of nodes, we leverage a deep learning architecture over graph, named *structure2vec* (Dai, Dai, and Song 2016) to obtain the latent representation of each node, which updates the embedding of node v at each iteration as:

$$\mu_v^{(k)} = relu(W_Q^1 x(v) + W_Q^2 \sum_{v_j \in \mathcal{N}(v)} \mu_{v_j}^{(k-1)}) \tag{4}$$

where $x(v)$ is the feature of node $v, \mathcal{N}(v)$ denotes the neighbors of node v.

Once the embeddings for each node is computed, we can figure out the value for each candidate node. Specifically, set $s_t = (G'_t, v^*, y^*)$, we use the embeddings $\mu_{v^*}^{(T)}$ for the target node v^* and the embeddings $\mu_a^{(T)}$ for each action a, respectively, after T iteration, since each action denotes a candidate node. Then the Q function can be formalized as:

$$Q(s_t, a_t|\theta) = MLP([\mu_{a_t}^{(T)}, \mu_{v^*}^{(T)}, y^*]) \tag{5}$$

where y^* is the one-hot label vector. The parameters θ of Q function includes $\{W_Q^1, W_Q^2\}$ and parameters in multilayer perceptron(MLP). Further, the deterministic policy $\pi(s_t|\theta) = \arg\max_{a \in \mathcal{A}} Q(s_t, a_t|\theta)$ is employed. We denote this method as *g-Faker*. Figure. 1 illustrates the process and the pseudocode is shown in supplement Algorithm.1.

Other methods

The RL based attack algorithm is suitable for black-box attack, where parameters of the targeted model are unknown. However, in other settings, e.g. white-box attack, in which attacker have the complete knowledge of the targeted model, other algorithms are preferred. Here we are going to introduce *Random Attack* which requires the least information and Gradient based Attack *GradAtt* in the white-box setting.

Random Attack The random attack is the simplest method that randomly chooses a node and establishes a link with the fake node at each step. It may be hard to change the classification result by randomly choosing nodes from all nodes. One simpler way is to select the nodes with target label, assuming the nodes with target label have a stronger positive influence than others. Though it requires the least information, it performs well.

GradAtt In white-box settings, the gradient-based attack has achieved great performance in continuous data. However, the methods cannot be directly applied in our cases due to the discreteness of data. To cope with the issue of the discreteness, we use a greedy algorithm to perform the attack. The loss function is defined as:

$$\mathcal{L}(A', X') = L(f(v^*; G'), y^*) \tag{6}$$

Where f and L are the given classifier and its loss function, respectively. At each step, we calculate the gradient information $\nabla_B \mathcal{L}(A', X')$ with respect to the edges of the fake nodes and choose the maximum element as the new edge to be added. In other words, we select the edge that most likely to affect the classification result at each step. Note that the fake node will firstly add an edge with the target node, otherwise there will be no gradients on the fake node. The process is illustrated in supplement Figure. 1 and the pseudocode is shown in supplement Algorithm. 2.

Table 1: The attack success rate of different methods with maximum step $k = 10$. The higher, the better.

	Polblogs	Cora	Citeseer
#Test target	115	708	535
Random-A	9.20%	0.51%	1.98%
Random-T	13.71%	7.34%	7.51%
GradAtt	40%	30.08%	36.45%
g-Faker	41.74%	32.77%	30.65%

Experiment

We use three well-known real-world datasets to evaluate our methods, namely Polblogs (Adamic and Glance 2005), Cora and Citeseer (Yang, Cohen, and Salakhutdinov 2016). The statistics of datasets are shown in supplement Table. 1.

In our experiments, we set the number of fake nodes $m = 1$, the features P and edges C remain zeros. And the maximum steps $k = \{1, ..., 10\}$. Note that the edge connected to the target node is not counted. For *g-Faker*, the probability change reward function is employed and iterations T of structure2vec is set to 2. For *Random Attack*, we test it in two settings: one randomly samples from all the nodes, denoted by *Random-A*, and the other one selects from the nodes with target label denoted by *Random-T*. The result is averaged over 5 individual runs.

Targeted attack

In this experiment, we select the nodes classified correctly in the validation and test set as our attack target nodes. For each target node, we take the other labels in the dataset as the target label. So the total number of attack targets are $|\mathcal{Y}| - 1$ times of the number of target nodes. The training set and testing set are divided in the ratio of 9: 1, and all methods are tested on the same test set.

Here, we do not run the brute force algorithm to get the upper bound of the practice, since the computational complexity is $|V|^k$. The attack success rate is reported in Table 1 when the maximum step $k = 10$. As is expected, Random-T leads to better performance compared to Random-A. It can be seen that for Polblogs and Cora, g-Faker achieves the highest performance with 41.74% and 32.77% success rate, respectively, even higher than white-box attack GradAtt. While for Citeseer, g-Faker is 5.8% lower than GradAtt. See supplement for further detailed experiments on target nodes with different degree and adding different number of edges (Section A in supplement material).

Further, we investigate the influence of the number of fake nodes with a fixed budget, which means you can insert one fake nodes with k neighbors or multiple nodes with less edges. The process can be complicated, since with more nodes, the influence of each node will be reduced, while it will also reduce the impact of original neighbors of the target node. We test $m = 1, 2, 3, 4$, $k = 10$ and the maximum number of perturbation become $k-m+1$, respectively, since each fake node will connect with the target node. We use the classification margin $S = Z^{v^*,y^*} - max_{y \neq y^*} Z^{v^*,y}$ to evaluate the effectiveness of attacks. Figure. 2 report the result using *GradAtt*. Each dot in Figure. 2 represents one attack target. We can see, from $m = 1$ to $m = 3$, the attack success rate is increasing while it fails when $m = 4$. As discussed before, when $m = 4$, each node will have only an average of 2 additional edges which makes it hard to affect the target node. Intuitively, it's better to add as many as possible fake nodes while ensuring the influence of the fake node, e.g. 3 to 5 neighbors in Cora according to supplement Figure. 2.

Table 2: Attack success rate of two attack types on Cora for GCN and GraphSage

	GCN		GraphSage	
	Evasion	Poison	Evasion	Poison
Random	7.34%	1.13%	1.97%	2.15%
GradAtt	30.08%	22.26%	7.63%	12.54%
g-Faker	32.77%	29.92%	13.28%	22.29%

Transferability of attacks

After exploring the effects of our attacks on fixed GCN model, now we examine its transferability to other deep learning models. Here we use another graph convolutional method named GraphSage (Yang, Cohen, and Salakhutdinov 2016), which generates embeddings by sampling and aggregating features from neighborhoods. Different from spectral-based GCN, GraphSage is a spatial-based algorithm. We use the default settings for the hyper-parameters in the code provided by the paper.

We evaluate the transferability of our attacks in two attack types: evasion and poison attack. For evasion attack, the parameters of the model are kept fixed, while the model is retrained on the modified data for poison attack, which is a more challenging task theoretically. The result of poison attack is averaged over 5 runs in our experiments.

Figure. 3 shows the results of both evasion and poison attack of GCN and GraphSage, respectively. Table. 2 reports the corresponding attack success rate. It is worth noting that the attack target is generated according to the result of GCN, so there are some nodes incorrectly classified in GraphSage. As we can see, although the adversarial examples are generated based on GCN, the evasion attack still works on GraphSage. For GCN, the performance of attacks drops slightly for poisoning compared to evasion. Yet, for GraphSage, the poisoning outperforms the evasion. Furthermore, g-Faker outperforms GradAtt and suffers fewer drops in attack success rate for poisoning for GCN. We argue that this comes from that the adversarial examples of GradAtt are generated based on the gradients of the fixed model, so some actions may only work for this model and its transferability is relatively weak.

Analysis of perturbations

Actions. Figure. 4 plots the top-10 selected actions(nodes) for each target label and its distribution for Cora dataset. The x-axis is the classification probabilities of the node in the clean graph and the y-axis denotes its average reward gain when the node is selected as the first action. The size of the dot represents its frequency, the larger,

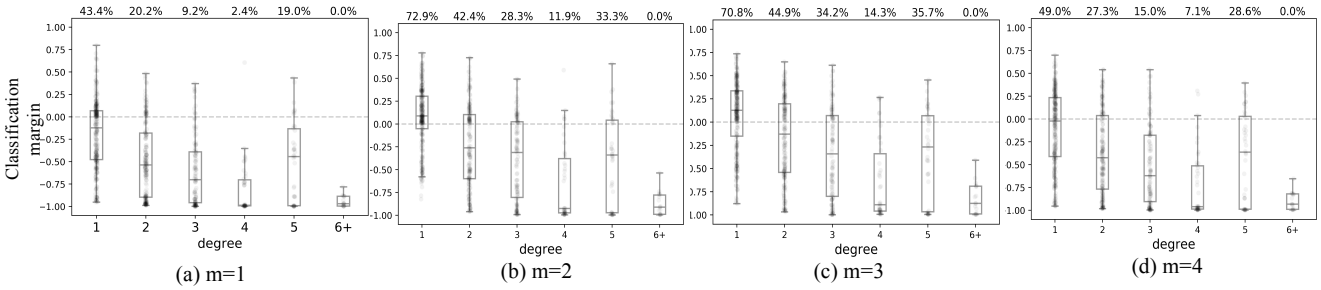

Figure 2: Margin distribution for different number of fake nodes with a fixed budget using GradAtt (the higher, the better).

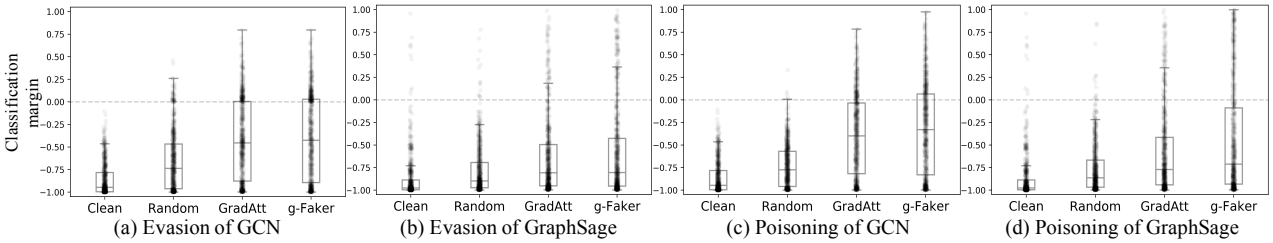

Figure 3: Results of two attack types on Cora using different algorithms.

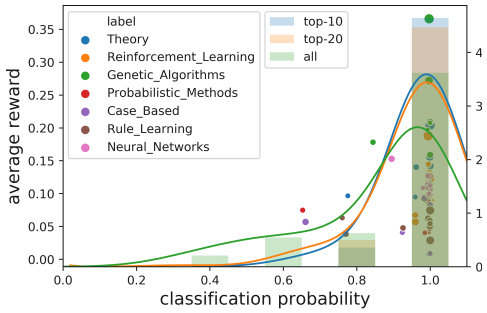

Figure 4: The top-10 actions for each target label and its average reward gain as the first action. Each dot represents a action and the size of it denotes its frequency, the larger, the more frequent.

the more frequent. Obviously, in line with our intuition, the nodes with higher classification certainty are preferred, since they contain more features about the target label. Moreover, the actions for target label 'Genetic_Algorithms' get a relatively higher average reward, while the target label 'Rule_Learning' and 'Probabilistic_Methods' receive a lower value. This indicates that it is easier to misguide the nodes to the class Genetic Algorithms. Intuitively, there exists some typical words/features for the specific class, e.g. mutation and crossover for Genetic Algorithms.

Degree distribution. We limit the number of the fake nodes and the degrees of the fake nodes in our experiment. However, it is still hard to say that the changes are *unnoticeable*. Here we use the test statistic proposed by (Zügner, Akbarnejad, and Günnemann 2018), $\Lambda(G, G')$, which estimates whether the degree distribution of two graph is stemmed from the same pow-law distribution. If $\Lambda(G, G') < \tau \approx 0.004$, the perturbations are regarded as *unnoticeable*.

Figure. 5 shows the distribution test statistic $\Lambda(G, G')$ of all adversarial examples. As seen, only a partial example, less than 10%, is over 0.004, and most of them are close to 0.001. Therefore, the perturbations generated by adding fake nodes are still accepted.

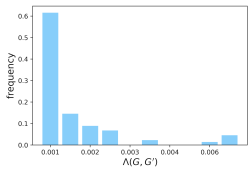

Figure 5: $\Lambda(G, G')$ distribution of all adversarial examples for Cora.

Conclusion

For the task of node classification, we demonstrate that the graph convolutional networks are vulnerable to adversarial attacks by adding fake nodes for a targeted attack, instead of changing existing edges and features. Nodes with significant effect towards the target label can be effectively selected by our methods. It is possible to successfully attack the GCN model with few perturbations. Furthermore, based on our extensive experiment, our attacks still work even for challenging poisoning attack. The attack policies learned on one deep learning model can also be generalized to others.

Acknowledgment

This work is supported by the National Key Research and Development Program of China under grant No.2017YFC1001703 and National Natural Science Foundation of China under grant No. 61802340.

References

Adamic, L. A., and Glance, N. 2005. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, 36–43. ACM.

Bhagat, S.; Cormode, G.; and Muthukrishnan, S. 2011. Node classification in social networks. In *Social network data analytics*. Springer. 115–148.

Bojchevski, A., and Günnemann, S. 2019. Adversarial attacks on node embeddings via graph poisoning. In *International Conference on Machine Learning*, 695–704.

Dai, H.; Li, H.; Tian, T.; Huang, X.; Wang, L.; Zhu, J.; and Song, L. 2018. Adversarial attack on graph structured data. *arXiv preprint arXiv:1806.02371*.

Dai, H.; Dai, B.; and Song, L. 2016. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, 2702–2711.

Faliszewski, P.; Gonen, R.; Koutecký, M.; and Talmon, N. 2018. Opinion diffusion and campaigning on society graphs. In *IJCAI*, 219–225.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. Explaining and harnessing adversarial examples (2014). *arXiv preprint arXiv:1412.6572*.

Howard, R. A. 1960. Dynamic programming and markov processes.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

LE, D. T.; Lauw, H. W.; and Fang, Y. 2017. Basket-sensitive personalized item recommendation. IJCAI.

LI, Z., and Yu, Y. 2016. Protein secondary structure prediction using cascaded convolutional and recurrent neural networks. IJCAI.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710. ACM.

Yang, Z.; Cohen, W. W.; and Salakhutdinov, R. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*.

Zhao, Z.; Yang, Q.; Lu, H.; Weninger, T.; Cai, D.; He, X.; and Zhuang, Y. 2018. Social-aware movie recommendation via multimodal network learning. *IEEE Transactions on Multimedia* 20(2):430–440.

Zügner, D.; Akbarnejad, A.; and 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*, 2847–2856. ACM.

Zügner, D., and Günnemann, S. 2019. Adversarial attacks on graph neural networks via meta learning. *arXiv preprint arXiv:1902.08412*.