Adversarial Attacks on Graph Neural Networks

[1] **GRAPH ATTENTION NETWORKS,** ICLR (2018)

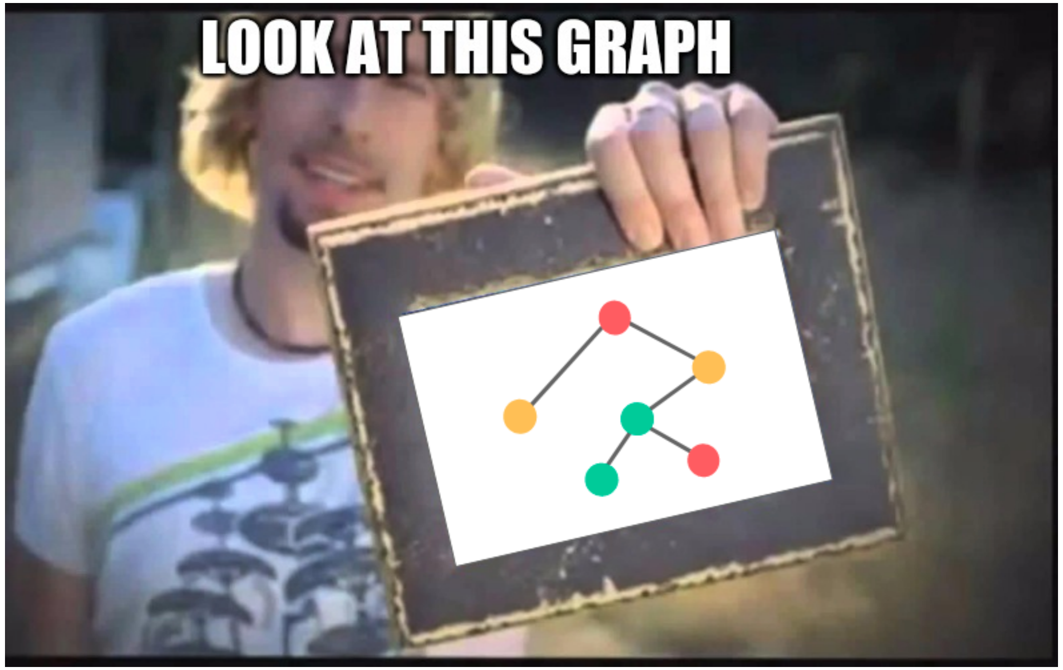
[2] **SADVERSARIAL ATTACKS ON GRAPH NEURAL NETWORKS VIA META LEARNING,** ICLR (2019)

Presenter: Guihong Wan, Oct/2020

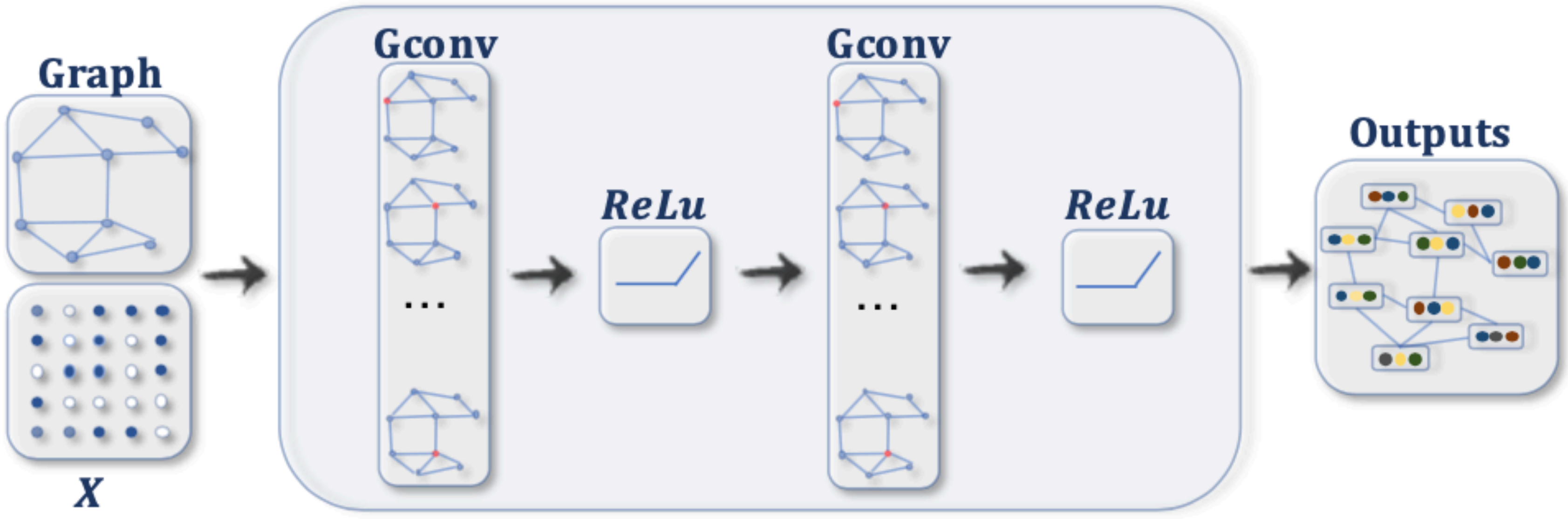
Outline

- **Part1: Graph networks review**
  - GCN
  - GAT
- **Part2: Adversarial attack on graph networks**
  - Background
  - Adversarial attack vis meta learning

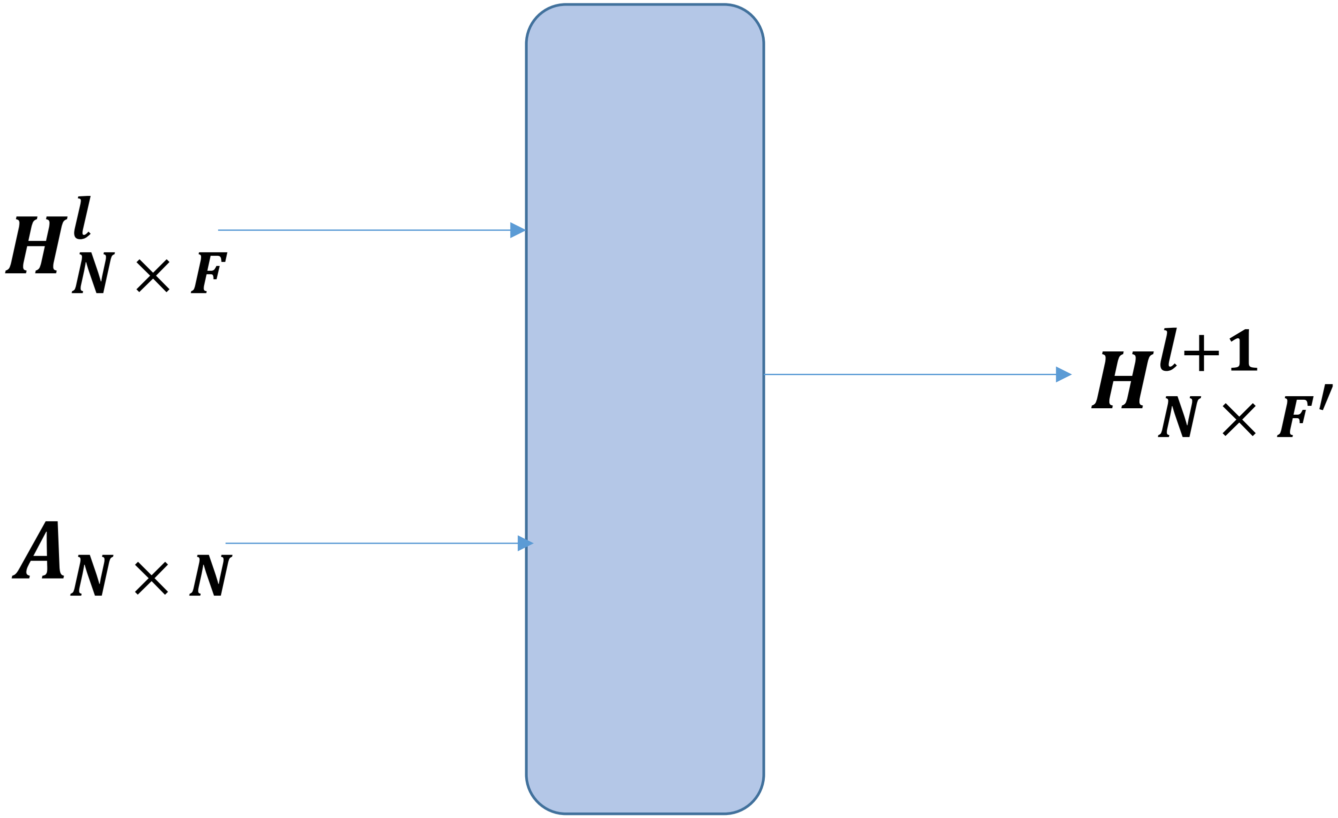
Review



Graph Neural Networks



A Gconv Layer



GCN

The l^{th} GCN Layer: $$H^{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{l}W^{l})$$

$H^0 = X$, otherwise the output of previous layer.

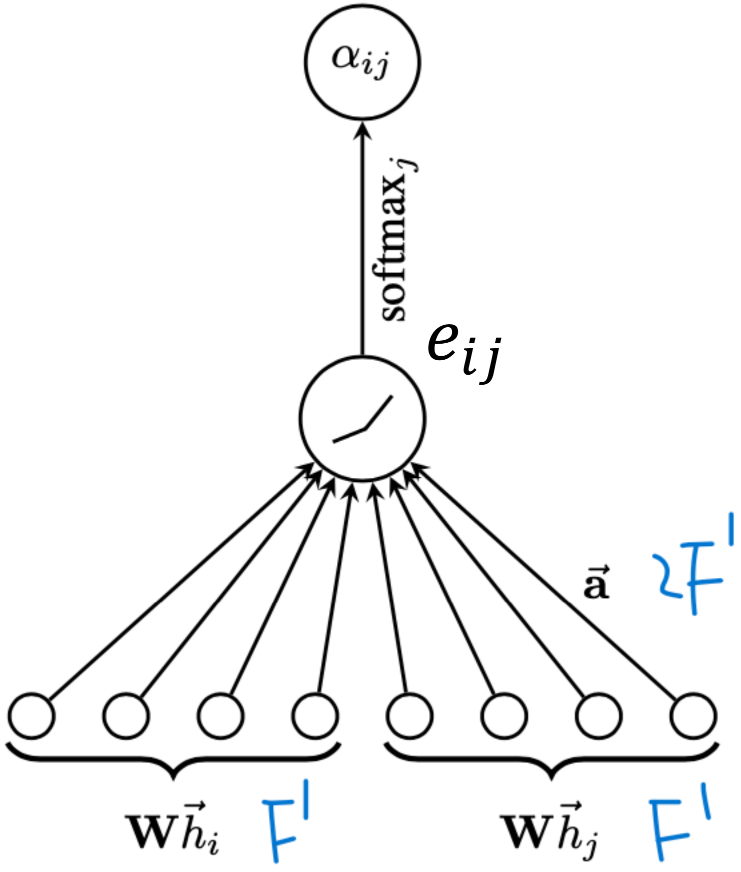
$\tilde{A} = A + I_N, \qquad \tilde{D} = \sum_j \tilde{A}_{ij}$

W: what to learn.

Main idea: learn a node v's representation
by aggregating its own feature x_v and its neighbors' feature x_u,
for all $u \in N(v)$.

Normalization: the multiplication will completely change the scale of the features.

Graph Attention Networks (GAT)



h_i : i^{th} data vector

Wh_i : linear transformation

$j \in N_i$: a neighbor node of node i

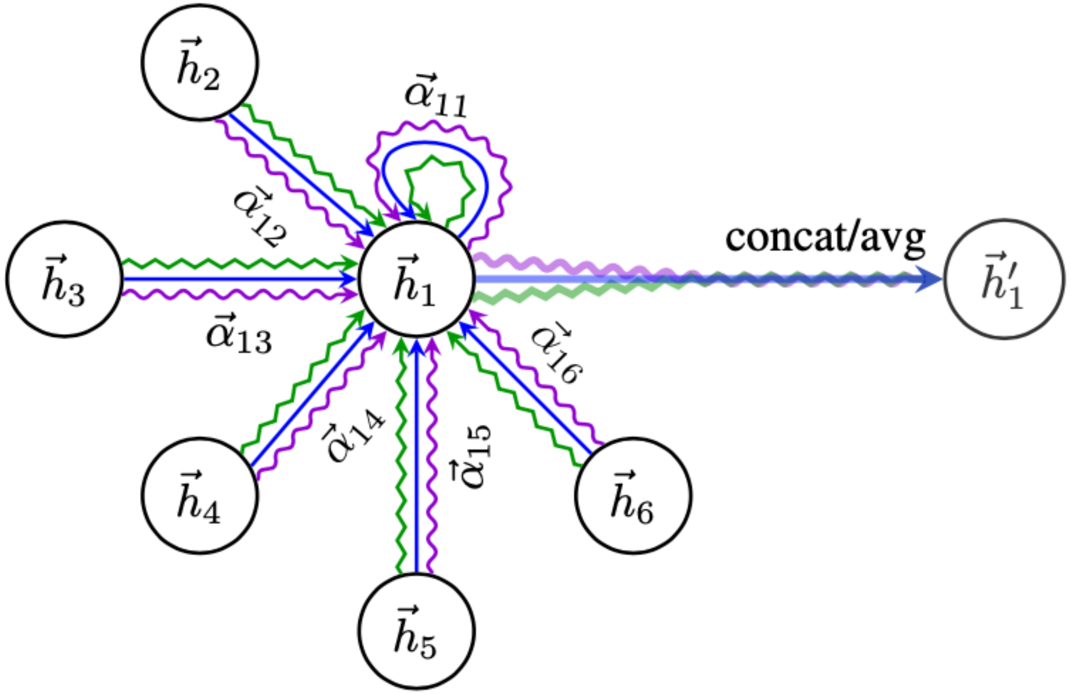
e_{ij} : a shared attentional mechanism

$$\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)$$

Softmax over all $j \in N_i$:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

GAT (cont.)



F'x1

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \underline{\mathbf{W}\vec{h}_j}\right)$$

$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^{k} \mathbf{W}^{k} \vec{h}_j\right)$$

K.F'x1

GAT (cont.)

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \underline{\mathbf{W}\vec{h}_j}\right)$$

In matrix form:

$$H_{N\times F} = [h_1, h_2, \dots, h_N]^T \longrightarrow H'_{N\times F'} = [h'_1, h'_2, \dots, h'_N]^T$$

1. $HW^T_{F\times F'} = Z_{N\times F'}$ ← linear transformation

2. $\alpha Z = \begin{bmatrix} \alpha_{11} & \cdots & \alpha_{1N} \\ \vdots & \ddots & \vdots \\ \alpha_{N1} & \cdots & \alpha_{NN} \end{bmatrix} \begin{bmatrix} z_{11} & \cdots & z_{1F'} \\ \vdots & \ddots & \vdots \\ z_{N1} & \cdots & z_{NF'} \end{bmatrix}$ ← Each feature is averaged over **all nodes**

3. $\tilde{A}_{N\times N} = A + I_N$

$$\boxed{\textbf{GAT: } H' = \tilde{A} \odot \alpha\, HW^T}$$

GAT VS GCN

GAT: $H' = \widetilde{A} \odot \alpha\, HW^T$

GCN: $H' = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}HW^T$

Adversarial Attack

1. Background
2. Key idea of the paper
3. Experimental results

ADVERSARIAL ATTACKS ON GRAPH NEURAL NETWORKS VIA META LEARNING, ICLR (2019)

Background

Goal : to investigate the ***robustness*** of graph neural networks.
Result: Small graph perturbations lead to a strong decrease in performance for graph neural networks.
Idea : to use meta-learning for the opposite: modifying the training data to ***worsen*** the performance for testing data.

Background: Supervised node classification

Task : semi-supervised node classification

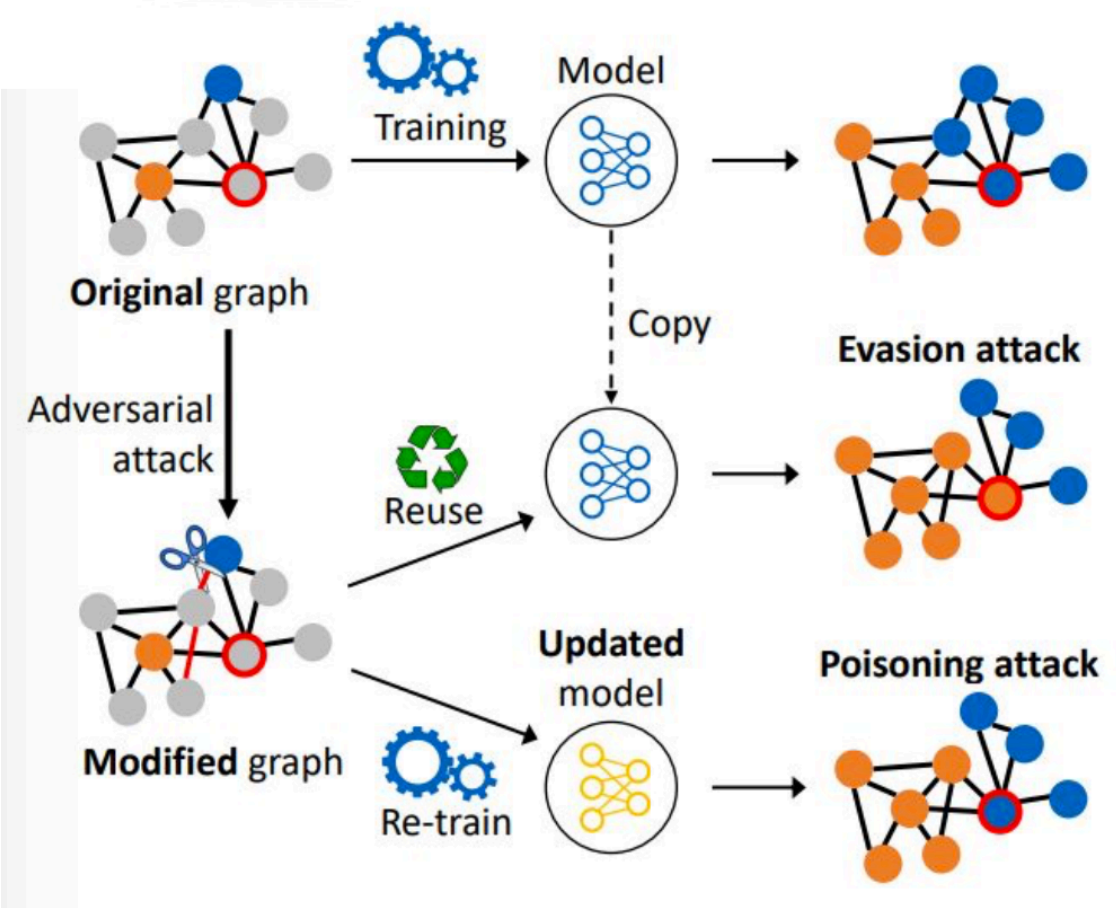
Given: the set of labeled nodes: V_L of graph $G(A, X)$

Goal : to learn a function f_θ, which maps each node to exactly one class. The parameters θ are learned by minimizing a loss function L_{train}:

$$\theta^* = \arg\min_{\theta} \mathcal{L}_{train}\left(f_\theta\left(G\right)\right)$$

$f_\theta(\mathrm{G})$: matrix of class probabilities: $[0,1]^{N \times C}$
L_{train}: loss only for labeled nodes

Background: Adversarial attacks



Evasion attack: at test time

Poisoning attack: at training time.

Problem Setup

Given a limited budget of perturbations Δ (e.g. the number of edges can be changed),

the goal is to insert/delete edges so that training on the perturbated graph leads to weaker classification performance.

This corresponds to solving the bilevel problem:

$$\min_{\hat{G} \in \Phi(G)} \quad \mathcal{L}_{\text{atk}}(f_{\theta^*}(\hat{G})) \quad s.t. \quad \theta^* = \arg\min_{\theta} \quad \mathcal{L}_{\text{train}}(f_{\theta}(\hat{G}))$$

Problem Setup (cont.)

$$\min_{\hat{G} \in \Phi(G)} \quad \mathcal{L}_{\text{atk}}(f_{\theta^*}(\hat{G})) \quad s.t. \quad \theta^* = \arg\min_{\theta} \quad \mathcal{L}_{\text{train}}(f_{\theta}(\hat{G}))$$

L_{train}: the training network tries to increase the classification accuracy on unlabeled nodes.

L_{atk} : loss the attacker aims to optimize. The attacker wants to decrease the accuracy on the unlabeled nodes.

$$\max_{\hat{G} \in \Phi(G)} \mathcal{L}_{test}\left(f_{\theta^*}(\hat{G})\right) \quad s.t.\ \theta^* = \arg\min_{\theta} \mathcal{L}_{train}(f_{\theta}(\hat{G}))$$

where , L_{atk} = - L_{test}

Problem Setup (cont.)

$$\min_{\hat{G}\in\Phi(G)} \quad \mathcal{L}_{\text{atk}}(f_{\theta^*}(\hat{G})) \quad s.t. \quad \theta^* = \arg\min_{\theta} \quad \mathcal{L}_{\text{train}}(f_{\theta}(\hat{G}))$$

$$\max_{\hat{G}\in\Phi(G)} \mathcal{L}_{test}\left(f_{\theta^*}(\hat{G})\right) \quad s.t.\ \theta^* = \arg\min_{\theta} \mathcal{L}_{train}(f_{\theta}(\hat{G}))$$

where , L_{atk} = - L_{test}

However, L_{test} is unknown. Two options:

1. L_{atk} = - L_{train}
2. L_{atk} = - L_{self}

Challenges

1. The number of possible edge perturbations is in $O(N^{2\Delta})$, ignoring symmetry. (modify Δ out of N^2)

2. The data (i.e. graph structure) is discrete, which means that gradient-based optimization is not directly applicable.

Idea

1. Treat the graph structure matrix as a hyperparameter
2. Relax the discreteness of the structure in order to obtain meta-gradients but perform discrete perturbations.

Modified graph

$$\boxed{\nabla_G^{\text{meta}} := \nabla_G \mathcal{L}_{\text{atk}}(f_{\theta^*}(G)) \quad s.t. \quad \theta^* = \text{opt}_\theta(\mathcal{L}_{\text{train}}(f_\theta(G)))}$$

Meta gradients

Meta-gradients (e.g., gradients w.r.t hyperparameters) are obtained by back propagating through the learning phase of a differentiable model.

$$\nabla_G^{\text{meta}} := \nabla_G \mathcal{L}_{\text{atk}}(f_{\theta^*}(G)) \quad s.t. \quad \theta^* = \text{opt}_\theta(\mathcal{L}_{\text{train}}(f_\theta(G)))$$

Meta-gradients indicate how the attacker loss after training: $\boldsymbol{L_{atk}}$ will change for a small perturbations on the training data.

$$\nabla_G^{\text{meta}} := \nabla_G \mathcal{L}_{\text{atk}}(f_{\theta^*}(G)) \quad s.t. \quad \theta^* = \text{opt}_\theta(\mathcal{L}_{\text{train}}(f_\theta(G)))$$

Meta gradients

For example, start from some initial θ_0

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} L_{train}(f_{\theta_t}(G))$$

The attacker’s loss after traning for T steps:

$$L_{atk}(f_{\theta_T}(G))$$

Then: $\nabla_G^{meta} = \nabla L_{atk}(f_{\theta_T}(G))$

$G, \theta_t \rightarrow f_{\theta_t}(G), L_{train} \rightarrow \theta_T = \theta_t - \alpha \nabla_{\theta_t} L_{train} \rightarrow L_{atk}$

This integrates the idea of meta learning.

Meta gradients

$$\nabla_G^{\text{meta}} := \nabla_G \mathcal{L}_{\text{atk}}(f_{\theta^*}(G)) \quad s.t. \quad \theta^* = \text{opt}_\theta(\mathcal{L}_{\text{train}}(f_\theta(G)))$$

$G, \theta_t \rightarrow f_{\theta_t}(G), L_{train} \rightarrow \theta_T = \theta_t - \alpha \nabla_{\theta_t} L_{train} \rightarrow L_{atk}$

This integrates the idea of meta learning.

Note: it is expensive to calculate meta-gradients, but there are lots of approximate solutions, e.g. MAML.

C. Finn P. Abbeel and S. Levine, Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

Discrete perturbation

Score function:

$$S = \nabla_{\hat{A}}^{meta} \odot \left(-2\,\hat{A} + 1\right)$$

e(u,v): maximum entry (u, v) in S

Discrete perturbation (cont.)

$S = \nabla_{\hat{A}}^{meta} \odot \left(-2\,\hat{A} + 1\right)$: flip the assign of $\nabla_{\hat{A}}^{meta}$

$$S_{(u,v)} = \nabla_G^{meta}(-2a_{uv}+1)$$

$$= \begin{cases} \nabla_G^{meta} & \text{if } a_{uv}=0 \\ -\nabla_G^{meta} & \text{if } a_{uv}=1 \end{cases}$$

Choose $e'_{(u,v)} = \arg\max S_{(u,v)}$

set $a_{uv} = 0$, if $a_{uv}=1$

set $a_{uv} = 1$, if $a_{uv}=0$

Discrete perturbation (cont.)

Score function: $S = \nabla_{\hat{A}}^{meta} \odot \left(-2\,\hat{A} + 1\right)$

Adjacency matrix: A

Adjacency_changes

$$\hat{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 1 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

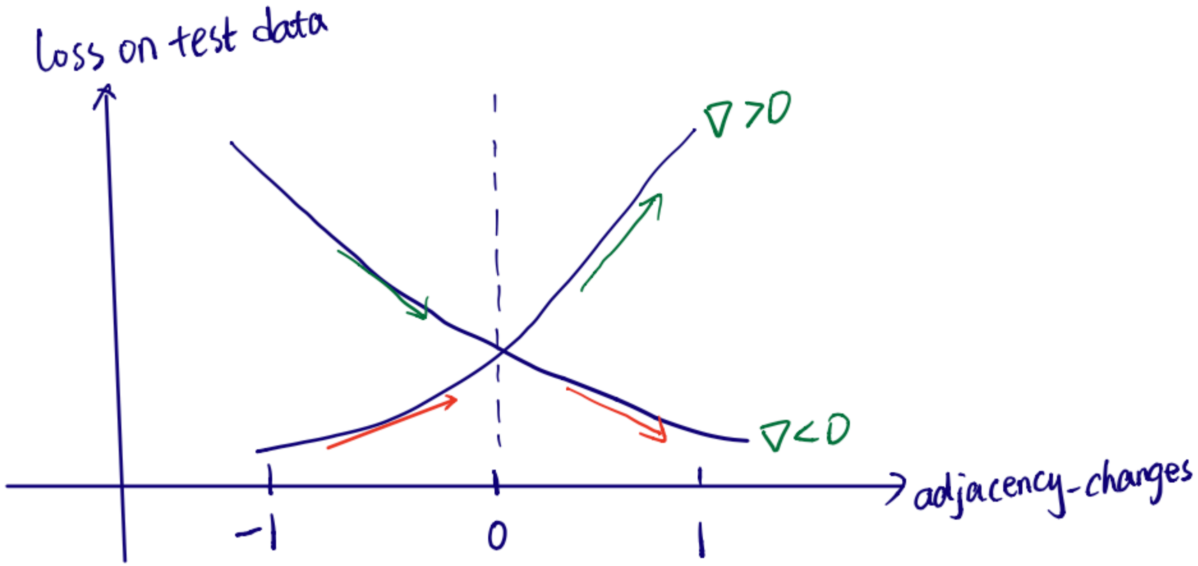
-1: deletion
1: addition

```
self.adjacency_meta_grad
  = tf.multiply( tf.gradients(test_loss, self.adjacency_changes)[0],
                 tf.reshape(self.modified_adjacency, [-1]) * -2 + 1,
                            name="Meta_gradient")
```

tf.gradients(test_loss, self.adjacency_changes)

Why flip the assign of $\nabla_{\hat{A}}^{meta}$?

The attacker wants to max L_{test}



① $-1/0 \xleftarrow[\nabla<0]{\text{deletion}} 0/1$

② $0/0 \xrightarrow{\text{addition}} 1/1$

③ $-1/0 \xrightarrow[\nabla>0]{\text{addition}} 0/1$

④ $0/0 \xleftarrow[\nabla<0]{\text{deletion}} 1/1$

Algorithm 1: Poisoning attack on graph neural networks with meta gradients and self-training

Input: Graph $G = (A, X)$, modification budget Δ, number of training iterations T, training class labels C_L

Output: Modified graph $\hat{G} = (\hat{A}, X)$

$\hat{\theta} \leftarrow$ train surrogate model on the input graph using known labels C_L;
$\hat{C}_U \leftarrow$ predict labels of unlabeled nodes using $\hat{\theta}$;
$\hat{A} \leftarrow A$;
while $\|\hat{A} - A\|_0 < 2\Delta$ **do**
  randomly initialize θ_0;
  $\theta_T\left\{\begin{array}{l}\textbf{for } t \textit{ in } 0 \ldots T-1 \textbf{ do} \\ \quad \theta_{t+1} \leftarrow \text{step}\left(\theta_t, \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{A}, X)); C_L\right); \quad \texttt{// update e.g. via gradient descent}\end{array}\right.$
  `// Compute meta gradient via backprop through the training procedure`
  $\nabla_{\hat{A}}^{\text{meta}} \leftarrow \nabla_{\hat{A}} \mathcal{L}_{\text{self}}(f_{\theta_T}(\hat{A}, X); \hat{C}_U)$;
  $S \leftarrow \nabla_{\hat{A}}^{\text{meta}} \odot (-2\hat{A} + 1)$; `// Flip gradient sign of node pairs with edge`
  $e' \leftarrow$ maximum entry (u, v) in S that fulfills constraints $\Phi(G)$;
  $\hat{A} \leftarrow$ insert or remove edge e' to/from $\hat{A}$; $\leftarrow$ modify the graph structure
$\hat{G} \leftarrow (\hat{A}, X)$;
return : $\hat{G}$

Experimental results

Table 2: Misclassification rate (in %) with 5% perturbed edges.

Attack	Cora GCN	Cora CLN	Cora DeepWalk	Citeseer GCN	Citeseer CLN	Citeseer DeepWalk	PolBlogs GCN	PolBlogs CLN	PolBlogs DeepWalk	Avg. rank
Clean	16.6 ± 0.3	17.3 ± 0.3	20.3 ± 1.0	28.5 ± 0.9	28.3 ± 0.9	34.8 ± 1.4	6.4 ± 0.6	7.6 ± 0.5	5.3 ± 0.5	7.4
DICE	18.0 ± 0.4	18.0 ± 0.2	22.8 ± 0.3	28.9 ± 0.3	29.1 ± 0.3	39.1 ± 0.4	11.2 ± 1.1	11.2 ± 0.8	10.2 ± 0.6	5.0
First-order	17.2 ± 0.3	17.6 ± 0.2	20.7 ± 0.2	28.3 ± 0.3	28.4 ± 0.3	34.0 ± 0.3	7.8 ± 0.9	7.6 ± 0.5	7.9 ± 0.6	7.1
Nettack*	-	-	-	31.9 ± 0.3	30.2 ± 0.4	$\mathbf{41.2 \pm 0.4}$	-	-	-	-
A-Meta-Train	21.8 ± 0.9	20.5 ± 0.3	25.0 ± 0.6	31.9 ± 0.7	30.1 ± 0.5	32.7 ± 0.5	11.9 ± 2.8	12.9 ± 2.5	5.8 ± 0.2	4.7
A-Meta-Both	20.7 ± 0.4	19.0 ± 0.3	$\mathbf{28.5 \pm 0.5}$	28.6 ± 0.4	28.7 ± 0.4	34.4 ± 0.4	19.8 ± 0.8	16.5 ± 1.3	21.5 ± 1.9	4.3
Meta-Train	22.0 ± 1.2	$\mathbf{21.7 \pm 0.4}$	26.1 ± 0.6	30.3 ± 1.0	29.0 ± 0.6	36.0 ± 0.2	16.3 ± 2.9	$\mathbf{18.7 \pm 2.3}$	14.5 ± 4.2	3.2
Meta-Self	$\mathbf{24.5 \pm 1.0}$	20.3 ± 0.4	28.1 ± 0.6	$\mathbf{34.6 \pm 0.7}$	$\mathbf{32.2 \pm 0.6}$	34.6 ± 0.7	$\mathbf{22.5 \pm 0.8}$	17.9 ± 1.7	$\mathbf{59.0 \pm 3.0}$	**2.3**
Meta w/ Oracle	21.0 ± 0.5	21.6 ± 0.3	27.8 ± 0.7	34.2 ± 0.9	32.9 ± 0.6	36.1 ± 0.7	25.6 ± 1.9	19.1 ± 1.4	52.3 ± 2.8	2.0

* Did not finish within three days on Cora-ML and PolBlogs

Result: Small graph perturbations lead to a strong decrease in performance for graph neural networks.

Experimental results

Table 5: Share (in %) of edge deletions (DEL) and insertions (INS) by Meta-Self on CORA-ML.

	$c_i = c_j$	$c_i \neq c_j$
DEL	15.3	3.9
INS	9.4	71.4

Thank you!