Adversarial Attacks on Graph Neural Networks via Meta Learning

Hoki Kim

Seoul National University

Adversarial Attacks on Graph Neural Networks via Meta Learning

- Daniel Zügner, Stephan Günnemann
- Technical University of Munich, Germany

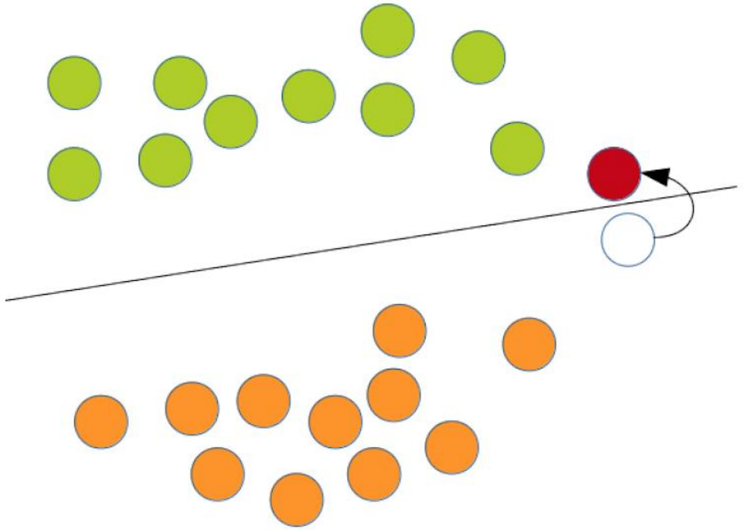
- ICLR 2019
- Keywords
  - Adversarial Attacks
  - Graph Neural Networks
  - Meta Learning

Adversarial Attacks on Graph Neural Networks via Meta Learning

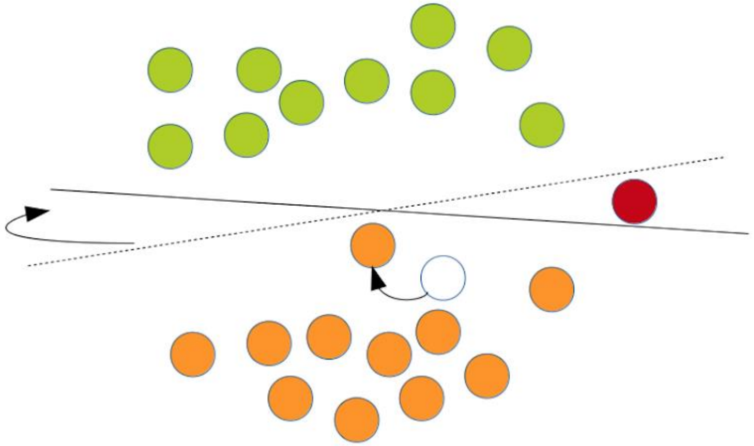
- **Adversarial Attacks**
  - Degrade performance of a machine learning model

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Adversarial Attacks**
  - Training time attack (poisoning) : Modifying few training examples to worsen the performance



Classical adversarial attack:
directly modifying the testing sample



Data poisoning:
modifying training samples intelligently

https://towardsdatascience.com/how-to-attack-machine-learning-evasion-poisoning-inference-trojans-backdoors-a7cb5832595c

Adversarial Attacks on Graph Neural Networks via Meta Learning

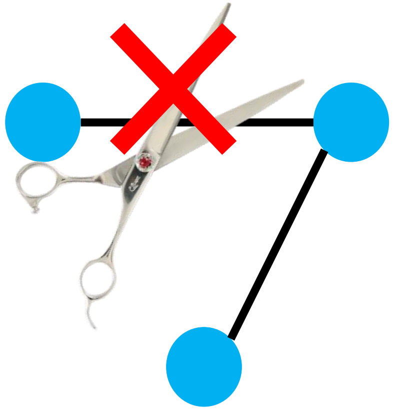
- **Problem Formulation**
  - **Task :** Semi-supervised node classification
  - **Goal :** Generate modified graph $\hat{G} = (\hat{A}, X)$ from the original graph $G = (A, X)$ to increase the misclassification rate of GNN trained with $\hat{G}$.
  - **Constraints :** adversarial attacks should be unnoticeable.

Limit the number of changes on edges

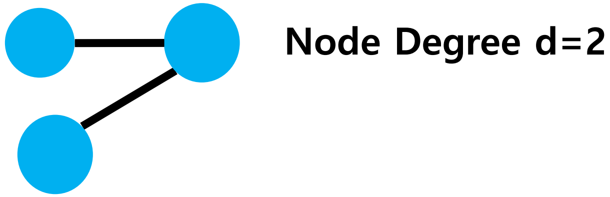
$$\left\|A - \hat{A}\right\|_0 \leq \Delta$$

$$A^T = A = \{0, 1\}^{N \times N}$$

Node becomes disconnected (i.e. a singleton) during the attack



Unnoticeability constraint on the degree distribution



Check $\hat{G}, G$ are from same distribution by using *likelihood ratio test*

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Problem Formulation**
  - **Task :** Semi-supervised node classification
  - **Goal :** Generate modified graph $\hat{G} = (\hat{A}, X)$ from the original graph $G = (A, X)$ to increase the misclassification rate of **GNN(f_θ)** trained with $\hat{G}$.
  - **Constraints $\boldsymbol{\Phi}(\boldsymbol{G})$**

$$\min_{\hat{G} \in \Phi(G)} L_{atk}\left(f_{\theta^*}(\hat{G})\right) = -L_{train}\left(f_{\theta^*}(\hat{G})\right)$$

$$s.t. \qquad \theta^* = \arg\min_{\theta} L_{train}\left(f_\theta(\hat{G})\right)$$

$(*)$ $\boldsymbol{L_{train}}$: loss function (e.g. cross-entropy)

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Meta Learning**
  - Train a model on a variety of learning tasks, such that it can solve new learning tasks using only a small number of training samples
  - Given task distribution $p(T)$ and it's corresponding loss function L_{T_i},

$$\min_{\theta} \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta^*}) \qquad s.t. \qquad \theta^* = \arg\min_{\theta} L_{T_i}(f_{\theta})$$

Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks."

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Meta Learning**
  - Given task distribution $p(T)$ and it's corresponding loss function L_{T_i},

$$\min_{\theta} \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta^*}) \quad s.t. \quad \theta^* = \arg\min_{\theta} L_{T_i}(f_{\theta})$$

Very similar to each other

$$\min_{\hat{G} \in \Phi(G)} L_{atk}\left(f_{\theta^*}(\hat{G})\right) \quad s.t. \quad \theta^* = \arg\min_{\theta} L_{train}\left(f_{\theta}(\hat{G})\right)$$

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Adversarial Attack + Meta Learning**
  - Find an optimal perturbation for dropping the accuracy of the final model.

(1) Train Model

$$\theta_T = \theta_{T-1} - \alpha \nabla_{\theta_{T-1}} L_{train}\left(f_{\theta_{T-1}}(G)\right)$$

(2) Get Meta Gradient

$$\nabla_G^{meta} = \nabla_G L_{atk}\left(f_{\theta_T}(G)\right) = \nabla_f L_{atk}\left(f_{\theta_T}(G)\right) \cdot \left[\nabla_G f_{\theta_T}(G) + \nabla_{\theta_T} f_{\theta_T}(G) \cdot \boxed{\nabla_G \theta_T}\right]$$

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Adversarial Attack + Meta Learning (Approx.) + Self-Training**
  - For reducing computational complexity,

$$\nabla_G^{meta} = \nabla_G L_{atk}\left(f_{\theta_T}(G)\right) = \nabla_f L_{atk}\left(f_{\theta_T}(G)\right) \cdot \left[\nabla_G f_{\theta_T}(G) + \nabla_{\theta_T} f_{\theta_T}(G) \cdot \cancel{\nabla_G \theta_T}\right]$$

$$\approx \nabla_f L_{atk}\left(f_{\tilde{\theta}_T}(G)\right) \cdot \nabla_G f_{\tilde{\theta}_T}(G)$$

$\tilde{\theta}_T$ *is independent of the data* G *and* $\tilde{\theta}_{T-1}$

Nichol, Alex, Joshua Achiam, and John Schulman. "On first-order meta-learning algorithms."

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Adversarial Attack + Meta Learning (Approx.) + Self-Training**
  - To change edges based on the gradient

(1) Flip the sign for **connected node** pairs as this yields the gradient for a change in the negative direction.

$$S(u,v) = \nabla_{a_{uv}}^{meta}(-2 \cdot a_{uv} + 1)$$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \nabla_{a_{uv}}^{meta} = \begin{bmatrix} 0 & -0.3 & 0.2 \\ -0.3 & 0 & 0.1 \\ 0.2 & 0.1 & 0 \end{bmatrix} \qquad S(u,v) = \begin{bmatrix} 0 & 0.3 & 0.2 \\ 0.3 & 0 & -0.1 \\ 0.2 & -0.1 & 0 \end{bmatrix}$$

(2) Select most effective one edge from S and change it.

$$e' = \underset{e=(u,v)\,:\,M(A,e)\in\Phi(G)}{\arg\max} S(u,v)$$

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Experiments**
  - Datesets
    - CORA-ML : scientific publications
    - CITESEER : scientific publications
    - POLBLOGS : weblogs on US politics
  - Datasets Split
    - labeled (10%) / unlabeled (90%) nodes
  - Networks
    - Graph Convolutional Networks (GCN) : 2-Layer + ReLU
    - Column Networks (CLN)
  - Details
    - Repeat all of our attacks on five different splits.
    - train all target classifiers ten times per attack.
    - the uncertainty indicates 95 % confidence intervals.

Table 6: Dataset statistics.

Dataset	N_{LCC}	E_{LCC}	D	K
CORA-ML	2,810	7,981	2,879	7
CITESEER	2,110	3,757	3,703	6
POLBLOGS	1,222	16,714	-	2
	Number of Nodes	Number of Edges	Dimension of Features	Number of classes

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Results**
  - Meta-Train (Meta + λ=1)
  - Meta-Self (Meta + λ=0)
  - A-Meta-Train (Approx. Meta + λ=1)
  - A-Meta-Both (Approx. Meta + λ=0.5)
  - A-Meta-Self (Approx. Meta + λ=0)

$$\nabla_G^{meta} = \sum_{t=1}^{T} \lambda \nabla_G L_{train}\left(f_{\widetilde{\theta}_t}(G)\right) + (1-\lambda)\nabla_G L_{self}\left(f_{\widetilde{\theta}_t}(G)\right)$$

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Results**
  - Change in Error rate

Table 1: Misclassification rate (in %) for different meta-gradient heuristics with 5% perturbed edges.

	CORA-ML		CITESEER	
	GCN	CLN	GCN	CLN
Clean	16.6 ± 0.3	17.3 ± 0.3	28.5 ± 1.0	28.3 ± 0.8
A-Meta-Train	21.2 ± 0.9	20.3 ± 0.3	31.8 ± 0.8	29.8 ± 0.5
A-Meta-Self	21.8 ± 0.7	18.9 ± 0.3	28.6 ± 0.4	28.5 ± 0.4
A-Meta-Both	22.5 ± 0.6	19.2 ± 0.3	28.9 ± 0.4	28.8 ± 0.4

5.9%p Error Increase   3.7%p Error Increase

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Results**
  - Change in accuracy for the number of perturbations
    - Meta-Self shows the best performance.

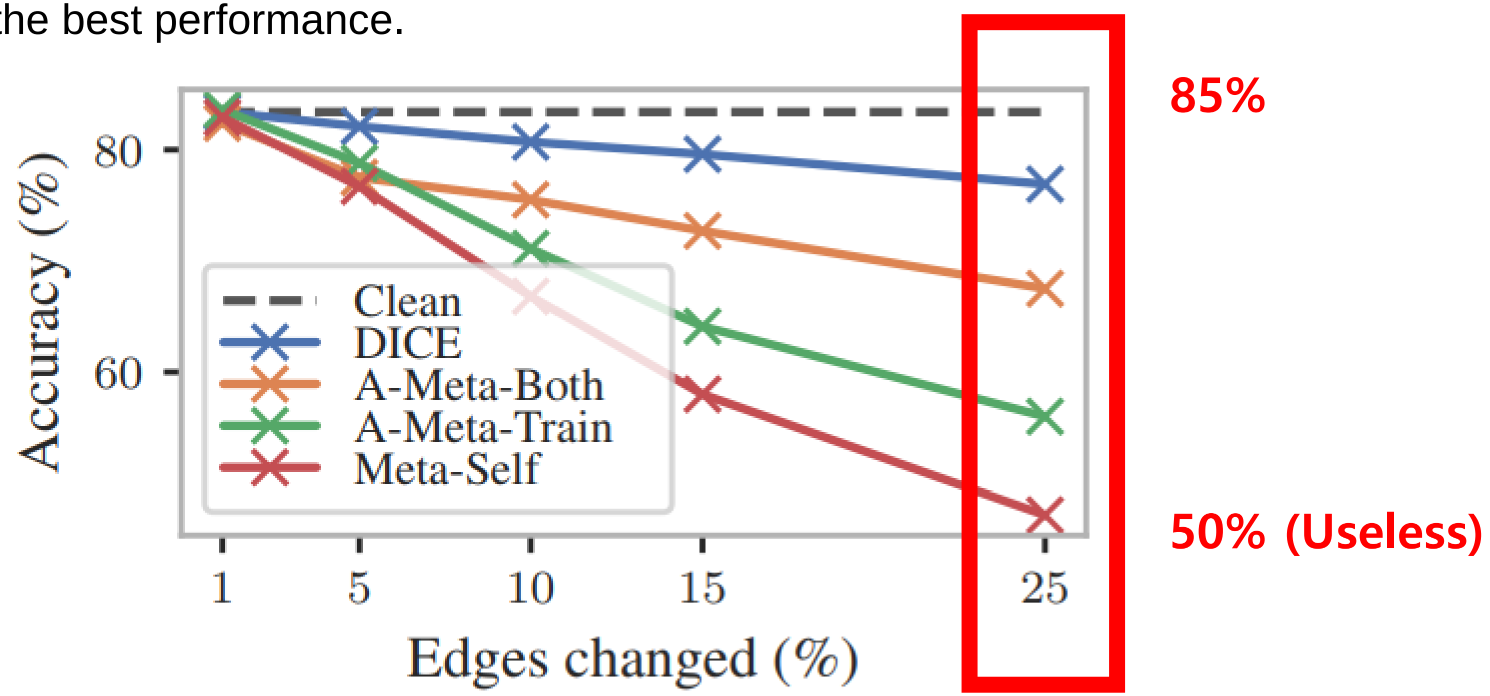


Figure 1: Change in accuracy of GCN on CORA-ML for increasing number of perturbations.

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Results**
  - Impact of graph structure and trained weights.
    - For all three measures no clear distinction can be made.

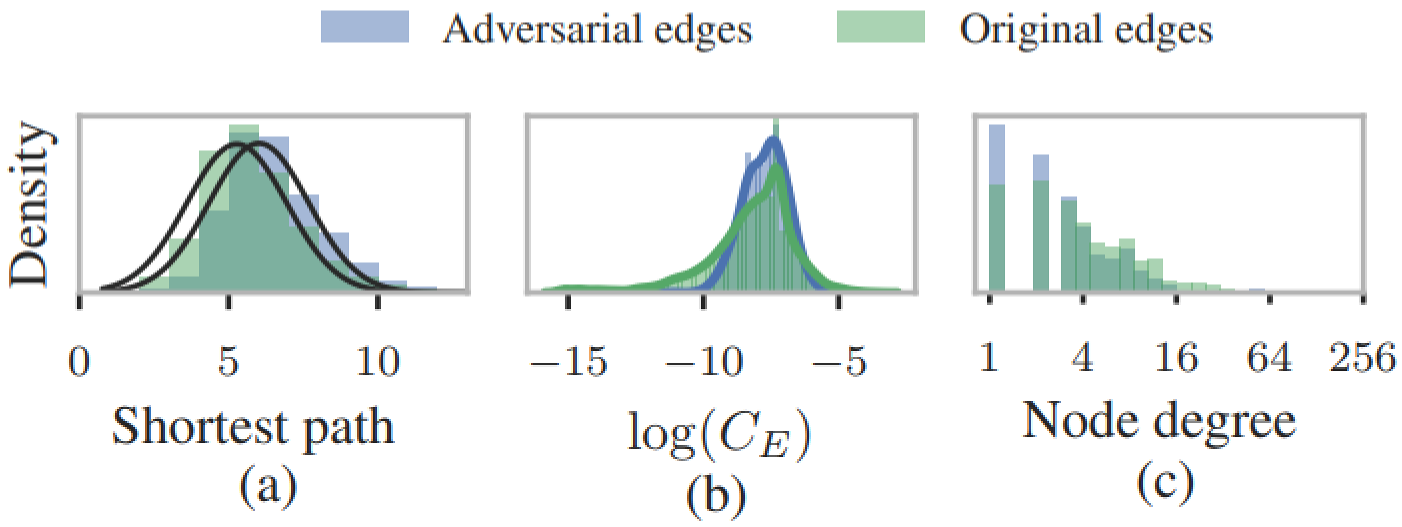


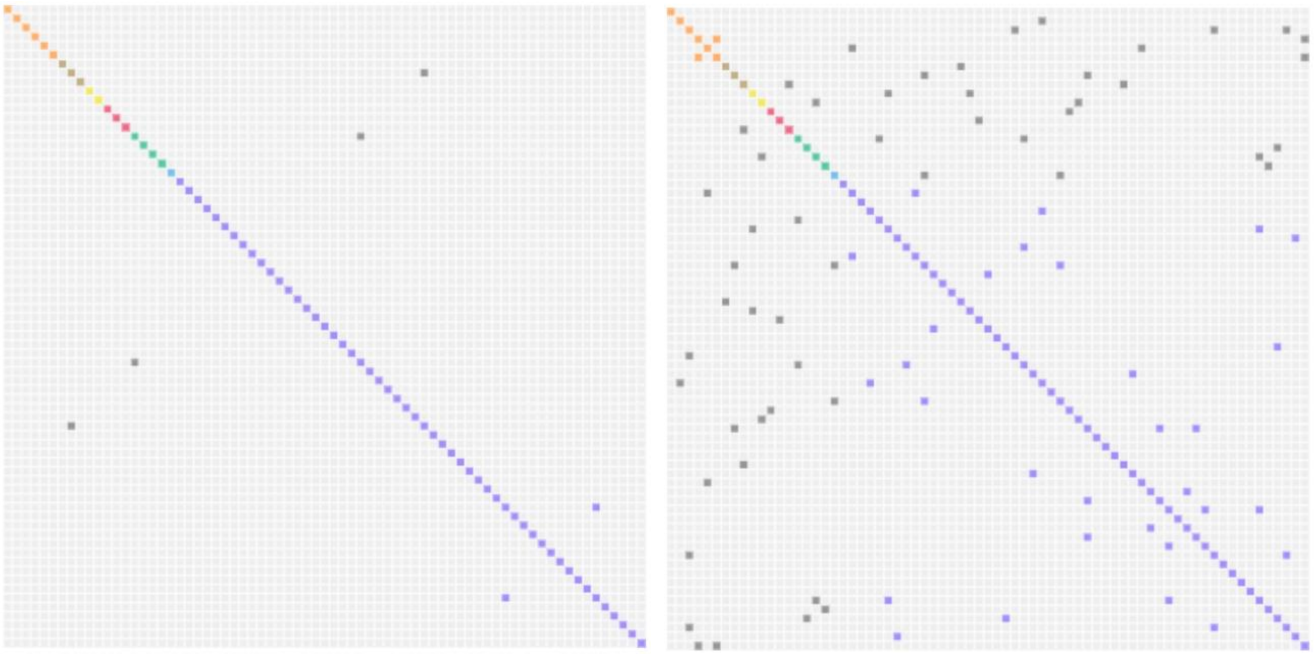
Figure 3: Analysis of adversarially inserted edges

(*) C_E: the edge betweenness centrality
(= the number of the shortest paths that go through an edge in a graph or network)

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Results (Personal Curiosity)**
  - Visualization of edges. (citeseer, only effected nodes – 71/2110)
  - No removed edges!!! = Only created edges.
  - Effected nodes(71) are not only from training nodes(52).

Before



After

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Conclusion**
  - **Use meta-gradients to solve** the bi-level optimization problem underlying the challenging class of **poisoning adversarial attacks.**
  - **Show that attacks** created using our meta-gradient approach consistently lead to a **strong decrease in classification performance** of graph convolutional models.
  - **Check small statistical differences** of **adversarial and ‘normal’ edges**.

Adversarial Attacks on Graph Neural Networks via Meta Learning

Adversarial Attacks on Graph Neural Networks via Meta Learning (Appendix)

Hoki Kim

Seoul National University

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Unnoticeability constraint on the degree distribution**
  - Check node degree distributions of $\hat{G}$, G stem from the same distribution

$\boldsymbol{Def}$)

$d_{min} = 2$ (in code)

$$D_G = \{d_v^G | v \in V, d_v^G \geq d_{min}\}$$

$$\alpha_G = 1 + |D_G| \cdot \left[\sum_{\{d_i \in D_G\}} log \frac{d_i}{d_{min} - \frac{1}{2}} \right]^{-1}$$

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

$\boldsymbol{Likelihood\ ratio\ test}$)

$$l(H_0) = l(D_G \cup D_{\hat{G}}), l(H_1) = l(D_G) + l(D_{\hat{G}})$$

$$\Lambda(G, \hat{G}) = -2l(H_0) + 2l(H_1) \sim \chi^2$$

H_0 : Null hypotheses (=Come from the same power law distribution)

Zügner, et al. "Adversarial attacks on neural networks for graph data." 2018.

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Algorithm**
  - Meta Learning v.s. Approximate Meta Learning

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;
  for t *in* $0 \ldots T-1$ **do**
    $\theta_{t+1} \leftarrow \mathrm{step}(\theta_t, \nabla_{\theta_t}\mathcal{L}_{\mathrm{train}}(f_{\theta_t}(\hat{A}, X)); C_L)$; `// update e.g. via gradient descent`
  `// Compute meta gradient via backprop through the training procedure`
  $\nabla_{\hat{A}}^{\mathrm{meta}} \leftarrow \nabla_{\hat{A}}\mathcal{L}_{\mathrm{self}}(f_{\theta_T}(\hat{A}, X); \hat{C}_U)$;
  $S \leftarrow \nabla_{\hat{A}}^{\mathrm{meta}} \odot (-2\hat{A} + 1)$; `// 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 $\hat{A}$;
$\hat{G} \leftarrow (\hat{A}, X)$;
return : $\hat{G}$

Algorithm 2: Poisoning attack on GNNs with approximate meta gradients and self-training

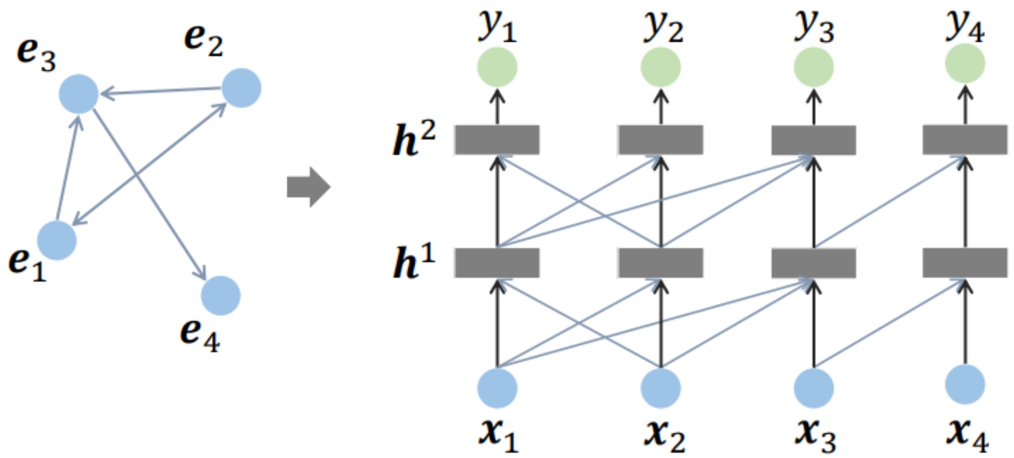
Input: Graph $G = (A, X)$, modification budget Δ, number of training iterations T, gradient weighting λ, training class labels C_L
Output: Modified graph $\hat{G} = (\hat{A}, X)$

$\hat{\theta} \leftarrow$ 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;
  $\nabla_{\hat{A}}^{\mathrm{meta}} \leftarrow \lambda\nabla_{\hat{A}}\mathcal{L}_{\mathrm{train}}(f_{\theta_0}(\hat{A}; X); C_L) + (1-\lambda)\nabla_{\hat{A}}\mathcal{L}_{\mathrm{self}}(f_{\theta_0}(\hat{A}; X); \hat{C}_U)$
  for t *in* $0 \ldots T-1$ **do**
    $\theta_{t+1} \leftarrow \mathrm{step}(\theta_t, \nabla_{\theta_t}\mathcal{L}_{\mathrm{train}}(f_{\theta_t}(\hat{A}, X)); C_L)$; `// update e.g. via gradient descent`
    $\tilde{\theta}_{t+1} \leftarrow \mathrm{stop\_gradient}(\theta_{t+1})$; `// no backprop through training`
    $\nabla_{\hat{A}}^{\mathrm{meta}} \leftarrow \nabla_{\hat{A}}^{\mathrm{meta}} + \lambda\nabla_{\hat{A}}\mathcal{L}_{\mathrm{train}}(f_{\tilde{\theta}_{t+1}}(\hat{A}; X); C_L) + (1-\lambda)\nabla_{\hat{A}}\mathcal{L}_{\mathrm{self}}(f_{\tilde{\theta}_{t+1}}(\hat{A}; X); \hat{C}_U)$
  $S \leftarrow \nabla_{\hat{A}}^{\mathrm{meta}} \odot (-2\hat{A} + 1)$; `// 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 $\hat{A}$;
$\hat{G} \leftarrow (\hat{A}, X)$;
return : $\hat{G}$

Adversarial Attacks on Graph Neural Networks via Meta Learning

▪ Column Network

- Consider edge information of graph as linking hidden features.



$$c_{ir}^t = \frac{1}{|\mathcal{N}_r(i)|} \sum_{j \in \mathcal{N}_r(i)} \boldsymbol{h}_j^{t-1}$$

$$\boldsymbol{h}_i^t = g\left(\boldsymbol{b}^t + W^t \boldsymbol{h}_i^{t-1} + \frac{1}{z} \sum_{r=1}^{R} V_r^t \boldsymbol{c}_{jr}^t\right)$$

$N(i)$: the set of all neighbors of i-th node e_i.
$N_r(i)$: $N(i) = \cup_{r \in \mathrm{R}} N_r(i)$ for relation r.

g : activation function.
z : pre-defined constant to prevent the sum of parameterized contexts from growing too large for complex relations.

W, V : weight matrices.
B : bias.

Pham, Trang, et al. "Column networks for collective classification." Thirty-First AAAI Conference on Artificial Intelligence. 2017.

Adversarial Attacks on Graph Neural Networks via Meta Learning (Code & Results)

Hoki Kim

Seoul National University

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Code URL** https://github.com/Harry24k/gnn-meta-attack
- **Code Information**
  - 이미 다양한 Reproduce 존재
    - https://github.com/danielzuegner/gnn-meta-attack (Official, Tensorflow)
    - https://github.com/ChandlerBang/pytorch-gnn-meta-attack (Pytorch)
    - https://github.com/Kaushalya/gnn-meta-attack-pytorch (Pytorch)
  - 위 목록 중 개인적으로 가장 깔끔하다고 판단되는 두 번째 코드를 기반으로 재구현

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Code Manual : Module**
  - Loader.py : 데이터 로드 관련 모듈
  - Models.py : 모델 구조 모듈
  - Train.py : 학습 관련 모듈
  - Metaattack.py : 공격 모듈

  - Poison.py : Metaattack 기반 Perturbed Data 생성 모듈
  - Main.py : 실행 모듈

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Code Manual : Loader.py**
  - 기존 코드에서 Torch로 불러오기 쉽도록 오른쪽과 같이 load_data 재구현
  - Random_state를 통해 reproducing하기 쉽도록 구현
  - 동시에 기존과 달리 train, test의 과정에서 각각 test, train label에 접근하지 못하게끔 label을 수정하여 return하도록 함.
    - Ex) Train Data 반환 시 Test Data의 Index에 해당하는 Label은 -1로 변환한 뒤 반환.

```python
def load_data(data_name, train=True, test_size=0.9, random_state=1,
              from_sparse=True, to_sparse=False):

    print('Loading {} dataset...'.format(data_name))
    adj, features, labels = get_adj(data_name, from_sparse)
    features = sp.csr_matrix(features, dtype=np.float32)

    labels = torch.LongTensor(labels)
    if to_sparse:
        adj = sparse_mx_to_torch_sparse_tensor(adj)
        features = sparse_mx_to_torch_sparse_tensor(features)
    else:
        features = torch.FloatTensor(np.array(features.todense()))
        adj = torch.FloatTensor(adj.todense())

    train_idx, test_idx, _, _ = train_test_split(list(range(len(labels))), labels.numpy(),
                                                 test_size=test_size, random_state=random_state)

    if train :
        labels[test_idx] = -1
    else :
        labels[train_idx] = -1

    return features, adj, labels
```

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Code Manual : Models.py**
  - 기존 코드에서 Adjacency Matrix를 Model에 입력하기 전에 Laplacian Filtering을 적용한 후 입력했던 것을, Model 안에 삽입하여 Forward 시 원래 Adjacency Matrix를 넣을 수 있도록 함.
  - 불필요한 파라미터(Use ReLU, Use Dropout 등) 제거를 통해 Model 단순화

```python
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, with_bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if with_bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

   def normalize_adj_tensor(self, adj):
       mx = adj + torch.eye(adj.shape[0]).to(next(self.parameters()).device)
       rowsum = mx.sum(1)
       r_inv = rowsum.pow(-1/2).flatten()
       r_inv[torch.isinf(r_inv)] = 0.
       r_mat_inv = torch.diag(r_inv)
       mx = r_mat_inv @ mx
       mx = mx @ r_mat_inv
       return mx
```

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Code Manual : Train.py**
  - Loader의 변경에 맞게 수정.
    - -1의 label 처리를 위해 select_index 구현

```python
def select_index(y, value, same=True) :
    if same :
        idx = (y == value).nonzero().view(-1)
    else :
        idx = (y != value).nonzero().view(-1)
    return idx
```

  - Train과 get_acc 함수 정의를 통해, 학습과 검증 분리

```python
def train(model, data, device, save_path=None, epochs=200, loss=None, optimizer=None):

    try:
        features, edges, labels = data
    except Exception as e:
        print(e)
        raise RuntimeError("d

    if loss is None :
        loss = nn.CrossEntrop

    if optimizer is None :
        optimizer = optim.Ada

    model = model.to(device)
    model.train()

    features, edges = feature
    labels = labels.to(device

    for i in range(epochs):
        pre = model(features,

        idx = select_index(la
        pre, Y = pre[idx], la
```

```python
def get_acc(model, data, device):
    # Set Cuda or Cpu
    device = torch.device(device)
    model.to(device)

    try:
        features, edges, labels = data
    except Exception as e:
        print(e)
        raise RuntimeError("data must be (features, edges, labels)")

    features, edges = features.to(device), edges.to(device)
    labels = labels.to(device)

    # Set Model to Evaluation Mode
    model.eval()

    pre = model(features, edges)

    idx = select_index(labels, -1, same=False)
    pre, Y = pre[idx], labels[idx]

    _, pre = torch.max(pre.data, 1)
    total = 0. + pre.size(0)
    correct = 0. + (pre == Y).sum()

    return (correct/total).item()*100
```

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Code Manual : Metaattack.py**
  - 기존 구조는 Model과 자동적으로 호환되지 않음
    - Dictionary를 활용한 Meta Gradient 계산 → Model 구조가 고정되어있음

```python
class Metattack(BaseMeta):

    def __init__(self, nfeat, hidden_sizes, nclass, nnodes, dropout, train_iters,
                 attack_features, device, lambda_=0.5, with_relu=False, with_bias=False, lr=0.1, momentum=0.9):

        super(Metattack, self).__init__(nfeat, hidden_sizes, nclass, nnodes, dropout, train_iters, attack_featur

        self.momentum = momentum
        self.lr = lr

        self.weights = []
        self.biases = []
        self.w_velocities = []
        self.b_velocities = []

        previous_size = nfeat
        for ix, nhid in enumerate(self.hidden_sizes):
            weight = Parameter(torch.FloatTensor(previous_size, nhid).to(device))
            bias = Parameter(torch.FloatTensor(nhid).to(device))
            w_velocity = torch.zeros(weight.shape).to(device)
            b_velocity = torch.zeros(bias.shape).to(device)
            previous_size = nhid

            self.weights.append(weight)
            self.biases.append(bias)
            self.w_velocities.append(w_velocity)
            self.b_velocities.append(b_velocity)
```

```python
def get_meta_grad(self, features, adj_norm, idx_train, idx_unlabeled, labels, labels_self_training):

    hidden = features
    for ix, w in enumerate(self.weights):
        b = self.biases[ix] if self.with_bias else 0
        if self.sparse_features:
            hidden = adj_norm @ torch.spmm(hidden, w) + b
        else:
            hidden = adj_norm @ hidden @ w + b
        if self.with_relu:
            hidden = F.relu(hidden)

    output = F.log_softmax(hidden, dim=1)

    loss_labeled = F.nll_loss(output[idx_train], labels[idx_train])
    loss_unlabeled = F.nll_loss(output[idx_unlabeled], labels_self_training[idx_unlabeled])
    loss_test_val = F.nll_loss(output[idx_unlabeled], labels[idx_unlabeled])
```

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Code Manual : Metaattack.py**
  - 따라서 외부 패키지(Higher)를 사용하여 해당 부분 재구현

```
with higher.innerloop_ctx(model, optimizer) as (fmodel, diffopt):
```

∇higher

```
for i in range(train_iters):
    pre = fmodel(features, edges)
    pre, Y = select_index(pre, labels)
    cost = loss(pre, Y)

    diffopt.step(cost)
```

(1) Train model

$$\theta_T = \theta_{T-1} - \alpha \nabla_{\theta_{T-1}} L_{train}\left(f_{\theta_{T-1}}(G)\right)$$

```
pre = fmodel(features, edges)
pre, Y = select_index(pre, labels)
cost = self.lambda_ * loss(pre, Y) + (1-self.lambda_) * loss(pre, Y)

return torch.autograd.grad(cost, self.adj_changes, retain_graph=False)[0]
```

(2) Get Meta Gradient

$$\nabla_G^{meta} = \nabla_G L_{atk}\left(f_{\theta_T}(G)\right)$$

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Code Manual : Main.py, Poison.py**
  - User가 Reproducing하기 쉽도록 이전 모듈들을 기반으로 설계함
  - 추가적으로 공격 후의 데이터를 저장하여 나중에 검증하기 쉽게 수정

Training

To train the model(s) in the paper, run this command:

```
# cora_ml
python main.py --train True --hidden 16 --data-name cora_ml --save-path sample.pth
```

Evaluation

To evaluate the model(s) saved locally, run this command:

```
# cora_ml
python main.py --train False --hidden 6 --data-name cora_ml --save-path sample.pth
```

Generate Poisoned Data with Meta Attack

Here is how to generate poisoned data :

```
# cora_ml
python poison.py --hidden 6 --lambda_ 0.5 --train-iters 15 --perturb-rate 0.05 --save-path sample.pth --data-name cora_m
```

```
_, _, full_labels = load_data(data_name=data_name, train=True

sA = sparse.csr_matrix(modified_adj.detach().cpu().numpy())
sB = sparse.csr_matrix(features.detach().cpu().numpy())
sC = full_labels.numpy()

loader = {}
np.savez('data/'+save_path,
         adj_data=sA.data,
         adj_indices=sA.indices,
         adj_indptr=sA.indptr,
         adj_shape=sA.shape,
         attr_data=sB.data,
         attr_indices=sB.indices,
         attr_indptr=sB.indptr,
         attr_shape=sB.shape,
         labels=sC)
```

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Results : Cora_ml**
  - GPU Memory의 한계로 인해, Meta Learning의 training iterations T를 기존 100보다 작은 15로 수정함.
  - 그 결과 약간의 성능 차이가 존재하나, 비슷한 경향을 얻을 수 있었음.

cora_ml

Description	Perturb Rate	Accuracy(Dropped)	Reported	Data Name
Clean	0.00	85.80%	83.40%	cora_ml.npz
Self	0.05	80.43%(5.37%p)	75.50%(7.90%p)	cora_ml_self_5.npz
Both	0.05	81.42%(4.38%p)	-	cora_ml_both_5.npz
Train	0.05	82.52%(3.28%p)	78.00%(5.40%p)	cora_ml_train_5.npz
Self	0.20	58.01%(27.79%p)	-	cora_ml_self_20.npz
Both	0.20	67.73%(18.07%p)	-	cora_ml_both_20.npz
Train	0.20	80.03%(5.77%p)	-	cora_ml_train_20.npz

정확도 28% 하락

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Results : Citeseer**
  - GPU Memory의 한계로 인해, Meta Learning의 training iterations T를 기존 100보다 작은 15로 수정함.
  - 그 결과 약간의 성능 차이가 존재하나, 비슷한 경향을 얻을 수 있었음.

citeseer

Description	Perturb Rate	Accuracy(Dropped)	Reported	Data Name
Clean	0.00	70.41%	72.50%	citeseer.npz
Self	0.05	64.35%(6.06%p)	65.40%(7.10%p)	citeseer_self_5.npz
Both	0.05	61.08%(9.33%p)	-	citeseer_both_5.npz
Train	0.05	68.98%(1.43%p)	69.70%(2.80%p)	citeseer_train_5.npz

정확도 9% 하락

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Results : Polblogs**
  - GPU Memory의 한계로 인해, Meta Learning의 training iterations T를 기존 100보다 작은 15로 수정함.
  - 그 결과 약간의 성능 차이가 존재하나, 비슷한 경향을 얻을 수 있었음.

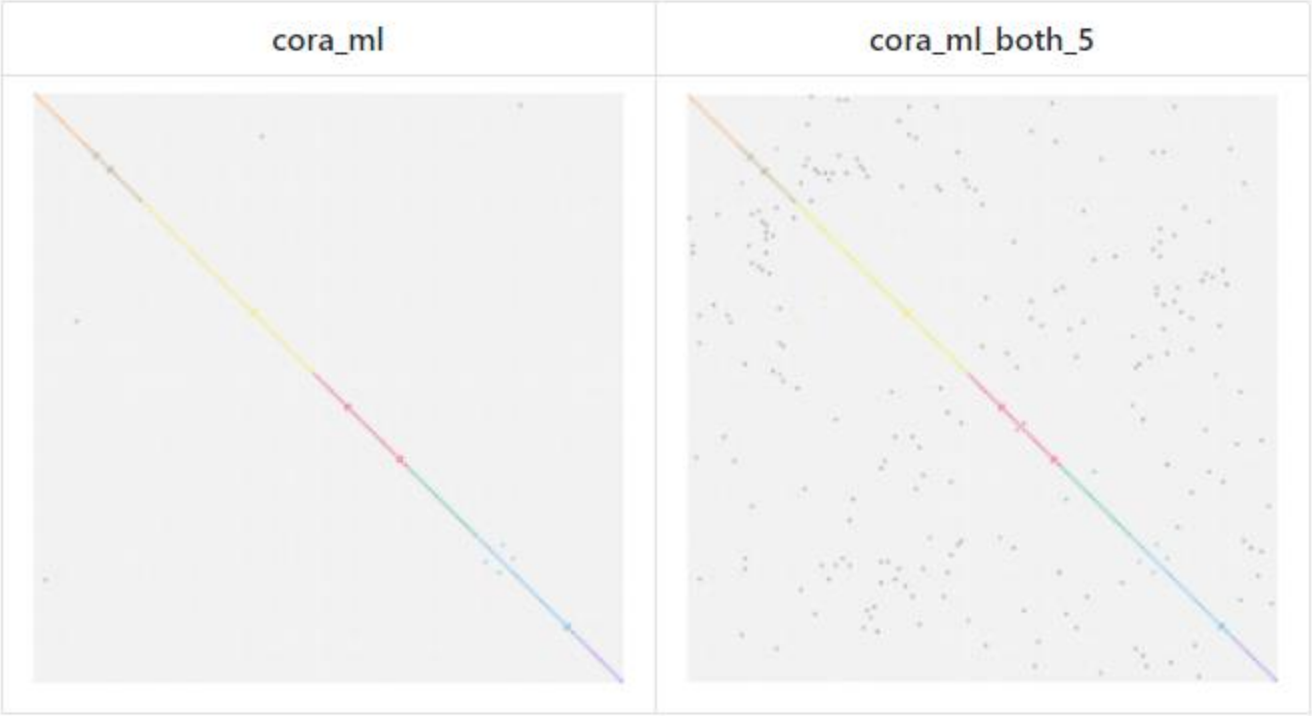
polblogs

Description	Perturb Rate	Accuracy(Dropped)	Reported	Data Name
Clean	0.00	93.18%	93.60%	polblogs.npz
Self	0.05	74.36%(18.82%p)	77.50%(16.10%p)	polblogs_self_5.npz
Both	0.05	78.27%(14.91%p)	-	polblogs_both_5.npz
Train	0.05	86.09%(7.09%p)	83.70%(9.90%p)	polblogs_train_5.npz

정확도 19% 하락

Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Results : Visualization**
  - 더 나아가, Edge의 변화를 관측하기 위해 Visualization 실행
  - 생긴 Edge들이 훨씬 많음을 육안으로 관찰 가능



Adversarial Attacks on Graph Neural Networks via Meta Learning

- **Results : Statistical Analysis**
  - 논문에서는 알 수 없었던 Node, Edge의 통계적 변화 관찰
  - cora_ml → cora_ml_both_5
  - Edges : 오직 생성을 통해 공격
    - Deleted Edges: 0
    - Created Edges: 798
  - Nodes : Train, Test 할 것 없이 모두 영향 받음
    - Effected Train Nodes: 282
    - Effected Test Nodes: 202

  - 다른 데이터 셋에서도 비슷한 경향 관측
    - Citeseer
      - Deleted Edges가 0은 아니나 Created Edge가 훨씬 많음
      - 마찬가지로 Train, Test 할 것 없이 모두 영향 받음