Robustness of GNNs: Adversarial Training

Presentation by Matthew Weston
mweston3@illinois.edu

Paper 1: Batch Virtual Adversarial Training (BVAT)

Intuition: Graph Convolutional Networks (GCNs) can benefit from regularization; adversarial training provides a way of ensuring that small perturbations to the input will generally not result in large perturbations to the output.

- This is helpful to GCNsfor the same reason it's helpful everywhere else. If the output distribution is smoother, then we're more likely to generalize well to inputs that are similar but not identical to those in our training set.
- This paper discusses how VAT

VAT (Virtual Adversarial Training)

VAT works to encourage a smooth, robust model by training against worst-case localized adversarial perturbation.

Defines local distributional smoothness (LDS) as below:

- $p(y \mid x, W)$ is the prediction distribution parameterized by W, the set of trainable parameters.
- D_{KL} is the KL divergence of two distributions.
- r_{vadv} is the virtual adversarial perturbation applied to the input, calculated to maximize the above KL divergence.

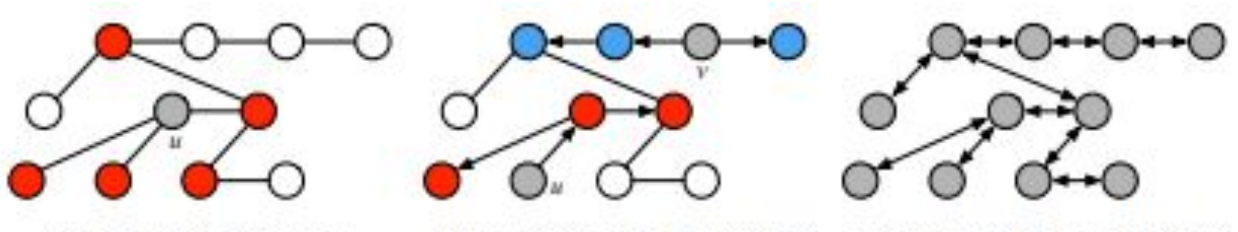
$$\mathrm{LDS}(x, W, r_{\mathrm{vadv}}) = D_{\mathrm{KL}}\big(p(y|x, \hat{W}) \| p(y|x + r_{\mathrm{vadv}}, W)\big).$$

VAT, Applied to Graphs

VAT is a means of regularizing the training process to smooth the output distribution of a classifier, relative to the input distribution

Applying VAT to graphs is difficult, since the prediction for any node depends on its entire receptive field.

This means that the virtual adversarial perturbation (VAP) for any node *x* will modify the features of every node in its receptive field, meaning that any batch including nodes whose receptive fields include *x* will result in an overall perturbation that is *not* x’s worst-case VAP.

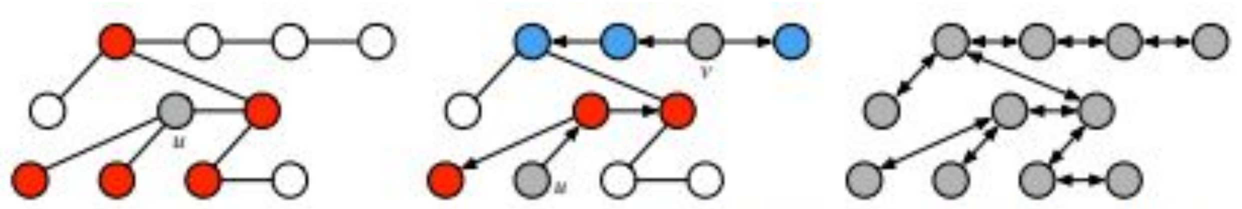


(a) Receptive Field of node *u* (b) Perturbations for nodes in S-BVAT (c) Perturbations for nodes in O-BVAT

Solving the Problem

There are two solutions to the problem described previously.

- The first is to take only sets of nodes whose receptive fields do not intersect.
  - This is only practical in a subset of situations. A GNN with many layers and few nodes, for example, makes this much more difficult.
- The second is to generate a set of adversarial perturbations that applies to every node in the graph.



(a) Receptive Field of node u (b) Perturbations for nodes in S-BVAT (c) Perturbations for nodes in O-BVAT

S-BVAT

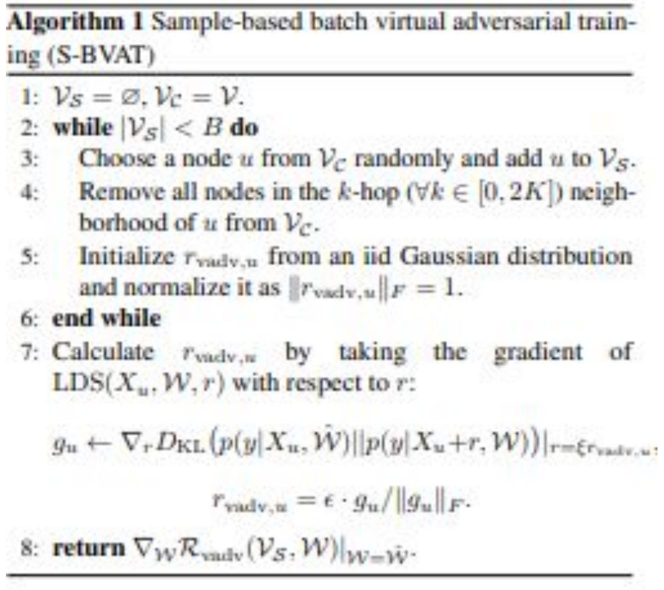
- Sample-based BVAT organizes itself in order to prevent adversarial perturbations for different nodes from interacting with each other.
- VAPs are generated for a subset V_S of node set V, whose receptive fields do not overlap
- For a GCN with K convolutional layers, we sample nodes to form a batch of size B as follows: (D_{UV} is a distance function)

$$\mathcal{V}_S = \{u | u \in \mathcal{V}\}, \quad \text{s.t. } |\mathcal{V}_S| = B, \ \forall u, v \in \mathcal{V}_S, D_{uv} \geq 2K.$$

S-BVAT, Continued

- Because the fields do not overlap, we can simply use the average LDS as our VAT regularization term.

$$\mathcal{R}_{\text{vadv}}(\mathcal{V}_S, \mathcal{W}) = \frac{1}{B}\sum_{u\in\mathcal{V}_S} LDS(X_u, \mathcal{W}, r_{\text{vadv},u}).$$



Algorithm 1 Sample-based batch virtual adversarial training (S-BVAT)

```
1: $\mathcal{V}_S = \varnothing, \mathcal{V}_C = \mathcal{V}$.
2: while $|\mathcal{V}_S| < B$ do
3:     Choose a node $u$ from $\mathcal{V}_C$ randomly and add $u$ to $\mathcal{V}_S$.
4:     Remove all nodes in the $k$-hop ($\forall k \in [0, 2K]$) neighborhood of $u$ from $\mathcal{V}_C$.
5:     Initialize $r_{\text{vadv},u}$ from an iid Gaussian distribution and normalize it as $\|r_{\text{vadv},u}\|_F = 1$.
6: end while
7: Calculate $r_{\text{vadv},u}$ by taking the gradient of LDS$(X_u, \mathcal{W}, r)$ with respect to $r$:
       $g_u \leftarrow \nabla_r D_{\text{KL}}\big(p(y|X_u, \hat{\mathcal{W}})\|p(y|X_u+r, \mathcal{W})\big)|_{r=\xi r_{\text{vadv},u}},$
       $r_{\text{vadv},u} = \epsilon \cdot g_u / \|g_u\|_F.$
8: return $\nabla_{\mathcal{W}} \mathcal{R}_{\text{vadv}}(\mathcal{V}_S, \mathcal{W})|_{\mathcal{W}=\hat{\mathcal{W}}}$.
```

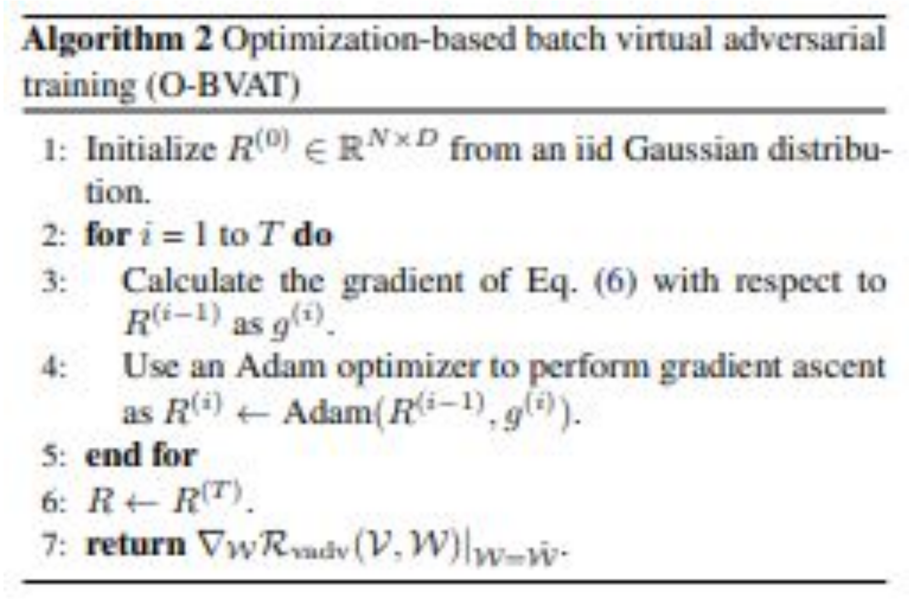
O-BVAT

- O-BVAT, or Optimization-based BVAT, is the second strategy proposed in this paper.
- O-BVAT maximizes the average LDS loss for the entire perturbation matrix R, which corresponds to the entire feature matrix X of our input.
  - We ensure that the neighborhood perturbations of each node are sufficiently adversarial, and disincentivize the norm of R so that the perturbations are sufficiently small.
- O-BVAT's perturbations are determined as follows: (||R||F^2 is the Frobenius norm of R, and gamma is a hyperparameter)

$$\max_{R} \frac{1}{N} \sum_{u \in \mathcal{V}} D_{\mathrm{KL}}\left(p(y|X_u, \hat{\mathcal{W}}) || p(y|X_u + R_u, \mathcal{W})\right) - \gamma \cdot \|R\|_F^2, \quad (6)$$

O-BVAT, Continued

- The function in the previous slide is optimized using Adam, for some number of iterations T.

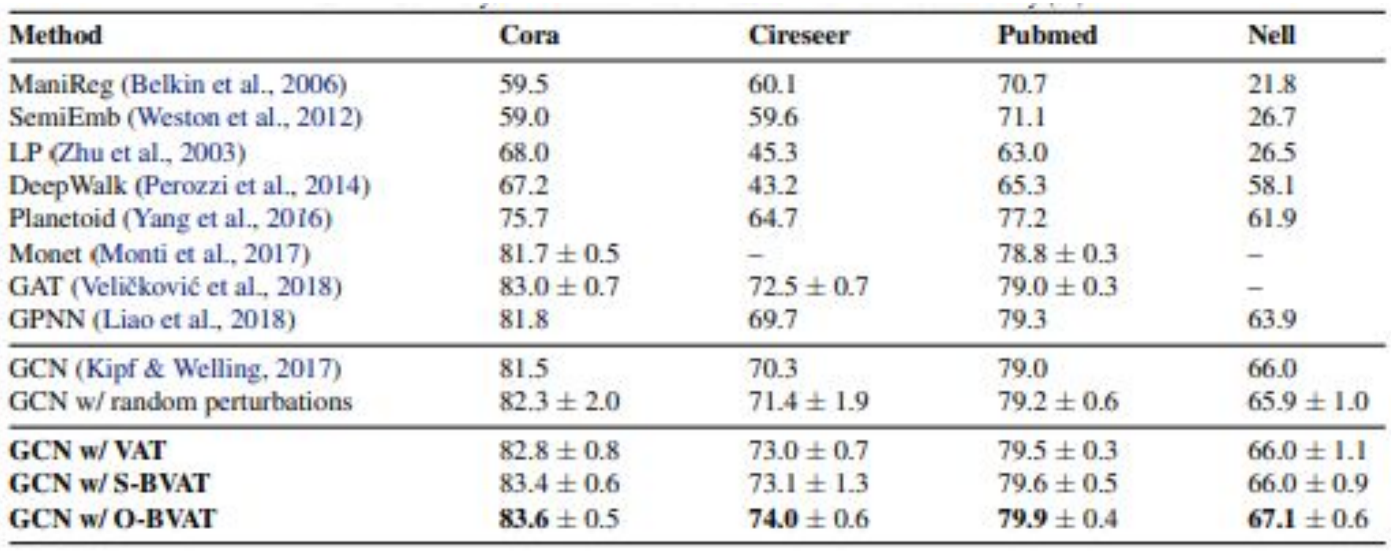


Algorithm 2 Optimization-based batch virtual adversarial training (O-BVAT)

```
1: Initialize $R^{(0)} \in \mathbb{R}^{N \times D}$ from an iid Gaussian distribution.
2: for $i = 1$ to $T$ do
3:     Calculate the gradient of Eq. (6) with respect to $R^{(i-1)}$ as $g^{(i)}$.
4:     Use an Adam optimizer to perform gradient ascent as $R^{(i)} \leftarrow \text{Adam}(R^{(i-1)}, g^{(i)})$.
5: end for
6: $R \leftarrow R^{(T)}$.
7: return $\nabla_{\mathcal{W}} \mathcal{R}_{\text{vadv}}(\mathcal{V}, \mathcal{W})|_{\mathcal{W}=\hat{\mathcal{W}}}$.
```

Experiments

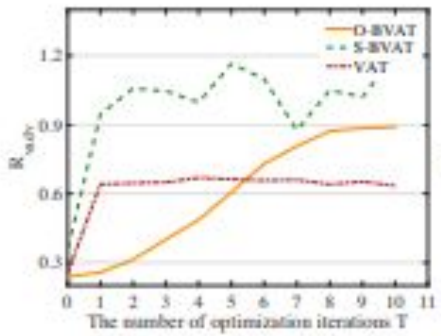
- The proposed algorithms are evaluated on four popular benchmarks, and are shown to significantly boost the performance of GCNs relative to an array of state-of-the-art methods.
- Overall results:

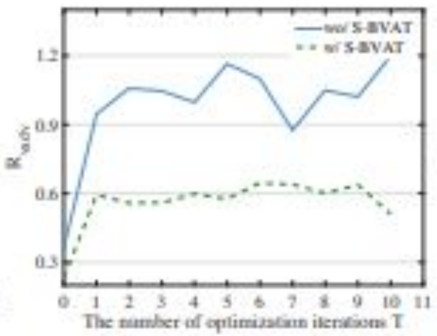


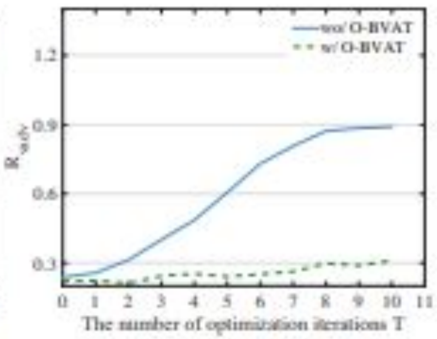
Method	Cora	Cireseer	Pubmed	Nell
ManiReg (Belkin et al., 2006)	59.5	60.1	70.7	21.8
SemiEmb (Weston et al., 2012)	59.0	59.6	71.1	26.7
LP (Zhu et al., 2003)	68.0	45.3	63.0	26.5
DeepWalk (Perozzi et al., 2014)	67.2	43.2	65.3	58.1
Planetoid (Yang et al., 2016)	75.7	64.7	77.2	61.9
Monet (Monti et al., 2017)	81.7 ± 0.5	–	78.8 ± 0.3	–
GAT (Veličković et al., 2018)	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3	–
GPNN (Liao et al., 2018)	81.8	69.7	79.3	63.9
GCN (Kipf & Welling, 2017)	81.5	70.3	79.0	66.0
GCN w/ random perturbations	82.3 ± 2.0	71.4 ± 1.9	79.2 ± 0.6	65.9 ± 1.0
GCN w/ VAT	82.8 ± 0.8	73.0 ± 0.7	79.5 ± 0.3	66.0 ± 1.1
GCN w/ S-BVAT	83.4 ± 0.6	73.1 ± 1.3	79.6 ± 0.5	66.0 ± 0.9
GCN w/ O-BVAT	$\mathbf{83.6} \pm 0.5$	$\mathbf{74.0} \pm 0.6$	$\mathbf{79.9} \pm 0.4$	$\mathbf{67.1} \pm 0.6$

Effectiveness of BVAT

- The authors evaluate S-BVAT and O-BVAT on their ability to generate adversarial perturbations.
  - The first graph shows the regularization term (which tracks how effectively we have altered model predictions) under a standard VAT algorithm, along with the two proposed algorithms
  - The second compares S-BVAT's regularization term on a model trained with it, and a model trained without it, and indicates how effectively it improves robustness.
  - The third does the same for O-BVAT.







Paper 2: Topology Attack and Defense for GNNs

Intuition: By perturbing the topology of a graph; including adding and deleting edges, the authors develop something akin to the sort of adversarial modifications we see for other kinds of machine learning models, where a few small adjustments can result in an adversarial prediction.

- Additionally, these perturbed samples can be fed back into the training process for the purpose of developing a more robust model.

Defining a Topology Attack

- A topology attack involves introducing a boolean, symmetric matrix S, consisting of values {0,1}, in size NxN, to a graph with N nodes.
  - Each value determines whether an edge between two nodes has been modified (added or removed, depending on its original state).
- We generate a perturbed graph topology A’. Ā is the supplement of A.
- Goal: An attacker wants to find the minimum edge perturbations encoded in S to mislead a GNN.

$$\mathbf{A}' = \mathbf{A} + \mathbf{C} \circ \mathbf{S}, \ \mathbf{C} = \bar{\mathbf{A}} - \mathbf{A}.$$

Attacker Loss

- Let Z(S, W; A, {x}) denote the prediction probability distribution of a GNN with parameters W on graph A, with perturbations S.
- Then, similarly to attacker loss when attacking image classifiers (under many models), we can describe attacker loss for perturbation attacks on GNNs as follows:

$$f_i(\mathbf{S}, \mathbf{W}; \mathbf{A}, \{\mathbf{x}_i\}, y_i) = \max\left\{ Z_{i,y_i} - \max_{c \neq y_i} Z_{i,c}, -\kappa \right\}$$

- K, here, is the confidence level associated with incorrect predictions.
- When developing an attack, we want to minimize the per-node attack loss

Attack Model 1 (Static GNN)

- In this attack model, we aim to attack a pre-defined GNN with known weights W.
- Our attack generation problem can thus be cast as follows:

$$\begin{array}{ll} \underset{\mathbf{s}}{\text{minimize}} & \sum_{i \in \mathcal{V}} f_i(\mathbf{s}; \mathbf{W}, \mathbf{A}, \{\mathbf{x}_i\}, y_i) \\ \text{subject to} & \mathbf{1}^\top \mathbf{s} \leq \epsilon, \ \mathbf{s} \in \{0, 1\}^n, \end{array}$$

- Here, S is replaced with its vector form s.

PGD Topology Attack

- The key alteration made in order to address attack model one is the transformation of s to its convex hull, which is a continuous function from zero to one.

$$\begin{array}{ll} \underset{\mathbf{s}}{\text{minimize}} & f(\mathbf{s}) := \sum_{i \in \mathcal{V}} f_i(\mathbf{s}; \mathbf{W}, \mathbf{A}, \{\mathbf{x}_i\}, y_i) \\ \text{subject to} & \mathbf{s} \in \mathcal{S}, \end{array}$$

- We now, however, face the problem of recovering a binary solution from our continuous one (obtained through projected gradient descent on attack loss).
    - Since s can be interpreted as a probability vector, a near-optimal binary perturbation can thus be achieved.

```
1: Input: $\mathbf{s}^{(0)}$, $\epsilon > 0$, learning rate $\eta_t$, and iterations $T$
2: for $t = 1, 2, \ldots, T$ do
3:     gradient descent: $\mathbf{a}^{(t)} = \mathbf{s}^{(t-1)} - \eta_t \nabla f(\mathbf{s}^{(t-1)})$
4:     call projection operation in (11)
5: end for
6: call Algorithm 1 to return $\mathbf{s}^*$, and the resulting $\mathbf{A}'$ in (4).
```

```
1: Input: probabilistic vector $\mathbf{s}$, $K$ is # of random trials
2: for $k = 1, 2, \ldots, K$ do
3:     draw binary vector $\mathbf{u}^{(k)}$ following
           $u_i^{(k)} = \begin{cases} 1 & \text{with probability } s_i \\ 0 & \text{with probability } 1 - s_i \end{cases}, \forall i$    (9)
4: end for
5: choose a vector $\mathbf{s}^*$ from $\{\mathbf{u}^{(k)}\}$ which yields the smallest
   attack loss $f(\mathbf{u}^{(k)})$ under $\mathbf{1}^T \mathbf{s} \leq \epsilon$.
```

Attack Model 2 (Responsive GNN)

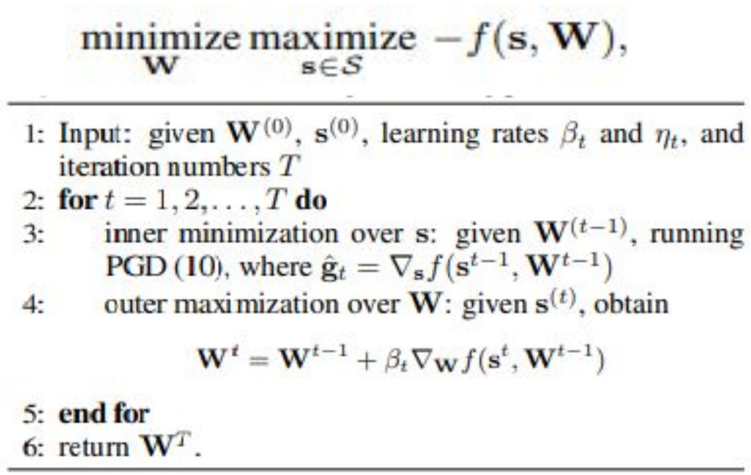
- Under the second attack model, we aim to target a retrainiable, interactive GNN. Thus, we are effectively carrying out a minimax strategy.

$$\underset{\mathbf{s}\in\mathcal{S}}{\text{minimize}}\,\underset{\mathbf{W}}{\text{maximize}}\ f(\mathbf{s},\mathbf{W}) = \sum_{i\in\mathcal{V}} f_i(\mathbf{s},\mathbf{W};\mathbf{A},\{\mathbf{x}_i\},y_i),$$

- This will require changes to our attack approach. In particular, we will use alternating optimization to solve the above problem.

Robust Training

- Any novel adversarial attack creates the opportunity for a new robust training method, and this is no exception.
- We intend to minimize classification loss, with the adversary aiming to maximize it.
    - Another minimax problem.

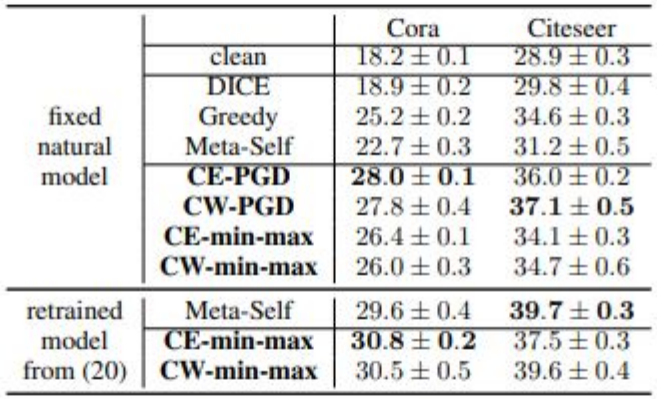


$$\underset{\mathbf{W}}{\text{minimize}}\ \underset{\mathbf{s}\in\mathcal{S}}{\text{maximize}}\ -f(\mathbf{s}, \mathbf{W}),$$

```
1: Input: given $\mathbf{W}^{(0)}$, $\mathbf{s}^{(0)}$, learning rates $\beta_t$ and $\eta_t$, and
   iteration numbers $T$
2: for $t = 1, 2, \ldots, T$ do
3:     inner minimization over s: given $\mathbf{W}^{(t-1)}$, running
       PGD (10), where $\hat{\mathbf{g}}_t = \nabla_{\mathbf{s}} f(\mathbf{s}^{t-1}, \mathbf{W}^{t-1})$
4:     outer maximization over $\mathbf{W}$: given $\mathbf{s}^{(t)}$, obtain
           $\mathbf{W}^t = \mathbf{W}^{t-1} + \beta_t \nabla_{\mathbf{W}} f(\mathbf{s}^t, \mathbf{W}^{t-1})$
5: end for
6: return $\mathbf{W}^T$.
```

Experiment: Attack Performance

- Two well-known benchmarking datasets, Cora and Citeseer, are used to evaluate the model. Both consist of unweighted adjacency matrices and sparse bag-of-words feature vectors for each node.
- Four attack methods are evaluated. For each of the two methods described, two different loss functions are tested. (CE and CW)

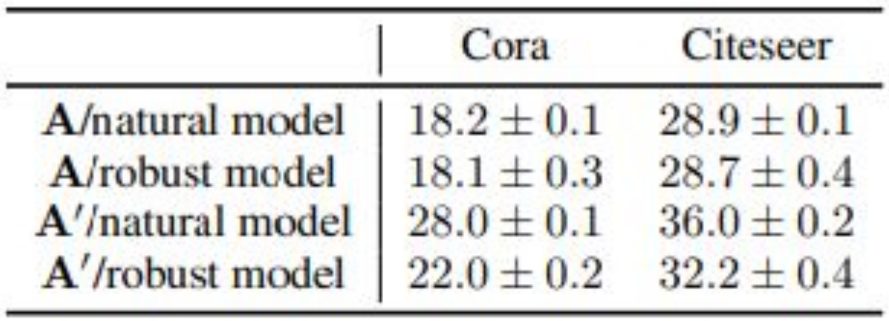


		Cora	Citeseer
fixed natural model	clean	18.2 ± 0.1	28.9 ± 0.3
	DICE	18.9 ± 0.2	29.8 ± 0.4
	Greedy	25.2 ± 0.2	34.6 ± 0.3
	Meta-Self	22.7 ± 0.3	31.2 ± 0.5
	CE-PGD	$\mathbf{28.0 \pm 0.1}$	36.0 ± 0.2
	CW-PGD	27.8 ± 0.4	$\mathbf{37.1 \pm 0.5}$
	CE-min-max	26.4 ± 0.1	34.1 ± 0.3
	CW-min-max	26.0 ± 0.3	34.7 ± 0.6
retrained model from (20)	Meta-Self	29.6 ± 0.4	$\mathbf{39.7 \pm 0.3}$
	CE-min-max	$\mathbf{30.8 \pm 0.2}$	37.5 ± 0.3
	CW-min-max	30.5 ± 0.5	39.6 ± 0.4

Table 1: Misclassification rates (%) under 5% perturbed edges

Experiment: Defense Performance

- In this section, we examine the performance of a model subjected to robust training.
- As we can see, robust training improves resilience against the adversarial attack
- As usual, however, it does not do so perfectly. Adversarial attacks still significantly increase the misclassification rate.



	Cora	Citeseer
A/natural model	18.2 ± 0.1	28.9 ± 0.1
A/robust model	18.1 ± 0.3	28.7 ± 0.4
A′/natural model	28.0 ± 0.1	36.0 ± 0.2
A′/robust model	22.0 ± 0.2	32.2 ± 0.4

Questions?