Training Robust Graph Neural Networks

Yizhou CHEN, Ting GAO, Haoxing ZHANG

tutors: Ana Neacșu & Jean-Christophe Pesque

04/06/2021

CentraleSupélec

Motivation

Introduction

Graph Neural networks (GNNs) are widely used today. It's applied in many domains like e-commerce, chemistry industry, etc.

Motivation

Introduction

Graph Neural networks (GNNs) are widely used today. It's applied in many domains like e-commerce, chemistry industry, etc.

Because of its highly complex and non-linear structures,it can be very sensitive to small perturbations of the input, which can result in a big deviation of the output prediction.

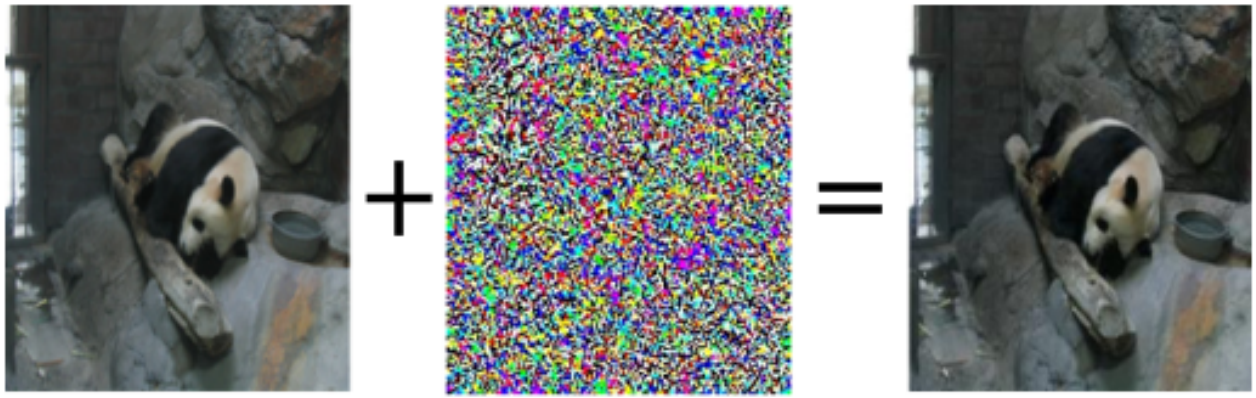


Figure: Adversarial example

GNN structure

Considering a graph with K nodes, each node has Q_i features, the Graph neural network contains m-layers. For each layer $i \in \{1, \cdots, m\}$, for each node v, if we note $y_v^{(i-1)}$ as its input, $y_v^{(i)}$ as its output and R_i an activation function, each neuron models the following relation:

$$y_v^{(i)} = R_i \left(\sum_{u \in \mathcal{N}\{v\} \cup \{v\}} w_{v,u}^{(i)} y_u^{(i-1)} + b_v^{(i)} \right) \tag{1}$$

If we stack all the vectors $(y_v^{(i)})_{1 \leq v \leq K}$ as $y^{(i)} \in \mathbb{R}^{KQ_i}$, $(b_v^{(i)})_{1 \leq v \leq K}$ as $b_i \in \mathbb{R}^{KQ_i}$, we have :

$$y^{(i)} = R_i(W_i y^{(i-1)} + b_i) \tag{2}$$

where $W_i = (A^T + I_d) \otimes w_{v,u}^i$ and $\otimes$ denotes the Kronecker product.

GNN structure

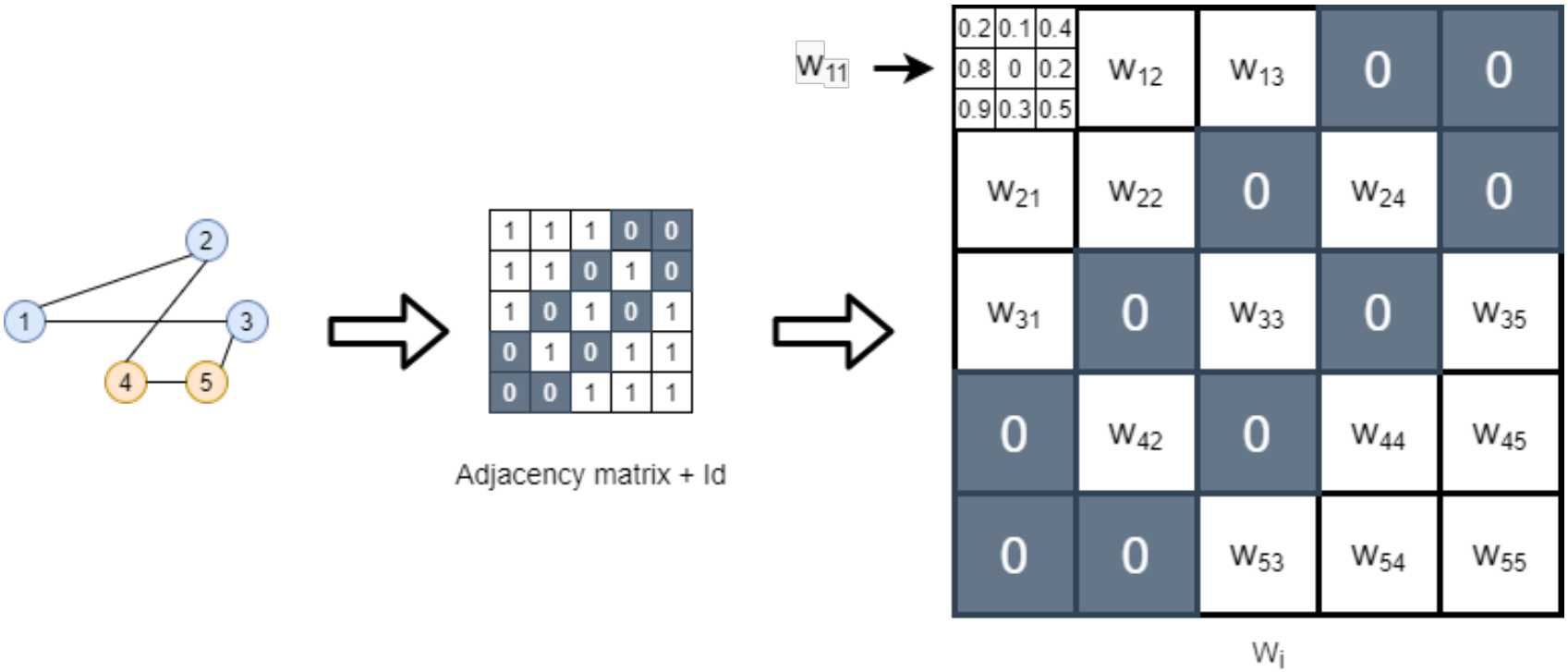


Figure: Visualisation of W, $w_{v,u}$ for a graph with 5 nodes

The zero components of this matrix can be identified by computing

$$\Gamma_i = (A^T + I_d) \otimes J_i \tag{3}$$

where $J_i \in \mathbb{R}^{Q_i \times Q_{i-1}}$ is an all-ones matrix.

Quantifying the robustness

The forward pass though a m-layer fully connected neural network can be viewed as a composition of functions $T = T_m \circ \cdots \circ T_0$. Each operator T_i models the input $x_i \in \mathbb{R}^{Q_{i-1}}$ as follows:

$$(\forall i \in 1 \ldots m) \quad T(x_i) = R_i(W_i x_i + b_i) \tag{4}$$

Assessment of robustness

If the input $x \in \mathbb{R}^{M_0}$ is perturbed by a small noise $z \in \mathbb{R}^{M_0}$, the effect can be quantified by the following inequality:

$$\| T(x + z) - T(x) \| \leq \theta \|z\|, \tag{5}$$

where $\|\cdot\|$ is the Euclidean norm and θ represents the Lipschitz constant of the system.

Quantifying the robustness

Theory support

- If activations are nonexpansive, loose upper bound on the Lipschitz constant:

$$\theta = \prod_{i=1}^{m} || W_i ||$$

- If additionally all the weights $(W_i)_{1 \leq i \leq m}$ are nonnegative, then $\| \prod_{i=1}^{m} W_i \|$ is the smallest Lipschitz constant of the NN. i.e.

$$\vartheta = \| W_m \cdots W_1 \|_s \tag{6}$$

Imposed constraint

For a network with m layers and for each layer $i, 1 \leq i \leq m$, its weight matrix is noted as W_i and its bias vector is noted as b_i.$\overline{\vartheta}$ is desired maximum Lipschitz.

- Non-negative weight matrix :

$$\mathcal{D}_i = \{W_i \in \mathbb{R}^{KQ_i \times KQ_{i-1}} | W_i > 0\} \tag{7}$$

- Graph topological structure :

$$\Omega_i = \{W_i \in \mathbb{R}^{KQ_i \times KQ_{i-1}} \mid (W_i)_{(u,v)} = 0 \text{ if } (\Gamma_i)_{(u,v)} = 0\} \tag{8}$$

- Norm constraint :

$$\|W_m, \cdots, W_1\|_s \leq \overline{\vartheta} \tag{9}$$

Imposed constraint

- Norm constraint : At iteration t, for each layer $i \in \{1, \cdots, m\}$, $W_{i,t}$ the estimated weight matrix at iteration t, then the above constraint becomes :

$$\mathcal{C}_{i,t} = \{W_i \in \mathbb{R}^{KQ_i \times KQ_{i-1}} | \|A_{i,t} W_i B_{i,t}\|_s \leq \overline{\vartheta}\} \tag{10}$$

with

$$A_{i,t} = W_{m,t} \cdots W_{i+1,t} \tag{11}$$

$$B_{i,t} = W_{i-1,t+1} \cdots W_{1,t+1} \tag{12}$$

for the first layer $i = 1$, $B_{i,t}$ is an identity matrix of dimension KM_1, for the last layer $i = m$, $A_{i,t}$ is the identity matrix of KM_m.

Imposed contraint

The problem is thus reformulated by the projection

$$\widehat{W}_{i,t+1} = P_{\mathcal{C}_{i,t+1} \cap \Omega_i \cap \mathcal{D}_i}(\overline{W}_{i,t+1}) \tag{13}$$

where $\overline{W}_{i,t+1}$ is the estimated matrix after stochastic gradient optimization.
If we note $\xi_i \in \mathbb{R}^{KQ_i \times (KQ_{i-1}+1)}$ combining weight matrix W_i and bias vector b_i. Note $\mathcal{S}$ is a closed set expressing the above three constraints, $\mathcal{S}_{i,t}$ is the constraint imposed at iteration t for layer i. Then :

$$\mathcal{S}_{i,t} = \{\xi_i | [(\xi_{j,t+1}^T)_{j<i} \xi_i^T (\xi_{j,t}^T)_{j>i}]^T \in \mathcal{S}\} \tag{14}$$

ADAM-based algorithm

We adopt an optimization method based on ADAM method where $\overline{W}_{i,t+1}$ can be calculated as follows :

$$\overline{W}_{i,t+1} = W_{i,t} - \gamma_t \mu_{i,t} / (\sqrt{\nu_{i,t}} + \epsilon) \tag{15}$$

$\mu_{i,t}$ denotes biased first moment estimate, $\nu_{i,t}$ denotes biased second raw moment estimate and γ_t is the learning rate.

Algorithm 1: ADAM-based algorithm

Partition N graphs into mini-batches $(\mathbb{M}_{l,h})_{1 \leq l \leq L}$
for *every* $l \in \{1, \cdots, L\}$ **do**
  $t = (h-1)L + l$
  for *every* $i \in \{1, \cdots, m\}$ **do**
    $g_{i,t} = \sum_{n \in \mathbb{M}_{l,h}} \nabla_i l(z_n, (\xi_{i,t})_{1 \leq i \leq m})$
    $\mu_{i,t} = \beta_1 \mu_{i,t-1} + (1-\beta_1) g_{i,t}$
    $\nu_{i,t} = \beta_2 \mu_{i,t-1} + (1-\beta_2) g_{i,t}^2$
    $\gamma_t = \gamma \sqrt{1-\beta_2^t}/(1-\beta_1^t)$
    $\xi_{i,t+1} = P_{\mathcal{S}_{i,t}}(\xi_{i,t} - \gamma_t \mu_{i,t} / (\sqrt{\nu_{i,t}} + \epsilon))$
  end
end

FISTA accelerated version of DFB algorithm

We adopt the FISTA accelerated version of DFB algorithm to calculate the projection on $\mathcal{S}_{i,t}$:

Algorithm 2: FISTA accelerated version of DFB algorithm

Let $Y_0 \in \mathbb{R}^{KQ_m \times KQ_0}$
Set $\gamma = 1/(\|\mathcal{L}\|)^2$
Set $\alpha \in]2, +\infty[$ **for** $l = 0, 1, \cdots$ **do**

$$\eta_l = \frac{l}{l+1+\alpha}$$
$$Z_l = Y_l + \eta_l(Y_l - Y_{l-1})$$
$$V_l = P_{\mathcal{D}_i \cap \Omega_i}(\overline{W}_i - A_{i,n}^T Z_l B_{i,n}^T)$$
$$\widetilde{Y}_l = Z_l + \gamma A_{i,n} V_l B_{i,n}$$
$$Y_{l+1} = \widetilde{Y}_l - \gamma P_{\mathcal{B}(0,\overline{\vartheta})}(\gamma^{-1}\widetilde{Y}_l)$$

end

FISTA accelerated version of DFB algorithm

The projection of $P_{\mathcal{D}_i \cap \Omega_i}$ can be deduced by $P_{\mathcal{D}_i} \circ P_{\Omega_i}$.

- Projection on $\mathcal{D}_i$:

$$P_{\mathcal{D}_i}(W_i) = ((\widetilde{W}_i)_{a,b})_{1 \leq a \leq KQ_i, 1 \leq b \leq KQ_{i-1}} \tag{16}$$

where $\forall a, b$ such that $1 \leq a \leq KQ_i, 1 \leq b \leq KQ_{i-1}$,

$$(\widetilde{W}_i)_{a,b} = \begin{cases} (W_i)_{a,b} & \text{if} \quad (W_i)_{a,b} \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

- Projection on Ω_i :

$$P_{\Omega_i}(W_i) = \Gamma \odot W_i \tag{18}$$

where Γ refers to equation (3), $\odot$ represents element wise product.

FISTA accelerated version of DFB algorithm

- Projection on $\mathcal{C}_{i,t}$:
  $P_{\mathcal{B}(0,\vartheta)}(Y_l)$ is accessed by decomposing its singular value. For $Y_l \in \mathbb{R}^{KM_m \times KM_0}$, Y_l can be writen as $Y_l = U_l \Lambda_l V_l^T$ where $U_l \in \mathbb{R}^{KM_m \times r}$, $V_l \in \mathbb{R}^{KM_0 \times r}$. U_l, V_l are both semi-unitary matrix, Λ_l is square diagonal of size $r \times r$: $\Lambda_l = \text{Diag}(\lambda_0, \cdots, \lambda_r)$, where $r \leq \min\{M_m, M_0\}$. The projection is obtained by formulating :

$$P_{\mathcal{B}(0,\vartheta)}(Y_l) = U_l \widetilde{\Lambda}_l V_l \tag{19}$$

where $\widetilde{\Lambda}_l = \text{Diag}(\widetilde{\lambda}_1, \cdots, \widetilde{\lambda}_r)$ and $\forall i \in \{1, \cdots, r\}$,

$$\widetilde{\lambda}_i = \begin{cases} \lambda_i & \text{if} \quad \lambda_i \leq \overline{\vartheta} \\ \overline{\vartheta} & \text{otherwise} \end{cases} \tag{20}$$

Dataset description

The model has been evaluated for artificial data.

- Real datasets are too large.
- The graphs of real data sets usually don't have the same topology.
- Easier to control the complexity of artificial data

Artificial dataset

N graphs with the same topological structure;

K nodes;

M features for each node;

C classes;

Specifications:

- All nodes of the same class are connected to each other.
- Features are generated from Normal distributions $\mathcal{N}(\mu, \Sigma)$.

Dataset description

We assume that the covariance matrix Σ for each class is the same and can be formulated as Equation (21) shows.

$$\Sigma = \begin{bmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma^2 \end{bmatrix} \tag{21}$$

Measure the complexity of dataset:

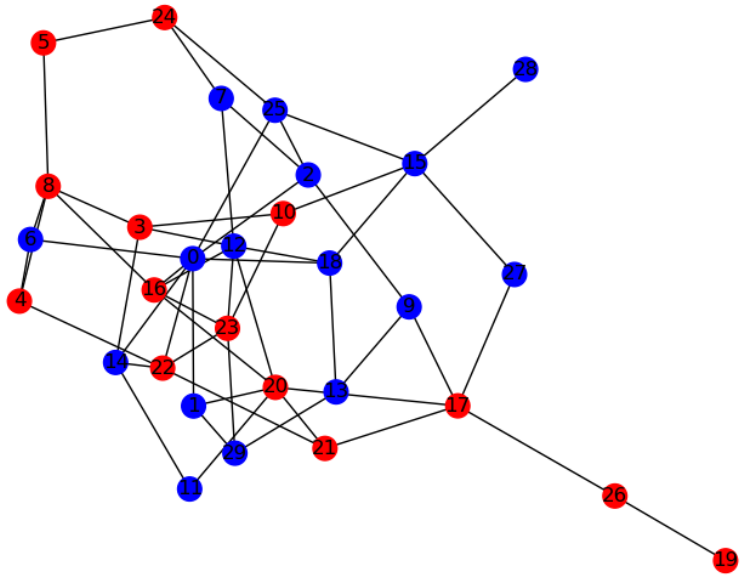
$$\eta = \frac{\sum_{0 \leq i < j \leq C} ||\mu_i - \mu_j||_2}{\sigma} \tag{22}$$

- σ : standard deviation
- μ_i : mean vector for the class i

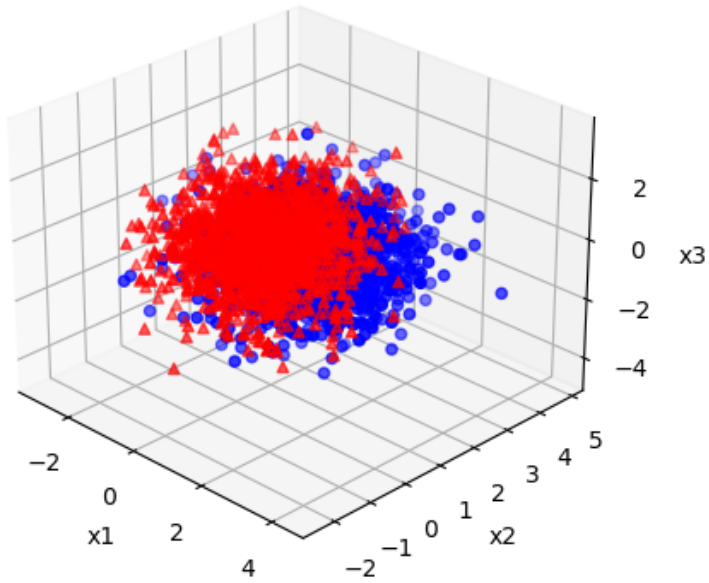
The smaller η is, the more overlapping we get and the more complex the dataset is

Dataset description

Examples



(a) graph structure



(b) Feature distribution

Figure: $N = 1000$, $K = 30$, $M = 3$, and $C = 2$, Here, we put $\sigma = 1$, $\mu_0 = [0.1, 0.8, -0.1]$ for red nodes, and $\mu_1 = [0.7, 1.5, -0.6]$ for blue nodes.

Algorithm of data generator

Notation:

- $\mathcal{R}(j)$: root node for class j $\quad (1 \leq j \leq C)$
- $\mathcal{F}(j)$: frontier for class j $\quad (1 \leq j \leq C)$
- node(j) : nodes assigned to class j $\quad (1 \leq j \leq C)$
- feature(n) : M features of the node n $\quad (1 \leq n \leq K)$
- neighbor(n) : the set of unlabelled neighbors of the node n $(1 \leq n \leq K)$

Algorithm of data generator

Algorithm 3: Generate data set with N graphs, M features and C classes

Define adjacency matrix A_d
for $i = \{1, \cdots, N\}$ **do**
  for $j = \{1, \cdots, C\}$ **do**
    Set $\mathcal{R}(j)$
    $\mathcal{F}(j)$ **add** $\mathcal{R}(j)$
  end
  while $\sum(size\{node\}) < \dim(A_d)$ **do**
    for $j = \{1, \cdots, C\}$ **do**
      $\mathcal{R}(j) \leftarrow random(\mathcal{F}(j))$
      $V \leftarrow random(neighbor(\mathcal{R}(j)))$
      $\mathcal{F}(j)$ **remove** $\mathcal{R}(j)$
      $node(j)$ **add** V
      $\mathcal{F}(j)$ **add** V
    end
  end
  for $j = \{1, \cdots, C\}$ **do**
    Set μ, Σ
    for n *in* $node(j)$ **do**
      $feature(n) \leftarrow \mathcal{N}(\mu, \Sigma)$
    end
  end
end

Model structure

Input: $X_0 = [x_1, x_2, \cdots, x_K] \in \mathbb{R}^{M \times K}$

Reshape layer: $X_0 \to Y_0 = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix} \in \mathbb{R}^{MK}$

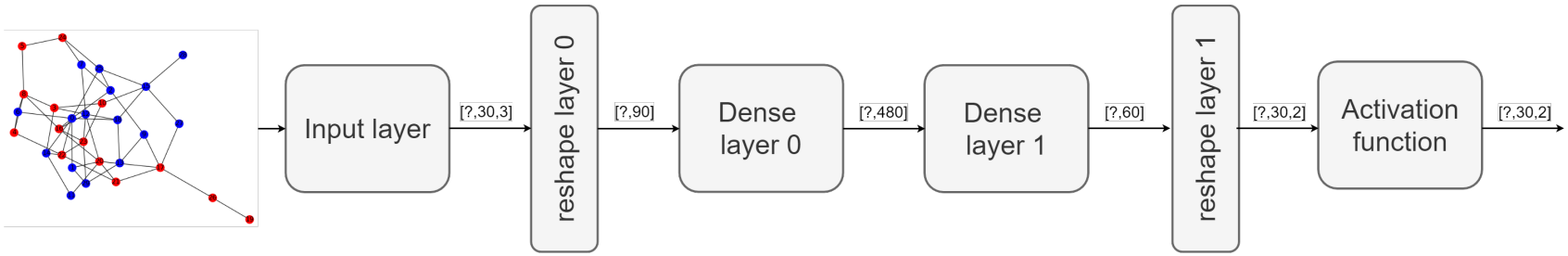
Dense layer i : $Y_i = \phi(W_i Y_{i-1} + b_i) \quad i = 0, 1$
($W_i \in \mathbb{R}^{Q_i K \times Q_{i-1} K}$ satisfies all the 3 constraints, i.e. constraint of non-negativity, constraint of graph topological structure, and constraint of Lipschitz constant)

Reshape layer: $Y_m = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} \in \mathbb{R}^{CK} \to [y_1, y_2, \cdots, y_K] \in \mathbb{R}^{C \times K}$

Soft-max layer

Model structure

Example



(a)

Figure: Model structure example. In this model, there are 2 dense layers, $K = 30$ nodes in each graph, $M = 3$ features for each node, and the output is the classification of each node in $C = 2$ classes

Training techniques

ADAM based optimizer

- Adaptive learning rates for each parameter
- Faster convergence speed

Validation set

- Draw one-tenth of the samples from the training set for validation
- Tune hyper-parameters such as batch size, the number of neurons ...

Three applied methods

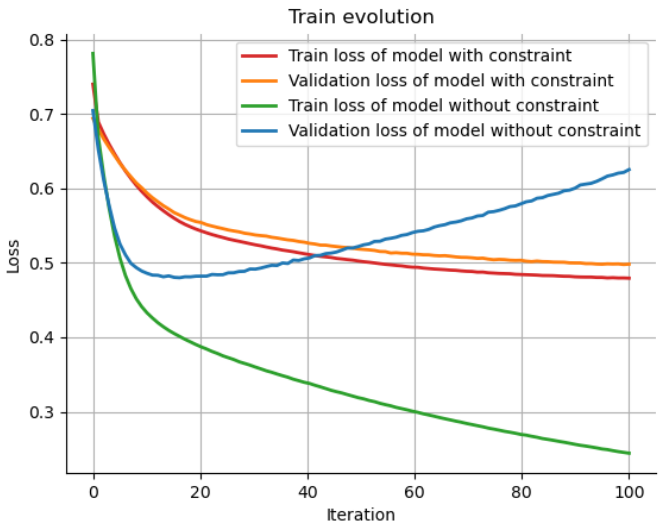
- Tensorboard visualization: visualize the training process
- Early stopping method: stop the training when the validation loss has stopped decreasing with a patience of epoch
- Function to impose Lipschitz constant constraint:
  - Called on each batch end
  - Make sure that each weight matrix is correctly projected to the constraint set.

Prevention of overfitting

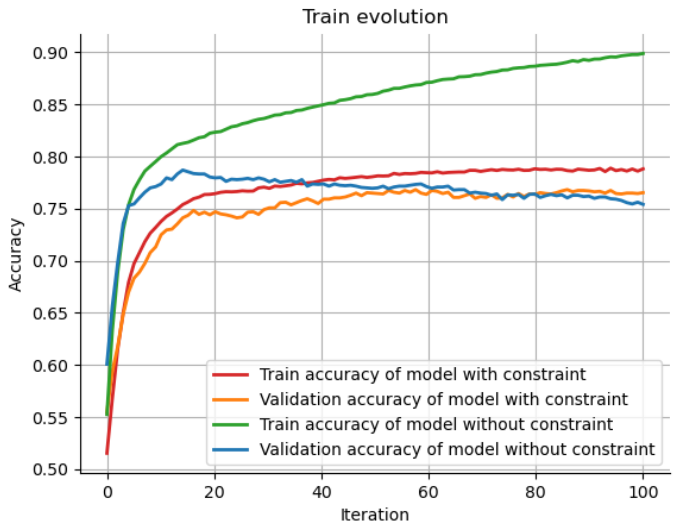
- Node classification (binary)
  - Train evolution: evolution of loss and accuracy on train and validation set
  - Performance on artificial datasets of different complexity
- Node classification (multi-class)
  - Performance of baseline models
  - Performance on artificial datasets of different complexity

Prevention of overfitting: Node classification (binary)

Train evolution



(a) Loss

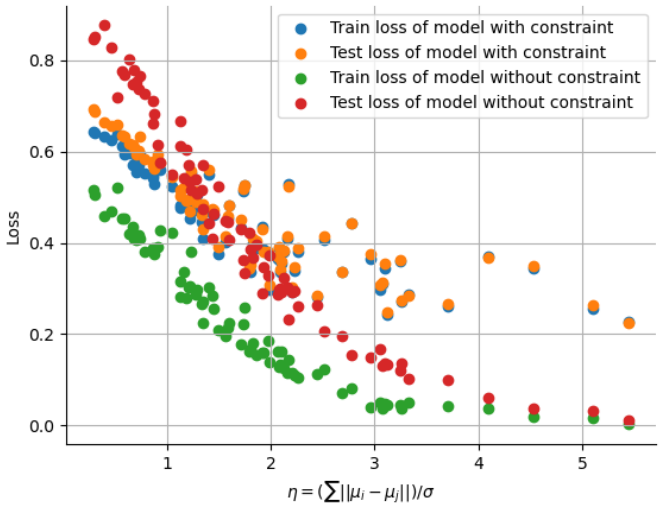


(b) Accuracy

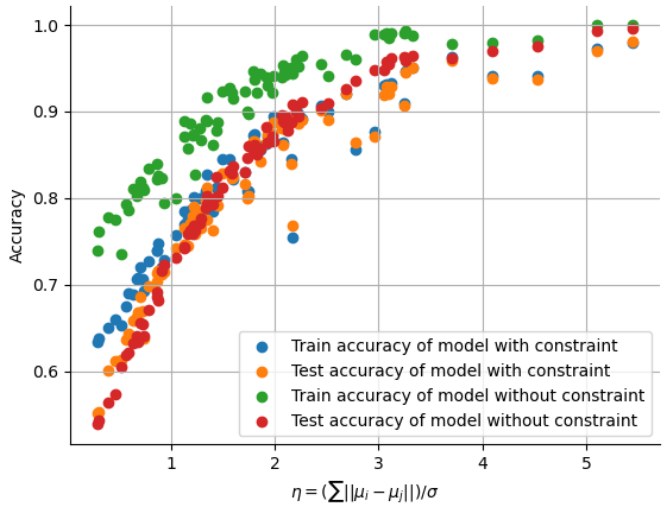
Figure: Train evolution of example 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 2]$ on the dataset where the sets of nodes of the same label for each graph are different, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 1$. Overfitting is observed for model without Lipschitz constant constraint.

Prevention of overfitting: Node classification (binary)

Performance on artificial datasets of different complexity



(a) Loss



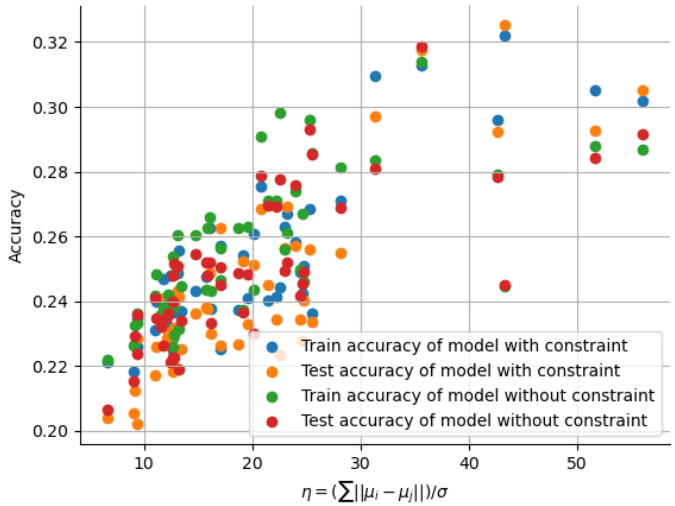
(b) Accuracy

Figure: Performance of example 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 2]$ on datasets of different complexity with $N = 1000$, $C = 2$, $M = 3$, and $K = 30$, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 1$.

Prevention of overfitting: Node classification (multi-class)

Performance of baseline model (all independent nodes)

(a) Loss

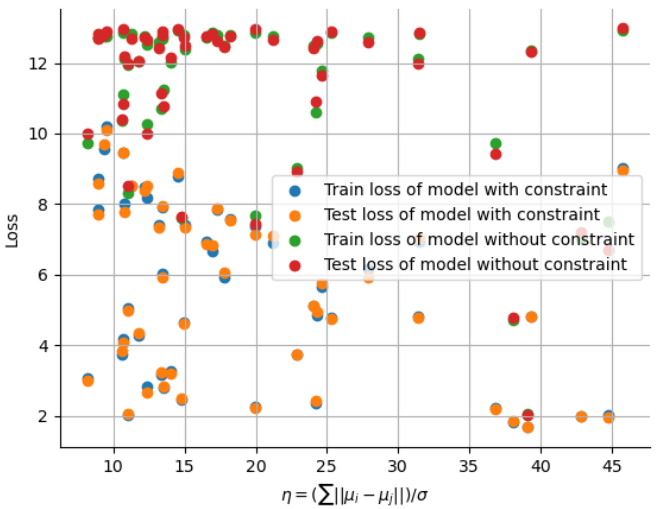


(b) Accuracy

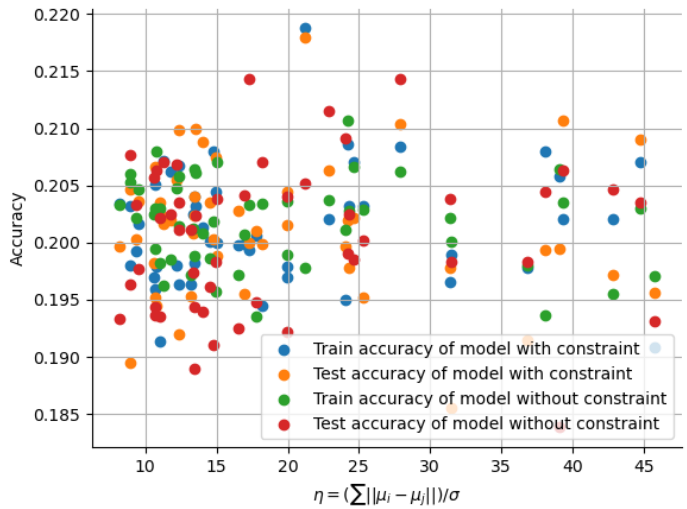
Figure: Performance of baseline 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 5]$, on datasets of different complexity where every node of each graph is independent ($A_d = 0$), $N = 1000$, $C = 5$, $M = 3$, and $K = 30$, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 5$.

Prevention of overfitting: Node classification (multi-class)

Performance of baseline model (fully connected graph)



(a) Loss

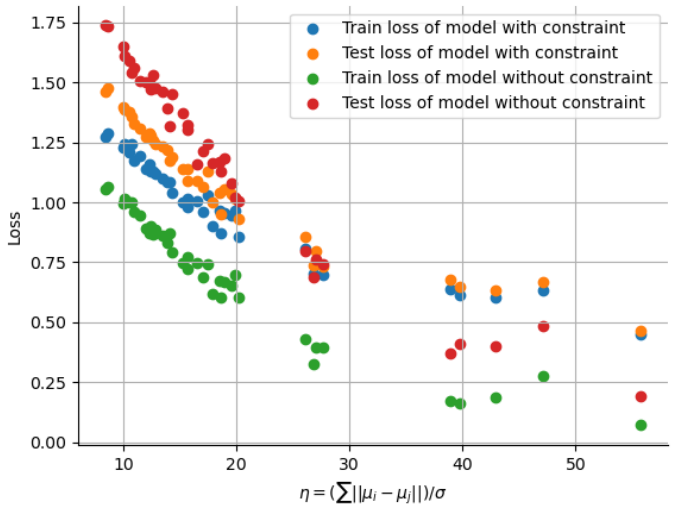


(b) Accuracy

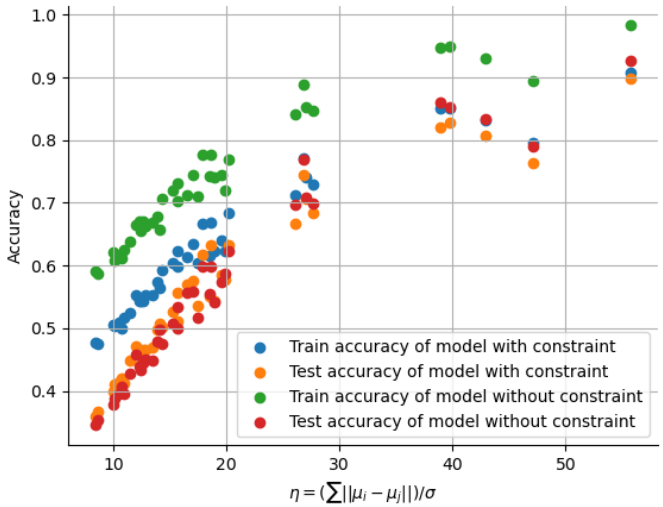
Figure: Performance of baseline 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 5]$, on datasets of different complexity where each graph is fully connected ($A_d + I_d = J_d$), $N = 1000$, $C = 5$, $M = 3$, and $K = 30$, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 5$.

Prevention of overfitting: Node classification (multi-class)

Performance of our proposed model



(a) Loss



(b) Accuracy

Figure: Performance of example 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 5]$ on data sets of different complexity, $N = 1000$, $C = 5$, $M = 3$, and $K = 30$, all nodes of each graph are not independent or fully connected and, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 5$.

Prevention of overfitting

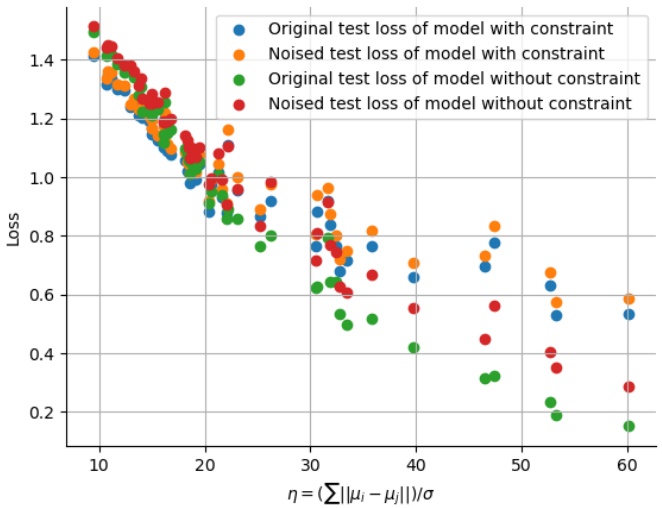
Brief summary

- By comparing with baseline models, GNN models can learn the information from graph topology.
- Imposing Lipschitz constant constraint in GNN training can largely reduce its overfitting.
- In node classification, When data is rather simple (the node feature distributions of different labels are quite different), GNN model with Lipschitz constant constraint performs almost as good as the model without the constraint; when data is rather complicated (the node feature distributions of different labels are quite similar), GNN model with Lipschitz constant constraint can even perform better.

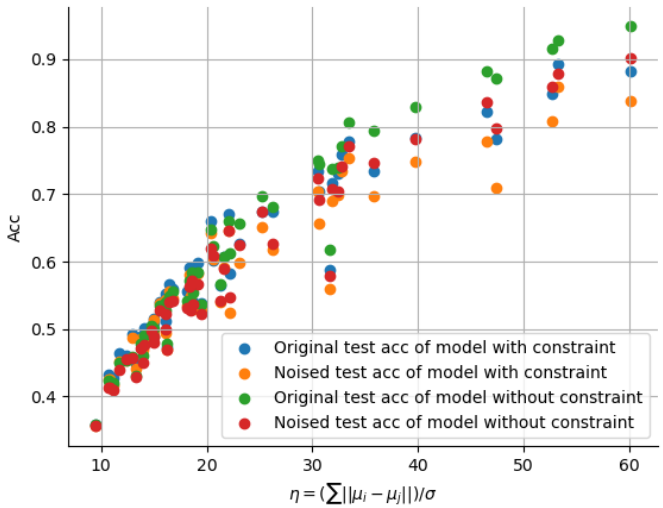
Robustness assessment

- Test with noisy input data
- Test with graph data of modified topological structure
- Test with adversarial samples
  - Samples generated from Projected Gradient Descent Attack
  - Samples generated from DeepFool

Test with noisy input data



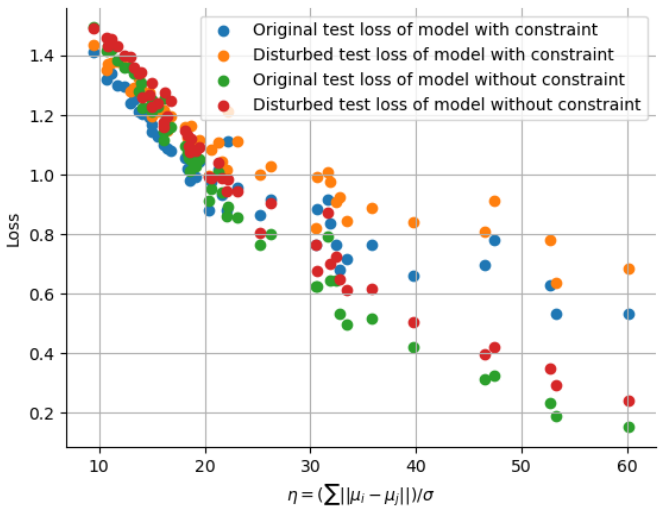
(a) Loss



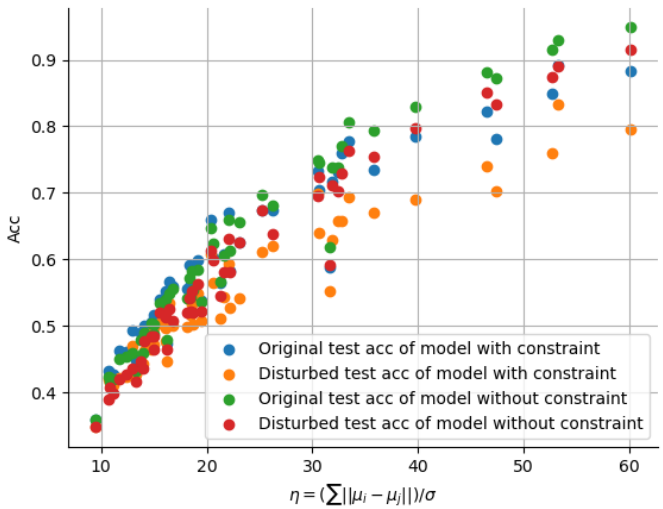
(b) Accuracy

Figure: Performance of example 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 5]$ on noisy test data of different complexity with $N = 1000$, $C = 5$, $M = 3$, and $K = 30$, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 5$.

Test with graph data of modified topological structure



(a) Loss



(b) Accuracy

Figure: Performance of example 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 5]$ on perturbed test data (A_d is modified) of different complexity with $N = 1000$, $C = 5$, $M = 3$, and $K = 30$, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 5$.

Test with adversarial samples from Projected Gradient Descent Attack

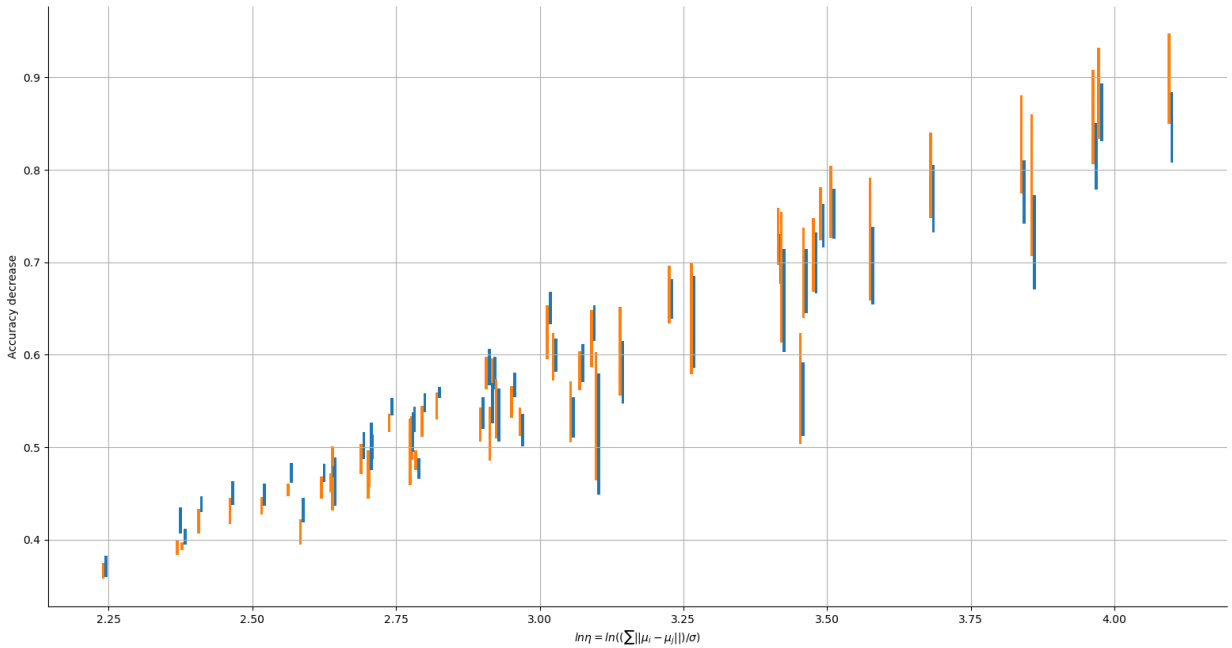


Figure: Accuracy decrease of the 2-dense-layer models with and without Lipschitz constant constraint on adversarial test data generated by Projected Gradient Descent Attack, blue bar for the model with constraint and orange bar for the model without constraint. For each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 5$.

Test with adversarial samples from DeepFool

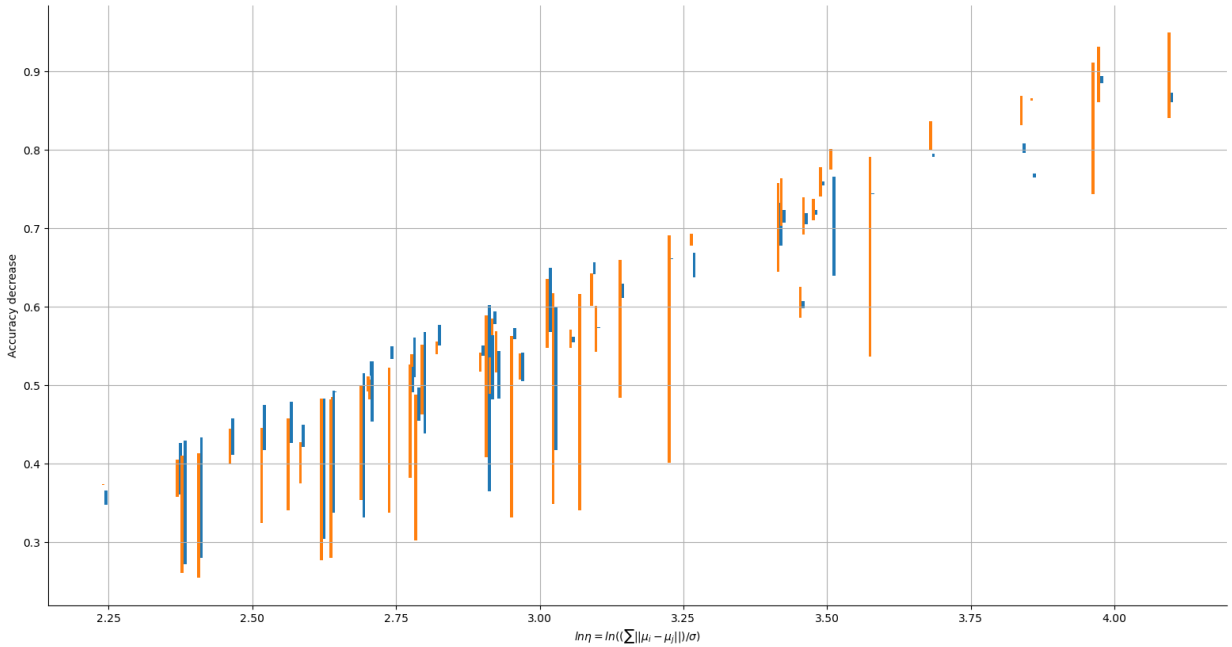


Figure: Accuracy decrease of the 2-dense-layer models with and without Lipschitz constant constraint on adversarial test data generated by DeepFool, blue bar for the model with constraint and orange bar for the model without constraint. For each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 5$.

Robustness assessment

Brief summary

- GNN model with Lipschitz constant constraint has a better robustness against noisy input data than the model without the constraint does.
- The robustness of model with and without the constraint is almost the same against the graph data whose topological structure is modified.
- GNN model with Lipschitz constant constraint has a better robustness against adversarial samples than the model without the constraint does.

Conclusion

- By imposing constraint on Lipschitz constant of the GNN model, we largely reduce the overfitting during the training process and improve its robustness against noisy inputs and adversarial inputs.
- For datasets with bigger graphs and more node features, the complexity of GNN and the number of dense layers should increase to have a good performance.
- Our proposed method focuses on positive weights and can be only applied to the graph datasets where each graph has the same topological structure, but it is difficult to find a real dataset that satisfies the requirements.

Links

- Our Github: https://github.com/SJTUzhou/Lipschitz_gnn
- DeepFool: https://arxiv.org/abs/1511.04599
- Projected Gradient Descent Attack: https://arxiv.org/abs/1706.06083