Training Robust Graph Neural Network by Applying Lipschitz Constant Constraint

Yizhou Chen,* Ting Gao,† Haoxing Zhang,‡ Ana Neacsu,§ and Jean-Christophe Pesquet¶
CentraleSupélec
(Dated: June 12, 2021)

Abstract. Graph Neural Networks (GNNs) show strong ability in learning information from graph data but it can be sensitive to noisy and adversarial inputs. In this work, we propose a training method to ensure a robust GNN model against noise and adversarial samples. Some previous related works have been done on the improvement of robustness of neural networks in the field of graph but mostly on the graph convolutional network (GCN). We propose an approach that is controlling Lipschitz constant of a traditional GNN network to improve its robustness by giving the mathematical induction and some experimental results obtained from artificially generated graph datasets with node classification task. From the aspects of prevention of overfitting, resistance to noisy data, to graph data with modified topology, and to adversarial inputs, we compare and discuss the performances of the models with and without Lipschitz constant constraint. The results show that GNN model with this constraint has a better robustness against noisy inputs and adversarial inputs, compared with the GNN model without this constraint.

I. INTRODUCTION

In recent years, the strong ability of Graph Neural Networks (GNNs) in modeling the relationship between graph nodes has made it achieve various breakthroughs in fields related to graph analysis. For example, in the field of e-commerce, GNNs can use the interaction between users and products to achieve highly accurate recommendations. In the field of chemistry, molecules are modeled as graphs, and GNNs can be used to predict their biological activity. Despite their massive success on Non-Euclidean data, GNNs, like other machine learning based systems, look like black boxex whose robustness against adversarial attacks is poorly controlled.

As GNN has 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. The Lipschitz constant is a well-known metric used in the literature to evaluate the robustness of neural networks to small disturbances. This constant provides an upper bound on the ratio between the deviation of output and input variations for a given distance.

In this work, we propose a learning scheme that controls the Lipschitz constant of GNN, by training the system under spectral norm constraints. The network requires positive weights, for which we can simplify the calculation of the Lipschitz constant.

The paper is organized as follows. In section II, we present a literature review and the theoretical framework of our work. We describe in section III how we impose the Lipschitz constant constraints and how we optimize our network. Section IV describes our dataset and shows experimental results. Section V is dedicated to concluding remarks.

II. BACKGROUND

A. Related work

Much work has been done in the area of neural networks (NN), and GNNs have recently attracted attention due to its splendid performance in handling problems such as node classification and connection prediction. However, with the development of high performance GNNs, the robustness problem arises.

Zügner et al. raised the issue of adversarial attacks on Graph Convolutional Network (GCN). In this paper, attacks on a certain node of the graph are studied. That is, for a node on a graph, by imposing some perturbations on the graph structure (structure attacks) or node attributes (feature attack), it is possible to make the wrongly classification of nodes. An algorithm *Nettack* is proposed to generate limited attacks.

Dai et al. only consider modifying edges in a graph as attack, three methods are proposed for different settings. The attack method *RL-S2V* learns the generalizable attack policy over the graph structure. *GradArgma* is applicable for white box attack by doing a greedy selection based on gradient information. The third method is *GeneticAlg*, which is the case where an evolutionary algorithm is used to deal with black box attacks and can obtain the classifier confidence. In addition, the paper points out that the defense of graph networks may be simpler than images, and that the effect of the attack algorithm can be reduced by dropping edges randomly during training.

Wu et al. attack graphs by measuring the importance of nodes and edges using integrated gradient, and then selecting the most influential features or edges for perturbation. In this work, a defense method is also proposed. Since the attack works better when edges con-

* yizhou.chen@student-cs.fr
† ting.gao@student-cs.fr
‡ haoxing.zhang@students-cs.fr
§ ana.neacsu@centralesupelec.fr
¶ jean-christophe.pesquet@centralesupelec.fr

necting nodes with low feature similarity are added, it is more efficient to remove edges connecting low feature similarity nodes in the preprocessing stage, which will not significantly reduce the prediction accuracy while making the graph network more robust.

Bojchevski and Günnemann bring the first study on attacking node embedding, which is to make the embedding generated by the new graph less effective by modifying a finite number of edges. It mainly exploits a property of random walk-based algorithm in spectral graph to generate embedding: the embedding obtained by Deepwalk is implicitly represented by the singular values or vectors obtained after the SVD decomposition of the Pointwise Mutual Information matrix, thus the initial bi-level optimization problem is transformed into a problem of finding the singular values of the co-occurrence matrix.

Zügner and Günnemann propose to modify the graph structure with Meta-gradient back propagation, adjusting the adjacency matrix of the graph as a hyperparametric problem. The attack scenario is a global attack on node classification under the unknown target classifier condition, and using a two-layer GCN as the surrogate model with an approximate speedup for the computation of the Meta-gradient.

There are also several solutions brought forward to deal with the adversarial attacks on GCN. Zhu et al. improve robustness of GCN by adopting Gaussian-based Graph Convolution Layers and attention mechanism. The basic idea is to replace the vector features of each node in the hidden layer with a Gaussian distribution, which can be represented by two parameters: the mean and variance of distribution, because of the independent additivity of Gaussian distribution, these two parameters can propagate aggregation on the graph. Moreover, they adjust the weights of the node's features based on the size of its variance, and the larger the variance of the node, the smaller its weight when passing messages. Experiments confirm that such a graph convolution layer is more robust.

Entezari et al. further find out that *Nettack* model developed by Das et al. is a high-rank attack. Based on this observation, this paper proposes an algorithm named *vaccinated GCN* which uses Low-Rank Approximation (LRA) on graph data to filter the noise from adversarial attacks in the graph. Experiments show that this method perform well against high-rank attacks *Nettack*, but less effectively against low-rank attacks.

However, to our best knowledge, there is few working dedicating to the study of robustness on GNN. Jin et al. propose a general framework *Pro-GNN* which learns a structural graph and a robust graph neural network model from the perturbed graph. Based on the fact that real-world graphs share some intrinsic properties like low-rank sparse graphs and similarity of features of two adjacent nodes, the loss function is designed and the optimal solution of the problem is solved by the Forward-Backward method. Therefore, this method is more suitable for the case where the noise severely affects the topology of the network and without perturbing the node properties of the network.

In this work we propose a new method of training a GNN, taking into account the topology of the input graph. Moreover, we control the robustness of the network using carefully crafted constraint sets, inspired by the work of Neacsu et al..

B. Nomenclature

In this work, we define our symbols as presented in table I.

Symbol	Definition
G	Graph
V	Node
E	Edge
K	Number of nodes in one graph
A_d	Adjacency matrix
W_i	Weight matrix of layer i
b_i	bias of layer i
ξ	Trained parameters
θ	Lipschitz constant
m	layer numbers of netwrok
$\mathcal{N}(v)$	Neighbors of vertice v
N	Number of graphs in a dataset
M	Number of features per node in graph
C	Number of classes in classification task
Q_i	Number of neurons per node in dense layer
η	Overlapping Measurement
μ	Mean value of Gaussian distribution
σ	Standard deviation of Gaussian Distribution
$\mathcal{N}(\mu, \sigma)$	Gaussian Distribution

Table I: Nomenclature

C. Quantifying the robustness of GNNs

Let $G = (V, E)$ be a graph composed of K vertices (nodes). We denote $V = \{1, \cdots, K\}$ the set of its vertices and E be the set of its edges. Each node $v \in V$ is associated with a feature vector $x_v \in \mathbb{R}^M$ where M is the dimension of features. For each vertex we define its neighbourhood $\mathcal{N}(v)$, as the set or nodes connected with it. The adjacency matrix $A_d = (A_{u,v})_{1 \leq u,v \leq K} \in \mathbf{R}^{K \times K}$ that describes the topology of the graph is defined as follows.

$$A_{u,v} = \begin{cases} 1 & \text{if} \quad u \in \mathcal{N}(v) \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Consider a m-layers Graph Neural Network. For each layer $i \in \{1, \cdots, m\}$, 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:

$$y_v^{(i)} = R_i \left(\sum_{u \in \mathcal{N}\{v\} \cup \{v\}} w_{v,u}^{(i)} y_u^{(i-1)} + b_v^{(i)} \right) \quad (2)$$

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 :

$$y^{(i)} = R_i(W_i y^{(i-1)} + b_i), \quad (3)$$

where $W_i = (A^T + I_d) \otimes w_{v,u}^i$ and $\otimes$ denotes the Kronecker product. Figure 1 gives a direct visualisation of W_i and $w_{v,u}$

Although the weight matrix W_i may be large, it is usually very sparse. The zero components of this matrix can be identified by computing

$$\Gamma_i = (A^T + I_d) \otimes J_i \quad (4)$$

where $J_i \in \mathbb{R}^{Q_i \times Q_{i-1}}$ is an all-ones matrix. Indeed, zero elements of Γ_i are located at the same positions as the elements of W_i which need to be set to zero.

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:

$$(\forall i \in 1 \dots m) \quad T(x_i) = R_i(W_i x_i + b_i) \quad (5)$$

If the input $x \in \mathbb{R}^{Q_0}$ is perturbed by a small noise $z \in \mathbb{R}^{Q_0}$, the effect can be quantified by the following inequality:

$$\|T(x+z) - T(x)\| \leq \theta \|z\|, \quad (6)$$

where $\|\cdot\|$ is the Euclidean norm and θ represents the Lipschitz constant of the system. Note that the Lipschitz constant is very important parameter since it upperbounds the effect of the perturbation and it can be viewed as a way to assess and control the sensitivity of the network against adversarial attacks.

For each layer $i \in \{1, \cdots, m\}$, if we note the output $y^{(i+1)}$ as a function of its input $y^{(i)}$ and trained parameters, then:

$$y^{(i+1)} = \phi_i(y^{(i)}, W_i, b_i). \quad (7)$$

Even for a shallow network, computing the accurate Lipschitz constant is a NP problem. Combettes and Pesquet show that for a network with non-expansive activation operators, the Lipschitz constant of the network can be accessed by the weight matrix of each layer, which means that the impact of activation operators can be ignored when considering the Lipschitz constant.

Szegedy et al. point out in their work that if θ_i denotes the Lipschitz constant of each layer, which means that

$$\phi_i(y^{(i)} + z^{(i)}, W_i) - \phi_i(y^{(i)}, W_i) \leq \theta_i \|z^{(i)}\|. \quad (8)$$

Then the Lipschitz constant of the network T can be deduced as

$$\theta = \prod_{i=1}^{m} \theta_i. \quad (9)$$

Moreover, for neural network, a standard approximation of the Lipschitz constant is given by

$$\theta = \prod_{i=1}^{m} \|W_i\|_s, \quad (10)$$

where $\|\cdot\|_s$ denotes the *spectral norm*, that can be computed as the maximum singular value. This bound is often over pessimistic and other solutions to derive better approximations have been proposed in literature.

In the work conducted by Combettes and Pesquet, neural networks with non-negative weight constraints are further discussed and a tighter constraint is proposed as follows :

$$\vartheta = \|W_m \cdots W_1\|_s \quad (11)$$

which means that it has the same Lipschitz constant as a neural network where the activation is an identity function. It is also worth mentioning that this approach only provides a lower bound analysis of true Lipschitz constant for network with arbitrary signs.

III. PROPOSED APPROACH

In this work, we keep the traditional GNN as presented before and implement FISTA accelerated version of a Dual Forward-backward algorithm. For each layer $i \in \{1, \cdots, m\}$, we consider $\xi_i \in \mathbb{R}^{KQ_i \times (KQ_{i-1}+1)}$ which consists of weight matrix W_i and bias vector b_i.

Constraint 1 : non negativity

To ensure the computation of tight bound of Lipschitz constant as explained before, we impose that for each layer i, the weight matrix to be non-negative :

$$\mathcal{D}_i = \{W_i \in \mathbb{R}^{KQ_i \times KQ_{i-1}} \quad | \quad W_i > 0\} \quad (12)$$

Constraint 2 : graph topological structure

The structure of weight matrix W_i should take into account the topological structure (**??**), which means W_i should be in the form of Γ_i. This is ensured by imposing that W_i belongs to the following vector space:

$$\Omega_i = \{W_i \in \mathbb{R}^{KQ_i \times KQ_{i-1}} \mid (W_i)_{(u,v)} = 0 \text{ if } (\Gamma_i)_{(u,v)} = 0\} \quad (13)$$

Constraint 3 : Lipschitz constant constraint

We set $\overline{\vartheta}$ as desired maximum Lipschitz, we need to impose the following constraint on the spectral norm of weight matrices:

$$\|W_m, \cdots, W_1\|_s \leq \overline{\vartheta} \quad (14)$$

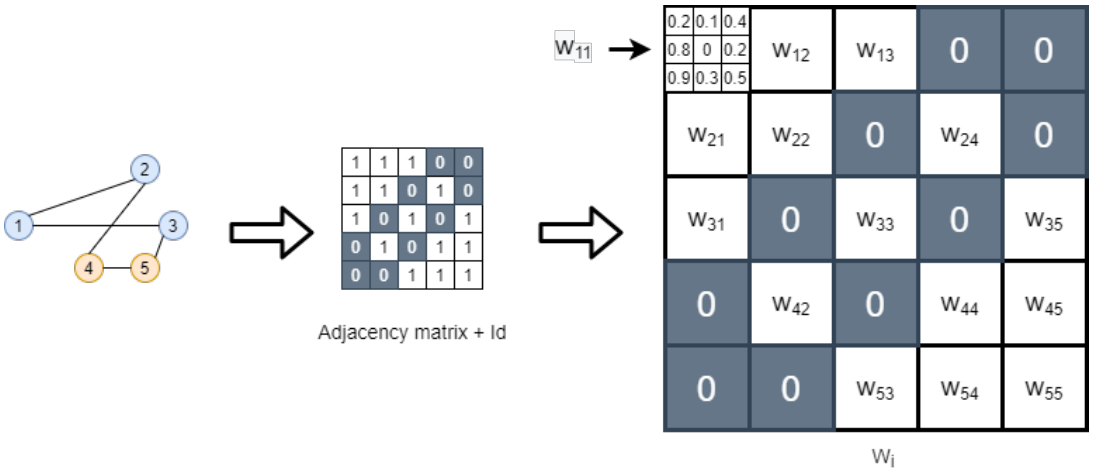


Figure 1: Visualisation of $W_i, w_{v,u}$ for a graph with 5 nodes

As we state before, calculating the accurate Lipschitz constant is a very difficult problem. Iteration methods are demanded to approximate the solution. At iteration t, for each layer $i \in \{1, \cdots, m\}$, $W_{i,t}$ the estimated weight matrix at iteration t the constraint (14) can be further expressed as :

$$\mathcal{C}_{i,t} = \{W_i \in \mathbb{R}^{KQ_i \times KQ_{i-1}} | \|A_{i,t} W_i B_{i,t}\|_s \leq \overline{\vartheta}\} \quad (15)$$

where the matrix $A_{i,t}$ and $B_{i,t}$ are respectively characterised by the matrix of previous layers and that of posterior layers, which is :

$$A_{i,t} = W_{m,t} \cdots W_{i+1,t} \quad (16)$$
$$B_{i,t} = W_{i-1,t+1} \cdots W_{1,t+1} \quad (17)$$

for the first layer $i = 1$, $B_{i,t}$ is an identity matrix of dimension KQ_1, for the last layer $i = m$, $A_{i,t}$ is the identity matrix of KQ_m.

The problem is thus reformulated by the projection

$$\widehat{W}_{i,t+1} = P_{\mathcal{C}_{i,t+1} \cap \Omega_i \cap \mathcal{D}_i}(\overline{W}_{i,t+1}) \quad (18)$$

where $\overline{W}_{i,t+1}$ is the estimated matrix after stochastic gradient optimization. Here we adopt an optimization method based on ADAM method proposed by Kingma and Ba where $\overline{W}_{i,t+1}$ can be calculated as follows :

$$\overline{W}_{i,t+1} = W_{i,t} - \gamma_t \mu_{i,t} / (\sqrt{\nu_{i,t}} + \epsilon) \quad (19)$$

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

We note $(\mathbb{M}_{l,h})_{1 \leq l \leq L}$ the mini-batches performed at epoch h. If our training data is composed of N graphs and for $1 \leq n \leq N$, z_n denotes the n-th pair of inputs and outputs, then our algorithm performs as Algorithm 1:

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**
    $g_{i,t} = \sum_{n \in \mathbb{M}_{l,h}} \nabla_i l(z_n, (\xi_{i,t})_{1 \leq i \leq m})$
    $\mu_{i,t} = \beta_1 \mu_{i,t-1} + (1-\beta_1) g_{i,t}$
    $\nu_{i,t} = \beta_2 \mu_{i,t-1} + (1-\beta_2) g_{i,t}^2$
    $\gamma_t = \gamma \sqrt{1-\beta_2^t}/(1-\beta_1^t)$
    $\xi_{i,t+1} = P_{\mathcal{S}_{i,t}}(\xi_{i,t} - \gamma_t \mu_{i,t}/(\sqrt{\nu_{i,t}} + \epsilon))$

where $\mathcal{S}_{i,t}$ is the constraint imposed at iteration t for layer i and $\mathcal{S}$ is a closed set expressing the above three constraints (12), (13) and (15):

$$\mathcal{S}_{i,t} = \{\xi_i | [(\xi_{j,t+1}^T)_{j<i} \xi_i^T (\xi_{j,t}^T)_{j>i}]^T \in \mathcal{S}\} \quad (20)$$

The essential step is thus to solve the projection expressed in (18), which is equal to find :

$$\widehat{W_{i,t+1}} = \underset{W \in \mathcal{C}_{i,t+1} \cap (\Omega_i \cap \mathcal{D}_{i,t})}{argmin} \|W - \overline{W}_{i,t+1}\| \quad (21)$$

where $\|\cdot\|$ designates the Frobenius norm.

In our case, we note $\mathcal{L}$ as follows :

$$\mathcal{L}(W_{i,t}) = A_{i,t} W_{i,t} B_{i,t} \quad (22)$$

The norm of $\mathcal{L}$ can be further expressed as :

$$\begin{aligned} \|\mathcal{L}(W_{i,t})\| &= \sup_{W_{i,t} \in \mathbb{R}^{KQ_i \times KQ_{i-1}} \setminus \{0\}} \frac{\|A_{i,t} W_{i,t} B_{i,t}\|}{\|W_{i,t}\|} \\ &= \|A_{i,t}\|_s \|B_{i,t}\|_s \end{aligned} \quad (23)$$

The equation (21) can be thus expressed as :

$$\begin{aligned} \widehat{W}_{i,t+1} = \underset{W \in \mathbb{R}^{KQ_i \times KQ_{i-1}}}{argmin} & (\frac{1}{2}\|W - \overline{W}_{i,t+1}\|^2 \\ & + \iota_{\mathcal{B}(0,\overline{\vartheta})}(A_{i,t} W B_{i,t}) + \iota_{\mathcal{D}_i}(W) + \iota_{\Omega_i}(W) \end{aligned} \quad (24)$$

where ι denotes the indicator function and $\mathcal{B}(0,\overline{\vartheta})$ is a closed ball of which the *spectral norm* of all elements is not superior than $\overline{\vartheta}$.

There is still no explicit solution for (24), the only way is to approximate the solution by an iterative way. Abboud et al. and Neacsu et al. present a Dual Forward-backward algorithm in their work. An accelerated version of this algorithm is provided as follows:

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

It is worth mentioning the set $\mathcal{D}_i$ is convex, and the intersection of $\mathcal{D}_i \cap \Omega_i$ is quite easy to handle, the projection of $P_{\mathcal{D}_i \cap \Omega_i}$ can be deduced by $P_{\mathcal{D}_i} \circ P_{\Omega_i}$.

The projection of W_i on $\mathcal{D}_i$ is easy to calculate :

$$P_{\mathcal{D}_i}(W_i) = ((\widetilde{W}_i)_{a,b})_{1 \le a \le KQ_i, 1 \le b \le KQ_{i-1}} \tag{25}$$

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

$$(\widetilde{W}_i)_{a,b} = \begin{cases} (W_i)_{a,b} & \text{if} \quad (W_i)_{a,b} \ge 0 \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

The projection $P_{\Omega_i}(W_i)$ can be accessed by :

$$P_{\Omega_i}(W_i) = \Gamma \odot W_i \tag{27}$$

where Γ refers to equation (4), $\odot$ represents element wise product.

The projection $P_{\mathcal{B}(0,\vartheta)}(Y_l)$ can be accessed by decomposing its singular values. For $Y_l \in \mathbb{R}^{KQ_m \times KQ_0}$, Y_l can be writen as $Y_l = U_l \Lambda_l V_l^T$ where $U_l \in \mathbb{R}^{KQ_m \times r}$, $V_l \in \mathbb{R}^{KQ_0 \times r}$. U_l, V_l are both semi-unitary matrix, Λ_l is square diagonal of size $r \times r$: $\Lambda_l = \mathrm{Diag}(\lambda_0, \cdots, \lambda_r)$, where $r \le \min\{Q_m, Q_0\}$. The projection is obtained by formulating :

$$P_{\mathcal{B}(0,\vartheta)}(Y_l) = U_l \widetilde{\Lambda}_l V_l \tag{28}$$

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

$$\widetilde{\lambda}_i = \begin{cases} \lambda_i & \text{if} \quad \lambda_i \le \overline{\vartheta} \\ \overline{\vartheta} & \text{otherwise} \end{cases} \tag{29}$$

IV. EXPERIMENTAL SETUP

A. Dataset description

In our experiment, a dataset should contain N graphs with the same topological structure, which means that each graph has the same edges and nodes. Each node is assigned with a label and M features.

Since the real data sets are generally too large, in the early stage of our experiment, the huge data sets are not conducive to validate our model. Besides, the graphs of real data sets usually do not have the same topology. For example, the Tox21 data set comprises 12,060 training samples and 647 test samples that represent chemical compounds, however, in each sample, one graph may contain 23 nodes while or contain 104 nodes. Therefore, it is necessary to build a data simulator to generate a synthetic data set for which we can appoint the required data size. The node features are generated from an M-dimension normal distribution depending on the label of the node.

We adopt a method to generate our data set based on the idea of frontier, here we consider generating a data set with N graphs, M features, and C classes.

Since all graphs generated shall share the same topological structure, in the first stage, we need to define the graph topology using an adjacency matrix A_d that is randomly initiated and symmetrical. Then we randomly choose C nodes from the graph as root nodes for C classes. For each class j, the root node $\mathcal{R}(j)$ is used to add new node to its class, which is realised by adding its unlabelled neighbors in the same class.

From the work of Jin et al., we know that there is a similarity of features between two adjacent nodes. To ensure this property, we appoint that all nodes of the same class are connected to each other. For class $1 \le j \le C$, we note $\mathcal{F}(j)$ as its frontier set which contains all the nodes selected for class j except those used as root nodes before.

If we note *node(j)* as node selected for class j and $neighbor(n)$ as the set of unlabelled neighbors of the node n , we can then formulate Algorithm 3.

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

In this way, we can obtain N graphs with the same

structure, but for a given class j, the sets of the nodes $node(j)$ are different from graph to graph.

To generate the data set where for class $1 \leq j \leq C$, the set $node(j)$ is the same for each graph. We can use the method proposed by Algorithm 4.

Algorithm 4: Generate data set with N graphs, M features and C classes with the same node sets

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

To simplify the problem, we suppose that node features are not correlated and have the same variance in order. We propose a way to measure the complexity of our data set. The measurement of overlapping among data of different class is showed by equation (30) :

$$\eta = \frac{\sum_{0 \leq i \leq j \leq C} ||\mu_i - \mu_j||_2}{\sigma} \tag{30}$$

where σ is the standard deviation of a normal distribution for one feature and μ_i is the mean vector of normal distribution for class i. Therefore, the smaller η is, the more overlapping we get and the more complex the data is. The covariance matrix of the M-dimension normal distribution for each label is the same and can be formulated as Equation (31) shows.

$$cov = \begin{bmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma^2 \end{bmatrix} \tag{31}$$

With the algorithm and overlapping measurement proposed, we are able to generate a data set containing N graphs, with each graph composing of K nodes, and each node is characterized by M features, all these nodes belong to C classes. Figure 2 gives an example of generating data in 3D of binary classification task. Figure 3 gives a closer view to one of the graphs. The color red and blue are used to distinguish the label of the nodes.

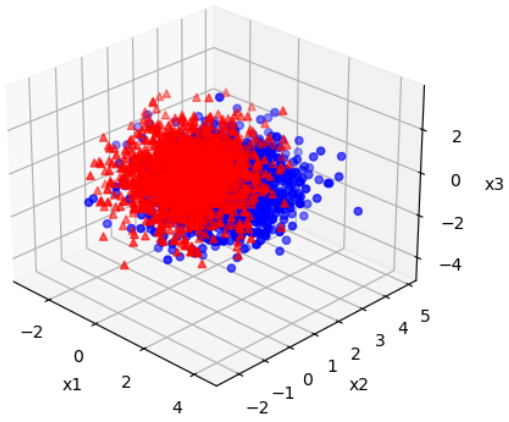


Figure 2: Data visualization example,$N = 1000$, $K = 30$, $M = 3$, and $C = 2$, Here, we put $\sigma = 1$, $\mu_0 = [0.1, 0.8, -0.1]$ for red nodes, and $\mu_1 = [0.7, 1.5, -0.6]$ for blue nodes.

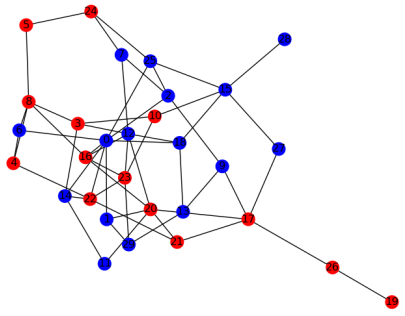


Figure 3: Graph visualization example

B. Model structure

Our model has an input layer, several dense layers, and a Softmax layer. Since the graphs in the dataset have the same topology, we take the whole graph as input and use batch training method. Figure 4 gives a simple view of an example of our proposed model.

We take x_i as the node feature vector for node i of the input graph where $i \in \{1, 2, \cdots, K\}$ and $x_i \in \mathbb{R}^M$. At the input layer, we simply input all the node features as a matrix X_0.

$$X_0 = [x_1, x_2, \cdots, x_K] \in \mathbb{R}^{M \times K} \tag{32}$$

At the first reshape layer, we stack the node features together into a column vector Y_0 as the output of this reshape layer, which yields:

$$Y_0 = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix} \in \mathbb{R}^{MK} \tag{33}$$

At dense layer i, if we take Q_i as the neuron number

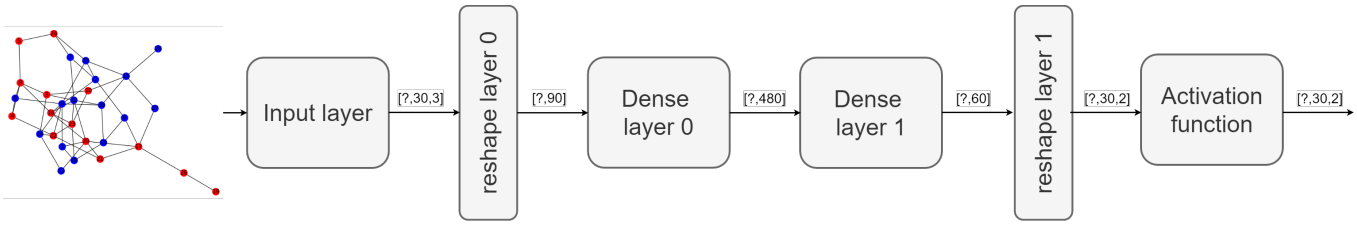


Figure 4: Model structure example. In this model, there are 2 dense layers, $K = 30$ nodes in each graph, $M = 3$ features for each node, and the output is the classification of each node in $C = 2$ classes

per node (we define $Q_0 = M$) and $A_d \in \mathbb{R}^{K\times K}$ the adjacency matrix that depends on the graph topology, then we can formulate the calculating process as follows, where $Y_i \in \mathbb{R}^{Q_i K}$ is the output of dense layer i, $b_i \in \mathbb{R}^{Q_i K}$ is the bias vector, and $\phi(\cdot)$ is an activation function. Here, to be more specific, $W_i \in \mathbb{R}^{Q_i K\times Q_{i-1}K}$ is the weight matrix that satisfies all the 3 constraints mentioned in Section III, i.e. constraint of non-negativity, constraint of graph topological structure, and constraint of Lipschitz constant, which means this weight matrix belongs to the intersection of these 3 constrained sets, referred to Equation (12), Equation (13), and Equation (15).

$$Y_i = \phi(W_i Y_{i-1} + b_i) \tag{34}$$

Suppose that there are m dense layers in total and then $Q_m = C$ because the number of neurons per node Q_m in the last dense layer should equal to the number of classes C.

$$Y_m = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} \in \mathbb{R}^{CK} \tag{35}$$

The second reshape layer then puts Y_m into the form $[y_1, y_2, \cdots, y_K] \in \mathbb{R}^{C\times K}$ with $y_i \in \mathbb{R}^C$. A Soft-max layer is used as the last activation function of our model to normalize the output to a probability distribution. The predicted label for node i in the graph is determined by:

$$\text{argmax}\{\text{Softmax}(y_i)\} \tag{36}$$

C. Training techniques

In the implementation part, we use TensorFlow framework and program with Python. During the training process, we applied ADAM optimizer which allows to get adaptive learning rates for each parameter. It combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that works well in most case. We use RELU as activate function in each dense layer.

In order to avoid over-fitting and tune hyper-parameters such as batch size and the number of neurons, we draw one-tenth of the samples from the training set for the validation set. The training set is used to train the different candidate classifiers, the validation set is used to compare their performances and decide which one to take.

Three methods are applied during the training process: tensorboard visualization, early stopping method, and the function to impose Lipschitz constant constraint. We use tensorboard to visualize the training process, monitoring the loss value and accuracy, by which we can clearly see how loss descends during training. It thus helps us understand the performance of our proposed approach. The function to impose Lipschitz constant constraint is called on each batch end, which constrains Lipschitz constant of the model and makes sure that each weight matrix is non-negative and is projected correctly with the graph adjacency matrix A_d. Early stopping method stops the training when the validation loss has stopped decreasing with a patience of epoch (We set this patience equal to 5). The maximum number of epochs for each training is defined as 1000.

The method to constrain Lipschitz constant constraint on each batch end is the FISTA algorithm that we mention in Section Proposed Approach.

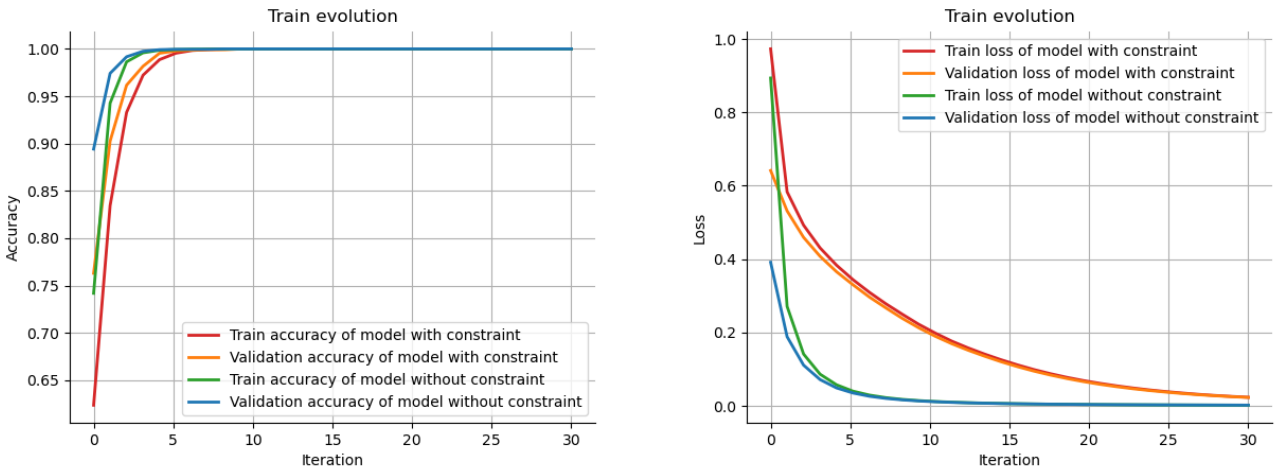
V. RESULT EVALUATION

A. Prevention of overfitting

Data sets generated by different algorithms

In the description of our artificial data sets, Algorithm 3 and Algorithm 4 generate artificially two kinds of data sets whose difference lies in whether the sets of nodes of the same label remain the same in different graphs.

Adopting Algorithm 4, we generate a data set with $N = 1000$, $K = 30$, $M = 3$, and $C = 2$, and design a simple model that contains only 2 dense layers with $[Q_0, Q_1, Q_2] = [3, 16, 2]$ to perform the node classification task. Figure 5a illustrates the variation of accuracy with the number of iterations. We notice that the accuracy on both training and validation data is almost 1 for models with or without Lipschitz constant constraint. While in Figure 5b, the loss decreases rapidly to 0 for both models. Therefore, this kind of data set is inappropriate for evaluation experiments. In the following evaluation, we only consider the graph



(a) Accuracy

(b) Loss

Figure 5: Performance 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 remain the same in different graphs, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 1$.

data set generated by Algorithm 3, where the sets of nodes of the same label are different from graph to graph.

Comparison of two models

We now perform the evaluation on the graph data set generated by Algorithm 3. In this experiment, we do not employ early stopping method to avoid overfitting. Figure 6a and Figure 6b respectively show the accuracy and loss on training and validation data set. We can observe that, for the model without Lipschitz constant constraint, there is a sharp decrease of loss on validation data set followed by a steady increase. While for the model with Lipschitz constant constraint, the loss on validation data set keeps decreasing with a slower speed. We can thus conclude that there exists an overfitting problem for the model without Lipschitz constant constraint but not for the model with Lipschitz constant constraint. In this single example, we deduce that Lipschitz constant constraint can prevent model from ovefitting.

Influence of overlapping ratio

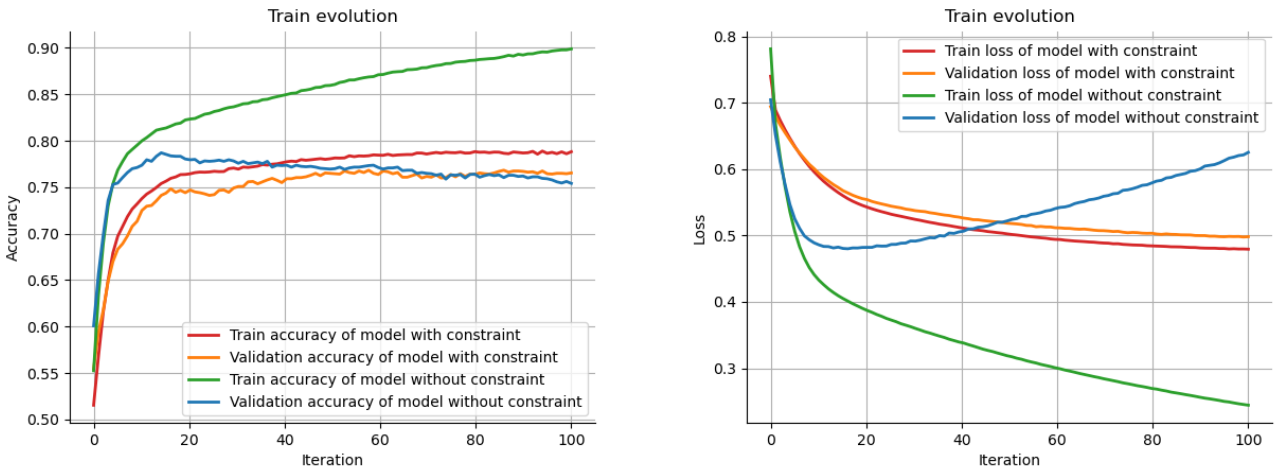
In the dataset description, we propose an indicator $\eta = \frac{\sum_{0 \leq i \leq j \leq C} ||\mu_i - \mu_j||_2}{\sigma}$ to measure the overlapping ratio of data of different class. In fact, this is also a measurement of complexity of the data set. A brief explanation can be brought up that the more data set of each class is intersected with other classes, which can be illustrated by a large standard deviation σ is and a small distance between the center μ of each class, it is with a higher probability that a node can be classified into several different classes at the same time, thus it is more difficult for the model to do the task. Therefore, the smaller η is, the more complex the data set is.

To prove that the overfitting has a relationship with different complexity of data sets, we randomly generate data sets with different η. We choose the mean value μ of each data cluster from the uniform distribution with range $[-2, 2]^M$ and σ from the uniform distribution with range $[0.5, 2.5]$. We then perform 50 groups of test and plot their final training loss and test loss in function of η in Figure 7a, training accuracy and test accuracy in Figure 7b.

From the results shown in Figure 7a and Figure 7b, we can observe that if the data set is complex enough, which means that the complexity of data sets η is below a certain threshold, then the model with Lipschitz constant constraint behaves better than the model without that constraint, and that no matter how η changes in the range $[0, 5]$, the difference between training and test accuracy (loss) of the model with Lipschitz constant constraint is always smaller than that of the model without Lipschitz constant constraint. We can deduce that the prevention of overfitting because of the existence of Lipschitz constant constraint can be realized for data sets of different complexity.

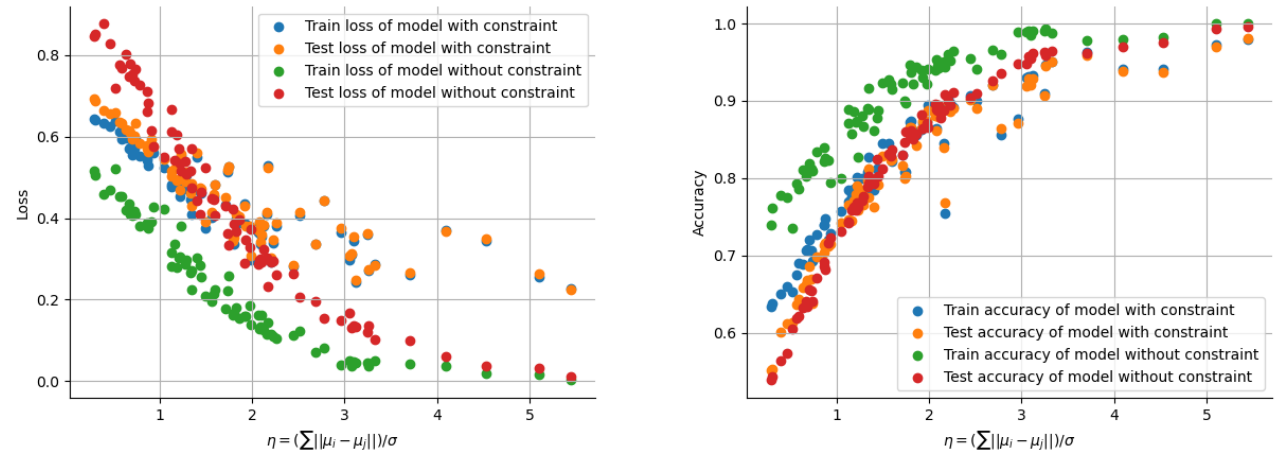
Baseline for multi-class classification

In order to avoid the contingency caused by the binary classification problem, we perform a multi-class classification with $C = 5$ with the rest of parameters remaining the same, i.e. $N = 1000$, $M = 3$, and $K = 30$. To verify the necessity of graph structure in the classification task, we propose two baseline models with the same model structure but with the sum of their adjacency matrices A_d and an identity matrix I_d equal to an identity matrix I_d, and an all-ones matrix J_d, respectively. The baseline model with its A_d a zero matrix represents the situation



(a) Accuracy (b) Loss

Figure 6: Performance of example 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 2]$ on the data set where the sets of nodes of the same label for each graph are different. Overfitting is observed for model without Lipschitz constant 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 $\overline{\vartheta} = 1$.



(a) Loss (b) Accuracy

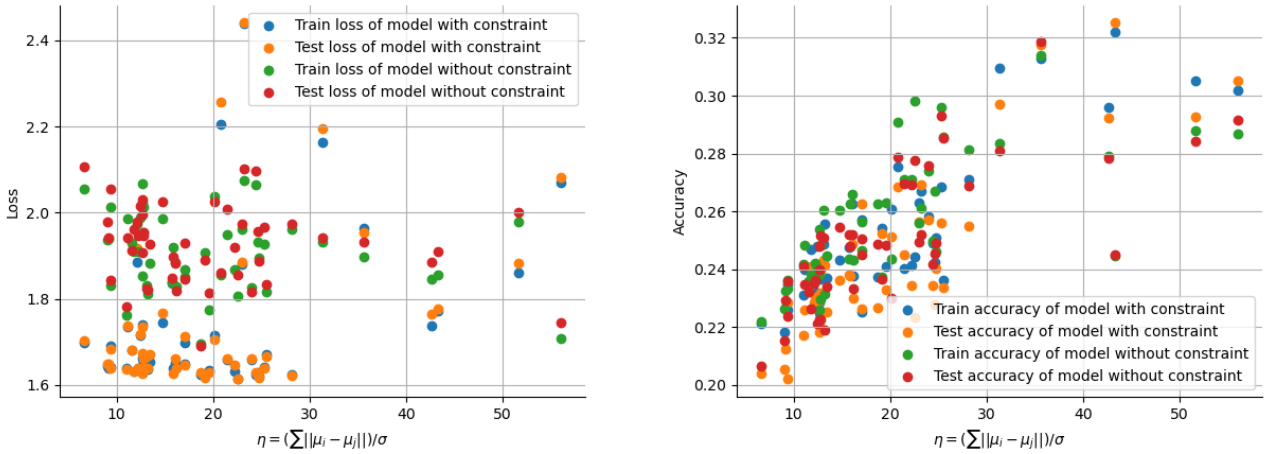
Figure 7: Performance of example 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 2]$ on datasets of different complexity with $N = 1000$, $C = 2$, $M = 3$, and $K = 30$, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 1$.

where all the nodes are independent. The other baseline model with $A_d + I_d = J_d$ represents the situation where it is a fully connected network and all the node features are associated with each other.

Figure 8a and 8b present the performance of the baseline model for which A_d is a zero matrix. Since there are in total 5 class, the accuracy of a random classifier should be 20%. We observe that this baseline model has actually learned some information from the data set because it has a test accuracy about 30% when the data set is not very complex (η is rather large).

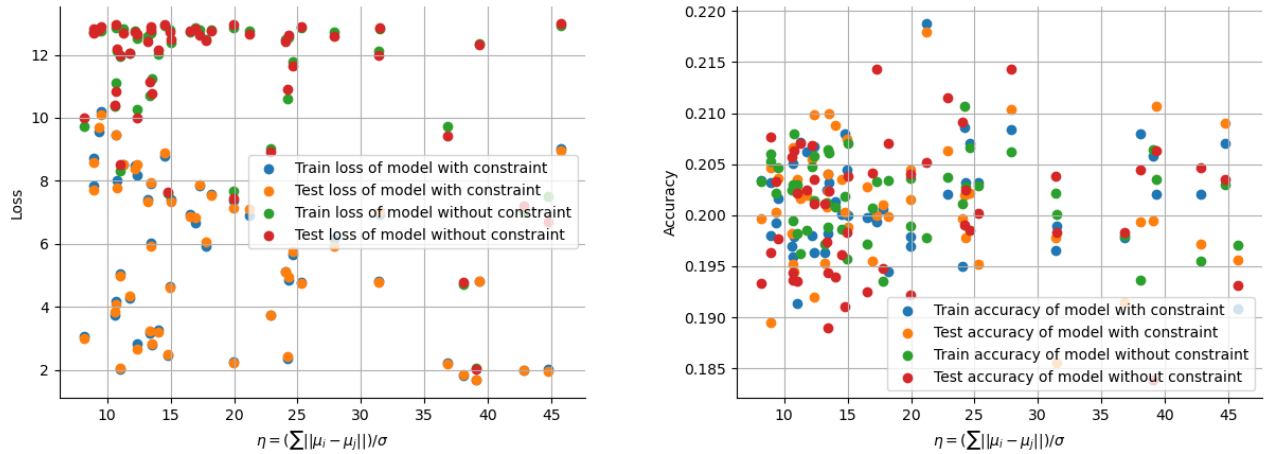
Figure 9a and 9b present the performance of the baseline model whose $A_d + I_d$ is an all-ones matrix J_d. No matter which data set it works on (no matter how η changes), the training and test accuracy is always around 20%, the same as that of a random classifier. It indicates that this baseline model has learned nothing extra from the data set.

The performance of these two baseline models explains from the opposite side that the graph structure is nec-



(a) Loss (b) Accuracy

Figure 8: 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 $\overline{\vartheta} = 1$.

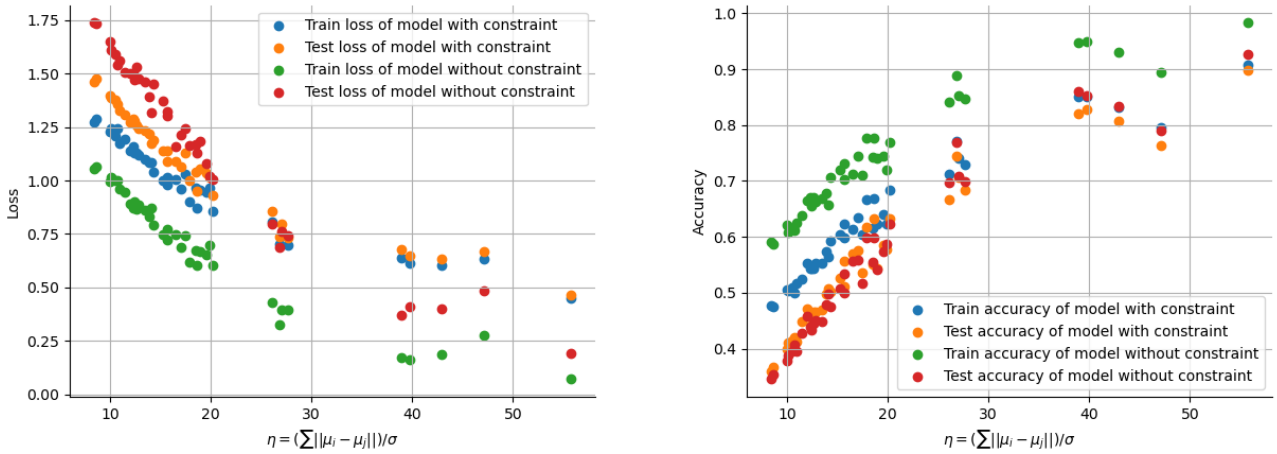


(a) Loss (b) Accuracy

Figure 9: 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 $\overline{\vartheta} = 5$.

essary for a classifier to perform well. That is also the reason why we introduce the adjacency matrix A_d completely instead of using some aggregate functions as GCN does to train a classifier.

We perform the same classification task ($C = 5$) on an example 2-dense-layer model. The accuracy shown in Figure 10b is all greater than 30%, which outperforms the baseline models. The difference between loss (resp. accuracy) on train and test sets of the model with Lipschitz constant constraint is smaller than that of the model without the constraint, which complies with the case of experiment when it is a binary classification task. From the above experiments with baseline models with $C = 5$ and example model with $C = 2$ and $C = 5$, we can see that the topological structure of graph is important and necessary for node classification which indicates the importance of using GNN to train a classifier, and that the GNN model with Lipschitz constant constraint shows a strong ability to resist the overfitting problem.



(a) Loss (b) Accuracy

Figure 10: 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, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, with $\overline{\vartheta} = 5$.

B. Improvement of robustness

We propose three methods for model robustness tests.

1. By adding noise: To test the robustness of our model to white noise, at first stage, we fit our model on the training set and test its performance on the test set, then at second stage, we add white noise to both training and test data sets. Then we can compare the results obtained using original data and perturbed data on different models (with or without Lipschitz constant constraint).

2. By modifying graph topological structure: similarly, we fit our model on the training set and test the performance on the test set at first. Then we modify the topological structure of the graph by deleting several edges randomly. This can be achieved by setting those corresponding weight blocks of the network to zero. After that, we test and compare the performance of original and modified models with and without Lipschitz constant constraint.

3. By testing with some primary adversarial attacks in the Adversarial Robustness Toolbox library. We select two evasion adversarial attacks that aim at generating adversarial sample to cheat the model. They are Projected Gradient Descent Attack proposed by Madry et al. and DeepFool proposed by Moosavi-Dezfooli et al.. Since our model has only several layers, we increase the perturbation level to compare the behaviors of the models.

To test the impact of Lipschitz constant constraint on model robustness, in the following experiments, we set the number of graph contained in data set $N = 1000$, the number of class $C = 5$, the number of features per node $M = 3$, and the number of node per graph $K = 30$.

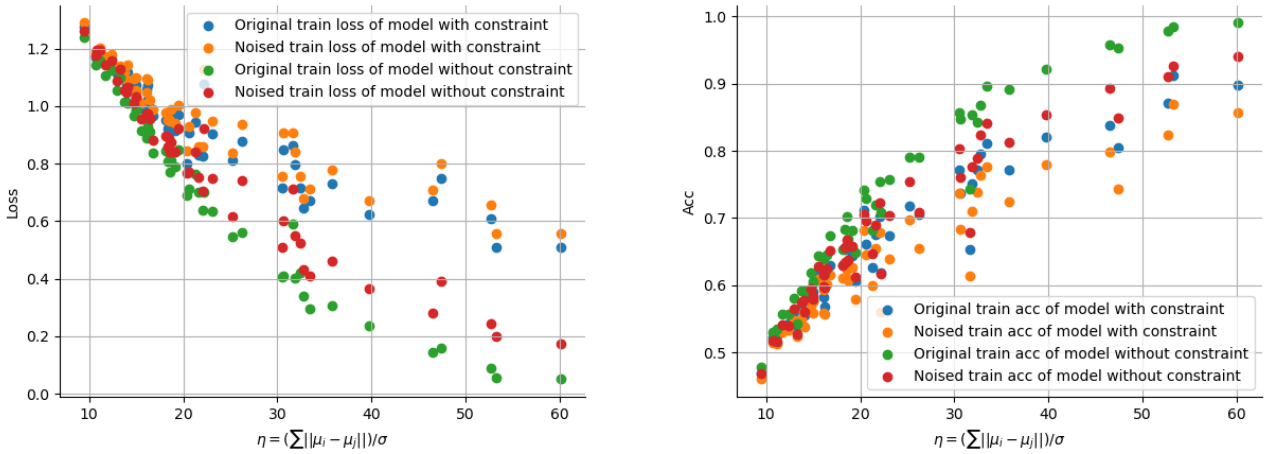
Perturbation caused by adding noise

In the first stage, we impose white noise on original train data and test data. Figure 11a and Figure 11b compare the training loss and accuracy of different models (with and without Lipschitz constant constraint) on the original train data and noisy train data. Figure 12a and Figure 12b compare the test loss and accuracy of different models (with and without Lipschitz constant constraint) on the original test data set and noisy test data.

We are more interested in test loss and accuracy. We observe that for test loss and accuracy in each experiments, when the overlapping ratio stays the same for each group, the difference between test loss (resp. accuracy) on original and noisy data of the model with Lipschitz constant constraint is smaller than that of the model without constraint. It signifies that Lipschitz constant constraint improve the resistance of graph neural network to white noise.

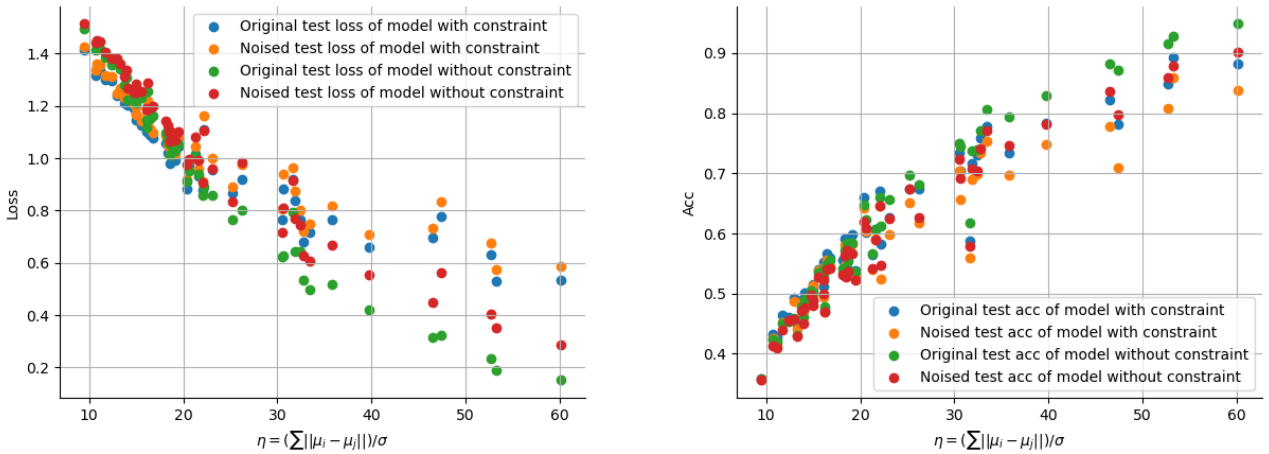
Perturbation caused by modifying graph structure

Now we consider to modify the graph topological structure by removing some edges from the graph. Figure 13a and Figure 13b compare the training loss and accuracy of different models (with and without Lipschitz constant constraint) on original train data and perturbed data whose graph topological structure has been modified. Similarly, Figure 14a and Figure 14b compare the test loss and accuracy for different models based on original test data and perturbed test data. In



(a) Training loss (b) Training accuracy

Figure 11: Performance of example 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 5]$ on noisy train 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 $\overline{\vartheta} = 5$.



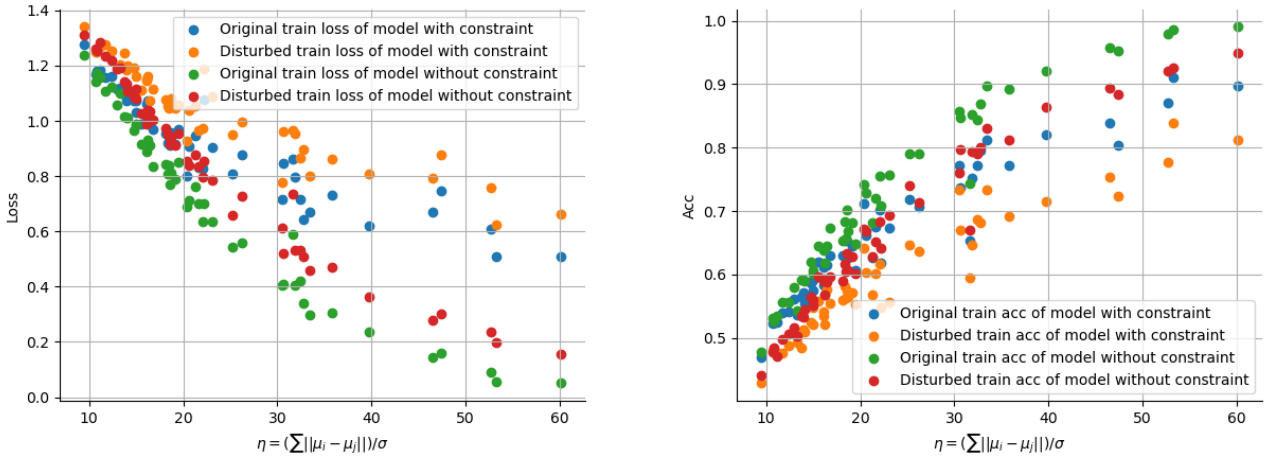
(a) Test loss (b) Test accuracy

Figure 12: 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 $N = 1000$, $C = 5$, $M = 3$, for each graph, μ is drawn from the uniform distribution $U[-2, 2]^M$ and similarly $\sigma \sim U[0.5, 2.5]$, 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 $\overline{\vartheta} = 5$.

our experiment, the topological structure of each graph is modified by randomly removing 20% of edges. We can see that the model with Lipschitz constant constraint doesn't show a significant improvement of robustness compared to that of model without Lipschitz constant constraint. However, this is reasonable if we consider the performance of baseline model, since the topological structure is essential to our model.

Adversarial attacks generated by DeepFool and PGD

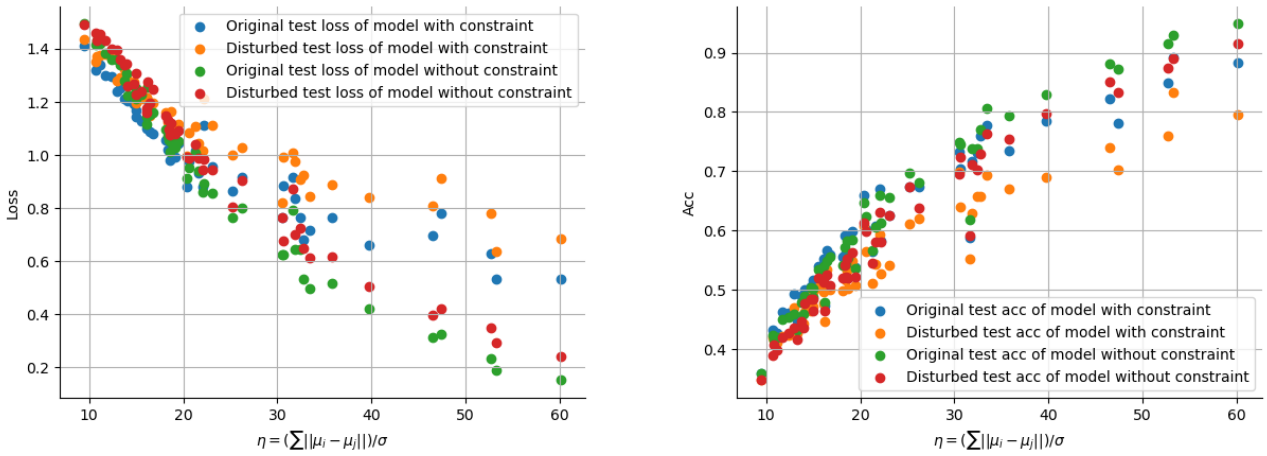
In this stage, we use DeepFool and Projected Gradient Descent Attack to generate adversarial test data based on the original test data. By comparing the decrease of accuracy on these adversarial samples, we can discuss about the robustness of the models with and without Lipschitz constant constraint. Different methods of generating adversarial samples have an impact on the accuracy decrease of the models. We can see in Figure 15 and Figure 16 that in general, the model with Lipschitz constant constraint has a less decrease in accuracy (the



(a) Training loss

(b) Training accuracy

Figure 13: Performance of example 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 5]$ on perturbed train 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 $\overline{\vartheta} = 5$.



(a) Test loss

(b) Test accuracy

Figure 14: 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 $\overline{\vartheta} = 5$.

length of blue bar is shorter than that of orange bar in general) and is therefore more robust against adversarial inputs.

From the results of these 3 test methods, we can argue that the model with Lipschitz constant constraint has a better performance than the model without the constraint does when dealing with noisy data and adversarial samples, and that the models have a similar performance when dealing with the graph data whose topological structure is modified.

C. Model comparison

The hyper parameters need to be tuned with different input data. These hyper parameters in our model includes the number of dense layers, the activate function, the number of neurons per node in each dense layer, the lower bound of Lipschitz constant to control, and batch size during the training. It is hard to tune all of these hyper parameters perfectly. However since our artificially generated data is quite simple compared to real life data,

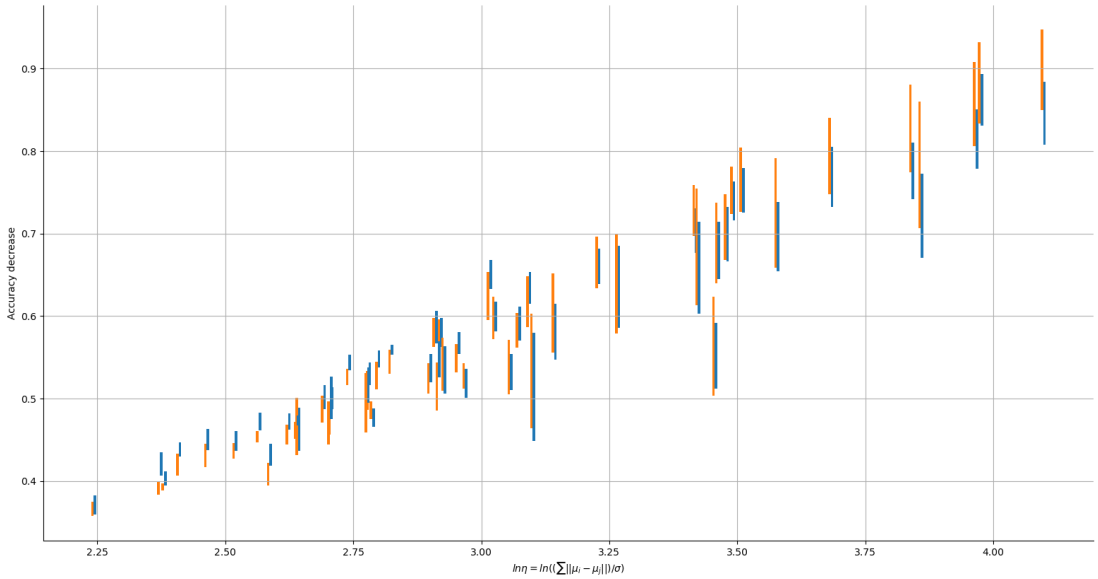


Figure 15: Accuracy decrease of the 2-dense-layer models with and without Lipschitz constant constraint on adversarial test data generated by Projected Gradient Descent Attack, blue bar for the model with constraint and orange bar for the model without constraint, for each graph, μ is drawn from the uniform distribution $U[-2,2]^M$ and similarly $\sigma \sim U[0.5,2.5]$, with $\overline{\vartheta} = 5$.

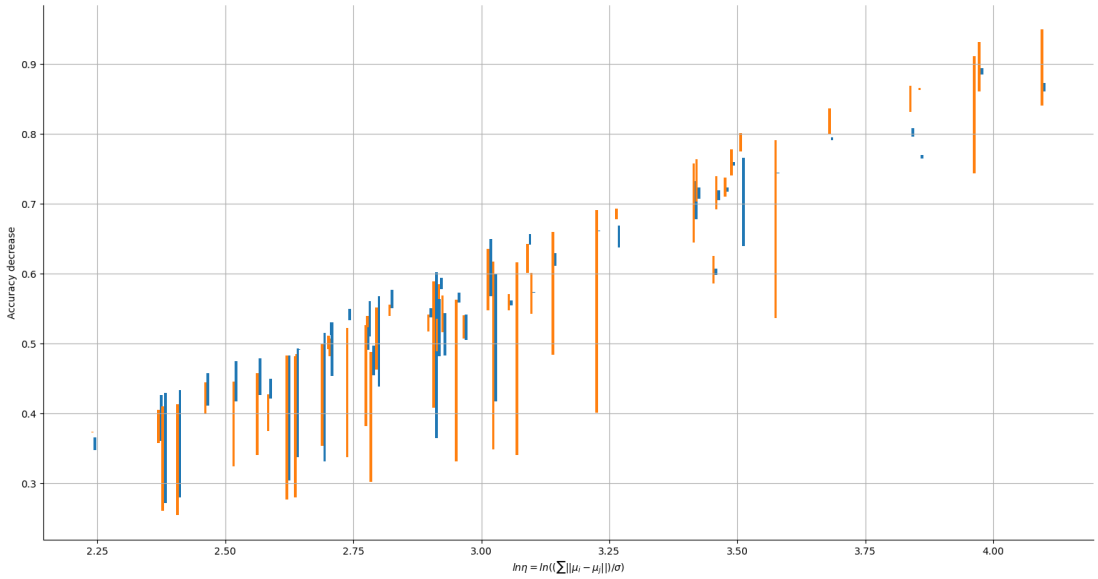
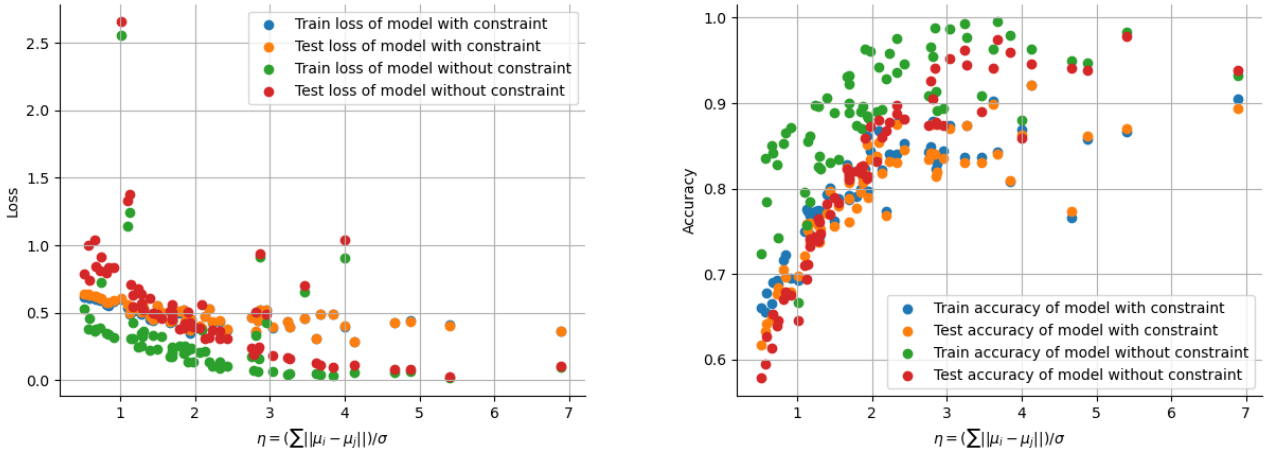
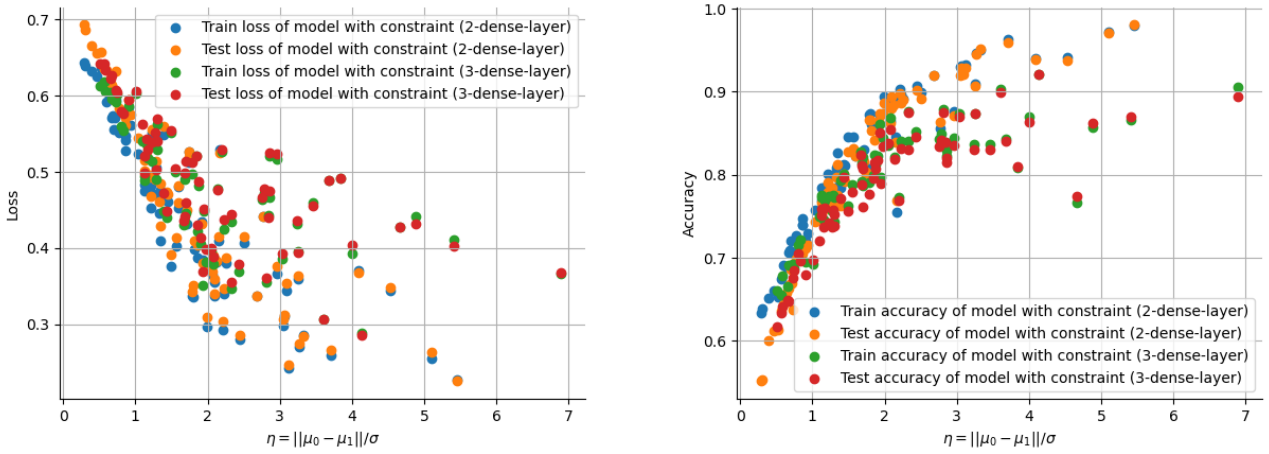


Figure 16: Accuracy decrease of the 2-dense-layer models with and without Lipschitz constant constraint on adversarial test data generated by DeepFool, blue bar for the model with constraint and orange bar for the model without constraint, for each graph, μ is drawn from the uniform distribution $U[-2,2]^M$ and similarly $\sigma \sim U[0.5,2.5]$, with $\overline{\vartheta} = 5$.



(a) Loss (b) Accuracy

Figure 17: Performance of the example 3-dense-layer model with $[Q_0, Q_1, Q_2, Q_3] = [3, 8, 16, 2]$ 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 $\overline{\vartheta} = 1$.



(a) Loss (b) Accuracy

Figure 18: Performance Comparison between the 2-dense-layer model with $[Q_0, Q_1, Q_2] = [3, 16, 2]$ and the 3-dense-layer model with $[Q_0, Q_1, Q_2, Q_3] = [3, 8, 16, 2]$ 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 $\overline{\vartheta} = 1$.

the existing model is powerful enough to handle with this kind of data. We do a small experiment to prove our hypothesis by simply adding one more dense layer in our existing model and compare their performances.

We first see in Figure 17a and Figure 17b how an example 3-dense-layer model with Lipschitz constant constraint performs compared with the model without the constraint. It shows that the test accuracy and loss is almost the same when data set is rather complex between the models with and without the constraint, but when data set is less complex, the model without the constraint outperforms the one with the constraint. From the comparison of performance of the 2-dense-layer model and the 3-dense-layer model in Figure 18a and Figure 18b, it can be seen that the 2-dense-layer model performs better in almost all the cases when the dataset parameters $C = 2$, $M = 3$, and $K = 30$. We argue that when the graph data is more complicated, for example a node with more features or a bigger graph, the model with more dense layers can have a better performance.

D. Advantages and drawbacks analysis

The advantages of the model with Lipschitz constant constraint applying on GNN training are that it can improve the model's robustness to noisy data and adversarial data and that it can avoid overfitting problem. Incorporating the adjacency matrix into our GNN model instead of using an aggregate function makes the model learn more information. The drawbacks are that currently this model can only apply on the graph data sets where each graph has the same topological structure, and that it cannot deal with large graphs containing too many nodes or features due to the limit of memory. Since the beginning of this project, we have been searching for graph data sets with the same graph topological structure but it is hard to find one that satisfies our requirements.

VI. CONCLUSION

This paper proposes to improve the robustness of GNNs by imposing Lipschitz constraint on weights of the model. We demonstrate the good performance of the proposed method by showing its performance on noisy data and adversarial data. When we train the model for data sets with bigger graphs and more features, the complexity of GNN and the number of dense layers should increase to fit the model. The proposed method focuses on positive weights and can 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. All the experiment codes can be found in our Github link https://github.com/SJTUzhou/Lipschitz_gnn.

[1] Feriel Abboud, Emilie Chouzenoux, Jean-Christophe Pesquet, Jean-Hugues Chenot, and Louis Laborelli. Dual Block Coordinate Forward-Backward Algorithm with Application to Deconvolution and Deinterlacing of Video Sequences. *Journal of Mathematical Imaging and Vision*, 59(3):415–431, November 2017. doi:10.1007/s10851-016-0696-y. URL `https://hal.archives-ouvertes.fr/hal-01418393`.

[2] Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning, 2019.

[3] Patrick L. Combettes and Jean-Christophe Pesquet. Lipschitz certificates for layered network structures driven by averaged activation operators, 2020.

[4] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data, 2018.

[5] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Hohman, Siwei Li, Li Chen, Michael E. Kounavis, and Duen Horng Chau. Shield: Fast, practical defense and vaccination for deep learning using jpeg compression. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & amp; Data Mining*, KDD '18, page 196–204, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi:10.1145/3219819.3219910. URL `https://doi.org/10.1145/3219819.3219910`.

[6] Negin Entezari, Saba A. Al-Sayouri, Amirali Darvishzadeh, and Evangelos E. Papalexakis. All you need is low (rank): Defending against adversarial attacks on graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 169–177, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368223. doi:10.1145/3336191.3371789. URL `https://doi.org/10.1145/3336191.3371789`.

[7] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & amp; Data Mining*, KDD '20, page 66–74, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi:10.1145/3394486.3403049. URL `https://doi.org/10.1145/3394486.3403049`.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[9] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[10] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks, 2016.

[11] Ana Neacsu, Jean-Christophe Pesquet, and Corneliu Burileanu. Accuracy-Robustness Trade-Off for Positively Weighted Neural Networks. In *ICASSP 2020 - 2020 IEEE International Conference on Acous-*

tics, Speech and Signal Processing, pages 8389–8393, Barcelona, France, May 2020. IEEE. doi: 10.1109/ICASSP40776.2020.9053803. URL https://hal.archives-ouvertes.fr/hal-03140282.

[12] Ana Neacsu, Kavya Gupta, Jean-Christophe Pesquet, and Corneliu Burileanu. Signal denoising using a new class of robust neural networks. In *EUSIPCO 2020 - 28th European Signal Processing Conference*, pages 1492–1496, Amsterdam, Netherlands, January 2021. IEEE. doi:10.23919/Eusipco47968.2020.9287630. URL https://hal-centralesupelec.archives-ouvertes.fr/hal-03115064.

[13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

[14] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense, 2019.

[15] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 1399–1407, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi:10.1145/3292500.3330851. URL https://doi.org/10.1145/3292500.3330851.

[16] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning, 2019.

[17] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Jul 2018. doi:10.1145/3219819.3220078. URL http://dx.doi.org/10.1145/3219819.3220078.