Learning from Negative Links

He Jiang, *Student Member, IEEE,* and Haibo He, *Fellow, IEEE*

***Abstract*—Recently, graph convolutional networks (GCNs) and their variants have achieved remarkable successes for the graph-based semi-supervised node classification problem. With a GCN, node features are locally smoothed based on the information aggregated from their neighborhoods defined by the graph topology. In most of the existing methods, the graph typologies only contain positive links which are deemed as descriptions for the feature similarity of connected nodes. In this paper, we develop a novel GCN-based learning framework that improves the node representation inference capability by including negative links in a graph. Negative links in our method define the inverse correlations for the nodes connected by them and are adaptively generated through a neural network based generation model. To make the generated negative links beneficial for the classification performance, this negative link generation model is jointly optimized with the GCN used for class inference through our designed training algorithm. Experiment results show that the proposed learning framework achieves better or matched performance compared with current state-of-the-art methods on several standard benchmark datasets.**

***Index Terms*—Semi-supervised learning, graph-based learning, graph convolutional networks.**

I. Introduction

Real-world data of various domains can be modeled by graphs due to their intrinsic nature of interconnection, such as social networks, communication systems, and protein/molecule structures. Accordingly, graph-based data mining techniques, which include node classification [1], link prediction [2], and clustering [3], have plenty of practical applications, *e.g.*, recommendation [4], [5], network optimization [6], [7], and community detection [8], [9], just to name a few. In this paper, we focus on one of the fundamental problems of graph mining, *i.e.*, the semi-supervised node classification, which concerns predicting node labels based on graph topologies and node features.

Classic semi-supervised node classification approaches largely depend on graph Laplacian regularization [1], [10], [11]. Generally, a graph Laplacian regularization term is used to smooth the label distribution for the whole graph, which is supported by the assumption that the connected nodes are likely to share the same label. In [1], an Gaussian random field model is proposed for semi-supervised learning, where the energy function is designed based on the graph Laplacian. An iteration algorithm that propagates node labels is introduced in [10], and a graph Laplacian regularization based optimization framework is also developed to obtained the solution of the iteration algorithm. Belkin *et al.* [11] present a semi-supervised learning algorithm where the loss function contains two regularization terms with one controlling the classifier complexity and the other smoothing the label distribution over the graph.

Recently, convolutional neural networks have been generalized to process graph-structured data, known as GCNs, and have achieved tremendous successes across a wide range of graph mining problems [12]–[14]. In a semi-supervised node classification problem, a GCN is employed to learn effective node representations. Popular models include GCN [15] and graph attention networks (GAT) [16]. Compared with the label similarity assumption practiced by the graph Laplacian based methods, GCN-based approaches extend the similarity assumption to node features. This idea is reflected by the way of the feature inference procedure of GCNs, where the feature vector of one node is updated by the weighted summation of its neighbors'. This process smooths the node feature based on its neighborhood and the coefficients of the neighbors can be determined via different strategies. For instance, with the GCN model of [15], the coefficients of the nodes are completely calculated according to the node degrees, while some other researchers believe that the neighbors of a node possess different importances so that an attention mechanism is introduced to adaptively assign the coefficients based on the node features [16].

Numerous algorithms have been proposed to improve the performance of GCNs. Adversarial training algorithms developed by [17]–[19] aims to utilize perturbations on node features to improve the robustness of GCNs. In [17], a virtual adversarial loss is built based on the KL divergence between node label distribution calculated from the original graph and the distribution calculated from the graph with node perturbations. This loss is used as a regularization term of the overall training loss function to improve the generalization performance of GCNs. Besides the virtual adversarial loss, Feng *et al.* [18] include a graph adversarial regularizer that encourages the graph adversarial examples to be classified similarly to the connected examples. An adversarially regularized graph embedding learning algorithm is proposed by [19], which aims to learn latent node codes that preserve graph topology information. In [20], a variational expectationmaximization algorithm is developed to enable the interactive learning of two GCNs with one learning node representations and the other modeling label dependencies. To learn disentangled node features, a neighborhood routing mechanism is proposed to assign the neighbors to different channels based on the underlying factors [21]. There also exist some works that deem the observed graph topology as a sample of the distribution that models the probability of node connectivity [22], [23]. And, generative models are employed to generate different graph topologies to train GCNs, such that the classification performances are improved, especially when the training sam-

This work was supported by the National Science Foundation under grant ECCS 1731672.

H. Jiang and H. He are with the Department of Electrical, Computer and Biomedical Engineering, University of Rhode Island, Kingston, RI 02881 USA (e-mail:jiang_he@uri.edu; haibohe@uri.edu).

ples are very few. Interested readers can refer to [24] and [25] for more detailed reviews of the GCN-based methods.

In most of the prior works, the GCN based methods are designed to deal with the graphs only containing positive links that are used to model the positive correlations of connected nodes. In this paper, we assume negative correlations among nodes also exist in a graph, which can be exploited to better infer node features or labels. We propose a novel learning framework that can generate meaningful negative links to describe inverse correlations of node pairs. A negative link is directed and specifies that the node at its head is the negative neighbor of the node at its tail. And it is associated with a negative weight to quantify the negative correlation. Furthermore, a GCN model that is compatible with graphs containing negative links is developed. In the feature propagation process of our model, the negative neighbor of a node is assigned with a negative coefficient which is determined by the weight of the corresponding negative link. An optimization algorithm is designed to enable the generated negative links to improve the semi-supervised learning performance of the GCN. We conduct thorough experiments on three benchmark datasets. Results demonstrate that our proposed method is able to outperform or match the performance of state-of-the-art models in most settings.

Note that there exist some other works concerning GCNs for graphs with negative links, such as [26] and [27]. However, the negative links in their work, which are generated based on the modeling strategies, have different semantics compared with ours. The target graph datasets of these works contain negative links inherently. For instance, in the Epinions dataset [28], which is tested by both [26] and [27], negative links exist originally and represent the ‘distrusting’ relationship of the connected users/nodes. In contrast, our work focus on the graphs that only contain positive links initially and negative links are generated as auxiliaries to improve the GCN performance. Therefore, our methods and these previous works have completely different designs and applications. The rest of this paper is organized as follows. In Section II, we review the formulation of graph-based semi-supervise node classification problem and the popular solutions. In Section III, we present the proposed negative link generation mechanism and the GCN model designed for the graphs with generated negative links. The optimization algorithm for our model is introduced in Section IV. Section V presents the experiment results to validate the performance of our method. Finally, we conclude our work in Section VI.

II. Background

A. Semi-supervised Node Classification

Let a graph be denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ is a finite set of nodes; $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the link set. The link e_{ij} is represented by an ordered node pairs (v_i, v_j). Additionally, the weights of links are specified by a symmetric adjacency matrix $A \in \mathbb{R}^{N\times N}$. The entry lie in row i and column j, *i.e.*, A_{ij}, is the weight of e_{ij}. Generally, $A_{ij} > 0$ if $(v_i, v_j) \in \mathcal{E}$; otherwise, $A_{ij} = 0$. If e_{ij} exists, v_j is called a neighbor of v_i. The set of neighbors of v_i is denoted by $\mathcal{N}_i$. For v_i, there is a feature vector x_i and a label $y_i \in \{1, 2, \ldots, C\}$ representing its class, where C is the total number of the classes. In a semi-supervised node classification problem, only a subset of nodes have the label information. Without loss of generality, we assume the fist M nodes, $\{v_1, \ldots, v_M\}$, are labeled. Let $\mathcal{F}$ denote the set of $N \times C$ matrices with non-negative entries. The one hot label matrix of all nodes is defined as $Y \in \mathcal{F}$, where $Y_{ij} = 1$ if v_i is labeled as $y_i = j$; otherwise, $Y_{ij} = 0$. The learning objective is to predict the classes of all unlabeled nodes. For this problem, several graph Laplacian regularization based approaches are proposed, which depends on the assumption that two connected nodes are likely to share a common label [10], [11]. For instance, in [10], the cost function is designed as

$$\begin{aligned}\mathcal{Q}(F) =&\frac{1}{2}\Big(\sum_{i=1}^{N}\sum_{j\in\mathcal{N}_i} A_{ij}\Big\|\frac{1}{\sqrt{D_{ii}}}F_i - \frac{1}{\sqrt{D_{jj}}}F_j\Big\|^2 \\ &+ \mu\sum_{i=1}^{N}\|F_i - Y_i\|^2\Big). \qquad (1)\end{aligned}$$

In the equation above, $F = [F_1, \ldots, F_N]^{\mathrm{T}}$; the C-dimensional vector F_i is the classification of v_i by labeling $y_i = \arg\max_{j\le C} F_{ij}$; and μ is a weighing factor. The first term of the right hand side of Eq. (1) provides the Laplacian regularization. By minimizing this term, the classification of the connected nodes are smoothed over the graph. Here, graph links are considered as descriptions of class similarities for the connected nodes. Recently, some researchers believe that the graph links can convey additional information besides the class similarities. As a result, GCN-based methods are proposed, which extend the similarities of the connected nodes from labels to features, and have achieved significant breakthroughs.

B. GCN-based Solution

A GCN extracts node features through a stack of graph convolutional layers. Denoting node features at layer l by $H^{(l)} \in \mathbb{R}^{N\times K^{(l)}}$ where $K^{(l)}$ is the feature dimension, the computation formula of $H^{(l)}$ is

$$H^{(l)} = \sigma(\hat{A}H^{(l-1)}W^{(l)}) \qquad (2)$$

where $\hat{A}$ is the “normalized” adjacency matrix; $H^{(l-1)} \in \mathbb{R}^{N\times K^{(l-1)}}$ is the feature matrix of the previous layer; and we have $H^{(0)} = X$ with $X = [x_1, \ldots, x_N]^{\mathrm{T}}$; $W^{(l)} \in \mathbb{R}^{K^{(l-1)}\times K^{(l)}}$ is a trainable weight matrix; and $\sigma(\cdot)$ is an element-wise nonlinear activation function. Eq. (2) can be decomposed into three consecutive steps: feature propagation, linear transformation, and nonlinear activation [29], which are illustrated by Fig. 1. At the feature propagation step, the feature vector of each node is smoothed by the features of its neighbors, where the coefficients of the neighbors are controlled by $\hat{A}$. Then, the smoothed node features are transformed to a new feature space through the weight matrix $W^{(l)}$. Finally, a nonlinear element-wise activation function, $\sigma(\cdot)$, is applied and the output serves as the input of the next layer. At the final layer, the nonlinear function is omitted to generate input of the softmax classifier.

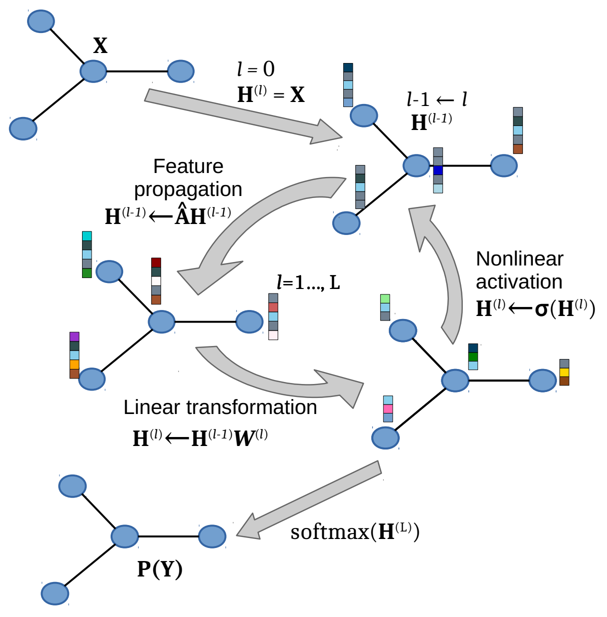

X
$l = 0$
$\mathbf{H}^{(l)} = \mathbf{X}$
l-1 ← l
$\mathbf{H}^{(l-1)}$
Feature propagation
$\mathbf{H}^{(l-1)} \leftarrow \hat{\mathbf{A}}\mathbf{H}^{(l-1)}$
Nonlinear activation
$\mathbf{H}^{(l)} \leftarrow \sigma(\mathbf{H}^{(l)})$
l=1..., L
Linear transformation
$\mathbf{H}^{(l)} \leftarrow \mathbf{H}^{(l-1)}\boldsymbol{W}^{(l)}$
softmax($\mathbf{H}^{(L)}$)
P(Y)

Fig. 1. Computation procedure of a GCN. The GCN transforms the node features through L graph convolutional layers and a softmax classifier is applied on the final representation. At layer l, the input comes from the output of layer $l-1$. Feature propagation is implemented by the matrix multiplication between 'normalized' adjacency matrix and the node feature vectors to smooth the node features. The smoothed feature vectors are still in the vector space of the input. Through the weight matrix $W^{(l)}$, the node features are transformed to the vector space of the output. The design of this figure is inspired by [29].

To improve the performance of this GCN framework, several approaches are proposed from different perspectives. For the GCN proposed by [15], the feature propagation matrix is designed as

$$\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2} \tag{3}$$

where $\tilde{A} = A + I_N$; $\tilde{D}$ is the degree matrix of $\tilde{A}$; and I_N is the $N \times N$ identity matrix. In this way, $\hat{A}$ is completely determined by the adjacency matrix. Velikovi *et al.* [16] consider the nodes within a neighborhood to be of different importances, and they propose an adaptive attention mechanism to learn the importances of the neighbors. Some researchers assume that the original graph topology is not enough to describe the node dependency. Accordingly, they deem the topology of a graph as a random variable and learn the corresponding distribution through graph generative models [22], [23]. Besides architecture innovations, modifications on training algorithms are also demonstrated to be beneficial to the performance of GCNs [17], [18], [20].

III. Negative Links for Node Feature Inference

Most of the previous GCN-based methods locally smooth the node feature by assigning the neighbors with positive coefficients, where a node is assumed to be positively correlated with its neighbors. In this paper, propose a novel GCN learning framework that improving the feature and label inference capability by considering the nodes of inverse correlated properties, called negative neighbors. As the original graph contains positive links only, in this section, we first propose a negative link generation model. Then, the GCN model designed for graphs with positive links [15] is modified to be compatible with the graph with negative links.

A. Negative Link Generation

To generate a negative link for a node, two questions need to be answered: 1) which node is selected as the negative neighbor, and 2) what is the weight of the negative link. Consequently, our proposed negative link generation model aims to solve these two questions for each v_i. The generation procedure is summarized by the following equations:

$$\text{for } j = 1, \ \ldots, \ N$$

$$\omega_{ij} = \begin{cases} -\infty & \text{if } y_i = y_j \\ f_\theta(z_i, z_j) & \text{otherwise} \end{cases} \tag{4}$$

$$i_c^{(-)} = \arg\max_j \{\omega_{ij} | y_j = c\}, c = 1, 2, \ldots, C \tag{5}$$

$$i^{(-)} = \arg\max_{i_c^{(-)}} \{\omega_{ii_c^{(-)}}\}_{c=1}^{C} \tag{6}$$

$$A_{ii^{(-)}} = -\frac{\exp(\omega_{ii^{(-)}})}{\sum_{c=1}^{C} \exp(\omega_{ii_c^{(-)}})}. \tag{7}$$

In Eq. (4), ω_{ij} is a quantity measuring the potential of v_j being a negative neighbor for v_i; z_i and z_j are the feature representations of v_i and v_j; f_θ is a trainable function parameterized by θ; and ω_{ij} is constrained to be negative infinity if v_i and v_j belong to a same class, which means we only seek negative neighbors from different classes. Eq. (5) finds the index of the nodes with highest ω_{ij} in each class, which is denoted by $i_c^{(-)}$ for class c. We consider $v_{i_c^{(-)}}$ as the representative of the potential negative neighbors in class c. Among the C representative negative neighbors (one for each class), the final negative neighbor $v_{i^{(-)}}$ is selected by Eq. (6). The generated negative link connecting v_i and $v_{i^{(-)}}$ is denoted by $e_{ii^{(-)}}$, which is directed, *i.e.*, from v_i to $v_{i^{(-)}}$. And the associated weight of this negative link $A_{ii^{(-)}}$ is calculated via Eq. (7), where a softmax function is implemented over $\{\omega_{ii_1^{(-)}}, \ldots, \omega_{ii_C^{(-)}}\}$.

In this way, other representative negative neighbors are used as references to assess the connecting strength of $e_{ii^{(-)}}$. And the negative sign at the right hand side of Eq. (7) indicates $v_{i^{(-)}}$ is negatively correlated to v_i. Another functionality of Eq. (7) is that it normalizes the magnitude of the generated negative links to (0, 1). In the original graph, the positive links are weighted by 1; and since the positive links reveal the 'true' structure of the data, it is reasonable to use the magnitude of the positive link as the upper bound of the generated negative links. Filling A at row i column $i^{(-)}$ with $A_{ii^{(-)}}$ for $i = 1, \ \ldots, \ N$, the updated adjacency matrix is denoted by A^* which can be asymmetric because the negative links are directed. It should be noted that a generated negative link can possibly connect two nodes that have already been connected with positive links, and if so, we substitute the existing entry of A with $A_{ii^{(-)}}$. We implement this operation due to the following two motivations. First, the positive links are usually built based on the modeling strategy, and GCN-based method use positive links to describe the similarities of the connected nodes; however, this relationship can be noisy which is also point out by [22], [23]. Second, in our method, the optimization algorithm, which is introduced in the next section, makes the generated negative links beneficial for the node feature inference so that we will substitute the positive link if it is conflicting with a generated negative link.

According to Eq. (4) and (5), the proposed negative generation process requires node labels, which are not available for the nodes outside the training set. Under this situation, we employ a GCN as an inference or recognition model to provide an estimation of the class for the unlabeled nodes. We denote this GCN by $g_\psi(X, A)$ where ψ is the weights; X is the node feature matrix; and A is the adjacency matrix. Moreover, this GCN can also be used to extract node representations simultaneously. An implementation scheme of our proposed negative link generation procedure is depicted in Fig. 2. With this architecture, the graph data is first input to the GCN to estimate the label distribution and to extract node representations. The labels of the nodes outside the training set can be sampled from the obtained distribution. The node feature representations, $\{z_i\}_{i=1}^N$, are set to the output feature of the last layer of the GCN. Then, a three-layer fully connected neural network is employed as the instantiation of $f_\theta(z_i, z_j)$, and the input is designed as $[z_i^\mathrm{T}, z_j^\mathrm{T}]$, *i.e.* the concatenation of the paired feature representation vectors. The obtained $\{\omega_{ij}\}$ is used to generate negative links through Eq. (5)-(7). We illustrate the generated negative links with red dashed lines in Fig. 2.

Remark 1: This negative link generation process is similar to the latent space model (LSM), which is usually employed to generate positive links [23], [30]. The parameters of the LSM in these previous works are trained to maximize the likelihood of the observed positive link. However, as there are no original negative links in the graph, the parameter of our generation model is trained based on the node classification performance, which is elaborated in the next section.

We expect that the generated negative links are beneficial for the inference capability of g_ψ. Naturally, the generated negative links should be evaluated through $g_\psi(X, A^*)$ that requires g_ψ to be compatible with A^*. However, as A^* is asymmetric and contains negative elements, most of the previous GCN models, designed for graphs with positive symmetric adjacency matrices, cannot be applied directly. Thus, in the remaining part of this section, we present the modifications on the GCN model of [15] to make it compatible with both the original graph and the graph with negative links.

B. Graph Convolution with Negative Links

Our proposed GCN model is developed by modifying the GCN introduced by [15], which is prevalent due to its stable performance and simplicity. The feature updating rule at layer l of the modified model is described by

$$H^{(l)} = \hat{A}^* \sigma(H^{(l-1)} W^{(l)}) \tag{8}$$

$$\hat{A}^* = \tilde{D}_{\text{out}}^{-1/2} \tilde{A}^* \tilde{D}_{\text{in}}^{-1/2} \tag{9}$$

where $\tilde{A}^* = A^* + I_N$; $\tilde{D}_{\text{in}}$ and $\tilde{D}_{\text{out}}$ are the 'modified' indegree matrix and outdegree matrix of $\tilde{A}^*$. $\tilde{D}_{\text{in}}$ and $\tilde{D}_{\text{out}}$ are diagonal matrices and the i-th diagonal elements of these two matrices are $(\tilde{D}_{\text{in}})_{ii} = \sum_j |\tilde{A}^*_{ji}|$ and $(\tilde{D}_{\text{out}})_{ii} = \sum_j |\tilde{A}^*_{ij}|$, respectively. Here, we use the term 'modified' because with the original definition in graph theory, the indegree matrix and the outdegree matrix are calculated based on the value of the entries of an adjacency matrix rather than the absolute value.

Eq. (9) defines the specific form of $\hat{A}^*$, which concerns the coefficients assigned to the neighbors when smoothing node features. When smoothing the feature vector of v_i, the coefficients of its neighbors are specified by the i-th row of A^* and we have

$$h_i \leftarrow \frac{1}{\sqrt{(\tilde{D}_{\text{out}})_{ii}}} \left(\frac{h_i}{\sqrt{(\tilde{D}_{\text{in}})_{ii}}} + \sum_{j \in \mathcal{N}_i} A^*_{ij} \frac{h_j}{\sqrt{(\tilde{D}_{\text{in}})_{jj}}} \right). \tag{10}$$

In this procedure, a node sends out it feature information through both negative links and positive links. The quantity of the information send out along a link is proportional to the magnitude of this link. Thus, the indegree of a node measures how much information is sent to the other nodes. To make the nodes with different indegrees have a comparable influence on the whole graph, the feature vectors are first normalized by the corresponding node indegrees. This normalization is important for the graph with generated negative links since a node may be the negative neighbor of many other nodes. Then, a node v_i aggregates the normalized features from both positive neighbors and negative neighbors. To keep the scale of the feature vector, the aggregated feature information is normalized by the node outdegree before being used to update h_i.

Remark 2: Eq. (9) can be deemed as a generalized form of Eq. (3) because when applying Eq. (9) to a graph with no negative link, $\hat{A}^*$ will automatically reduce to $\hat{A}$. Therefore, the modified GCN is also applicable to the original graph that contains no negative links.

Eq. (8) specifies the feature transformation procedure from layer $l-1$ to layer l. With the feature transformation designed by [15], namely, Eq. (2), $\hat{A}$ is multiplied with $H^{(l-1)}$ first, which means the input of each layer is smoothed. However, in our method, we smooth the node feature at the feature space of the output. In this way, the GCN output of one node is directly affected by the neighbors and the effects of negative links can be more explicitly identified. Furthermore, we found that this modification does not significantly impact the GCN performance for the graph without the generated negative links based on the empirical results shown in Table I. In this experiment, we build two GCNs which share the same architecture (two layers, 16 hidden units, and ReLU

TABLE I
NODE CLASSIFICATION ACCURACIES (IN PERCENT) OF THE ORIGINAL PROPAGATION MODEL AND OUR MODIFIED PROPAGATION MODEL ON THE GRAPH WITHOUT GENERATED NEGATIVE LINKS

Propagation model	Cora	Citeseer	Pubmed
$H^{(l)} = \sigma(\hat{A}H^{(l-1)}W^{(l)})$	81.4	70.0	77.6
$H^{(l)} = \hat{A}\sigma(H^{(l-1)}W^{(l)})$	81.5	70.1	77.6

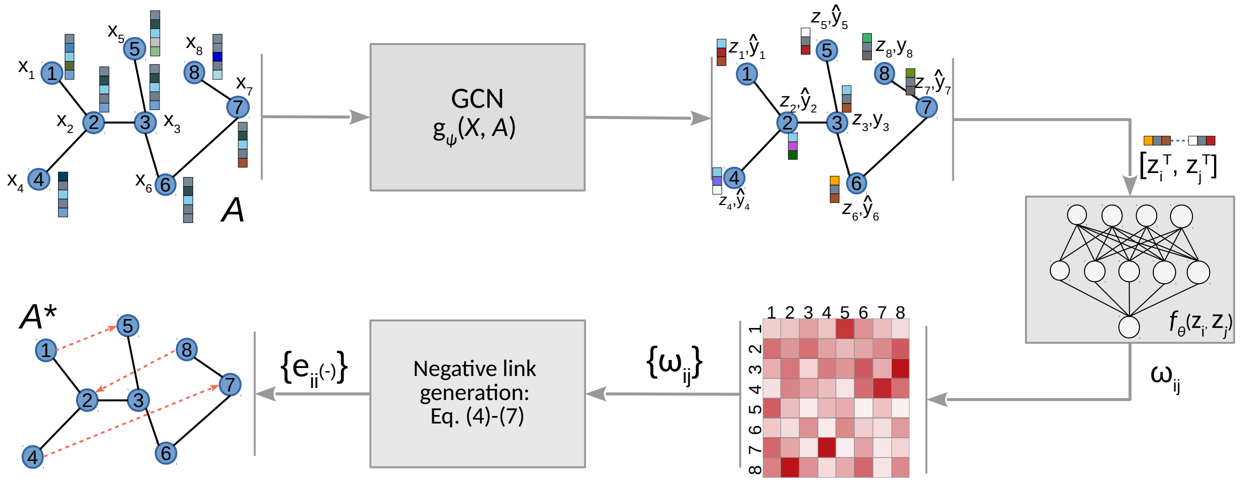


Fig. 2. Illustration of the proposed negative link generation process. The GCN estimates the classes for unlabeled nodes ($\hat{y}_i$) based on the original graph and extracts the node representation vector (z_i) for each node. A fully connected neural network is used to calculate ω_{ij} and its input is set to the concatenation of the node representation vectors. The red squares reflect the magnitude of $\{\omega_{ij}\}$. And a square colored with dark red indicates a large value of the corresponding ω_{ij}. A generated negative link is represented by the red dashed line, which specifies that the node at its head is a negative neighbor of the node at its tail.

activation [31]). The feature propagation models of them are set to Eq. (2) and Eq. (8), respectively. With identical training parameters and initial weights, these two GCNs are trained on three benchmark datasets, *i.e.*, Cora, Citeseer and Pubmed [32] that are specifically introduced in Section V-A. The average test performances of ten independent trials are reported in Table I, which are very close to each other. However, this modification is important to our proposed method, which will be discussed thoroughly in Section V-C.

So far we have introduced the proposed negative link generation mechanism and the modified GCN model for the graph with negative links. In the next section, we design an optimization algorithm to train our model.

IV. Proposed Learning Framework

There are two sets of parameters in our model that need to be trained, which are the weights of g_ψ and the weights of f_θ. In this section, we first present the loss function design and the training algorithm. Then, an analytic discussion is provided to further explain our method.

A. Loss Function and Optimization

With our model, the label distribution for each node can be estimated based on the original graph as well as the graph with generated negative links. The class distribution calculated by $g_\psi(X, A)$ is denoted as $\hat{Y} \in \mathcal{F}$ with $\hat{Y}_{ic}$ being the probability of v_i belonging to class c. And the class probability matrix is written as $\hat{Y}^* \in \mathcal{F}$ if it is calculated by $g_\psi(X, A^*)$. Note that $\hat{Y}^*$ depends on both ψ and θ while $\hat{Y}$ is only related to ψ.

The cross-entropy loss of $\hat{Y}$ and $\hat{Y}^*$ over the labeled samples is

$$\ell_1(\theta, \psi) = -\sum_{i=1}^{M}\sum_{c=1}^{C} Y_{ic}\log \hat{Y}_{ic} - \sum_{i=1}^{M}\sum_{c=1}^{C} Y_{ic}\log \hat{Y}^*_{ic}. \quad (11)$$

By minimizing $\ell_1(\theta, \psi)$ using the gradient for θ, the negative link generation model is tuned to assign large weights to the negative links that are beneficial for reducing the supervised loss. When updating ψ based on $\ell_1(\theta, \psi)$, the inference function $g_\psi(X, A)$ is tuned to make more accurate prediction on the labeled nodes. Meanwhile, ψ is also trained to deal with the graph with negative links since the supervised loss of $\hat{Y}^*$ provides the gradient for ψ as well. Besides Eq. (11), we also add the following cross entropy between the Y^* and Y over the unlabeled samples as the other part of the training loss:

$$\ell_2(\theta, \psi) = -\sum_{i=M+1}^{N}\sum_{c=1}^{C} \hat{Y}_{ic}\log \hat{Y}^*_{ic}. \quad (12)$$

The significance of Eq. (12) can be explained from two aspects. First, with $\ell_2(\theta, \psi)$, the utilization efficiency of the generated negative links can be largely improved. If the training loss is set to $\ell_1(\theta, \psi)$, only a small part of the generated negative links can be utilized for model training, which are the ones within the receptive field of the labeled nodes, while if $\ell_2(\theta, \psi)$ is also included, all the generated negative links participate in the optimization since $\ell_1(\theta, \psi)$ and $\ell_2(\theta, \psi)$ jointly cover all nodes in the graph. Second, $\ell_2(\theta, \psi)$ enables the inference function $g_\psi(X, A)$ to learn from the generated negative links. Minimizing $\ell_2(\theta, \psi)$, the difference between $\hat{Y}$ and $\hat{Y}^*$ are reduced. In this process, the inference function $g_\psi(X, A)$ is trained to generate similar label distribution to $\hat{Y}^*$ which is inferred based on the graph containing negative links.

In a semi-supervised learning problem, there are usually many more unlabeled nodes than the labeled nodes such that $\ell_2(\theta, \psi)$ can be considerably larger than $\ell_1(\theta, \psi)$. Therefore, the overall loss function of our model is set to

$$\mathcal{L}(\psi, \theta) = \frac{1}{M}\ell_1(\psi, \theta) + \frac{\alpha}{N-M}\ell_2(\psi, \theta) \quad (13)$$

Algorithm 1 Training Algorithm

```
Input: $A$, $X$, $\{y_i\}_1^M$
Parameter: $\psi$, $\theta$
Output: Prediction of $\{y_i\}_{M+1}^N$
1: while training( ) do
2:     Calculate $\hat{Y}$ through $g_\psi(X, A)$
3:     Sampling $\{y_i\}_{i=M+1}^N$ based on $\hat{Y}$
4:     Uniformly sampling $T$ nodes in each class as potential
       negative neighbors
5:     Generate $A^*$ through Eq.(4)-(7)
6:     Calculate $\mathcal{L}(\psi, \theta)$ through Eq. (11)-(13)
7:     Update $(\psi, \theta)$ with $(\frac{\partial \mathcal{L}(\psi, \theta)}{\partial \psi}, \frac{\partial \mathcal{L}(\psi, \theta)}{\partial \theta})$
8: end while
9: Calculate $\hat{Y}$ with $g_\psi(X, A)$
10: $\hat{y}_i = \arg\max_{j \leq C} \hat{Y}_{ij}$ for $i = M+1, \ldots, N$
11: return $\{\hat{y}_i\}_{M+1}^N$
```

where $\ell_1(\psi, \theta)$ and $\ell_2(\psi, \theta)$ are normalized by the involved node numbers to make them comparable; and α is a hyper-parameter that controls the portions of these two terms. With the loss function of Eq. (13), the model can be trained by the backpropagation algorithm. However, when generate A^*, the generation process of Eq. (4)-(7) searches the negative neighbors for one node from the whole graph, which can be of low efficiency. To reduce the computation complexity, we set the search space to a small subset of the nodes rather than the whole graph. Specifically, in the negative link generation process, after obtaining the estimation of the node classes, we randomly sample T nodes ($T \ll N$) in each class, which are used as the potential negative neighbors for that class. The reason for this operation is that there are no ground-truth negative neighbors for one node, and it is unlikely to find a very special negative neighbor by searching the whole graph. Consequently, the optimization procedure of our model can be summarized by Algorithm. 1. Note that at the early stage of the algorithm, it is possible that the nodes been assigned to a class is less than T because g_ψ is not well trained and can generate very biased classification. In this case, we duplicate the nodes in that class until the total number is greater than T such that the consistency of our algorithm can be kept.

Our algorithm updates ψ and θ simultaneously based on the gradients of Eq. (13). This follows the optimization approach applied by [33], which solves the semi-supervised learning problem for non graph-structured data. An alternative way to optimize a model that contains two sets of parameters is updating these two parameters iteratively. However, this approach can increase the complexity of model implementation and introduce more hyperparameters that need tuning, for instance, the length of the iteration. In most cases, this iterative optimization approach is preferred to be avoided. For instance, in [34], even if the trainable parameters have the min-max relationship, the authors still avoid the iterative training process by utilizing a gradient reversal layer. Therefore, we apply the joint optimization instead of the iterative optimization scheme. In this way, the weights of $f_\theta(\cdot)$ do suffer some oscillations, but it does not affect the stability of the GCN model, which is shown in Fig. 3. The details are provided in Section V.B. And the objective of our method is not to generate negative links, instead, we aim to improve the performance of the inference GCN model by the generated negative links. Therefore, the joint optimization scheme is reasonable for our model.

B. Discussion

In our method, the final predictions of the node labels are provided by the inference model *i.e.*, $g_\psi(X, A)$, which takes the original graph as input, and the generated negative links are used to provide extra information to improve the performance of the inference function. Similar approaches can be found in prior works, such as [20] and [23]. In [20], the proposed model contains two GCNs with one inferring the node labels based on the node features and the other one providing the label dependencies that is utilized to improve the performance of the inference model. In [23], a link generation model is proposed where a fully connected neural network is used to generate labels for each node at first, and then the node features and the generated labels are together employed to generate the links. This process is trained to maximize the likelihood of the observed links and minimize the likelihood of the nonexistent links. With the approach of [23], a GCN-based recognition model is used to predict the labels of the nodes, which is trained based on the labeled nodes as well as the node labels generated by the fully connected neural network of the link generation process.

Unlike the GCNs [26], [27] designed for the signed graph, which has separate kernels for positive links and negative links, the positive links and the negative links operate on the common node features. This is because the negative links generated by our method represent a negative correlation of the features of the connected nodes. Moreover, the generated negative links are utilized to improve the performance of the inference model as the weights of the GCN is trained on both the original graph and the graph with generated negative links. If we introduce another specialized kernel for the negative links, the generated negative links cannot be informative for the inference model and the connection between the generated negative links and the inference model cannot be built. This is also the reason that our method can not be applied to the signed graphs. The negative links in a signed graph can convey complex relations [26]. Applying our model directly to the signed graph will treating the negative links as the negation of positive links, which is an incorrect assumption [35].

The proposed negative link generation process creates one negative link for each node, which is conservative. In our current work, we do not generate multiple negative links for one node since there is no prior statistic model for the number of negative links and it is relatively safe to just generate the one that we are most confident about. Although experiment results in Section V-B demonstrate that our method can achieve state-of-the-art performance in most settings, a generation mechanism that creates multiple negative links for one node still deserves consideration, which is planned for future work.

The inference function $g_\psi(X, A)$ is a GCN working on the original graph and the computation complexity of layer l is $\mathcal{O}(|\mathcal{E}|K^{(l)}K^{(l-1)})$ where $|\mathcal{E}|$ is the total number of the links in the original graph [15]. For the graph with generated negative links, the total link number is $|\mathcal{E}|+N$ as there is one negative link for each node. Thus, the computation complexity for the layer l of $g_\psi(X, A^*)$ is $\mathcal{O}((|\mathcal{E}|+N)K^{(l)}K^{(l-1)})$ which is slightly higher than that of $g_\psi(X, A)$ due to the increasing of the link number. The original computation cost for generating negative links is $\mathcal{O}(N^2)$ since we need to compute ω_{ij} for $1 \leq i, j \leq N$. But, in our proposed training algorithm, it is effectively lowered to $\mathcal{O}(NT)$ by randomly sampling the potential negative neighbors. And since usually we have $N \gg T$, the computation cost for negative link generation is significantly reduced.

Our method can be alternatively regarded as a data-augmentation technique since the weights of the GCN, *i.e.*, ψ, is trained based on both the graph with negative links and the original graph. In this way, the GCN is trained on more data so that the generalization performance can be effectively enhanced. From this perspective, our method shares some similarities with the adversarial training algorithms [17], [18]. In a adversarial training algorithms, besides the supervised loss of the original training data, the loss function also contains the divergence between the prediction based on the original data and the prediction based on the data with adversarial perturbations. The adversarial perturbations are added on the node features to provide data augmentation. In comparison, our method achieves the data augmentation by modifying the graph topology, *i.e.*, adding negative links.

Note that our method is different from the negative sampling method applied by [36] and [37]. The negative sampling method is proposed in [38], which is used to solve the word representation learning problem. In this problem, the objective is to learn an effective word representation that can be used to make a prediction for the target word based on the context. Specifically, the prediction function can be written as

$$p(w|c) = \frac{\exp(v_w^{\mathrm{T}} v_c)}{\sum_{i=1}^{W} \exp(v_i^{\mathrm{T}} v_c)}$$

where $p(w|c)$ denotes the probability of the target word(w) occurring in the context (c), v_w is the representation of w, v_c is the representation of c, and the denominator is the partition function(the summation over the whole vocabulary). To learn high-quality word representations, the likelihood of the training set should be maximized. But since the partition function in the denominator involves the calculation over the whole vocabulary, the computation and the optimization can be expensive. In this case, the negative sampling method is proposed as an approximation [38]. The key idea is to use some sampled non-target data to approximate the effect of the partition function. When applying this method to the graph representation learning [36], [37], the context is the node neighbors, the original partition function is calculated over the whole graph, and the corresponding negative samples are the non-connected nodes. Interested readers can refer to [39] for a detailed review of the negative sampling method. However, in our method, the negative link of a node defines a negative neighbor of this node, whose features are negatively correlated. Moreover, similar to the word representation learning problem, the negative sampling method is usually applied to unsupervised graph representation learning. For instance, the GraphSage [37] model only includes the negative sampling term in the loss function when it is applied to the unsupervised learning problem. While our method is designed for the semi-supervised learning problem and the training node labels are necessary for our method.

V. Experiments

In this section, we empirically assess the performance of our proposed method on three widely used benchmark datasets in different settings. Moreover, we also design some auxiliary experiments to analyze our model.

A. Experiment Settings

We test the performance of our method on three citation networks, *i.e.*, Cora, Citeseer, and Pubmed [32]. In these datasets, nodes represent documents that are categorized into different classes; the feature of a node is a bag-of-words vector; and two nodes are connected by a bidirectional link if either one occurs in the citation list of the other. The basic statistics of these three citation networks are listed in Table II. For each dataset, 20 labels per class are provided for training; 500 labels are used for validation, and 1000 labels are used for testing. We strictly follow the data split protocol of [40], which is commonly applied by most of the prior methods. The performance of the GCN models are evaluated based on the testing accuracy in different settings. The following comparative methods are included for the performance evaluation:

TABLE II
Dataset Statistics

Datasets	Nodes	Links	Features	Classes
Cora	2708	5429	1433	7
Citeseer	3327	4732	3703	6
Pubmed	19717	44338	500	3

- GCN [15]: it is the basic form of GCN where the feature propagation coefficients of the neighbors are determined by the graph topology.
- GAT [16]: it assigns different coefficients to the neighbors of a node based on their features in the feature smoothing process.
- Bayesian GCN [22]: this method generates multiple graph topologies and utilizes the generated topologies to infer the node labels with a Bayesian approach.
- LSM-GAT and SBM-GCN [23]: it trains the generative models (LSM and SBM) based on the observed graph topology, and the node labels created by the generative model is exploited to improve the performance of the GCN-based recognition model.
- DisenGCN [21]: it aims to extract node features that belong to disentangled channels; at each channel, the node feature is jointly determined by the feature vectors of the neighborhood and the coefficient of a neighbor

reflects the extent of they being connected due to this channel.

- GMNN [20]: this model consists of two GCNs with one functioning as the recognition function and the other being used to model the label dependencies; these two GCNs learns interactively through the unlabeled nodes.
- O-BVAT [17]: it creates adversarial training samples by adding random perturbations on node features, and the GCN is trained on the adversarial samples to improve the generalization performance.
- GGP-X [41]: it is a graph Gaussian process that infers the node feature based on the neighbors with a Bayesian approach.
- GIL [42]: this method infers the label of a query node based on the nodes in the reference set; the similarity score between a query node and a reference node is calculated according to their properties, which is used to evaluate the probability of sharing the same label.

In all experiments, the hyperparameters of our model are kept the same. The GCN in our model consists of two layers ($L = 2$) with 16 hidden units, *i.e.* $K^{(1)} = 16$. The ReLU activation function is applied at the output of the first layer. Dropout [43] is applied at the input of the two layers and the dropout rate is set to 0.5. These parameters above follow the classic settings of prior works [15], [17], [20], [22]. $f_\theta(z_i, z_j)$ is instantiated by a three-layer fully connected neural network, where the sigmoid activation function is applied at the hidden layer and there is no activation function at the output layer. The hidden units number $K^{(f)}$ for $f_\theta(z_i, z_j)$, the sampling volume T for each class in **Algorithm** 1, and the coefficient α in Eq. (13) are jointly searched according to the classification performance on the validation set of Cora. The search space is specified as follows: $K^{(f)} \sim [8, 16, 32, 64]$, $D \sim [32, 64, 128, 256]$, $\alpha \sim [0.6, 0.8, 1.0, 1.2, 1.4]$. According to the average validation accuracy of 20 trials in each setting, these parameters are set to $K^{(f)} = 16$, $T = 128$, $\alpha = 0.8$. The RMSProp optimizer [44] is employed to train the model where the initial learning rate and the weight decay coefficient are set to 0.05 and 5e-4, respectively.

B. Node Classification Performance

The node classification performance of our method is evaluated by the testing accuracy with different experiment settings. With the first setting, we use the samples of the training set to train our model and the validation set is utilized to early stop the training. This setting is applied by numerous previous works and the comparison with these methods is shown in Table III. The performance of the comparative methods are directly copied from the original literature and the results of our model are the average classification accuracies of 50 trials. In Table III, the result with a bold marker is the best performance on that dataset. We can see that our method achieves the best performance on all datasets. Moreover, the performances of our model on Cora and Citeseer past t-test with the significance level of 0.001 compared with the second-best methods, which are marked by '$\star$' in Table III. On Pubmed, the p-value for the t-test of our method is 0.002.

TABLE III
CLASSIFICATION ACCURACIES (%) WITH THE STANDARD DATA SPLIT. THE **BOLD MARKER** DENOTES THE BEST PERFORMANCE ON THAT DATASET, AND THE STAR MARKER ($\star$) MEANS THE BEST ALGORITHM PASTS T-TEST COMPARED WITH THE SECOND-BEST ALGORITHM WITH THE SIGNIFICANCE LEVEL OF 0.001. THE ($\pm$) ERROR BAR DENOTES THE STANDARD DEVIATION AND FOR SOME OF THE METHODS, THIS TERM IS OMITTED FOR NOT BEING REPORTED BY THE ORIGINAL LITERATURE.

Algorithm	Cora	Citeseer	Pubmed
GCN	81.5	70.3	79.0
GAT	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
Bayesian GCN	81.2 ± 0.8	72.2 ± 0.6	76.6 ± 0.7
LSM_GAT	82.9 ± 0.3	73.1 ±0.5	77.6 ± 0.7
SBM_GCN	82.9 ± 0.3	74.0 ± 0.3	77.4± 0.4
DisenGCN	83.7	73.4	80.5
GMNN	83.7	73.1	81.8
O-BVAT	83.6 ± 0.5	74.0 ± 0.6	79.9 ± 0.4
Ours	***84.4** ± 0.5	***74.8** ± 0.4	**82.2** ± 0.5

The second experiment setting allows us to use the samples in the validation set for training, which is also adopted by the following two papers: [41] (GGP-X) and [42] (GIL). We compare our method with the results claimed in these two papers. Since we use the validation set to train our model, we no longer do early stopping during training. Instead, we train our model for 200 epochs in each trial and report the average results of 50 trials in Table IV. It can be observed that our method outperforms others for all three datasets and the improvements are statistically significant.

TABLE IV
CLASSIFICATION ACCURACIES (%) WITH VALIDATION SET BEING UTILIZED FOR TRAINING. THE **BOLD MARKER** AND THE STAR MARKER ($\star$) ARE CONSISTENT WITH THE DEFINITIONS IN TABLE III

Algorithm	Cora	Citeseer	Pubmed
GGP-X	84.7	75.6	82.4
GIL	86.2	74.1	83.1
Ours	***87.1** ± 0.4	***76.5** ± 0.3	***84.0**± 0.4

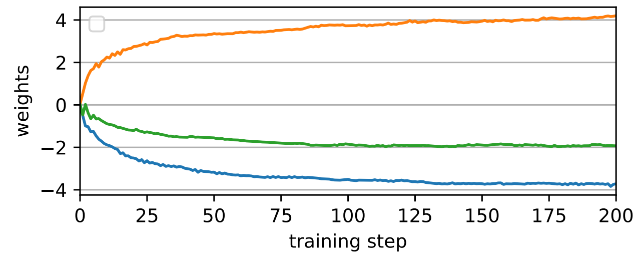
We also test our method under the data-scarce situation where the training sample is reduced to the half, *i.e.*, ten samples for each class. This experiment setting is also applied by [22] (Bayesian GCN) and [23] (LSM-GAT and SBM-GCN). In this experiment, we randomly select 10 samples for each class to create a reduced-label training set. Then 10 independent trials are conducted for this training set. We repeat this process five times, such that 50 trials in total are implemented with the reduced-label setting. The average testing accuracies are reported in Table V. We can see that our method significantly outperforms the other two algorithms.

TABLE V
CLASSIFICATION ACCURACIES (%) WITH REDUCED-LABEL TRAINING. THE **BOLD MARKER** AND THE STAR MARKER ($\star$) ARE CONSISTENT WITH THE DEFINITIONS IN TABLE III.

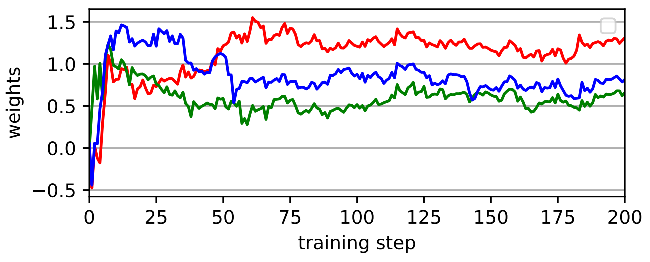
Algorithm	Cora	Citeseer	Pubmed
Bayesian GCN	76.6 ± 0.8	70.8 ± 0.6	72.3 ± 0.8
LSM-GAT	79.2 ± 0.4	69.1 ± 0.4	69.9 ± 0.3
SBM-GCN	78.0 ± 0.2	70.3 ± 0.6	71.0 ± 0.4
Ours	***82.5** ± 0.9	***71.8** ± 0.7	***75.0** ± 0.6

C. Model Analysis

Now we conduct some auxiliary experiments to analyze our method. In our model, the parameter of the negative link generator (θ) and the GCN model (ψ), are jointly trained. To verify the stableness of our model, we plot the evolution trajectories of the weights for the GCN model and the negative link generator during the training on the Cora dataset, which is shown in Fig. 3. We sampled three components from ψ and θ respectively and plot the corresponding trajectories in Fig. 3 (a) and (b). From Fig.3 (b), it can be observed that the negative link generator suffers certain oscillations. The possible reason is that the negative neighbors for each node are generated through random sampling in our training process. However, the weights of the GCN model evolve smoothly, which is shown in Fig. 3 (a). This phenomenon can be explained as: there is only one generated link for each node and the effect of the negative links is much less than the original positive links; second, the generated negative links are normalized through Eq. (7), and accordingly, the fluctuation from the generator can be attenuated for the GCN.



(a)



(b)

Fig. 3. Evolution of the sampled weights in our model during the training on Cora: (a) sampled weights from ψ; (b) sampled weights from θ.

In our model, the feature propagation operation is moved from the input to the output for each layer. This modification has little influence when the GCN deals with the original graph where there are no negative links, validated by Table I. However, this modification is important for our method. To demonstrate this point, we implement our method with both the original propagation model and our modified propagation model. The average testing results of 10 independent trials on three benchmark datasets are reported in Table VI. It can be observed that the modified feature propagation model consistently improves the performance of our method over the three datasets.

TABLE VI
COMPARISON OF NODE CLASSIFICATION ACCURACIES (%) OF OUR METHOD UNDER DIFFERENT FEATURE PROPAGATION MODEL

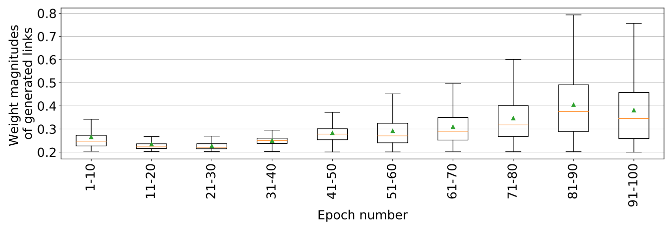
Propagation model	Cora	Citeseer	Pubmed
$H^{(l)} = \sigma(\hat{A}^* H^{(l-1)} W^{(l)})$	83.6 ± 0.5	72.9 ± 0.6	81.6 ± 0.4
$H^{(l)} = \hat{A}^* \sigma(H^{(l-1)} W^{(l)})$	84.4 ± 0.5	74.8 ± 0.4	82.2 ± 0.5

In our method, the label distribution calculated by $g_\psi(X, A)$ and the label distribution calculated by $g_\psi(X, A^*)$ are regularized to be similar to each other by minimizing their cross entropy over the unlabeled nodes, which means these two models learn interactively. If we substitute $\hat{Y}^*$ with $\hat{Y}$ for Eq. (11) and (12), our method will convert to a self-training algorithm [45], which annotates the unlabeled nodes with its prediction and use these annotated nodes to train itself. The comparison of our method and the self-training counterpart is shown in Table VII. We see that our method is able to generate better results than the self-training algorithm, which demonstrates the benefit of the negative links.

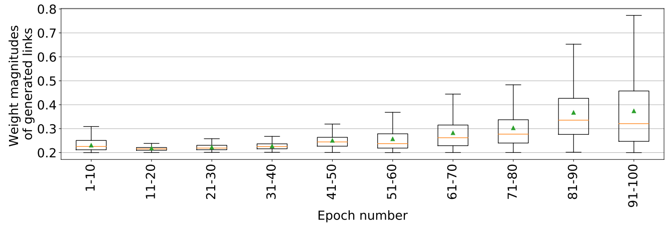
TABLE VII
COMPARISON OF NODE CLASSIFICATION ACCURACIES (%) WITH SELF-TRAINING

Algorithm	Cora	Citeseer	Pubmed
Self-training	82.8 ± 0.4	73.0 ± 0.5	80.6 ± 0.4
Ours	84.4 ± 0.5	74.8 ± 0.4	82.2 ± 0.5

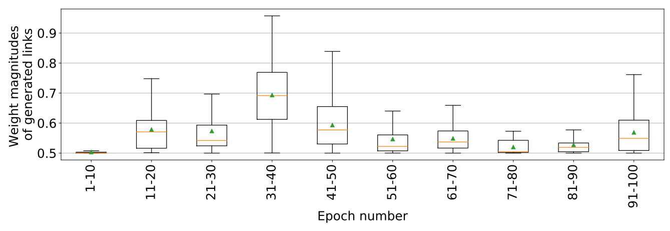
The next experiment aims to inspect the learning process of the negative link generation model. For this purpose, we check the weights of the generated negative links during the training process, which is illustrated by Fig. 4. In this figure, each boxplot shows the statistic of the magnitudes of negative links generated in 10 consecutive training epochs, where the box specifies the first quartile and the third quartile; the orange bar in the box shows the median; the whiskers under and above the box extend to the minimum and the maximum; and the green triangle indicates the mean. For the Cora dataset, there are six representatives ($v_{i_c^{(-)}}$) of the potential negative neighbors for a node since the total class number is seven and we do not seek negative neighbors from the same class. If ω_{ij} for all potential negative neighbors are similar, the magnitude of the generated negative link will be low and the minimum is 0.167. This corresponds to the situation where our negative link generation model cannot tell which potential neighbor is better than the others. From Fig. 4(a), we see that at the beginning, the weights of the generated negative links are very low while as the training going, the generated links tend to have large weights which indicates our model learns to find the special node among other potential negative neighbors. A similar phenomenon can be found for the Citeseer dataset from Fig. 4(b). However, base on Fig. 4(c), the learning procedure is not ideal for the Pubmed dataset, which may also be the potential reason for the limited performance improvement of our method on this dataset. It is possible that learning a negative link pattern is more difficult on Pubmed since it has many more nodes and edges, which can consequently lead to more complex node relations.



(a)



(b)



(c)

Fig. 4. Boxplots of the magnitude statistics of generated negative links in every 10 consecutive training epochs for three datasets: (a) Cora, (b) Citeseer, (c) Pubmed. The box specifies the first quartile and the third quartil. The orange bar in the box shows the median. The whiskers under and above the box extend to the minimum and the maximum. The green triangle indicates the mean.

We present the visualization of the node feature extracted by the first layer of our GCN model for the testing samples. The corresponding t-SNE [46] embedding is plotted in Fig. 5. For the three datasets, the node feature extracted by our model is plotted at the right column of three subfigures with Fig. 5(a), Fig. 5(c), and Fig. 5(c) respectively corresponding to Cora, Citeseer, and Pubmed. We also provide the visualizations of the features extracted by the GCN model from [15] as a comparison, which lie at the left column in Fig. 5. In each subfigure, the nodes of different classes are represented by different colors. It can be observed that for Cora and Citeseer, our method is able to extract more discriminative features than the GCN baseline. For Pubmed, the improvement is not that obvious. This phenomenon is consistent with the observation from Fig. 4(c). Thus, improving the performance of our method on large scale datasets will be the focus of our future work.

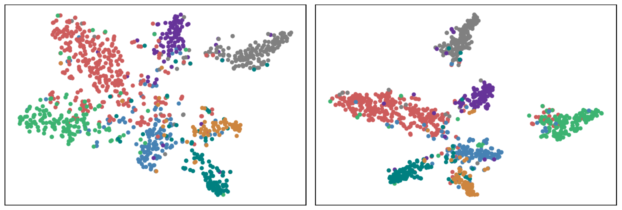
We also modify our model to apply it on the link prediction task. For this purpose, we apply the inner product link prediction model introduced by [47]:

$$\hat{A}_{ij} = \sigma(z_i^T z_j) \tag{14}$$

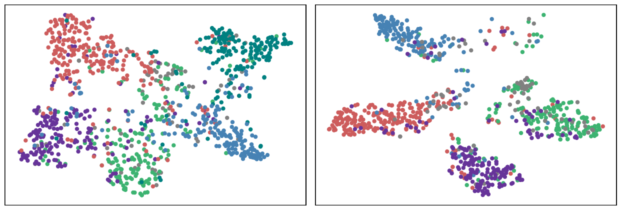
where $\hat{A}_{ij}$ is the probability of the link between node i and node j exists, $\sigma(\cdot)$ is the sigmoid function, and $z_i^{\mathrm{T}} z_j$ is the inner product of the node features. It can be observed that the link prediction only depends on the node feature with this prediction model. Thus, a good performance also denotes the high quality of the learned features. To apply our model to link prediction task, the loss function of our model is modified as

$$\mathcal{J}(\psi, \theta) = \mathcal{L}(\psi, \theta) + \sum_{A_{ij} \in A} A_{ij} \log(\hat{A}_{ij}) \tag{15}$$

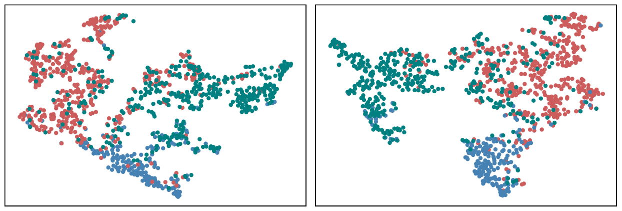
where the first term is the loss of Eq. (13) and the second term denotes the cross entropy loss of the predicted links. We follow the dataset modification scheme of [47] for the evaluation of the link prediction performance. Specifically, 5% links are removed from the original citation dataset and these links are used as validation set; another 10 % links are removed and used as the test set. In the test and validation sets, we also randomly sample some unconnected node pairs as the non-edges case and the number of non-edges case is equal to the corresponding edge number. The experiment results are listed in Table VIII. It can be observed that our method performs better comparing the link prediction model based on the standard GCN model [47] (GAE) on the Cora and Citeseer dataset. And since we apply the inner product link prediction model and the prediction is only determined by the node features, this demonstrate that our method can generate better nodes features, which is the major purpose of our work.



(a)



(b)



(c)

Fig. 5. T-SNE embedding plot of the feature extracted by the first layer of the GCNs on three datasets: (a) Cora, (b) Citeseer, (c) Pubmed. The plots lie at the left column exhibit the node features extracted by the original GCN model of [15]. The plots lie at the right column exhibit the node features extracted by our model.

TABLE VIII
LINK PREDICTION RESULTS FOR THE CITATION NETWORK. AUC IS THE AREA UNDER THE ROC CURVE AND AP REPRESENTS THE AVERAGE PRECISION SCORES

Algorithm	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
GAE	91.0± 0.02	92.0 ± 0.03	89.5 ± 0.04	89.9 ± 0.05	**96.4** ± 0.00	**96.5** ± 0.00
Ours	**91.5** ± 0.02	**92.6** ± 0.02	**90.4** ± 0.02	**91.5** ± 0.01	96.3 ± 0.01	96.5 ± 0.01

VI. Conclusion

Most of the previous GCN-based methods for semi-supervised node classification depends on the positive correlation of the connected nodes. In this paper, we have introduced a novel learning approach that learns to infer node labels by further considering the inverse correlated nodes. A trainable negative link generation model has been proposed to generate negative links that are beneficial for the classification performance. To avoid searching neighbors from the whole dataset, which is of high computation cost, we have designed a high-efficiency training algorithm that randomly searches negative neighbors in a small subset of nodes. Currently, our method conservatively generates one neighbor of each node while a generation mechanism that creates multiple negative links for one node is planned for future work. Experiments on three widespread benchmark datasets have demonstrated that our method can generate better or competitive performance compared with state-of-the-art methods. We have found that the performance improvement of our method on the large scale dataset appears limited compared with the performance on the small dataset. Therefore, we will also focus on improving our method on large scale datasets in the future.

References

[1] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.

[2] M. Al Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social network data analytics*. Springer, 2011, pp. 243–275.

[3] K. Zhan, C. Zhang, J. Guan, and J. Wang, "Graph learning for multiview clustering," *IEEE transactions on cybernetics*, vol. 48, no. 10, pp. 2887–2895, 2017.

[4] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 3697–3707.

[5] F. Xiong, X. Wang, S. Pan, H. Yang, H. Wang, and C. Zhang, "Social recommendation with evolutionary opinion dynamics," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2018.

[6] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 6348–6358.

[7] H. He and H. Jiang, "Deep learning based energy efficiency optimization for distributed cooperative spectrum sensing," *IEEE Wireless Communications*, vol. 26, no. 3, pp. 32–39, 2019.

[8] L. Yang, X. Cao, D. Jin, X. Wang, and D. Meng, "A unified semi-supervised community detection framework using latent space graph regularization," *IEEE transactions on cybernetics*, vol. 45, no. 11, pp. 2585–2598, 2014.

[9] Z. Chen, X. Li, and J. Bruna, "Supervised community detection with line graph neural networks," *arXiv preprint arXiv:1705.08415*, 2017.

[10] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Advances in neural information processing systems*, 2004, pp. 321–328.

[11] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of machine learning research*, vol. 7, no. Nov, pp. 2399–2434, 2006.

[12] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *arXiv preprint arXiv:1812.04202*, 2018.

[13] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[14] X. Zhou, F. Shen, L. Liu, W. Liu, L. Nie, Y. Yang, and H. T. Shen, "Graph convolutional network hashing," *IEEE transactions on cybernetics*, 2018.

[15] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[17] Z. Deng, Y. Dong, and J. Zhu, "Batch virtual adversarial training for graph convolutional networks," *arXiv preprint arXiv:1902.09192*, 2019.

[18] F. Feng, X. He, J. Tang, and T.-S. Chua, "Graph adversarial training: Dynamically regularizing based on graph structure," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[19] S. Pan, R. Hu, S.-f. Fung, G. Long, J. Jiang, and C. Zhang, "Learning graph embedding with adversarial training methods," *IEEE transactions on cybernetics*, 2019.

[20] M. Qu, Y. Bengio, and J. Tang, "Gmnn: Graph markov neural networks," *arXiv preprint arXiv:1905.06214*, 2019.

[21] J. Ma, P. Cui, K. Kuang, X. Wang, and W. Zhu, "Disentangled graph convolutional networks," in *International Conference on Machine Learning*, 2019, pp. 4212–4221.

[22] Y. Zhang, S. Pal, M. Coates, and D. Ustebay, "Bayesian graph convolutional neural networks for semi-supervised classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5829–5836.

[23] J. Ma, W. Tang, J. Zhu, and Q. Mei, "A flexible generative framework for graph-based semi-supervised learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 3276–3285.

[24] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[25] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[26] T. Derr, Y. Ma, and J. Tang, "Signed graph convolutional networks," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 929–934.

[27] X. Shen and F.-L. Chung, "Deep network embedding for graph representation learning in signed networks," *IEEE transactions on cybernetics*, 2018.

[28] P. Massa and P. Avesani, "Controversial users demand local trust metrics: An experimental study on epinions. com community," in *AAAI*, 2005, pp. 121–126.

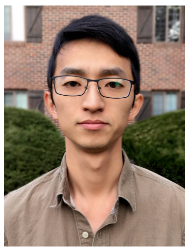
[29] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," *arXiv preprint arXiv:1902.07153*, 2019.

[30] P. D. Hoff, A. E. Raftery, and M. S. Handcock, "Latent space approaches to social network analysis," *Journal of the american Statistical association*, vol. 97, no. 460, pp. 1090–1098, 2002.

[31] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[32] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[33] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, “Semi-supervised learning with deep generative models,” in *Advances in neural information processing systems*, 2014, pp. 3581–3589.
[34] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, “Domain-adversarial training of neural networks,” *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.
[35] J. Tang, X. Hu, and H. Liu, “Is distrust the negation of trust? the value of distrust in social media,” in *Proceedings of the 25th ACM conference on Hypertext and social media*, 2014, pp. 148–157.
[36] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.
[37] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *arXiv preprint arXiv:1706.02216*, 2017.
[38] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *arXiv preprint arXiv:1310.4546*, 2013.
[39] Z. Yang, M. Ding, C. Zhou, H. Yang, J. Zhou, and J. Tang, “Understanding negative sampling in graph representation learning,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1666–1676.
[40] Z. Yang, W. W. Cohen, and R. Salakhutdinov, “Revisiting semi-supervised learning with graph embeddings,” *arXiv preprint arXiv:1603.08861*, 2016.
[41] Y. C. Ng, N. Colombo, and R. Silva, “Bayesian semi-supervised learning with graph gaussian processes,” in *Advances in Neural Information Processing Systems*, 2018, pp. 1683–1694.
[42] C. Xu, Z. Cui, X. Hong, T. Zhang, J. Yang, and W. Liu, “Graph inference learning for semi-supervised classification,” *arXiv preprint arXiv:2001.06137*, 2020.
[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[44] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.
[45] K. Nigam and R. Ghani, “Analyzing the effectiveness and applicability of co-training,” in *Proceedings of the ninth international conference on Information and knowledge management*, 2000, pp. 86–93.
[46] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.
[47] T. N. Kipf and M. Welling, “Variational graph auto-encoders,” *arXiv preprint arXiv:1611.07308*, 2016.



He Jiang (S’17) received the B.S. degree from North China Electric Power University, Beijing, China, in 2012, the M.S. degree from the School of Electrical and Electronics Engineering, Huazhong University of Science and Technology, Wuhan, China, in 2015, and the Ph.D. degree in electrical engineering from University of Rhode Island in 2021. His research interests include adaptive dynamic programming, reinforcement learning, optimal control, machine learning, data mining, and various applications.

Haibo He (SM’11-F’18) received the B.S. and M.S. degrees in electrical engineering from the Huazhong University of Science and Technology in 1999 and 2002, respectively, and the Ph.D. degree in electrical engineering from Ohio University in 2006. He is currently the Robert Haas Endowed Chair Professor at the Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island. His research interests include computational intelligence, machine learning, data mining, and various applications. He is currently the Editor-in-Chief of the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.