

AM-GCN: Adaptive Multi-channel Graph Convolutional Networks

Xiao Wang
Beijing University of Posts and
Telecommunications
xiaowang@bupt.edu.cn

Meiqi Zhu
Beijing University of Posts and
Telecommunications
zhumeiqi@bupt.edu.cn

Deyu Bo
Beijing University of Posts and
Telecommunications
bodeyu@bupt.edu.cn

Peng Cui
Tsinghua University
cuip@tsinghua.edu.cn

Chuan Shi*
Beijing University of Posts and
Telecommunications
shichuan@bupt.edu.cn

Jian Pei
Simon Fraser University
jpei@cs.sfu.ca

ABSTRACT

Graph Convolutional Networks (GCNs) have gained great popularity in tackling various analytics tasks on graph and network data. However, some recent studies raise concerns about whether GCNs can optimally integrate node features and topological structures in a complex graph with rich information. In this paper, we first present an experimental investigation. Surprisingly, our experimental results clearly show that the capability of the state-of-the-art GCNs in fusing node features and topological structures is distant from optimal or even satisfactory. The weakness may severely hinder the capability of GCNs in some classification tasks, since GCNs may not be able to adaptively learn some deep correlation information between topological structures and node features. Can we remedy the weakness and design a new type of GCNs that can retain the advantages of the state-of-the-art GCNs and, at the same time, enhance the capability of fusing topological structures and node features substantially? We tackle the challenge and propose an adaptive multi-channel graph convolutional networks for semi-supervised classification (AM-GCN). The central idea is that we extract the specific and common embeddings from node features, topological structures, and their combinations simultaneously, and use the attention mechanism to learn adaptive importance weights of the embeddings. Our extensive experiments on benchmark data sets clearly show that AM-GCN extracts the most correlated information from both node features and topological structures substantially, and improves the classification accuracy with a clear margin.

KEYWORDS

Graph convolutional networks, network representation learning, deep learning

*Corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).
KDD '20, August 23–27, 2020, Virtual Event, USA

© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7998-4/20/08...\$15.00
<https://doi.org/10.1145/3394486.3403177>

ACM Reference Format:

Xiao Wang, Meiqi Zhu, Deyu Bo, Peng Cui, Chuan Shi, and Jian Pei. 2020. AM-GCN: Adaptive Multi-channel Graph Convolutional Networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining USB Stick (KDD '20), August 23–27, 2020, Virtual Event, USA*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3394486.3403177>

1 INTRODUCTION

Network data is ubiquitous, such as social networks, biology networks, and citation networks. Recently, Graph Convolutional Networks (GCNs), a class of neural networks designed to learn graph data, have shown great popularity in tackling graph analytics problems, such as node classification [1, 31], graph classification [7, 37], link prediction [13, 36] and recommendation [6, 34].

The typical GCN [14] and its variants [11, 16, 27, 30, 36] usually follow a message-passing manner. A key step is feature aggregation, i.e., a node aggregates feature information from its topological neighbors in each convolutional layer. In this way, feature information propagates over network topology to node embedding, and then node embedding learned as such is used in classification tasks. The whole process is supervised partially by the node labels. The enormous success of GCN is partially thanks to that GCN provides a fusion strategy on topological structures and node features to learn node embedding, and the fusion process is supervised by an end-to-end learning framework.

Some recent studies, however, disclose certain weakness of the state-of-the-art GCNs in fusing node features and topological structures. For example, Li *et al.* [15] show that GCNs actually perform the Laplacian smoothing on node features, and make the node embedding in the whole network gradually converge. Ni and Maehara [20] and Wu *et al.* [30] prove that topological structures play the role of low-pass filtering on node features when the feature information propagates over network topological structure. Gao *et al.* [8] design a Conditional Random Field (CRF) layer in GCN to explicitly preserve connectivity between nodes.

What information do GCNs really learn and fuse from topological structures and node features? This is a fundamental question since GCNs are often used as an end-to-end learning framework. A well informed answer to this question can help us understand the capability and limitations of GCNs in a principled way. This motivates our study immediately.

As the first contribution of this study, we present experiments assessing the capability of GCNs in fusing topological structures and node features. Surprisingly, our experiments clearly show that the fusion capability of GCNs on network topological structures and node features is clearly distant from optimal or even satisfactory. Even under some simple situations that the correlation between node features/topology with node label is very clear, GCNs still cannot adequately fuse node features and topological structures to extract the most correlated information (shown in Section 2). The weakness may severely hinder the capability of GCNs in some classification tasks, since GCNs may not be able to adaptively learn some correlation information between topological structures and node features.

Once the weakness of the state-of-the-art GCNs in fusion is identified, a natural question is, “*Can we remedy the weakness and design a new type of GCNs that can retain the advantages of the state-of-the-art GCNs and, at the same time, enhance the capability of fusing topological structures and node features substantially?*”

A good fusion capability of GCNs should substantially extract and fuse the most correlated information for classification task, however, one biggest obstacle in reality is that the correlation between network data and classification task is usually very complex and agnostic. The classification can be correlated with either the topology, or node features, or their combinations. This paper tackles the challenge and proposes an adaptive multi-channel graph convolutional networks for semi-supervised classification (AM-GCN). The central idea is that we learn the node embedding based on node features, topological structures, and their combinations simultaneously. The rationale is that the similarity between features and that inferred by topological structures are complementary to each other and can be fused adaptively to derive deeper correlation information for classification tasks.

Technically, in order to fully exploit the information in feature space, we derive the k -nearest neighbor graph generated from node features as the feature structural graph. With the feature graph and the topology graph, we propagate node features over both topology space and feature space, so as to extract two specific embeddings in these two spaces with two specific convolution modules. Considering the common characteristics between two spaces, we design a common convolution module with a parameter sharing strategy to extract the common embedding shared by them. We further utilize the attention mechanism to automatically learn the importance weights for different embeddings, so as to adaptively fuse them. In this way, node labels are able to supervise the learning process to adaptively adjust the weight to extract the most correlated information. Moreover, we design the consistency and disparity constraints to ensure the consistency and disparity of the learned embeddings.

We summarize our main contributions as follows:

- We present experiments assessing the capability of GCNs in fusing topological structures and node features and identify the weakness of GCN. We further study the important problem, i.e., how to substantially enhance the fusion capability of GCN for classification.
- We propose a novel adaptive multi-channel GCN framework, AM-GCN, which performs graph convolution operation over both topology and feature spaces. Combined with attention mechanism, different information can be adequately fused.

- Our extensive experiments on a series of benchmark data sets clearly show that AM-GCN outperforms the state-of-the-art GCNs and extracts the most correlation information from both node features and topological structures nicely for challenging classification tasks.

The rest of the paper is organized as follows. In Section 2 we experimentally investigate the capability of GCNs in fusing node features and topology. In Section 3, we develop AM-GCN. We report experimental results in Section 4, and review related work in Section 5. We conclude the paper in Section 6.

2 FUSION CAPABILITY OF GCNS: AN EXPERIMENTAL INVESTIGATION

In this section, we use two simple yet intuitive cases to examine whether the state-of-the-art GCNs can adaptively learn from node features and topological structures in graphs and fuse them sufficiently for classification tasks. The main idea is that we will clearly establish the high correlation between node label with network topology and node features, respectively, then we will check the performance of GCN on these two simple cases. A good fusion capability of GCN should adaptively extract the correlated information with the supervision of node label, providing a good result. However, if the performance drops sharply in comparison with baselines, this will demonstrate that GCN cannot adaptively extract information from node features and topological structures, even there is a high correlation between node features or topological structures with the node label.

2.1 Case 1: Random Topology and Correlated Node Features

We generate a random network consisting of 900 nodes, where the probability of building an edge between any two nodes is 0.03. Each node has a feature vector of 50 dimensions. To generate node features, we randomly assign 3 labels to the 900 nodes, and for the nodes with the same label, we use one Gaussian distribution to generate the node features. The Gaussian distributions for the three classes of nodes have the same covariance matrix, but three different centers far away from each other. In this data set, the node labels are highly correlated with the node features, but not the topological structures.

We apply GCN [14] to train this network. For each class we randomly select 20 nodes for training and another 200 nodes for testing. We carefully tune the hyper-parameters to report the best performance and avoid over smoothing. Also, we apply MLP [21] to the node features only. The classification accuracies of GCN and MLP are 75.2% and 100%, respectively.

The results meet the expectation. Since the node features are highly correlated with the node labels, MLP shows excellent performance. GCN extracts information from both the node features and the topological structures, but cannot adaptively fuse them to avoid the interference from topological structures. It cannot match the high performance of MLP.

2.2 Case 2: Correlated Topology and Random Node Features

We generate another network with 900 nodes. This time, the node features, each of 50 dimensions, are randomly generated. For the

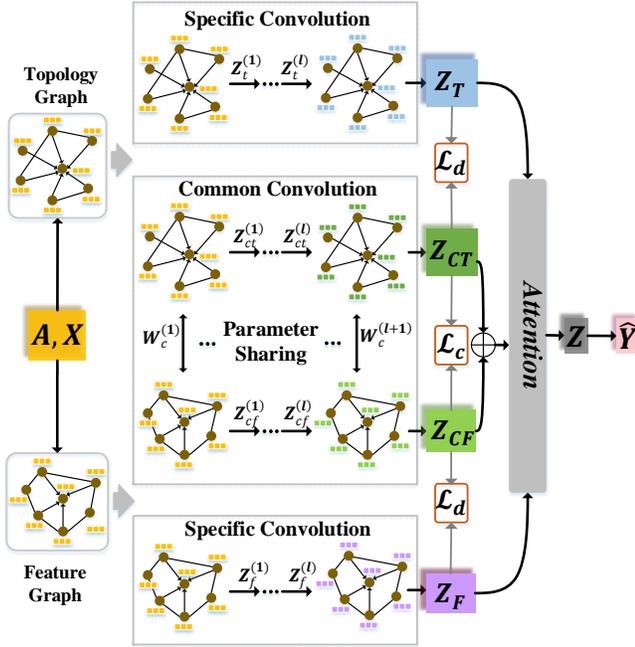


Figure 1: The framework of AM-GCN model. Node feature \mathbf{X} is to construct a feature graph. AM-GCN consists of two specific convolution modules, one common convolution module and the attention mechanism.

topological structure, we employ the Stochastic Blockmodel (SBM) [12] to split nodes into 3 communities (nodes 0-299, 300-599, 600-899, respectively). Within each community, the probability of building an edge is set to 0.03, and the probability of building an edge between nodes in different communities is set to 0.0015. In this data set, the node labels are determined by the communities, i.e., nodes in the same community have the same label.

Again we apply GCN to this network. We also apply DeepWalk [22] to the topology of the network, that is, the features are ignored by DeepWalk. The classification accuracies of GCN and DeepWalk are 87% and 100%, respectively.

DeepWalk performs well because it models network topological structures thoroughly. GCN extracts information from both the node features and the topological structures, but cannot adaptively fuse them to avoid the interference from node features. It cannot match the high performance of DeepWalk.

Summary. These cases show that the current fusion mechanism of GCN [14] is distant from optimal or even satisfactory. Even the correlation between node label with network topology or node features is very high, the current GCN cannot make full use of the supervision by node label to adaptively extract the most correlated information. However, the situation is more complex in reality, because it is hard to know whether the topology or the node features are more correlated with the final task, which prompts us to rethink the current mechanism of GCN.

3 AM-GCN: THE PROPOSED MODEL

Problem Settings: We focus on semi-supervised node classification in an attributed graph $G = (\mathbf{A}, \mathbf{X})$, where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the

symmetric adjacency matrix with n nodes and $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the node feature matrix, and d is the dimension of node features. Specifically, $A_{ij} = 1$ represents there is an edge between nodes i and j , otherwise, $A_{ij} = 0$. We suppose each node belongs to one out of C classes.

The overall framework of AM-GCN is shown in Figure 1. The key idea is that AM-GCN permits node features to propagate not only in topology space, but also in feature space, and the most correlated information with node label should be extracted from both of these two spaces. To this end, we construct a feature graph based on node features \mathbf{X} . Then with two specific convolution modules, \mathbf{X} is able to propagate over both of feature graph and topology graph to learn two specific embeddings \mathbf{Z}_F and \mathbf{Z}_T , respectively. Further, considering that the information in these two spaces have common characteristics, we design a common convolution module with parameter sharing strategy to learn the common embedding \mathbf{Z}_{CF} and \mathbf{Z}_{CT} , also, a consistency constraint \mathcal{L}_c is employed to enhance the "common" property of \mathbf{Z}_{CF} and \mathbf{Z}_{CT} . Besides, a disparity constraint \mathcal{L}_d is to ensure the independence between \mathbf{Z}_F and \mathbf{Z}_{CF} , as well as \mathbf{Z}_T and \mathbf{Z}_{CT} . Considering that node label may be correlated with topology or feature or both, AM-GCN utilizes an attention mechanism to adaptively fuse these embeddings with the learned weights, so as to extract the most correlated information \mathbf{Z} for the final classification task.

3.1 Specific Convolution Module

Firstly, in order to capture the underlying structure of nodes in feature space, we construct a k -nearest neighbor (k NN) graph $G_f = (\mathbf{A}_f, \mathbf{X})$ based on node feature matrix \mathbf{X} , where \mathbf{A}_f is the adjacency matrix of k NN graph. Specifically, we first calculate the similarity matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$ among n nodes. Actually, there are many ways to obtain \mathbf{S} , and we list two popular ones here, in which \mathbf{x}_i and \mathbf{x}_j are feature vectors of nodes i and j :

1) **Cosine Similarity:** It uses the cosine value of the angle between two vectors to measure the similarity:

$$\mathbf{S}_{ij} = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}. \quad (1)$$

2) **Heat Kernel:** The similarity is calculated by the Eq. (2) where t is the time parameter in heat conduction equation and we set $t = 2$.

$$\mathbf{S}_{ij} = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{t}}. \quad (2)$$

Here we uniformly choose the Cosine Similarity to obtain the similarity matrix \mathbf{S} , and then we choose top k similar node pairs for each node to set edges and finally get the adjacency matrix \mathbf{A}_f .

Then with the input graph $(\mathbf{A}_f, \mathbf{X})$ in feature space, the l -th layer output $\mathbf{Z}_f^{(l)}$ can be represented as:

$$\mathbf{Z}_f^{(l)} = \text{ReLU}(\tilde{\mathbf{D}}_f^{-\frac{1}{2}} \tilde{\mathbf{A}}_f \tilde{\mathbf{D}}_f^{-\frac{1}{2}} \mathbf{Z}_f^{(l-1)} \mathbf{W}_f^{(l)}), \quad (3)$$

where $\mathbf{W}_f^{(l)}$ is the weight matrix of the l -th layer in GCN, ReLU is the Relu activation function and the initial $\mathbf{Z}_f^{(0)} = \mathbf{X}$. Specifically, we have $\tilde{\mathbf{A}}_f = \mathbf{A}_f + \mathbf{I}_f$ and $\tilde{\mathbf{D}}_f$ is the diagonal degree matrix of $\tilde{\mathbf{A}}_f$. We denote the last layer output embedding as \mathbf{Z}_F . In this way, we can

learn the node embedding which captures the specific information \mathbf{Z}_F in feature space.

As for the topology space, we have the original input graph $G_t = (\mathbf{A}_t, \mathbf{X}_t)$ where $\mathbf{A}_t = \mathbf{A}$ and $\mathbf{X}_t = \mathbf{X}$. Then the learned output embedding \mathbf{Z}_T based on topology graph can be calculated in the same way as in feature space. Therefore, the specific information encoded in topology space can be extracted.

3.2 Common Convolution Module

In reality, the feature and topology spaces are not completely irrelevant. Basically, the node classification task, may be correlated with the information either in feature space or in topology space or in both of them, which is difficult to know beforehand. Therefore, we not only need to extract the node specific embedding in these two spaces, but also to extract the common information shared by the two spaces. In this way, it will become more flexible for the task to determine which part of information is the most correlated. To address this, we design a *Common-GCN* with parameter sharing strategy to get the embedding shared in two spaces.

First, we utilize *Common-GCN* to extract the node embedding $\mathbf{Z}_{ct}^{(l)}$ from topology graph $(\mathbf{A}_t, \mathbf{X})$ as follows

$$\mathbf{Z}_{ct}^{(l)} = \text{ReLU}(\tilde{\mathbf{D}}_t^{-\frac{1}{2}} \tilde{\mathbf{A}}_t \tilde{\mathbf{D}}_t^{-\frac{1}{2}} \mathbf{Z}_{ct}^{(l-1)} \mathbf{W}_c^{(l)}), \quad (4)$$

where $\mathbf{W}_c^{(l)}$ is the l -th layer weight matrix of *Common-GCN* and $\mathbf{Z}_{ct}^{(l-1)}$ is the node embedding in the $(l-1)$ th layer and $\mathbf{Z}_{ct}^{(0)} = \mathbf{X}$. When utilizing *Common-GCN* to learn the node embedding from feature graph $(\mathbf{A}_f, \mathbf{X})$, in order to extract the shared information, we share the same weight matrix $\mathbf{W}_c^{(l)}$ for every layer of *Common-GCN* as follows:

$$\mathbf{Z}_{cf}^{(l)} = \text{ReLU}(\tilde{\mathbf{D}}_f^{-\frac{1}{2}} \tilde{\mathbf{A}}_f \tilde{\mathbf{D}}_f^{-\frac{1}{2}} \mathbf{Z}_{cf}^{(l-1)} \mathbf{W}_c^{(l)}), \quad (5)$$

where $\mathbf{Z}_{cf}^{(l)}$ is the l -layer output embedding and $\mathbf{Z}_{cf}^{(0)} = \mathbf{X}$. The shared weight matrix can filter out the shared characteristics from two spaces. According to different input graphs, we can get two output embedding \mathbf{Z}_{CT} and \mathbf{Z}_{CF} and the common embedding \mathbf{Z}_C of the two spaces is:

$$\mathbf{Z}_C = (\mathbf{Z}_{CT} + \mathbf{Z}_{CF})/2. \quad (6)$$

3.3 Attention Mechanism

Now we have two specific embeddings \mathbf{Z}_T and \mathbf{Z}_F , and one common embedding \mathbf{Z}_C . Considering the node label can be correlated with one of them or even their combinations, we use the attention mechanism $\text{att}(\mathbf{Z}_T, \mathbf{Z}_C, \mathbf{Z}_F)$ to learn their corresponding importance $(\alpha_t, \alpha_c, \alpha_f)$ as follows:

$$(\alpha_t, \alpha_c, \alpha_f) = \text{att}(\mathbf{Z}_T, \mathbf{Z}_C, \mathbf{Z}_F), \quad (7)$$

here $\alpha_t, \alpha_c, \alpha_f \in \mathbb{R}^{n \times 1}$ indicate the attention values of n nodes with embeddings $\mathbf{Z}_T, \mathbf{Z}_C, \mathbf{Z}_F$, respectively.

Here we focus on node i , where its embedding in \mathbf{Z}_T is $\mathbf{z}_T^i \in \mathbb{R}^{1 \times h}$ (i.e., the i -th row of \mathbf{Z}_T). We firstly transform the embedding through a nonlinear transformation, and then use one shared attention vector $\mathbf{q} \in \mathbb{R}^{h \times 1}$ to get the attention value ω_T^i as follows:

$$\omega_T^i = \mathbf{q}^T \cdot \tanh(\mathbf{W} \cdot (\mathbf{z}_T^i)^T + \mathbf{b}). \quad (8)$$

Here $\mathbf{W} \in \mathbb{R}^{h \times h}$ is the weight matrix and $\mathbf{b} \in \mathbb{R}^{h \times 1}$ is the bias vector. Similarly, we can get the attention values ω_C^i and ω_F^i for node i in embedding matrices \mathbf{Z}_C and \mathbf{Z}_F , respectively. We then normalize the attention values $\omega_T^i, \omega_C^i, \omega_F^i$ with softmax function to get the final weight:

$$\alpha_T^i = \text{softmax}(\omega_T^i) = \frac{\exp(\omega_T^i)}{\exp(\omega_T^i) + \exp(\omega_C^i) + \exp(\omega_F^i)}. \quad (9)$$

Larger α_T^i implies the corresponding embedding is more important. Similarly, $\alpha_C^i = \text{softmax}(\omega_C^i)$ and $\alpha_F^i = \text{softmax}(\omega_F^i)$. For all the n nodes, we have the learned weights $\alpha_t = [\alpha_T^i], \alpha_c = [\alpha_C^i], \alpha_f = [\alpha_F^i] \in \mathbb{R}^{n \times 1}$, and denote $\alpha_T = \text{diag}(\alpha_t), \alpha_C = \text{diag}(\alpha_c)$ and $\alpha_F = \text{diag}(\alpha_f)$. Then we combine these three embeddings to obtain the final embedding \mathbf{Z} :

$$\mathbf{Z} = \alpha_T \cdot \mathbf{Z}_T + \alpha_C \cdot \mathbf{Z}_C + \alpha_F \cdot \mathbf{Z}_F. \quad (10)$$

3.4 Objective Function

3.4.1 Consistency Constraint. For the two output embeddings \mathbf{Z}_{CT} and \mathbf{Z}_{CF} of *Common-GCN*, despite the *Common-GCN* has the shared weight matrix, here we design a consistency constraint to further enhance their commonality.

Firstly, we use L_2 -normalization to normalize the embedding matrix as $\mathbf{Z}_{CTnor}, \mathbf{Z}_{CFnor}$. Then, the two normalized matrix can be used to capture the similarity of n nodes as \mathbf{S}_T and \mathbf{S}_F as follows:

$$\begin{aligned} \mathbf{S}_T &= \mathbf{Z}_{CTnor} \cdot \mathbf{Z}_{CTnor}^T, \\ \mathbf{S}_F &= \mathbf{Z}_{CFnor} \cdot \mathbf{Z}_{CFnor}^T. \end{aligned} \quad (11)$$

The consistency implies that the two similarity matrices should be similar, which gives rise to the following constraint:

$$\mathcal{L}_c = \|\mathbf{S}_T - \mathbf{S}_F\|_F^2. \quad (12)$$

3.4.2 Disparity Constraint. Here because embeddings \mathbf{Z}_T and \mathbf{Z}_{CT} are learned from the same graph $G_t = (\mathbf{A}_t, \mathbf{X}_t)$, to ensure they can capture different information, we employ the Hilbert-Schmidt Independence Criterion (HSIC) [24], a simple but effective measure of independence, to enhance the disparity of these two embeddings. Due to its simplicity and neat theoretical properties, HSIC has been applied to several machine learning tasks [10, 19]. Formally, the HSIC constraint of \mathbf{Z}_T and \mathbf{Z}_{CT} is defined as:

$$\text{HSIC}(\mathbf{Z}_T, \mathbf{Z}_{CT}) = (n-1)^{-2} \text{tr}(\mathbf{R}\mathbf{K}_T\mathbf{R}\mathbf{K}_{CT}), \quad (13)$$

where \mathbf{K}_T and \mathbf{K}_{CT} are the Gram matrices with $k_{T,ij} = k_T(\mathbf{z}_T^i, \mathbf{z}_T^j)$ and $k_{CT,ij} = k_{CT}(\mathbf{z}_{CT}^i, \mathbf{z}_{CT}^j)$. And $\mathbf{R} = \mathbf{I} - \frac{1}{n} \mathbf{e}\mathbf{e}^T$, where \mathbf{I} is an identity matrix and \mathbf{e} is an all-one column vector. In our implementation, we use the inner product kernel function for \mathbf{K}_T and \mathbf{K}_{CT} .

Similarly, considering the embeddings \mathbf{Z}_F and \mathbf{Z}_{CF} are also learned from the same graph $(\mathbf{A}_f, \mathbf{X})$, their disparity should also be enhanced by HSIC:

$$\text{HSIC}(\mathbf{Z}_F, \mathbf{Z}_{CF}) = (n-1)^{-2} \text{tr}(\mathbf{R}\mathbf{K}_F\mathbf{R}\mathbf{K}_{CF}). \quad (14)$$

Then we set the disparity constraint as \mathcal{L}_d where:

$$\mathcal{L}_d = \text{HSIC}(\mathbf{Z}_T, \mathbf{Z}_{CT}) + \text{HSIC}(\mathbf{Z}_F, \mathbf{Z}_{CF}). \quad (15)$$

Table 1: The statistics of the datasets

Dataset	Nodes	Edges	Classes	Features	Training	Test
Citeseer	3327	4732	6	3703	120/240/360	1000
UAI2010	3067	28311	19	4973	380/760/1140	1000
ACM	3025	13128	3	1870	60/120/180	1000
BlogCatalog	5196	171743	6	8189	120/240/360	1000
Flickr	7575	239738	9	12047	180/360/540	1000
CoraFull	19793	65311	70	8710	1400/2800/4200	1000

3.4.3 Optimization Objective. We use the output embedding \mathbf{Z} in Eq. (10) for semi-supervised multi-class classification with a linear transformation and a softmax function. Denote the class predictions for n nodes as $\hat{\mathbf{Y}} = [\hat{y}_{ic}] \in \mathbb{R}^{n \times C}$ where \hat{y}_{ic} is the probability of node i belonging to class c . Then the $\hat{\mathbf{Y}}$ can be calculated in the following way:

$$\hat{\mathbf{Y}} = \text{softmax}(\mathbf{W} \cdot \mathbf{Z} + \mathbf{b}), \quad (16)$$

where $\text{softmax}(x) = \frac{\exp(x)}{\sum_{c=1}^C \exp(x_c)}$ is actually a normalizer across all classes.

Suppose the training set is L , for each $l \in L$ the real label is \mathbf{Y}_l and the predicted label is $\hat{\mathbf{Y}}_l$. Then the cross-entropy loss for node classification over all training nodes is represented as \mathcal{L}_t where:

$$\mathcal{L}_t = - \sum_{l \in L} \sum_{i=1}^C \mathbf{Y}_{li} \ln \hat{\mathbf{Y}}_{li}. \quad (17)$$

Combining the node classification task and constraints, we have the following overall objective function:

$$\mathcal{L} = \mathcal{L}_t + \gamma \mathcal{L}_c + \beta \mathcal{L}_d, \quad (18)$$

where γ and β are parameters of the consistency and disparity constraint terms. With the guide of labeled data, we can optimize the proposed model via back propagation and learn the embedding of nodes for classification.

4 EXPERIMENTS

4.1 Experimental Setup

Datasets Our proposed AM-GCN is evaluated on six real world datasets which are summarized in Table 1, moreover, we provide all the data websites in the supplement for reproducibility.

- **Citeseer** [14]: Citeseer is a research paper citation network, where nodes are publications and edges are citation links. Node attributes are bag-of-words representations of the papers and all nodes are divided into six areas.
- **UAI2010** [28]: We use this dataset with 3067 nodes and 28311 edges which has been tested in graph convolutional networks for community detection in [28].
- **ACM** [29]: This network is extracted from ACM dataset where nodes represent papers and there is an edge between two papers if they have the same author. All the papers are divided into 3 classes (*Database*, *Wireless Communication*, *DataMining*). The features are the bag-of-words representations of paper keywords.
- **BlogCatalog** [18]: It is a social network with bloggers and their social relationships from the BlogCatalog website. Node attributes are constructed by the keywords of user profiles,

and the labels represent the topic categories provided by the authors, and all nodes are divided into 6 classes.

- **Flickr** [18]: Flickr is an image and video hosting website, where users interact with each other via photo sharing. It is a social network where nodes represent users and edges represent their relationships, and all the nodes are divided into 9 classes according to interest groups of users.
- **CoraFull** [2]: This is the larger version of the well-known citation network Cora dataset, where nodes represent papers and edges represents their citations, and the nodes are labeled based on the paper topics.

Baselines We compare AM-GCN with two types of state-of-the-art methods, covering two network embedding algorithms and six graph neural network based methods. Moreover, we provide all the code websites in the supplement for reproducibility.

- **DeepWalk** [22] is a network embedding method which uses random walk to obtain contextual information and uses skip-gram algorithm to learn network representations.
- **LINE** [25] is a large-scale network embedding method preserving first-order and second-order proximity of the network separately. Here we use LINE (1st+2nd).
- **Chebyshev** [5] is a GCN-based method utilizing Chebyshev filters.
- **GCN** [14] is a semi-supervised graph convolutional network model which learns node representations by aggregating information from neighbors.
- **kNN-GCN**. For comparison, instead of traditional topology graph, we use the sparse k -nearest neighbor graph calculated from feature matrix as the input graph of GCN and represent it as kNN-GCN.
- **GAT** [27] is a graph neural network model using attention mechanism to aggregate node features.
- **DEMO-Net** [31] is a degree-specific graph neural network for node classification.
- **MixHop** [1] is a GCN-based method which mixes the feature representations of higher-order neighbors in one graph convolution layer.

Parameters Setting To more comprehensively evaluate our model, we select three label rates for training set (i.e., 20, 40, 60 labeled nodes per class) and choose 1000 nodes as the test set. All baselines are initialized with same parameters suggested by their papers and we also further carefully turn parameters to get optimal performance. For our model, we train three 2-layer GCNs with the same hidden layer dimension ($nhid1$) and the same output dimension ($nhid2$) simultaneously, where $nhid1 \in \{512, 768\}$ and $nhid2 \in \{32, 128, 256\}$. We use $0.0001 \sim 0.0005$ learning rate with Adam optimizer. In addition, the dropout rate is 0.5, weight decay $\in \{5e-3, 5e-4\}$ and $k \in \{2 \dots 10\}$ for k -nearest neighbor graph. The coefficient of consistency constraint and disparity constraints are searched in $\{0.01, 0.001, 0.0001\}$ and $\{1e-10, 5e-9, 1e-9, 5e-8, 1e-8\}$. For all methods, we run 5 times with the same partition and report the average results. And we use Accuracy (ACC) and macro F1-score (F1) to evaluate performance of models. For the reproducibility, we provide the specific parameter values in the supplement (Section A.3).

Table 2: Node classification results(%). (Bold: best; Underline: runner-up.)

Datasets	Metrics	L/C	DeepWalk	LINE	Chebyshev	GCN	kNN-GCN	GAT	DEMO-Net	MixHop	AM-GCN
Citeseer	ACC	20	43.47	32.71	69.80	70.30	61.35	<u>72.50</u>	69.50	71.40	73.10
		40	45.15	33.32	71.64	<u>73.10</u>	61.54	73.04	70.44	71.48	74.70
		60	48.86	35.39	73.26	74.48	62.38	<u>74.76</u>	71.86	72.16	75.56
	F1	20	38.09	31.75	65.92	67.50	58.86	<u>68.14</u>	67.84	66.96	68.42
		40	43.18	32.42	68.31	<u>69.70</u>	59.33	69.58	66.97	67.40	69.81
		60	48.01	34.37	70.31	<u>71.24</u>	60.07	71.60	68.22	69.31	70.92
UAI2010	ACC	20	42.02	43.47	50.02	49.88	<u>66.06</u>	56.92	23.45	61.56	70.10
		40	51.26	45.37	58.18	51.80	<u>68.74</u>	63.74	30.29	65.05	73.14
		60	54.37	51.05	59.82	54.40	<u>71.64</u>	68.44	34.11	67.66	74.40
	F1	20	32.93	37.01	33.65	32.86	<u>52.43</u>	39.61	16.82	49.19	55.61
		40	46.01	39.62	38.80	33.80	<u>54.45</u>	45.08	26.36	53.86	64.88
		60	44.43	43.76	40.60	34.12	54.78	48.97	29.05	<u>56.31</u>	65.99
ACM	ACC	20	62.69	41.28	75.24	<u>87.80</u>	78.52	87.36	84.48	81.08	90.40
		40	63.00	45.83	81.64	<u>89.06</u>	81.66	88.60	85.70	82.34	90.76
		60	67.03	50.41	85.43	<u>90.54</u>	82.00	90.40	86.55	83.09	91.42
	F1	20	62.11	40.12	74.86	<u>87.82</u>	78.14	87.44	84.16	81.40	90.43
		40	61.88	45.79	81.26	<u>89.00</u>	81.53	88.55	84.83	81.13	90.66
		60	66.99	49.92	85.26	<u>90.49</u>	81.95	90.39	84.05	82.24	91.36
BlogCatalog	ACC	20	38.67	58.75	38.08	69.84	<u>75.49</u>	64.08	54.19	65.46	81.98
		40	50.80	61.12	56.28	71.28	<u>80.84</u>	67.40	63.47	71.66	84.94
		60	55.02	64.53	70.06	72.66	<u>82.46</u>	69.95	76.81	77.44	87.30
	F1	20	34.96	57.75	33.39	68.73	<u>72.53</u>	63.38	52.79	64.89	81.36
		40	48.61	60.72	53.86	70.71	<u>80.16</u>	66.39	63.09	70.84	84.32
		60	53.56	63.81	68.37	71.80	<u>81.90</u>	69.08	76.73	76.38	86.94
Flickr	ACC	20	24.33	33.25	23.26	41.42	<u>69.28</u>	38.52	34.89	39.56	75.26
		40	28.79	37.67	35.10	45.48	<u>75.08</u>	38.44	46.57	55.19	80.06
		60	30.10	38.54	41.70	47.96	<u>77.94</u>	38.96	57.30	64.96	82.10
	F1	20	21.33	31.19	21.27	39.95	<u>70.33</u>	37.00	33.53	40.13	74.63
		40	26.90	37.12	33.53	43.27	<u>75.40</u>	36.94	45.23	56.25	79.36
		60	27.28	37.77	40.17	46.58	<u>77.97</u>	37.35	56.49	65.73	81.81
CoraFull	ACC	20	29.33	17.78	53.38	56.68	41.68	<u>58.44</u>	54.50	47.74	58.90
		40	36.23	25.01	58.22	60.60	44.80	<u>62.98</u>	60.28	57.20	63.62
		60	40.60	29.65	59.84	62.00	46.68	<u>64.38</u>	61.58	60.18	65.36
	F1	20	28.05	18.24	47.59	52.48	37.15	<u>54.44</u>	50.44	45.07	54.74
		40	33.29	25.43	53.47	55.57	40.42	<u>58.30</u>	56.26	53.55	59.19
		60	37.95	30.87	54.15	56.24	43.22	<u>59.61</u>	57.26	56.40	61.32

4.2 Node Classification

The node classification results are reported in Table 2, where L/C means the number of labeled nodes per class. We have the following observations:

- Compared with all baselines, the proposed AM-GCN generally achieves the best performance on all datasets with all label rates. Especially, for ACC, AM-GCN achieves maximum relative improvements of 8.59% on BlogCatalog and 8.63% on Flickr. The results demonstrate the effectiveness of AM-GCN.
- AM-GCN consistently outperforms GCN and kNN-GCN on all the datasets, indicating the effectiveness of the adaptive fusion mechanism in AM-GCN, because it can extract more useful information than only performing GCN and kNN-GCN, respectively.
- Comparing with GCN and kNN-GCN, we can learn that there does exist structural difference between topology graph and feature graph and performing GCN on traditional topology graph does not always show better result than on feature graph. For example, in BlogCatalog, Flickr and UAI2010, the feature graph performs better than topology. This further confirms the necessity of introducing feature graph in GCN.
- Moreover, compared with GCN, the improvement of AM-GCN is more substantial on the datasets with better feature graph (kNN), such as UAI2010, BlogCatalog, Flickr. This implies that AM-GCN introduces a better and more suitable kNN graph for label to supervise feature propagation and node representation learning.

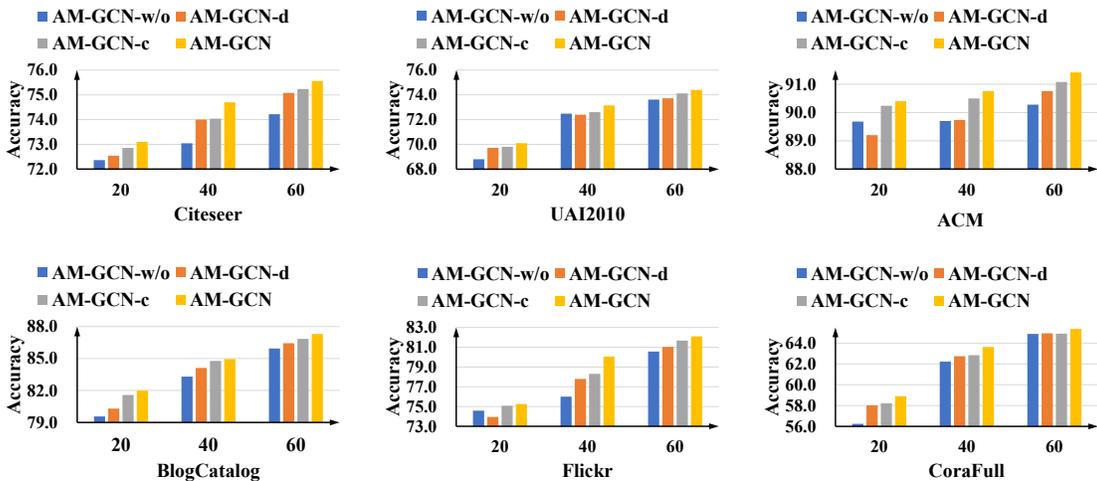


Figure 2: The results(%) of AM-GCN and its variants on six datasets.

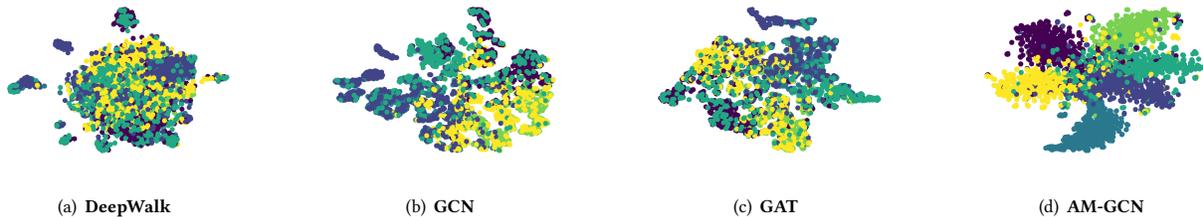


Figure 3: Visualization of the learned node embeddings on BlogCatalog dataset.

4.3 Analysis of Variants

In this section, we compare AM-GCN with its three variants on all datasets to validate the effectiveness of the constraints.

- **AM-GCN-w/o**: AM-GCN without constraints \mathcal{L}_c and \mathcal{L}_d .
- **AM-GCN-c**: AM-GCN with the consistency constraint \mathcal{L}_c .
- **AM-GCN-d**: AM-GCN with the disparity constraint \mathcal{L}_d .

From the results in Figure 2, we can draw the following conclusions: (1) The results of AM-GCN are consistently better than all the other three variants, indicating the effectiveness of using the two constraints together. (2) The results of AM-GCN-c and AM-GCN-d are usually better than AM-GCN-w/o on all datasets with all label rates, verifying the usefulness of the two constraints. (3) AM-GCN-c is generally better than AM-GCN-d on all datasets, which implies the consistency constraint plays a more vital role in this framework. (4) Comparing the results of Figure 2 and Table 2, we can find that AM-GCN-w/o, although without any constraints, still achieves very competitive performance against baselines, demonstrating that our framework is stable and competitive.

4.4 Visualization

For a more intuitive comparison and to further show the effectiveness of our proposed model, we conduct the task of visualization on BlogCatalog dataset. We use the output embedding on the last layer of AM-GCN (or GCN, GAT) before *softmax* and plot the learned

embedding of test set using t-SNE [26]. The results of BlogCatalog in Figure 3 are colored by real labels.

From Figure 3, we can find that the results of DeepWalk, GCN, and GAT are not satisfactory, because the nodes with different labels are mixed together. Apparently, the visualization of AM-GCN performs best, where the learned embedding has a more compact structure, the highest intra-class similarity and the clearest distinct boundaries among different classes.

4.5 Analysis of Attention Mechanism

In order to investigate whether the attention values learned by our proposed model are meaningful, we analyze the attention distribution and attention learning trend, respectively.

Analysis of attention distributions. AM-GCN learns two specific and one common embeddings, each of which is associated with the attention values. We conduct the attention distribution analysis on all datasets with 20 label rate, where the results are shown in Figure 4. As we can see, for Citeseer, ACM, CoraFull, the attention values of specific embeddings in topology space are larger than the values in feature space, and the values of common embeddings are between them. This implies that the information in topology space should be more important than the information in feature space. To verify this, we can see that the results of GCN are better than *k*NN-GCN on these datasets in Table 2. Conversely, for UAI2010, BlogCatalog and Flickr, in comparison with Figure 4 and Table 2, we can find *k*NN-GCN performs better than GCN, meanwhile, the

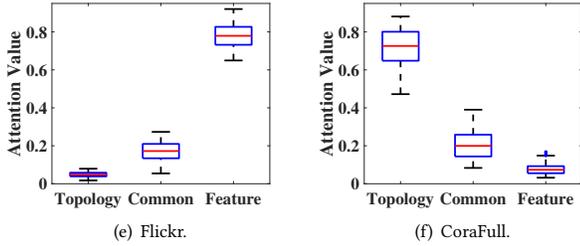
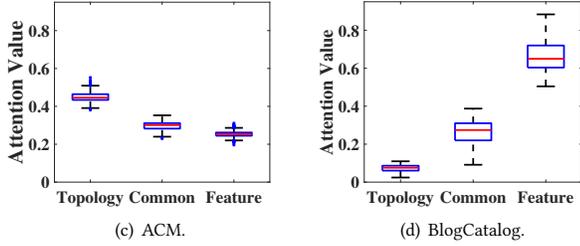
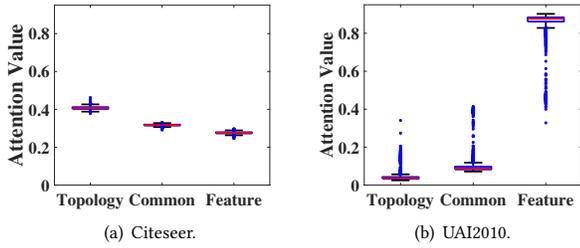


Figure 4: Analysis of attention distribution.

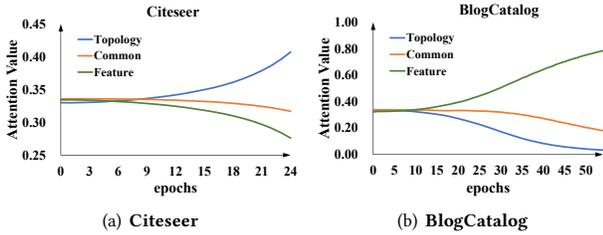


Figure 5: The attention changing trends w.r.t epochs.

attention values of specific embeddings in feature space are also larger than those in topology space. In summary, the experiment demonstrates that our proposed AM-GCN is able to adaptively assign larger attention value for more important information.

Analysis of attention trends. We analyze the changing trends of attention values during the training process. Here we take Citeseer and BlogCatalog with 20 label rate as examples in Figure 5, where x -axis is the epoch and y -axis is the average attention value. More results are in supplement A.4.1. At the beginning, the average attention values of Topology, Feature, and Common are almost the same, with the training epoch increasing, the attention values become different. For example, in BlogCatalog, the attention value for topology gradually decreases, while the attention value for feature keeps increasing. This phenomenon is consistent with the conclusions in Table 2 and Figure 4, i.e., k NN-GCN with feature graph

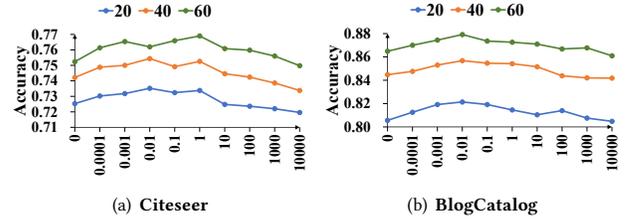


Figure 6: Analysis of parameter γ .

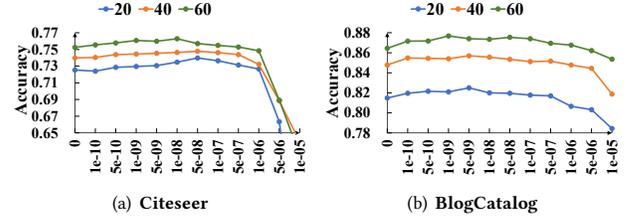


Figure 7: Analysis of parameter β .

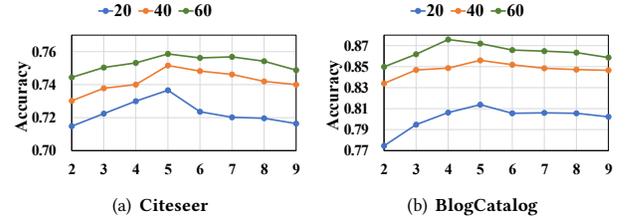


Figure 8: Analysis of parameter k .

performs better than GCN and the information in feature space is more important than in topology space. We can see that AM-GCN can learn the importance of different embeddings step by step.

4.6 Parameter Study

In this section, we investigate the sensitivity of parameters on Citeseer and BlogCatalog datasets. More results are in A.4.2.

Analysis of consistency coefficient γ . We test the effect of the consistency constraint weight γ in Eq. (18), and vary it from 0 to 10000. The results are shown in Figure 6. With the increase of the consistency coefficient, the performances raise first and then start to drop slowly. Basically, AM-GCN is stable when the γ is within the range from $1e-4$ to $1e+4$ on all datasets. We can also see that the curves of 20, 40, 60 label rates show similar changing trend.

Analysis of disparity constraint coefficient β . We then test the effect of the disparity constraint weight β in Eq. (18), and vary it from 0 to $1e-5$. The results are shown in Figure 7. Similarly, with the increase of β , the performances also raise first, but the performance will drop quickly if β is larger than $1e-6$ for Citeseer in Figure 7(a), while for BlogCatalog, it is relatively stable.

Analysis of k -nearest neighbor graph k . In order to check the impact of the top k neighborhoods in k NN graph, we study the performance of AM-GCN with various number of k ranging from 2 to 10 in Figure 8. For Citeseer and BlogCatalog, the accuracies increase first and then start to decrease. It may probably be that

if the graph becomes denser, the feature is easier to be smoothed, and also, larger k may introduce more noisy edges.

5 RELATED WORK

Recently, graph convolutional network (GCN) models [4, 9, 17, 23, 33, 35] have been widely studied. For example, [3] first designs the graph convolution operation in Fourier domain by the graph Laplacian. Then [5] further employs the Chebyshev expansion of the graph Laplacian to improve the efficiency. [14] simplifies the convolution operation and proposes to only aggregate the node features from the one-hop neighbors. GAT [27] introduces the attention mechanism to aggregate node features with the learned weights. GraphSAGE [11] proposes to sample and aggregate features from local neighborhoods of nodes with mean/max/LSTM pooling. DEMO-Net [31] designs a degree-aware feature aggregation process. MixHop [1] aggregates feature information from both first-order and higher-order neighbors in each layer of network, simultaneously. Most of the current GCNs essentially focus on fusing network topology and node features to learn node embedding for classification. Also, there are some recent works on analyzing the fusion mechanism of GCN. For example, [15] shows that GCNs actually perform the Laplacian smoothing on node features, [20] and [30] prove that topological structures play the role of low-pass filtering on node features. To learn more works on GCNs, please refer to the elaborate reviews [32, 38]. However, whether the GCNs can adaptively extract the correlated information from node features and topological structures for classification remains unclear.

6 CONCLUSION

In this paper, we rethink the fusion mechanism of network topology and node features in GCN and surprisingly discover it is distant from optimal. Motivated by this fundamental problem, we study how to adaptively learn the most correlated information from topology and node features and sufficiently fuse them for classification. We propose a multi-channel model AM-GCN which is able to learn suitable importance weights when fusing topology and node feature information. Extensive experiments well demonstrate the superior performance over the state-of-the-art models on real world datasets.

7 ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (No. 61702296, 61772082, 61806020, U1936104, U1936219, 61772304, U1611461, 61972442), the National Key Research and Development Program of China (No. 2018YFB1402600, 2018AAA0102004), the CCF-Tencent Open Fund, Beijing Academy of Artificial Intelligence (BAAI), and a grant from the Institute for Guo Qiang, Tsinghua University. Jian Pei’s research is supported in part by the NSERC Discovery Grant program. All opinions, findings, conclusions and recommendations are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Sami Abu-El-Hajja, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *ICML*. 21–29.

[2] Aleksandar Bojchevski and Stephan GÄjinnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *ICLR*.

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *ICLR*.

[4] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*. 3844–3852.

[6] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph Neural Networks for Social Recommendation. In *WWW*. 417–426.

[7] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. In *ICML*. 2083–2092.

[8] Hongchang Gao, Jian Pei, and Heng Huang. 2019. Conditional Random Field Enhanced Graph Convolutional Neural Networks. In *SIGKDD*. 276–284.

[9] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-Scale Learnable Graph Convolutional Networks. In *SIGKDD*. 1416–1424.

[10] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard SchÄullkopf. 2005. Measuring statistical dependence with hilbert-schmidt norms. In *ALT*. 63–77.

[11] Will Hamilton, Zhitaoying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.

[12] Brian Karrer and M. E. J. Newman. 2011. Stochastic blockmodels and community structure in networks. *Physical Review E* 83, 1 (2011), 16107.

[13] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *arXiv preprint arXiv:1611.07308*.

[14] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[15] Qimai Li, Zhichao Han, and Xiaoming Wu. 2018. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI*. 3538–3545.

[16] Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and wenwu zhu. 2019. Disentangled Graph Convolutional Networks. In *ICML*. 4212–4221.

[17] Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. 2019. Graph Convolutional Networks with EigenPooling. In *SIGKDD*. 723–731.

[18] Zaiqiao Meng, Shangsong Liang, Hongyan Bao, and Xiangliang Zhang. 2019. Co-Embedding Attributed Networks. In *WSDM*. 393–401.

[19] Donglin Niu, Jennifer G. Dy, and Michael I. Jordan. 2010. Multiple Non-Redundant Spectral Clustering Views. In *ICML*. 831–838.

[20] Hoang Nt and Takatori Maehara. 2019. Revisiting Graph Neural Networks: All We Have is Low-Pass Filters. *arXiv preprint arXiv:1905.09550* (2019).

[21] S.K. Pal and S. Mitra. 1992. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on Neural Networks* 3, 5 (1992), 683–697.

[22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. 701–710.

[23] Meng Qu, Yoshua Bengio, and Jian Tang. 2019. GMNN: Graph Markov Neural Networks. In *ICML*. 5241–5250.

[24] Le Song, Alex Smola, Arthur Gretton, Karsten M. Borgwardt, and Justin Bedo. 2007. Supervised feature selection via dependence estimation. In *ICML*. 823–830.

[25] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.

[26] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605.

[27] Petar VeljÄjkoviÄ, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro LiÄs, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[28] Wenjun Wang, Xiao Liu, Pengfei Jiao, Xue Chen, and Di Jin. 2018. A Unified Weakly Supervised Framework for Community Detection and Semantic Matching. In *PAKDD*. 218–230.

[29] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous Graph Attention Network. In *WWW*. 2022–2032.

[30] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*. 6861–6871.

[31] Jun Wu, Jingrui He, and Jiejun Xu. 2019. Demo-net: Degree-specific graph neural networks for node and graph classification. In *SIGKDD*. 406–415.

[32] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

[33] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks. In *ICLR*.

[34] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *SIGKDD*. 974–983.

[35] Zhitaoying, Ines Chami, Christopher RÄl, and Jure Leskovec. 2019. Hyperbolic Graph Convolutional Neural Networks. In *NeurIPS*. 4869–4880.

[36] Jiaxuan You, Rex, and Jure Leskovec. 2019. Position-aware Graph Neural Networks. In *ICML*. 7134–7143.

[37] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Chen Yixin. 2018. An End-to-End Deep Learning Architecture for Graph Classification. In *AAAI*. 4438–4445.

[38] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2018. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202* (2018).

A SUPPLEMENT

In the supplement, for the reproducibility, we provide our experimental environment and all the baselines and datasets websites. The implementation details, including the detailed hyper-parameter values for all the experiments, are also provided. Finally, we show more additional results to support the conclusions in our paper.

A.1 Experiments Settings

All experiments are conducted with the following setting:

- Operating system: CentOS Linux release 7.6.1810
- CPU: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
- GPU: GeForce GTX 1080 Ti
- Software versions: Python 3.7; Pytorch 1.1.0; Numpy 1.16.2; SciPy 1.3.1; NetworkX 2.4; scikit-learn 0.21.3

A.2 Baselines and Datasets

The publicly available implementations of Baselines can be found at the following URLs:

- DeepWalk, LINE: <https://github.com/thunlp/OpenNE>
- Chebyshev: <https://github.com/tkipf/gcn>
- GCN in Pytorch: <https://github.com/tkipf/pygcn>
- GAT in Pytorch: <https://github.com/Diego999/pyGAT/>
- DEMO-Net: <https://github.com/jwu4sml/DEMO-Net>
- MixHop: <https://github.com/samihaija/mixhop>

And the datasets used in this paper can be found as the following URLs:

- Citeseer: <https://github.com/tkipf/pygcn>
- UAI2010: <http://linqs.umiaccs.umd.edu/projects/projects/lbc/index.html>
- ACM: <https://github.com/Jhy1993/HAN>
- BlogCatalog: <https://github.com/mengzaiqiao/CAN>
- Flickr: <https://github.com/mengzaiqiao/CAN>
- CoraFull: <https://github.com/abojchevski/graph2gauss/>

A.3 Implementation Details

The codes of AM-GCN are based on the Graph Convolutional Networks in PyTorch version¹. And for the reproducibility of our proposed model, we also list the parameter values used in our model in Table 3.

A.4 Additional Results

In this section, we provide the additional results of our experiments including analysis of attention trends on the other four datasets and parameter study on UAI2010 and Flickr datasets.

A.4.1 Analysis of attention trends. Following the setting from the Section 4.5, we give the additional analysis of attention trends on the other four datasets in Figure 9. The changing process of attention values follows the same way with the results in Figure 5. What's more, the final learned attention values are consistent with the corresponding distributions in Figure 4, from which we can further verify the effectiveness of attention mechanism.

¹<https://github.com/tkipf/pygcn>

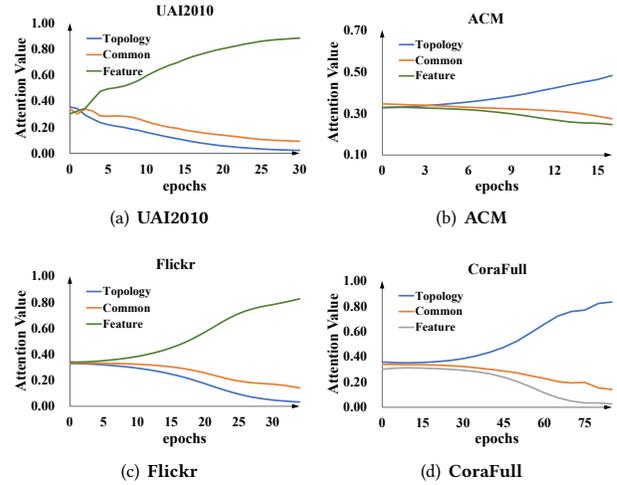


Figure 9: Changing trends on another four datasets.

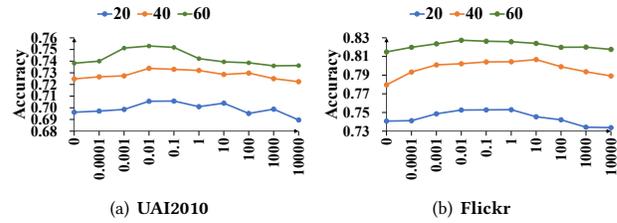


Figure 10: Analysis of parameter γ .

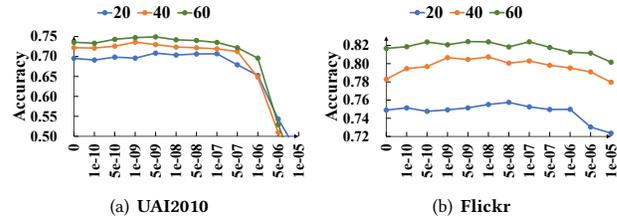


Figure 11: Analysis of parameter β .

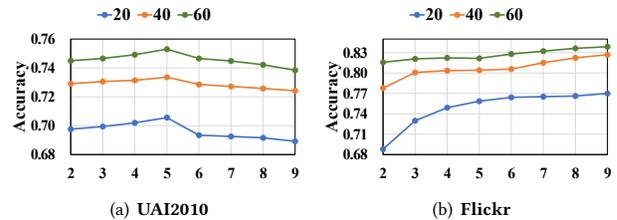


Figure 12: Analysis of parameter k .

Table 3: Model Hyperparameters.

Datasets	L/C	nhid1	nhid2	dropout	lr	weight-decay	epoch _{max}	k	γ	β
Citeseer	20	768	256	0.5	0.0005	5e-3	25	7	0.001	5e-10
	40	768	128	0.5	0.0005	5e-3	25	7	0.001	5e-8
	60	768	128	0.5	0.0005	5e-3	25	7	0.001	5e-8
UAI2010	20	512	128	0.5	0.0005	5e-4	50	5	0.001	1e-9
	40	512	128	0.5	0.0005	5e-4	70	5	0.01	1e-9
	60	512	128	0.5	0.0005	1e-5	70	5	0.01	1e-9
ACM	20	768	256	0.5	0.0005	5e-4	20	5	0.001	1e-8
	40	768	256	0.5	0.0005	5e-4	20	5	0.001	1e-8
	60	768	256	0.5	0.0001	6e-4	30	5	0.001	1e-8
BlogCatalog	20	512	128	0.5	0.0002	1e-5	55	5	0.001	5e-8
	40	512	128	0.5	0.0005	5e-4	40	5	0.001	5e-8
	60	512	128	0.5	0.0005	8e-4	50	5	0.01	5e-8
Flickr	20	512	128	0.5	0.0003	5e-4	60	5	0.01	1e-10
	40	512	128	0.5	0.0005	1e-5	40	5	0.01	1e-10
	60	512	128	0.5	0.0005	5e-4	40	5	0.01	1e-10
CoraFull	20	512	32	0.5	0.001	5e-4	300	6	0.0001	1e-10
	40	512	32	0.5	0.001	5e-4	300	6	0.00001	1e-10
	60	512	32	0.5	0.001	5e-4	300	6	0.0001	1e-10

A.4.2 Parameters Study. To further check the stability and applicability of parameters γ and β , we show the corresponding results on UAI2010 and Flickr datasets in Figure 10 and Figure 11. Combining the results in Section 4.6, we can see the consistency and disparity constraints have stable performance on a large range while the performance with disparity constraint may decrease when β

is larger than some suitable boundary. In Figure 12, we test the impact of k in k -nearest neighbor graph on UAI2010 and Flickr datasets. For UAI2010, the performance increases first and then starts to decrease 2 to 10. And for Flickr, a larger k may import richer structural information for feature graph which also profits AM-GCN.