

# Anisotropic Graph Convolutional Network for Semi-supervised Learning

Mahsa Mesgaran and A. Ben Hamza

Concordia Institute for Information Systems Engineering  
Concordia University, Montreal, QC, Canada

## Abstract

Graph convolutional networks learn effective node embeddings that have proven to be useful in achieving high-accuracy prediction results in semi-supervised learning tasks, such as node classification. However, these networks suffer from the issue of over-smoothing and shrinking effect of the graph due in large part to the fact that they diffuse features across the edges of the graph using a linear Laplacian flow. This limitation is especially problematic for the task of node classification, where the goal is to predict the label associated with a graph node. To address this issue, we propose an anisotropic graph convolutional network for semi-supervised node classification by introducing a non-linear function that captures informative features from nodes, while preventing oversmoothing. The proposed framework is largely motivated by the good performance of anisotropic diffusion in image and geometry processing, and learns nonlinear representations based on local graph structure and node features. The effectiveness of our approach is demonstrated on three citation networks and two image datasets, achieving better or comparable classification accuracy results compared to the standard baseline methods.

**Keywords:** Network embedding; graph convolutional networks; anisotropic diffusion; classification.

## 1 Introduction

Graphs are ubiquitous in a wide array of application domains, ranging from social networks [1–3] and transportation systems [4] and cyber-security [5] to brain networks [6] graph signal processing [7], and video analysis [8–10]. They provide a flexible way to inherently represent real-world entities as a set of nodes and their interactions as a set of links/edges. This interconnection of entities and their pairwise relationships forms a graph structure when visualized.

With the prevalence and increasing proliferation of graph-structured data in real-world applications, there has been a surge of interest in developing efficient representations of graphs. Network embedding has recently emerged as a powerful paradigm for representing and analyzing graph-structured data [11–14]. The idea is to learn low-dimensional embedding vectors, such that both structural and semantic information are captured. These learned embeddings can then be used as input to various machine learning algorithms for downstream tasks, such as link prediction, visualization, recommendation, community detection, and node classification. The latter task is the focus of

this paper. The objective of node classification is to predict the most probable labels of nodes in a graph [15]. In a social network, for instance, we want to predict user labels such as their interest, beliefs or other characteristics [1], while in a citation network, we want to classify documents based on their topics.

There is a sizable body of literature on network embedding that has centered around the use of random walks and neural language models to learn effective low-dimensional embedding vectors of graph nodes [15–17]. Perozzi *et al.* [16] introduce DeepWalk, a deep learning based framework that learns latent representations of nodes in a graph by leveraging local information obtained from truncated random walks. Each random walk is treated as a sentence that is fed into the skip-gram language model [18], which maximizes the co-occurrence probability among the words that appear within a window in a sentence. Another popular approach that also uses random walks is node2vec [17], a semi-supervised algorithm for feature learning in graphs that can be regarded as a generalization of DeepWalk. While DeepWalk performs a uniform random walk, node2vec uses a second-order random walk approach to generate network neighborhoods for nodes via breadth-first and depth-first sampling strategies. However, node2vec involves a number of parameters that require fine-tuning for each dataset and each task.

In recent years, the advent of deep learning has sparked groundswell of interest in the adoption of graph neural networks (GNNs) for learning latent representations of graphs [19–28]. A plethora of GNNs is based on convolutional neural networks (CNNs) and network embedding. Defferrard *et al.* [24] introduce the Chebyshev network (ChebyNet), an efficient spectral-domain graph convolutional neural network that uses recursive Chebyshev polynomial spectral filters to avoid explicit computation of the Laplacian eigenvectors. These filters are localized in space, and the learned weights can be shared across different locations in a graph. An efficient variant of GNNs is graph convolutional networks (GCNs) [25], which is an upsurging semi-supervised graph-based deep learning framework that uses an efficient layer-wise propagation rule based on a first-order approximation of spectral graph convolutions. Hamilton *et al.* [26] propose GraphSAGE, a general inductive framework that generates embeddings by sampling and aggregating features from the local neighborhood of a graph node. Veličković *et al.* [27] present the graph attention network, which is a graph-based neural network architecture that uses an attention mechanism to assign self-attention scores to neighboring node embeddings. These scores indicate the importance of graph nodes to their corresponding neighbors on the feature aggregation process. Xu *et al.* [28] present theoretical foundations for analyzing the expres-

sive power of GNNs in an effort to capture different graph structures, and develop a graph isomorphism network whose goal is to map isomorphic graphs to the same representation and non-isomorphic ones to different representations.

While graph convolutional networks have achieved state-of-the-art performance on semi-supervised node classification tasks, they tend, however, to oversmooth the learned feature embeddings of graph nodes [29]. This is due largely to the fact that the graph convolution of the GCN model is a special form of graph Laplacian smoothing, which repeatedly and simultaneously adjusts the location of each graph node to the weighted average (i.e. geometric center) of its neighboring nodes. This averaging process causes an oversmoothing effect on the graph, as it reduces the high-frequency graph information and tends to flatten the graph. Moreover, oversmoothing causes features at nodes within each connected component to converge to the same value. Hence, nodes from different classes may be predicted to have similar labels, resulting in misclassification errors. Another drawback of Laplacian smoothing is shrinkage of the graph, as repeated iterations of the smoothing process causes the shrinking effect.

In this paper, we propose an anisotropic graph convolutional network (AGCN), which adopts the concept of anisotropic diffusion, previously used in image and geometry processing tasks, to overcome the aforementioned issues. The idea behind our proposed model is to integrate a nonlinearity term into the graph convolution to make it non-linear and/or anisotropic, resulting in a feature-preserving graph neural network. The main contributions of this work can be summarized as follows:

- We introduce a novel anisotropic graph convolutional network for semi-supervised learning.
- We learn efficient representations for node classification in an end-to-end fashion.
- We demonstrate that AGCN can be integrated into existing graph-based convolutional networks for semi-supervised learning using both co-training and self-training.
- Our extensive experimental results show competitive or superior performance of AGCN over standard baseline methods on several benchmark datasets.

The rest of this paper is organized as follows. In Section 2, we review important relevant work. In Section 3, we present the problem formulation and propose an anisotropic graph convolutional network architecture for semi-supervised learning. We discuss in detail the main components of the proposed framework and analyze the model complexity. In Section 4, we present experimental results to demonstrate the competitive performance of our approach on five standard benchmark datasets, including three citations networks and two image datasets. Finally, we conclude in Section 5 and point out future work directions.

## 2 Related Work

The basic goal of node classification is to predict the most probable labels of nodes in a graph. Graph convolutional networks (GCNs) have recently become the de facto model for semi-supervised node classification [25]. GCN uses an efficient layer-wise propagation rule, which is based on a first-order approximation of spectral graph convolutions. The feature vector of each graph node is updated by essentially applying a weighted sum of the features of its neighboring nodes. Monti *et al.* [30] present a mixture of networks (MoNet) model, a spatial-domain graph convolutional neural network that employs a mixture of Gaussian kernels with learnable parameters to model the weight function of pseudo-coordinates, which are associated to the neighboring nodes of each graph node. Liao *et al.* [31] propose the Lanczos network (LanczosNet), which employs the Lanczos algorithm to construct low-rank approximations of the graph Laplacian in order to facilitate efficient computations of matrix powers. Veličković *et al.* [32] present deep graph infomax, an unsupervised graph representation learning approach, which relies on training an encoder model to maximize the mutual information between local and global representations in graphs. Xu *et al.* [33] introduce a graph wavelet neural network, which is a GCN-based architecture that uses spectral graph wavelets in lieu of graph Fourier bases to define a graph convolution. Despite the fact that spectral graph wavelets can yield localization of graph signals in both spatial and spectral domains, they require explicit computation of the Laplacian eigenbasis, leading to a high computational complexity, especially for large graphs. In order to avoid this issue, recursive Chebyshev polynomial spectral filters can be employed.

While GCNs have shown great promise, achieving state-of-the-art performance on semi-supervised node classification, they are prone to oversmoothing the node features. In fact, the neighborhood aggregation scheme (i.e. graph convolution) of GCN is tantamount to applying Laplacian smoothing [29], which replaces each graph node with the average of its immediate neighbors. Therefore, repeated application of GCN yields smoother and smoother versions of the initial node features as the number of the network’s layers increases. As a result, the node features in deeper layers will eventually converge to the same value, and hence become too similar across different classes. Wu *et al.* [34] introduce a simple graph convolution by removing the nonlinear transition functions between the layers of graph convolutional networks and collapsing the resulting function into a single linear transformation via the powers of the normalized adjacency matrix with added self-loops for all graph nodes. However, this simple graph convolution acts as a low-pass filter, which attenuates all but the zero frequency, causing oversmoothing. Recently, significant strides have been made toward remedying the issue of oversmoothing in GCNs [35, 36]. Xu *et al.* [35] propose jumping knowledge networks, which employ dense skip connections to connect each layer of the network with the last layer to preserve the locality of node representations in order to circumvent oversmoothing. More recently, a normalization layer, which helps avoid oversmoothing by preventing learned representations of distant nodes from becoming indistinguishable,

has been proposed in [36]. This normalization layer is performed on intermediate layers during training, and the aim is to apply smoothing over nodes within the same cluster while avoiding smoothing over nodes from different clusters. While these approaches have shown slightly improved results using deeper GCNs, the issue of oversmoothing still remains a daunting task, as performance gains do not usually reflect the benefits of increasing the network depth.

### 3 Method

In this section, we describe the problem statement and introduce an anisotropic graph convolutional network for semi-supervised node classification. In particular, we examine the main building blocks of the proposed network architecture and analyze the complexity of the model. We also show that our proposed aggregation scheme seamlessly incorporates both the graph structure and the node features without sacrificing performance in an effort to alleviate oversmoothing of the learned node representations.

#### 3.1 Problem Formulation

Let  $\mathbb{G} = (\mathcal{V}, \mathcal{E})$  be a graph, where  $\mathcal{V} = \{1, \dots, N\}$  is the set of  $N$  nodes and  $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$  is the set of edges. We denote by  $\mathbf{A} = (\mathbf{A}_{ij})$  an  $N \times N$  adjacency matrix (binary or real-valued) whose  $(i, j)$ -th entry  $\mathbf{A}_{ij}$  is equal to the weight of the edge between neighboring nodes  $i$  and  $j$ , and 0 otherwise. We also denote by  $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)^\top$  an  $N \times F$  feature matrix of node attributes, where  $\mathbf{x}_i$  is an  $F$ -dimensional row vector for node  $i$ .

Learning latent representations of nodes in a graph aims at encoding the graph structure into low-dimensional embeddings, such that both structural and semantic information are captured. More precisely, the purpose of network/graph embedding is to learn a mapping  $\varphi : \mathcal{V} \rightarrow \mathbb{R}^P$  that maps each node  $i$  to a  $P$ -dimensional vector  $\mathbf{z}_i$ , where  $P \ll N$ . These learned node embeddings can then be used as input to learning algorithms for downstream tasks, such as node classification.

Given the labels of a subset of the graph nodes (or their corresponding final output embeddings), the objective of semi-supervised learning is to predict the unknown labels of the other nodes. More specifically, let  $\mathcal{D}_K = \{(\mathbf{z}_i, y_i)\}_{i=1}^K$  be the set of labeled final output node embeddings  $\mathbf{z}_i \in \mathbb{R}^P$  with associated known labels  $y_i \in \mathcal{Y}_K$ , and  $\mathcal{D}_U = \{\mathbf{z}_i\}_{i=K+1}^{K+U}$  be the set of unlabeled final output node embeddings, where  $K + U = N$ . Then, the problem of semi-supervised node classification is to learn a classifier  $f : \mathcal{V} \rightarrow \mathcal{Y}_K$ . That is, the goal is to predict the labels of the set  $\mathcal{D}_U$ .

It is important to note that for multi-class classification problems, the label of each node  $i$  (or its final output embedding  $\mathbf{z}_i$ ) in the labeled set  $\mathcal{D}_K$  can be represented as a  $C$ -dimensional one-hot vector  $\mathbf{y}_i \in \{0, 1\}^C$ , where  $C$  is the number of classes.

#### 3.2 Proposed Approach

Graph convolutional networks learn a new feature representation for each node such that nodes with the same labels have similar features [25]. Given a graph  $\mathbb{G} = (\mathcal{V}, \mathcal{E})$  with adjacency matrix  $\mathbf{A} \in \mathbb{R}^{N \times N}$  and feature matrix  $\mathbf{X} \in \mathbb{R}^{N \times F}$ , the layer-wise feature diffusion rule of an  $L$ -layer GCN is given by

$$\mathbf{S}^{(\ell)} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(\ell)}, \quad \ell = 0, \dots, L-1, \quad (1)$$

where  $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$  is the adjacency matrix with self-added loops,  $\mathbf{I}_N$  is the identity matrix,  $\tilde{\mathbf{D}} = \text{diag}(\tilde{d}_i)$  is the diagonal degree matrix whose  $i$ -th diagonal entry is the degree of node  $i$  with added self-loops, and  $\mathbf{H}^{(\ell)} \in \mathbb{R}^{N \times F_\ell}$  is the input feature matrix of the  $\ell$ -th layer with  $F_\ell$  feature maps. The input of the first layer is the original feature matrix  $\mathbf{H}^{(0)} = \mathbf{X}$ .

Using the feature diffusion rule of GCN is tantamount to applying a weighted sum of the features of neighboring nodes normalized by their degrees, which essentially performs Laplacian smoothing on the graph [29, 34]. In other words, the smooth feature matrix  $\mathbf{S}^{(\ell)}$  is obtained by applying Laplacian smoothing to the input feature matrix at the  $\ell$ -th layer. Intuitively, the Laplacian flow repeatedly and simultaneously adjusts the location of each graph node to the geometric center of its neighboring nodes. Although the Laplacian smoothing flow is simple and fast, it produces, however, the shrinking effect and an oversmoothing result.

Motivated by the good performance of anisotropic diffusion in image and mesh denoising [37–39], and in an effort to tackle the issues of oversmoothing and shrinking effect of GCN, we propose an anisotropic graph convolutional network (AGCN) for semi-supervised node classification by incorporating a nonlinear smoothness term into the GCN feature diffusion rule. This nonlinearity term, which quantifies the dissimilarity between learned node embeddings, plays a pivotal role in preventing these learned representations from becoming increasingly similar, and hence alleviates the issue of oversmoothing. In addition, it tackles the shrinking effect by precluding the learned node representations from converging to the same value.

**Anisotropic feature diffusion.** We define a layer-wise anisotropic feature diffusion rule for node features in the  $\ell$ -th layer as follows:

$$\mathbf{G}^{(\ell)} = \left(1 - \exp(-\beta \text{tr}^2(\mathbf{H}^{(\ell)\top} \tilde{\mathbf{L}} \mathbf{H}^{(\ell)}))\right) \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(\ell)}, \quad (2)$$

where  $\beta$  is a nonnegative hyper-parameter that is often fine-tuned via grid search, and  $\text{tr}(\mathbf{H}^{(\ell)\top} \tilde{\mathbf{L}} \mathbf{H}^{(\ell)})$  is a Laplacian smoothness term given by

$$\text{tr}(\mathbf{H}^{(\ell)\top} \tilde{\mathbf{L}} \mathbf{H}^{(\ell)}) = \frac{1}{2} \sum_{i,j=1}^N \tilde{\mathbf{A}}_{ij} \|\mathbf{h}_i^{(\ell)} - \mathbf{h}_j^{(\ell)}\|^2, \quad (3)$$

with  $\mathbf{H}^{(\ell)} = (\mathbf{h}_1^{(\ell)}, \dots, \mathbf{h}_N^{(\ell)})^\top$ ;  $\mathbf{h}_i^{(\ell)}$  is an  $F_\ell$ -dimensional hidden representation (embedding) vector of the  $i$ -th node at the  $\ell$ -th layer,  $\text{tr}(\cdot)$  denotes the trace operator,  $\|\cdot\|$  denotes the 2-norm, and  $\tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{A}}$  is an  $N \times N$  Laplacian matrix.

For each pair of similar embeddings at the  $\ell$ -th layer, the Laplacian smoothness term enforces their predictions to be close

to each other. The strength of this smoothness is determined by the weight of the edge between neighboring nodes, meaning that connected nodes will have similar predictions.

The Laplacian smoothness term plays a crucial role not only in explicitly taking into consideration the correlation between embeddings, but also in preserving the locality of nodes to be embedded. In other words, two nodes or their attributes  $\mathbf{x}_i$  and  $\mathbf{x}_j$  that are close to each other in the original graph (i.e. adjacent nodes in  $\mathcal{V}$ ) are encoded as embeddings  $\mathbf{h}_i^{(\ell)}$  and  $\mathbf{h}_j^{(\ell)}$  that are more likely to be close to each other in the embedding vector space. Such a locality-preserving property is of paramount importance in classification tasks.

The nonlinearity term  $1 - \exp(-\beta \text{tr}^2(\mathbf{H}^{(\ell)\top} \tilde{\mathbf{L}} \mathbf{H}^{(\ell)}))$  can be regarded as an oversmoothing “stopping” function. In fact, it only incurs a small penalty when similar nodes with a large smoothness strength  $\tilde{\mathbf{A}}_{ij}$  have different learned embeddings. Hence, it reduces the oversmoothing effect on the learned graph features.

**Anisotropic aggregation procedure.** The anisotropic feature diffusion rule can be written in vector form as follows:

$$\mathbf{g}_i^{(\ell)} = \sum_{j=1}^N \alpha_{ij}^{(\ell)} \mathbf{h}_j^{(\ell)} \quad (4)$$

where  $\mathbf{g}_i^{(\ell)}$  is the  $i$ -th row of  $\mathbf{G}_i^{(\ell)}$ , and  $\alpha_{ij}^{(\ell)}$  is the layer-wise weight coefficient given by

$$\alpha_{ij}^{(\ell)} = \left(1 - \exp(-\beta \text{tr}^2(\mathbf{H}^{(\ell)\top} \tilde{\mathbf{L}} \mathbf{H}^{(\ell)}))\right) \frac{\tilde{\mathbf{A}}_{ij}}{\sqrt{\tilde{d}_i \tilde{d}_j}} \quad (5)$$

which is equal to the nonlinearity term times the weight of the GCN neighborhood aggregation. The anisotropic aggregation scheme is illustrated in Figure 1. Notice how features are propagated from 1-hop (i.e. immediate) neighbors to multi-hop (i.e. distant) neighbors as the number of layers increases.

Unlike the GCN weight which only takes into account the topological structure of the graph, our proposed anisotropic feature diffusion rule seamlessly leverages both the topological structure and nodal attributes for aggregating node representations. Also, it is important to note that compared to existing neighborhood aggregation approaches [25–27], the weight coefficient  $\alpha_{ij}^{(\ell)}$  of our aggregation scheme is layer-aware, and hence prevents the learned representations from becoming indistinguishable thanks to the synergy between the nonlinearity term that helps mitigate oversmoothing of the node features and the GCN weight that captures the graph structure.

As illustrated in Figure 1, the proposed anisotropic feature diffusion rule follows a neighborhood aggregation or a message passing algorithm to learn a node representation by propagating representations of its immediate neighbors, where the latent representation of each node is initialized to the node’s input features. The latent representation of each graph node at a given layer is defined as a weighted sum of its immediate neighbors’ representations from the previous layer. As the number of layers increases, the node features are propagated to higher-order neighborhoods.

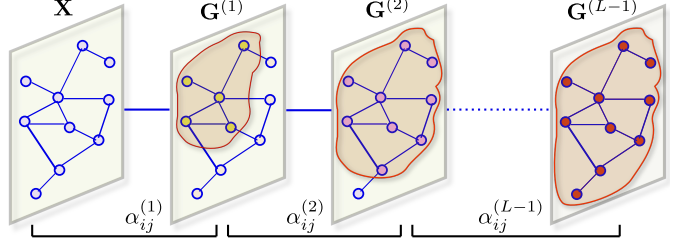


Figure 1: Schematic layout of the anisotropic aggregation procedure.

**Learning embeddings.** Given the anisotropically smooth feature matrix  $\mathbf{G}^{(\ell)}$  at the  $\ell$ -th layer as an input, the output feature matrix  $\mathbf{G}^{(\ell+1)}$  of our proposed AGCN model is obtained by applying the following layer-wise propagation rule:

$$\mathbf{G}^{(\ell+1)} = \sigma(\mathbf{G}^{(\ell)} \mathbf{W}^{(\ell)}), \quad \ell = 0, \dots, L-1, \quad (6)$$

which is basically a node embedding transformation that projects the input  $\mathbf{G}^{(\ell)} \in \mathbb{R}^{N \times F_\ell}$  into a trainable weight matrix  $\mathbf{W}^{(\ell)} \in \mathbb{R}^{F_\ell \times F_{\ell+1}}$  with  $F_{\ell+1}$  feature maps, followed by a point-wise non-linear activation function  $\sigma(\cdot)$  such as  $\text{ReLU}(\cdot) = \max(0, \cdot)$ , assuming that  $F_{\ell+1} \leq F_\ell \ll N$ . It is worth pointing out that after feature aggregation, performing the AGCN layer-wise propagation rule amounts to applying a multi-layer perceptron to the anisotropic feature matrix. In other words, a node latent representation at layer  $\ell$  is transformed linearly via a learned weight matrix to produce the node latent representation at the next layer.

**Model prediction.** The embedding  $\mathbf{G}^{(L)}$  of the last layer of AGCN contains the final output node embeddings, and captures the neighborhood structural information of the graph within  $L$  hops. This final node representation can be used as input for downstream tasks such as graph classification, clustering, visualization, link prediction, and node classification. Since the latter task is the focus of this paper, we apply a softmax classifier as follows:

$$\hat{\mathbf{Y}} = \text{softmax}(\mathbf{G}^{(L)} \mathbf{W}^{(L)}), \quad (7)$$

where  $\mathbf{W}^{(L)} \in \mathbb{R}^{F_L \times C}$  is a trainable weight matrix of the last layer,  $C$  is the total number of classes,  $\text{softmax}(\mathbf{x}) = \exp(\mathbf{x}) / \sum_{c=1}^C \exp(\mathbf{x}_c)$  is an activation function applied row-wise, and  $\hat{\mathbf{Y}} \in \mathbb{R}^{N \times C}$  is the matrix of predicted labels for graph nodes.

**Model complexity.** For simplicity, we assume the feature dimensions are the same for all layers, i.e.  $F_\ell = F$  for all  $\ell$ , with  $F \ll N$ . The time complexity of an  $L$ -layer AGCN is  $\mathcal{O}(L|\mathcal{E}|F + LN F^2)$ , where  $|\mathcal{E}|$  denotes the number of graph edges. Note that multiplying the normalized adjacency matrix with an embedding costs  $\mathcal{O}(|\mathcal{E}|F)$  in time, while multiplying an embedding with a weight matrix costs  $\mathcal{O}(N F^2)$ . Also, noting that  $\text{tr}(\mathbf{H}^{(\ell)\top} \tilde{\mathbf{L}} \mathbf{H}^{(\ell)}) = \text{tr}(\tilde{\mathbf{L}} \mathbf{H}^{(\ell)} \mathbf{H}^{(\ell)\top})$ , it follows that computing this trace operator requires  $N F^2$  scalar multiplications. Hence, the nonlinearity term of AGCN has complexity  $\mathcal{O}(N F^2)$ .

For memory complexity, an  $L$ -layer AGCN requires  $\mathcal{O}(LNF + LF^2)$  in memory, where  $\mathcal{O}(LNF)$  is for storing all embeddings and  $\mathcal{O}(LF^2)$  is for storing all layer-wise weight matrices.

Therefore, the proposed AGCN model has the same time and memory complexity as GCN, while being effective at alleviating the issue of oversmoothing.

**Model training.** For semi-supervised multi-class classification, the neural network weight parameters are learned by minimizing the cross-entropy loss function

$$\mathcal{L} = - \sum_{i \in \mathcal{Y}_K} \sum_{c=1}^C \mathbf{Y}_{ic} \log \hat{\mathbf{Y}}_{ic}, \quad (8)$$

over the set  $\mathcal{Y}_K$  of all labeled nodes using gradient descent, where  $\mathbf{Y}_{ic}$  is equal 1 if node  $i$  belongs to class  $c$ , and 0 otherwise; and  $\hat{\mathbf{Y}}_{ic}$  is the  $(i, c)$ -element of the matrix  $\hat{\mathbf{Y}}$  from the softmax function, i.e. the probability that the network associates the  $i$ -th node with class  $c$ .

## 4 Experiments

In this section, we conduct extensive experiments to evaluate the performance of the proposed AGCN framework on several benchmark datasets and carry out a comprehensive comparison with several baseline methods. In all experiments, we consider a two-layer AGCN for semi-supervised node classification

$$\hat{\mathbf{Y}} = \text{softmax}(\text{ReLU}(\mathbf{G}^{(0)} \mathbf{W}^{(0)}) \mathbf{W}^{(1)}), \quad (9)$$

where  $\mathbf{W}^{(0)} \in \mathbb{R}^{F \times F_1}$  is a trainable input-to-hidden weight matrix for a hidden layer with  $F_1$  feature maps,  $\mathbf{W}^{(1)} \in \mathbb{R}^{F_1 \times C}$  is a trainable hidden-to-output weight matrix with  $C$  denoting the number of classes, and  $\mathbf{G}^{(0)}$  is an  $N \times F$  matrix given by

$$\mathbf{G}^{(0)} = \left(1 - \exp(-\beta \text{tr}^2(\mathbf{X}^\top \tilde{\mathbf{L}} \mathbf{X}))\right) \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}. \quad (10)$$

**Datasets.** We demonstrate and analyze the performance of the proposed AGCN model on three citation networks (Cora, Citseer, and Pubmed) and two image datasets (MNIST and CIFAR10). The summary descriptions of these benchmark datasets are as follows:

- Cora is a citation network dataset consisting of 2,708 nodes representing scientific publications and 5,429 edges representing citation links between publications. All publications are classified into 7 classes (research topics). Each node is described by a binary feature vector indicating the absence/presence of the corresponding word from the dictionary, which consists of 1,433 unique words.
- Citeseer is a citation network dataset composed of 3,312 nodes representing scientific publications and 4,723 edges representing citation links between publications. All publications are classified into 6 classes (research topics). Each node is described by a binary feature vector indicating the absence/presence of the corresponding word from the dictionary, which consists of 3,703 unique words.

- Pubmed is a citation network dataset containing 19,717 scientific publications pertaining to diabetes and 44,338 edges representing citation links between publications. All publications are classified into 3 classes. Each node is described by a TF/IDF weighted word vector from the dictionary, which consists of 500 unique words.
- CIFAR10 is an image dataset consisting of 60,000 natural color images, each of which is  $32 \times 32 \times 3$  in size and has three color channels (RGB), as shown in Figure 2 (left). All images in the dataset are classified into 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 testing images. In our experiments, we randomly select 10,000 images (1,000 images per class) to perform evaluation. For each image, a convolutional neural network is used to extract a feature descriptor, followed by graph construction using the  $k$ -nearest neighbors algorithm with  $k = 8$ .
- MNIST is an image dataset consisting of 70,000 grayscale images of handwritten digits from 0 to 9 (i.e. 10 classes), as shown in Figure 2 (right). There are 60,000 training images and 10,000 testing images taken from American Census Bureau employees and American high school students, respectively. We randomly select 1,000 images from each digit for evaluation. Each image is  $28 \times 28$  in size, and hence it is represented as a 784-dimensional feature vector. The  $k$ -nearest neighbors algorithm with  $k = 8$  is used to construct the graph for each handwritten digit.



Figure 2: Sample images from CIFAR10 (left) and MNIST (right).

**Baseline methods.** We evaluate the performance of AGCN against several graph-based feature learning models, including DeepWalk [16], Chebyshev networks (ChebyNet) [24], GCN [25], mixture model network (MoNet) [30], graph attention network (GAT) [27], jumping knowledge network (JK-Net) [35], deep graph infomax (DGI) [32], graph wavelet neural network (GWNN) [33], Lanczos network (LanczosNet) [31], graph isomorphism network (GIN) [28], simple graph convolution (SGC) [34], and GCN with pair normalization (GCN-PN) [36]. For baselines, we mainly consider methods that are closely related to AGCN and/or the ones that are state-of-the-art node classification frameworks. A brief description of these standard baselines can be summarized as follows:

- DeepWalk is a deep learning based framework that learns



latent representations of nodes in a graph by leveraging local information obtained from truncated random walks.

- ChebyNet is an efficient spectral-domain graph convolutional neural network that uses recursive Chebyshev polynomial spectral filters to avoid explicit computation of the Laplacian eigenvectors.
- GCN is a semi-supervised graph-based deep learning framework that uses an efficient layer-wise propagation rule that is based on a first-order approximation of spectral graph convolutions.
- MoNet is a spatial-domain graph convolutional neural network that employs a mixture of Gaussian kernels with learnable parameters to model the weight function of pseudo-coordinates, which are associated to the neighboring nodes of each graph node.
- GAT is graph-based neural network architecture that uses an attention mechanism to assign self-attention scores to neighboring node embeddings.
- JK-Net is a graph representation learning approach that learns to selectively exploit information from neighborhoods of differing locality and combines different aggregations at the last layer.
- DGI is graph representation learning framework, which leverages local mutual information maximization across the graph’s patch representations to learn node embeddings in an unsupervised manner.
- GWNN is a spectral convolutional neural network, which uses spectral graph wavelets for node feature aggregation.
- LanczosNet is a multiscale graph convolutional network for learning node embedding, which leverages the Lanczos algorithm to construct a low rank approximation of the graph Laplacian for graph convolution.
- GIN is a simple graph neural network architecture, which maps isomorphic graphs to the same representation and non-isomorphic ones to different representations. It is proved to be as powerful as the Weisfeiler-Lehman test for graph isomorphism.
- SGC is a simplified GCN architecture, which consists of a linear feature propagation scheme, followed by multi-class logistic regression.
- GCN-PN is a graph convolutional network, which uses a pair normalization layer to help prevent oversmoothing in deeper graph neural networks.

We also compare our approach with multi-layer perceptron (MLP), manifold regularization (ManiReg) [40], semi-supervised embedding (SemiEmb) [41], LP [42], iterative classification algorithm (ICA) [43], and Planetoid [44].

**Implementation details.** For fair comparison with prior work, we follow the same experimental setup as [25]. For the CIFAR10 and MNIST image datasets, we randomly select 3,000

images as labeled samples and used the remaining images as unlabeled samples. For unlabeled samples, we select 1,000 images for validation and used the remaining 6,000 images as test samples. All the reported accuracy results of AGCN are averaged over 10 runs with different splits for training, validation and test sets. We train the proposed AGCN model for 200 epochs using Adam optimizer [45] with learning rate 0.01. The training is stopped when the validation loss does not decrease after 10 consecutive epochs. The values of the cross-entropy metric are recorded at the end of each epoch on the training set. The performance comparison between AGCN and GCN over training epochs on the training set is illustrated in Figure 3, which shows that the proposed AGCN model yields lower training loss values, indicating higher predictive accuracy. The value of the hyperparameter  $\beta$  is optimized using grid search over the set  $\{0, 0.1, 0.2, \dots, 5\}$ .

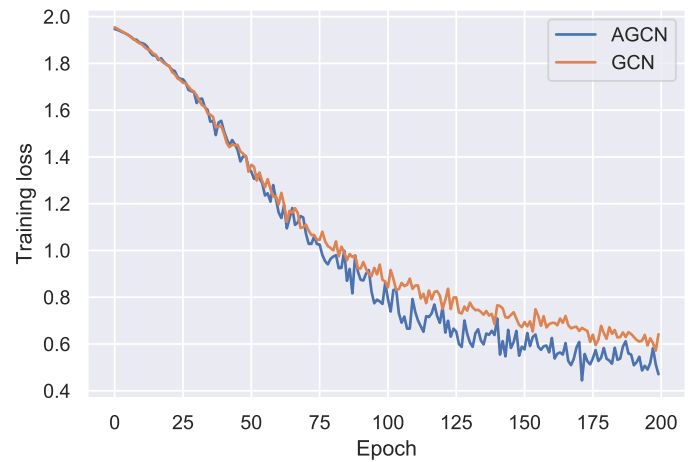


Figure 3: Model training history comparison between GCN and proposed AGCN model on the Cora dataset.

## 4.1 Results

The performance of our model is evaluated by conducting a comprehensive comparison with standard baseline methods for node classification using average accuracy as an evaluation metric. The average classification accuracy results in percent are summarized in Table 1. Results for baseline methods on the citation networks are taken from the GAT paper [27], and from the corresponding baseline papers for the image datasets. As shown in Table 1, our AGCN model outperforms GCN on the citation networks as well as on the image datasets. While DeepWalk does well on the MNIST dataset, it performs poorly on the citation networks compared to AGCN. In addition, AGCN outperforms GAT on the image datasets and the Pubmed citation network, and performs on par with GAT on the Cora dataset. The average accuracy of AGCN on the CIFAR10 dataset is 70%, indicating a performance improvement of 3.4% over GAT. Similarly, AGCN performs on par with DGI on Citeseer, but yields better performance on Cora and Pubmed. Also, AGCN achieves better performance than GWNN, a spectral approach that requires computing an eigendecomposition of the Lapla-

cian, which is usually time- and space-consuming. We can see that AGCN outperforms GIN and LanczosNet across all the citation network datasets, with performance gains of 5.4% and 3.5% on Cora and 5.4% and 3.5% on Citeseer, respectively. Moreover, AGCN performs better than SGC and JK-Net on Cora and Pubmed. Interestingly, despite its simplicity, AGCN achieves a higher accuracy than GCN-NP with improvements of 5.8% and 4% on Citeseer and Cora, respectively. These results demonstrate the significant prediction ability of AGCN in semi-supervised node classification. It is important to mention that both JK-Net and GCN-PN use additional steps such as skip connections and normalization layers to tackle the issue of oversmoothing on graph neural networks, while our proposed AGCN model integrates an oversmoothing prevention term into its neighborhood aggregation scheme using a single step. In other words, the proposed AGCN network is trained in an end-to-end fashion, and eliminates the need to augment deep GNN models with normalization layers or by inserting residual/skip connections between the network’s layers in order to improve performance.

Table 1: Classification accuracy results on three citations networks and two image datasets. Boldface numbers indicate the best classification performance.

Method	Average accuracy (%)				
	Cora	Citeseer	Pubmed	MNIST	CIFAR10
MLP	55.1	46.5	71.4	—	—
ManiReg	59.5	60.1	70.7	94.6	59.7
SemiEmb	59.0	59.6	71.7	—	—
LP	68.0	45.3	63.0	83.4	60.4
DeepWalk	67.2	43.2	65.3	<b>95.3</b>	61.3
ICA	75.1	69.1	73.9	—	—
Planetoid	75.7	64.7	77.2	—	—
ChebyNet	81.2	69.8	74.4	—	—
GCN	81.5	70.3	79.0	91.0	61.0
MoNet	81.7	—	78.8	—	—
GAT	<b>83.0</b>	72.5	79.0	92.8	66.6
DGI	82.3	71.8	76.8	—	—
GWNN	82.8	71.7	79.1	—	—
LanczosNet	79.5	66.2	78.3	—	—
GIN	77.6	66.1	77.0	—	—
SGC	81.0	71.9	78.9	—	—
JK-Net	82.7	<b>73.0</b>	77.9	—	—
GCN-PN	79.0	66.0	78.0	—	—
AGCN	<b>83.0</b>	71.8	<b>79.5</b>	93.1	<b>70.0</b>

Using box plots, Figure 4 displays the visual differences in terms of accuracy among AGCN, GAT and GCN on the Cora, Citeseer and Pubmed citation networks. As can be seen, the distribution of the AGCN model has less variability than GCN and GAT on document classification tasks. For instance, the median accuracy score for AGCN on the Pubmed dataset indicates a significant difference in performance between AGCN and the two baseline methods. In addition, the box for AGCN is short,

meaning that the accuracy values consistently hover around the average accuracy. However, the box for GAT is taller, implying variable accuracy values compared to AGCN.

**Co-training and self-training results.** Using the co-training and self-training approaches [29], we compare AGCN with GCN on the Cora, Citeseer and Pubmed datasets with label rates (i.e. proportion of labeled nodes that are used for training) 0.036, 0.052, and 0.003, respectively.

We apply co-training and self-training approaches as well as their intersection and union to train our AGCN model and compare it to GCN. The accuracy results for these four approaches are reported in Figures 5, 6 and 7 on the Cora, Citeseer and Pubmed citation networks, respectively, using training rates of 0.5%, 2% and 4% for Cora and Citseer, and 0.03%, 0.05% and 0.1% for Pubmed. In each bar plot, the bars display the mean and standard error accuracy over 10 runs for both AGCN and GCN using co-training, self-training, union and intersection. In co-training, a partially absorbing random walk is used to find the confidence of node  $i$  belongs to class  $c$ . The most confident nodes are then added to the training set with label  $c$  to train the AGCN model. In self-training, AGCN is applied to find the most confident nodes based on the softmax scores  $\hat{\mathbf{Y}} \in \mathbb{R}^{N \times C}$  given by Eq. (9). Then, the most confident nodes are added to the labeled set. On the other hand, union and intersection are a combination of co-training and self-training. Union expands the label set with the most confident predictions obtained by random walk and those obtained by AGCN. As can be seen in these figures, our AGCN framework outperforms GCN in most of the cases, particularly for small training sizes. Moreover, notice that the standard deviations are much smaller than the accuracy improvements, indicating that AGCN is robust to random selection of training and test data. Overall, AGCN is consistently the best performing method, delivering robust classification accuracy results.

## 4.2 Statistical Significance Analysis

In this subsection, we conduct statistical significance tests to compare GCN, GAT and AGCN with the objective of selecting the best performing model. More precisely, we apply one-way analysis of variance (ANOVA) to verify whether there is a statistical difference between their mean accuracy scores. ANOVA tests the hypothesis that all group means are equal versus the alternative hypothesis that at least one group is different from the others:

$$\begin{aligned} H_0 : \mu_1 = \mu_2 = \mu_3 \\ H_1 : \text{not all group means are equal} \end{aligned} \quad (11)$$

where  $\mu_1, \mu_2, \mu_3$  denote the population means for GCN, GAT and AGCN, respectively. While ANOVA is based on the assumption that all sample populations are normally distributed, it is, however, known to be robust to modest violations of the normality assumption.

We perform one-way ANOVA for the accuracy scores data obtained by GCN, GAT and AGCN on the Cora, Citeseer and Pubmed datasets. These results correspond to 10 runs with different splits for training, validation and test sets. As shown in

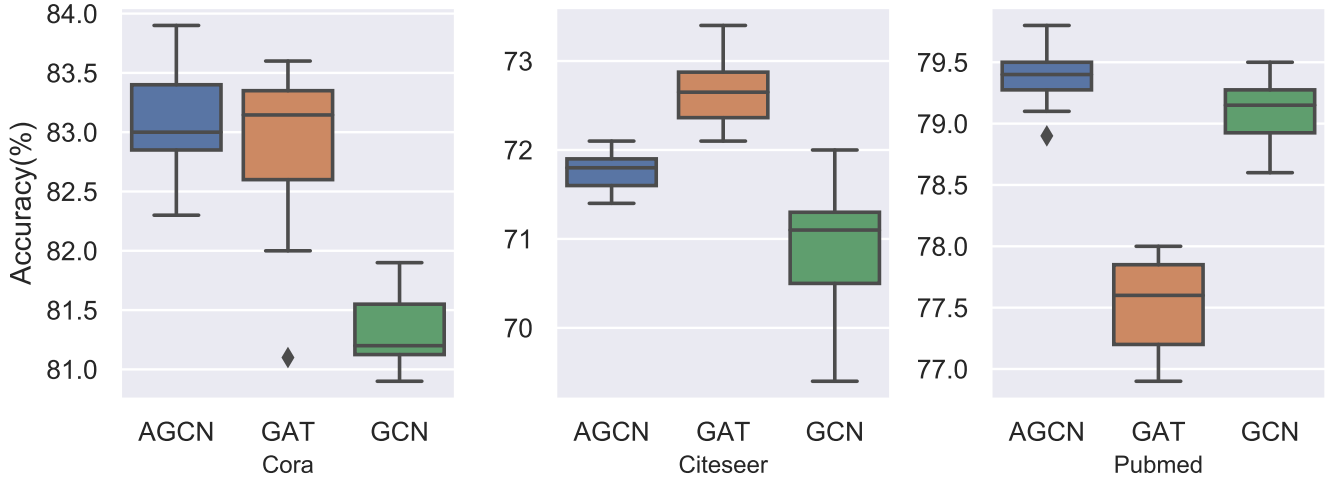


Figure 4: Accuracy distributions of AGCN, GAT and GCN on the Cora, Citeseer and Pubmed citation networks.

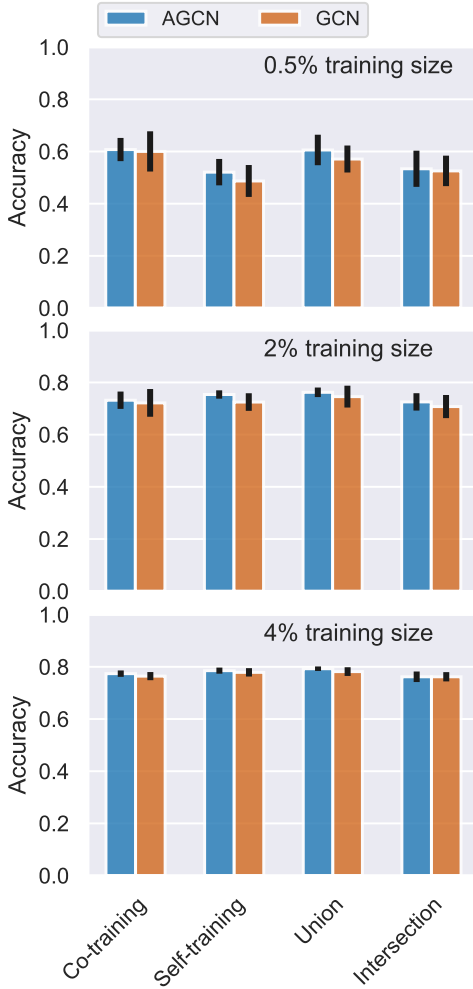


Figure 5: Classification accuracy of AGCN compared to GCN for different training set sizes on the Cora dataset.

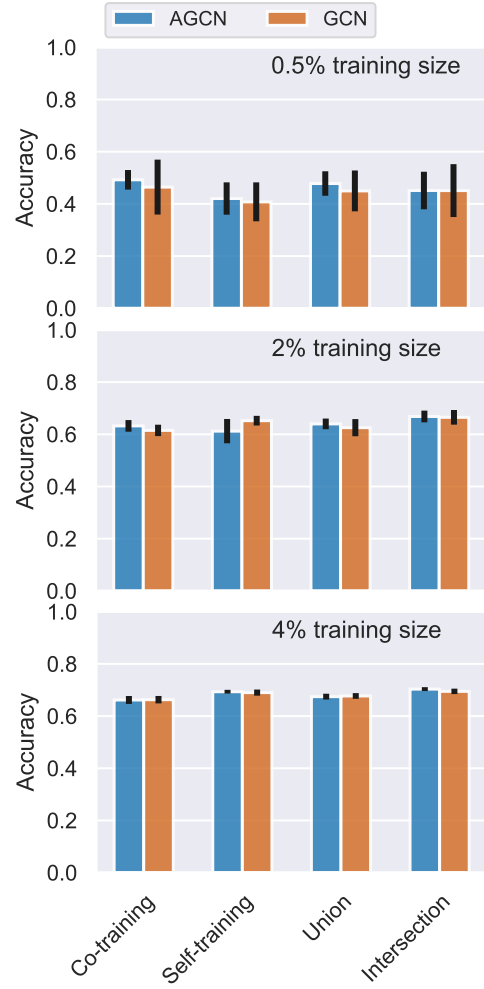


Figure 6: Classification accuracy of AGCN compared to GCN for different training set sizes on the Citeseer dataset.

Table 2, the small  $p$ -values ( $< 0.05$ ) indicate that differences between accuracy means are statistically significant, where  $\alpha = 0.05$  is the significance level. A significance level of 0.05 indi-

cates a 5% risk of concluding that a difference exists when there is no actual difference.

Since our ANOVA analysis shows an overall statistically sig-



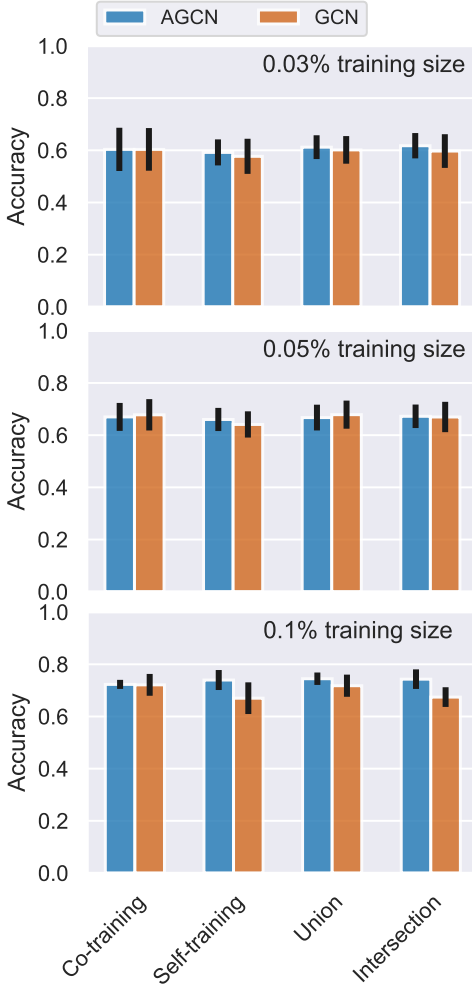


Figure 7: Classification accuracy of AGCN compared to GCN for different training set sizes on the Pubmed dataset.

Table 2: One-way ANOVA  $p$ -values for the accuracy scores data obtained by AGCN, GCN and GAT on the Cora, Citeseer and Pubmed datasets.

	Cora	Citeseer	Pubmed
$p$ -value	$1.0 \times 10^{-3}$	$2.51 \times 10^{-7}$	$1.21 \times 10^{-6}$

nificant difference in group means, we now need to determine which specific groups (compared with each other) are different in terms of mean accuracies by performing multiple pairwise comparison (post-hoc comparison) analysis using Tukey’s test, which compares all possible pairs of means. The pairwise multiple comparison results of GCN, GAT, and AGCN methods using Tukey’s test on the Cora, Citeseer and Pubmed datasets are shown in Figures 8, 9 and 10, respectively. The table above each figure reports the results of the multiple comparison of means. In the case of the Cora dataset, for instance, we can see from the result shown in the table of Figure 8 that we reject the hypotheses that “AGCN and GCN” and “GAT and GCN” have the same mean, but we fail to reject the “AGCN and GAT” pair and

conclude that they have equal mean accuracies. In the mean-diff column, the difference of accuracy means between related groups is reported, followed by lower and upper limits for 95% confidence intervals for the true mean difference. In the reject column, “True” indicates that there is significant evidence to reject the null hypothesis at the given significance level, i.e. there is a significant statistical difference between the means. The results from Tukey’s test reported in the tables of Figures 9 and 10 show that we reject the hypotheses that “AGCN and GAT”, “AGCN and GCN” and “GAT and GCN” have the same mean, indicating statistical significant differences.

In each figure, the 95% confidence intervals plots are displayed as horizontal bars. The blue bar in Figure 8 shows the comparison interval for the AGCN mean accuracy, which does not overlap with the comparison interval for the GCN mean accuracy, shown in red. The comparison interval for the GAT mean accuracy, shown in gray, overlaps with the comparison interval for the AGCN mean accuracy. Hence, the accuracy means for AGCN and GAT are not significantly different from each other on the Cora dataset, meaning that both methods performs on par with each other.

In Figure 9, the blue bar shows the comparison interval for the AGCN mean accuracy, which does not overlap with the comparison intervals for the GCN and GAT mean accuracies, shown in red. The disjoint comparison intervals indicate that the group means are significantly different from each other. Similarly, the blue bar in Figure 10 shows the comparison interval for the AGCN mean accuracy, which does not overlap with the comparison intervals for the GCN and GAT mean accuracies, shown in red. Hence, we conclude that AGCN is the best performing method on the Pubmed dataset. This visual observation is consistent with the results reported in Table 1.

group1	group2	meandiff	lower	upper	reject
AGCN	GAT	-0.0004	-0.00472	0.00392	False
AGCN	GCN	-0.0079	-0.01222	-0.00358	True
GAT	GCN	-0.0075	-0.01182	-0.00318	True

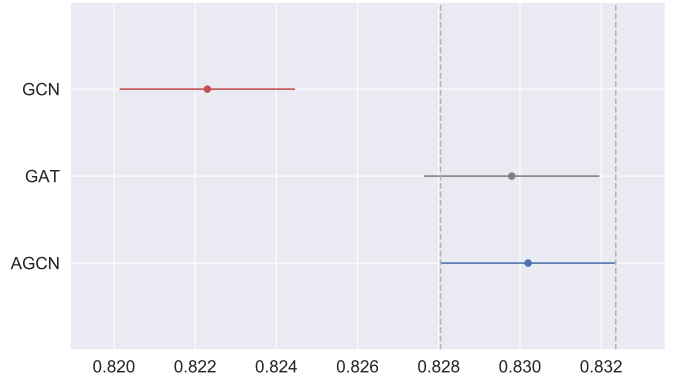


Figure 8: Pairwise multiple comparison between AGCN, GAT, and GCN methods using Tukey’s test on the Cora dataset.

group1	group2	meandiff	lower	upper	reject
AGCN	GAT	0.0078	0.00211	0.01349	True
AGCN	GCN	-0.0094	-0.01509	-0.00371	True
GAT	GCN	-0.0172	-0.02289	-0.01151	True

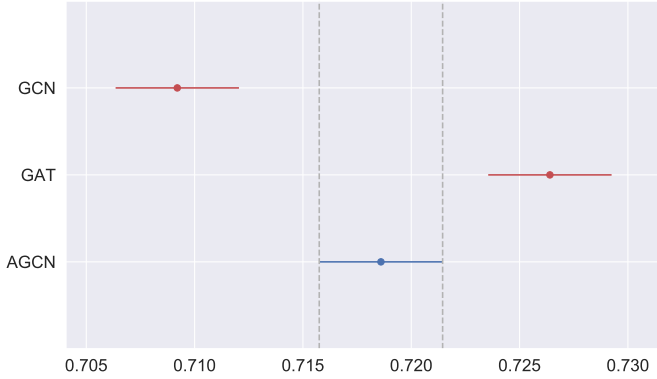


Figure 9: Pairwise multiple comparison between AGCN, GAT, and GCN methods using Tukey’s test on the Citeseer dataset.

group1	group2	meandiff	lower	upper	reject
AGCN	GAT	-0.009	-0.01193	-0.00607	True
AGCN	GCN	-0.0048	-0.00773	-0.00187	True
GAT	GCN	0.0042	0.00127	0.00713	True

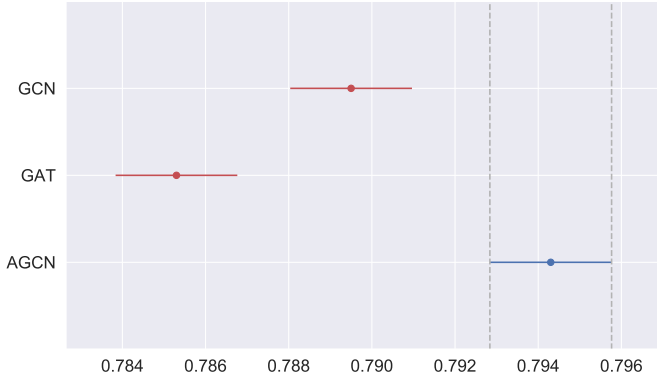


Figure 10: Pairwise multiple comparison between AGCN, GAT, and GCN methods using Tukey’s test on the Pubmed dataset.

### 4.3 Visualization

The feature embeddings learned by AGCN can be visualized using the t-Distributed Stochastic Neighbor Embedding (t-SNE) [46], which is a dimensionality reduction technique that is particularly well-suited for embedding high-dimensional data into a two- or three-dimensional space. Figure 11 displays the t-SNE embeddings of the output embeddings by the first convolutional layer of AGCN (top) and GCN (bottom) on the MNIST dataset. As can be seen, the two-dimensional embeddings corresponding to AGCN are more separable than the ones corresponding to GCN. With GCN features, the points are not discriminated very well, while with AGCN features the points are

discriminated much better and clearly show the clusters corresponding to the ten digit labels of the MNIST dataset. Hence, AGCN learns more discriminative features for node classification tasks, indicating the superior performance of anisotropic diffusion over linear diffusion. Moreover, Figure 11 shows that the AGCN approach is exploratory in nature in the sense that it can discover patterns and meaningful sub-groups in a dataset.

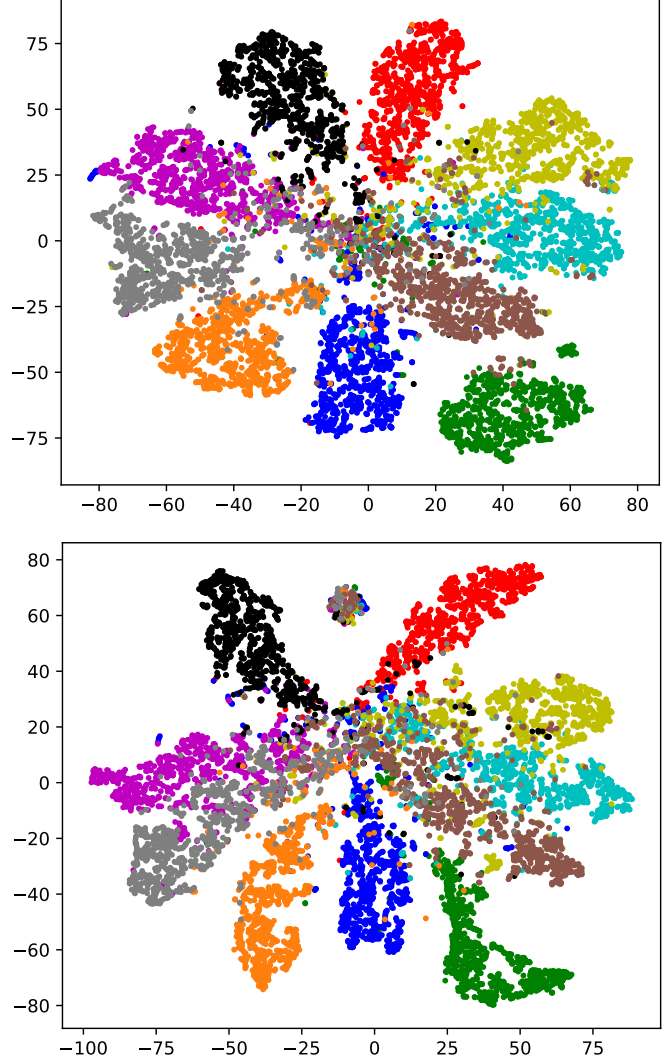


Figure 11: t-SNE feature visualization of the output embeddings by the first convolutional layer of AGCN (top) and GCN (bottom), respectively, on the MNIST dataset. Each color denotes a class.

### 4.4 Robustness to Oversmoothing

In order to assess robustness to oversmoothing, we study the performance variation for our multi-layer model on the Cora dataset with respect to the number of layers. Figure 12 shows how the node classification accuracy changes with the network’s depth. As can be seen, there is a sharp drop in GCN’s accuracy when the number of layers is larger than 4, while AGCN’s performance does not significantly degrade as the number of lay-

ers increases. Note that the performance gap between AGCN and GCN substantially increases when the network’s depth is beyond 4, indicating that AGCN is more robust to oversmoothing. It is worth pointing out that the objective of our proposed anisotropic convolution is to mitigate the oversmoothing issue that causes drops in classification performance for multi-layer GCNs, and not to show that the deeper AGCN, the better. Also, increasing the depth of the network leads to increased number of parameters, causing overfitting and hence resulting in performance drop. As shown in Figure 12, a 6-layer AGCN yields an accuracy of 80%, outperforming several baselines including LanczosNet, GIN and GCN-PN. The latter baseline is designed specifically for tackling the issue of oversmoothing in GCNs by employing a normalization layer after each convolutional layer.

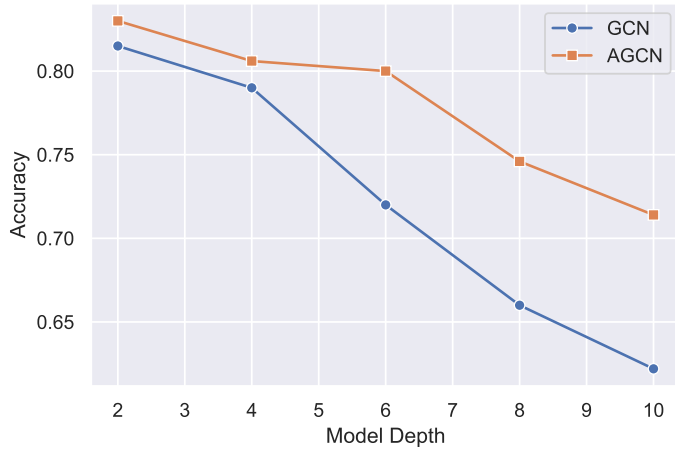


Figure 12: Performance comparison between AGCN and GCN on the Cora dataset as we increase the number of layers.

#### 4.5 Parameter Sensitivity Analysis

For each benchmark dataset used in the experiments, we test the performance of AGCN using different values for the hyper-parameter  $\beta$  of the anisotropic diffusion term. In the case of the Pubmed dataset, for instance, the effect of  $\beta$  on the performance of AGCN is illustrated in Figure 13, which shows the accuracy of AGCN along with the standard error for different values of  $\beta$ . As can be seen,  $\beta = 0.4$  yields the best value for the AGCN accuracy on the Pubmed dataset.

#### 4.6 Discussion

While the proposed AGCN model shows promising results for end-to-end learning on graphs and mitigates the issue of oversmoothing, its performance is, however, tied to optimizing via grid search with cross-validation the hyper-parameter of the anisotropic diffusion term for each dataset. Another shortcoming of AGCN is that it only takes into account immediate neighbors. This limitation can be circumvented through higher-order message passing by leveraging multi-hop neighbors using powers of the adjacency matrix and hence aggregating learned node representations from both immediate and distant neighbors.

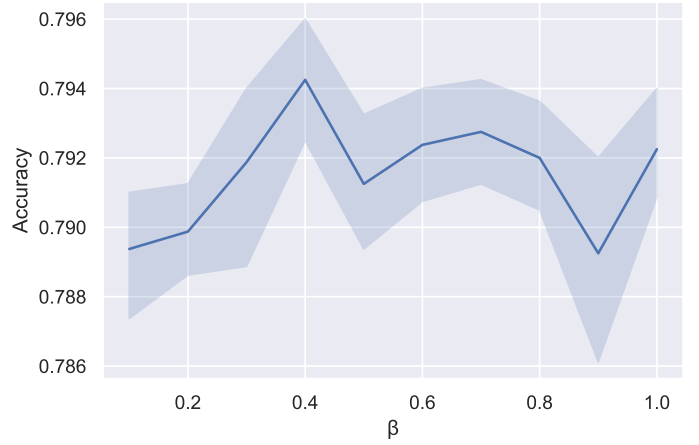


Figure 13: AGCN accuracy results for different values of  $\beta$  on the Pubmed dataset.

## 5 Conclusion

In this paper, we introduced an anisotropic graph convolutional network for semi-supervised node classification on graph-structured data by learning efficient representations in an end-to-end fashion. We incorporated a nonlinear smoothness term into the feature diffusion rule of the convolutional neural network in a bid to tackle the issues of oversmoothing and shrinking effect. We demonstrated through extensive experimental results the competitive or superior performance of AGCN in terms of classification accuracy over standard baseline methods on several benchmarks, including citation networks and image datasets. We also showed that AGCN can be integrated into existing graph-based convolutional networks for semi-supervised learning using both co-training and self-training. In addition, we performed a statistical analysis using analysis of variance and pairwise multiple comparison, showing that the performance of our model is better or comparable with the baselines. For future work, we plan to extend the proposed framework to heterogeneous information networks.

## References

- [1] S. Bhagat, G. Cormode, and S. Muthukrishnan, *Node Classification in Social Networks*. In: Aggarwal C. (eds), Springer, 2011.
- [2] F. Huang, X. Li, S. Zhang, J. Zhang, J. Chen, and Z. Zhai, “Overlapping community detection for multimedia social networks,” *IEEE Transactions on Multimedia*, vol. 19, no. 8, pp. 1881–1893, 2017.
- [3] L. Xu, T. Bao, L. Zhu, and Y. Zhang, “Trust-based privacy-preserving photo sharing in online social networks,” *IEEE Transactions on Multimedia*, vol. 21, no. 3, pp. 591–602, 2019.
- [4] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, “Attention based spatial-temporal graph convolutional networks for

- traffic flow forecasting,” in *AAAI Conference on Artificial Intelligence*, pp. 922–929, 2013.
- [5] S. Wang, Z. Chen, X. Yu, D. Li, J. Ni, L. Tang, J. Gui, Z. Li, H. Chen, and P. Yu, “Heterogeneous graph matching networks for unknown malware detection,” in *International Joint Conference on Artificial Intelligence*, pp. 3762–3770, 2013.
  - [6] S. Ktena, S. Parisot, E. Ferrante, M. Rajchl, M. Lee, B. Glocker, and D. Rueckert, “Metric learning with spectral graph convolutions on brain connectivity networks,” *NeuroImage*, vol. 169, no. 0, pp. 431–442, 2018.
  - [7] A. Ortega, P. Frossard, J. Kovacevic, J. Moura, and P. Vandergheynst, “Graph signal processing: Overview, challenges, and applications,” *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.
  - [8] Y. Chen, J. Wang, Y. Bai, G. C. n  n, and V. Saligrama, “Probabilistic semantic retrieval for surveillance videos with activity graphs,” *IEEE Transactions on Multimedia*, vol. 21, no. 3, pp. 704–716, 2019.
  - [9] Q. Peng and Y.-M. Cheung, “Automatic video object segmentation based on visual and motion saliency,” *IEEE Transactions on Multimedia*, vol. 21, no. 12, pp. 3083–3094, 2019.
  - [10] J. Gao and C. Xu, “CI-GNN: Building a category-instance graph for zero-shot video classification,” *IEEE Transactions on Multimedia*, 2020.
  - [11] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, “Deep learning via semi-supervised embedding,” in *Proc. International Conference on Machine Learning*, pp. 1168–1175, 2008.
  - [12] Z. Yang, W. Cohen, and R. Salakhutdinov, “Revisiting semi-supervised learning with graph embeddings,” in *Proc. International Conference on Machine Learning*, pp. 40–48, 2016.
  - [13] P. Goyal and E. Ferrara, “Graph embedding techniques, applications, and performance: A survey,” *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.
  - [14] L. Lu, Y. Lu, R. Yu, H. Di, L. Zhang, and S. Wang, “GAIM: Graph attention interaction model for collective activity recognition,” *IEEE Transactions on Multimedia*, vol. 22, no. 2, pp. 524–539, 2020.
  - [15] W. Hamilton, R. Ying, and J. Leskovec, “Representation learning on graphs: Methods and applications,” *IEEE Data Engineering Bulletin*, 2017.
  - [16] B. Perozzi, R. Al-Rfou, and S. Skiena, “DeepWalk: Online learning of social representations,” in *International Conference on Knowledge Discovery and Data Mining*, pp. 701–710, 2014.
  - [17] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *International Conference on Knowledge Discovery and Data Mining*, pp. 855–864, 2016.
  - [18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in Neural Information Processing*, pp. 3111–3119, 2013.
  - [19] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated graph sequence neural networks,” in *International Conference on Learning Representations*, 2016.
  - [20] X. Bresson and T. Laurent, “Residual gated graph convnets,” *arXiv preprint arXiv:1711.07553*, 2018.
  - [21] C. Zhuang and Q. Ma, “Dual graph convolutional networks for graph-based semi-supervised classification,” in *Proc. World Wide Web Conference*, 2018.
  - [22] H. Gao, Z. Wang, and S. Ji, “Large-scale learnable graph convolutional networks,” in *Conference on Knowledge Discovery and Data Mining*, 2018.
  - [23] W. Huang, T. Zhang, Y. Rong, and J. Huang, “Adaptive sampling towards fast graph representation learning,” in *Advances in Neural Information Processing*, pp. 1–10, 2018.
  - [24] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *Advances in Neural Information Processing*, pp. 3844–3852, 2016.
  - [25] T. Kipf and M. Welling, “Semi supervised classification with graph convolutional networks,” in *International Conference on Learning Representations*, pp. 1–14, 2017.
  - [26] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in Neural Information Processing*, pp. 1024–1034, 2017.
  - [27] P. Veli  kovi  , G. Cucurull, A. Casanova, A. Romero, P. Li  , and Y. Bengio, “Graph attention networks,” in *International Conference on Learning Representations*, 2018.
  - [28] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?,” in *International Conference on Learning Representations*, 2019.
  - [29] Q. Li, Z. Han, and X. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning,” in *AAAI Conference on Artificial Intelligence*, pp. 3538–3545, 2018.
  - [30] F. Monti, D. Boscaini, J. Masci, E. R. J. Svoboda, and M. Bronstein, “Geometric deep learning on graphs and manifolds using mixture model CNNs,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5425–5434, 2017.

- [31] R. Liao, Z. Zhao, R. Urtasun, and R. Zemel, “LanczosNet: Multi-scale deep graph convolutional networks,” in *International Conference on Learning Representations*, 2019.
- [32] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, “Deep graph infomax,” in *International Conference on Learning Representations*, 2019.
- [33] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng, “Graph wavelet neural network,” in *International Conference on Learning Representations*, 2019.
- [34] F. Wu, T. Zhang, A. de Souza Jr., C. Fifty, T. Yu, and K. Weinberger, “Simplifying graph convolutional networks,” in *Proc. International Conference on Machine Learning*, 2019.
- [35] K. Xu, C. Li, Y. Tian, T. Sonobe, K. ichi Kawarabayashi, and S. Jegelka, “Representation learning on graphs with jumping knowledge networks,” in *Proc. International Conference on Machine Learning*, 2018.
- [36] L. Zhao and L. Akoglu, “PairNorm: Tackling oversmoothing in GNNs,” in *International Conference on Learning Representations*, 2020.
- [37] J. Weickert, *Anisotropic Diffusion in Image Processing*. ECMI Series, Teubner-Verlag, 1998.
- [38] M. Black, G. Sapiro, D. Marimont, and D. Heeger, “Robust anisotropic diffusion,” *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 421–432, 1998.
- [39] Y. Zhang and A. B. Hamza, “Vertex-based diffusion for 3-D mesh denoising,” *IEEE Transactions on Image Processing*, vol. 16, no. 4, pp. 1036–1045, 2007.
- [40] 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, pp. 2399–2434, 2006.
- [41] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, “Deep learning via semi-supervised embedding,” in *Neural Networks: Tricks of the Trade*, pp. 639–655, 2012.
- [42] X. Wu, Z. Li, A. So, J. Wright, and S. f. Chang, “Learning with partially absorbing random walks,” in *Advances in Neural Information Processing*, pp. 3077–3085, 2012.
- [43] 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. 93–106, 2008.
- [44] Z. Yang, W. Cohen, and R. Salakhudinov, “Revisiting semi-supervised learning with graph embeddings,” in *Proc. International Conference on Machine Learning*, pp. 40–48, 2016.
- [45] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations*, 2015.
- [46] L. van der Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of Machine Learning Research*, pp. 2579–2605, 2008.