Graph Convolutional Networks with Two-Stage Local Feature Aggregation

Haimin Zhang, Min Xu*, Guoqiang Zhang

School of Electrical and Data Engineering, Faculty of Engineering and IT, University of Technology Sydney

15 Broadway, Ultimo, NSW 2007, Australia

Abstract

Graph convolutional networks have achieved significant success in various non-Euclidean tasks. A key step in most current graph convolutional networks is aggregating features from a node's local neighborhood. However, in the aggregation step, the interrelationship information among the local neighbour nodes is not considered. In this paper, we propose a graph convolution method that learns a node's representation through two-stage aggregation. In the first stage, we follow the conventional approach to learn a node's representation by aggregating features from its local neighborhood. In the second stage, we learn an embedding between the first-stage aggregated feature and each of the node's neighbour node feature and then aggregate these learned embeddings. Finally, the aggregated features from the two stages are fused together as the node's representation. The aggregated feature in the first stage encodes the information of the local neighbourhood. Therefore, in our second stage, the information between a neighbour node and the rest of the neighbour nodes in the local neighbourhood is learned. Extensive experiments are conducted on four different tasks, including superpixel graph classification, node classification, graph regression and edge prediction, on six popular benchmark datasets. The experimental results show that our graph convolutional network achieves improved performance compared to the state-of-the-art models.

Keywords: Graph convolutional networks, graph representation learning, two-stage

*Corresponding author

Email addresses: Haimin.Zhang@uts.edu.au (Haimin Zhang), Min.Xu@uts.edu.au (Min Xu), Guoqiang.Zhang@uts.edu.au (Guoqiang Zhang)

feature aggregation.

1. Introduction

Graph-structured data are commonly seen in the real world. Online social networks, citation networks, knowledge graphs and biological networks—all of these types of data can be represented using graphs. It is important and of considerable significance to develop models that is able to learn and generalize from graph-structured data. The past years have seen increasing studies on graph representation learning, including generalizations of convolutional neural networks to non-Euclidean data, methods for deep graph embedding, and message propagation approaches [1]. These advances have led to new state of the art results in a variety of domain tasks, including recommender systems [2, 3], drug discovery [4], visual question answering [5] and time series analysis [6].

A graph is basically a collection of nodes, *e.g.*, entities, along with a set of edges between pairs of these nodes [1]. Let us use an online social network as an example, we could use nodes to represent individuals and use edges to represent a connection between two neighbouring individuals. Unlike images and natural languages, the underlying data of which can be represented in a grid-like structure, graph-structured data have an underlying structure that is in a non-Euclidean space. It is more difficult to deal with arbitrarily structured graphs. There have been many studies on generalizing neural networks to graph-structured data. Early attempts utilized recursive neural networks to process data represented as directed acyclic graphs. Later, graph neural networks that can be applied to a general class of graphs were proposed [7, 8]. These models involve a recurrent process that iteratively propagates node features until these node features reach an equilibrium or stable state.

Motivated by the considerable success of convolutional neural networks for different computer vision tasks, recent years have seen increasing efforts to generalize convolutions to the graph domain. Bruna *et al*., [9] first derived the fundamental graph convolutional network model as a generalization of convolutions to non-Euclidean data. Later, various graph convolutional network models were developed. Compared with

recurrent-based models, graph convolutional networks are much convenient and efficient for learning on graph-structured data. Today, graph convolutional networks have become the dominant approach for a wide variety of graph-based tasks. Most existing graph convolutional networks can be categorized into the spectral-based approach and the spatial-based approach. The spectral-based approach, which generalizes the notion of signal and convolution to the graph domain, constructs graph convolutions via an extension of the Fourier transform [9]. Unlike the spectral-based approach, the spatial-based approach directly defines convolutions on graph nodes and perform feature aggregation from spatially localized neighbours [10, 11].

In both spectral and spatial graph convolutional networks, the graph convolution basically learns a node's representation by aggregating features from the node's local neighborhood. For example, GCN models [12] apply the average pooling function to aggregate features from a node's local neighbourhood. GraphSAGE models [10] learn a function that generates node embeddings by sampling and aggregating features from a node's local neighborhood. In graph attention network (GAT) models [11], a self-attention function is applied to compute an attention weight for each neighbour node in aggregation. In GatedGCN models [13], the edge gating mechanism is utilized, enabling discriminative information to be propagated through edges in feature aggregation.

In all of these models, we observe that the aggregation function is applied once for a node in a graph convolutional layer. The aggregated feature encodes the information of the node's local neighbours. However, the interrelationship information among the neighbour nodes is not considered. For example, for aggregation with the average pooling function, the neighbourhood features are aggregated together independently. The information between a neighbour node feature and the rest of the neighbour node features is not encoded in the aggregated feature. In this paper, we propose a two-stage aggregation framework that enables the model to encode the interrelationship information among local neighbour nodes to improve graph representation learning. An illustration of our framework is demonstrated in Figure 1. In the first stage, we follow the conventional approach to learn a node's representation by aggregating features

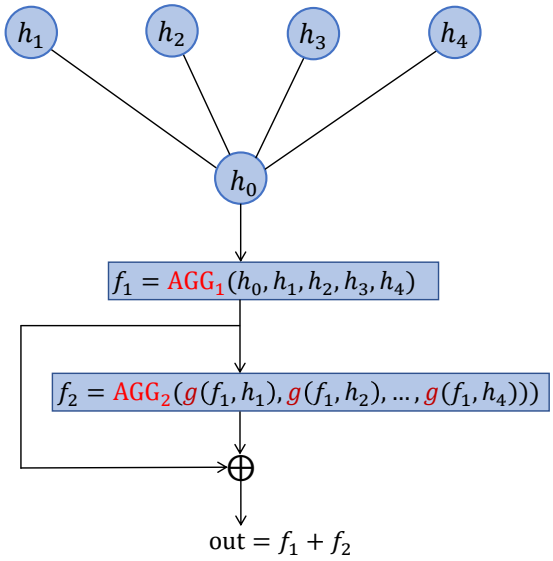


Figure 1: An illustration of our two-stage local feature aggregation framework for graph representation learning. g is a function that learns an embedding between the first-stage aggregated feature and each of the neighbour node feature.

from the node's local neighborhood. In the second stage, we learn an embedding between the first-stage aggregated feature and each of the node's neighbour node feature and then aggregate these learned embeddings. For a node, the first-stage aggregated feature encodes the information of the node's the local neighbourhood. Therefore, our second-stage aggregation enables the model to encode the interrelationship information among the local neighbour nodes. Finally, the aggregated features from the two stages are fused together as the node's representation.

We evaluate our graph convolutional network on four graph-based tasks, including superpixel graph classification, node classification, edge prediction and graph regression. The experiments are conducted on six benchmark datasets, including MNIST, CIFAR10, PATTERN, CLUSTER, TSP and ZINC [14]. We show that applying our second-stage aggregation consistently improves the performance on the six benchmark datasets. This demonstrates that our framework is a general framework for graph representation learning. To the best of our knowledge, our model achieves the state of the art performance compared to previous models.

The main contributions of this paper can be summarized as follows:

- We propose a framework for graph representation learning that generates node embeddings through two-stage local feature aggregation. Unlike previous message passing models, we introduce an additional aggregation stage, in which we learn an embedding between the first-stage aggregated feature and each of the node's neighbour node feature and then aggregate these learned embeddings. This enables our model to encode the interrelationship information of the local neighbour nodes and therefore helps to improve the graph representation learning performance.

- We conduct extensive experiments on six benchmark datasets to validate our graph convolutional network. We demonstrate that our framework is a general framework that consistently improves the graph representation learning performance for different graph domain tasks, advancing the state of the art results on the benchmark datasets.

2. Related Work

Generalizing neural networks to graph-structured data has been studied for decades. The past years have seen a surge in research on graph convolutional networks as generalizations convolutional neural networks to graph-structured data. Recent graph convolutional networks can be categorized into the spectral-based approach and the spatial-based approach [15]. The spectral-based approach is motivated based on the spectral graph theory. This approach constructs graph convolutions via an extension of the Fourier transform. Bruna *et al.* [9] proposed the first spectral graph neural network, in which the convolution operation is applied in the Fourier domain based on the eigen-decomposition of the graph Laplacian. Defferrard *et al.* improved the issue of non-spatially localized kernels in Bruna's [9] work by introducing Chebyshev graph networks (ChebNets). ChebNets utilize Chebyshev expansion of the graph Laplacian to approximate the kernels, resulting in spatially localized kernels. Finally, Kipf *et al.* [12] introduced the popular GCN model that uses a propagation rule based on the first-order approximation of spectral convolutions. In spectral graph convolutional

networks, the learned kernels are dependent on the graph structure, therefore a graph convolutional network model trained on a specific graph cannot be generalized to other graph structures.

Unlike the spectral-based approach, the spatial-based approach directly defines convolution on the graph nodes in a local neighbourhood. This approach is much efficient, and the trained models can be generalized to arbitrarily structured graphs. A major challenge in the spatial-based approach is to deal with variable-sized neighbours. To address this challenge, several sampling-based methods, including nodewise sampling-based [10], layerwise sampling-based [16] and adaptive layerwise sampling-based, have been proposed. Hamilton *et al.* [10] introduced the GraphSAGE framework that generates node embeddings by sampling a fixed-size set of neighbors and aggregating local neighbourhood features with an aggregation function. Velickovic *et al.* [11] proposed to integrate the self-attention method into graph convolutional networks, enabling the models to focus on important nodes in feature aggregation. Zhang *et al.* [17] proposed gated attention networks, in which the self-attention method is applied to the outputs of multiple attention heads to improve the overall performance. Bresson *et al.* [13] proposed residual gated graph convnets (GatedGCN), integrating edge gates, residual learning [18] and batch normalization [19] into graph convolutional networks. Sun *et al.* [20] proposed an RNN-like graph network model referred to as AdaGCN, incorporating AdaBoost into the graph network model. The AdaGCN model extracts information from high-order neighbours, integrating information from different hops of neighbours into the model in an Adaboost way.

Motivated by success of deep convolutional neural networks, more recent years have seen increasing efforts for developing deep graph convolutional networks. Klicpera *et al*. [21] proposed a personalized pagerank-based propagation rule for graph convolutional networks, which ensures the locality using teleports to enable stacking deeper graph convolutional layers. Zhao *et al.* [22] proposed a normalization method referred to as PairNorm that ensures the total pairwise feature distance across layers to be constant to tackle the oversmoothing issue in deep graph convolutional networks. Rong *et al.* [23] introduced a variant of the Dropout method referred to as DropEdge

for regularizing graph convolutional networks. The DropEdge method stochastically drops out a set of edges from the original graph in each training epoch, working like a message passing reducer and a data augmentor. Xu *et al.* [24] introduced the JKNet model, in which dense skip connections are used to combine the output of each layer to preserve the locality of node representations to reducing the oversmoothing issue. Feng *et al.* [25] proposed graph random networks, wherein a random propagation strategy is used to perform high-order feature propagation to reduce oversmoothing.

3. Methodology

This section describes our graph convolutional network that learns through two-stage local feature aggregation. We first introduce the notations of graphs, and then we describe the details of our two-stage aggregation framework for embedding generation. Finally, we describe applying our graph convolutional network on four downstream tasks.

3.1. Notations

Formally, a graph is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes (or vertices) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges between these nodes. $N = |\mathcal{V}|$ is the number of nodes in $\mathcal{G}$. An edge going from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ is denoted as $(u, v) \in \mathcal{E}$. Conveniently, a graph can be represented through an adjacent matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, in which $\mathbf{A}_{u,v} = 1$ if $(u, v) \in \mathcal{E}$ and $\mathbf{A}_{u,v} = 0$ otherwise. The degree matrix of $\mathcal{G}$, which is a diagonal matrix, is denoted as $\mathbf{D} \in \mathbb{R}^{N \times N}$, in which $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$ is the set of features or attributes associated with the nodes.

The graph Laplacian is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, and the symmetric normalized Laplacian is defined as $\mathbf{L}^{sym} = \mathbf{I}_N - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where I_N is an identity matrix. The symmetric normalized Laplacian $\mathbf{L}^{sym}$ is positive semidefinite and can be factorized as $\mathbf{L}^{sym} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, where $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_N] \in \mathbb{R}^{N \times N}$ is a matrix of eigenvectors and $\mathbf{\Lambda} = \text{diag}(\lambda_1, ..., \lambda_N)$ is a diagonal matrix of its eigenvalues.

3.2. Two-Stage Local Feature Aggregation

Graph convolutional networks are a general approach for learning on graph-structured data. A key property of graph convolutional networks is that they are permutation invariant. That is the models are not dependent on the arbitrary ordering of the rows or columns in the adjacency matrix. The basic graph convolutional network model can be motivated in a variety of ways. From one perspective, the basic graph convolutional network model is developed based on the spectral graph theory, as a generalization of Euclidean convolutions to the non-Euclidean graph domain [9]. The graph convolution is defined as the multiplication of signal $\mathbf{s} \in \mathbb{R}^N$ with a filter g_θ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^N$ in the Fourier domain:

$$g_\theta * \mathbf{s} = \mathbf{U} g_\theta^*(\mathbf{\Lambda}) \mathbf{U}^T \mathbf{s}, \tag{1}$$

where $*$ denotes the convolution operation. g_θ can be understood as a function of the eigenvalues, *i.e.,* $g_\theta^*(\mathbf{\Lambda})$.

Meanwhile, the graph convolution can be defined on graph nodes, performing aggregation from spatially closed neighbourhood. This behaviour is analogous to that of the convolutional kernels in convolutional neural networks, which aggregate features from spatially-defined patches in an image. Both spectral and spatial graph convolutional networks have been demonstrated to be message passing neural networks [26] that use a form of neural message passing in which messages are exchanged between nodes and updated using a neural network [27]. During each message-passing iteration, a hidden representation $\mathbf{h}_v^{(l)}$ corresponding to node v is updated according to information aggregated from v's local neighbourhood. This message passing rule can be expressed as follows:

$$\mathbf{h}_v^{(l+1)} = f(\mathbf{h}_v^{(l)}, \text{AGGRGATE}(\mathbf{h}_u^{(l)}, \forall u \in \mathcal{N}(v))), \tag{2}$$

where f and AGGRGATE are a differentiable function, and $\mathcal{N}(v))$ is the set of v's local neighbours.

The intuition behind the message-passing framework is that, at each iteration, each

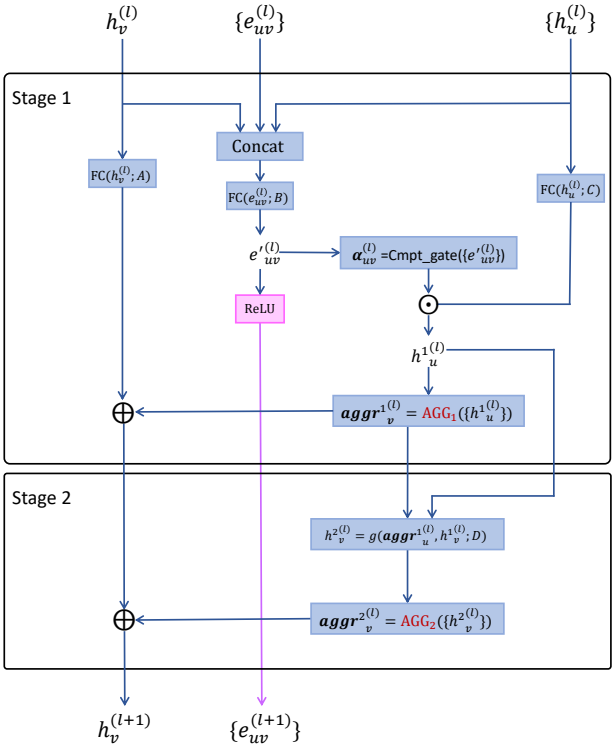


Figure 2: Our graph convolutional layer that learns through two-stage local feature aggregation. In the first stage, we use the edge gate method to aggregate local neighbourhood features. In the second stage, we learn an embedding between the first-stage aggregated feature and each of the feature propagated through the edge gate and then aggregate these learned embeddings. The aggregated features from the two stages are fused together as the node's representation.

node aggregates feature information from its local neighborhood, and as these iterations progress each node representation encodes information from further reaches of the graph. Since the AGGREGATE function takes a set of features as input, graph convolutional networks defined in this way are permutation equivariant.

In the basic message passing framework, the aggregation operation is applied once for each node in a graph convolutional layer. The aggregated feature for a node encodes the feature information of the node's the local neighbourhood. However, the interrelationship information among the local neighbour nodes is not encoded. That is, the features of the local neighbour nodes, which may be multiplied by an attention weight or an edge gate vector, are aggregated together in an independent manner. The model

performance could be restricted without integrating the interrelationship information. As far as we know, existing studies have not considered utilizing the interrelationship information among local neighbour nodes for graph representation learning.

We propose a two-stage local feature aggregation framework that enables the model to encode the interrelationship information among local neighbour nodes. In the first stage, we follow the conventional message-passing approach to learn a node's representation by aggregating feature information from the node's local neighborhood. Then in the second stage, we learn an embedding between the first-stage aggregated feature and each of the node's neighbour node feature and then aggregate these learned embeddings. Finally, the aggregated features from the two stages are fused together as the node's representation. Because the aggregated feature in the first stage encodes the information of the node's the local neighbourhood, the learned embedding regarding a neighbour node encodes the information between the neighbour node and the rest neighbour nodes. Therefore, our second-stage aggregated feature encodes the interrelationship information among the local neighbour nodes.

3.3. Our Graph Convolutional Network Model

Figure 2 shows our graph convolutional layer that learns through two-stage aggregation. We follow the edge gate approach [13] to aggregate local neighbourhood features in our first stage. Note that our framework is a general framework, any aggregation methods in existing graph convolutional networks can be applied in our first stage. We explicitly associate an edge feature for each edge and learn an edge gate leveraging the edge feature for feature propagation. The input to our graph convolutional layer is a set of node features and a set of edge features, and our graph convolutional layer outputs a new set of node features and a new set of edge features. For edge $(u \in \mathcal{N}(v), v)$ going from node u to node v, we concatenate three features in relation to the edge: the edge's start node feature, the edge's input feature and the edge's end node feature. The concatenated feature is fed to a fully connected layer with the ReLU

activation function:

$$\mathbf{e'}_{uv}^{(l)} = \text{ReLU}(\mathbf{B}^{(l)} \cdot \text{CONCAT}(\mathbf{h}_u^{(l)}, \mathbf{e}_{uv}^{(l)}, \mathbf{h}_v^{(l)})), \tag{3}$$

where $\mathbf{B}^{(l)} \in \mathbb{R}^{d^{(l)} \times 3d^{(l)}}$ is the trainable parameter matrix (the bias term is omitted for succinctness) and CONCAT is the concatenation operation, and $d^{(l)}$ is the dimension of $\mathbf{e}_{vu}^{(l)}$, $\mathbf{h}_u^{(l)}$ and $\mathbf{h}_v^{(l)}$.

We describe our two-stage local feature aggregation method as follows. In the first stage, we first compute a gate vector for edge (u, v) as follows:

$$\boldsymbol{\alpha}_{uv}^{(l)} = \frac{\text{Sigmoid}(\mathbf{e'}_{uv}^{(l)})}{\sum_{u \in \mathcal{N}(v)} \text{Sigmoid}(\mathbf{e'}_{uv}^{(l)}) + \epsilon}, \tag{4}$$

where ϵ is a small constant number. The feature propagated from node u to node v through the edge gate can be described as follows:

$$\mathbf{h}^1{}_u^{(l)} = \boldsymbol{\alpha}_{uv}^{(l)} \odot (\mathbf{C}^{(l)} \cdot \mathbf{h}_u^{(l)}), \tag{5}$$

where $\odot$ denotes the Hadamard product operator, and $\mathbf{C}^{(l)}$ is a linear transformation matrix. Then we aggregate the propagated features from v's local neighbourhood as follows:

$$\begin{aligned} \mathbf{aggr}^1{}_v^{(l)} &= \sum_{u \in \mathcal{N}(v)} \mathbf{h}^1{}_u^{(l)} \\ &= \sum_{u \in \mathcal{N}(v)} \boldsymbol{\alpha}_{uv}^{(l)} \odot (\mathbf{C}^{(l)} \cdot \mathbf{h}_u^{(l)}). \end{aligned} \tag{6}$$

Unlike in GAT models, in which an attention weight is computed for each node feature in feature aggregation, the edge gate approach learns a gate vector in $\mathbb{R}^{d^{(l)}}$ space for each node feature, allowing discriminative information to be propagated through the edge gate.

In our second stage, we concatenate $\mathbf{aggr}^1{}_v^{(l)}$ with each of the feature propagated through the edge gate, *i.e.,* $\mathbf{h}^1{}_u^{(l)}, u \in \mathcal{N}(v)$, and learn an embedding with a fully

connected layer from the concatenated feature.

$$\mathbf{h}^{2}{}_{u}^{(l)} = \mathbf{D}^{(l)} \cdot \text{CONCAT}(\mathbf{aggr}^{1}{}_{v}^{(l)}, \mathbf{h}^{1}{}_{u}^{(l)}), \tag{7}$$

where $\mathbf{D}^{(l)} \in \mathbb{R}^{d^{(l)} \times 2d^{(l)}}$ is the weight matrix of the fully connected layer. We then aggregate these learned embeddings as follows:

$$\begin{aligned} \mathbf{aggr}^{2}{}_{v}^{(l)} &= \sum_{u \in \mathcal{N}(v)} \mathbf{h}^{2}{}_{u}^{(l)} \\ &= \sum_{u \in \mathcal{N}(v)} \mathbf{D}^{(l)} \cdot \text{CONCAT}(\mathbf{aggr}^{1}{}_{v}^{(l)}, \mathbf{h}^{1}{}_{u}^{(l)}), \end{aligned} \tag{8}$$

Because $\mathbf{aggr}^{1}{}_{v}^{(l)}$ encodes the feature information of node v's local neighbour nodes, the learned embedding $\mathbf{h}^{2}{}_{u}^{(l)}$ encodes the information between node u and and the rest of node v's neighbour nodes, *i.e.,* $\mathcal{N}(v) \backslash u$. Therefore, the second-stage aggregated feature $\mathbf{aggr}^{2}{}_{v}^{(l)}$ encodes the interrelationship information among v's local neighbour nodes

After obtaining the aggregated features from the two stages, we update the node feature and edge feature as follows:

$$\begin{aligned} \mathbf{h}_{v}^{(l+1)} &= \text{ReLU}(\mathbf{A}^{(l)}\mathbf{h}_{v}^{(l)} + \mathbf{aggr}^{1}{}_{v}^{(l)} + \mathbf{aggr}^{2}{}_{v}^{(l)}) + \mathbf{h}_{v}^{(l)} \\ \mathbf{e}_{uv}^{(l+1)} &= \text{ReLU}(\mathbf{e}'^{(l)}_{uv}) + \mathbf{e}_{uv}^{(l)}, \end{aligned} \tag{9}$$

where $\mathbf{A}^{(l)} \in \mathbb{R}^{d^{(l)} \times d^{(l)}}$ is a trainable transformation matrix applied to $\mathbf{h}_u$. In the above equation, the residual learning method [18] is also used for updating the node feature and edge feature. The residual learning method introduces an additional identity mapping between the input and the output. This can help to reduce the over-smoothing issue in graph convolutional networks, while also improving the stability in optimization.

Algorithm 1 describes the process for node embedding generation in which the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the set of node features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$ are given as input. The edge feature $\mathbf{e}_{u,v}^{(0)}, \forall (u, v) \in \mathcal{E}$ is initialized by a learnable vector by default. Each step in the outer loop of Algorithm 1 proceeds as follows, where l denotes the current graph

Algorithm 1: The two-stage local feature aggregation algorithm for embedding generation.

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; input node features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; number of graph convolutional layers L; input edge features $\{\bar{\mathbf{e}}_{uv}, \forall (u,v) \in \mathcal{E}\}$.

Output: Vector representations $\{\mathbf{h}_v^{(L)}, \forall v \in \mathcal{V}\}$.

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$

$\mathbf{e}_{uv}^{(0)} \leftarrow \bar{\mathbf{e}}_{uv}, \forall (u,v) \in \mathcal{E}$

for $l = 0, ..., L-1$ **do**

  for $v \in \mathcal{V}$ **do**

    /* first stage aggregation. */

    $\mathbf{e}'^{(l)}_{uv} \leftarrow \text{ReLU}(\mathbf{B}^{(l)} \cdot \text{CONCAT}(\mathbf{h}_u^{(l)}, \mathbf{e}_{uv}^{(l)}, \mathbf{h}_v^{(l)})),\ u \in \mathcal{N}(v)$

    $\boldsymbol{\alpha}_{uv}^{(l)} \leftarrow \frac{\text{Sigmoid}(\mathbf{e}'^{(l)}_{uv})}{\sum_{u \in \mathcal{N}(v)} \text{Sigmoid}(\mathbf{e}'^{(l)}_{uv}) + \epsilon}$

    $\mathbf{aggr}^{1}{}_v^{(l)} \leftarrow \sum_{u \in \mathcal{N}(v)} \boldsymbol{\alpha}_{uv}^{(l)} \odot (\mathbf{C}^{(l)} \cdot \mathbf{h}_u^{(l)})$

    /* second stage aggregation. */

    $\mathbf{aggr}^{2}{}_v^{(l)} \leftarrow \sum_{u \in \mathcal{N}(v)} \mathbf{D}^{(l)} \cdot \text{CONCAT}(\mathbf{aggr}^{1}{}_v^{(l)}, \mathbf{h}^{1}{}_u^{(l)})$

    /* update node features and edge features.*/

    $\mathbf{h}_v^{(l+1)} \leftarrow \text{ReLU}(\mathbf{A}^{(l)}\mathbf{h}_u^{(l)} + \mathbf{aggr}^{1}{}_u^{(l)} + \mathbf{aggr}^{2}{}_u^{(l)}) + \mathbf{h}_u^{(l)}$

    $\mathbf{e}_{uv}^{(l+1)} \leftarrow \text{ReLU}(\mathbf{e}'^{(l)}_{uv}) + \mathbf{e}_{uv}^{(l)}$

  end

end

convolutional layer. First, a gate vector is learned for each edge, and each node $v \in \mathcal{V}$ aggregates the features propagated from its immediate neighbours $\{u, \forall u \in \mathcal{N}(v)\}$ into a single vector $\mathbf{aggr}^{1}{}_v^{(l)}$ using Equation (6). Then we learn an embedding between the aggregated feature $\mathbf{aggr1}_v^{(l)}$ and each of the features propagated through the edges and aggregate the learned embeddings into single vector $\mathbf{aggr}^{2}{}_v^{(l)}$. Finally, $\mathbf{h}_v^{(l+1)}$ and $\mathbf{e}_{uv}^{(l+1)}$ are updated using Equation (9).

For downstream tasks, we apply a task-based layer to the hidden node embeddings generated by the final graph convolutional layer to generate the task-dependent output. In this research, we empirically evaluate our graph convolutional network on four tasks: graph classification, node classification, edge prediction and graph regression. We briefly introduce the task-based layer for the four tasks as follows.

Graph Classification. For graph classification, the task-based layer aggregates the hidden node embeddings using average pooling, obtaining the following graph-level

feature:

$$\mathbf{g} = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \mathbf{h}_v^{(L)} \tag{10}$$

The obtained graph-level feature is then fed to a two-layer feed forward network (FFN) that outputs a vector of logits over all categories:

$$\mathbf{y} = \mathbf{Q}_1 \cdot \text{ReLU}(\mathbf{P}_1 \cdot \mathbf{g}), \tag{11}$$

where $\mathbf{P}_1 \in \mathbb{R}^{d^{(L)} \times d^{(L)}}$ and $\mathbf{Q}_1 \in \mathbb{R}^{c_1 \times d^{(L)}}$, c_1 is the number of graph categories. For training the model, the average cross-entropy loss is applied to minimize the error between the outputs and the targets on the training dataset.

Node Classification. For the node classification task, a shared two-layer FFN is applied to the hidden node embeddings, generating a vector of logits for each node:

$$\mathbf{y}_v = \mathbf{Q}_2 \cdot \text{ReLU}(\mathbf{P}_2 \cdot \mathbf{h}_v^{(L)}),\ v \in \mathcal{V}, \tag{12}$$

where $\mathbf{P}_2 \in \mathbb{R}^{d^{(L)} \times d^{(L)}}$ and $\mathbf{Q}_2 \in \mathbb{R}^{c_2 \times d^{(L)}}$, c_2 is the number of node categories. The model is trained by minimizing the the weighted cross-entropy loss between the outputs and the targets on the training dataset.

Edge Prediction. To make a prediction for the edge between node u and node v, we concatenate the embeddings of the two nodes and then feed the concatenated feature to a two-layer FFN that generates a vector of logits over the edge categories:

$$\mathbf{y}_{uv} = \mathbf{Q}_3 \cdot \text{ReLU}(\mathbf{P}_3 \cdot \text{CONCAT}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})), \tag{13}$$

where $\mathbf{P}_3 \in \mathbb{R}^{d^{(L)} \times d^{(L)}}$ and $\mathbf{Q}_3 \in \mathbb{R}^{c_3 \times d^{(L)}}$, c_3 is the number of edge categories. The standard cross entropy loss function is utilized to minimize the error between the predicted results and the targets for training the model.

Graph Regression. For graph regression, we also use Equation (10) to generate the

graph-level feature and then feed the obtained feature to a two-layer FFN that generates a score for the graph:

$$\mathbf{y} = \mathbf{Q}_4 \cdot \mathrm{ReLU}(\mathbf{P}_4 \cdot \mathbf{g}), \tag{14}$$

where $\mathbf{P}_4 \in \mathbb{R}^{d^{(L)} \times d^{(L)}}$ and $\boldsymbol{Q}_4 \in \mathbb{R}^{1 \times d^{(L)}}$. The model is trained by minimizing the average L1-loss between the outputs and the targets on the training dataset.

4. Experiments

We experimentally validate our two-stage aggregation graph convolutional network on four tasks to demonstrate that our framework is a general approach to improve the graph representation learning performance.

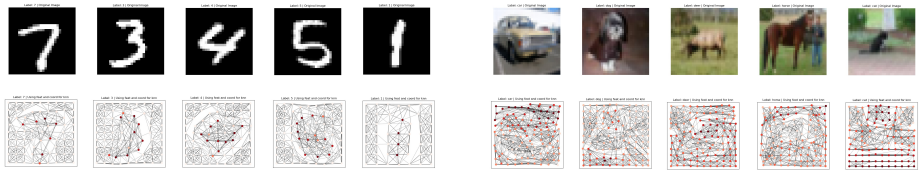
4.1. Experimental Setup

Datasets. Our experiments are conducted on six recently released benchmark datasets [14]: MNIST, CIFAR10, PATTERN, CLUSTER, TSP and ZINC.

- **MNIST** and **CIFAR-10**. These two datasets are used for evaluation on the superpixel graph classification task. The original images in MNIST [32] and CIFAR10 [33] are converted to superpixel graphs. The superpixels, which are extracted using the SLIC technique [34], represent small regions of homogeneous intensity in images. Figure 3 shows samples of the superpixel graphs in the two datasets.

- **PATTERN** and **CLUSTER**. These two datasets are used for evaluation on the node classification task. The graphs in the two datasets are generated using the stochastic block model [35], which is widely used for modeling communities in social networks. The complexity of the communites can be adjusted through modulating the intra- and extra-community connections.

- **TSP**. This dataset is constructed to evaluate graph convolutional networks on solving the travelling salesman problem (TSP), which is a classical NP-Hard combinatorial problem. We validate if the predicted edges of our model belonging to the optimal TSP solution obtained by the Concorde solver [36].



(a) MNIST (b) CIFAR-10

Figure 3: Examples of superpixel graphs (second row) and corresponding original images (first row) in MNIST and CIFAR10. The node features of the superpixel graphs consist of the superpixel's coordinates and intensity.

Table 1: Statistics of the six benchmark datasets for our experiments.

Task	Dataset	Graphs	Nodes	Nodes/graph	Training	Validation	Test	Categories
Graph classification	MNIST	70K	4,939,668	40-75	55,000	5000	10,000	10
	CIFAR10	60K	7,058,005	85-150	45,000	5000	10,000	10
Node classification	PATTERN	14K	1,664,491	44-188	10,000	2000	2000	2
	CLASTER	12K	1,406,436	41-190	10,000	1000	1000	6
Link prediction	TSP	12K	3,309,140	50-500	10,000	1000	1000	2
Graph regression	ZINC	12K	277,864	9-37	10,000	1000	1000	–

- **ZINC**. This dataset is a subset (12K) of the ZINC molecular graphs (250K) dataset [37]. We validate our model for regressing the molecular property (or called constrained solubility) on this dataset.

The statistics of the six datasets are reported in Table 1.

Evaluation metrics. Following Dwivedi *et al.* [14], the following metrics are used for performance evaluation on different tasks.

- **Accuracy**. For superpixel graph classification on MNIST and CIFAR10, classification accuracy is used for evaluating our model performance. For node classification on PATTERN and CLUSTER, weighted accuracy is used for evaluating our model performance.

- **F1 score**. For the edge prediction task on TSP, due to high class imbalance, *i.e.,* only the edges in the TSP tour are labeled as positive, the F1 score for the positive class is used for evaluating our model performance.

- **MAE (mean absolute error)**. The MAE is used for evaluating our model on graph regression on the ZINC dataset.

Table 2: Experimental results of our graph convolutional network on the graph superpixel classification task on MNIST and CIFAR10. The best result of the baseline models is shown in violet.

Model	MNIST			Model	CIFAR10		
	L	Parameters	Accuracy (↑)		L	Parameters	Accuracy (↑)
MLP	4	104K	95.340±0.138	MLP	4	104K	56.340±0.181
GCN [12]	4	101K	90.705±0.218	GCN [12]	4	102K	55.710±0.381
GraphSAGE [10]	4	104K	97.312±0.097	GraphSAGE [10]	4	105K	65.767±0.308
MoNet [28]	4	104K	90.805±0.032	MoNet [28]	4	104K	54.655±0.518
GAT [11]	4	110K	95.535±0.205	GAT [11]	4	111K	64.223±0.455
GatedGCN [13]	4	104K	97.340±0.143	GatedGCN [13]	4	104K	67.312±0.311
GIN [29]	4	105K	96.485±0.252	GIN [29]	4	106K	55.255±1.527
RingGNN [30]	2	105K	11.350±0.000	RingGNN [30]	2	105K	19.300±16.108
	2	505K	91.860±0.449		2	505K	39.165±17.114
	8	506K	Diverged		8	510K	Diverged
3WLGNN [31]	3	108K	95.075±0.961	3WLGNN [31]	3	109K	59.175±1.593
	3	502K	95.002±0.419		3	503K	58.043±2.512
	8	501K	Diverged		8	502K	Diverged
	4	144K	98.228±0.038		4	144K	75.001±0.257
Ours	8	284K	98.512±0.048	**Ours**	8	284K	76.395±0.101
	12	424K	**98.635±0.049**		12	424K	**76.515±0.081**

Baseline models. To demonstrate the superior performance of our graph convolutional network, we compare the performance our model against a wide variety of baseline models including MLP (multilayer perceptron), GCN [12], GraphSAGE [10], MoNet [28], GAT [11], GatedGCN [13], GIN [29], RingGNN [30] and 3WLGNN [31].

Implementation details. We closely follow the implementation details in [14]. The Adam algorithm [38] is used for training our model. The learning rate is initialized to 10^{-3} and reduced by a factor of 2 when the validation loss has not been reduced for a number of epochs (10, 20 or 30). We apply the SSFG regularization method to the hidden node and edge features outputted in each graph convolutional layer. The training procedure is terminated when the learning rate is reduced to smaller than 10^{-6}. Our model is implemented in Pytorch with the DGL library [39]. The experiments are conducted on a GPU with 24 GB memory. We validate our model with different graph convolutional layers, *e.g.,* 4, 8, 12, 16. For each experiment, we validate our model for 4 times using different random seeds and report the mean and standard deviation of the 4 runs.

4.2. Experimental Results

Graph classification. We validate our model with three different graph convolutional layers, *i.e.*, $L \in \{4, 8, 12\}$. The quantitative results on superpixel graph classi-

Table 3: Experimental results of our graph convolutional network on the node classification task on PATTERN and CLUSTER. The best result of the baseline models is shown in violet.

Model	PATTERN			Model	CLUSTER		
	L	Parameters	Accuracy (↑)		L	Parameters	Accuracy (↑)
MLP	4	105K	50.519±0.000	MLP	4	106K	20.973±0.004
GCN [12]	4	101K	63.880±0.074	GCN [12]	4	102K	53.445±2.029
	16	501K	71.892±0.334		16	502K	68.498±0.976
GraphSAGE [10]	4	102K	50.516±0.001	GraphSAGE [10]	4	102K	50.454±0.145
	16	503K	50.492±0.001		16	503K	63.844±0.110
MoNet [28]	4	104K	85.482±0.037	MoNet [28]	4	104K	58.064±0.131
	16	511K	85.582±0.038		16	512K	66.407±0.540
GAT [11]	4	110K	75.824±1.823	GAT [11]	4	111K	57.732±0.323
	16	527K	78.271±0.186		16	528K	70.587±0.447
GatedGCN [13]	4	104K	84.480±0.122	GatedGCN [13]	4	104K	60.404±0.419
	16	502K	85.568±0.088		16	503K	73.840±0.326
GIN [29]	4	101K	85.590±0.011	GIN [29]	4	104K	58.384±0.236
	16	509K	85.387±0.136		16	518K	64.716±1.553
RingGNN [30]	2	105K	86.245±0.013	RingGNN [30]	2	105K	42.418±20.063
	2	505K	86.244±0.025		2	524K	22.340±0.000
	8	506K	Diverged		8	514K	Diverged
3WLGNN [31]	3	104K	85.661±0.353	3WLGNN [31]	3	106K	57.130±6.539
	3	503K	85.341±0.207		3	507K	55.489±7.863
	8	582K	Diverged		8	587K	Diverged
Ours	4	143K	85.669±0.034	**Ours**	4	144K	63.212±0.130
	8	283K	86.336±0.136		8	284K	73.055±0.135
	12	423K	86.601±0.084		12	424K	75.458±0.108
	16	563K	**86.643±0.047**		16	564K	**76.163±0.101**

fication is shown in Table 2, comparing our graph network against the baseline models. We see that our model with 4 graph convolutional layers obtains 98.228% and 75.001% accuracy on MNIST and CIFAR10, respectively. Further increasing the number of graph convolutional layers to 8 and 12 results in additional performance gains. GatedGCN achieves the best performance on the two datasets among the baseline models. When compared to GatedGCN, our model with 12 graph convolutional layers improves the performance by 1.295% and 9.203% on the two datasets respectively. To the best of our knowledge, our graph convolutional network achieves the state of the art performance on the two datasets.

Node classification. On the node classification task, we validate our model with four different graph convolutional layers, *i.e.*, $L = 4, 8, 12, 16$. Table 3 compares the results of our graph convolutional network with the baseline models. Our models with 16 graph convolutional layers achieves 86.643% and 76.163% weighted accuracy on PATTERN and CLUSTER, respectively. We see that our model outperforms all the baseline models. RingGNN [30] achieves the best performance among the base-

Table 4: Experimental results on the link prediction task on TSP. The best result of the baseline models is shown in violet. OOM denotes "out of memory".

Model	L	TSP	
		Parameters	F1 (↓)
MLP	4	97K	0.544±0.0001
GCN [12]	4	96K	0.630±0.0001
GraphSAGE [10]	4	90K	0.665±0.003
MoNet [28]	4	99K	0.641±0.002
GAT [11]	4	96K	0.671±0.002
GatedGCN [13]	4	98K	0.791±0.003
GIN [29]	4	99K	0.656±0.003
RingGNN [30]	2	107K	0.643±0.024
	2	508K	0.704±0.003
	8	506K	Diverged
3WLGNN [31]	3	106K	0.694±0.073
	3	507K	0.288±0.311
	8	509K	OOM
Ours	4	132K	0.811±0.001
	8	253K	0.832±0.001
	12	373K	0.841±0.001
	16	495K	**0.844±0.001**

line models on PATTERN. When compared to RingGNN, our model achieves 0.398% improved performance on PATTERN. GatedGCN [13] achieves the best performance among the baseline models on the CLUSTER dataset. Compared with GatedGCN, our models with $L = 16$ achieves a 2.323% performance improvement.

Link prediction. The experimental results for link prediction on TSP is reported in Table 4. We see that our graph convolutional network outperforms the baseline models by a large margin, demonstrating the effectiveness of our two-stage local feature aggregation framework for improving graph representation learning for link prediction. GatedGCN achieves an F1 score of 0.791, which is the best result among the baseline models. Our model with $L = 16$ outperforms GatedGCN by 0.053. Once again, our model achieves the state of the art performance.

Graph regression. Table 5 reports our results on graph regression on the ZINC dataset. Note that we only use node features (types of heavy atoms) for training our model while ignoring the bond types between atoms. Compared with GatedGCN-E

Table 5: Experimental results on the graph regression task on ZINC. The best result of the baseline models is shown in violet. Note that we only use node features to train our model.

Model	L	ZINC	
		Parameters	MAE (↓)
MLP	4	109K	0.706±0.006
GCN [12]	4	103K	0.459±0.006
	16	505K	0.367±0.011
GraphSAGE [10]	4	95K	0.468±0.003
	16	505K	0.398±0.002
MoNet [28]	4	106K	0.397±0.010
	16	504K	0.292±0.006
GAT [11]	4	102K	0.475±0.007
	16	531K	0.384±0.007
GatedGCN [13]	4	106K	0.435±0.011
GatedGCN-E [13]	4	106K	0.375±0.003
GIN [29]	4	103K	0.387±0.015
	16	510K	0.526±0.051
RingGNN [30]	2	98K	0.512±0.023
3WLGNN [31]	3	102K	0.407±0.028
Ours	4	146K	0.268±0.005
	8	285K	0.233±0.004
	12	425K	**0.226±0.002**

[13] , which uses bond types to initialize edge features, our model with $L = 12$ results in 0.149 reduced MAE. MoNet [28] achieves the best performance among the baseline models. Compared with MoNet [28], our model with $L = 12$ reduces the MAE from 0.292 to 0.226. The results show that the proposed method is effective to improve graph representation learning for graph regression.

Tackling the over-smoothing issue. Over-smoothing is a common issue with graph convolutional networks. The over-smoothing issue comes with repeatedly applying graph convolutions, resulting in node features converging to similar values. Because of the over-smoothing issue, most graph convolutional networks are restricted to shallow layers, *e.g.,* 2 to 4. Further increasing the number of graph convolutional layers will lead to reduced performance.

Figure 4 shows the training and test accuracies/F1 scores/MAEs of our model with different graph convolutional layers on the benchmark datasets. We see that our train-

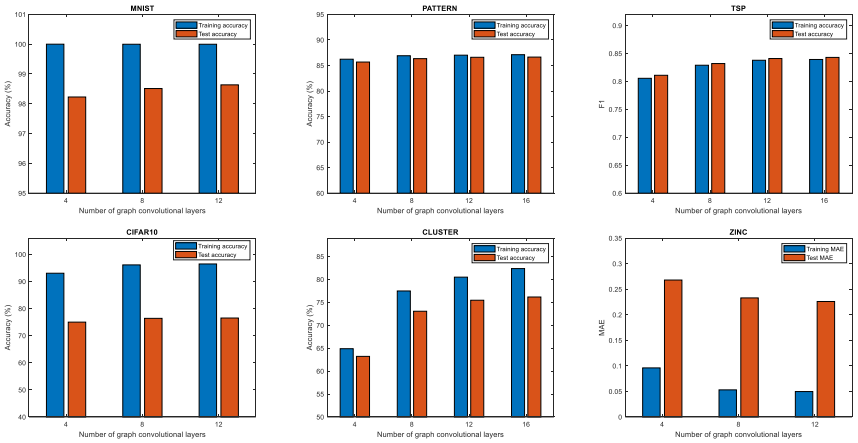


Figure 4: Training and test accuracies/F1 scores/MAEs of our model with different graph convolutional layers. We show that our training and test performances improve consistently as the number of graph convolutional layers increases, which suggests that our graph convolutional network is effective to reduce the oversmoothing issue.

Table 6: Ablation study: effect of each feature component for learning a node's representation (see Eq. (9)) on the overall performance.

Method	Accuracy (↑) MNIST (L=12)	CIFAR10 (L=12)	PATTERN (L=16)	CLUSTER (L=16)	F1 (↑) TSP (L=16)	MAE (↓) ZINK (L=12)
$\mathbf{Ah} + \mathbf{aggr}^1$	98.140	72.032	85.723	75.245	0.818	0.281
$\mathbf{Ah} + \mathbf{aggr}^2$	98.490	76.383	86.589	76.158	0.840	0.227
$\mathbf{aggr}^1 + \mathbf{aggr}^2$	98.573	76.497	86.620	76.056	0.843	0.226
$\mathbf{Ah} + \mathbf{aggr}^1 + \mathbf{aggr}^2$	98.635	76.515	86.643	76.163	0.844	0.226

ing and test performances improve consistently as the number of graph convolutional layers increases. This suggests that our two-stage local feature aggregation method helps to address the over-smoothing issue.

Ablation study. In our graph convolutional layer, three features, *i.e.,* $\mathbf{Ah}_v$, $\mathbf{aggr}_v^1$ and $\mathbf{aggr}_i^2$, are fused together to update the representation of node $i \in \mathcal{V}$ (see Equation (9)). We conduct an ablation study to show the importance of each feature contributing to the overall performance. Note when only $\mathbf{Ah}_v$ and $\mathbf{aggr}_v^1$ are used, our graph convolutional network is equivalent to the GatedGCN model. As shown in Table 6, we see that integrating the second-stage aggregated feature consistently improves the graph representation learning performance on the benchmark datasets. We also see that the use of $\mathbf{aggr}_v^2$ performs better than the use of $\mathbf{aggr}_v^1$. The results demonstrate that modeling the interrelationship information among the nodes in a local neighbourhood

effectively contributes to performance improvement.

5. Conclusion

Learning discriminative node embeddings is significant for graph representation learning. In this paper, we proposed a graph representation learning framework that generates node embeddings through two-stage local feature aggregation in a graph convolutional layer. In the first stage, we follow the conventional message passing approach to generate node embeddings by aggregating features from a node's local neighborhood. In the second stage, we learn a feature between the first-stage aggregated feature and each of the node's neighbour node feature and then aggregate these learned features. The aggregated features from the two stages are added up together as a node's representation. For a node, the first-stage aggregated feature encodes the feature information from the node's local neighbourhood. By applying our second-stage aggregation, the interrelationship information among the node's local neighborhood nodes is integrated in the node's representation. This helps to learn improved node embeddings for learning on graph-structured data. We experimentally evaluated our graph convolutional network on four graph domain tasks, including superpixel graph classification, node classification, edge prediction and graph regression, on six popular benchmark datasets including MNIST, CIFAR10, PATTERN, CLUSTER, TSP and ZINC. We demonstrated that our graph convolutional network achieves improved performance compared to a wide variety of baseline models. We conducted ablations of our model to show that integrating the interrelationship information among local neighbour nodes into a node's representation consistently improves the graph representation learning performance, which demonstrates that our two-stage aggregation framework is a general framework for learning on graph-structured data. The idea behind our approach can generalized to transformer models, which essentially treats the tokens as a fully connected graph. In our future, we will evaluate our idea on transformer models for different sequence blearing tasks and computer vision tasks.

6. Acknowledgement

The authors would like to thank the editor and reviewers for reviewing our manuscript.

References

[1] W. L. Hamilton, Graph representation learning, Synthesis Lectures on Artifical Intelligence and Machine Learning 14 (3) (2020) 1–159.

[2] Q. Dai, X.-M. Wu, L. Fan, Q. Li, H. Liu, X. Zhang, D. Wang, G. Lin, K. Yang, Personalized knowledge-aware recommendation with collaborative and attentive graph convolutional networks, Pattern Recognition 128 (2022) 108628.

[3] S. Feng, C. Xu, Y. Zuo, G. Chen, F. Lin, J. XiaHou, Relation-aware dynamic attributed graph attention network for stocks recommendation, Pattern Recognition 121 (2022) 108119.

[4] X.-b. Ye, Q. Guan, W. Luo, L. Fang, Z.-R. Lai, J. Wang, Molecular substructure graph attention network for molecular property identification in drug discovery, Pattern Recognition 128 (2022) 108659.

[5] W. Zheng, L. Yin, X. Chen, Z. Ma, S. Liu, B. Yang, Knowledge base graph embedding module design for visual question answering model, Pattern Recognition 120 (2021) 108153.

[6] D. Cheng, F. Yang, S. Xiang, J. Liu, Financial time series forecasting with multi-modality graph neural network, Pattern Recognition 121 (2022) 108218.

[7] M. Gori, G. Monfardini, F. Scarselli, A new model for learning in graph domains, in: Proceedings. 2005 IEEE international joint conference on neural networks, Vol. 2, 2005, pp. 729–734.

[8] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE transactions on neural networks 20 (1) (2008) 61–80.

[9] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, International Conference on Learning Representations.

[10] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Advances in neural information processing systems, 2017, pp. 1024–1034.

[11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph Attention Networks, International Conference on Learning RepresentationsAccepted as poster.
URL https://openreview.net/forum?id=rJXMpikCZ

[12] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: International Conference on Learning Representations (ICLR2017), 2017.

[13] X. Bresson, T. Laurent, Residual gated graph convnets, arXiv preprint arXiv:1711.07553.

[14] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, X. Bresson, Benchmarking graph neural networks, arXiv preprint arXiv:2003.00982.

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

[16] J. Chen, T. Ma, C. Xiao, Fastgcn: fast learning with graph convolutional networks via importance sampling, International Conference on Learning Representations.

[17] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, D.-Y. Yeung, Gaan: Gated attention networks for learning on large and spatiotemporal graphs, UAI 2018.

[18] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[19] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International conference on machine learning, PMLR, 2015, pp. 448–456.

[20] K. Sun, Z. Zhu, Z. Lin, Adagcn: Adaboosting graph convolutional networks into deep models, in: International Conference on Learning Representations, 2020.

[21] J. Klicpera, A. Bojchevski, S. Günnemann, Combining neural networks with personalized pagerank for classification on graphs, in: International Conference on Learning Representations, 2019.

[22] L. Zhao, L. Akoglu, Pairnorm: Tackling oversmoothing in gnns, International Conference on Learning Representations.

[23] Y. Rong, W. Huang, T. Xu, J. Huang, Dropedge: Towards deep graph convolutional networks on node classification, in: International Conference on Learning Representations, 2020.

[24] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, S. Jegelka, Representation learning on graphs with jumping knowledge networks, in: International Conference on Machine Learning, PMLR, 2018, pp. 5453–5462.

[25] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, J. Tang, Graph random neural networks for semi-supervised learning on graphs, Advances in Neural Information Processing Systems 33.

[26] M. Balcilar, G. Renton, P. Héroux, B. Gaüzère, S. Adam, P. Honeine, Analyzing the expressive power of graph neural networks in a spectral perspective, in: International Conference on Learning Representations, 2020.

[27] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, in: International conference on machine learning, PMLR, 2017, pp. 1263–1272.

[28] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, M. M. Bronstein, Geometric deep learning on graphs and manifolds using mixture model cnns, in:

Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5115–5124.

[29] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, in: International Conference on Learning Representations, 2019.

[30] Z. Chen, S. Villar, L. Chen, J. Bruna, On the equivalence between graph isomorphism testing and function approximation with gnns.

[31] H. Maron, H. Ben-Hamu, H. Serviansky, Y. Lipman, Provably powerful graph networks, Advances in neural information processing systems 32.

[32] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[33] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images.

[34] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, Slic superpixels compared to state-of-the-art superpixel methods, IEEE transactions on pattern analysis and machine intelligence 34 (11) (2012) 2274–2282.

[35] E. Abbe, Community detection and stochastic block models: recent developments, The Journal of Machine Learning Research 18 (1) (2017) 6446–6531.

[36] D. Applegate, R. Bixby, V. Chvatal, W. Cook, Concorde tsp solver.

[37] J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad, R. G. Coleman, Zinc: a free tool to discover chemistry for biology, Journal of chemical information and modeling 52 (7) (2012) 1757–1768.

[38] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representations, 2015.

[39] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, et al., Deep graph library: Towards efficient and scalable deep learning on graphs, in: ICLR workshop on representation learning on graphs and manifolds, 2019.