Pattern Recognition 128 (2022) 108661

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Node-Feature Convolution for Graph Convolutional Networks

Li Zhang[a], Heda Song[b], Nikolaos Aletras[a], Haiping Lu[a,*]

[a] Department of Computer Science, The University of Sheffield, 211 Portobello, Sheffield S1 4DP, United Kingdom
[b] Department of Computer Science. University of Nottingham. United Kingdom

ARTICLE INFO

Article history:
Received 9 July 2020
Revised 15 March 2021
Accepted 19 March 2022
Available online 26 March 2022

Keywords:
Graph
Representation learning
Graph convolutional networks
Convolutional neural networks

ABSTRACT

Graph convolutional network (GCN) is an effective neural network model for graph representation learning. However, standard GCN suffers from three main limitations: (1) most real-world graphs have no regular connectivity and node degrees can range from one to hundreds or thousands, (2) neighboring nodes are aggregated with fixed weights, and (3) node features within a node feature vector are considered equally important. Several extensions have been proposed to tackle the limitations respectively. This paper focuses on tackling all the proposed limitations. Specifically, we propose a new node-feature convolutional (NFC) layer for GCN. The NFC layer first constructs a feature map using features selected and ordered from a fixed number of neighbors. It then performs a convolution operation on this feature map to learn the node representation. In this way, we can learn the usefulness of both individual nodes and individual features from a fixed-size neighborhood. Experiments on three benchmark datasets show that NFC-GCN consistently outperforms state-of-the-art methods in node classification.

1. Introduction

Graphs, such as social, biological and citation networks, are ubiquitous data structures capturing interactions between individual nodes [1,2]. Nodes in a graph are typically associated with feature vectors. For example, in a citation network, nodes represent documents, edges represent the citation links between documents, and node features represent textual information often as bag-of-words, i.e., sparse vectors of weighted word frequencies in a document.

Graph Convolutional Network (GCN) [3] was recently proposed to model graphs using neural networks, with successful applications in node classification, link prediction, recommendation [4,5]. GCN effectively combines structure information and node features in a graph. It represents a node by aggregating the feature vectors of all its neighbors, analogous to the receptive field of a *convolutional kernel* in Convolutional Neural Networks (CNNs) [6].

Nevertheless, the standard GCN has three limitations: (1) the number of neighbors typically varies (e.g., from one to hundreds) across nodes so sparsely connected nodes have insufficient information while densely connected nodes may have redundant information; (2) the relevance of individual neighbors of a node is fixed (depending on the node degree), lacking the flexibility to characterize different relationships between nodes; (3) individual features in neighbor feature vectors may have different usefulness but this is not modeled. Figure 1 shows an example. The central node 0 belongs to Class A (*Neural Networks*) and it can be cited (i.e., connected) by papers from Class B (*Probabilistic Methods*). Node 5 from Class B may contain some common features with the central node 0 from Class A, e.g., *neuron*, and also some features more unique for Class B, e.g., *posterior*. Thus, the feature *neuron* should be more important than *posterior* for representing the central node. However, these features are equally weighted in existing GCN methods.

GCN extensions have been proposed to address the mentioned limitations: (1) sampling-based methods sample a fixed-size set of neighbors or learn an adaptive receptive fields for the given node; (2) neighbor weighting-based methods learn to treat different neighbors differently instead of simple aggregation. However, the sampling-based methods can not treat different features within a feature vector differently and directly weighting all neighbors may bring in too much noisy information and further influence the result. Besides, weighting each neighbors will be time consuming and unnecessary, especially for dense graphs. Existing works did not solve all the limitations together.

In this paper, we propose a novel method called Node-Feature Convolution for Graph Convolutional Network (NFC-GCN) to solve all the mentioned problems. Our method learns to assign different

* Corresponding author at The University of Sheffield, 211 Portobello, Sheffield, S1 4DP, United Kingdom.
E-mail addresses: lzhang72@sheffield.ac.uk (L. Zhang), Heda.Song@nottingham.ac.uk (H. Song), n.aletras@sheffield.ac.uk (N. Aletras), h.lu@sheffield.ac.uk (H. Lu).

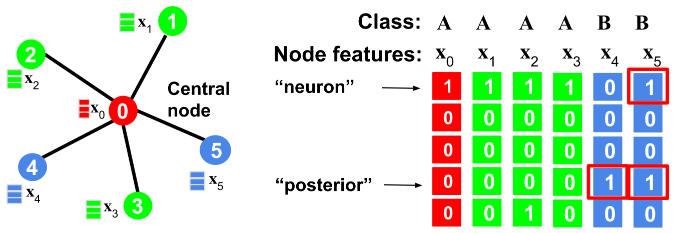


Fig. 1. A six-node subgraph from the Cora dataset [7]. Each node corresponds to a machine learning paper, with a bag-of-words feature vector $\mathbf{x}_i$ $(i = 0, 1, 2, ..., 5)$. Nodes 0–3 belong to Class A (*Neural Networks*), and nodes 4–5 belong to Class B (*Probabilistic Methods*). Individual features in $\mathbf{x}_i$ are not equally important for representing the central node 0.

weights to individual node features to get a new representation of a given node in three steps: (1) we first select a fixed-size set of neighbors (*Neighbor Selection*) according to the similarity between the feature vectors of the given node and its neighbors to construct a fixed-size feature map; (2) subsequently, we introduce a convolutional layer (*Node-Feature Convolution (NFC)*), to learn a first-level representation by assigning different weights to node features; (3) finally, we feed the output of the NFC layer to a *Standard GCN* to obtain a second-level node representation.

Our key contributions are: (1) We propose a new architecture, the NFC layer for GCN-based models, to enable *end-to-end learning* of weights for different features within the feature vector by applying the CNN; (2) we apply the neighbor selection strategy to only select most related neighbors for neighborhood aggregation, which alleviates the neighborhood explosion problem, computational burden and allows for a deeper model; (3) the number of our model's parameters are not related to the dimension of input data, because it inherits the advantage of CNN.

The rest of the paper is organized as follows. We review preliminaries and related works in Section 2. Then we present the proposed method in Section 3. Next, we report the experimental results in Section 4. Finally, we conclude this paper in Section 5.

2. Preliminaries and Related Works

2.1. Notations

We focus on modeling graphs with node features available. We consider graphs with a feature vector associated with each node. Let $\mathcal{G}= (\mathcal{V}, \mathcal{E}, \mathbf{X})$ denote an undirected graph with N nodes $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$, where $i, j = 1, ..., N$, an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, and a feature matrix $\mathbf{X} \in \mathbb{R}^{N\times D}$ containing N D-dimensional feature vectors.

2.2. Graph Representation Learning

Graph representation leaning methods can be categorized into factorization-, random walk- and neural network-based approaches [8].

1. **Factorization-based approaches.** Early methods for learning node representations mainly focus on matrix factorization approaches, such as Locally Linear Embedding (LLE) [9], Laplacian Eigenmaps (LE) [10], Graph Factorization (GF) and HOPE [11]. These methods represent the connections between nodes in the form of a matrix and obtain node embeddings by factorizing the matrix.
2. **Random walk-based approaches.** Random walks over graphs have been used to capture the structural relationships between nodes. Inspired by Word2Vec, DeepWalk generated random paths over a graph, and then applied the SkipGram model to maximize the co-occurrence probability of the neighbors conditioned on a given node embedding [12]. Node2vec [13] extended DeepWalk with biased-random walks, while Walklets [14] modified the random walk process by skipping over nodes in the graph. Diffusion based network embedding [15] records all the visited nodes and transforms the single-trace random walks into multiple-trace random walks.
3. **Neural network-based approaches.** Graph neural networks (GNNs) have been introduced in [16]. They consist of an iterative process which propagates the node states until the node representation reaches a stable fixed point. More recently, several improved methods have been proposed. In [17], gated recurrent units were introduced in the propagation step. The neural graph fingerprints method [18] further introduced a convolution-like propagation rule. PATCHY-SAN [19] selected and normalized a fixed-size neighborhood for a given node, then CNNs were used to learn the neighborhood structure information. Structure Deep Network Embedding (SDNE) [20] extended the traditional deep autoencoder to learn from the node structure information to get a low-dimensional embedding for each node.

2.3. Graph Convolutional Networks

The factorization-based, random walk-based and mentioned neural network-based methods (such as PATCHY-SAN, SDNE) only utilize graph structure (without node features or attributes) to learn new node representations. GCN was proposed as an effective graph representation learning model that naturally combines structure information and node features in the learning process [3]. It was derived from conducting graph convolution in the spectral domain [21]. The spectral convolutions on graphs can be defined as the multiplication of a signal $\mathbf{x} \in \mathbb{R}^N$ with a filter $g_{\mathbf{w}}$ parameterized by $\mathbf{w} \in \mathbb{R}^N$ in the Fourier domain as following:

$$g_{\mathbf{w}} \star \mathbf{x} = \mathbf{U} g_{\mathbf{w}}(\mathbf{\Lambda})\mathbf{U}^T\mathbf{x}. \tag{1}$$

$g_{\mathbf{w}}$ can be seen as a function of eigenvalues of $\mathbf{L}$, and $\mathbf{L}$ is defined as

$$\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T, \tag{2}$$

where $\mathbf{I}_N$ is an identity matrix, $\mathbf{D}$ is a diagonal degree matrix with $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. $\mathbf{\Lambda} = diag([\lambda_0, ..., \lambda_{N-1}]) \in \mathbb{R}^{N\times N}$ and $\lambda_0, ..., \lambda_{N-1}$ are eigenvalues of $\mathbf{L}$. Eq. (1) incurs expensive computation of the Laplacian eigenvectors ($\mathcal{O}(N^2)$). To circumvent this problem, a polynomial function computed recursively from $\mathbf{L}$ can be used to approximate $g_{\mathbf{w}}(\mathbf{\Lambda})$ [22]. The Chebyshef polynomial are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. Then a filter can be parameterized as the truncated expansion with order K that can be written as:

$$g_{\mathbf{w}}(\mathbf{\Lambda}) \approx \sum_{k=0}^{K} \mathbf{w}_k T_k(\tilde{\mathbf{\Lambda}}), \tag{3}$$

where $\tilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{max} - \mathbf{I}_N$ is a diagonal matrix of scaled eigenvalues in [-1,1]. The filter operation can be written as:

$$g_{\mathbf{w}} \star \mathbf{x} \approx \sum_{k=0}^{K} \mathbf{w}_k T_k(\tilde{\mathbf{L}})\mathbf{x}, \tag{4}$$

where $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I}_N$. $T_k(\tilde{\mathbf{L}})$ can be calculated using the recurrence relation: $T_k(\tilde{\mathbf{L}}) = 2\tilde{\mathbf{L}}T_{k-1}(\tilde{\mathbf{L}}) - T_{k-2}(\tilde{\mathbf{L}})$, and the entire filter operation cost is $\mathcal{O}(KN)$. In [3], GCN simplified Eq. (4) and limited $K = 1$, and the filter parameter $\mathbf{w}_k$ was shared over the whole graph.

For signal $\mathbf{X} \in \mathbb{R}^{N\times D}$ with D input channels (i.e., a D-dimensional feature vector for every node) and D_1 filters, Eq. (4) (the convolved signal matrix) can be written as:

$$\mathbf{H}^{(1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)}\right), \quad (5)$$

where

$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$$

is a normalized adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_{\mathbf{N}},$$

$\tilde{\mathbf{D}}$ is defined with its diagonal entries as

$$\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij},$$

$\mathbf{W}^{(0)} \in \mathbb{R}^{D \times D_1}$ is a trainable input-to-hidden weight matrix, $\sigma(\cdot)$ denotes an activation function such as the $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$, and $\mathbf{H}^{(1)} \in \mathbb{R}^{N \times D_1}$ is the matrix of activation in the first layer. Thus, the propagation rule can be written as:

$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right), \quad (6)$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{D_{l-1} \times D_l}$ is a layer-specific trainable weight matrix and $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times D_l}$ is the matrix of activation in the l-th layer. $\mathbf{H}^{(0)} = \mathbf{X}$ is the node feature matrix.

A GCN layer (Eq. (6)) consists of two steps: (1) aggregating the given node and its neighbors' feature vectors with different weights (according to the node degrees):

$$\hat{\mathbf{h}}_i^{(l)} = \mathbf{h}_i^{(l)} + \sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{d_i d_j}} \mathbf{h}_j^{(l)} (j \in \mathcal{N}_i), \quad (7)$$

where d_i and d_j are the node degrees of node v_i and node v_j respectively, and $\mathbf{h}_i^{(l)}$, $\mathbf{h}_j^{(l)}$ are the representations of v_i, v_j of the l-th layer, $\mathcal{N}_i$ is the neighborhood of v_i in the graph; (2) feeding the averaged feature vector to a fully-connected neural network.

GCN has significantly advanced the state-of-the-art in graph representation learning but it has three major limitations:

- **Varied neighborhood size.** GCN learns a new node representation from features of all its neighbors. In real-world graphs, the number of neighbors for a given node can range from one to hundreds or even thousands. Therefore, some nodes may not have sufficient number of neighbors to aggregate information, while some other nodes may have their own features $\mathbf{h}_i^{(l)}$ being "washed out" due to aggregating information from too many neighbors [23]. Moreover, varying number of neighbors can lead to neighborhood explosion which subsequently causes computational problems, e.g., excessive space (memory) [3] and time complexity [24,25].
- **Fixed neighbor weighting.** GCN aggregates neighbors with fixed weights inversely proportional to the central node and neighbors' node degrees [5]. Once the graph structure is given, the weights are fixed. This weighting strategy does not consider node features at all, limiting its ability to effectively capture the relationships between nodes.
- **Equal weighting of individual features.** GCN does not select or weight individual features in a feature vector. As discussed in Section 1 (Fig. 1), for a particular node, features from neighbors of different classes may have different importance compared to those from neighbors of the same class. Therefore, we hypothesize that we can obtain better node representations by weighting features individually within feature vectors and across neighbors.

2.4. GCN Extensions

Several methods have been proposed to deal with the first two limitations above:

- **Sampling-based methods.** Instead of considering all neighbors, some methods apply sampling strategies to only aggregate a part of neighbors. GraphSAGE [24] uniformly sampled a fixed number of neighbors and aggregated them with a sum, mean, LSTM or maxpooling aggregator as follows:

$$\hat{\mathbf{h}}_i^{(l)} \approx \mathbf{h}_i^{(l)} + \mathit{aggregator}(\mathbf{h}_j^{(l)}, j \in \hat{\mathcal{N}_i}), \quad (8)$$

where $\hat{\mathcal{N}_i}$ is the neighborhood generated by a fixed-length random walk, and they can come from a different number of hops, or search depth, away from a given node. FastGCN [25] interpreted graph convolutions as integral transforms of embedding functions and directly sampled the nodes in each layer independently. It approximated $\hat{\mathbf{h}}_i^{(l)}$ with s i.i.d. samples $v_1, \ldots, v_s$ as:

$$\hat{\mathbf{h}}_i^{(l)} \approx \frac{N}{s} \sum_{v_s \sim q(v)} \hat{\mathbf{A}}_{ij} \mathbf{h}_s^{(l)} / q_{(v_s)}, \quad (9)$$

where the importance distribution for each node v_i is $q(v_i) \propto \|\mathbf{A}(:, i)\|^2$. JP-networks [26] sampled learned intermediate representations for a given node to get the final node representation. Besides sampling strategies, some dropout tricks are proposed. DropEdge [27] randomly removed a set of edges, and Graph DropConnect (GDC) [28] and GeniePath [29] learned the connections in a graph, jointly with GNN model parameters. These sampling-based methods mainly focus on how to select neighbors, but do not treat the selected neighbors differently in the latter aggregation step.
- **Neighbor weighting-based methods.** In GCN, a central node aggregates all neighbors with fixed weights decided by the graph structure (node degree). Many algorithms have been proposed to treat the neighbors differently in neighborhood aggregation process. Disentangled graph convolutional network (DisenGCN) [30] proposed a neighborhood routing mechanism to identify the factor that may have caused the link from a given node to one of its neighbors, and accordingly send the neighbor to the channel responsible for that factor. Then each channel can perform an aggregation independently, which means each *cluster* of neighbors are treated differently in DisenGCN. Inspired by attention mechanisms [31], Graph Attention Networks (GAT) introduced an attention mechanism to dynamically assign weights to different neighbors [32] as:

$$\hat{\mathbf{h}}_i^{(l)} = \mathbf{h}_i^{(l)} + \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{h}_j^{(l)}, \quad (10)$$

where α_{ij} is the learned weights with a shared attention mechanism. Although each node is treated differently, all individual features in a feature vector share the same weight, without considering their *individual importance*. Masked GCN [33] learned a diagonal mask matix that can determine which attributes can be propagated to the central node, LA-GCN [34] and GNN-Film respectively introduced an auxiliary model and feature-wise linear modulations (FiLM) [35] for a feature-wise modulation in the neighborhood aggregation process. The two methods considered all neighbors, which is not necessary and directly learning the mask or FiLM need a very huge model, especially for graphs with high-dimensional node features. Learnable Graph Convolutional Layer (LGCL) [36] applied CNN on the *reorganized embeddings* (learned from GCN) of the central node and its neighbors, rather than the original features. The reorganization of node embeddings broke the original correspondence between node representations.

To the best of our knowledge, our work is the first one to consider both neighbor sampling and neighbor reweighting in both node and feature level. Our model only selects the fixed-size and most related neighbors and this strategy can effectively alleviate the neighborhood explosion problem and allow for a deeper

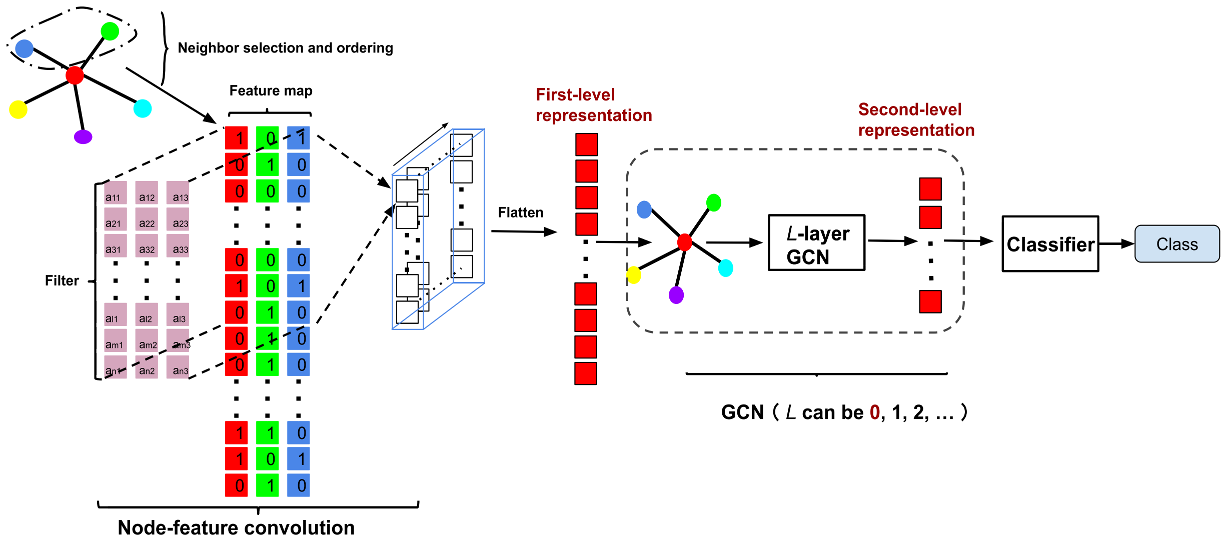

Neighbor selection and ordering
Feature map
Filter
Flatten
First-level representation
Second-level representation
L-layer GCN
Classifier
Class
GCN (L can be 0, 1, 2, ...)
Node-feature convolution

Fig. 2. NFC-GCN architecture. NFC-GCN consists of three main steps: (1) *Neighbor selection and ordering*; (2) *Node-feature convolution* operating on node-feature maps to obtain a flattened first-level node representation; and (3) *GCN:* the first-level NFC representation is passed through an *L*-layer GCN model (*L* is a hyperparameter) to learn a second-level node representation is passed to a classifier. The figure is best viewed in color/on screen.

model. Then we apply CNN on the feature map containing features from the selected neighbors to learn different weights for different features from neighboring nodes.

3. Proposed NFC-GCN

Figure 2 shows the proposed NFC-GCN model. The key idea of our approach is to design a node-feature convolution layer to learn different weights for different features in different neighbors before any aggregation.

3.1. Neighbor Selection and Ordering

To deal with varying node degrees in a graph, we select the most useful neighbors to obtain a feature map with fixed size. Nodes can be ordered using common node centrality metrics such as node degree, betweenness centrality, eigenvector centrality and PageRank. Previous works mainly focus on selecting nodes without considering node features. In this paper, we perform neighbor selection and ordering based on the cosine similarity between the central node v_i and its neighbors v_j:

$$sim_{ij} = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}, \tag{11}$$

where $\mathbf{x}_i \in \mathbb{R}^D$ and $\mathbf{x}_j \in \mathbb{R}^D$ are the feature vectors of v_i and v_j respectively. By specifying a hyperparameter *feature map bandwidth* n, we select the top $n-1$ neighbors with the highest similarity with the central node.

In practice, sparsely connected nodes may have less than $n-1$ neighbors. In this case, we select from the central node and all its neighbors based on probabilities *proportional* to the similarity sim_{ij} to get the desired feature map bandwidth n. For each node v_i, we obtain a **local feature map** $\mathbf{X}_i^{'} \in \mathbb{R}^{D\times n}$

$$\mathbf{X}_i^{'} = \left\{\mathbf{x}_i, \{\mathbf{x}_{j'}, j' \in \mathcal{N}'_i\}\right\}, \tag{12}$$

consisting of the feature vectors of the given node i and its selected neighbors $j' \in \mathcal{N}'_i$, where $\mathcal{N}'_i$ represents the selected neighbors of node v_i. This feature map can be seen from two dimensions: (1) the first dimension represents D node features, e.g., bag-of-words features with a fixed order for citation networks; (2) the second dimension represents the n nodes, including the central node and the $n-1$ selected neighbors. These nodes are ordered according to the neighbors' feature similarity with the central node from high to low.

Selecting a fixed number of neighbors can prevent neighborhood explosion and central node being "washed out". According to [26], the influence distribution of v_j on v_i can show how much a change in a neighbor v_j affects the final representation of the central node v_i in the last layer.

The *influence score and distribution* definition in [26] states that for a graph $\mathcal{G}= (\mathcal{V}, \mathcal{E}, \mathbf{X})$, $\mathbf{h}_i^{(0)}$ is the input feature and $\mathbf{h}_i^{(l)}$ is the learned hidden feature of v_i at the l-th layer (Eq. (6)). The influence score $I(i, j)$ of v_i by any v_j is the sum of the absolute values of the entries of the Jacobian matrix $\frac{\partial \mathbf{h}_i^{(l)}}{\partial \mathbf{h}_j^{(0)}}$. After $\mathcal{O}(\log|N|)$ iterations of neighborhood aggregation using all neighbors, the representation of each node is "influenced almost equally by any other node" [37]. Thus, the final node representation captures mainly the global graph, with limited information about individual nodes. Performing neighbor selection for each GCN layer can alleviate the neighborhood explosion.

3.2. Node-Feature Convolution (NFC)

As a representation learning method, CNN works on fixed-size grids (e.g., images) or sequences (e.g., sentences) to tackle various problems such as image classification [2], machine translation [31], text or sentence classification [38] successfully. The *Neighbor Selection and Ordering* step enables us to apply a convolution operation on the node-feature map $\mathbf{X}_i^{'}$ obtained.

In the citation graph, nodes usually represent documents, edges represent the citation links between documents, and node features represent textual information often as bag-of-words. The 0/1-valued feature vector of a node corresponds to an ordered word list from a dictionary, which is analogous to the ordered words in a sentence or document of an NLP task [38]. For example, a part of the ordered word list of Cora includes "Machine", "Markov", "Monte-Carlo", "Neural", "Network", then the local feature pattern "11100" could be highly related to the category of "Reinforcement Learning" and "10011" could be related to the category of "Neural Networks". The adjacent features in a node feature vector are re-

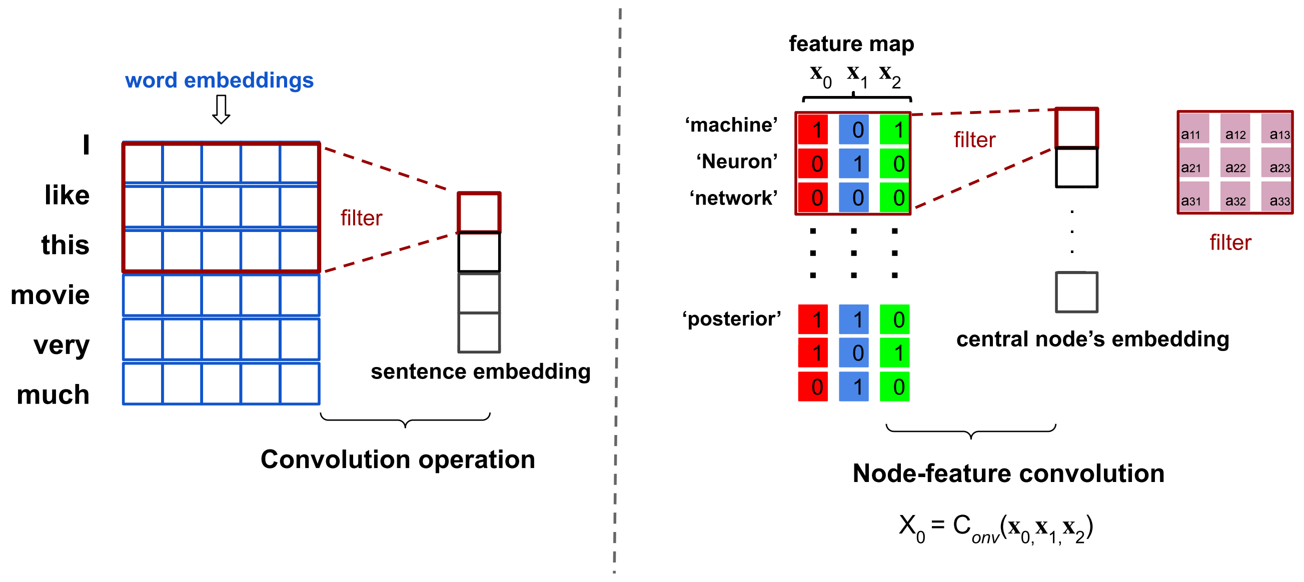


Fig. 3. Convolution on sentence and node feature map. The node feature corresponds to the word list, and the number of neighbors corresponds to the dimension of each word vector.

lated. Therefore, we can use convolutions to find the potential local patterns that indicate a category.

As shown in Fig. 3, a fixed-size convolutional kernel scans over ordered words to obtain the representation of a sentence. Each 0/1-valued feature of our citation datasets (e.g. Cora, Citeseer) indicates the absence/presence of a corresponding word from a dictionary, which naturally inspires us to use 1-D convolution to scan over the feature vector of a node.

We perform convolution with C filters of size k and stride s on the local feature map $\mathbf{X}_i^{'} \in \mathbb{R}^{D\times n}$ of each node as

$$\vec{\mathbf{X}}_i = C_{onv}(\mathbf{X}_i^{'}). \tag{13}$$

The number of input channels is n. The output $\vec{\mathbf{X}}_i$ is of dimension $D' \times C$, where D' is determined by k, s, and C, the hyperparameters of NFC. Then, we flatten the output as following:

$$\mathbf{h}_i^{(0)} = flatten(\vec{\mathbf{X}}_i), \tag{14}$$

which is the first-level node representation.

3.3. GCN Layers

Nodes with sparse connectivity (few first-order neighbors) may have insufficient information and need higher-order neighbors' information to obtain better representations. Better representation of a given node can be obtained by considering L-order neighbors, where the best value for L depends on the data. Therefore, we pass the NFC representation through L additional GCN layer(s) to enable a central node aggregating information from higher-order neighbors. An aggregation operator works on the first-level node representation (Eq. (14)) to learn another new representation of node i, as in Eq. (6).

After L GCN layers, the final representation $\mathbf{h}_i^{(L)}$ will be passed to a fully-connected layer with a $softmax$ activation function. For multi-class classification, the loss function is defined as the cross-entropy error over all labeled examples:

$$\mathcal{L} = -\sum_{l\in\mathcal{V}_l}\sum_{f=1}^{F}\mathbf{Y}_{lf}\ln\mathbf{h}_l^{(L)}, \tag{15}$$

where $\mathcal{V}_l$ is the set of node indices that have labels and F is the dimension of output features equaling to the number of classes.

Algorithm 1 Pseudocode for the proposed NFC-GCN

```
Input: $\mathcal{G}$ = $(\mathcal{V}, \mathcal{E}, \mathbf{X})$ with $N$ nodes;
Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$;
Feature matrix $\mathbf{X} \in \mathbb{R}^{N\times D}$;
Labeled nodes $\mathcal{V}_l$;
Label indicator matrix $\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l|\times F}$;
The number of selected neighbors is $(n$-$1)$;
The parameters in the node-feature convolution process: filter size $k$, stride $s$, the number of filters: $C$, the convolution operation $C_{onv}(\cdot)$
Output: Vector representation $\mathbf{h}_i^{(L)}$
for each $v_i \in \mathcal{V}_l$ do
    if $d_i > n-1$ then
        choose $n-1$ neighbors according to similarity from high to low
    else if then
        select $(n-1-d_i)$ nodes based on probabilities proportional to the similarity
    end if
    $\mathbf{X}'_i = \{\mathbf{x}_i, \mathbf{x}_{j'}, j' \in \mathcal{N}'_i\}_n$
    $\vec{\mathbf{X}}_i = C_{onv}(\mathbf{X}'_i)$
    $\mathbf{h}_i^{(0)} = flatten(\vec{\mathbf{X}}_i)$
    for each layer $l$, $l$=1,...,$L$ do
        $\mathbf{h}_i^{(l)} = \sigma(\mathbf{W}^{(l)}(\mathbf{h}_i^{(l-1)} + \sum_{j\in\mathcal{N}_i}\mathbf{h}_j^{(l-1)}))$;
    end for
end for
```

$\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l|\times F}$ is a label indicator matrix. Algorithm 1 summarizes the general framework of NFC-GCN.

Figure 4 demonstrates the neighbor selection achieved via different models (GCN-GCN, NFC-GCN, NFC-NFC). Considering a two-layer GCN (i.e., GCN-GCN) in Fig. 4(b), after the first propagation (first GCN layer), each node (e.g., the red node) only contains the first-order neighbors' information (green nodes). After the second propagation (second GCN layers), each node aggregates information from all its first-order and second-order neighbors (purple nodes). Figure 4 (c) shows NFC with one GCN layer (i.e., NFC-GCN). In the first propagation, each node (the red node) only aggregates information of the top two most similar first-order neighbors

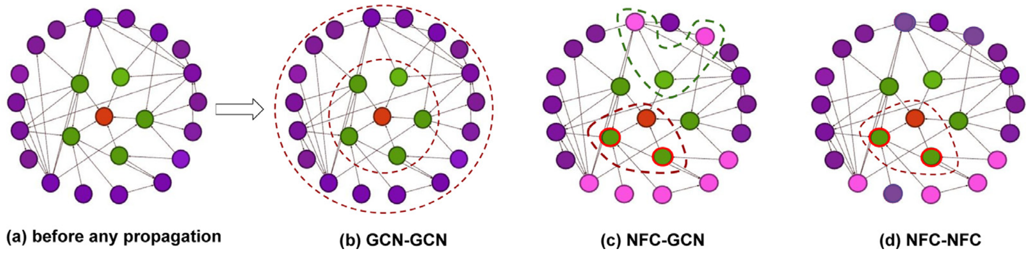


Fig. 4. Comparison with different layers. Take the red node as an example, the red node's first-order and second-order neighbors are respectively green and purple nodes, as shown on the left. After a two-layer GCN, the central node contains information from all first-order neighbors and second-order neighbors as shown in (b) GCN-GCN. After one NFC and one GCN layer, each node contains information from all its first-order (directly) and part of its second-order neighbors' information (indirectly) as in (c) NFC-GCN. After two NFC layers ((d) NFC-NFC), the central node only contains the two most similar first-order neighbors and part (less than in NFC-GCN) of second-order neighbors' information.

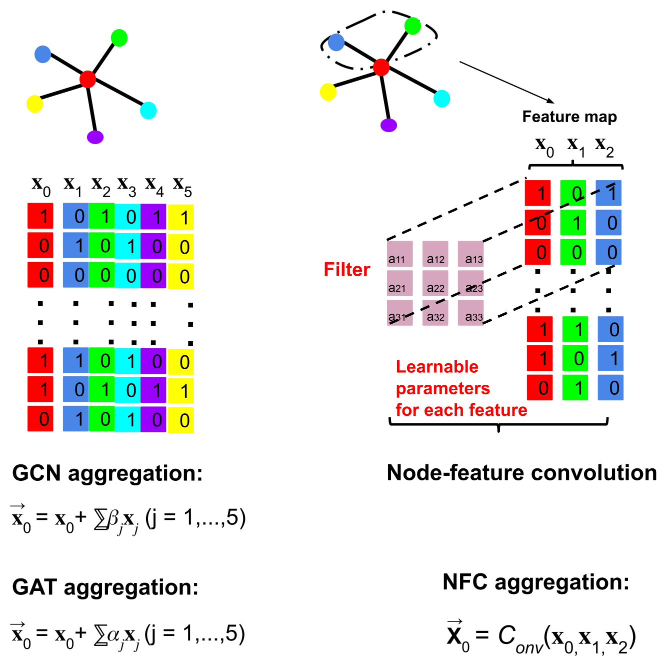

x0 x1 x2 x3 x4 x5
Feature map
x0 x1 x2
Filter
a11 a12 a13
a21 a22 a23
a31 a32 a33
Learnable parameters for each feature
GCN aggregation:
$\vec{x}_0 = x_0 + \sum \beta_j x_j$ (j = 1,...,5)
Node-feature convolution
GAT aggregation:
$\vec{x}_0 = x_0 + \sum \alpha_j x_j$ (j = 1,...,5)
NFC aggregation:
$\vec{X}_0 = C_{onv}(x_0, x_1, x_2)$

Fig. 5. Differences between GCN, GAT, and NFC-GCN. In the aggregation process, both GCN and GAT aggregate all the neighbors with different weights. The weights β_j, for each neighbor related to node degree are fixed in GCN. While α_j is learnable in GAT. But all the features in each feature vector share the same weights β_j, or $\alpha_i, i, j \in (1, 5)$. In contrast, our method performs convolution operation on the selected node-feature map to assign different weights (such as $a_{11}, a_{12}, \ldots, a_{33}$) to different features in different neighbors.

(two green nodes with red circles). After the first propagation (NFC layer), each green node's representation also contains information from its two most similar neighbors as well (the dash green curve in (c) NFC-GCN). After the second GCN layer, the central node's representation contains information from all the first-order neighbors (green nodes) but only part of (selected) second-order neighbors (pink nodes). Therefore, NFC-GCN only selects part of the neighbors for the central node's representation learning even after adding GCN layers. Compared with NFC-GCN, each node contains less information from its first-order and second-order neighbors after two NFC layers. Because the central node (red node) aggregates two green nodes with the red circle, each green node contains information of its two most related neighbors (pink nodes) after the first NFC layer. In the second propagation (after the second NFC layer), the central node still aggregates two first-order neighbors who already contain two of their own first-order neighbors' (pink nodes) information. Therefore, after two NFC layers, each node contains two first-order neighbors and part of second-order neighbors information (indirectly). Compared to the standard GCN model (GCN-GCN - $\cdots$), NFC-GCN and NFC-NFC can alleviate the neighborhood explosion (over smoothing problem) and help avoid the central node being "washed out" due to aggregating too many neighbors. A pure NFC model (NFC-NFC-$\cdots$) can reduce the considered neighbors further. But how much information (the number of neighbors) should be considered for the best representation learning of a given node has no precise answer.

3.4. Computational Complexity

A key part in our method is the NFC-layer, and the filters are shared by all nodes in a graph. Therefore, the computation of the parameters in the filters can be parallelized across all nodes. The computational complexity of a GCN layer as shown in Eq. (6) is $\mathcal{O}(N \times D_{l-1} \times D_l)$, while an NFC-layer (Eq. (13)) is $\mathcal{O}(N \times C \times k \times n)$. In GCN, the models' complexity is related to the node feature dimension ($\mathbf{W}^{(0)} \in \mathbb{R}^{D_1 \times D}$) and this may lead to many parameters in the model if the dimension of the original node feature is high. It should be emphasized that the NFC layer inherits the advantage of CNN whose parameters are not related to the image size, so the complexity of NFC layer is not influenced by the node feature (C, k, n are hyperparameters).

3.5. Differences with other GCN Extensions

Sampling-based methods. Our method selects the neighbors according to their similarities with the central node from high to low. While, GraphSAGE [24] and DropEdge [27] select the neighbors randomly with random work or drop out some edges in a graph. FastGCN [25] directly sampled the nodes in each hidden layer independently and JP-networks [26] sampled learned intermediate representations for a given node to get the final node representation. Graph DropConnect (GDC) [28] and GeniePath [29] automatically learn the connections in a graph.

Sampling-based methods. Our method selects the neighbors according to their similarities with the central node from high to low. In contrast, GraphSAGE [24] and DropEdge [27] select the neighbors randomly with random work or drop out some edges in a graph. FastGCN [25] directly sampled the nodes in each hidden layer independently and JP-networks [26] sampled learned intermediate representations for a given node to get the final node representation. Graph DropConnect (GDC) [28] and GeniePath [29] automatically learn the connections in a graph.

Neighbor weighting-based methods. In DisenGCN and GAT, all the features within a feature vector still shared the same weight. In contrast, our method selects part of the neighbors and learns to assign different weights to different features in different neighbors. Masked GCN, LA-GCN and GNN-Film can do a feature-level attention, but they consider all neighbors in the learning process, which may introduce much noisy information. Besides, the complexity of their models is related to the node feature, which can be time-consuming for high-dimension input. Our model only selects the most related neighbors before aggregation and applies CNN in the aggregation. Our model inherits the advantage of CNN and its model size is not related to the input data. LGCL used CNN after a GCN layer, so it inherited the limitations of GCN as we mentioned above. In contrast, our method uses CNNs in the first step to extract useful information from raw node features. Besides, LGCL constructed the feature map by selecting the k-largest values for each feature from *all* neighbors' embeddings (learned from one GCN layer), which broke the correspondence in the original node embedding. In our method, the feature map is constructed from the central node and neighbors' (selected and ordered) raw node features. Therefore, the constructed feature map has a consistent structure, which is suitable for CNNs to perform on.

4. Experiments

In this section, we conduct extensive experiments on three real-world benchmark datasets to evaluate our proposed model from three main aspects: 1) the node classification performance in terms of accuracy and convergence; 2) the effectiveness of NFC aggregation; 3) parameter sensitivity. Finally, we analyze the limitations and advantages of our method.

4.1. Experimental Setup

Datasets. We use three citation network benchmark datasets, Cora, Citeseer and PubMed, that have been widely used in previous related work. We use the same train/validation/test splits as in [25]. Table 1 shows an overview of the datasets.

- **Cora.** The Cora dataset contains 2,708 documents (nodes) classified into seven classes (i.e., Neural Networks, Rule Learning, Probabilistic Methods,..., Reinforcement Learning) and 5,429 citation links (edges). We treat the citation links as (undirected) edges and construct a binary, symmetric adjacency matrix. Each document has a 1,433 dimensional sparse bag-of-word feature vector and a class label.
- **Citeseer.** The Citeseer dataset contains 3,327 documents classified into six classes (i.e., Agents, AI,..., ML) and 4,732 links. Each document has a 3,703 dimensional sparse bag-of-word feature vector and a class label.
- **PubMed.** The PubMed dataset contains 19,717 documents classified into three classes (Diabetes Mellitus Type Experimental, Diabetes Mellitus Type 1, Diabetes Mellitus Type 1) and 44,338 links. Each document has a 500 dimensional sparse bag-of-word feature vector and a class label.

Compared methods. We compare NFC-GCN against nine competing methods in total. We consider four representative non-GCN methods: Locally Linear Embedding (LLE) [9], Laplacian Eigenmaps (LE) [10], Graph Factorization (GF) [39] and DeepWalk [12] that only utilise graph structure in the node representation learning. We use the implementation provided by authors of [40] with standard settings used in their paper. In order to ensure the baselines have sufficient diversity, we compare against seven state-of-the-art models: GCN,[1] two sampling based methods (FastGCN,[2] GraphSAGE[3]) and four neighbor weighting-based methods (DisenGCN,[4] GAT,[5] LA-GCN,[6] LGCL[7]). We all use the publicly available implementation and report the mean accuracy of 100 runs with random weight initializations.

- **GCN.** Graph Convolutional Networks [3] is the standard baseline. In our experiments, we use a two-layer GCN model. For the key hyperparameters, we swept the number of hidden units in the set {16, 32, 64, 128}, L2 regularization $\{5\times10^{-3}, 5\times10^{-4}, 5\times10^{-5}\}$, dropout rate {0.2, 0.4, 0.6}, learning rate {0.01, 0.001, 0.0001}. We set the max training epoch to 1000 and early stopping to 10.
- **Sampling-based methods.** We compare our method with FastGCN [25] and GraphSAGE [24]. We split the train/validation/test as [25], so we use the same hyperparameters as in their paper. For FastGCN, we use two hidden layers, the batch size is 256 for Cora, Citeseer and 1024 for PubMed, the sample sizes are 400, 400 and 100 for Cora, Citeseer and PubMed, the learning rate is 0.001, and dropout is set as zero. For GraphSAGE, we apply the mean aggregator (GraphSAGE-mean usually gets the best results) and use two layers with neighborhood sample sizes 25 (for the first layer) and 10 (for the second), and the batch size is the same with FastGCN.
- **Neighbor weighting-based methods**. For these methods, we swept the common key hyperparameters: hidden units, L2 regularization, dropout rate and learning rate, as in GCN. Graph attention networks [32] learns to assign different weights to different neighbors. In the experiment, we apply two GAT layers and the number of attention heads are in the set {2,4,8}. Learnable Graph Convolutional Layer (LGCL) [36] performs convolution on the reconstructed node embeddings after one GCN layer. K ranges between {8,16,32} for the K-component feature vectors and dropout $\in$ {0.2, 0.4, 0.6, 0.8 } is applied on both input feature vectors and adjacency matrices in LGCN. We apply two LGCL layers for Cora and Citeseer, one LGCL layer for PubMed and stop the training within 10000 epochs. DisenGCN [30] can treat different cluster of neighbors differently and we set the number of channels $\in$ {4,8,16,32}. We apply neural networks as the auxiliary model in LA-GCN [34] with the number of layers $\in$ {1,2,3}. The max training epoch is set to 1000 for DisenGCN and LA-GCN.

Hyperparameters for NFC-GCN. We swept the common key hyperparameters: hidden units, L2 regularization, dropout rate and learning rate, as in GCN. Other key hyperparameters for NFC layer are set as: the number of neighbors $\in$ {1,2,3,4,5}, the filter size $\in$ {32, 64, 128}, the number of filters $\in$ {8,16,32, 64}, the number of stride $\in$ {16, 32, 64}. We employ the early stopping strategy based on the validation accuracy and train 200 epochs at most. As mentioned earlier, Compared with GCN layer, NFC layer is a more powerful tool, but it is more time consuming when the number of filters or neighbors is high. Considering both accuracy and efficiency, we use different combinations: NFC-GCN, NFC-NFC to learn the final node embeddings. Our code are available online.[8]

Table 1
Overview of the three datasets with standard splits as in the Fast-GCN [25] (Val. means Validation).

Dataset	Nodes	Edges	Features	Classes	Train/Val./Test
Cora	2,708	5,429	1,433	7	1,208/500/1,000
Citeseer	3,327	4,732	3,703	6	1,827/500/1,000
PubMed	19,717	44,338	500	3	18,217/500/1,000

[1] https://github.com/tkipf/gcn.
[2] https://github.com/matenure/FastGCN.
[3] https://github.com/williamleif/GraphSAGE.
[4] https://jianxinma.github.io/disentangle-recsys.html.
[5] https://github.com/PetarV-/GAT.
[6] https://github.com/LiZhang-github/LA-GCN.
[7] https://github.com/HongyangGao/LGCN.
[8] https://github.com/LiZhang-github/NFC-GCN.

Table 2
Node classification accuracy (mean ± std over 100 runs, **Best**, Second best)

Methods	Cora	Citeseer	PubMed
LLE	30.5 ± 0.02 %	20.5 ± 0.07 %	39.8 ± 0.01 %
LE	29.6 ± 0.05 %	21.2 ± 0.01%	39.8 ± 0.01 %
GF	30.7 ± 0.01 %	20.9 ± 0.03 %	39.9 ± 0.01 %
DeepWalk	55.2 ± 0.08%	44.1 ± 0.05%	77.6 ± 0.03%
GCN	87.1 ± 0.12%	78.1 ± 0.11%	87.3 ± 0.09%
Fast-GCN	85.0 ± 0.24%	77.6 ± 0.63%	88.0 ± 0.32%
GraphSAGE	82.2 ± 0.69%	71.4 ± 0.89%	87.1 ± 0.47%
GAT	86.9 ± 0.13%	77.7 ± 0.22%	87.2 ± 0.02%
LGCL	87.7 ± 0.11%	78.8 ± 0.20%	85.1 ± 0.06%
DisenGCN	87.4 ± 0.17%	77.0 ± 0.30%	87.2 ± 0.02%
LA-GCN	88.8± 0.12 %	78.7 ± 0.33 %	88.9 ± 0.10 %
NFC-GCN	88.7 ± 0.13%	**79.4 ±0.22%**	89.7± 0.07%
NFC-NFC	**89.6 ±0.14%**	78.9± 0.24%	**90.4 ±0.08%**

Table 3
Node classification accuracy for different aggregation methods with five neighbors and only one aggregation step (i.e., without further GCN layers).

Aggregation	Cora	Citeseer	PubMed
GCN	64.8%	74.1%	80.0%
GAT	64.2%	74.2%	82.2%
NFC	**86.0%**	**78.9%**	**89.7%**
Improvement	21.2%	4.8%	7.5%

4.2. Node Classification and Convergence

In this section, we compare the node classification accuracy with baselines, besides we compare the convergence with three mostly related methods: GCN, GAT, and LGCL.

- **Node classification.** Results in classification accuracy are summarized in Table 2. For the first four non-GCN methods, they do not perform well for they only utilise the structure information. FastGCN and GraphSAGE focus on improving the training efficiency, so they have slightly poorer results than GCN. GAT and LA-GCN utilize neural networks to learn attention scores or masks for a given node's neighbors. It is time consuming especially for Cora and Citeseer with high-dimensional node features. DisenGCN applies neighborhood routing mechanism to cluster a given node's neighbors, which is more suitable for dense graphs. While Cora, Citeseer and PubMed are relatively sparse and their median node degrees are 4, 3, 3 respectively. LGCL can get competitive results on Cora and Citeseer, while it does not perform well on PubMed. Two possible reasons are 1) it inherits the limitation of GCN and 2) it reorganises the original embedding in the process of constructing feature maps as mentioned in Section 3.5.
Our models NFC-GCN and NFC-NFC achieved state-of-the-art performance across all the datasets. This suggests that applying the NFC layer to work on the most related and fixed-size neighbors can be beneficial for learning node representation. NFC-GCN can aggregate more neighborhood information, and it may be more suitable for sparse graph (NFC-GCN gets better result on Citeseer.) NFC-NFC can further alleviate the over-smoothing problem and may be more suitable for dense graph. Considering efficiency, NFC-GCN is faster. For Cora, Citeseer and PubMed, times for each training epoch are 9, 9.4, and 14 seconds respectively, while they are around 18, 18 and 28 seconds in NFC-NFC. We can flexibly combine NFC layer and GCN layer, depending on the requirement of downstream tasks.
In addition, our method achieves better performance in fewer training epochs (less than 100 epoch) on all the datasets. However, we should note that the training time per epoch for our method is more than GCN, GAT, Fast-GCN, possibly due to the larger number of parameters in our model. In future, we can investigate ways to improve the per-epoch computational efficiency.
- **Accuracy, loss over training epochs.** Figures 6(a) and 6(b) show how the training accuracy, training loss change with respect to the number of training epochs. We do not use early stopping in our model for a better comparison with GCN, GAT and LGCL. Our method achieves a good performance in a few training epochs, while GCN, GAT and LGCL need more than a hundred training epochs. Moreover, training accuracy/loss of NFC-GCN change in a more stable way with the same optimization parameters on the same training data as GCN, GAT and LGCL. This confirms that the first-level node representation learned from the node-feature convolution improves the subsequent classification tasks.

On the whole, Table 2, Figs. 6(a) and 6(b) show that our method has a better performance on both node classification accuracy and convergence.

4.3. NFC Aggregation Study

We further present studies of three different aggregation methods: GCN aggregation, GAT aggregation, and NFC aggregation. And we also illustrate the effectiveness of CNNs on the ordered node features.

- **Effectiveness of NFC aggregation.** To show the effectiveness of NFC (our key contribution), we compare GCN and GAT aggregation methods with NFC using a fixed-size set of neighbors.[9] Then we feed the aggregated representation of each node to a classifier directly without adding additional GCN layers. We carry out this experiment over all datasets and choose 5 neighbors for each aggregation methods. The results are summarized in Table 3. Our method increases the testing accuracy greatly over GCN/GAT, demonstrating that NFC-based aggregation can learn a more effective node representation for subsequent tasks. It should also be emphasized that only the NFC can achieve competitive performance without additional GCN layers.
Besides the quantitative evaluation, we also investigate the effectiveness of different aggregation methods qualitatively. We provide t-SNE [41] visualizations to map the embeddings obtained from GCN, GAT and NFC aggregation on the Cora dataset in 2D space. In Fig. 7, all the embeddings exhibits discernible clustering in the projected 2D space. The GAT visualization is poorer than the GCN visualization, which is consistent with the results in Table 3. NFC aggregation obtains the best visualization with nodes clustered into the most compact clusters.
- **NFC on node-feature maps.** A further evaluation is conducted to illustrate how NFC works on node-feature maps. We feed the central node's feature vector to the classifier directly (Cen), which is treated as a baseline. For comparison, we first perform a convolutional operation only on the given node's feature vector (one channel) and feed the new representation to the classifier (Conv(C)). This is used to illustrate how the convolution works on the node feature dimension. Next, we perform a NFC operation on the feature map comprised of the given node and five neighbors (six channels), and feed the new representation to the classifier (Conv(CN)). The experimental results are shown in Table 4. The main finding is that the NFC not only operates effectively on the node feature dimension, but also extracts

[9] LGCL uses GCN aggregation over the raw node features while its first layer is the same as GCN.

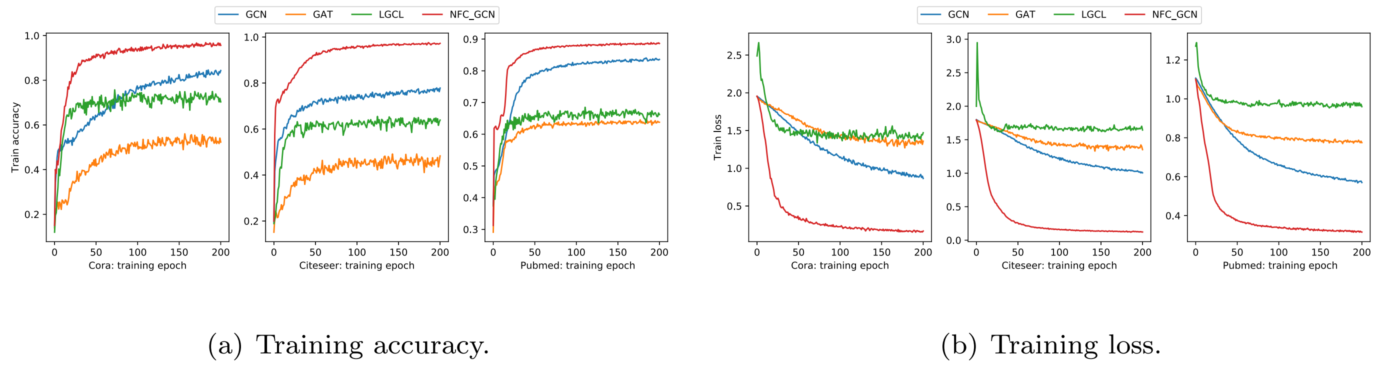


Fig. 6. Comparison of training accuracy and loss with respect to the training epochs.

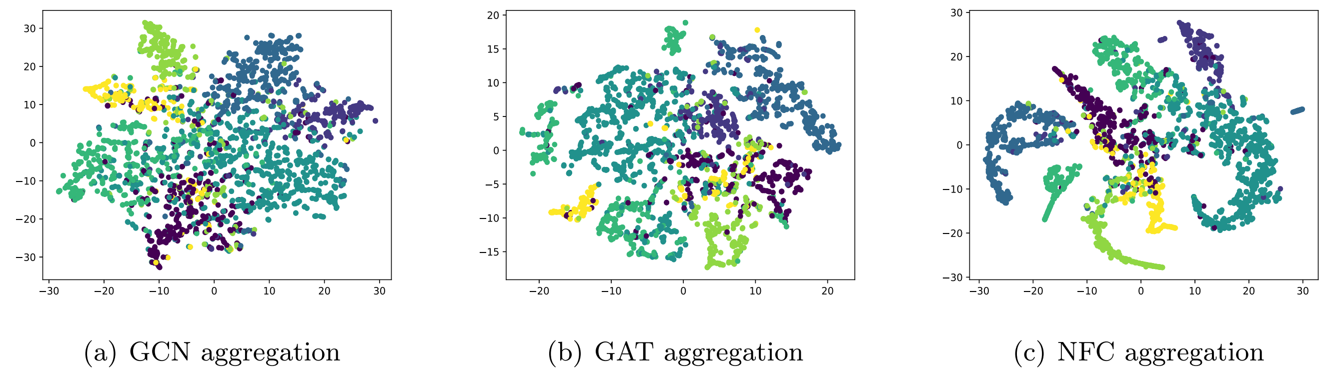


Fig. 7. Visualization of the embeddings on the Cora dataset. We map the embeddings learned from GCN, GAT and NFC aggregation to the 2-D space using t-SNE. Node colors denote classes.

Table 4
The effectiveness of NFC aggregation. Cen: central node (without convolution operation); Cov(C): central node with convolution operation; Cov(CN): node-feature map (containing central node and neighbors' features) with convolution operation.

Dataset	Cen	Cov(C)	Cov(CN)
Cora	54.7%	72.9%	86.0%
Citeseer	69.9%	73.1%	78.9%
PubMed	80.9%	85.7%	89.7%

Table 5
Node degree statistics.

Dataset	Highest	Lowest	Average	Median
Cora	168	1	4.9	4
Citeseer	99	1	3.7	3
PubMed	171	1	5.5	3

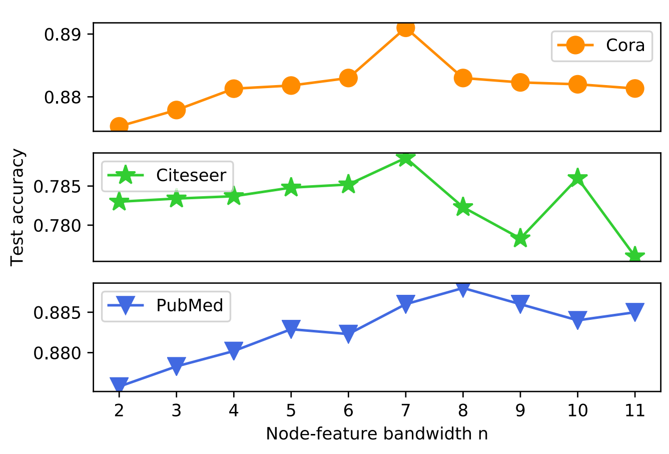
more useful information from different channels in the node-feature map.

This study shows that the advantage of NFC (aggregation assigning different weights to different features for different neighbors), as shown in Table 3 and Fig. 7. Besides, applying the convolution operation on the ordered features and including neighbors information can both benefit to the central node's representation learning, as shown in Table 4.

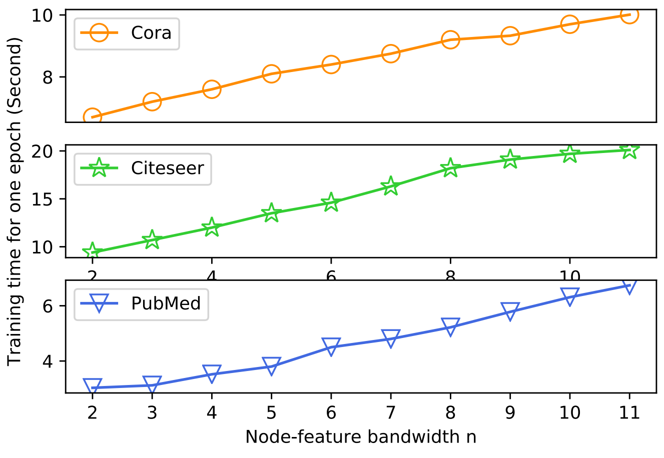
4.4. Parameter Sensitivity

Finally, we examine the parameter sensitivity of our model, including the node bandwidth, model depth and training set size.

- **Node bandwidth.** We first study the effect of varying the node bandwidth n in the node-feature convolution process. Table 5 shows the distribution of node degrees over the three datasets. The node degree varies from one to 171, which illustrates that there is a need to select the neighbors for a given node. The average node degree is three or four, and we vary n from two to 11. The results in Fig. 8(a) show consistent improvement in accuracy with increasing n from two to seven. A higher n implies more feature diversity, and this can be especially helpful for the representation learning of nodes with sparse connectivity and features. However, for n greater than seven, performance drops. A possible explanation is that aggregating too many neighbors has a negative influence (the central node's own information being washed away) on the given node's representation learning. Note that the computation time per training epoch also increases with n, therefore, there is a trade-off between the classification accuracy and computational time when choosing n.
- **Model depth.** We study the influence of model depth (number of GCN layers) on classification performance in the data splits in [3]. We change the GCN layers from one to five and the results are summarised in Fig. 9. Our method is less sensitive to the number of hidden layers. This indicates that the NFC-GCN representation is more robust to model depth. It performs well even when a higher-order neighborhood is considered. Note that the best test accuracy for our method is not better than GCN on PubMed. One possible reason is that this data splits in [3] have only 60 labeled training nodes that is too small for NFC-GCN. Another possible reason is that we did not tune NFC-GCN hyperparameters (set as in Section 4.1) for different layers. With further tuning and optimisation, NFC-GCN has the potential to get better results in classification accuracy for deeper models.



(a) Test accuracy.



(b) Time per epoch.

Fig. 8. Effect of node bandwidth n on accuracy and time per epoch.

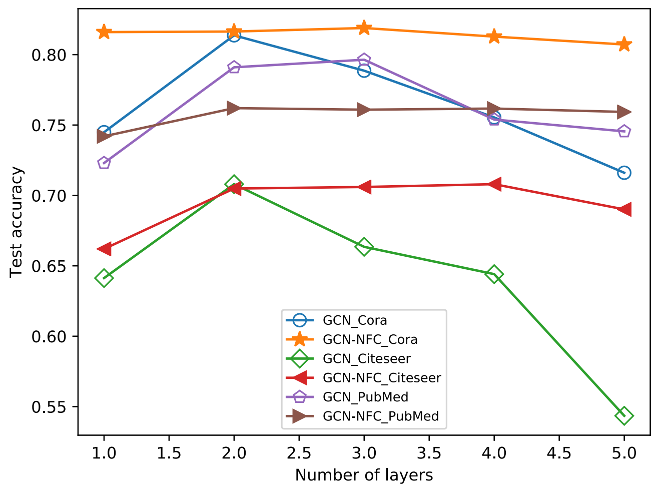


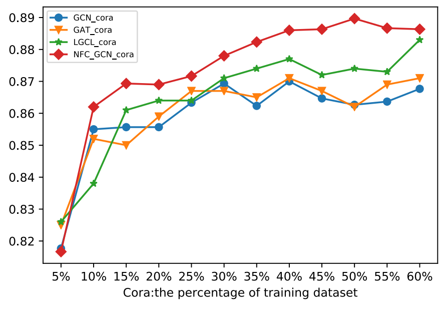
Fig. 9. Performance comparison on deeper models. On the Cora, Citeseer and PubMed datasets, we employ the same experimental setups and increase layers of GCN and NFC-GCN to up to five. GCN-NFC has a better overal performance for deeper models and its test accuracy is more steady than GCN when we increase the number of layers.

- **Training set size.** Finally, we compared our method against GCN, GAT and LGCL with 5%, 10%,...,60% training size on Cora, Citeseer and PubMed. The test and validation sets as well as the hyperparameters are the same as described in Section 4.1. The results are shown in Fig. 10. On the whole, NFC-GCN outperforms other methods with a large margin, especially when the training size is large. However, the superiority decreases when the training set size becomes smaller. Compared with GCN, NFC-GCN has more parameters to learn and thus more training data, which is typical for deep learning models.

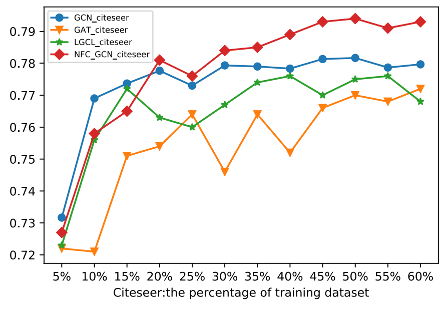
4.5. Discussion

NFC-GCN embodies the ideas from GCN and its extensions such as sampling-based methods and GAT. NFC-GCN applies convolution layer on a fixed-size node-feature map to assign different weights to different features in different neighbors. In the following, we discuss the limitations and new opportunities for this new architecture.

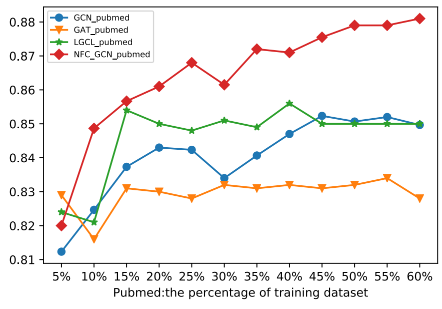
- **Limitations.** The main limitation is that NFC-GCN requires more training samples and the computation cost for each epoch is higher than GCN. Nonetheless, deep learning models are known to work well on larger datasets and be computationally expensive. One future work will aim to improve the performance of our method on the small training datasets and also the computational efficiency.
- **Representation learning ability.** When we only use one NFC layer to learn nodes' new representations and feed them directly to a classifier layer, we can obtain a competitive perfor-



(a) Cora



(b) Citeseer



(c) PubMed

Fig. 10. Effect of training set size. NFC-GCN outperforms others on the whole and it gets more significant improvement with a larger training set.

mance compared to GCN and its extensions. One potential future direction is to get rid of the GCN layer totally.

- **New architectures and deeper models.** The NFC layer can be used in conjunction with any of the competing methods (not only GCN). In particular, NFC-GCN allows for a deeper model, and has the potential to get better classification accuracy with well-tuned hyperparameters. Furthermore, deeper models can enable other powerful machine learning techniques to be better applied to graphs, such as transfer learning.

5. Conclusion

In this paper, we proposed a novel model: Node-Feature Convolution for Graph Convolutional Network (NFC-GCN). We constructed a new node-feature convolutional (NFC) layer to work on a fixed-size feature map that contains features from selected neighbors. NFC can both alleviate neighborhood explosion problem and assign different weights to different features from different neighbors in an *end-to-end* fashion, addressing the main limitations of existing GCN models. Experimental results showed that the proposed NFC-GCN (NFC-NFC) outperforms current competing GCNs on three benchmark datasets in node classification. The main limitation of NFC is that its computational cost increases with the number and size of filters. We also demonstrated that NFC-GCN is more robust in deeper architectures compared to existing GCNs and has the potential to get better results. In addition, the NFC layer can be a plug-in module and integrated with other GCN extensions, e.g., FastGCN [25], jumping knowledge networks [26], GMWW [42] and MixHop [43], enabling these methods to have a feature-level attention.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported in part by the China Scholarship Council (CSC) under Grant 201706080010 and in part by the UK Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/R014507/1.

References

[1] Y. Luo, T. Guan, J. Yu, P. Liu, Y. Yang, Every node counts: Self-ensembling graph convolutional networks for semi-supervised learning, Pattern Recognition 106 (2020) 107451.
[2] Y. Yang, Y. Qi, Image super-resolution via channel attention and spatial graph convolutional network, Pattern Recognition 112 (2021) 107798.
[3] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: Proceedings of the 6th International Conference on Learning Representations, 2017, pp. 1–15.
[4] M. Schlichtkrull, T.N. Kipf, P. Bloem, R. van den Berg, I. Titov, M. Welling, Modeling relational data with graph convolutional networks, in: European Semantic Web Conference, 2018, pp. 593–607.
[5] F. Hu, Y. Zhu, S. Wu, W. Huang, L. Wang, T. Tan, Graphair: Graph representation learning with neighborhood aggregation and interaction, Pattern Recognition 112 (2021) 107745.
[6] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al., Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[7] A.K. McCallum, K. Nigam, J. Rennie, K. Seymore, Automating the construction of internet portals with machine learning, Information Retrieval 3 (2) (2000) 127–163.
[8] W.L. Hamilton, R. Ying, J. Leskovec, Representation learning on graphs: Methods and applications, IEEE Data Eng. Bull. 40 (2017) 52–74.
[9] S.T. Roweis, L.K. Saul, Nonlinear dimensionality reduction by locally linear embedding, science 290 (5500) (2000) 2323–2326.
[10] M. Belkin, P. Niyogi, Laplacian eigenmaps and spectral techniques for embedding and clustering, in: Proceedings of the 14th International Conference on Neural Information Processing Systems, 2001, pp. 585–591.
[11] M. Ou, P. Cui, J. Pei, Z. Zhang, W. Zhu, Asymmetric transitivity preserving graph embedding, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 1105–1114.
[12] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge discovery and data mining, 2014, pp. 701–710.
[13] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge discovery and data mining, 2016, pp. 855–864.
[14] B. Perozzi, V. Kulkarni, H. Chen, S. Skiena, Don't walk, skip!: Online learning of multi-scale network embeddings, in: Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017, 2017, pp. 258–265.
[15] Y. Shi, M. Lei, H. Yang, L. Niu, Diffusion network embedding, Pattern Recognition 88 (2019) 518–531.
[16] M. Gori, G. Monfardini, F. Scarselli, A new model for learning in graph domains, Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. 2 (2005) 729–734.
[17] Y. Li, D. Tarlow, M. Brockschmidt, R.S. Zemel, Gated graph sequence neural networks, in: Proceedings of the 5th International Conference on Learning Representations, 2016, pp. 1–15.
[18] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: Proceedings of the 28th Advances in Neural Information Processing Systems, 2015, pp. 2224–2232.
[19] M. Niepert, M. Ahmed, K. Kutzkov, Learning convolutional neural networks for graphs, in: Proceedings of the 33rd International Conference on Machine Learning, 2016, pp. 2014–2023.
[20] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1225–1234.
[21] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, in: Proceddings of the 3rd International Conference on Learning Representations, 2014, pp. 1–15.
[22] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: Proceedings of the 29th Advances in Neural Information Processing Systems, 2016, pp. 3844–3852.
[23] Q. Li, Z. Han, X. Wu, Deeper insights into graph convolutional networks for semi-supervised learning, in: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018, pp. 3538–3545.
[24] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Proceedings of the 30th Advances in Neural Information Processing Systems, 2017, pp. 1025–1035.
[25] J. Chen, T. Ma, C. Xiao, Fastgcn: Fast learning with graph convolutional networks via importance sampling, in: Proceedings of the 7th International Conference on Learning Representations, 2018, pp. 1–15.
[26] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, S. Jegelka, Representation learning on graphs with jumping knowledge networks, in: Proceedings of the 35th International Conference on Machine Learning, 2018, pp. 5453–5462.
[27] Y. Rong, W. Huang, T. Xu, J. Huang, Dropedge: Towards deep graph convolutional networks on node classification, in: International Conference on Learning Representations, 2020, pp. 1–17.
[28] A. Hasanzadeh, E. Hajiramezanali, S. Boluki, M. Zhou, N. Duffield, K. Narayanan, X. Qian, Bayesian graph neural networks with adaptive connection sampling, in: International Conference on Machine Learning, 2020, pp. 4094–4104.
[29] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, Y. Qi, Geniepath: Graph neural networks with adaptive receptive paths, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019, pp. 4424–4431.
[30] J. Ma, P. Cui, K. Kuang, X. Wang, W. Zhu, Disentangled graph convolutional networks, in: International Conference on Machine Learning, 2019, pp. 4212–4221.
[31] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: Proceedings of the 4th International Conference on Learning Representations, 2015, pp. 1–15.
[32] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, in: Proceedings of the 7th International Conference on Learning Representations, 2018, pp. 1–12.
[33] L. Yang, F. Wu, Y. Wang, J. Gu, Y. Guo, Masked graph convolutional network, in: IJCAI, 2019, pp. 4070–4077.
[34] L. Zhang, H. Lu, A feature-importance-aware and robust aggregator for gcn, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 2020, pp. 1813–1822.
[35] E. Perez, F. Strub, H. De Vries, V. Dumoulin, A. Courville, Film: Visual reasoning with a general conditioning layer, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2018, p. 4424==4431.
[36] H. Gao, Z. Wang, S. Ji, Large-scale learnable graph convolutional networks, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, 2018, pp. 1416–1424.
[37] S. Hoory, N. Linial, A. Wigderson, Expander graphs and their applications, Bulletin of the American Mathematical Society 43 (4) (2006) 439–561.
[38] Y. Kim, Convolutional neural networks for sentence classification, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 2014, pp. 1746–1751.
[39] A. Ahmed, N. Shervashidze, S.M. Narayanamurthy, V. Josifovski, A.J. Smola, Distributed large-scale natural graph factorization, in: Proceedings of the 22th International Conference on World Wide Web, 2013, pp. 37–48.

[40] P. Goyal, E. Ferrara, Graph embedding techniques, applications, and performance: A survey, Knowledge-Based Systems 151 (2018) 78–94.

[41] L.v.d. Maaten, G. Hinton, Visualizing data using t-sne, Journal of machine learning research 9 (Nov) (2008) 2579–2605.

[42] M. Qu, Y. Bengio, J. Tang, Gmnn: Graph markov neural networks, in: Proceedings of the 36th International Conference on Machine Learning, 2019, pp. 5241–5250.

[43] S. Abu-El-Haija, B. Perozzi, A. Kapoor, H. Harutyunyan, N. Alipourfard, K. Lerman, G.V. Steeg, A. Galstyan, Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing, in: Proceedings of the 36th International Conference on Machine Learning, 2019, pp. 21–29.

Li Zhang received the B.Sc. degree from Northeast University, China in 2014, and M.Sc. degree with distinction from Department of Information Science and Engineering, Northeast University, China in 2017. She is now ah.D. student in the Department of Computer Science, University of Sheffield, UK. Her research interests are graph representation learning, graph neural networks, and knowledge graph.

Heda Song received the B.Sc. degree from Northeast University, China in 2013, and M.Sc. degree with distinction from Department of Information Science and Engineering, Northeast University, China in 2017. He is now ah.D. student in the Department of Computer Science, University of Nottingham, UK. His research interests are meta-learning and few-shot learning.

Dr. Nikolaos Aletras is a Senior Lecturer in Natural Language Processing (NLP) in the Computer Science Department, University of Sheffield, co-affiliated with the Machine Learning (ML) group. His research interests include social media analysis, NLP in the legal domain and topic modelling. His work has received wide media coverage including the BBC, the Guardian, the Wall Street Journal, the Washington Post and the New Scientist.

Haiping Lu received the B.Eng. and M.Eng. degrees in electrical and electronics engineering from Nanyang Technological University, Singapore, in 2001 and 2004, respectively, and the Ph.D. degree in electrical and computer engineering from University of Toronto, Canada, in 2008. Currently, he is a Senior Lecturer in Machine Learning at the Department of Computer Science, University of Sheffield, UK. He received prestigious awards from NIHR, Wellcome Trust, Amazon, AAAI, IEEE, and the Alan Turing Institute.