Proceedings Paper:
Zhang, L. and Lu, H. (2020) A feature-importance-aware and robust aggregator for GCN. In: CIKM '20: Proceedings of the 29th ACM International Conference on Information & Knowledge Management. CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, 19-23 Oct 2020, Online conference. Association for Computing Machinery (ACM), pp. 1813-1822. ISBN: 9781450368599.

https://doi.org/10.1145/3340531.3411983

A Feature-Importance-Aware and Robust Aggregator for GCN

Li Zhang
Department of Computer Science,
University of Sheffield
Sheffield, UK
lzhang72@sheffield.ac.uk

Haiping Lu
Department of Computer Science,
University of Sheffield
Sheffield, UK
h.lu@sheffield.ac.uk

ABSTRACT

Neighborhood aggregation is a key step in Graph Convolutional Networks (GCNs) for graph representation learning. Two commonly used aggregators, sum and mean, are designed with the homophily assumption that connected nodes are likely to share the same label. However, real-world graphs are noisy and adjacent nodes do not necessarily imply similarity. *Learnable* aggregators are proposed in Graph Attention Network (GAT) and Learnable Graph Convolutional Layer (LGCL). However, GAT considers node importance but not the importance of different features. The convolution aggregator in LGCL considers feature importance but it can not directly operate on graphs due to the irregular connectivity and lack of orderliness. In this paper, we firstly unify the current learnable aggregators in a framework: Learnable Aggregator for GCN (LA-GCN) by introducing a shared auxiliary model that provides a customized schema in neighborhood aggregation. Under this framework, we propose a new model called LA-GCN$_{\text{Mask}}$ consisting of a new aggregator function, *mask aggregator*. The auxiliary model learns a specific mask for each neighbor of a given node, allowing both node-level and feature-level attention. This mechanism learns to assign different importance to both nodes and features for prediction, which provides interpretable explanations for prediction and increases the model robustness. Experiments on seven graphs for node classification and graph classification tasks show that LA-GCN$_{\text{Mask}}$ outperforms the state-of-the-art methods. Moreover, our aggregator can identify both the important nodes and node features simultaneously, which provides a quantified understanding of the relationship between input nodes and the prediction. We further conduct experiments on noisy graphs to evaluate the robustness of our model. Experiments show that LA-GCN$_{\text{Mask}}$ consistently outperforms the state-of-the-art methods, with up to 15% improvements in terms of accuracy compared to the second best.

KEYWORDS

Graph convolutional networks, Mask aggregator, Feature-level attention

CIKM'20, October 19-23, 2020, GALWAY, IRELAND

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00

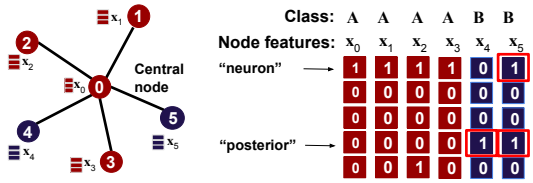


Figure 1: A six-node subgraph from the Cora [25]. Each node corresponds to a machine learning paper, with a bag-of-words feature vector $\mathbf{x}_i$ ($i = 0, 1, 2, ..., 5$). Nodes 0–3 belong to Class A (*Neural Networks*), and nodes 4–5 belong to Class B (*Probabilistic Methods*). Individual features in $\mathbf{x}_i$ are not equally important for representing the central node 0.

ACM Reference Format:
Li Zhang and Haiping Lu. 2020. A Feature-Importance-Aware and Robust Aggregator for GCN . In *Proceedings of CIKM'20: ACM International Conference on Information and Knowledge Management (CIKM'20).* ACM, GALWAY, IRELAND, 10 pages.

1 INTRODUCTION

Graph Convolutional Network (GCN) is a powerful tool for machine learning on graphs [21]. It learns a new representation of a given node by aggregating information from neighbors, which naturally combines graph structure and node features in the learning process. The aggregator in GCN aggregates the feature vectors of its neighbors with fixed weights that are inversely proportional to the central and neighbors' node degrees. Later, some other aggregators were proposed: mean, pooling, LSTM [14] and sum aggregator [37]. In supervised learning, these aggregation strategies are designed with the assumption that connected nodes in a graph are likely to share the same label, i.e. homophily, which has been widely used in graph neural networks (GNNs) [6, 9] and graph-Laplacian regularization methods including label propagation, manifold regularization and deep semi-supervised embedding [5, 36, 45].

Edges in real graphs are often noisy or defined via user-defined thresholds [22]. Therefore, the edges may not clearly correspond to label agreement uniformly across the graph [31]. In Cora, Citeseer and PubMed, 19%, 26%, and 20% of the edges, respectively, connect with nodes from different classes. Besides, each feature within a neighbor feature vector may play a different role for the central node's representation learning [15, 24, 44]. 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. In such cases, mean, pooling, or sum aggregators are not optimal choices in learning useful representations from the noisy neighborhood of the central node. It is necessary to filter both node and feature information before aggregation, especially for graphs with node features.

Thera are also *learnable* aggregators proposed to automatically filter the neighborhood information. Graph Attention Network (GAT) borrows the idea of attention mechanisms [3, 32] to learn to assign different weights to different neighbors in aggregation [33]. However, all individual features in a feature vector are treated equally. Learnable Graph Convolutional Layer (LGCL) performs convolution operation in the aggregation process, which can assign different weights to different features [10]. But, using regular convolution operation on graphs requires the number of neighboring nodes for each node remains the same, and these neighboring nodes are ordered. LGCL transforms the graph into grid-like structure by selecting the top-d values in each feature dimension from all the neighbors. The covolutional operator mixes or reorganizes neighborhood information, which makes it difficult to interpret the learned representation because we can not distinguish which node and feature have a salient influence on the prediction result.

This paper aims to design a more adaptive and interpretable aggregator satisfying the following five **D**esirables.

- **D1 & D2:** To deal with graph structures, the aggregator should **1)** be able to handle variable-sized neighbors [26, 33], and **2)** be invariant to the ordering of neighbors [26]. Unlike images and sentences, graphs usually have no regular connectivity and neighboring nodes have no natural ordering.
- **D3 & D4:** To enhance the discriminating power, the aggregator should **3)** be discriminative to node-level and feature-level neighborhood information [10, 33], **4)** be able to discriminate graph structures in the embedding space [37]. Real-world graphs are noisy and the aggregator should automatically identify the important information from the neighborhood.
- **D5:** For practical applications where interpretability is needed, the aggregator should **5)** be able to explain learned representations in relation to the prediction and robust to structure and feature noise. Real-world data are often noisy so aggregating information from noisy graph structures and node features can cause significant difficulties in accurate prediction and useful interpretation [40]. An explainable and robust aggregator can increase the trustworthiness and real-world performance.

To this end, we unify current learnable aggregators in a general framework: learnable aggregator for GCN (LA-GCN). This framework introduces an auxiliary model that can extract customized high-level knowledge from a given node's neighbors to guide the aggregation process. Under this framework, we propose a new model called LA-GCN$_{\text{Mask}}$ consisting of a new aggregator function, *mask aggregator*, and a carefully designed auxiliary model shared by all nodes in a graph that satisfies **D1** and **D2**. Firstly, a given node and its neighbors are fed into the auxiliary model to get a specific mask for each neighbor. Then the mask aggregator performs a Hadamard product between the feature vector of each neighbor and its corresponding mask before aggregation. In this way, the mask aggregator can learn to assign different weights to different features in different neighbors, which leads to better discriminativeness to node and feature information (**D3**) and also enables better interpretation of the learned representation (**D5**). The proposed aggregator sums up all the filtered neighbors of the central node as its learned representation, meeting **D4**.

We evaluate LA-GCN$_{\text{Mask}}$ on three popular citation graphs and one large social graph for node classification, and three bioinformatics graphs for graph classification. Our results confirm that node-level and feature-level attention of neighborhood information in aggregation can lead to significant performance gains. In addition, we visualize the learned mask to show that it can identify important features and nodes, which provides an interpretable explanations for prediction. Finally, we study the robustness of our model on graphs with structure and node feature noise.

In both structure-noisy and feature-noisy graphs, LA-GCN$_{\text{Mask}}$ consistently outperforms popular baselines (GCN, GAT and LGCL), with up to 9.82% (Cora) and 15.05% (Citeseer) improvement on structure-noisy graphs and 10.67% (Cora) and 3.60% (Citeseer) improvement on node feature-noisy graphs, in terms of node classification accuracy.

In summary, our contributions are threefold:

- We unify several existing methods in a LA-GCN framework and propose a new mask aggregator, a new attention mechanism allowing both node-level and feature-level attention.
- We comprehensively evaluate the superiority of the proposed LA-GCN$_{\text{Mask}}$ on seven graphs with different sizes and types for both node and graph classification tasks.
- We demonstrate that the proposed model can provide interpretable explanation for the prediction, also study the robustness of our model on both structure and node feature noisy graphs.

2 RELATED WORK

Graph Neural Networks (GNNs) were introduced in Gori et al. [12] and Scarselli et al. [29] as a generalization of recursive neural networks that can directly deal with a more general class of graphs, e.g. cyclic, directed and undirected graphs. Node representation learning via GNNs consists of two key steps: neighborhood aggregation and feature transformation.

In neighborhood aggregation step, neighbors of a given node are aggregated (with or without the central node) to get the aggregation result. In feature transformation stage, the central node first combines with the aggregation result to get a combined vector, which is followed by a linear mapping or a multi-layer perceptrons (MLPs) [17, 18] to get the new representation of a given node. This schema iteratively updates the representation of a node by aggregating representations of its neighbors and transformation, which can also be treated as a general neural message-passing process [11] or relational inductive bias model [4].

Later, Hamilton et al. [14] proposed mean, pooling and LSTM aggregators in GraphSAGE. Mean aggregator simply takes an elementwise mean of a given node's neighbors, pooling aggregator applies an elementwise max-pooling on the neighbors and LSTM

aggregator applies LSTM [16] to a random permutation of the neighbors. Note that the neighborhood in GraphSAGE can come from a different number of hops, or search depth, away from a given node. After getting the aggregated result, GraphSAGE concatenates it with the central node and feeds the concatenated feature vector into a fully connected layer with nonlinear activation function in the feature transformation stage. However, mean, pooLing aggregators are not injective functions and fail to distinguish different graph structures. Xu et al. [37] generalizes the WL algorithm [35], a powerful algorithm known to distinguish graph structures, and proposes sum aggregator in graph isomorphism network (GIN). Instead of summing the labels [35], GIN sums a given node with its neighbors in the neighborhood aggregation step.

The mentioned aggregators are all predefined heuristics that connected nodes tend to be similar, but it is debatable, because real-world graphs are noisy or have edges that do not correspond to label agreement. Some strategies have been proposed to make the aggregator learnable. In GAT [33], the aggregator aggregates the neighbors corresponding to the learned attention coefficients that indicate the importance between two nodes. However, all the features are still treated equally within a feature vector, for each feature shares the same weight in the aggregation. LGCL [10] applies convolution operation on the reconstructed neighbors' feature map (choosing the top-d values in each feature dimension from all the neighbors). The reconstruction can achieve the transformation from graphs to grid-like data, but it breaks the original correspondence between node features. Besides, the filter in the convolution process can only work on fix-sized feature maps, and using convolution in the aggregation process is not suitable to learn from unordered neighbors with variable-size.

Recalling the desirable characteristics of an aggregator in Sec. 1, we find that above mentioned aggregators can not satisfy all these desiderata.

3 LA-GCN

We unify current learnable aggregator in one framework: LA-GCN, that utilizes an auxiliary model to guide the aggregation process, which enables the aggregator to satisfy all the desirables. In this section, we first describe the framework, followed by theoretical motivation for our model: LA-GCN$_{\text{Mask}}$. Then, we compare our model with prominent GCN based methods.

3.1 Notation and Problem Definition

An undirected graph with N nodes can be represented as $\mathcal{G}=(\mathcal{V}, \mathcal{E}, \mathbf{X})$, where node $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$ $(i, j = 1, ..., N)$, 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. The neighborhood $\mathcal{N}_i = \{v_i \in \mathcal{V} \mid (v_i, v_j) \in \mathcal{E}\}$ is the set of adjacent nodes of v_i. A hidden representation of node v_i learned by the k-th layer of a model is denoted by $\mathbf{h}_i^{(k)} \in \mathbb{R}^{d_k}$ $(d_k < D)$ and we initialize $\mathbf{h}_i^{(0)} = \mathbf{X}_i$.

Predictions on graphs are made by first embedding nodes $\mathbf{X}$ into a low-dimensional space $\mathbf{H}$, which is used for down-stream tasks, such as node classification, graph classification.

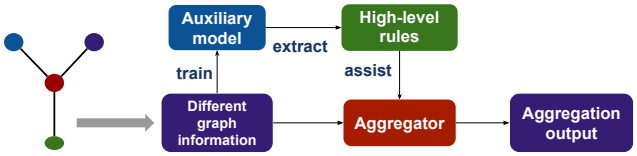


Figure 2: LA-GCN framework. The key idea is to utilize an auxiliary model to assist the aggregator to deal with different neighborhood information in a customized schema.

3.2 Framework

A key challenge is how to design an efficient aggregator that suits for each node in a graph since each node has different neighbors no matter the numbers or categories, and satisfies the mentioned desirables. Intuitively, this requires each node with a specific model, which is quite impossible, for real-world graphs can contain millions or billions nodes [13].

Inspired by the weight sharing property of CNNs [23] and attention mechanism [32], we use a shared auxiliary model to extract high-level knowledge or rules from the given graph information, and the learned rules are used to assist the aggregation process as shown in Fig 2. It is a flexible and general framework that can unify mentioned GAT [33] and LGCL [10], and we give a detail comparison in Section 3.4.

3.3 Methodology

Under this framework, we carefully design our auxiliary model and propose a new aggregation function: mask aggregator. Our ultimate goal is to design an aggregator that can satisfy all the disirables. We start from the theoretical study of the aggregator function, which enables the formulation of our aggregator that simultaneously satisfies our desirables.

3.3.1 ***Theoretical Studies of aggregator.*** In this subsection, we mainly study the aggregator function from graph datasets's perspective and the aggregator's expressive capacity.

In generic graphs, the numbers of neighboring nodes usually differ for different nodes in a graph, and there is no order information based on which we can order them to ensure the output is deterministic. These special characters of graph datasets require the aggregator should be a permutation-invariant function that can deal with variable-sized and unordered neighbors (**D1, D2**).

Permutation invariant study. Permutation invariance is an important property for aggregator since there is no natural order in most real graphs. The neighborhood aggregation scheme iteratively updates the representation of a node by aggregating representations of it neighbors. To mathematically formalize the above insight, the aggregation process can be generically written as follows:

$$\mathbf{s}_i^{(k-1)} = f_{ag}^{(k)}(\mathbf{h}_j^{(k-1)}, j \in \mathcal{N}_i), \tag{1}$$

where $f_{ag}^{(k)}$ is the predefined aggregation function (aggregator) in the k-th layer of a model.

The aggregator $f_{ag}^{(k)}$ can be seen as a function over the full multiset of node neighbors. Following [37], a multiset is a generalized concept of a set [41] that the same element can appear multiple

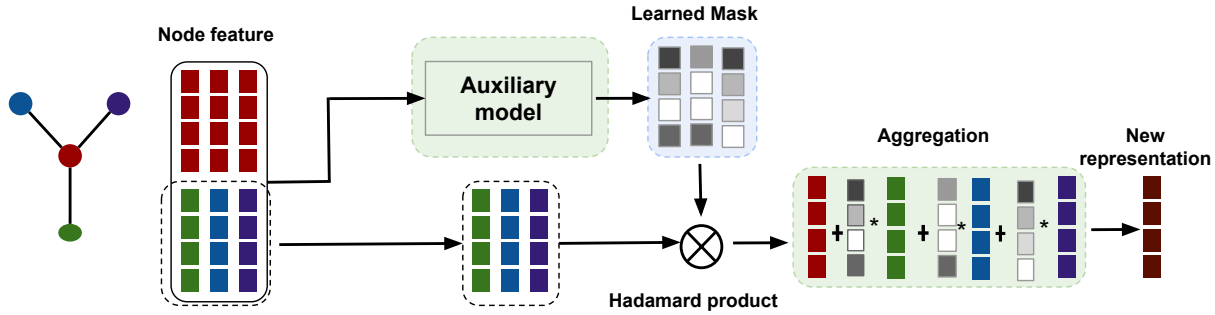


Figure 3: LA-GCN$_{\text{Mask}}$ consists of three steps: 1) train an auxiliary model with a given node and the feature vectors of its neighbors; 2) generate the mask for each neighbor from the auxiliary model; 3) aggregate the neighbors (after multiplying the corresponding mask) to get a new representation of the central node.

times since different nodes can have identical feature vector. Recall that one of the desiderata is that the aggregator $f_{ag}^{(k)}$ should be a mutiset permutation invariant function. Following [41], a permutation invariant function on multiset can be defined as:

Definition 1. *A function f is permutation-invariant if*

$$f(\{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_{|\mathcal{N}_i|}\}) = f(\{\mathbf{h}_{\pi_{(1)}}, \mathbf{h}_{\pi_{(2)}}, ..., \mathbf{h}_{\pi_{(|\mathcal{N}_i|)}}\}) \quad (2)$$

for any permutation π and $|\mathcal{N}_i|$ is the length of the sequence.

We will use $\Pi_{|\mathcal{N}_i|}$ to represent the mutiset of all permutations of the integers 1 to $|\mathcal{N}_i|$ and $\mathbf{h}_\pi$, $\pi \in \Pi_{|\mathcal{N}_i|}$, represents a reordering of the mutliset according to π. The following theorem in [41] shows the relation between set and permutation invariant function.

Theorem 1. *A function operating on a multiset $\{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_{(|\mathcal{N}_i|)}\}$ having elements from a countable universe, is a valid set function. It is invariant to the permutation of instances in the multiset, if it can be decomposed in the form $\rho(\sum_{\pi \in \Pi_{|\mathcal{N}_i|}} \phi(\mathbf{h}_\pi))$, for suitable transformations ϕ and ρ.*

The structure of permutation invariant function in Theorem 1 hints a general strategy for inference over mutiset. In other words, the key is to add up all representations and then apply nonlinear transformation.

Sum, mean, pooling aggregators and aggregators in GCN and GAT can be formulated as this format. GCN and GAT add up all neighborhood neighbors with fixed weights or learnable weights, as shown in Eq. 3 and Eq. 4, respectively.

$$\mathbf{s}_i^{(k-1)} = f_{agg}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} / \sqrt{d_i d_j}. \quad (3)$$

where d_i and d_j are the node degrees of node v_i and node v_i respectively.

$$\mathbf{s}_i^{(k-1)} = f_{aga}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{h}_j^{(k-1)}, \quad (4)$$

where α_{ij} is a learnable attention coefficient that indicates the importance of v_j to v_i. But convolution aggregator in LGCL is not permutation-invariant function, for the output of the aggregator will change if the inputs are reordered, and it can not deal with variable-sized data directly.

Algorithm 1 LA-GCN$_{\text{Mask}}$ (one iteration)

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ with N nodes;
Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$;
Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$;
Auxiliary model f ;
Output: Vector representation $\mathbf{h}_i^{(k)}$
for each $v_i \in \mathcal{V}$ **do**
  for $j \in \mathcal{N}_i$ **do**
    $\mathbf{m}_j^{(k-1)} = f\,(\|\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)})$
    $\mathbf{s}_i^{(k-1)} = \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} * \mathbf{m}_j^{(k-1)}$
    $\mathbf{h}_i^{(k)} = \sigma(\mathbf{W}^{(k)}(\mathbf{h}_i^{(k-1)} + \mathbf{s}_i^{(k-1)}))$
  end for
end for

Discriminative power study. From the aggregator's expressive capacity, there are mainly three tasks: One is to learn the interactions between self node and its neighborhood (node-level distinguish); The second one is to learn the interactions between different dimensions of the node features, which will extract useful combinatory features automatically (feature-level distinguish). The final one is to discriminate graph structures (structure-level distinguish) (**D3, D4**).

Sum aggregator are injective function in structure-level, while mean and pooling aggregators are not, which has been proved in [37]. Notice that this property may suit better for graph classification task where graph structure plays a key role. Adding up all neighbors' feature vectors may change the scale of the feature, which may not be good for node classification task. However, they all can not treat the neighborhood information differently in both node-level and feature-level.

The aggregation process in GCN and GAT, as shown in Eq. 3 and Eq. 4, can discriminate the neigborhood information in node-level, however all the features are treated equally within the feature vector $\mathbf{h}_j^{(k)}$, for each feature shares the same weight $(d_i d_j)^{-1/2}$ or α_{ij}. The convolution aggregator in LGCL allows for feature-level attention, but it is not an optimal choice to deal with variable size inputs and unordered graph datasets.

3.3.2 ***Mask Aggregator***. Based on the theoretical study and analysis, we design our aggregator function by extending sum aggregation function . Besides, the expected aggregator could do feature-wise and node-wise modulation of the neighborhood information in the aggregation process, which naturally inspires us to filter the neighborhood information before aggregation and the mask aggregator are shown as following:

$$\mathbf{s}_i^{(k-1)} = f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} * \mathbf{m}_j^{(k-1)}, \tag{5}$$

where $\mathbf{h}_j^{(k-1)} \in \mathbb{R}^{d_{k-1}}$, $\mathbf{m}_j^{(k-1)} \in \mathbb{R}^{d_{k-1}}$ is a specific mask for each neighbor, produced by the auxiliary model. Then we Hadamard product to multiply each neighbor and its corresponding mask before summation (**D3,D4**).

Theorem 2. *$f_{agm}^{(k)}$ is a permutation-invariant function acting on finite but arbitrary length sequences $\mathbf{h}_j^{(k-1)}, j \in \mathcal{N}_i$.*

Proof. Given $H = \left\{\mathbf{h}_1^{(k-1)}, \mathbf{h}_2^{(k-1)}, ..., \mathbf{h}_{(|\mathcal{N}_i|)}^{(k-1)}\right\}$, a finite multiset, and $\mathbf{h}_j^{(k-1)} \in \mathbb{R}^{d_{k-1}}$, our aggregator aims to fuse it into a fixed output $\mathbf{s}_i^{(k-1)} \in \mathbb{R}^{d_{k-1}}$ as follows:

$$\mathbf{s}_i^{(k-1)} = f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} * \mathbf{m}_j^{(k-1)}, \tag{6}$$

where $\mathbf{m}_j^{(k-1)} \in \mathbb{R}^{d_{k-1}}$ is a specific mask for each neighbor, produced by the auxiliary model. We first feed the graph information to an auxiliary model to get a mask $\mathbf{m}_j^{(k-1)}$ for each node $\mathbf{h}_j^{(k-1)}$. For a trained auxiliary model, $\mathbf{m}_j^{(k-1)}$ is a specific and fixed mask (vector) for each neighbor's latent vector $\mathbf{h}_j^{(k-1)}$.

There exists a mapping function $\phi : \mathbb{R}^{d_{k-1}} \rightarrow \mathbb{R}^{d_{k-1}}$ that can formulate $\mathbf{h}_j^{(k-1)} * \mathbf{m}_j^{(k-1)}$ to $\phi(\mathbf{h}_j^{(k-1)})$, and Eq. 6 can be written as:

$$\mathbf{s}_i^{(k-1)} = f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \phi(\mathbf{h}_j^{(k-1)}), \tag{7}$$

and ρ can be seen as $\rho = 1$. Eq. 8 can be seen as a permutation of H, according to [41].

Next, we prove there exist an injective mapping function ϕ, so that $f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)})$ is unique for each finite multiset H.

Since H is countable, each $\mathbf{h}_j^{(k-1)} \in$ H can be mapped to a unique element to prime numbers $p(H)$: $\mathbb{R}^M \rightarrow \mathbb{P}$ by a function $p(\mathbf{h}_j^{(k-1)})$. We can choose $\phi(\mathbf{h}_j^{(k-1)}) = -\log p(\mathbf{h}_j^{(k-1)})$. Therefore,

$$f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \phi(\mathbf{h}_j^{(k-1)}) = \log p(\mathbf{h}_j^{(k-1)}) \tag{8}$$

takes a unique value for each distinct H.

Besides, the dimension d_{k-1} of the latent space should be at least as large as the maximum number of input elements $|\mathcal{N}_i|$, which is both necessary and sufficient for continuous permutation-invariant functions [34].

For the universal approximation theorem [17], any continuous function can be approximated by a neural network, we can use mutlti-layer perceptrons (MLPs) to model and learn ϕ and $\rho = 1$. □

Besides the provement, we state the derivatives with regard to our aggregator. Assume parametrizations W_ϕ for ϕ, we have

$$\partial_{W_\phi}(\sum_{j \in \mathcal{N}_i} \phi(\mathbf{h}_j'^{(k-1)})) = \sum_{j \in \mathcal{N}_i} \partial_{W_\phi} \phi(\mathbf{h}_j'^{(k-1)}),$$

this result shows the ordering is also irrelevant for the optimization process.

Theorem 2 shows that $f_{agm}^{(k)}$ of the multiset is a permutation-invariant function (**D2**). The learned mask can shows which features or neighbors are important, and filter the noisy information, which makes the aggregation results easier to explain and robust (**D5**).

A natural follow-up question is how to get the mask $\mathbf{m}_j^{(k-1)}$. Under our framework, mask is learned from an auxiliary model and we hope the auxiliary model can 1) extract useful and high-level knowledge (e.g., focusing on important nodes and features) from neighborhood information to guide a better aggregation for the central node's representation learning; 2) deal with different size input datasets without reorganization.

Motivated by this, we feed both central node and its neighbors into the auxiliary model. The auxiliary model can be an arbitrary neural network that has no requirement for size or order of the input datasets, e.g., MLP can be applied as the auxiliary model, CNN or RNN can not be (**D1**). Considering the trade-off between performance and efficiency, we apply an MLP with a single hidden layer.

Given node and its neighbors ($\{\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}, j \in \mathcal{N}_i\}$), we feed each node-neighbor pair to an auxiliary model, defined as following:

$$\begin{aligned} \mathbf{m}_j^{(k-1)} &= MLP^{(k)}(\|\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}) \\ &= \sigma(\mathbf{W}_m^{(k)}(\|\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)})), \end{aligned} \tag{9}$$

where σ is the activation function, e.g., sigmoid, RELU, $\mathbf{W}_m^{(k)} \in \mathbb{R}^{2d_{k-1} \times d_{k-1}}$ is the weight matrix and $\|$ denotes column-wise concatenation. The update rule for v_i is

$$\mathbf{h}_i^{(k)} = \sigma(\mathbf{W}^{(k)}(\mathbf{h}_i^{(k-1)} + \mathbf{s}_i^{(k-1)})), \tag{10}$$

where $\mathbf{W}^{(k)} \in \mathbb{R}^{d_k \times d_{k-1}}$ is the learnable weight matrix. After K iterations/layers, the final representation $\mathbf{h}_i^{(K)} \in \mathbb{R}^{d_K}$.

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

$$\mathcal{L} = -\sum_{l \in \mathcal{V}_l} \sum_{f=1}^{F} \mathbf{Y}_{lf} \ln \mathbf{h}_l^{(K)}, \tag{11}$$

where $\mathcal{V}_l$ is the set of node indices that have labels and d_K is the dimension of output features equaling to the number of classes. $\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l| \times F}$ is a label indicator matrix.

For graph classification, adding up all $\mathbf{h}_i^{(K)}$ or more sophisticated graph-level pooling can be applied to get the entire graph's representation.

Table 1: Outline of related work in term of fulfilled (√) and missing (×) desirable characteristics (D3-n means node-level attention and D3-f means feature-level attention in Desirable 3).

Desirable	Mean	Mean_w	Sum	Sum_{lw}	*Conv*	Mask
D1	√	√	√	√	×	√
D2	√	√	√	√	×	√
D3-n	×	√	×	√	√	√
D3-f	×	×	×	×	√	√
D4	×	×	√	√	√	√
D5	×	×	×	×	×	√

3.4 Connections with Existing GCN Extensions

We compare our model with prominent GCN based models and we study all these model from three aspects:

- **Aggregator** Sum [37] and mean [14] are two most commonly seen aggregators. The aggregator in GCN [21] can be seen as a weighted mean aggregator (mean_w), and the weight is $(d_i d_j)^{-1/2}$, where d_i, d_j are the node degree of central node v_i and neighbor v_j. In GAT [33], the aggregator is a learnable weighted summation (sum_{lw}). Convolutional operation (*Conv*) is used to aggregate the neighborhood information in LGCL [10]. Our aggregator function can be seen as an extention of sum_{lw}, which applies learnable masks to filter the neighborhood information before summation. We summarize the relationship between desiderata and mentioned aggregator in Table 1. Our aggregator satisfies all desiderata, enabling a leap in model capacity. Furthermore, analyzing the learned mask may lead to benefits in interpretability.
- **Auxiliary model**. GCN, GraphSAGE [14] and GIN [37] do not use any auxiliary model to guide the aggregation process, and sum or mean the neighborhood directly. while a shared convolutional layer and a shared single-layer feed forward neural network are used in LGCL and GAT respectively. Considering the limitation of CNN and RNN whose input data should be ordered and fixed-size, we use a shared single-layer feed forward neural network.
- **Input and output of the auxiliary model**. For GAT, the input is node-neighbor pairs (input), and the auxiliary model learns from them to get coefficients (output) between nodes, which allows the aggregator to focus on most relevant nodes. The aggregator adds up each neighbor corresponding to the learned weight to get the aggregation output. However, GAT only learns node-level attention. LGCL uses the reorganized neighbor's embedding (input) that selects the d-largest values for each feature from neighbors to calculate the convolutional filter's weights (output). This strategy allows for feature-level attention, but it can not deal with variable size inputs (the number of adjacent nodes usually varies for different nodes in a graph), due to the limitation of convolution operation. While we concatenate the central and neighbor before feed in the auxiliary model, which can be viewed as a simple form of a " skip connection " between different search depths and get the learned mask for each given node's neighbor.

Table 2: Overview of datasets for node classification.

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
Reddit	232,965	11,606,919	602	41	152K/23K/55K

Table 3: Overview of datasets for graph classification.

Datasets	Graphs	Classes	Avg. nodes
MUTAG	188	2	18
PROTEINS	1,113	2	39
PTC	344	2	26

3.5 Computational Complexity

A key part in our method is the auxiliary model, and it is a shared model by all nodes in a graph. So, the computation of the mask can be parallelized across all nodes, which is highly efficient. The computational complexity of Eq. (10) is $O(|\mathcal{E}| \times d_k \times d_{k-1} + |\mathcal{E}| \times 2d_{k-1} \times d_{k-1})$ and is in par with GCN ($O(|\mathcal{E}| \times d_k \times d_{k-1})$). As for the memory requirement, it grows linearly in the size of the dataset and we perform mini-batch training to deal with this issue.

4 EXPERIMENTS

We perform evaluation on node classification and graph classification, and study our model's interpretability and robustness.

4.1 Datasets

We conduct node classification on three citation graphs (Cora, Citeseer and PubMed) and one social network (Reddit), which have been widely used in [1, 7, 10, 14, 21, 33, 38, 42]. Dataset statistics are summarized in Table 2 and Table 3.

- **Node classification.** In citation graphs, nodes correspond to documents and edges to (undirected) citations. Node features correspond to a sparse bag-of-words representation of a document or frequency-inverse document frequency (TF-IDF) of the word in the document. Each node has one class label, e.g., each document in Cora has one of the seven class labels (corresponding to seven machine learning subareas) [30]. Reddit is a large online discussion forum where users post and comment on content in different topical communities. The node label is the community, or "subreddit", that a post belongs to. The link means the same user comments on both posts. Hamilton et al. [14] concatenates the the average embedding of the post title, the average embedding of all the post's comments, the post's score, and the number of comments made on the post as node features.

- **Graph classification.** We use 3 bioinformatics datasets [37]. MUTAG is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds with 7 discrete labels. PROTEINS is a dataset where nodes are secondary structure elements (SSEs) and there is an edge between two nodes if they are neighbors in the amino-acid sequence or in 3D space. It has 3 discrete labels, representing helix, sheet or turn. PTC is a dataset of 344 chemical compounds that reports the carcinogenicity for male and female rats and it has 19 discrete labels.

4.2 Baselines and Experimental Setting

Node classification. We compare against 6 strong baselines: GCN [21], GAT [33], FastGCN [7], GraphSAGE-mean [14], LGCL [10] and MixHop [1] using the publicly released implementations. We split the train/validation/test as [7, 14]. Our code are available online. [1]

In our model, we first utilize one GCN layer to reduce the dimension of the node feature to 64-dimension for Cora, PubMed and 128-dimension for Citeseer and Reddit. Then we apply a one-layer neural network as the auxiliary model to learn masks for neighbors, whose input dimension is 128×64 (Cora, PubMed) and 256×128 (Citeseer and Reddit). Hyperparameters are optimized with the validation set [7]. Throughout the experiments, we use the Adam optimizer [20] with learning rate 0.005 for Cora and PubMed, 0.002 for Citeseer, and 0.01 for Reddit. We fix the dropout rate to 0.5 for the hidden layers' inputs and add an L2 regularization of 0.0001. We employ the early stopping strategy based on the validation accuracy and train 200 epochs at most. For Reddit, we use the mini-batch training and the batch size (512) is set to be the same as FastGCN and GraphSAGE.

For a fair comparison, we also use the hidden layer size of 64 units for GCN on Cora, PubMed and 128 for Citeseer, which ensures the architecture is the same with ours model (except the auxiliary model part). We use the same architecture as in the original papers for GAT, LGCL, FastGCN, GraphSAGE and MixHop. We report results over 20 runs with random weight matrix initialization.

Graph classification. Here, we report the results for WL subtree, DCNN [2], PATCHYSAN [27], DGCNN [43], AWL [19] and GIN with its variants as in paper [37]. Following [37], we use 10-fold cross-validation (nine folds for training and one for testing) for graph classification. Because of the small dataset sizes, hyperparameters selection using a validation set is extremely unstable. Thus, we report the average and standard deviation of validation accuracies across the 10 folds within the cross-validation as in [27, 37, 39]. We use the following hyperparameters for MUTAG, PTC and PROTEINS: 0.005 (learning rate), 16 (the number of hidden units), 0.5 (dropout ratio), 32 (batch size). We replace the sum aggregator in GIN with our learnable aggregator and MLPs with two layers are applied after aggregation. Batch normalization [20] is applied on each hidden layer.

4.3 Node and Graph classification Results

4.3.1 Compared with Baselines. Results for node classification and graph classification are summarized in Table 4 and Table 5. We observe that LA-GCN$_{Mask}$ outperforms all the mentioned methods across all datasets except MUTAG.

Table 4: Node classification accuracy (%). The best results are in bold and the second best ones are underlined.

Methods	Cora	Citeseer	PubMed	Reddit
GCN	88.0 ± 0.47	77.8 ± 0.13	86.8 ± 0.81	93.0 ± 0.47
GAT	80.4 ± 0.17	75.7 ± 0.30	85.0 ± 0.02	–
LGCL	86.9 ± 0.20	77.5 ± 0.22	84.1 ± 0.13	–
FastGCN	85.0 ± 0.30	77.6 ± 0.60	88.0 ± 0.30	93.7 ± 0.62
GraphSAGE	82.2 ± 0.80	71.4 ± 1.00	87.1± 0.60	94.6 ± 0.40
MixHop	88.3± 0.82	–	85.6 ± 0.71	–
Ours	**89.1± 0.17**	**78.7 ± 0.53**	**89.1 ± 0.21**	**95.1 ± 0.23**

Table 5: Graph classification accuracy (%). The best results are in bold and the second best ones are underlined.

	Methods	MUTAG	PROTEIN	PTC
Baselines	WL subtree	90.4	75.0	59.9
	DCNN	67.0	61.3	56.6
	PATCHYSAN	**92.6**	75.9	60.0
	DGCNN	85.8	75.5	58.6
	AWL	87.9	–	–
GNN variants	GIN	89.4	76.2	64.6
	Sum-1-Layer	90.0	76.2	63.1
	Mean-MLP	83.5	75.5	66.6
	Mean-1-Layer	85.6	76.0	64.2
	Max-MLP	84.0	76.0	64.6
	Max-1-Layer	85.1	75.9	63.9
LA-GCN$_{Mask}$ (Ours)		90.0	**80.5**	**72.2**
	Improvement	-	4.30	5.60

For node classification, GCN outperforms GAT, which is consistent with the results reported in [1, 38]. FastGCN and GraphSAGE focus on improving the training efficiency so they have slightly worse results than GCN. LGCL reorganizes the original embedding in the process of constructing feature maps and it does not perform well particularly on PubMed. MixHop utilizes different hop neighbors information and gets the second best performance on Cora, but does not perform well on Citeseer and PubMed. One possible reason is that it does not filter the neighborhood information, which may aggregate some noisy information from higher-order neighbors.

For graph classification, Table 5 compares LA-GCN$_{Mask}$ with GIN, other GNN variants, as well as other strong baselines. In general, GNN variants perform better than the mentioned baselines, and the main reason is that they can not combine node features, which might limit the models' capacity. GNNs with sum aggregator tend to fit the training sets better than mean and max-pooling aggregators. Further, we can see that replacing the sum aggregator in GIN can significantly improve the accuracy on PROTEIN and PTC datasets by 4.5% and 5.6%, excluding MUTAG. One possible reason for the poor performance on MUTAG is that our model may not be fully trained due to the small training sample size.

[1] https://github.com/LiZhang-github/LA-GCN

Table 6: Node classification with different label size (%). The best results are in bold and the second best ones are underlined.

Datasets	Methods	1%	2%	3%	4%	5%	10%	20%	30%	40%	50%	Average
Cora	GCN	58.41	71.70	75.83	79.27	82.28	**85.50**	85.57	**86.93**	87.00	86.27	79.87
	GAT	45.31	58.79	68.12	71.23	77.49	85.20	85.90	86.70	87.10	86.20	74.66
	LGCL	60.58	71.25	75.55	79.28	**82.53**	84.85	**86.03**	86.58	86.85	87.13	80.06
	Ours	**63.50**	**73.61**	**76.70**	**79.49**	81.11	84.34	85.58	86.58	**87.68**	**87.72**	**80.60**
Citeseer	GCN	42.76	69.29	71.66	**72.50**	**73.32**	**76.90**	77.77	77.93	77.83	78.17	70.93
	GAT	47.78	63.57	54.38	50.48	72.10	75.40	74.60	75.20	77.00	77.02	64.95
	LGCL	**57.80**	66.92	**72.32**	71.28	73.10	76.34	76.38	76.86	77.07	77.02	72.51
	Ours	57.35	**69.52**	71.02	72.03	**72.16**	76.48	**78.49**	**78.12**	**79.29**	**79.35**	**73.26**
PubMed	GCN	79.92	80.46	79.18	79.28	79.62	82.47	84.30	83.40	84.70	85.07	81.48
	GAT	78.56	79.48	78.02	78.62	78.64	81.60	83.00	83.20	83.20	83.20	80.48
	LGCL	**81.95**	**82.70**	**83.10**	82.93	81.30	82.50	85.37	84.60	85.46	85.74	83.32
	Ours	80.00	82.44	81.35	**83.13**	**82.67**	**85.50**	**86.70**	**87.50**	**87.50**	**87.30**	**84.09**

4.3.2 Training Size Study. We also compare our method with closely related methods, GCN (mean_w aggregator), GAT (sum_{lw} aggregator) and LGCL (*Conv* aggregator), in two scenarios: small training size (1%, 2%,..., 5% for Cora and Citeseer, 0.5%, 0.6%, 0.7%, 0.8%, 0.9% for PubMed [2]) and large training size (10%, 20%,..., 50%) for Cora, Citeseer and PubMed. Results are summarized in Table 6. Note that Reddit is too large for GAT, so we only report results on three citation graphs.

Table 6 shows node classification results with different training sample sizes. On the whole, our model has achieved competitive performance in small training sample size and got better performance, especially with more training data. The main reason is that our model has more parameters than GCN, and we briefly summarized the number of parameters of GCN, GAT and LA-GCN. GCN's parameter is less than ours on Cora, Citeseer and PubMed 8.16%, 6.46% and 20.0%, respectively. Compared with GCN, our model need more training samples. For a more intuitive comparison, we average these results for each method under different training size. LGCL and LA-GCN_{Mask} outperform GCN and GAT, which indicates that being discriminative to feature-level is crucial for node classification.

4.3.3 Aggregator Study. To show the effectiveness of our aggregator, we compare our aggregator with three fundamental aggregators: mean, sum and maxpooling[3]. Results are summarized in Table 8.

Table 7: Node structure and feature statistics. (H.Nd.: Highest Node degree, L.Nd.: Lowest Node degree, M.Nd.: Median Node degree, and A.Nd: Average node degree. Fea.De. means feature density).

Dataset	H.Nd.	L.Nd.	M. Nd.	A.Nd	Fea.De.
Cora	168	1	4	4.9	1.26%
Citeseer	99	1	3	3.7	0.84%
PubMed	171	1	3	5.5	1.00%

[2] Compared with Cora and Citeseer, PubMed has more nodes. So, we choose the training size with smaller percentages.

[3] We use the same model architecture, besides the aggregator, and we name them as: GCN_{mean}, GCN_{sum} and $\text{GCN}_{pooling}$.

Table 8: Different aggregators for node classification (%).

Dataset	Cora	Citeseer	PubMed
GCN_{mean}	87.7 ± 0.21	77.7 ± 0.22	86.0 ± 0.13
GCN_{sum}	85.5 ± 0.49	77.1 ± 0.45	85.2 ± 0.52
$\text{GCN}_{pooling}$	84.7 ± 3.26	**79.1 ± 0.44**	86.2 ± 0.30
Ours	**89.1± 0.17**	78.7 ± 0.53	**89.1 ± 0.21**

Table 8 shows that our aggregator works better on Cora and PubMed than other aggregators, but not on Citeseer. The main reason is that Citeseer is more sparse in both graph structure and node feature, as shown in Table 7. The median of the neighbors' number is 3 and the feature density is 0.84% (3703 is divided by 18, the average number of "1" in a feature vector). So, max-pooling may be the best way that can collect most information from the neigborhood, which benefits the later feature transformation stage. But the result of pooling aggregator may not be very stable, and the standard deviation (3.26%) is almost ten times higher than other aggregators on Cora. As for mean and sum aggregator, mean aggregator performs better both in accuracy and stability in general. Adding up all neighbors' feature vectors may change the scale of the feature, which may not be good for node classification task.

4.4 Interpretability

Sum, mean, pooling and *Conv* aggregators mix or reorganize neighborhood information, which makes it difficult to interpret the learned representation because we can not distinguish which node and feature have a salient influence on the prediction result. While our aggregator provides a learned mask for each neighbor, which provide a qualitative and quantitative understanding of the relationship between input nodes and the prediction. In this subsection, we aim to answer the following **Q**uestions: **(1)** what we expected, **(2)** what we learned and **(3)** what we concluded.

For **Q1**, the expectation intuitively is that the learned mask should assign more weights to important neighbors and features.

In order to answer **Q2**, we visualize the learned masks for a representative node: Node 4 in Cora with neighbors from different

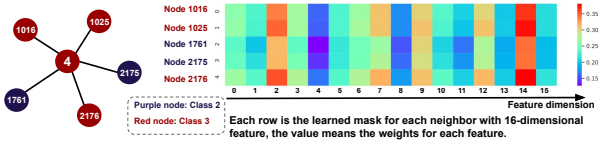


Figure 4: Visualization of the learned mask. The proposed aggregator can focus on important neighborhood information (e.g. the neighbors from the same class, or some highly relevant features) with the learned mask. The values showed in the heat map are the real values of the weights.

classes, as shown in Fig. 4. Central node 4 and its neighbors 1016, 1025 and 2176 belong to the same class, while neighbors 1761, 2175 belong to another class (class 2). From Fig. 4, we see that neighbors (1016, 1025, 2176) from the same class are assigned more weights (the values in the learned mask) than the other two neighbors (1761, 2175) on the whole. Besides, the mask gives high importance scores to some specific feature dimensions. We also analyze how GCN and GAT aggregate node 4's neighbors. GCN assigns weights - 0.2, 0.16, 0.16, 0.17, 0.2 to nodes 1016, 1025, 2176, 1761, 2175 respectively, depending on node 4 and its neighbors' node degree. The neighbors are treated differently in node-level, but it is not as we expected. It is reasonable to expect that node 1025 and 2176 (from the same class with central node) should be given higher scores than node 1761 and 2175. For GAT, the learned attention weights are all around 0.17, and the neighbors are not treated significantly differently.

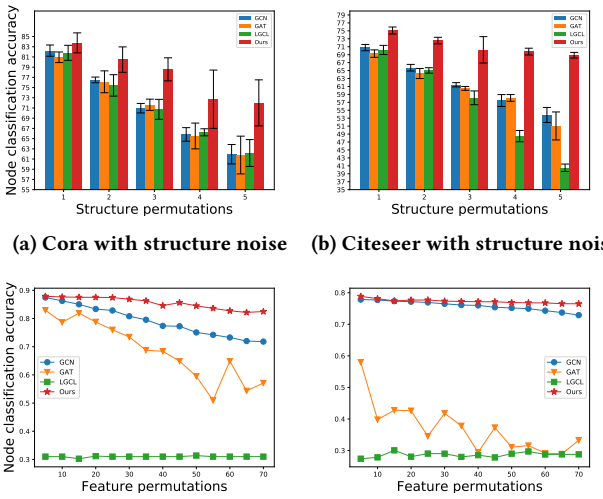
This indicates that the auxiliary model learns the expected rules (focusing on the important neighbors and features), which is used to assist our aggregator to jointly consider node-level as well as feature-level modulation of neighborhood information in the aggregation process (**Q3**). However, all features in one feature vector share the same weights in both GCN and GAT.

4.5 Robustness

Because real-world graphs are noisy, an essential criterion is that the model should be robust. As shown in [46], permutations to both graph structures and node features are harmful. To study the robustness of LA-GCN$_{\text{Mask}}$, we test our model on both structure noisy graphs, i.e., changes to adjacency matrix, and node feature noisy graphs, i.e., changes to node feature matrix.

We follow [8] to utilize the simplest attack methods. Given a target node, we randomly delete or add edges to the graph. For structure attack, the budget for each node is from one to five, which means that we are allowed to randomly add or delete one to five neighbors for each node. Following [47], per-node changes to the node attributes are at most 5% of the node feature dimension. The node feature vector in Cora and Citeseer only contains 0 or 1, so we randomly flip the features for feature attack. We compare our model with GCN, GAT and LGCL on both structure and node feature noisy graphs. Considering the unstable problem caused by the noisy data for these models, we report the average of top 10 results over 40 runs for each method, as shown in Fig 5.

Fig. 5a and Fig. 5b shows that the performance gets worse with the attack budget increasing. Our model gets the best performance,



(a) Cora with structure noise **(b) Citeseer with structure noise**

(c) Cora with feature noise **(d) Citeseer with feature noise**

Figure 5: Robustness studies: (a) and (b) show the node classification accuracy on structure noisy graphs, and (c) and (d) show the node classification accuracy on node feature noisy graphs.

especially with more structures changed. When the structure permutation is 5, the second best can only achieve 62% and 54 % classification accuracy on Cora and Citeseer respectively, while ours are 72% and 68%. In this case, the feature vector of the central node is still well preserved and our aggregator can effectively identify those features good for the classification of the central node from noisy neighborhood information.

For node feature noisy graphs, as shown in Figs. 5c and 5d, GAT and LGCL degrades significantly and our method shows strong robustness. Compared with GCN, the improvement is not as significant as in structure attack experiments. In this scenario, the central node's feature is also polluted in some extend, which may mislead the learned mask in the neighborhood aggregation process.

5 CONCLUSION

In this paper, we unified current aggregators in a framework: LA-GCN, with an auxiliary model to guide the neighborhood aggregation process. Considering most real-world graphs with no regular connectivity and order, we carefully designed the auxiliary model under this framework and proposed a new aggregator: mask aggregator. The proposed model allows end-to-end training and both node-level and feature-level attention for neighborhood information. LA-GCN$_{\text{Mask}}$ provides a variety of benefits, from an easy implementation with a much better performance, to interpretability, to robustness in noisy graphs. We evaluated LA-GCN$_{\text{Mask}}$ against six state-of-the-art methods on variable type and size graphs for node classification and six strong baselines on graph classification. Experimental results showed the superior performance of LA-GCN$_{\text{Mask}}$ over other methods on the whole, particularly a remarkable improvement on noisy graphs. Furthermore, analyzing the learned mask provided a straightforward interface for make

sense out of prediction and quantified understanding of the relationship between input nodes and prediction. In addition, the proposed mask aggregator can be integrated with other GCN variants such as FastGCN [7], jumping knowledge networks [38], GMWW [28] and MixHop [1].

6 ACKNOWLEDGMENTS

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. We would like to thank Dr Nikolaos Aletras, Prof. Eleni Vasilaki, Dr. Mark Stevenson, and our group member - Yan Ge, Shuo Zhou, Heda Song for their helpful discussions.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Hrayr Harutyunyan, Nazanin Alipourfard, Kristina Lerman, Greg Ver Steeg, and Aram Galstyan. 2019. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *ICML*.
[2] James Atwood and Don Towsley. 2016. Diffusion-convolutional Neural Networks. In *NeurIPS*.
[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*.
[4] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).
[5] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research* (2006).
[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *ICLR*.
[7] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.
[8] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2019. Adversarial Attack on Graph Structured Data. In *International Conference on Machine Learning (ICML)*.
[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NeurIPS*.
[10] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-Scale Learnable Graph Convolutional Networks. In *SIGKDD*.
[11] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *ICML*.
[12] Michele Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* 2 (2005), 729–734.
[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*.
[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.
[15] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* (2017).
[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* (1997).
[17] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* (1991).
[18] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* (1989).
[19] Sergey Ivanov and Evgeny Burnaev. 2018. Anonymous Walk Embeddings. In *ICML*.
[20] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.
[21] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[22] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In *NeurIPS*.
[23] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* (1998).
[24] John Boaz Lee, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh. 2019. Attention models in graphs: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 13, 6 (2019), 1–25.
[25] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.
[26] Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. 2019. Janossy Pooling: Learning Deep Permutation-Invariant Functions for Variable-Size Inputs. In *ICLR*.
[27] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*.
[28] Meng Qu, Yoshua Bengio, and Jian Tang. 2019. GMNN: Graph Markov Neural Networks. In *ICML*.
[29] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* (2009).
[30] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* (2008).
[31] Otilia Stretcu, Krishnamurthy Viswanathan, Dana Movshovitz-Attias, Emmanouil Platanios, Sujith Ravi, and Andrew Tomkins. 2019. Graph Agreement Models for Semi-Supervised Learning. In *NeurIPS*.
[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*.
[33] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.
[34] Edward Wagstaff, Fabian B. Fuchs, Martin Engelcke, Ingmar Posner, and Michael A. Osborne. 2019. On the Limitations of Representing Functions on Sets. In *ICML*.
[35] B. Yu. Weisfeiler and A. A. Leman. 1968. Reduction of a graph to a canonical form and an algebra arising during this reduction.
[36] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. 2012. Deep learning via semi-supervised embedding. In *Neural networks: Tricks of the trade*. Springer, 639–655.
[37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.
[38] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*.
[39] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1365–1374.
[40] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GNN Explainer: A Tool for Post-hoc Explanation of Graph Neural Networks. In *NeurIPS*.
[41] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. 2017. Deep Sets. In *NeurIPS*.
[42] Li Zhang, Heda Song, and Haiping Lu. 2018. Graph node-feature convolution for representation learning. *arXiv preprint arXiv:1812.00086* (2018).
[43] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. In *AAAI*.
[44] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. 2018. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4320–4328.
[45] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*.
[46] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *SIGKDD*.
[47] Daniel Zügner and Stephan Günnemann. 2019. Certifiable robustness and robust training for graph convolutional networks. In *SIGKDD*.