Learnable Aggregator for GCN

Li Zhang
University of Sheffield
lzhang72@sheffield.ac.uk

Haiping Lu
University of Sheffield
h.lu@sheffield.ac.uk

Abstract

Neighborhood aggregation is a key operation in Graph Convolutional Network (GCN). Sum and mean of the neighborhood information are two most popular aggregation methods, but they have two main limitations: 1) can not treat each neighbor differently, 2) and can not treat each feature within a neighbor's feature vector differently. In this paper, we propose a learnable aggregator by borrowing idea from meta-learning. Specially, it trains a meta-learner in the aggregation process to produce a specific mask for each neighbor, allowing the aggregator to learn to assign different weights to different features within a feature vector. We illustrate the strength of our method on both node classification and graph classification task.

Key words: GCN, aggregator, meta-learning, mask.

1 Introduction

Graph convolutional network (GCN) is an effective neural network model for graphs that can combine structure information and node features in the learning process [14]. It represents a node by aggregating the feature vectors of its neighbors with fixed weights inversely proportional to the central and neighbors' node degrees. Later, some other aggregators were proposed: mean aggregator, LSTM aggregator, pooling aggregator [7] and sum aggregator [20]. However, these aggregators are mostly limited to predefined heuristics and assumption that connected nodes in the graph are likely to share the same label [23, 3]. The heuristic aggregator and assumption might restrict modeling capacity, as node can be connected with neighbors from different class and each feature within neighbor's feature vector plays different role for the central node's representation learning.

We desire a more flexible and intelligent aggregator that can be: 1) adaptive to deal with various neighborhood information in node-level and feature-level [19, 5]; 2) invariant to neighbors' order (real graphs mostly have no regular connectivity and natural ordering) [15]; 3) discriminative to graph structures by mapping them to different location in the embedding space [20]; 4) explainable for the aggregation results. The aggregator incorporates both graph structures and node features, which lead results difficult to interpret [22].

Graph attention network (GAT) borrows the idea of attention mechanisms and can learn to assign different weights to different neighbors in the aggregation process [19]. However, all individual features in a feature vector are treated equally, which do not satisfy Desirable 1. Learnable graph convolutional layer (LGCL) applies convolution operation in the aggregation process, which can assign different weights to different features [5]. But, the convolution operation performs on reorganized embeddings (selecting the $d-$largest values for each feature from neighbors), for the number of adjacent nodes usually varies for different nodes in a graph. LGCL satisfies Desirable 1, but not fully Desirable 2 and Desirable 4 (breaking the original correspondence between node features). The key challenge is to design a flexible and learnable aggregator that can satisfy the mentioned desirables.

Meta-learning methods learn a meta-learner to extract meta-knowledge from a number of different tasks and use it to assist in unseen tasks [18]. Here we adapt this idea to deal with the various

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

neighborhood information in a graph. We treat different neighborhood information as different tasks to train a meta-learner in the aggregation process, and it can learn some high-level rules (e.g., focusing on the important neighbors and features for node representation learning) to guide the aggregator to aggregate neighborhood information. Under this framework, we propose our method: learnable aggregator for GCN (LA-GCN), which utilizes a given node and its neighbors to train a meta-learner. It can produce a mask for each neighbor. Then we use Hadamard product to multiply the neighbor's feature vector with the corresponding mask in the aggregation process, allowing the aggregator learn to assign different weights for different features in different neighbors.

Our contributions are summarized as follows: 1) we propose a framework for graph representation learning by borrowing idea from meta-learning that unifies existing approaches, e.g., GAT, LGCL; 2) we propose LA-GCN with a flexible and learnable aggregator that can treat both the node and features within a feature vector differently, meanwhile satisfies the expected desirables. LA-GCN outperforms state-of-the-art methods in semi-supervised node classification task and graph classification task; 3) furthermore, we also analyze the learned mask to show the interpretability that are not well equipped in existing works.

2 Background

Graph Convolutional Network. Kipf and Welling [14] proposed graph convolutional network (GCN) as an effective graph representation model that can naturally combine structure information and node features in the learning process. Let $\mathcal{G}$= $(\mathcal{V}, \mathcal{E}, X)$ 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 $\mathbf{A}$, and a feature matrix $\mathbf{X}$. The propagation rule of GCN can be summarized by the following expression:

$$\mathbf{h}_i^{(k)} = \sigma(\mathbf{W}^{(k)}(\mathbf{h}_i^{(k-1)} + \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} / \sqrt{d_i d_j})), \tag{1}$$

where $\mathbf{h}_i^{(k)}$ is the representation of v_i after $k-$layer GCN, $\mathbf{W}^{(k)} \in \mathbb{R}^{C \times F}$ is the learned weight matrix, $\mathcal{N}_i$ is a set of nodes adjacent to v_i, d_i and d_j are the node degrees of node v_i and node v_i respectively. We initialize $\mathbf{h}_i^{(0)} = \mathbf{X}_i$. A key part in the propagation is the aggregation process and we define a GCN aggregator as f_{agg}:

$$\mathbf{s}_i^{(k-1)} = f_{agg}^{(k)}(\mathbf{h}_j^{(k-1)}, j \in \mathcal{N}_i) = \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} / \sqrt{d_i d_j}. \tag{2}$$

A node's representation captures the information within k-hop neighbors after k iterations of aggregation, and this can be treated as a general neural message-passing process [6] or relational inductive bias model [2].

Aggregator. Later, Hamilton et al. [7] proposed mean, LSTM, and pooling aggregators. Mean aggregator simply takes an elementwise mean of $\mathbf{h}_j^{(k-1)} (j \in \mathcal{N}_i)$, LSTM aggregator applies LSTM [9] to a random permutation of the neighbors and pooling aggregator applies an elementwise max-pooling on the neighbors after a linear mapping. Mean and max-pooling aggregators are well-defined functions as they are permutation invariant, but they can not always satisfy Desirable 2, which has been proved in [20]. Xu et al. [20] proposed that sum aggregator (an injective function) is the most expressive in graph representation learning.

The mentioned GCN, mean, LSTM, pooling and sum aggregators are all predefined heuristics. Some strategies have been proposed to make the aggregator learnable. In GAT, the aggregator can be formulated as: $\mathbf{s}_i^{(k-1)} = f_{aga}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{h}_j^{(k-1)}$, where α_{ij} is a learnable attention coefficient that indicates the importance of v_j to v_i. However, all the features are treated equally within the feature vector $\mathbf{h}_j^{(k)}$, for each feature shares the same weight α_{ij} in the aggregation. LGCL applies convolution operation on the reconstructed neighbors' feature in the aggregation $\mathbf{s}_i^{(k-1)} = f_{agc}^{(k)}(\mathbf{h}_j^{(k-1)}, j \in \mathcal{N}_i) = Cov(\hat{\mathbf{H}}_\mathbf{i}^{(k)})$, where $\hat{\mathbf{H}}_\mathbf{i}^{(k)}$ is the reconstructed neighbor's 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. The filter in the convolution process works on fix-sized feature map, and using convolution in the aggregation process is not that suitable to learn from neighbors with variable-size.

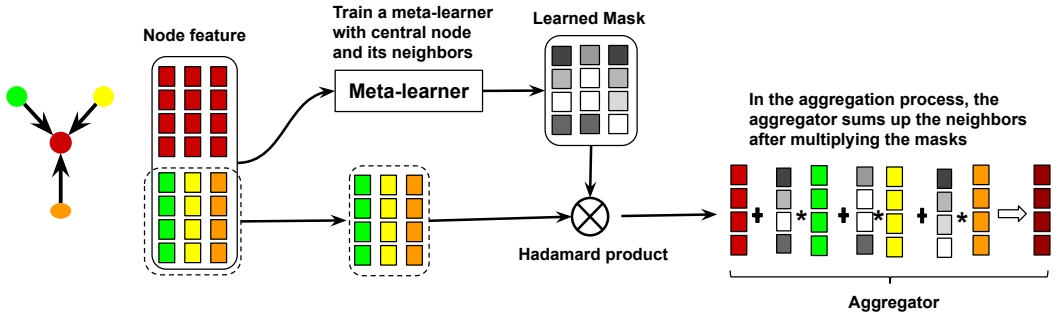

Figure 1: LA-GCN can be mainly divided into three steps: 1) train a meta-learner with a given node and its neighbors' feature vectors; 2) get the mask for each neighbor from the meta-learner; 3) aggregate the neighbors (after multiplying the corresponding mask) to get the central node's new representation.

3 LA-GCN.

Model: Our method is shown in Fig.1, and we replace the aggregator $f_{agg}^{(k)}$ in Eq. 2 with $f_{agm}^{(k)}$, a permutation-invariant function, that satisfies the desirables mentioned in Section 1. As far as we know, sum aggregator is an injective function and the most powerful in GNNs, which could precisely capture the graph structures [20].[1] Instead of summing all the neighbors directly, we extend sum aggregator as:

$$\mathbf{s}_i^{(k-1)} = f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} * \mathbf{m}_j^{(k-1)}, \tag{3}$$

where $\mathbf{m}_j^{(k-1)}$ is the mask learned by the meta-learner. The meta-learner can be a multi-layer perceptrons (MLPs) [11, 10] or autoencoders [8, 13], and be trained on different tasks (each given node, each given node's neighbors or each given node with its neighbors in a graph).

In this paper, we train the meta-learner (an MLP with a single hidden layer) on a given node and its neighbors ($\{\ \mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}, j \in \mathcal{N}_i\}$), and it is defined as following:

$$\mathbf{m}_j^{(k-1)} = MLP^{(k)}(\|\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}) = \sigma(\mathbf{W}_m^{(k)}(\|\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)})), \tag{4}$$

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

$$\mathbf{h}_i^{(k)} = \sigma(\mathbf{W}^{(k)}(\mathbf{h}_i^{(k-1)} + \mathbf{s}_i^{(k-1)})). \tag{5}$$

Our method has several interesting properties: 1) it can simultaneously assign different weights to different neighbors (node-level attention) and features within a feature vector (feature-level attention) in the aggregation process; 2) it allows for dealing with variable-size inputs for we use a shared meta-learner for each node; 3) for it can be seen as an extension of sum-aggregator, which could satisfy Desirable 3. 4) we can easily see which neighbors or features are important to the central node by visualizing the learned masks (Desirable 4).

A general framework. Our framework is general and can unify GAT and LGCL. The meta-learner in GAT is the self-attention strategy (a single-layer feed forward neural network), and it treats each node as a task to learn coefficients between nodes (node-level attention). While, our method utilizes both the central node and neighbors to train the meta-learner and it learns a mask for each neighbor (both node and feature-level attention). GAT can be seen as a special case under our framework that all the values in a mask is the same. The meta-learner in LGCL can be seen as the convolutional layer and the learning task is a given node with neighbors' information, but the convolutional layer is not suitable to deal with graphs with no regular connectivity.

Computational complexity: A key part in our method is the meta-learner, and it is a shared model by all nodes in a graph. So, the computation of the mask can be parallelized across all nodes. The computational complexity of Eq. 5 is $\mathcal{O}(\mid \mathcal{E} \mid \times C \times F + \mid \mathcal{E} \mid \times 2F \times F)$ and is on par with GCN

[1] Sum aggregator works better for graph classification. For node classification, sum aggregator can change the feature' scale and mean aggregator or GCN aggregator are more resealable for this task.

Table 1: Node classification accuracy (%)

Methods	Cora	Citeseer	PubMed	Reddit
GCN	88.0	77.8	86.8	93.0
GAT	80.4	75.7	85.0	–
FastGCN	85.0	77.6	88.0	93.7
GraphSAGE_mean	82.2	71.4	87.1	94.6
LGCL	86.9	77.5	84.1	–
MixHop	88.3	73.9	85.6	–
LA-GCN	**89.1 %**	**78.7%**	**89.1%**	**95.1%**

Table 2: Graph classification accuracy (%)

Methods	MUTAG	PROTEIN	PTC
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	–	–
GIN	89.4	76.2	64.6
LA-GCN	90.0	**80.5**	**72.2**

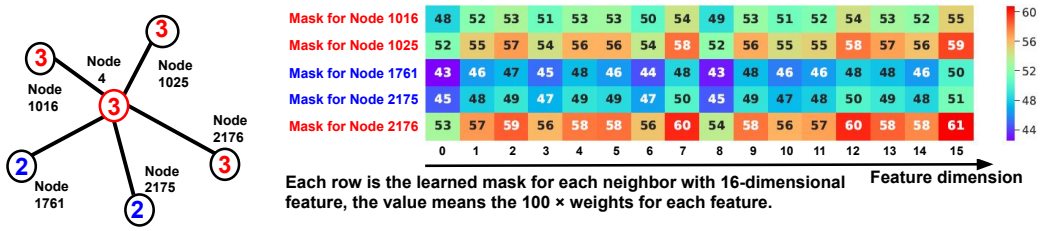

Figure 2: Visualization of the learned mask. The aggregator can focus on the important neighborhood information (the neighbors from the same class, or some special features) with the learned mask. The values showed in the heat map are 100 $\times$ the real values.

($\mathcal{O}(\mid \mathcal{E} \mid \times C \times F)$). 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 and conclusion.

We perform LA-GCN on two tasks: node classification and graph classification.

Datasets: We conduct node classification on three citation graphs (Cora, Citeseer and PubMed) and one social network (Reddit), which have been widely used in [14, 4, 21, 7, 19, 5, 1]. For graph classification task, we use 3 bioinformatics datasets [20]. For node classification, we split the train/validation/test as [7, 4]. Following [20, 16], we utilize 10-fold cross-validation (using 9 folds for training and 1 for testing). Please see Appendices A for more details.

Baselines and experimental setting. 1) For node classification, we compare against 6 strong baselines: GCN [14], GAT [19], FasTGCN [4], GraphSAGE-mean [7], LGCL [5] and MixHop [1]. In our model, we first utilize one-layer GCN to reduce the dimension of the node feature and apply a one-layer neural network as the meta-learner. 2) For graph classification, we report the results as in paper [20]. Please see Appendices B for details of hyperparameters

Discussion. Results for node classification and graph classification are summarized in Table 1 and Table 2. Our method can get competitive results for the two tasks. Besides, we analyse what the meta-learner learned by visualizing the learned masks as shown in Fig 2. Central node 4 and its neighbors 1016, 1025 and 2176 belong to the same class, and neighbors 1761, 2175 belong to other class (class 2). The left table in Fig. 2 shows the learned mask for each neighbor. From the table, we can see that neighbors (1016,1025,2176) from the same class are assigned more weighs (the values in the learned mask) than the other two neighbors (1761, 2175) on the whole. Besides, each feature within a feature vector is treated differently. This indicates that the meta-learner learns the expected rules (focusing on the important neighbors and features) on how to aggregate the neighbors.

Conclusion: We proposed a framework that train a meta-learner for the aggregator and our method outperform state-of-the-art methods on both node classification and graph classification tasks. It is possible to implement our method (learn a flexible and adaptive aggregator) to other frameworks, e.g., FastGCN [4], jumping knowledge networks [21], more recently GMWW [17] and MixHop [1].

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] 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).

[3] 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).

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

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

[6] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *ICML*.

[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

[8] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* (2006).

[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* (1997).

[10] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* (1991).

[11] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* (1989).

[12] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

[13] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. In *ICLR*.

[14] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[15] Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. 2019. Janossy Pooling: Learning Deep Permutation-Invariant Functions for Variable-Size Inputs. In *ICLR*.

[16] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *ICML*.

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

[18] Sebastian Thrun. 1998. Lifelong learning algorithms. In *Learning to learn*.

[19] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

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

[21] 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*.

[22] 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*.

[23] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*.

Appendices

A Datasets for node classification and graph classification.

The datasets for node classification and graph classification are summarized in Table 3 and Table 4. For more details for the datasets, please refer to [14, 7] and [20].

Table 3: Overview of datasets for node classification.

Dataset	Nodes	Edges	Features	Classes	Training	Validation	Testing
Cora	2,708	5,429	1,433	7	1,625	540	540
Citeseer	3,327	4,732	3,703	6	1,996	665	665
PubMed	19,717	44,338	500	3	1,1830	3,943	3,943
Reddit	232,965	11,606,919	602	41	152,410	23,699	55,334

Table 4: Overview of datasets for graph classification.

Datasets	MUTAG	PROTEINS	PTC
Graphs	188	1,113	344
Classes	2	2	2
Avg nodes	78	39	26

B Hyperparameters.

For node classification. We compare against 6 strong baselines: GCN [14], GAT [19], FasTGCN [4], GraphSAGE-mean [7], LGCL [5] and MixHop [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 meta-leaner to learn the masks for neighbors, whose input dimension is 128×64 (Cora, PubMed) and 256×128 (Citeseer and Reddit). Throughout the experiments, we use the Adam optimizer [12] with learning rate 0.005 for Cora and PubMed, 0.002 for Citeseer, 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 meta-learner part). We use the same architecture as in the original papers for GAT, LGCL and MixHop, for these algorithms have many hyperparameters. The results for FastGCN and GraphSAGE are from FastGCN [4], for we use the same training/validation/testing with Chen et al. [4]. We report the mean accuracy of 15 runs with random weight matrix initialization.

For graph classification. For graph classification, we report the results for WL subtree, DCNN, PATCHYSAN, DGCNN, AWL and GIN (sum-aggregation) as in paper [20]. For our method, the learning rate is 0.005, the number of GCN layer is three, and MlPs have 2 layer and the hidden dimension is 16 for MUTAG, PTC and PROTEINS. The batch size is 32 and bath normalization is applied on every hidden layer.