Gated Graph Pooling with Self-Loop for Graph Classification

Xiaolong Fan*, Maoguo Gong*, Hao Li*, Yue Wu† and Shanfeng Wang‡
*School of Electronic Engineering
Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education
Xidian University, Xi'an, Shaanxi Province 710071, China
Email: xiaolongfan@outlook.com; gong@ieee.com; haoli@xidian.edu.cn
†School of Computer Science and Technology, Xidian University, Xi'an, Shaanxi Province 710071, China
Email: ywu@xidian.edu.cn
‡School of Cyber Engineering, Xidian University, Xi'an, Shaanxi Province 710071, China
Email: sfwang@xidian.edu.cn

***Abstract*—Graph classification is a practical problem in many different domains including bioinformatics, chemoinformatics, social network analysis, and etc. For the graph classification task, the existing graph neural network approaches usually generate graph features using graph pooling at each step. However, this strategy of pooling only at the current step ignores the impact of self-loop. To eliminate this limitation, we propose a novel self-loop graph pooling strategy that can utilize the node information of the current step and the graph representation information of the previous step to generate an effective representation for the graph classification task. Further to measure the importance of self-loop, we also develop a gated approach, gated graph pooling with self-loop, that utilizes the simple fusion gate to enhance the representation capacity of the model. We evaluate our model on common benchmark datasets and experimental results have demonstrated the superior performance improvement on predictive accuracy.**

***Index Terms*—Graph Representation Learning, Graph Neural Network, Graph Pooling, Graph Classification.**

I. Introduction

Deep neural networks have achieved great success on Euclidean data analysis such as computer vision [1] and natural language processing [2] since they can automatically learn both low- and high-level feature representations of objects. However, there is an increasing number of applications where data are generated from the non-Euclidean domain. A typical non-Euclidean data structure is the graph which models a set of objects (*i.e.*, nodes) and their relationships (*i.e.*, edges). Many real-world data can be naturally organized in many complex graph structures, and a canonical task is graph classification [3]. In this task, the input data from bioinformatics [4], chemoinformatics [5], cyber-security [6], and social network analysis [7] are the form of the graph, and the output is to predict the graph label. The main difficulty in solving this task is how to find the appropriate way to extract node features and then aggregate node representations to generate graph representation benefiting the classification task.

To extract node features and utilize the underlying information encoded in graphs, graph neural network (GNN) is developed as a possible solution. Being one of the earliest works on graph neural networks, the graph neural network models (GNNMs) [3] recursively update node latent representations until convergence. To ensure convergence, the representation function must be a contraction mapping. The gated graph neural networks (GGNNs) [8] utilize the gated recurrent units (GRUs) [9] as update function and fix the number of recurrent steps which means it no longer needs to ensure convergence and can be trained via back-propagation through time (BPTT) algorithm. More recently, a large number of graph neural network models broadly follow the message passing neural network (MPNN) [10] framework which contains a message passing phase and a readout phase where the message passing is to aggregate each node's representations of neighbors to generate its new representation and the readout is then to capture the global graph information from node representation space. The key to extracting a node representation is the message passing function which aggregates each node's neighbor representations to generate its new representation. Graph convolutional networks (GCNs) [11] employ the Laplace matrix to normalize the adjacency matrix and allow the aggregate and update function to share the same structure by adding self-loop. The graph sample and aggregate (GraphSAGE) [12] model develop the local neighborhood sampling strategy and then the aggregate functions, including mean, sum and max function, to generate the representations of sampled nodes.

After obtaining the discriminative node features, the GNN model usually coarsens a graph into sub-graphs or sum or average over the node representations by graph pooling (*i.e.*, readout). To produce embedding vectors and predict class probabilities of the graph instances, the existing graph representation models read from the node representation space after sufficient iterations. A sufficient number of iterations is key to achieving good discriminative power but different nodes may have different neighborhood ranges so that node representations from earlier iterations may sometimes generalize better. For a better structure-aware approach, layer aggregation [13] was proposed to aggregate layer-wise representations with concatenation, max-pooling or other feasible approaches. To avoid the information loss, the concatenation based aggregator is usually used as the layer aggregation function to adapt the

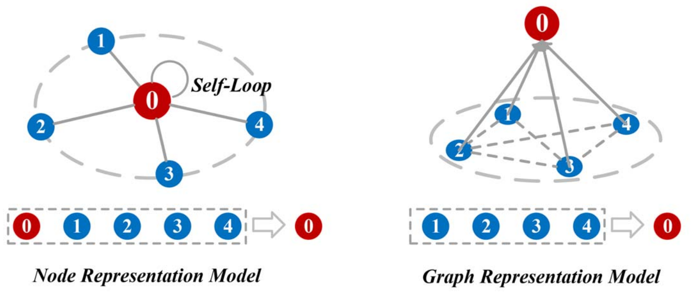

Fig. 1. Comparison of node representation and graph representation model. *Left:* classical graph representation model. *Right:* classical graph representation model. obviously, this graph representation model ignores the self-loop.

effective neighborhood size for each node as needed.

However, although achieving the superior performance, the existing pooling approaches, including original readout and layer aggregation readout, ignore the self-loop of graph representation. From the perspective of super node [10], obtaining a graph embedding is equivalent to generating super node embedding where each node in the initial graph serves as a neighborhood of super node, and node information are passed from the node space to the super node along the virtual directed edge. Apparently as shown in Figure 1, the existing pooling approaches ignore the impact of self-loop that is important in node representation learning.

Based on the above analysis, we therefore propose a novel self-loop graph pooling strategy that can utilize the node information of the current step and the graph representation information of the previous step to generate an effective representation for the graph classification task. The addition of self-loop enhances the flow of information and makes the network easier to train. To further measure the importance of self-loop, we also develop a self-loop graph gated pooling approaches using a simple fusion gate. Similar to the update gate in the gated recurrent unit (GRU) [9], this fusion gate helps the model to determine how much of the self-loop information from the previous step needs to be passed along to the subsequent steps. From the perspective of message passing, the existing models only design a feasible aggregation function while the proposed pooling approaches design both aggregation and update functions by adding a self-loop for the graph classification task. At the last step, the layer aggregation is employed to aggregate the layer-wise graph representation so that the final output can adaptively choose the different hop neighborhood information. We evaluate our model on popular graph benchmark datasets: MUTAG, PROTEINS, PTC_MR, D&D, NCI1, and MSRC_21. By comparing average accuracy after 10-fold cross validation with other basline models, experiments have demonstrated the superior performance improvement on predictive accuracy. To summarize, we highlight two main contributions of this paper as follows:

- We analyze the popular graph neural network pooling models for graph classification and conclude that these pooling models ignore the self-loop in readout phase, leading to limited representation capacity. Following this analysis, we therefore propose a novel self-loop graph pooling approach that can utilize the node information of the current step and the graph representation information of the previous step to generate an effective representation for the graph classification task.
- We develop a self-loop graph gated pooling approach to further measure the importance of self-loop using a simple fusion gate. This fusion gate helps the model to determine how much of the self-loop information from the previous step needs to be passed along to the subsequent steps.

II. Related work

Existing techniques for graph classification can be grouped into two categories: kernel methods and graph neural network.

A. Kernel Method

The kernel method calculates the similarity between the graphs, and then we can use kernel-based supervised algorithms such as support vector machines (SVM) [14] to obtain the boundaries between different classes without having to do feature extraction to transform them to fixed-length, real-valued feature vectors. There exist many kernel methods range from random walks [15], shortest paths [4] to subtree [16], etc. However, the topology of graphs in reality is very complicated so the simple Graph Kernel cannot solve the graph classification problem well. Weifsfeiler-Lehman (WL) [17] method was proposed to improve the performance of graph kernel. Similar to label propagation, the WL method needs to perform T iterations and each iteration builds a new graph. The node labels of the graph are updated according to the neighbor node labels of the previous iteration, and then the updated Graph Kernel is calculated. In the WL method, the similarity of the graphs is synthesized by the Graph Kernel during T iterations. Although graph kernels have achieved great success, they are computationally expensive and the feature selection is separate from the classifier.

B. Graph Neural Network

In order to solve the above challenges, graph neural networks (GNNs) are developed with an end-to-end fashion. Commonly, GNNs broadly follow the Message Passing [10] framework which contains a message passing phase and a readout phase, where the message passing is to aggregate each node's feature vector of neighbors to compute its new feature vector and the readout is then to generate the graph feature. The origin readout function only aggregates from node representation space after T iteration steps. This scheme may lead to over-smooth [18] due to local neighborhood properties. To adapt to different local neighborhood properties, layer aggregation [13] was proposed to tackle this challenge. Despite achieving superior performance, both schemes ignore self-loop for graph representation learning. Recently, several pooling based models have been proposed including top-k

TABLE I
MAIN SYMBOLS AND DESCRIPTIONS IN THIS PAPER.

Notations	Descriptions
G	input labelled graph
A	adjacency matrix
v, w	node in graph G
$\mathcal{N}(v)$	the neighborhood of node v
R	node embedding representation matrix
$\hat{\mathbf{y}}$	graph embedding representation vector

pooling [19], [20], differentiable pooling (DiffPool) [21]. Top-k pooling propagates only part of the input and this part is not uniformly sampled from the input and thus can thus select some local part of the input graph, completely ignoring the rest. DiffPool can generate hierarchical representations of graphs and provide a general solution to hierarchically pool nodes across a broad set of input graphs.

III. PRELIMINARY

In this section, we will formalize the graph classification problem and present the relevant research briefly. In the following, we define that the bold lowercase denotes a vector and uppercase denotes a matrix or a set of vectors. Table I presents the main symbols and descriptions in this paper.

A. Problem Definition

Given a graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{v_1, v_2, ..., v_{n_v}\}$ is a set of $n_v = |\mathcal{V}|$ nodes and $\mathcal{E} \subseteq (\mathcal{V}, \mathcal{V})$ is a set of $n_e = |\mathcal{E}|$ edges between nodes. In a graph G, let $A \in \mathbb{R}^{n_v \times n_v}$ to represent the adjacency matrix where $A_{ij} = \omega_{ij} = 1$ if $e_{ij} \in \mathcal{E}$ and $A_{ij} = 0$ if $e_{ij} \notin \mathcal{E}$. $X_v \in \mathbb{R}^{n_v \times d_v}$ and $X_e \in \mathbb{R}^{n_e \times d_e}$ are used to denote features for nodes and edges respectively. Given a graph set $S = \{(G_1, \mathbf{y}_1), (G_2, \mathbf{y}_2), ..., (G_L, \mathbf{y}_L)\}$, the aim of graph representation learning is to build a learning model $\mathcal{F}_\Theta : G_i \to \hat{\mathbf{y}}_i$ to generate graph representation vector $\hat{\mathbf{y}}_i$ for the test graph G_i, where the parameter Θ can be optimized via the standard supervised learning objective. The final graph representation $\hat{\mathbf{y}}_i$ can be used to perform graph classification or regression tasks.

B. Message Passing Framework

The message passing neural network (MPNN) [7] unifies several recently introduced graph neural networks, which consists of two phases, namely the message passing phase and the readout phase.

In the message passing phase, given a graph G, graph neural network aims to learn a representation vector $\mathbf{h}_v^t$of an arbitrary node v in graph G by a node aggregate and update function. Here, GNN iteratively updates the representation of a node by a graph aggregate function $M_t(\cdot)$ and graph update function $U_t(\cdot)$ in the form of

$$\mathbf{m}_v^{(t+1)} = \sum_{w \in \mathcal{N}(v)} M_t(\mathbf{h}_w^{(t)}), \tag{1}$$

$$\mathbf{h}_v^{(t+1)} = U_t(\mathbf{h}_v^{(t)}, \mathbf{m}_v^{(t+1)}). \tag{2}$$

where $\mathbf{h}_v^{(t)}$ and $\mathbf{h}_w^{(t)}$ are the hidden states of node v and w at step t. $\mathbf{m}_v^{(t+1)}$ is the hidden state of the neighborhood of node v. After efficient iteration, the node embedding representation $\mathbf{r}_v^{(T)} = \mathbf{h}_v^{(T)}$ can be obtained for the subsequent tasks. Note that we omit the edge features since our experimental datasets only utilize node features to predict the graph label. In practice, many state-of-the-art node representation models are designed with the same aggregate and update function via adding the self-loop, and focus on developing a appropriate scheme to guarantee the superior performance such as graph convolutional network (GCN) and graph attention network (GAT).

In the readout phase, the the node embedding representation $r_v \in R$ can be used to generate the final graph embedding representation vector $\hat{\mathbf{y}}$ by a readout function $R(\cdot)$ in the form of

$$\hat{\mathbf{y}} = R(\{\mathbf{r}_v | v \in G\}). \tag{3}$$

The three functions $M_t(\cdot)$, $U_t(\cdot)$ and $R(\cdot)$ are learned via optimizing a learnable parameter set Θ in an end to end fashion.

As stated in section I, the strategy of readout at final step may lead to over-smooth since different nodes may have different neighborhood ranges so that node representations from earlier iterations may sometimes generalize better. To eliminate this limitation, layer aggregation is developed to aggregate layer-wise graph representations by a aggregate function $AGG(\cdot)$ in the form of

$$\hat{\mathbf{y}}^{(t)} = R_t(\{\mathbf{h}_v^{(t)} | v \in G\}), \tag{4}$$

$$\hat{\mathbf{y}} = AGG(\hat{\mathbf{y}}^{(t)} | t \in T). \tag{5}$$

where $R_t(\cdot)$ represents readout at step t and T is the number of iteration step. The aggregation function $AGG(\cdot)$ may exists many feasible design schemes such as concatenation, summation or average pooling. According the multiset [1] theory, average pooling cannot distinguish some simple subgraph structures so the summation pooling is recommended as a readout function. However, although achieving superior performance, the existing pooling approaches, including pooling at final step and pooling with layer aggregation, ignore the self-loop of graph representation. From the perspective of super node, obtaining a graph embedding is equivalent to generating super node embedding where each node in the initial graph serves as a neighborhood of super node, and node information are passed from the node space to the super node along the virtual directed edge. For the sake of brevity, we suppose that the function U_t and R_t have the same architecture $f(\cdot)$. Hence the node and graph representation models can be represented as

$$\mathbf{h}_v^{(t+1)} = f(\mathbf{h}_v^{(t)}, \mathbf{m}_v^{(t+1)}), \tag{6}$$

$$\hat{\mathbf{y}}^{(t)} = f(\mathbf{h}_v^{(t)}). \tag{7}$$

[1]Xu et al. [22] developed a deep multisets theory, which is parameterizing universal multiset functions with the neural network, and a multiset is a generalized concept of a set that allows elements in it have multiple instances.

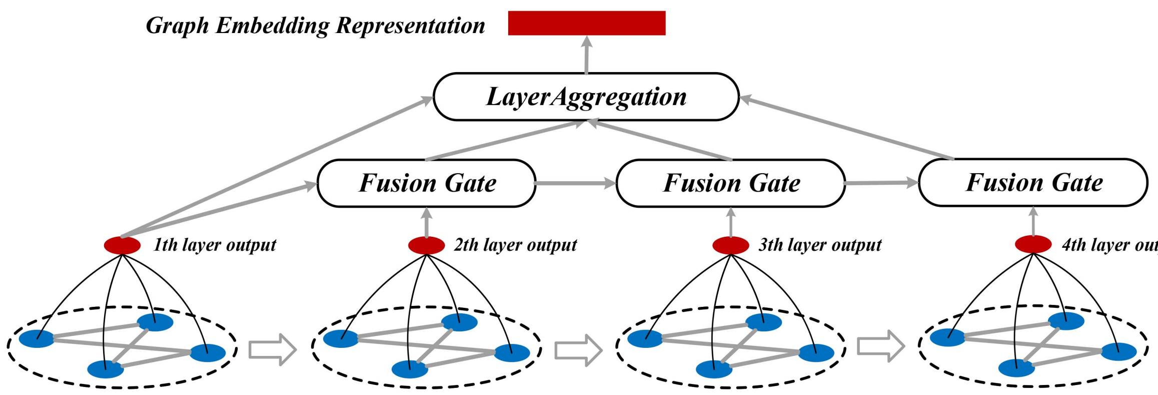

Fig. 2. The architecture of the proposed gated graph pooling with self-loop. The blue circles denote the nodes in graph and the red circles denote the graph representation. In message passing phase, each nodes are aggregated to generate the node representation embedding and then the layer-wise graph representation embedding can be obtained by aggregating node representations and self-loop representation. At the last step, the layer aggregation is employed to aggregate the layer-wise graph representation so that the final output can adaptively choose the different hop neighborhood information.

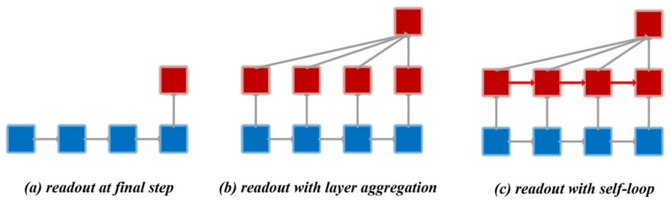
(a) readout at final step
(b) readout with layer aggregation
(c) readout with self-loop

Fig. 3. The information flow between iteration steps on existing and proposed architectures. The blue squares represent the message passing phase and the red squares represent the readout phase, and the red arrow represents the self-loop of the graph representation.

Intuitively, the existing pooling approaches ignore the impact of self-loop that is important in node representation learning (see Figure 1).

IV. Method

In the following, we will present the proposed graph pooling with self-loop and the gated graph pooling with self-loop respectively.

A. Graph pooling with self-loop

Based on the above analysis in section III-B, we first consider using a simple and intuitive method to add self-loop to the graph representation embedding. Similar to node representation learning model, we directly add the $t-1$th step graph representation embedding to tth step readout in form of

$$\hat{\hat{\mathbf{y}}}^{(t)} = R_t(\{\mathbf{h}_v^{(t)} | v \in G\}), \tag{8}$$

$$\hat{\mathbf{y}}^{(t)} = f(\hat{\mathbf{y}}^{(t-1)} + \hat{\hat{\mathbf{y}}}^{(t)}), \tag{9}$$

where $\hat{\mathbf{y}}^{(t-1)}$ represents the $t-1$th graph representation embedding and $f(\cdot)$ represents classical neural network like multi-layer perception (MLP).

After obtaining the layer-wise graph representation embedding with self-loop, we also utilize the layer aggregation scheme to aggregate the layer-wise graph representation embedding so that different local structures can adaptively choose different number of iterations, i.e.,

$$\hat{\mathbf{y}} = AGG(\hat{\mathbf{y}}^{(t)} | t \in T). \tag{10}$$

Inspired by graph isomorphism network (GIN) [22], we choose the simple but powerful aggregation function, i.e., summation, to compute the final graph representation embedding.

B. Fusion gate

To measure the importance of self-loop, we also utilize the dimension-wise dual fusion gate to compute the final graph representation embedding. Inspired by directional self-attention network [23], this fusion gate is accomplished by

$$F = \sigma(f(\hat{\mathbf{y}}^{(t-1)}, \hat{\hat{\mathbf{y}}}^{(t)})), \tag{11}$$

$$\hat{\mathbf{y}} = F \odot \hat{\mathbf{y}}^{(t-1)} + (1-F) \odot \hat{\hat{\mathbf{y}}}^{(t)}, \tag{12}$$

where $\odot$ represents entry-wise product and σ represents the $Sigmoid$ nonlinear activation function to obtain a value between $0 \sim 1$. Similar to the update gate in the gated recurrent unit (GRU), this fusion gate helps the model to determine how much of the self-loop information from the previous step needs to be passed along to the subsequent steps.

C. The overall framework

As shown in Figure 2, the overall framework of the proposed gated graph pooling with self-loop is composed of message passing phase, readout phase, fusion gate and layer aggregation. In message passing phase, each nodes are aggregated to generate the node representation embedding and then the layer-wise graph representation embedding can be obtained by aggregating node representations and self-loop representation. At the last step, the layer aggregation is employed to aggregate the layer-wise graph representation so that the final output can adaptively choose the different hop neighborhood information. Figure 3 illustrates the self-looping information flow and Algorithm 1 shows the pseudo code of proposed gated graph pooling with self-loop.

Algorithm 1 The pseudo code of proposed gated graph pooling with self-loop

Input: node feature $X_v \in \mathbb{R}^{n_v \times d_v}$, adjacency matrix $A \in \mathbb{R}^{n_v \times n_v}$, layer number T

Output: graph representation $\hat{\mathbf{y}} \in \mathbb{R}^n$

```
1: Procedure gated graph pooling with self-loop
2: $H_v^{(0)} \leftarrow X_v$; $\mathbf{h}_G^{(0)} \leftarrow 0$
3: for all $t = 1, 2, ..., T$ do
4:     $H_v^{(t)} = M_t(H_v^{(t-1)}, A)$
5:     $\hat{\hat{\mathbf{y}}}^{(t)} = R_t(H_v^{(t)})$
6:     $F = \sigma(f(\hat{\mathbf{y}}^{(t-1)}, \hat{\hat{\mathbf{y}}}^{(t)}))$
7:     $\hat{\mathbf{y}}^{(t)} = F \odot \hat{\mathbf{y}}^{(t-1)} + (1 - F) \odot \hat{\hat{\mathbf{y}}}^{(t)}$
   $\hat{\mathbf{y}} = AGG(\hat{\mathbf{y}}^{(t)} | t = 1, 2, ..., T)$
8: return graph representation $\hat{\mathbf{y}}$
```

TABLE II
THE STATISTICS OF BENCHMARK CLASSIFICATION DATASETS.

	MUTAG	PROTEINS	PTC_MR	D&D	NCI1	MSRC_21
Num. of Graphs	188	1113	344	1178	4110	563
Avg. Num. of Nodes	17.93	39.06	14.29	284.32	29.87	77.52
Avg. Num. of Edges	19.79	72.82	14.69	715.66	32.30	198.32
Node Labels	+	+	+	+	+	+
Node Attr. (Dim.)	-	+(1)	-	-	-	-
Num. of Classes	2	2	2	2	2	20

V. EXPERIMENT

In this section, we will give the details of the datasets used for validation, the experimental setup, and the baseline methods. Our algorithms are implemented with PyTorch Geometric [24] and run on a NVIDIA TITAN RTX graphic card.

A. Experimental Datasets

The benchmark datasets for graph classification, including MUTAG [25], PROTEINS [26], PTC_MR [25], D&D [17], NCI1 [17] and MSRC_21 [27], are to do multi-class or binary classification. MUTAG [25] is a mutagenic aromatic and heteroaromatic nitro compounds dataset and their graph label indicate whether the mutagenic effect on bacteria exists; PROTEINS [26] represents protein structures which are helix, sheet and turn; PTC_MR [25] is a chemical compounds dataset which represents the carcinogenicity for male and female rats; D&D [17] is a dataset of 1178 protein structures, where the nodes in each graph represent amino acids, and the prediction task is to classify the protein structures into enzymes and non-enzymes; NCI1 [17] is also a chemical compound dataset that is used for activity against non-small cell lung cancer cell lines. MSRC_21 [27] dataset is a real-world image dataset where the nodes of each graph are derived by over-segmenting the images resulting in one graph among the superpixels of each image. The statistics of these datasets are shown in Table II. For all benchmark datasets, the categorical node labels are used as initial node feature vectors and node attributes are not fed into models.

B. Experimental Setup

Experimental configurations will be presented in the following. Empirically for graph benchmark datasets, we perform 10-fold cross-validation which we sampled 10% of each training fold to serve as a validation set and report the average accuracy and standard deviation of 10-fold validation. To ensure a fair comparison, we set the number of network layers T to be 5 and perform the final prediction via a fully-connected layer. The training batch size is fixed to be 128 and the dimension of hidden units is 32. We use Adam [28] optimizer with an initial learning rate of 0.01 and decay the learning rate by 0.5 every 50 epochs. Each nonlinear activation function follows the Rectified Linear Unit (ReLU) [29]. All the learnable parameters are initialized by Glorot Initialization [30] and the bias is initialized by zero vector. To make the model more stable, we utilize the Batch Normalization in each hidden layer and apply a fixed dropout rate of 0.5 after readout phase.

C. Baseline Methods

In our evaluation, we develop our architectures based on the powerful graph classification method GIN [22], and compare our models with GIN [22] and existing graph pooling methods ranging from Set2Set [31], GraphSAGE [12], SortPool [32] to Top-k pooling [20]. To further demonstrate the effectiveness for other message passing schemes, we utilize graph convolutional network (GCN) [11] as message passing function. Here, the readout functions are set to original readout, layer aggregation readout, pooling with self-loop, and gated pooling with self-loop. For all baseline methods, we use the same hyperparameter settings to ensure fair comparisons. We present the detailed baseline descriptions as follow:

1) GIN: Graph isomorphism network (GIN) [22] is a powerful GNN under the message passing framework which only utilizes multi-layer perceptrons and summation pooling. In our evaluation, we set the parameter $\epsilon = 0$.

2) GCN: Graph convolutional network (GCN) [11] employs the Laplace matrix to normalize the adjacency matrix and allow the aggregate and update function to share the same structure by adding self-loop.

3) Set2Set: Set2Set [31] is a powerful global pooling operator based on iterative content-based attention. Here, we set the number of iterations is 4.

4) GraphSAGE: The graph sample and aggregate (GraphSAGE) [12] model develops the local neighborhood sampling strategy and then the aggregate functions, including mean, sum and max function, to generate the representations of sampled nodes. Here, we choose the sum function as aggregate function.

5) SortPool: SortPool [32] sorts graph nodes in a consistent order so that traditional neural networks can be trained on the graphs. We set the k of SortPool is 10 and the other settings are same with the proposed methods.

6) Top-k pooling: Top-k pooling [19], [20] selects the top-k nodes from a given input graph. We set the graph pooling ratio is 0.8 and the layer aggregation is used to aggregate the layer-wise graph embedding outputs.

D. Experimental Results

In this subsection, we show the classification results and analysis to investigate the performance of the proposed meth-

TABLE III
COMPARISON OF THE 10-FOLD CROSS VALIDATION AVERAGE CLASSIFICATION ACCURACY (±STANDARD DEVIATION) ON DIFFERENT BENCHMARK DATASETS. LA MEANS LAYER AGGREGATION

Method	**Dataset**					
	MUTAG	PROTEINS	PTC_MR	D&D	NCI1	MSRC_21
Set2Set	83.9±7.2	77.9±3.6	56.2 ±8.5	60.5±5.6	70.3±9.5	90.2±4.8
GraphSAGE	82.2±9.6	76.4±3.9	54.7±7.2	74.7±5.6	62.6±1.1	92.0±2.0
SortPool	85.8±1.7	76.9±4.8	54.1±9.4	74.6±8.3	64.1±4.6	74.6±9.2
Top-k Pooling	87.8±6.9	77.7±4.0	59.1±6.2	77.2±2.8	78.8±1.8	93.8±2.0
GCN	87.8±6.5	76.9±3.5	62.9 ±6.3	78.8±2.0	77.3±1.5	89.8±2.8
GCN with LA	88.6±5.0	78.9±3.0	65.6±5.9	76.9±2.8	76.1±1.6	91.7±2.6
GCN with Self-loop	88.6±5.0	79.2±3.3	63.5±5.4	81.2±2.2	81.3±1.8	95.2±2.7
Gated GCN Pooling	**90.6±5.4**	**79.4±3.2**	**65.4±6.3**	**81.9±1.5**	**81.5±1.5**	**95.5±2.6**
GIN	88.1±4.2	75.1±3.8	62.1±4.4	77.3±1.5	77.5±2.0	91.9±1.8
GIN with LA	90.7±3.7	76.2±2.8	64.4±5.9	78.4±2.1	78.4±2.1	94.1±2.3
GIN with Self-loop	90.7±3.7	79.4±4.2	64.7±3.7	81.1±2.1	82.2±1.4	95.5±1.8
Gated GIN Pooling	**91.7±3.7**	**80.5±3.2**	**66.8±5.3**	**82.2±2.4**	**82.7±1.6**	**95.8±2.3**

ods. Table III shows the classification results for comparison with baselines on the graph classification benchmark datasets. We notice that proposed pooling methods achieve the classification performance improvement on all datasets compared with baselines, indicating the effectiveness of our approaches. Compared all proposed methods with pooling baseline methods, we can find that all proposed methods significantly outperform the pooling baseline methods on all datasets. Compared GIN and self-loop scheme, we can find that the self-loop scheme shows superior performance improvement on predictive accuracy and proves the effectiveness of the proposed self-loop strategy. Also compared self-loop scheme with gated graph pooling, the gate enhanced method consistently beat the self-loop scheme on all datasets, which further illustrates the effectiveness of the developed fusion gate. Besides, compared GCN with layer aggregation and GCN with self-loop (or gated pooling), we can see that GCN with self-loop improves the performance in comparison to layer aggregation architecture. The experiment results illustrate the scalability of proposed architectures which means combining our framework with other message passing based architectures can improve these models performance.

E. Sensitivity Analysis

To investigate the influence of hidden unit dimensions to our gated graph pooling model, the number of dimensions, $\{8, 16, 32, 64\}$, are orderly employed to evaluate performance on all graph classification datasets during the training period. We used the same other hyperparameter settings to ensure a fair comparison. Intuitively, Figure 4 shows that the 10-fold cross-validation results for different numbers of hidden units. Several observations can be received. First, the performance gradually improves if we increase the number of hidden units, then the performance tends to be saturated when hidden units to be 32. Second, the results indicate that the accuracy does not increase significantly as the dimension increased for some datasets when the dimension is greater than 32, *e.g.*, PTC_MR, MUTAG and PROTEINS.

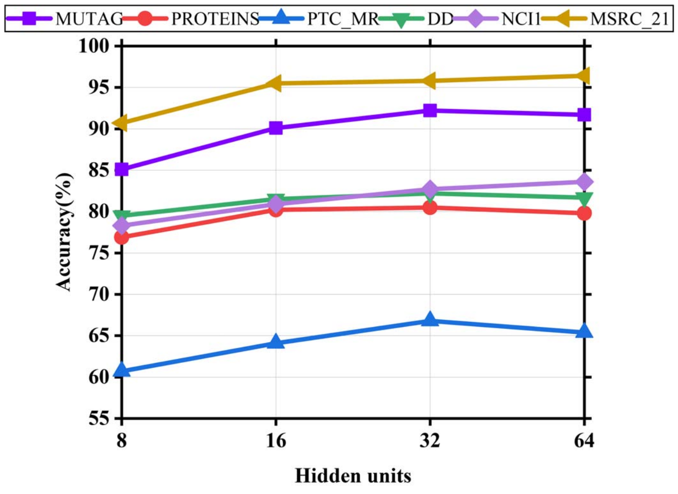

Fig. 4. Performance with a varying number of hidden unit dimensions on graph benchmark datasets. For clarity, we do not present the error bars.

F. Visualization of Gate Values

To illustrate the proposed fusion gate concretely, we conduct an experiment to visualize the gate values of the self-loop and the current step output for each layer on the MUTAG dataset. For the learned gate matrix, we compute the average weight as the gate value. Figure 5 shows the heat-map for all hidden layers. From Figure 5, we can observe that the self-loop contributes differently to the output at each layer. It illustrates the fusion gate can be learned to adaptively assign different importance to self-loop.

VI. CONCLUSION

In this paper, we first propose a novel self-loop graph pooling strategy that can utilize the node information of the current step and the graph representation information of the previous step to generate an effective representation for the graph classification task. Further to measure the importance of self-loop, we also develop a gated approach, gated graph pooling

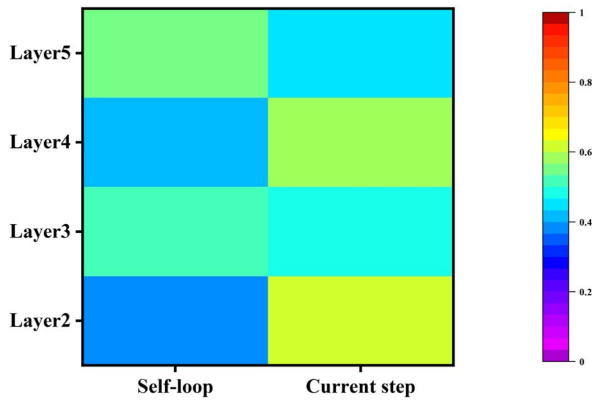

Fig. 5. Visualization of the gate values of the self-loop and the current step output for each layer on the MUTAG dataset.

with self-loop, that utilizes the simple fusion gate to enhance the representation capacity of the model. To illustrate the effectiveness of the proposed methods, we conduct a series experiments including classification accuracy comparison, the comparison of the dimension of hidden units, and visualization of gate values for fusion gate. Experimental results have demonstrated that our proposed methods can improve the performance on graph classification task.

In the future, there are two interesting works can be done: (1) exploring feasible ways to successfully aggregate the self-loop representations in readout phase. (2) developing high-order graph neural networks with self-loop that have stronger representation capability.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.

[3] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[4] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proceedings of the IEEE international conference on data mining (ICDM'05)*. IEEE, 2005, pp. 8–pp.

[5] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Proceedings of the Advances in Neural Information Processing Systems*, 2015, pp. 2224–2232.

[6] D. H. P. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, "Polonium: Tera-scale graph mining and inference for malware detection," in *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 2011, pp. 131–142.

[7] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *Proceedings of the ACM International Conference on Web Search and Data Mining*. ACM, 2011, pp. 635–644.

[8] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks." in *Proceedings of the International Conference on Learning Representations*, 2016.

[9] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *ArXiv Preprint arXiv:1406.1078*, 2014.

[10] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the International Conference on Machine Learning*, 2017, pp. 1263–1272.

[11] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations*, 2017.

[12] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[13] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proceedings of the International Conference on Machine Learning*, 2018, pp. 5449–5458.

[14] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[15] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning theory and kernel machines*. Springer, 2003, pp. 129–143.

[16] N. Shervashidze and K. Borgwardt, "Fast subtree kernels on graphs," in *Proceedings of the Advances in Neural Information Processing Systems*, 2009, pp. 1660–1668.

[17] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[18] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[19] H. Gao and S. Ji, "Graph u-nets," in *Proceedings of the International Conference on Machine Learning*, 2019, pp. 2083–2092.

[20] B. Knyazev, G. W. Taylor, and M. Amer, "Understanding attention and generalization in graph neural networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2019, pp. 4204–4214.

[21] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proceedings of the Advances in Neural Information Processing Systems*, 2018, pp. 4800–4810.

[22] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proceedings of the International Conference on Learning Representations*, 2019.

[23] T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, and C. Zhang, "Disan: Directional self-attention network for rnn/cnn-free language understanding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[24] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *Proceedings of the ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[25] N. M. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," in *Proceedings of the International Conference on Machine Learning*, 2012.

[26] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," in *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*, 2005, pp. 47–56.

[27] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: efficient graph kernels from propagated information," *Machine Learning*, vol. 102, no. 2, pp. 209–245, 2016.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2015.

[29] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the International on Machine Learning*, 2010, pp. 807–814.

[30] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[31] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: sequence to sequence for sets," in *Proceedings of the International Conference on Learning Representations*, 2015.

[32] M. Zhang, Z. Cui, M. Neumann, and C. Yixin, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, pp. 4438–4445.