DAGCN: Dual Attention Graph Convolutional Networks

Fengwen Chen *, Shirui Pan†, Jing Jiang*, Huan Huo‡ Guodong Long*
* Centre for Artificial Intelligence, FEIT, University of Technology Sydney, Australia
† Faculty of Information Technology Monash University, Australia
‡ School of software, FEIT, University of Technology Sydney, Australia
Email: fengwen.chen@student.uts.edu.au, shirui.pan@monash.edu, jing.jiang@uts.edu.au
huan.huo@uts.edu.au, guodong.long@uts.edu.au

***Abstract*—Graph convolutional networks (GCNs) have recently become one of the most powerful tools for graph analytics tasks in numerous applications, ranging from social networks and natural language processing to bioinformatics and chemoinformatics, thanks to their ability to capture the complex relationships between concepts. At present, the vast majority of GCNs use a neighborhood aggregation framework to learn a continuous and compact vector, then performing a pooling operation to generalize graph embedding for the classification task. These approaches have two disadvantages in the graph classification task: (1)when only the largest sub-graph structure (k-hop neighbor) is used for neighborhood aggregation, a large amount of early-stage information is lost during the graph convolution step; (2) simple average/sum pooling or max pooling utilized, which loses the characteristics of each node and the topology between nodes. In this paper, we propose a novel framework called, dual attention graph convolutional networks (DAGCN) to address these problems. DAGCN automatically learns the importance of neighbors at different hops using a novel attention graph convolution layer, and then employs a second attention component, a self-attention pooling layer, to generalize the graph representation from the various aspects of a matrix graph embedding. The dual attention network is trained in an end-to-end manner for the graph classification task. We compare our model with state-of-the-art graph kernels and other deep learning methods. The experimental results show that our framework not only outperforms other baselines but also achieves a better rate of convergence.**

I. Introduction

Graph structured or network data are rapidly becoming ubiquitous in our daily lives, e.g., World Wide Web network, transportation networks, and protein interaction networks. Researchers have conducted extensive research on many important machine learning applications in graph with both supervised and unsupervised fashion [1], such as vertex classification [2], anomaly detection [3], link prediction [4] and recommendation system [5], but the complexity of graph data imposes great challenges for many tasks including one of the central tasks in the field, graph classification (but not $node$ classification), which aims to assign a class label to an entire graph. In a cheminformatics dataset, for instance, atoms are represented by graph nodes and chemical bonds are represented by graph edges. A graph classification model can be applied to a dataset for many applications, from detecting molecular status, such as cancer activity detection or solubility detection, molecular properties, such as toxicity detection.

To solve the problem of graph classification, the most widely used strategy consists of graph statistic-based methods which are able to represent the graph in various aspects. Graph kernel [6], [7] is the most popular of these techniques; it employs a kernel function to measure the positive semi-definite graph similarity between pairs of graphs [8]. The classification task can then be conducted on a similarity matrix by using supervised algorithms like Support Vector Machine [9]. By decomposing the graph into sub-structures, the graph kernel is capable of directly processing the graph data without transforming it into feature vectors. As a result, it has achieved dramatic success in node classification, link prediction, node clustering and so on.

Graph kernel-based algorithms nevertheless still suffer from natural limitations, such as the exponential growth of computation operations and the fixed feature design, which will be discussed in more detail in Section IV. Other algorithms [10] attempt to distinguish and select the sub-graph features for graph classification by recursively applying an aggregation process on each node with the attributes from local neighbors to learn the node representations. The graph feature is then generated according to all the learned node representations in the graph.

Deep learning-based approaches like graph neural network have also been applied diffusely for network representation. These approaches embed the given graph and the side information associated with it into a continuous and compact vector space. After embedding, the graphs sharing common patterns are expected to be close to each other in the vector space, therefore classical machine learning methods can be applied to the embedded vector for graph classification. However, while the graph-structured data preserves more relational information than other data formats, it also incurs more complicated noise. How to learn a good representation while screening out the interference caused by the complex noise of each node in a graph has become a significant challenge. Moreover, sub-graphs which consist of multiple nodes, or even the entire graph are required in the graph classification task to achieve a more comprehensive analysis. Hence, obtaining the graph representation based on node representation is another non-

negligible challenge.

Many researches have concentrated on re-factorizing neural network architectures to directly process structured graph data [11]–[15]. However, graph data are complex in many ways; for example, the topological structure information of different sub-graphs is fickle when the size is varied. Most existing graph neural network frameworks are limited by two factors when dealing with this scattered information because: 1) these frameworks ignore the significance of different hop neighbors. Only the final aggregation output is used, i.e., only the largest sub-graph is used to learn the node representation. 2) they mainly apply average/sum pooling or max pooling which fails to leverage the valuable information of a node or sub-graph in the graph. While conducting graph classification, we attempt to pay more attention to the graph signature [16] (i.e. the special node or sub-graph), which is only a small segment of the entire graph. In contrast, a simple average/sum pooling or max pooling could result in a model that is constructed on too much irrelevant information.

To address the above problems, we propose a novel framework named Dual Attention Graph Convolution Network (DGCNN). The core idea of the proposed DGCNN is to identify and maximize the importance of the nodes or sub-graph when conducting graph classification. We first merge the attention technique in the graph convolution operation to capture the arbitrary local structure information in a graph. A self-attention pooling layer then generates an adaptive combination representation matrix, in which each row in the learned matrix represents one perspective of the graph. Our contributions in this paper are threefold:

- We propose a novel attention graph convolution technique which is capable of leveraging the information from different hop neighbors rather than the k-hop only;
- We propose a novel graph self-attention pooling technique which extracts a more informative embedding matrix containing multiple significant nodes or sub-graphs;
- We conduct experiments and compare our method with both deep learning-based methods and graph kernel-based algorithms. The experiment results demonstrate that the proposed Dual Attention Graph Convolutional Network (DAGCN) outperforms the deep learning benchmarks for graph classification and are highly comparable with state-of-the-art graph kernels.

II. Related Work

There have been many attempts on graph classification tasks in the literature. The earliest experiments can be traced back to 1998 when Frasconi et al. [17] used a recursive neural network to process directed acyclic graphs. Subsequently, Gori et al. [18] introduced Graph Neural Networks (GNNs) to extend the neural network for graph-structured data. GNNs normally consist of an aggregation process which aggregates the node features a certain number of times or until equilibrium is reached to produce an embedding for each node. This idea has been broadly adopted and improved in many tasks [19]–[21].

With the great success of computer vision, there is an increasing interest in generalizing convolutions to the graph domain. Bruna et al. [22] first generalized the convolution operation to the graph's spatial domain after the original data have been transformed by Graph Fourier Transform (GFT). Since the computation of eigenvectors is involved, computational complexity has become a serious issue. Many researchers have worked on optimizing the convolution filters to reduce the computational complexity [13], [19], [23], [24]. However, the learning process in all the aforementioned spectral approaches usually depends on the Laplacian eigenbasis, which handles the entire graph at one time. Thus, the issue of scalability and computational complexity still cannot be overcome.

Duvenaud et al. [25] introduced a spatial GCN that directly defines the convolutions on a graph without a transform. Each node propagates the features from its 1-hop neighbors to generate a differentiable fingerprint which simulates the circular fingerprints.After Kipf et al. [26] simplified the concept, Atwood et al. [27] extended this idea by propagating n different hops to the center node with different weights. A common challenge of these approaches is how to define the range of neighborhoods to aggregate and the strategies for obtaining information from neighbors. More recently, Niepert et al. [28] and Hamilton et al. [29] addressed the challenge in another way by sampling a fixed-size neighborhood for each node and then performing the aggregation. Lately, Tran et al. [30] further optimizing GCNs by extending the basic graph convolution operator. These approaches have achieved high levels of performance and have increased the scope of GCN applications. Given rapid developments in the field of GCN, we point readers to our recent, comprehensive review in [31].

An important component that usually comes with CNNs, the pooling layer, can also be generalized to graph-structured data. It is a down-sampling strategy that largely reduces the spatial size of the input while roughly retaining its location relationships. Mean pooling is the most commonly used graph pooling strategy due to its conciseness. Easily mean all node's information could also solve the issues of rotational invariance and yield better performance [19]. To better preserve the relationship between nodes, Defferrard et al. [13] and Zhang et al. [32] proposed approaches that perform pooling after the nodes have been rearranged in a meaningful order using a different strategy. This could be viewed as selecting similar parts of different graphs so that the preserved node relation can be used effectively. Overall, the essence of pooling is to reduce the size of the input (usually the node representation) by losing some information. Deciding which information to retain is the key to the model.

Attention mechanisms have already become the standard in many fields for a number of tasks [33], [34]. The most important advantage of attention mechanisms is that they are able to handle the variably sized inputs by focusing on the most relevant parts of the inputs to make decisions. When attention is implemented on the same input, it is called Self-Attention [35]. There is little literature on the topic of

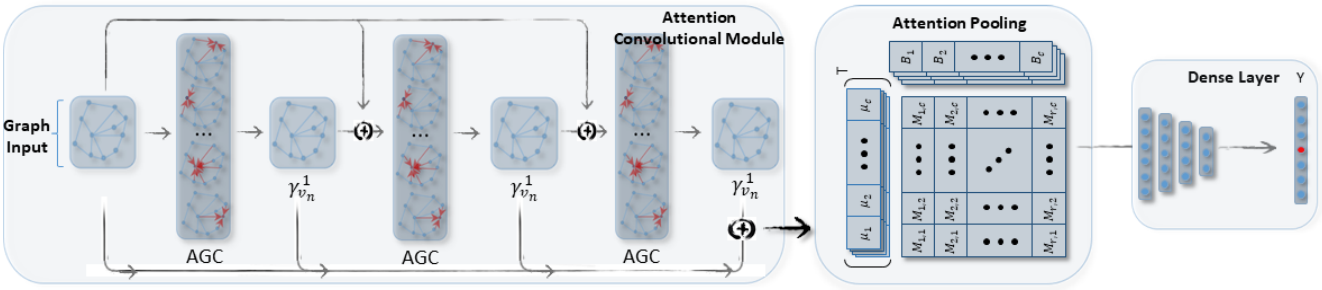

Fig. 1. The architecture of the dual attention graph convolution network (DAGCN). The model consists of three parts: (1) The left tier is the attention graph convolution module with three AGC layers ($m = 3$) which learns the hierarchical local substructure features by aggregating the hops of its neighbors. (2) The middle part is the attention pooling layer, the matrix B is the attention coefficient matrix. (3) The final graph embedding matrix M is then sent to a dense layer for final predictions.

attention mechanisms on graph-structured data. Velickovic et al. [14] employed attention to dynamically compute the weight of each node's neighbors during aggregation. Attention mechanisms have mainly been used in aggregation processing. A few attempts have been made to extend attention beyond aggregation [16], [36], but some issues have still never been studied. Inspired by recent works and the defect of them, we propose our model which uses an attention technique to maximize the use of information that underlies the original graph input.

III. Problem Definition and Framework

A graph is represented as $g = (V_g, E_g, A_g, X_g)$, where V_g is a vertex set v_i, i = 1, ..., n. E_g represents the linkages between nodes, denoted as $e_{i,j} =< v_i, v_j >\in E, i \neq j$. An unweighted adjacency matrix $A_g \in \{0,1\}^{N_i \times N_i}$ represents the graph's topological structure by setting $A_{i,j} = 1$ if $e_{i,j} \in E_g$, otherwise $A_{i,j} = 0$. N_i is the size of the graph g_i. $X \in \mathbb{R}^{n \times c}$ indicates the c channel content features associated with each node v_i.

Given a set of graphs $G = (g_1, g_2, \cdots g_n)$ with their labels $Y = (y_1, y_2, \cdots, y_n)$, the **goal** of our paper is to learn a function $f(g_i) \rightarrow y_i \in L$, where $L = \{c_1, \cdots, c_{|L|}\}$ is the class labels for the graphs. In this paper, we will develop a novel graph convolutional network which employs dual attentions at both node level and graph level, for graph classification.

A. Overall Framework

Our objective is to learn a classifier which could classify the given graph G. To achieve this, we propose a novel dual attention graph convolution network (DAGCN). Figure 1 demonstrates the work-flow of DAGCN which consists of two modules: the attention graph convolution module and the attention pooling module.

- **Attention Graph Convolution Module** The attention graph convolution module is constructed of several attention graph convolution layers. Each layer takes the features X and adjacency matrix A to extract the hierarchical local substructure features of the vertices from different hops of neighbor.
- **Attention Pooling Layer** The attention pooling layer uses the nodes' embedding to learn multiple graph representation from different aspect and outputs a fixed size, matrix graph embedding.

IV. Dual Attention Graph Convolution

The DAGCN consists of three parts: (1) the attention graph convolution module; (2) the self-attention pooling layer; and (3) the fully connected classifier. In this section, we first address the problem of traditional GCNs and, then propose our attention graph convolution module and self-attention pooling layer.

A. Traditional Graph Convolution

We start by describing the traditional graph convolution layer and then propose DAGCN to address the shortcomings. The most general form of graph convolution with depth of k can be expressed recursively by a broadly followed convolution structure denoted as:

$$H^{k+1} = \phi(\widetilde{A}\widetilde{D}^{-1}H^kW) \qquad H^0 = X, \tag{1}$$

where $\widetilde{A} = A + I_n$ is the adjacency matrix with self-connection for each node, $\widetilde{D}$ is the diagonal node degree matrix of $\widetilde{A}$, $\widetilde{A}\widetilde{D}^{-1}$ represents the normalized graph structure, and W is the model parameter that will be trained. After applying this operation k times, H^k becomes a node properties vector that contains k-hop local structure information.

Note that, during the repetition of Equation 1, with the exception of H^k, the result in every step can only be used to generate the next convolution result. During this process, a large amount of information will be lost, and only the last convolution result H^k, which represents the largest sub-graph, could be used for later tasks. This kind of operation can cause a significant loss of information. Only the k-hop local structure would be captured by the convolutional layer. Our attention convolution layer aims to solve this issue by attentionally

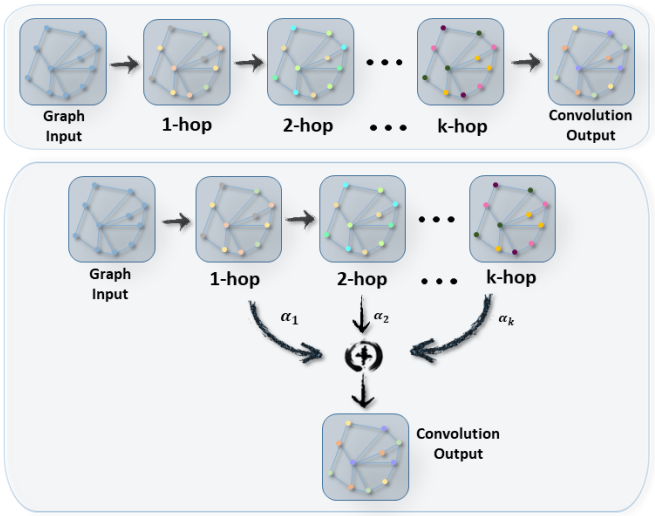

Fig. 2. Traditional Graph Convolution Layer (up): Only the final output which contains the largest sub-structure (k-hop neighbor substructure) is used. Attention Graph Convolution Layer (down): valuable information is extracted from every convolution step to generate a hierarchical node representation.

aggregating the information from each convolution step. The comparison of two graph convolution layers is shown in Figure 2.

B. Our Proposed Attention Graph Convolution (AGC)

The vast majority of graph neural networks are currently driven by Equation 1 which employs k-hop message aggregation mechanism. This enables the node representation to capture the local structural information of k-hop neighbors, but as the number of layers increases, a large amount of early information is lost during each convolution step, which severely affects the final prediction output and also limits the capacity of the model. The core idea of our attention graph convolution (AGC) layer is to enhance the model to not only depend on the k-hop convolution result, but also to capture valuable information from every single hop. The convolution result will thus be a hierarchical representation containing the most valuable information from different hop convolution processes. We exhibit attention behavior and implement it on Equation 1 to form a hierarchical node representation γ_{v_n} as below:

$$\gamma_{v_n} = \sum_{i=1}^{k} \alpha_i H_{v_n}^k \tag{2}$$

For simplicity, we use vanilla attention to identify the importance of each hop's aggregation result, in which α is the attention weight and $H_{v_n}^k$ represents node v_n's local structure in $k-hops$. The final node representation contains the hierarchical structure information. Figure 2 compares the traditional convolution layer and the attention convolution layer.

To maximize the advantages of deep learning and learn deeper latent features, we use the Residual Learning technique [37] to stack m attention convolution layers and develop an attention graph convolutional module to obtain a better final node representation γ_{v_n}. The input of each AGC layer is the sum of the previous layer's output and the original X. Lastly, we use a dense layer to process the combination of outputs from each convolution layer, illustrated as the Attention Graph Convolution Module in Fig 1.

$$\gamma_{v_n}^{m+1} = \sum_{i=1}^{k} \alpha_i H_{v_n}^k \qquad H_{v_n}^0 = \gamma_{v_n}^m + X \tag{3}$$

$$\gamma_{v_n} = \text{Dense}(\{\gamma_{v_n}^0, \gamma_{v_n}^1, ..., \gamma_{v_n}^m\}, \theta) \tag{4}$$

where $Dense()$ is a dense layer that combines the outputs from every attention graph convolution layer. We now have the node representation γ for all vertices $v \in G$. For simplicity, we denote the graph as a matrix G with size n-by-c where each row is a node's representation.

$$G = (\gamma_{v_1}, \gamma_{v_2}, ..., \gamma_{v_n})$$

C. Self Attention Pooling

To perform graph classification task, we would like to generate the graph-level representation from the node's representation. Most previous works use mean/max pooling [19] or sort pooling [13], [32] to generate a graph representation vector by aggregating all node representation vectors. We believe that simple max/mean pooling or pooling after the sort is ineffective and unnecessary, and therefore propose a self-attention pooling layer as a replacement. The goal is to encode an arbitrary graph into a fixed size embedding matrix while maximizing the information underlying the nodes' representation. Figure 3 presents a sample model showing how a coefficient matrix is generated for the attention pooling layer.

We use the attention mechanism by taking the graph node representation learned from the convolution module as the input to output the weights vector α.

$$\beta = \text{softmax}(u_2 \text{tanh}(u_1 G^T)) \tag{5}$$

In this equation, u_1 and u_2 are weight matrices with the shape of c-by-c and c-by-r respectively, where r is a

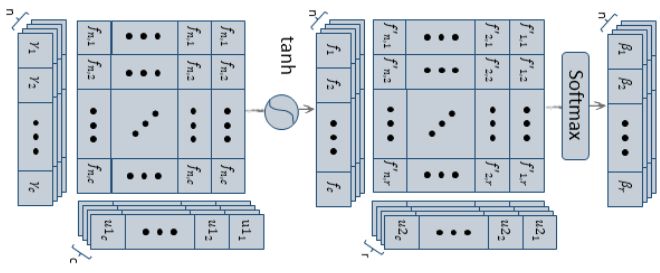

Fig. 3. Process of generating Self-Attention Pooling coefficient matrix.

hyperparameter that we set for the number of subspaces to learn the graph representation from the node representation. When $r \geq 1$, α becomes a weight matrix instead of a vector, and Equation 5 can then be written as

$$B = \text{softmax}(u_2 \text{tanh}(u_1 G^T)) \tag{6}$$

Each row of B represents one node's weight in a different sub-space. The $softmax$ function is performed along the second dimension of its input. We then conduct a weighted summation according to B from Equation 6 to obtain the graph representation matrix M with shape n-by-r.

$$M = B\dot{G} \tag{7}$$

Algorithm 1 Procedure of DAGCN

Input:

T: Iterations for updating.
A: Unweighted adjacency matrix;
v_n: Feature vector of node V_n
K: The number of hops for convolution operation;
M: The number of attention graph convolution layers

Output:

Y: Prediction outcome.

Model initialization. $k, m \Leftarrow 0$, $H^0_{v_n} \Leftarrow v_n$
for iterator = 1, 2, 3, ..., T **do**
  for $m = 1$ **to** M **do**
    for $k = 1$ **to** K **do**
      K-hop graph convolution. Equation (1)
    end for
    Attention aggregation from each hop. Equation (2)
    Store the result and prepare input for next layer. $\gamma^m_{v_n} \Leftarrow H_{v_n}, H^0_{v_n} \Leftarrow H_{v_n} + X$
  end for
  Generalize final node representation γ_{v_n} for node $v_n \in g$. Equation (4).
  Generalizing coefficient matrix for attention pooling layer. Equation (6)
  Weighted sum over graph g. Equation (7)
  Update the all weight parameters with stochastic gradient.
end for
return $Y \in \mathbb{R}^{n \times |C|}$ Equation (8)

We now have a graph representation matrix in which each row is a graph representation in one sub-space, and the overall matrix produces a comprehensive representation for the graph. Lastly, a fully-connected layer followed by a softmax layer takes M as the input to accomplish the graph classification.

$$Y = softmax(ZM + C) \tag{8}$$

We thus obtain the final classification result Y. The step algorithm is summarized in Algorithm 1

V. Experiments and Results

We construct two sets of experiments to evaluate DAGCN with both graph kernel and GCNs methods in a graph classification task. Both experiments are based on several popular benchmark datasets. The reported result shows that DAGCN outperforms the state-of-the-art deep learning methods and yields a competitive result compared to graph kernels. Details of the code and data are available at https://github.com/dawenzi123/DAGCN

A. Datasets & Baselines

We use seven benchmark bioinformatics datasets to evaluate our DAGCN model according to the accuracy of the graph classification task. The datasets used are: NCI1, D&D, ENZYMES, NCI109, PROTEINS and PTC. Brief data information is listed in Table I, and a detailed dataset description can be found in [38]. For the baselines, we compare our framework with major families of graph kernels in the literature and some newly deep learning approaches. For the Graph Kernel Baselines, we compare DAGCN with five state-of-the-art graph kernels: a) Random Walk (RW) [39], b) Shortest Path Kernel (SP) [40], c) Graphlet Kernel (GK) [41], d) Weisfeiler-Lehman (WL) [7], and e) Deep Graph Kernels (DGK) [38]. In the same benchmark datasets, we also compare our DAGCN model with four deep learning approaches for graph classification. Because of the large amount of literature related to GCN, we could not compare every method. DCNN, PSCN, ECC and DGCNN are four recently proposed state-of-the-art GCNs which are most related to our approach.

TABLE I

Dataset	NCI1	D&D	ENZYMES	MUTAG	NCI109	PROTEINS	PTC
Nodes (max)	111	5748	126	28	111	620	109
Nodes (avg.)	29.80	284.32	32.60	17.93	29.60	39.06	25.56
Graphs	4110	1178	600	188	4127	1113	344

B. Graph Kernel Configuration

For the graph kernel parameter setting, the height parameters of WL and PK are chosen from the set $\{0, 1, 2, 3, 4, 5\}$. For the Random Walk (RW) kernel, we set the decay parameter as λ, following the suggestion in [7]. Results for the others were borrowed from previous works [28], [32], [38]. All the experimental setups were the same so that a fair comparison could be made.

For PSCN, ECC and DGCNN, we adopted the best results from the paper [42], since their experiment settings are the same as ours. For DCNN, we conducted the experiment based on the standard setting discussed below. For fairness, we also removed the edge features from all datasets, as most of the graph data were missing edge features and the methods we compared do not leverage edge features.

We attempted not to fine tune our model to improve performance. The same configuration with rough default values were shared between two sets of experiences. The hidden layer size for all dense layers and convolution layers was set to 64, k was chosen from sets $\{1, 5, 10\}$, and the chosen

number of hops was $k \in \{3, 5, 10\}$. For the general setting, we adopted the same procedure as previous works [32] so that a fair comparison could be made. We used the Adam [43] optimization policy with L2 regularization and learning rate selected from $\{0.01, 0.001, 0.0001\}$ to ensure the best play of the model. The batch size was fixed as 50, and 10-fold cross validation was implemented (9 folds for training, 1 fold for testing) to report the average classification accuracy and standard deviations.

C. Experimental Result

Table II shows the average classification accuracy of the compared deep learning methods."$-$" in the table means that either the source code is not available or the previous report did not contain a related result. From the results, we can see that our proposed model consistently outperforms all other methods on six of the seven datasets, and is second best on D&D. In particular, there is a 7% improvement in classification accuracy on NCI1 and more than 8% on NCI109, with a 1% - 3% accuracy gain on the other four datasets (excluding D&D). DAGCN outperforms DCNN and ECC in every case, proving our hypothesis that simple summing the node features is ineffective and will result in the loss of topology information. PSCN performs about the same as our model on PROTEINS and PTC but is much worse on NCI1 because it is more likely to overfit predefined node ordering. We avoid this problem by using attention pooling which dynamically learns the valuable node distributions over the graph. The improvement achieved by DAGCN can be explained as follows. 1) By using an attention mechanism to aggregate different hop neighbors, DAGCN is able to access more information underlying the graph input, thus achieving better performance. 2) By using the attention pooling layer, DAGCN is able to capture multiple graph signatures on the fly without losing any individual node or global topology information.

We also compare DAGCN with state-of-the-art graph kernels. The result in table III show that DAGCN is very competitive with state-of-the-art graph kernels. Our model is consistent among the top-2 in terms of performance on all datasets. This is a 1% - 3% improvement in accuracy on most datasets, with a high of 9% improvement for ENZYMES, compared with graph kernels other than WL.

VI. Case Study

The experiment results clearly demonstrate the classification performance of DAGCN compared with other deep learning GCNs. We also compare the efficiency of DAGCN with one of the most recent deep learning models, DGCNN, on NCI1, ENZYMES and NCI109, three benchmark datasets on which the learning process is observed to be relatively stable. Since the most significant learning process occurs in the early stages of training, we set the iteration number for both models on all datasets to 200. Although DAGCN has a Residual Learning structure to enhance performance, we limit the number of attention graph convolution layers m to 1 to make this comparison fair. The learning rate has the same setting as DGCNN's default, and all other parameters have the default setting previously mentioned. Figure 4 shows that DAGCN not only achieves better classification accuracy, but also has a better rate of convergence.

Compared with deep learning methods, DAGCN has obvious advantages over graph kernels. Although the overall state-of-the-art in the graph classification task is still dominated by graph kernels, DAGCN is the most practical in its ability to address efficiency and several other issues which most graph kernels suffer from.

Computational complexity. Graph kernels first need to compute the similarity between each two graphs in the training dataset to form a similarity matrix. Given a dataset of size N, then $N(N-1)/2$ computation steps are required. This number will grow exponentially when the size of the dataset is increased. In addition, calculating the similarity between a pair of graphs is also an exponential operation based on the number of nodes in the graph. This limits the power of the graph kernels only working for small data-set with small graph. By design, the computational complexity of DAGCN grows linearly for both the dataset size and graph size.

Static graph features. Graph kernels can also broadly be divided into two parts. First, a similarity matrix is constructed by the pre-defined kernel function, and a deep learning model then learns the classification rules. The two steps are independent of each other. The first step can be envisaged as human feature engineering, after which, the features are fixed and are not optimized during the training process. Similar datasets might share some common features as a result of common natural properties (i.e., two bio-informatics datasets). But datasets from different fields must have different properties (e.g., social network and protein network). Although our model is also created from two modules, it still an end-to-end model. All parameters will be optimized during the training process giving DAGCN more advantage on generality.

Single structure. Due to their nature, graph kernels can only focus on a certain scope of graph according to their kernel function. As a result, either global structure or local properties are lost. Our attention pooling layer enables us to learn hierarchical structure information that includes both local and global properties.

VII. Conclusion and Future Work

In this paper, we have proposed a novel Dual Attention Graph Convolutional Network (DAGCN) model with the core idea of maximally exploiting the original information underlying the graph input. We used an attention mechanism to address the weakness of traditional GCN models, in which information is largely lost in every convolution step. Our attention convolution layer design is capable of capturing more hierarchical structure information than other models and provides a much more informative representation of both individual nodes and the whole graph. The attention pooling layer generates a fixed size, comprehensive graph representation matrix by using a self-attention mechanism to focus on the different aspects of graph. The experimental results show that

TABLE II
COMPARISON WITH DEEP LEARNING METHODS

Dataset	NCI1	ENZYMES	MUTAG	NCI109	PROTEINS	PTC
DCNN	56.61±1.04	42.44±1.76	-	57.47±1.22	61.29±1.60	56.60±2.89
PSCN	76.34±1.68	-	-	-	75.00±2.51	62.29±5.68
ECC	76.82	45.67	-	75.03	-	-
DGCNN	74.44±0.47	51.00±7.29	85.83±1.66	75.03±1.72	75.54±0.94	58.59±2.47
DAGCN	**81.68±1.69**	**58.17±8.76**	**87.22±6.1**	**81.46±1.51**	**76.33±4.3**	**62.88±9.61**

TABLE III
COMPARISON WITH GRAPH KERNELS

Dataset	NCI1	ENZYMES	MUTAG	NCI109	PROTEINS	PTC
RW	-	24.16±1.64	79.17±2.07	>1 Day	74.22±0.42	57.85±1.30
SP	73.00±0.24	40.10±1.50	-	73.00±0.24	75.07±0.54	58.24±2.44
GK	62.28±0.29	26.61±0.99	81.39±1.74	62.60±0.19	71.67±0.55	57.26±1.41
WL	**82.19±0.18**	52.22±1.26	84.11±1.91	**82.46±0.24**	74.68±0.49	57.97±0.49
DGK	80.31±0.46	53.43±0.91	-	80.32±0.33	75.68±0.54	60.08±2.55
DAGCN	81.68±1.69	**58.17±8.76**	**87.22±6.1**	81.46±1.51	**76.33±4.3**	**62.88±9.61**

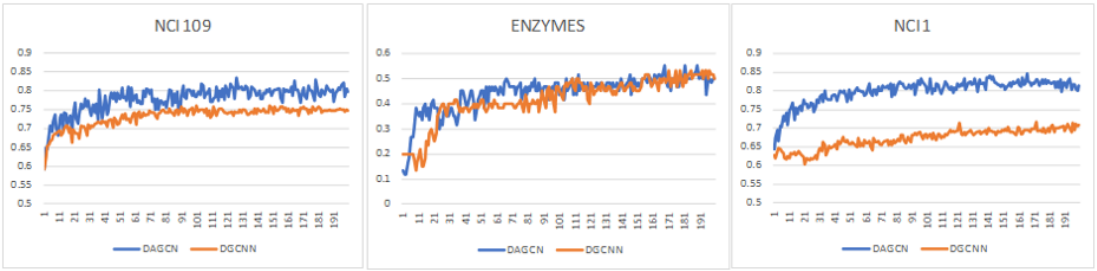

Fig. 4. Learning curve for DAGCN (blue) and DGCNN (orange)

our model outperforms other deep learning methods and most graph kernels in a range of datasets.

In future work, We intend to implement and validate our model on more complex graphs such as EHRs data and social networks. We will also analyze graph convolution in greater depth to discover how information is distributed at different convolution level. Lastly, we observe that it would be better to test a larger number of attention architectures to mimic the nature of the dataset, since our model only employs one basic attention architecture for all datasets.

ACKNOWLEDGMENT

This research was funded by the Australian Government through the Australian Research Council (ARC) under grants 1) LP160100630 partnership with Australia Government Department of Health and 2) LP150100671 partnership with Australia Research Alliance for Children and Youth (ARACY) and Global Business College Australia (GBCA). We acknowledge the support of NVIDIA Corporation and MakeMagic Australia with the donation of GPU used for this research.

REFERENCES

[1] D. Bacciu, F. Errica, and A. Micheli, "Contextual graph markov model: A deep and generative approach to graph processing," *arXiv preprint arXiv:1805.10636*, 2018.

[2] S. Chuang and M. R. Henderson, "Three-dimensional shape pattern recognition using vertex classification and vertex-edge graphs," *Computer-Aided Design*, vol. 22, no. 6, pp. 377–387, 1990.

[3] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 631–636.

[4] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," 2018.

[5] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens, "Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation," *IEEE Transactions on knowledge and data engineering*, vol. 19, no. 3, pp. 355–369, 2007.

[6] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.

[7] 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.

[8] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Artificial Intelligence and Statistics*, 2009, pp. 488–495.

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

[10] S. Pan, J. Wu, X. Zhuy, C. Zhang, and P. S. Yuz, "Joint structure feature exploration and regularization for multi-task graph classification," in *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 2016, pp. 1474–1475.

[11] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 3697–3707.

[12] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," *Network*, vol. 11, no. 9, p. 12, 2016.

[13] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[14] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, vol. 1, no. 2, 2017.

[15] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang, "Finding the best not the most: regularized loss minimization subgraph selection for graph classification," *Pattern Recognition*, vol. 48, no. 11, pp. 3783–3796, 2015.

[16] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1666–1674.

[17] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE transactions on Neural Networks*, vol. 9, no. 5, pp. 768–786, 1998.

[18] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 2. IEEE, 2005, pp. 729–734.

[19] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[20] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[21] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding." in *IJCAI*, 2018, pp. 2609–2615.

[22] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[23] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," *AAAI*, pp. 3546–3553, 2018.

[24] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *arXiv preprint arXiv:1705.07664*, 2017.

[25] 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 *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[26] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[27] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 1993–2001.

[28] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

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

[30] D. Van Tran, N. Navarin, and A. Sperduti, "On filter size in graph convolutional networks," *arXiv preprint arXiv:1811.10435*, 2018.

[31] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[32] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of AAAI Conference on Artificial Inteligence*, 2018.

[33] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[34] T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, and C. Zhang, "Disan: Directional self-attention network for rnn/cnn-free language understanding," *arXiv preprint arXiv:1709.04696*, 2017.

[35] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.

[36] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. A. Alemi, "Watch your step: Learning node embeddings via graph attention," in *Advances in Neural Information Processing Systems*, 2018, pp. 9197–9207.

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

[38] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.

[39] 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.

[40] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Data Mining, Fifth IEEE International Conference on*. IEEE, 2005, pp. 8–pp.

[41] R. Kondor, N. Shervashidze, and K. M. Borgwardt, "The graphlet spectrum," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 529–536.

[42] S. Verma and Z.-L. Zhang, "Graph capsule convolutional neural networks," *arXiv preprint arXiv:1805.08090*, 2018.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.