g-Inspector: Recurrent Attention Model on Graph

Zhiling Luo *IEEE Member*, Yinghua Cui, Sha Zhao and Jianwei Yin

***Abstract*—Graph classification problem is becoming one of research hotspots in the realm of graph mining, which has been widely used in cheminformatics, bioinformatics and social network analytics. Existing approaches, such as graph kernel methods and graph Convolutional Neural Network, are facing the challenges of non-interpretability and high dimensionality. To address the problems, we propose a novel recurrent attention model, called g-Inspector, which applies the attention mechanism to investigate the significance of each region to make the results interpretable. It also takes a shift operation to guide the inspector agent to discover the next relevant region, so that the model sequentially loads small regions instead of the entire large graph, to solve the high dimensionality problem. The experiments conducted on standard graph datasets show the effectiveness of our g-Inspector in graph classification problems.**

***Index Terms*—Graph Classification, Graph Mining, Recurrent Neural Network, Reinforcement Learning**

I. Introduction

As a prevalent data structure, graph has been widely used to model key features and complex relationships for complicated objects, such as molecular graph structures, biological protein-protein interaction networks and social networks [1]. Many complex problems in the areas of cheminformatics, bioinformatics and social network analytics can be formulated as graph problems and solved by leveraging graph mining techniques [2]. In this field, graph classification has been a main research branch and a hot topic as well, which assigns labels to graphs [3].

Graph classification is of highly practical significance in a wide variety of real-world applications, such as drug activity predictions, toxicology tests, and categorizing scientific publications [4][5][6]. For example, in molecular medicine, representing a chemical compound as a graph can help quickly judge whether the compound has the inhibiting effect on a specific cancer cell, which can significantly reduce time, efforts and excessive resource on drug testing [4][7].

Existing approaches for graph classification can be roughly divided into three categories. The first category is subgraph methods, which classifies graph object by subgraph patterns [3][8]. The second category is graph kernel which leverages kernel methods to compute the similarity among graphs [9][10]. The similarity is computed based on the entire graph, and all the subgraph structures in the graphs were treated fairly. It is ambiguous how each structure contributes to the classification. The third category is graph neural networks (GNNs), which applies deep learning based methods on graph domain [11][12][13] [14][15][16].It is hard to know the significance of each feature for the classification, since the features are automatically integrated through deep neural networks. Taken together, the methods in both of the categories cannot decide which structure is important to the graph classification, and thereby it is difficult to interpret the results. Consequently, it is hard to retrieve valuable knowledge from graphs to support many practical applications. Meanwhile, there is no clear boundary among multiple different classes.

Graph objects can be with up to billions of vertices and edges in practical applications, such as various social networks. Not all the subgraph structures are helpful for classification. It is natural to select the relevant structures and ignore the irrelevant ones so as to make results interpretable. Motivated by the idea, we introduce an *attention* mechanism to select a series of task relevant regions to pay attention to. In this task, a region refers to a neighborhood subgraph structure of a given central vertex.

Attention has become an essential part of deep learning models and achieved great success in various tasks, e.g. machine translation[17], image classification[18], caption generation[19] and speech recognition[20]. The attention mechanism helps to reduce the number of parameters, and improve computational efficiency[21]. Moreover, adopting the attention mechanism make results interpretable [19]. One can understand the process of the model by visualizing the regions where the attention is paid to.

Unfortunately, the attention mechanism cannot be directly applied for graph classification. In the aforementioned work, the data is well-structured and with determined and low dimensionality, such as image, text or video. However, the size of different graphs is variational, and a space with a high dimensionality equal to that of the largest graph is required to represent different graphs.

In this paper, to address the problem, we propose a novel recurrent attention model, called *g-Inspector*, where we introduce an inspector module applying the attention to investigate the significance of each region to classification. We also design a shift operation to guide the inspector to discover the next relevant region under the rule of reinforcement learning. By the recurrent process consisting of attention inspection and shift operation, we sequentially load small relevant regions instead of the entire graph, to avoid the high dimensionality problem. The main contributions of this paper are as follows:

- The recurrent attention model, g-Inspector, overcomes the challenge of high dimensionality problem. Moreover, it is interpretable owing to the ability of measuring the contribution of each region to the classification.
- We introduce a shift operation across the graph which works under the reinforcement learning rule. It adaptively selects a series of task relevant regions instead of searching the entire graph, and significantly prunes the search space.

Z. Luo, Y. Cui, S. Zhao and J. Yin are with the Department of Computer Science, Zhejiang University, Hangzhou, China, 310027.
E-mail: {luozhiling, yhcui, szhao, yjw}@zju.edu.cn

- The experiments conducted on public graph datasets verify the accuracy, scalability and parameter selection of the model.

To better illustrate our work, we organize the remaining part of this paper as follows. Section II briefly reviews the related researches from literature and summarizes their advantages and disadvantages in solving graph classification problem. Section III, formalizes this problem, introduces the g-Inspector model and its training method. Section IV reports our experimental result on open datasets. The conclusion and future work is discussed in section V.

II. Related work

In this section, we briefly review the literature on graph classification, which can be roughly divided into three categories: *subgraph methods*, *graph kernels* and *GNNs*. Besides, we take a quick look at the research studies on attention mechanism and its application on graphs.

As the first category, subgraph methods leverage subgraph patterns for graph classification. For example, gSpan explored depth-first search to mine frequent subgraphs[22]. gSSC performed semi-supervised feature selection for graphs in a progressive way together with the subgraph feature mining process[4]. Pan et. al. [3] introduced a boosting method, especially designed fro cost-sensitive classification. An incremental subgraph feature selecting technique is proposed in [8]. The major drawback of these subgraph methods is the limitation on scalability. The high-dimension challenge comes with the growth of graph size has not been explored well and has high computational complexity.

The second category on graph classification is graph kernel based methods. The original purpose of the kernels on graphs was to measure the similarity of nodes on a single graph[23]. Graph kernels allow kernel-based learning approaches such as SVMs to work directly on graphs. Some variants have considered that the similarity can be measured between different graphs[24], subgraphs or certain graphlets[10]. However, the features extracted from graphs that are used to compute the kernel matrix are not independent[9], and all graph kernels have a training complexity at least quadratic in the number of graphs which is infeasible for large-scale problems[25].

In the third category, graph neural networks(GNNs) extend existing neural network methods to work with graph-structured data. Diffusion CNNs[13] learned diffusion-based representations from graph-structured data by scanning a diffusion process across each node. NgramCNN [12] studied the Ngram on a graph adjacency matrix and diagonal convolution to extract subgraph features. PSCN[11] constructed the receptive fields by extracting locally connected regions from graphs. DGCNN[26] proposed an end-to-end deep learning architecture with a spatial graph convolution layer and a SortPooling layer. DiffPool[27] is a differentiable graph pooling module which can be combined with existing GNNs to extract the hierarchical representations of graphs. GNN is computationally expensive because it required many iterations between gradient descent steps, and the network's behavior is hard to explain.

Some studies have tried to apply the attention mechanism on graphs for graph representing [28][29]. Choi et al. proposed GRAM[28], an attention model for healthcare representation learning, which is based on medical ontology and only applicable on directed acyclic graphs. Petar et al. introduced GATS[29], graph attention networks which focused on node classification of graph-structured data by using attention to compute the hidden representation of each node. Lee et al. proposed MCN[30], a motif-based graph attention model for node classification task, which generalizes GCNs by using weighted multi-hop motif-induced adjacencies to capture higher-order neighborhoods. Compared with these studies, we use an attention mechanism for graph classification instead of using graph representation directly. Specifically, our model uses an inspector module to select attention regions instead of learning attention weights.

We have noticed the publication of another research applying attention on graphs, GAM [31], which is independent and simultaneous with ours. Coincidentally, both their proposed model GAM and our work are inspired by the recent work on the visual attention-based model, RAM[18]. However, GAM is significantly different from ours, and the main differences are as follows: (1) Our g-Inspector has better scalability. It integrates graph structure feature, vertex/edge attributes at the same time, while GAM can only handle the vertex attributes. (2) At each time step, our g-Inspector extracts a subgraph instead of visiting a single node in GAM. More subgraph information can be collected and inferred, which makes it possible for the inspector to gain more information about the graph and make better decisions. (3) In GAM, the step module takes a step from the current vertex to one of its one-hop neighborhood. Our g-Inspector takes the shift operation to expand the scope of observation rather than just a small portion of the graph.

III. Proposed method

A. Problem definition

A graph G is defined by a tuple $(V, E, \mathsf{f}_v, \mathsf{f}_e, \mathcal{S}_v, \mathcal{S}_e)$ where $V = \{v_i\}_{i=1}^{|V|}$ is the set of vertices and $E \subseteq V \times V$ is the set of edges. An edge in E connecting vertex v_i with vertex v_j can be denoted as (v_i, v_j), where $v_i, v_j \in V$. $\mathcal{S}_v$ is the set of vertex-label, or called node attribute, and $\mathcal{S}_e$ the set of edge-label, or called edge attribute, respectively. $\mathsf{f}_v : V \to \mathcal{S}_v$ is the assignment from a vertex to a vertex-label. Similarly, $\mathsf{f}_e : E \to \mathcal{S}_e$ is the assignment mapping from an edge to an edge-label. Both $\mathcal{S}_v$ and $\mathcal{S}_e$ can either be qualitative attribute or quantitative attribute. Consider that by modeling a social network as a graph, the vertex represents a person, and the edge represents the friendship relation. The person's gender, age and preference are the vertex-labels. A qualitative attribute is the gender, whose $\mathcal{S}_v$= {Male, Female, Others}. A quantitative attribute is the age, whose $\mathcal{S}_v$ is the integer number> 0. In this way, our problem framework covers both the qualitative and quantitative attributes.

With graph and basic elements formally defined, we can describe the following classification problem.

Definition 1 (Graph Classification) *Given a collection of graphs* $\mathcal{G} = \{G_1, \cdots, G_n\}$*, let* $\mathcal{L} = \{l_1, \cdots, l_K\}$ *denote the*

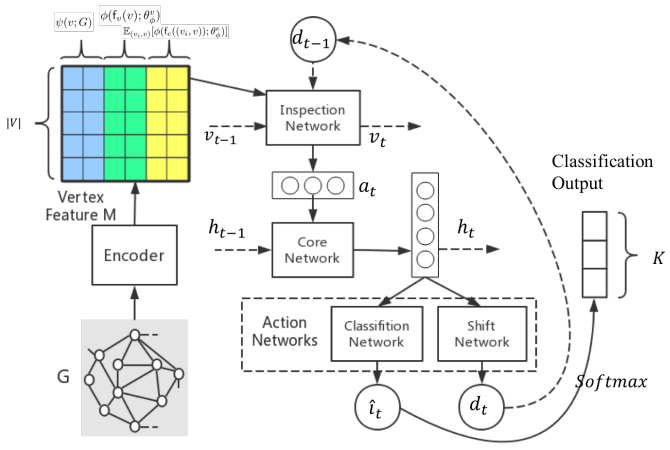

Fig. 1: Method overview

set of K class labels, learn an assigning function $f : \mathcal{G} \to \mathcal{L}$ that maps a graph to one class label in $\mathcal{L}$.

Graph classification tasks are associated with the whole graph, which aim to estimate the label of entire graph object. Like other classification problems, the key is to extract the features of input, namely the graph, and then quantify their contributions to the class labels. The challenges come from three aspects.

- **Feature Complexity Challenge** It is difficult to simultaneously cover two different kinds of graph features, one of which is the connection information from subgraph structures, and another the vertex/edge attributes. For example, GAM [31] can only handle the graphs with attributes. Our idea is to encode the attributes by merging vertex-labels, edge-labels and the graph embedding vectors together.
- **High Dimensionality Challenge** The size of a graph object can be very large. Loading the entire graph cause the high dimensionality problem. Our idea is to sequentially load small relevant regions of the graph via an inspection network.
- **Interpretable Ability Challenge** Most existing methods cannot interpret the results. In g-Inspector, we introduce an attention mechanism to measure the contribution of each region, making the results interpretable.

Facing these challenges, we combine together these possible ideas to build a complete model, called g-Inspector.

B. Method overview

The overview of our is shown in Fig. 1, which consists of four parts. (1) *Encoder* of graphs to embed the graph structure and vertex/edge-label. (2) *Inspection network*, containing the inspector to capture a part of the graph, and taking shift operation to efficiently shift the inspector on the graph. (3) *Core network* to keep the memory from previously observed subgraph features and merge the newly captured information by a recurrent cell. (4) *Action network* to guide the shift of the inspector and predict the classification label. These four parts are detailed illustrated one-by-one in the rest of this section.

C. Encoder

To overcome the feature complexity challenge, we leverage an encoder to merge the structural information with the attributes of vertexes and edges. For a graph G, each vertex v is encoded as a vector $M(v)$ consisting of three parts.

- $\psi(v; G)$ is the embedding vector of v in the graph G.
- $\phi(\mathsf{f}_v(v); \theta^v_\phi)$ is the encoding of the vertex attribute parameterized by θ^v_ϕ.
- $\mathbb{E}_{(v_i,v)}[\phi(\mathsf{f}_e((v_i, v)); \theta^e_\phi)]$ is the expectation on encoding of the edge attribute parameterized by θ^v_ϕ. (v_i, v) denotes any edge connecting v, and whose encoding is $\phi(\mathsf{f}_e((v_i, v)); \theta^e_\phi$.

In summary, given a vertex v in graph G, $M(v)$ represents the vertex feature with both structural information and attributes considered. Without considering the attributes of vertex or edge, we only take the structure of graph into account in practice. For the implementation, we employ DeepWalk as ψ, which can encode the relationship between the vertices in a continuous vector space with a relatively low dimensionality[32]. Other embedding algorithms, e.g. node2vec could be a good choice. The slightly promotion or demotion of accuracy caused by replacing different embedding algorithm such as replacing Deepwalk by node2vec is not our major technical contribution.

D. Inspection network

The inspection network is the essential component of our method, which observes a task relevant region of graph and shifts its central vertex under the instructions from the action network. A region refers to a neighborhood subgraph structure of a given central vertex.

1) Inspector: Given the whole graph G, the inspector collects the subgraph features from a region determined by a central vertex v. To specify the region of v, we leverage the concept of neighborhood.

Definition 2 (k-hop neighborhood) *$\forall v \in V$, let $\mathcal{N}_k(v)$ denote the k-hop neighborhood of vertex v. $\mathcal{N}_k(v)$ can be defined in a recursive form*

$$\mathcal{N}_k(v) = \begin{cases} \{v\} & \text{if } k = 0 \\ \{u | u \in (V - \bigcup_i^{k-1} \mathcal{N}_i(v)) \wedge \\ \exists v' \in \mathcal{N}_{k-1}(v), (u, v') \in E\} & \text{ow.} \end{cases} \quad (1)$$

With the help of neighborhood $\mathcal{N}_k(v)$, we can describe the region around v from variant granularity.

Definition 3 (k-order region) *For a vertex v in graph G, its k-order region is the vertex set $\mathcal{R}_k(v)$ which is also recursively defined:*

$$\mathcal{R}_k(v) = \begin{cases} \mathcal{N}_0(v) & \text{if } k = 0 \\ \mathcal{N}_k(v) \bigcup \mathcal{R}_{k-1}(v) & \text{ow.} \end{cases} \quad (2)$$

For the smallest granularity, the region has only one vertex, namely v itself.

To take use of these regions, given the central vertex v, the inspector mixes the vertex features from its k-order region in two ways: The first and straight-forward way is the *stacked*

k-*order region* which take average of all regions whose order is less than k.

$$\rho(v;G) = \sum_{i=1}^{k} \mathbb{E}_{v' \in \mathcal{R}_i(v)}[M(v')] \tag{3}$$

The second way is introducing the decay factor $\gamma_\rho \in (0,1)$ as the *stacked decay k-order region* and give the i-order region with weight γ_ρ^{i-1}.

$$\rho(v;G) = \sum_{i=1} \gamma_\rho^{i-1} \mathbb{E}_{v' \in \mathcal{R}_i(v)}[M(v')] \tag{4}$$

In practice, the former way is easy to implement by setting k as a constant but hard to generalize on different graph datasets. Therefore, by default, we take use of Eq. (4) and stop summation at n-order region if γ_ρ^n is smaller than a constant threshold.

In summary, $\rho(v;G)$ represents the mixed features from all observing regions around vertex v by inspector.

2) Shift on graph: The inspector takes only the features from the region around v. In order to observe more information, we can *shift* the inspector on the graph. The procedure is elaborated upon below. Given the entire graph G, consider that at the time step t, the inspector stays at v_t, and then $\rho(v_t;G)$ is observed. At the next step $t+1$, the inspector shifts from v_t to v_{t+1} under the instruction d_t, which is generated from action network and will be discussed later. At v_{t+1}, $\rho(v_{t+1};G)$ is observed. To explain how to get v_{t+1} from v_t in detail, we introduce a distance metric at the vertex.

Definition 4 (Distance between Vertices) *Given a graph G, the distance r_{v_i,v_j} between $v_i \in V$ and $v_j \in V$ is the Euclidean distance $\|M(v_i) - M(v_j)\|_2$ on feature matrix.*

Based on distance metric, we prepare a *ranking structure* D representing the relatively distance between each two vertices. for each vertex v_i, we rank all vertices in a descending order in terms of the distance from v_i. We use notation $D(v_i, j)$ to represent the vertex having j-th smallest distance from v_i. It ensures that

$$\forall v_i \in V, k \in 1, ..., |V|, r_{v_i,D(v_i,k)} \geq r_{v_i,D(v_i,k-1)}. \tag{5}$$

Note that D is independent with the inspector and can be computed off-line. With the help of D, we can localize any vertex by describing the order and location with respect to another vertex. For example, consider the graph in Fig. 2 (a), we can localize the green vertex, by pointing out the relative position with respect to the red one (v_1), namely $D(v_1, 3)$. Based on this mechanism, we can define the *atomic shift* operation on an integer instruction d.

Definition 5 (Atomic Shift) *Given the central v_t of the inspector at the time step t, the next observation central $v_{t+1} = D(v_t, d)$ is determined by instruction d.*

To improve the efficiency, we consider the *serial shift* at one step. For the serial shift, the instruction $\boldsymbol{d} = \{d^i\}_{i=1}^m$ is a vector, used to guide the shift of the inspector, where each dimension is a scalar corresponding to an atomic operation.

At each time step t, the initial position of the inspector is at the previous central vertex v_{t-1}, according to the instruction vector $\boldsymbol{d}_{t-1} = \{d_{t-1}^1, \cdots, d_{t-1}^m\}$. The inspector takes the first shift operation from the current vertex v_{t-1} to an intermediate vertex calculated by $\mathbf{D}(v_{t-1}, d_{t-1}^1)$, then uses this intermediate vertex as the start of the second atomic shift operation to get the next start vertex according to d_{t-1}^2. The rest can be repeated in the same manner. After m times of atomic shift operations, the inspector moves from the original start node v_{t-1} to v_t, see Fig. 2 (b).

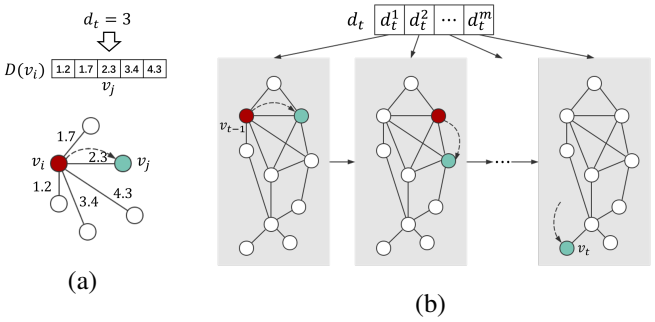

Fig. 2: **(a) Atomic Shift Operation:** Given a graph G, the start vertex v_i and a instruction $d = 3$. The inspector shifts to the green vertex with the 3rd smallest distance, i.e. $D(v_i, 3) = v_j$. **(b) Serial Shift Operation:** A serial shift operation $\boldsymbol{d}_{t-1}$ with m-dimensionality, consists of m atomic shift operations.

Definition 6 (Serial Shift) *Given the central vertex v_{t-1} of the inspector at time step $t-1$ and instruction $\boldsymbol{d}_{t-1}$, the next observation central vertex $v_t = \hat{v}^m$ where*

$$\hat{v}^i = \begin{cases} v_{t-1} & \text{if } i = 1 \\ D(\hat{v}^{i-1}, d_{t-1}^i) & \text{ow.} \end{cases} \tag{6}$$

The detailed procedure is as follows.

- step 1 At each time step t, the initial position of the inspector is at the previous central vertex v_{t-1}. And then the inspector receives the instruction vector $\boldsymbol{d}_{t-1} = \{d_{t-1}^1, \cdots, d_{t-1}^m\}$. Setting the intermediate vertex $\hat{v}^1 = v_{t-1}$ and repeating step 2 by taking iterator $i \in \{1, ..., m\}$.
- step 2 Taking an atomic shift operation on $\hat{v}^{i-1}$ under instruction d_{t-1}^i and setting the result as the intermediate vertex $\hat{v}^i = \mathbf{D}(\hat{v}^{i-1}, d_{t-1}^i)$.
- step 3 When $i = m$, setting $v_t = \hat{v}^m$.

For a large graph object, the serial shift is more efficient, which is studied in the experimental part. In the following discussion, we use $v_t = f_j(v_{t-1}, \boldsymbol{d}_{t-1}; G)$ to represent the shift procedure.

The structure of the inspection network, shown in Fig. 3, which integrates two features as a feature vector a_t by a two-layer neural network. The first feature is the vertex feature $\rho(v_t;G)$ derived from the region by the inspector. The second feature is the instruction $\boldsymbol{d_t}$. The inspection network, illustrated in Fig. 3, is formalized as follows

$$\begin{aligned} z_d &= \sigma(\boldsymbol{d}_{t-1}; \theta_a^0) \\ z_\sigma &= \sigma(\rho(v_t;G); \theta_a^1) \\ a_t &= \sigma([z_d, z_\sigma]; \theta_a^2) \end{aligned} \tag{7}$$

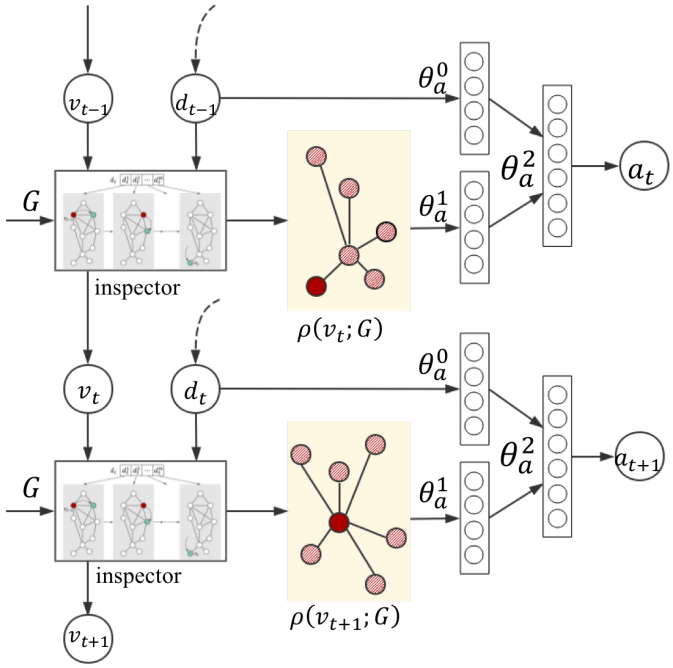

Fig. 3: Inspection Network

where z_σ and z_d are intermediate variables, $\theta_a = \{\theta_a^0, \theta_a^1, \theta_a^2\}$ is the network parameter, $[\cdot, \cdot]$ is the vector concatenation operation and $\sigma(\cdot)$ is the activation function.

In summary, the inspector module leverages the shift operation, especially serial shift operation, to observe a serial of regions of original graph under the instructions and compute the features a_t.

3) Inspector initialization: In the inspector module, despite the trainable parameters θ_a, the output features a_t is impacted by two factors: the instructions and the initial vector v_0. The initial vector, representing the inspector center v at time temp $t = 0$, determines which region is observed at first.

For each input graph, three methods available can be used to initialize the inspector center v_0. The simplest one is to randomly select a vertex from the input graph. An ideal method is to choose v_0 from the dense area of the input graph. Also, we can also get a vertex v_0 under the expert's guidance.

Our g-Inspector is not sensitive to the initial value. Each vertex and its neighbor region has an implicit contribution score to classification label. If initializing the start at the vertex with high score, the model can easily achieve good accuracy. If at the vertex with low score, our g-Inspector will shift quickly leaving current vertex by the shift instruction vector with a large value. And this policy is well trained by reinforcement rule in training phase.

E. Core network

The core network in the g-Inspector model, is a recurrent network maintaining a hidden state to keep the memory from previously observed subgraph features. In the model, the number of hidden layers, denoted by T, indicates that the inspector has observed the graph for T times.

The hidden state, initialized randomly and integrates its previous memory h_{t-1} with the feature a_t obtained by the inspection network. Formally speaking, $h_t = f_h(h_{t-1}, a_t; \theta_h)$ where θ_h is the neural network parameter. The hidden state explicitly encodes the inspector's knowledge of the graph, such as subgraph structures and attributes.

F. Action networks

The hidden state h_t contains all information that the inspector has collected from previous observations. Two actions are executed based on the hidden state at each step:

1) Generating the instruction $\boldsymbol{d}_t = f_d(h_t; \theta_d)$ by the *shift network*, determining where the inspector will explore in the next step.
2) Predicting the label $\hat{l}_t$ of the input graph by the *classification network* which applies softmax function.

$$Pr(\hat{l}_t = i) = \frac{\exp f_c(h_t; \theta_c)_i}{\sum_j \exp f_c(h_t; \theta_c)_j} \tag{8}$$

Besides, the action network provides guidance as to the time point to stop observing the graph when the inspector has collected sufficient information to make the accurate prediction.

From the perspective of reinforcement learning, the inspector is an *agent* interacting with the *environment*, i.e. the input graph, via a Partially Observable Markov Decision Process (POMDP). At time step t, the inspector, collects the partially *observation* a_t (see Eq. (7)), receives the *reward* signal r_t from the environment, and makes two *actions*: generating the shift instruction $\boldsymbol{d}_t$ and predicting label $\hat{l}_t$. Given the state, actions generated from actions networks are implicitly determined by the *policy* $\pi((\boldsymbol{d}_t, \hat{l}_t)|s_{1:t}; \theta_d, \theta_c)$ where $s_{1:t}$ is the memory on history $s_{1:t} = a_1, \boldsymbol{d}_1, \hat{l}_1, \cdots, a_{t-1}, \boldsymbol{d}_{t-1}, \hat{l}_{t-1}, a_t$, which can be replaced by the hidden state h_t. The parameters of policy are estimated by optimizing the accumulative reward $R = \sum_{t=1}^{T} r_t$. In our graph classification task, when the classification result is correct, let $r_T = 1$, otherwise let $r_T = 0$.

Taking the g-Inspector as a whole agent, the entire policy, denoted by $\pi((\boldsymbol{d}_t, \hat{l}_t, h_t)|\rho(v, G), h_{t-1}; \Theta)$ is determined by $\Theta = \{\theta_h, \theta_a, \theta_d, \theta_c\}$.Then, the total reward $J(\Theta) = \mathbb{E}_{p(s_{1:T};\Theta)}[R]$ is also the function of Θ.

Theorem III.1 (Sufficient Condition on Classification) *Given the graph G, the optimal Θ which maximizes $J(\Theta)$ is the optimal classifying mapping by assigning $f(G; \Theta) = \hat{l}_T$.*

The theorem III.1 shows that optimal solution $\Theta = \arg_\Theta \max J(\Theta)$ is the sufficient condition for the optimal graph classification.

Proof: Assume that exists another Θ' and a graph G whose correct label is l, such that $\pi(\hat{l}_T = l|G; \Theta') > \pi(\hat{l}_T = l|G; \Theta)$. Namely, the expectation $\mathbb{E}_{p(s_{1:T};\Theta')}[R] > \mathbb{E}_{p(s_{1:T};\Theta)}[R]$. According to the definition of J, $J(\Theta') > J(\Theta)$ which is against with $\Theta = \arg_\Theta \max J(\Theta)$. **QED**

With the theorem III.1, we can solve $\arg_\Theta \max J(\Theta)$ to archive the solution of proposed problem in Sec. III-A.

G. Training

In this section, we will introduce the training algorithm of our model. As mentioned above, the inspector needs to learn a policy by tuning the parameters $\Theta = \{\theta_h, \theta_a, \theta_d, \theta_c\}$ in the conjunction of the core network, the inspection network and the action networks. Since each policy of the inspector induces a distribution of the possible interaction sequences $s_{1:T}$, we want to maximize the total reward under this distribution: $J(\Theta) = \mathbb{E}_{p(s_{1:T};\Theta)}[R]$, where $p(s_{1:T};\Theta)$ depends on the policy representing the occurrence probability of the interaction sequence $s_{1:T}$.

Since the interaction sequences may be very large and difficult to exhaustive calculate, it is a non-trivial task to maximize J. Fortunately, by considering the problem as a POMDP, we use the technique introduced in [33], to obtain a sample approximation of the gradient of J as shown in [18], given by:

$$\nabla_\Theta J \approx \frac{1}{M_J} \sum_{i=1}^{M_J} \sum_{t=1}^{T-1} \nabla_\Theta \log \pi(\boldsymbol{d}_t^i, \hat{l}_t^i | s_{1:t}^i; \Theta) \gamma_J^{T-t} R^i \quad (9)$$

where $s_{1:t}^i$ is the interaction sequence generated by the inspector according to the current policy for $i = 1, \cdots, M_J$ episodes, and $\gamma_J \in (0, 1)$ is a discount factor. The equation (9) is also known as the REINFORCE rule [34], which obtains samples of interaction sequences $s_{1:T}$ by the inspector with current policy. Then, we adjust the parameters Θ of the inspector. It increases the log-probability of the chosen actions that result in a correct prediction and decreases the log-probability of low rewards corresponding actions. The policy trained in this way allows the agent to increase the chance to shift the inspector towards a relevant region the next time when facing the same state. To compute $\nabla_\Theta \log \pi(\boldsymbol{d}_t^i, \hat{l}_t^i | s_{1:t}^i; \Theta)$, we deal with the gradient of the network at each time step which can be computed by back-propagation [35]. In particular, our inspector observes the graph in the first $T-1$ step, therefore, we adjust the log-probability for $t = 1, \cdots, T-1$.

To reduce the variance, we add a baseline b to obtain an optimized $\nabla_\Theta J$ which is equal to the equation (9) in expectation but has lower variance[18]:

$$\frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T-1} \nabla_\Theta \log \pi(\boldsymbol{d}_t^i, \hat{l}_t^i | s_{1:t}^i; \Theta)(\gamma^{T-t} R^i - b_t^i) \quad (10)$$

where $b_t^i = f_b(s_{1:t}^i; \theta_b)$ to compute the cumulative reward depend on $s_{1:t}^i(h_t^i)$. Training the parameters according to equation (10), allows us to increase the log-probability of the chosen actions that result in greater cumulative reward, and decrease the log-probability of the actions with smaller obtained cumulative than that of baseline. The parameter θ_b of f_b is trained by reducing the mean squared error of $R^i - b_t^i$.

IV. Experiment

This section reports the experimental results to show the advantages of our g-Inspector. We compare the performance of our model with other approaches. Also, we examine an example to illustrate the effectiveness of the attention mechanism in graph classification problem, and discuss the scalability and interpretabe ability of our model. At last the source code of g-Inspector is open-sourced [1].

A. Experimental setup

1) Datasets: We evaluate our model on three bioinformatics datasets: MUTAG, NCI1, ENZYMES, and two social network datasets: IMDB-BINARY and IMDB-MULTI. The MUTAG dataset consists of 188 chemical compounds divided into 2 classes according to their mutagenic effect on a bacterium. The NCI1 dataset was published by National Cancer Institute(NCI), which includes 4110 chemical compounds and is divided into 2 classes on the activation against non-small cell lung cancer. ENZYMES has 600 protein tertiary structures, each of which belongs to one of the 6 classes. IMDB-BINARY and IMDB-MULTI are movie collaboration datasets collected from IMDB, where nodes represent actors/actresses and the edges connect the actors/actresses who appear in the same movie.

2) Compared methods: We compare g-Inspector against the following representative approaches: Random Walk kernel(RW) [36], Graphlet kernel (GK) [10], Deep Graph Kernel(DGK)[9] (Code from [37]), DCNN [13] (Code from [38]), DGCNN [26], PSCN [11], GAM and GAM-mem [31]. Note that the codes for GAM and GAM-mem are neither released on the Internet nor accessible from the authors, we refer to the performance results reported in [31].

3) Implementation: Our g-Inspector is implemented by Python 2.7 and TensorFlow 1.4.1, and executed on the server with 32GB memory, 2.4GHz Intel CPU. In all experiments, the following setting are the same. We use DeepWalk[32] to represent each node as a 64-dimensionality vector. In the inspection network, we select a random node in the input graph to initialize the start central vertex v_0 and the shift instruction $\boldsymbol{d}_0$ is initialized by a uniform distribution. Both the feature vector a and the hidden state are with the same dimensionality (256). In the training procedure, we set the number of samples $M_J = 20$ and the discount factor $\gamma_J = 1$. We train the g-Inspector using stochastic gradient descent with a batch size $= 20$, the learning rate of 10^{-3}, and momentum of 0.9.

Beside the basic implementation of g-Inspector, we implement a multi-agent version, called g-Inspector-mem. It contains multiple inspectors to explore the graph independently and concurrently. Then it integrates their observations on the same graph into a memory cell by averaging their hidden states. At last, the classification network, see section III-F, leverages the memory cell to conduct the prediction. This extension is similar to GAM-mem[31].

B. Performance comparisons

The performance results are reported in Tab. I, in which each column represents a dataset, the second to the forth rows represent the statistical report, and the rest represent the compared methods. For GAM, we compare the single agent version (GAM) and its multi-agent version (GAM-mem). The

[1] https://github.com/zhilingluo/g-Inspector

TABLE I: Statistics of datasets and accuracy for g-Inspector compared with baselines

Datasets	MUTAG	NCI1	ENZYMES	IMDB-BINARY	IMDB-MULTI
Graph#	188	4110	600	1000	1500
Class#	2	2	6	2	3
Max.Node#	28	111	126	136	89
Avg.Node#	17	29	32	19	13
RW	83.72 ± 1.50	48.15 ± 0.50	24.26 ± 1.64	64.54 ± 1.22	34.54 ± 0.76
GK	81.66 ± 2.11	62.28 ± 0.29	26.61 ± 0.99	65.87 ± 0.98	43.89 ± 0.38
DGK	$82.66 \pm 1.45(0.028s)$	$62.48 \pm 0.25(10.636s)$	$27.08 \pm 0.79(0.765s)$	$66.96 \pm 0.56(1.089s)$	$44.55 \pm 0.52(2.430s)$
DCNN	$77.19 \pm 0.00(0.004s)$	$56.92 \pm 2.338(0.195s)$	$16.22 \pm 1.937(0.025s)$	$51.20 \pm 1.6(0.049s)$	$35.33 \pm 1.485(0.041s)$
DGCNN	85.83 ± 1.66	74.44 ± 0.47	—	70.03 ± 0.86	47.83 ± 0.85
PSCN	88.95 ± 4.37	$\mathbf{76.34 \pm 1.68}$	—	$\mathbf{71.00 \pm 2.29}$	45.23 ± 2.84
GAM	—	64.17 ± 0.05	—	—	—
GAM-mem	—	67.71 ± 0.04	—	—	—
g-Inspector	$87.05 \pm 4.81(0.026s)$	$65.63 \pm 1.95(0.076s)$	$37.13 \pm 4.78(0.051s)$	$64.47 \pm 2.42(0.077s)$	$46.72 \pm 2.36(0.069s)$
g-Inspector-mem	$\mathbf{90.19 \pm 2.34(0.032s)}$	$67.79 \pm 1.24(0.112s)$	$\mathbf{38.75 \pm 2.45(0.068s)}$	$65.71 \pm 1.57(0.092s)$	$\mathbf{47.92 \pm 1.33(0.086s)}$

experiments are repeated 10 times in the same setting over 10-fold cross-validation. All results are presented as the average classification accuracy $\pm$ standard deviation.

We make three-fold interesting observations. *First*, the g-Inspector outperforms the other methods on MUTAG, ENZYMES, and IMDB-MULTI, with accuracy gain on average by 16.30% and up to 22.53%. Concretely, using g-Inspector, the accuracy gain for MUTAG is 13.00%, for ENZYMES is 22.53% and for IMDB-MULTI is 13.38%. For the two left datasets NCI1 and IMDB-BINARY, g-Inspector is also competitive.

Second, the execution time of g-Inspector is short than DGK on all the datasets, and also shorter than DCNN on NCI1. This is mainly because the forward procedure of g-Inspector is efficient. For example, DGK takes 1.089 seconds, 14 times of 0.077 of g-Inspector, and 12 times of 0.092 of g-Inspector-mem.

Third, the g-Inspector is more effective and efficient than GAM. For model design, g-Inspector moves to a farther vertex by the shift operation, compared with GAM moving to a one-hop neighborhood vertex at one step. From the experiment results, (1) multi-agent version (both GAM-mem and g-Inspector-mem) achieves higher classification accuracy than the basic version; (2) our g-Inspector gets 65.63% which is better than 64.17% of GAM; and (3) g-Inspector-mem gets 67.79% which is better than 67.71% of GAM-mem.

C. Result illustration of attention mechanism

In order to better understand the nature of the attention mechanism, we inspect an compound $C_{14}H_{23}NO_2$ from MUTAG from MUTAG. Figure 4 (a) illustrates the structure of this compound and the shift process of the inspector . In $C_{14}H_{23}NO_2$ from MUTAG, the nitro structure consisting of vertex 14, 15 and 16, determines the class of this compound. In this case, the inspector shifts from the initial carbon atom at the vertex 3, and takes 6 shift operations over time. Figure 4 (b) demonstrates the classification accuracy at each time step. For example, at the first time step, the vertex 3 and its neighborhoods, see Fig. 4 (b-i), are observed, and the current classification accuracy is only 59.9%. When the inspector shifts to the vertex 7, it can observe the one-hop neighborhood,

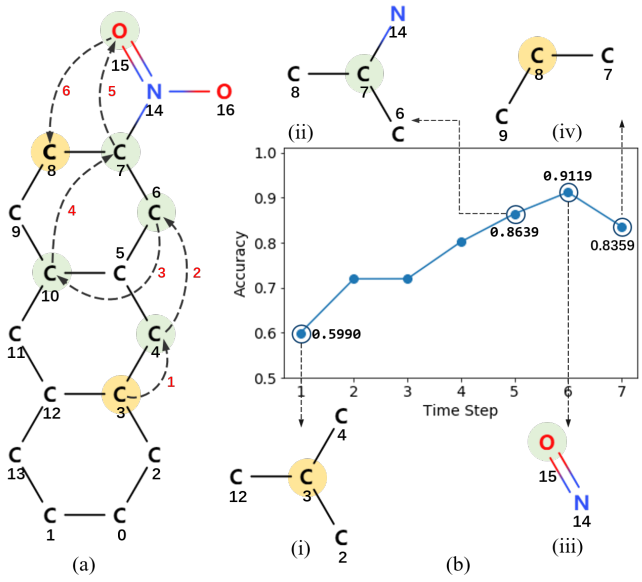

Fig. 4: **(a)** The inspector shift process on $C_{14}H_{23}NO_2$ from MUTAG. **(b)** The classification accuracy varies over the shift of inspector.

see (b-ii), which contains the nitrogen atom vertex 14, and the classification accuracy can reach 86.39%. When the inspector shifts to the oxygen atom vertex 15, it achieves the best classification accuracy of 91.19%, because both an oxygen vertex 15 and the nitrogen vertex 14 can be observed, see (b-iii). When the inspector shifts from the vertex 15 to the vertex 8, see (b-iv), the accuracy is reduced to 83.59%, in practice, we used an early-stop to stop the inspector's shift and output the optimal classification result at (b-iii).

D. Interpretable Ability of g-Inspector

Compared with other graph classification methods, our proposed model can make the results interpretable. Specifically, our g-Inspector can help to measure the contribution of each subgraph region to find out the key subgraph regions for classification. Our g-Inspector can assign each observed region a contribution score by normalizing the gain of the corre-

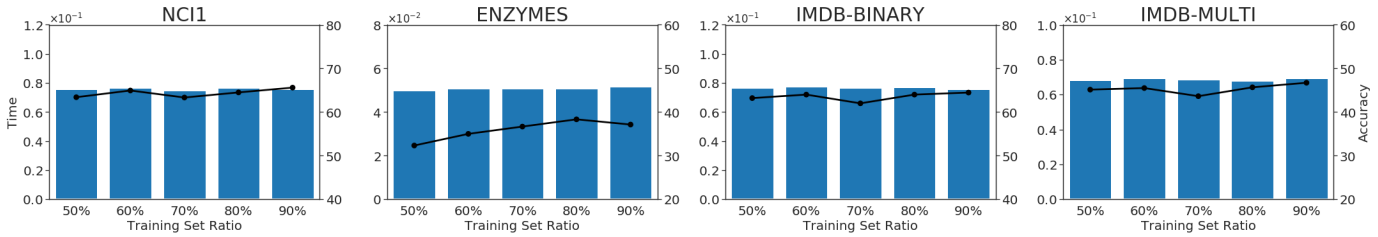

Fig. 5: The performance with different size of training set on four datasets.

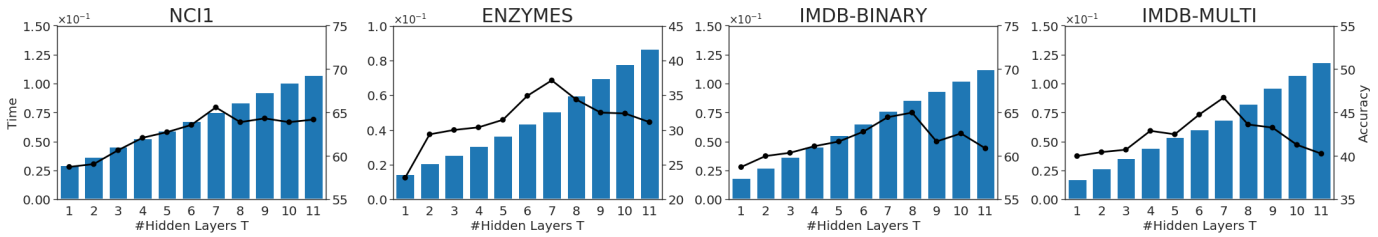

Fig. 6: The performance with different hidden layers number T on four datasets.

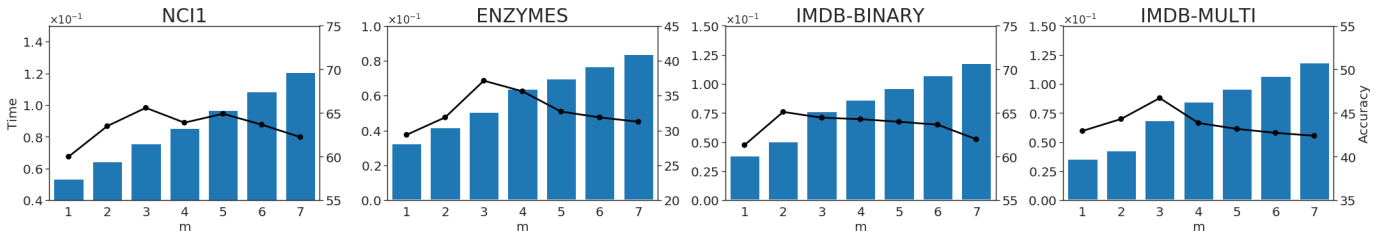

Fig. 7: The performance with different inspector shift instruction length m on four datasets.

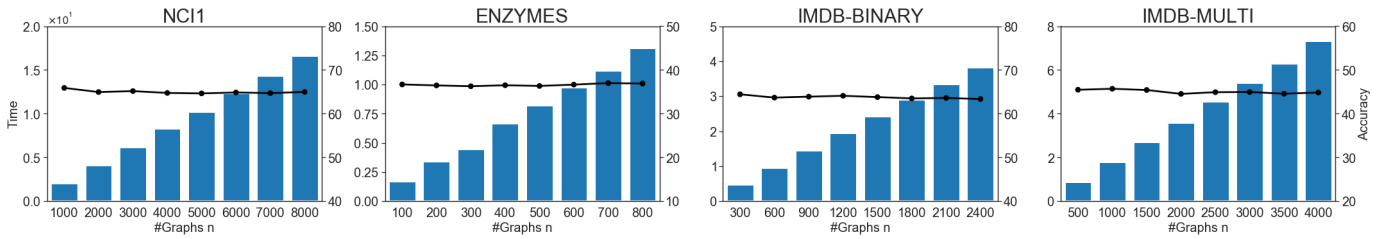

Fig. 8: The performance with different graph number on four datasets.

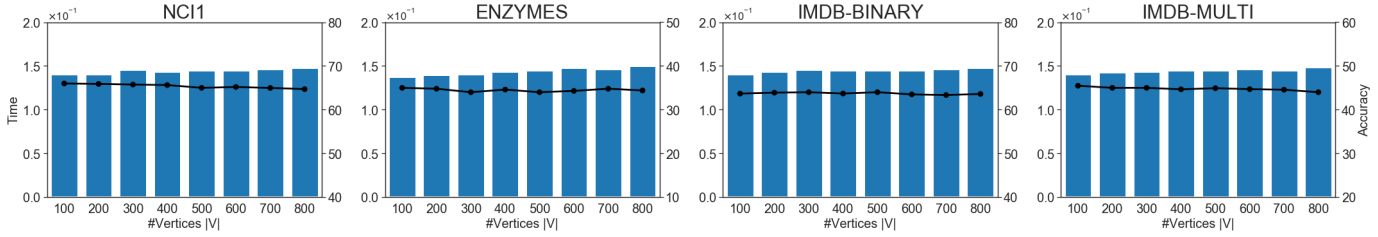

Fig. 9: The performance with different vertex number on four datasets.

sponding classification accuracy. For example, the scores of the four regions shown in Fig. 4(b) are 0.19, 0.27, 0.28, 0.26. It concludes that the third region (b-iii) is more significant. We can discover the significant regions for a classification task by intersections of the high-score regions from each graph in a set, e.g. the nitro structure in MUTAG, which making the results interpretable.

E. Performance on different amounts of training data

It is essential to investigate the performance with different size of training data and test data. We design this experiment by dividing the datasets into sub-datasets with different proportion of training set, varying from 90% to 50%, on NCI1, ENZYMES, IMDB-BINARY and IMDB-MULTI. The results are reported in Fig. 5, where the x-axis represents the training

TABLE II: Statistics of datasets and accuracy for g-Inspector compared with baselines

Datasets	MUTAG	NCI1	ENZYMES	IMDB-BINARY	IMDB-MULTI
g-Inspector	$87.05 \pm 4.81(0.026s)$	$65.63 \pm 1.95(0.076s)$	$37.13 \pm 4.78(0.051s)$	$64.47 \pm 2.42(0.077s)$	$46.72 \pm 2.36(0.069s)$
g-Inspector (node2vec)	$86.50 \pm 2.49(0.027s)$	$64.78 \pm 2.59(0.076s)$	$36.73 \pm 2.37(0.053s)$	$62.33 \pm 2.51(0.078s)$	$44.32 \pm 0.89(0.065s)$
g-Inspector (no-shift)	$81.00 \pm 3.53(0.007s)$	$58.75 \pm 1.48(0.030s)$	$23.13 \pm 4.20(0.015s)$	$58.73 \pm 1.74(0.019s)$	$40.00 \pm 1.43(0.018s)$
g-Inspector (atomic-shift)	$83.00 \pm 3.35(0.015s)$	$60.00 \pm 0.94(0.054s)$	$29.38 \pm 3.96(0.033s)$	$61.33 \pm 1.47(0.039s)$	$42.95 \pm 1.17(0.036s)$
g-Inspector (0-order)	$85.50 \pm 2.85(0.026s)$	$64.40 \pm 2.07(0.067s)$	$34.40 \pm 3.58(0.054s)$	$63.80 \pm 1.14(0.071s)$	$44.40 \pm 1.51(0.058s)$

size proportion and the left and right sides of the y-axis represent the test time and accuracy respectively. It supports that (1) the test time keep steady because it is independent with training set size; and (2) the accuracy increases a little with the growth of training set proportion. When the training set is extremely small, e.g. 50% for IMDB-BINARY, the classification accuracy is still competitive (66%, close to most baselines).

F. Step number tuning

This experiment examined the effectiveness and efficiency of the model with different # hidden layer, denoted by T, which represents the maximum step number of shift operation. In this experiment, we set $m = 3$, vary the parameter T from 1 to 11, on NCI1, ENZYMES, IMDB-BINARY and IMDB-MULTI, and report the results in Fig. 6. The x-axis represents parameter T and the left and right sides of the y-axis are the test time and accuracy, respectively. We make two observations: (1) For most datasets, the execution time grows as T increases. The larger T means more recurrent layers in the core net, resulting in larger time complexity in computation for forward procedure of the g-Inspector. (2) The optimal setting of T varies from dataset to dataset. For IMDB-BINARY, the classification accuracy grows when $T \leq 8$ but decreases when $T > 8$. For other datasets, the classification accuracy grows when $T \leq 7$ and decreases when $T > 7$. A very large T increases the difficulty of pre-stop and causes overfitting on the training set.

G. Shift instruction length tuning

We conducted another experiment to report the performance with varying the instruction length m from $1, 11$, on NCI1, ENZYMES, IMDB-BINARY and IMDB-MULTI. Figure 7 illustrates the results on both accuracy and test time. We make the following interesting observations: (1) The test time increases greatly when the length varies from 1 to 7, as shown by the height of the bar(s) in Fig. 7. It is because the longer instruction vector requires more computation in both action network and inception network. (2) With the growth of m, the accuracy increases at first, achieves the optimal value and then keeps steady or even decreases. For example, for ENZYMES, the accuracy is 37% at $m = 3$ and decreases to 32% at $m \geq 6$. Because the longer instruction, meaning more atomic shift operations at a time step, represents higher efficiency and provides greater chance to scan more regions in graph. It eventually obtains features observed by the inspector and promotes the classification accuracy. When m is too large, the effectiveness will be canceled due to the over-fitting.

H. Scalability on graph number

To evaluate the scalability of g-Inspector, we conducted an experiment varying the graph number in datasets and reported the performance. In this experiment, we control the graph number by randomly removing and duplicating graph objectives in test set. The experiment results on accuracy and test time are reported in Fig. 8. There are two observations: (1) The test time grows linearly with respect to the growing of graph number. For example, in NCI1, the test time is around 0.25 seconds when graph number is 1000. And the test time reaches 1.7 seconds when graph number is 8000. (2) The accuracy keeps steady with different graph number. For example, in IMDB-BINARY, the accuracy is about 65% when graph number is either 300 or 2400. The experiment results show that g-Inspector works well for a mass of graph objectives.

I. Scalability on vertex number

The last experiment is about the scalability of g-Inspector on vertex number of graph. The key advantage of g-Inspector is that it just loads a small region of the graph object instead of loading the entire graph via the inspection network. It means that it has the good time complexity for big graph object. To evaluate this scalability, we conduct this experiment by evaluate g-Inspector on graph objects having different vertex number. We generate the big graph objects by duplicating the nodes and connecting edges from original graph. The experiment results are illustrated by Fig. 9, from which we can make the observation: Both test time and accuracy remain unchanged with the growth of #vertices. For example, in NCI1, the accuracy is always 68% and the test time is 0.013 seconds when #vertices = 100, or 800.

J. Ablations

We perform ablation experiments to validate several key parts of g-Inspector. *First*, we design an experiment to investigate the impact of embedding algorithm by employing node2vec as ψ, which can preserve higher-order proximity between nodes to a low-dimensional space the same as Deepwalk[39]. We compare the normal version

(g-Inspector) using DeepWalk as the encoder and a variant version g-Inspector(node2vec), shown in Tab.2. The g-Inspector(node2vec) denotes the model using node2vec as the encoder. When comparing g-Inspector and g-Inspector(node2vec) in Tab.2, we can see that the accuracy decreased on average by 1.27% up to 2.40% on all datasets. Even the accuracy on different version is slightly different, our g-Inspector is always competitive with baseline. The accuracy promotion or demotion caused by replacing different embedding algorithm is not our major technical contribution.

Second, we design an experiment to investigate the effectiveness of the shift operation. More specifically, we compare the normal version (g-Inspector), and the other two variant versions, shown in Tab. 2. The g-Inspector(no-shift) denotes the model in which the inspector cannot be shifted. The g-Inspector(atomic-shift) denotes the model where only atomic shift operation is permitted and serial shift operation is forbidden. When comparing g-Inspector and g-Inspector(no-shift) in Tab. 2, we can see that enabling the inspector shift makes the accuracy higher on average by 7.88% up to 14% on all datasets. The shift operation makes the inspector observe more than one region of graph to retrieve valuable information for classification. Compared to atomic shit, serial shift operation provides accuracy improvement on average by 4.87% and up to 7.75%. The serial shift operation speeds up the shift process, and observes more regions subject to the constraint steps.

Third, we design an experiment to investigate the impact of the size of observing regions by inspector, which was defined in the *k-order region*. We compare the normal version (g-Inspector) where $k = 1$ and a variant version g-Inspector(0-order) where $k = 0$, shown in Tab.2. The g-Inspector(0-order) denotes the model in which the inspector only observes a single node at once time. When comparing g-Inspector and g-Inspector(0-order) in Tab.2, we can see that enabling the inspector to observe a wider range makes the accuracy higher on average by 1.70% up to 2.73% on all datasets. When the scope of observation is wider, the inspector can collect more information to makes better decisions.

V. Conclusion

To address the challenges of graph classification, we proposed a recurrent attention model on graphs, called g-Inspector. We introduced an inspector module applying the attention to investigate the significance of each region to classification. It is interpretable owing to the ability of measuring the contribution of each region to the classification. Besides, we introduced a shift operation across the graph selecting a series of task relevant regions instead of searching the entire graph, avoiding the high dimensionality problem. Our experimental results showed that our g-Inspector model is competitive and achieves a higher accuracy compared with existing methods.

VI. Acknowledgments

We would like to thank all reviewers and editors for the constructive suggestions. Dr. Sha Zhao and Dr. Jianwei Yin are the corresponding authors. This work was supported by National Natural Science Foundation of China (No. 61802340, No. 61802342).

References

[1] H. Cai, V. W. Zheng, and K. Chang, "A comprehensive survey of graph embedding: problems, techniques and applications," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[2] R. A. Rossi, R. Zhou, and N. Ahmed, "Deep inductive graph representation learning," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[3] S. Pan, J. Wu, and X. Zhu, "Cogboost: Boosting for fast cost-sensitive graph classification," *IEEE Transactions on Knowledge and Data Engineering*, no. 1, pp. 1–1, 2015.

[4] X. Kong and P. S. Yu, "Semi-supervised feature selection for graph classification," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 793–802.

[5] ——, "Multi-label feature selection for graph classification," in *Proceedings of the 10th IEEE ICDM International Conference on Data Mining*, 2010, pp. 274–283.

[6] S. Pan and X. Zhu, "Graph classification with imbalanced class distributions and noise," in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2013, pp. 1586–1592.

[7] L. Dehaspe, H. Toivonen, and R. D. King, "Finding frequent substructures in chemical compounds," in *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1998, pp. 30–36.

[8] H. Wang, P. Zhang, X. Zhu, I. W.-H. Tsang, L. Chen, C. Zhang, and X. Wu, "Incremental subgraph feature selection for graph classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 1, pp. 128–142, 2017.

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

[10] 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.

[11] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceedings of the 33rd International Conference on Machine Learning*, 2016, pp. 2014–2023.

[12] Z. Luo, L. Liu, J. Yin, Y. Li, and Z. Wu, "Deep learning of graphs with ngram convolutional neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 10, pp. 2125–2139, 2017.

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

[14] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 793–803.

[15] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1416–1424.

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

[17] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 13th Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1412–1421.

[18] V. Mnih, N. Heess, A. Graves *et al.*, "Recurrent models of visual attention," in *Advances in Neural Information Processing Systems*, 2014, pp. 2204–2212.

[19] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 2048–2057.

[20] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in Neural Information Processing Systems*, 2015, pp. 577–585.

[21] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," *arXiv preprint arXiv:1412.7755*, 2014.

[22] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Proceedings of the 2nd IEEE ICDM International Conference on Data Mining*, 2002, pp. 721–724.

[23] K. M. Borgwardt and H. P. Kriegel, "Shortest-path kernels on graphs," in *Proceedings of the 5th IEEE ICDM International Conference on Data Mining*, 2006, pp. 74–81.

[24] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017, pp. 2429–2435.

[25] 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.

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

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

[28] E. Choi, M. T. Bahadori, L. Song, W. F. Stewart, and J. Sun, "Gram: graph-based attention model for healthcare representation learning," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 787–795.

[29] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[30] J. B. Lee, R. A. Rossi, X. Kong, S. Kim, E. Koh, and A. Rao, "Higher-order graph convolutional networks," *arXiv preprint arXiv:1809.07697*, 2018.

[31] 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 and Data Mining*, 2018, pp. 1666–1674.

[32] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 701–710.

[33] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[34] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[35] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber, "Solving deep memory pomdps with recurrent policy gradients," in *International Conference on Artificial Neural Networks*. Springer, 2007, pp. 697–706.

[36] 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.

[37] P. Yanardag and S. Vishwanathan, "Deep graph kernels code," http://www.mit.edu/~pinary/kdd/, 2017.

[38] J. Atwood, "Dcnn code," https://github.com/jcatw/dcnn, 2017.

[39] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 855–864.

Zhiling Luo is an Assistant Research Professor in Computer Science at Zhejiang University, China. He received his B.S. and Ph.D. degree in Computer Science from Zhejiang University in 2012 and 2017, respectively. He was the visiting scholar of Georgia Institute of Technology, US, in 2016. His research interests include service computing, machine learning and data mining.

Yinghua cui is a postgraduate student at College of Computer Science, Zhejiang University. Her research interests include deep learning and data mining.

Sha Zhao is currently a Postdoctoral Research Fellow of the College of Computer Science and Technology, Zhejiang University. She received the Ph.D. degree from Zhejiang University, Hangzhou, China, in 2017. She visited the Human-Computer Interaction Institute at Carnegie Mellon University as a visiting PhD student from 2015 to 2016. She received the Best Paper Award of ACM UbiComp'16. Her research interests include pervasive computing, data mining, and machine learning.

Jianwei Yin is currently a professor in the College of Computer Science, Zhejiang University, China. He received his Ph.D. in Computer Science from Zhejiang University in 2001. He is the visiting scholar of Georgia Institute of Technology, US, in 2008. His research interests include service computing, cloud computing and information integration. Currently Prof. Yin is the AE of IEEE Transactions on Service Computing.