Segmented GRAPH-BERT for Graph Instance Modeling

Jiawei Zhang

Abstract

In graph instance representation learning, both the diverse graph instance sizes and the graph node orderless property have been the major obstacles that render existing representation learning models fail to work. In this paper, we will examine the effectiveness of GRAPH-BERT on graph instance representation learning, which was designed for node representation learning tasks originally. To adapt GRAPH-BERT to the new problem settings, we re-design it with a segmented architecture instead, which is also named as SEG-BERT (Segmented GRAPH-BERT) for reference simplicity in this paper. SEG-BERT involves no node-order-variant inputs or functional components anymore, and it can handle the graph node orderless property naturally. What’s more, SEG-BERT has a segmented architecture and introduces three different strategies to unify the graph instance sizes, i.e., full-input, padding/pruning and segment shifting, respectively. SEG-BERT is pre-trainable in an unsupervised manner, which can be further transferred to new tasks directly or with necessary fine-tuning. We have tested the effectiveness of SEG-BERT with experiments on seven graph instance benchmark datasets, and SEG-BERT can out-perform the comparison methods on six out of them with significant performance advantages.

1. Introduction

Different from the previous graph neural network (GNN) works (Zhang et al., 2020; Veličković et al., 2018; Kipf & Welling, 2016), which mainly focus on the node embeddings in large-sized graph data, we will study the representation learning of the whole graph instances in this paper. Representative examples of graph instance data studied in research include the human brain graph, molecular graph, and real-estate community graph, whose nodes (usually only in tens or hundreds) represent the brain regions, atoms and POIs, respectively. Graph instance representation learning has been demonstrated to be an extremely difficult task. Besides the diverse input graph instance sizes, attributes and extensive connections, the inherent node-orderless property (Meng & Zhang, 2019) brings about more challenges on the model design.

Existing graph representation learning approaches are mainly based on the convolutional operator (Krizhevsky et al., 2012) or the approximated graph convolutional operator (Hammond et al., 2011; Defferrard et al., 2016) for pattern extraction and information aggregation. By adapting CNN (Krizhevsky et al., 2012) or GCN (Kipf & Welling, 2016) to graph instance learning settings, (Niepert et al., 2016; Verma & Zhang, 2018; Zhang et al., 2018; Zhang & Chen, 2019; Chen et al., 2019) introduce different strategies to handle the node orderless properties. However, due to the inherent learning problems with graph convolutional operator, such models have also been criticized for serious performance degradation on deep architectures (Zhang & Meng, 2019). Deep sub-graph pattern learning (Meng & Zhang, 2019) is another emerging new-trend on graph instance representation learning. Different from these aforementioned methods, (Meng & Zhang, 2019) introduces the IsoNN (Isomorphic Neural Net) to learn the sub-graph patterns automatically for graph instance representation learning, which requires the identical-sized graph instance input and cannot handle node attributes.

In this paper, we introduce a new graph neural network, i.e., SEG-BERT (Segmented GRAPH-BERT), for graph instance representation learning based on the recent GRAPH-BERT model. Originally, GRAPH-BERT is introduced for node representation learning (Zhang et al., 2020), whose learning results have been demonstrated to be effective for various node classification/clustering tasks already. Meanwhile, to adapt GRAPH-BERT to the new problem settings, we re-design it in several major aspects: (a) SEG-BERT involves no node-order-variant inputs or functional components. SEG-BERT excludes the nodes’ relative positional embeddings from the initial inputs; whereas both the self-attention (Vaswani et al., 2017) and the representation fusion component in SEG-BERT can both handle the graph node orderless property naturally. (b) SEG-BERT unifies...
Graph Representation Learning Unification: We examine the effectiveness of Graph-BERT in graph instance representation learning task in this paper. The success of this paper will help unify the currently disconnected representation learning tasks on nodes and graph instances, as discussed in Zhang [2019], with a shared methodology, which will even allow the future model transfer across graph datasets with totally different properties, e.g., from social networks to brain graphs.

Node-Orderless Graph Instances: We introduce a new graph neural network model, i.e., SEG-BERT, for graph instance representation learning by re-designing Graph-BERT. SEG-BERT only relies on the attention learning mechanisms and the node-order-invariant inputs and functional components in SEG-BERT allow it to handle the graph instance node orderless properties very well.

Segmented Architecture: We design SEG-BERT in a segmented architecture, which has a reasonable-sized input portal. To unify the diverse sizes of input graph instances to fit the input portals, SEG-BERT introduces three strategies, i.e., full-input, padding/pruning and segment shifting, which can work well for different learning scenarios, respectively.

Pre-Train & Transfer & Fine-Tune: We study the unsupervised pre-training of SEG-BERT on graph instance studies, and explore to transfer such pre-trained models to the down-stream application tasks directly or with necessary fine-tuning. In this paper, we will pre-train SEG-BERT with unsupervised node attribute reconstruction and graph structure recovery tasks, and further fine-tune SEG-BERT on supervised graph classification as the down-stream tasks.

The remaining parts of this paper are organized as follows. We will introduce the related work in Section 2. Detailed information about the SEG-BERT model will be introduced in Section 4 whereas the pre-training and fine-tuning of SEG-BERT will be introduced in Section 5 in detail. The effectiveness of SEG-BERT will be tested in Section 6. Finally, we will conclude this paper in Section 7.

2. Related Work

Several interesting research topics are related to this paper, which include graph neural networks, graph representation learning and BERT.

GNNs and Graph Representation Learning: Different from the node representation learning [Kipf & Welling 2016; Velickovic et al. 2018], GNNs proposed for the graph representation learning aim at learning the representation for the entire graph instead (Narayanan et al. 2017). To handle the graph node permutation invariant challenge, solutions based various techniques, e.g., attention (Chen et al. 2019; Meltzer et al. 2019), pooling (Meltzer et al. 2019; Ranjan et al. 2019; Jiang et al. 2018), capsule net (Mallea et al. 2019), Weisfeiler-Lehman kernel (Kriege et al. 2016) and sub-graph pattern learning and matching (Meng & Zhang 2019), have been proposed.

For instance, Mallea et al. 2019 studies the graph classification task with a capsule network; Chen et al. 2019 defines a dual attention mechanism for improving graph convolutional network on graph representation learning; and Meltzer et al. 2019 introduces an end-to-end learning model for graph representation learning based on an attention pooling mechanism. On the other hand, Ranjan et al. 2019 focuses on studying the sparse and differentiable pooling method to be adopted with graph convolutional network for graph representation learning; Kriege et al. 2016 examines the optimal assignments of kernels for graph classification, which can out-perform the Weisfeiler-Lehman kernel on benchmark datasets; and Jiang et al. 2018 proposes to introduce the Gaussian mixture model into the graph neural network for representation learning.
As an emerging new-trend, Meng & Zhang (2019) explores the deep sub-graph pattern learning and proposes to learn interpretable graph representations by involving sub-graph matching into a graph neural network.

**BERT: Transformer** (Vaswani et al. 2017) and BERT (Devlin et al. 2018) based models have almost dominated NLP and related research areas in recent years due to their great representation learning power. Prior to that, the mainstream sequence transduction models in NLP are mostly based on complex recurrent (Hochreiter & Schmidhuber 1997; Chung et al. 2014) or convolutional neural networks (Kim 2014). However, as introduced in (Vaswani et al., 2017), which dispense with recurrence and convolutions entirely. Based on Transformer, Devlin et al. (2018) further introduces BERT for deep language understanding, which obtains new state-of-the-art results on eleven natural language processing tasks. By extending Transformer and BERT, many new BERT-based models, e.g., T5 (Raffel et al. 2019), ERNIE (Sun et al. 2019) and RoBERTa (Liu et al. 2019), can even outperform the human beings on almost all NLP benchmark datasets. Some extension trials of BERT on new areas have also been observed. In Zhang et al. (2020), the authors explore to extend BERT for graph representation learning, which discard the graph links and learns node representations merely based on the attention mechanism.

### 3. Problem Formulation

In this section, we will first introduce the notations used in this paper. After that, we will provide the definitions of several important terminologies, and then introduce the formal statement of the studied problem.

#### 3.1. Notations

In the sequel of this paper, we will use the lower case letters (e.g., \(x\)) to represent scalars, lower case bold letters (e.g., \(\mathbf{x}\)) to denote column vectors, bold-face upper case letters (e.g., \(X\)) to denote matrices, and upper case calligraphic letters (e.g., \(A\)) to denote sets or high-order tensors. Given a matrix \(X\), we denote \(X(i,:)\) and \(X(:,j)\) as its \(i_{th}\) row and \(j_{th}\) column, respectively. The \((i_{th}, j_{th})\) entry of matrix \(X\) can be denoted as either \(X(i,j)\). We use \(X\) and \(X^\top\) to represent the transpose of matrix \(X\) and vector \(x\). For vector \(x\), we represent its \(L^p\)-norm as \(\|x\|_p = (\sum_i |x(i)|^p)^{\frac{1}{p}}\). The Frobenius-norm of matrix \(X\) is represented as \(\|X\|_F = (\sum_{i,j} |X(i,j)|^2)^{\frac{1}{2}}\). The element-wise product of vectors \(x\) and \(y\) of the same dimension is represented as \(x \odot y\), whose concatenation is represented as \(x \sqcup y\).

#### 3.2. Terminology Definitions

Here, we will provide the definitions of several important terminologies used in this paper, which include graph instance and graph instance set.

**Definition 1 (Graph Instance):** Formally, a graph instance studied in this paper can be denoted as \(G = (V, E, w, x)\), where \(V\) and \(E\) denote the sets of nodes and links in the graph, respectively. Mapping \(w : E \rightarrow \mathbb{R}\) projects links in the graph to their corresponding weight. For unweighted graphs, we will have \(w(e_{i,j}) = 1, \forall e_{i,j} \in E\) and \(w(e_{i,j}) = 0, \forall e_{i,j} \in V \times V \setminus E\). For the nodes in the graph instance, they may also be associated with certain attributes, which can be represented by mapping \(x : V \rightarrow X\) (here, \(X = \mathbb{R}^{d_x}\), denotes the attribute vector space and \(d_x\) is the space dimension).

Based on the above definition, given a node \(v_i\) in graph instance \(G\), we can represent its connection weights with all the other nodes in the graph as \(w_i = [w(e_{i,j})]_{v_j \in V} \in \mathbb{R}^{|V| \times 1}\). Meanwhile, the raw attribute vector representation of \(v_i\) can also be simplified as \(x_i = x(v_i)\). The size of graph instance \(G\) can be denoted as the number of involved nodes, i.e., \(|V|\). For the graph instances studied in this paper, they can be in different sizes actually, which together can be represented as a graph instance set.

**Definition 2 (Graph Instance Set):** For each graph instance studied in this paper, it can be attached with some pre-defined class labels. Formally, we can represent labeled graph instances studied in this paper as set \(\mathcal{G} = \{(G_i, y_i)\}_{i=1}^n\), where \(y_i \in Y\) denotes the label vector of \(G_i\) (here, \(Y = \mathbb{R}^{d_y}\) is the class label vector space and \(d_y\) is the space dimension).

For representation simplicity, in reference to the graph instance set (without labels), we can also denote it as \(\mathcal{G}\), which will be used in the following problem statement.

#### 3.3. Problem Formulation

Based on the notations and terminologies defined above, we can provide the problem statement as follows.

**Problem Statement:** Formally, given the labeled graph instance set \(\mathcal{G}\), we aim at learning a mapping \(f : \mathcal{G} \rightarrow \mathbb{R}^{d_h}\) to project the graph instances to their corresponding latent representations (\(d_h\) denotes the hidden representation space dimension). What’s more, we cast extra requirements on the mapping \(f\) in this paper: (a) representations learned by \(f\) should be invariant to node orders, (b) \(f\) can accept graph instances in various sizes, as well as diverse cate-
4. The SEG-BERT Model

In this section, we will provide the detailed information about the SEG-BERT model for graph instance representation learning. As illustrated in Figure 1, the SEG-BERT model has several key components: (1) graph instance serialization to reshape the various-sized input graph instances into node list, where the node orders will not affect the learning results; (2) initial embedding extraction to define the initial input feature vectors for all nodes in the graph instance; (3) input size unification to fit the input portal size of graph-transformer; (4) graph-transformer to learn the nodes’ representations in the graph instance (or segments) with several layers; (5) representation fusion to integrate the learned representations of all nodes in the graph instance; and (6) functional component to compute and output the learning results. These components will all be introduced in this section in detail, whereas the learning detail of SEG-BERT will be discussed in the follow-up Section 5 instead.

4.1. Graph Serialization and Initial Embeddings

Formally, given a graph instance \( G \in \mathcal{G} \) from the graph set, we can denote its structure as \( G = (\mathcal{V}, \mathcal{E}, w, x) \), involving node set \( \mathcal{V} \) and link set \( \mathcal{E} \), respectively. To simplify the notations, we will not indicate the graph instance index subscript in this section. Both the initial inputs and the functional components in SEG-BERT are node-order-invariant, i.e., the nodes’ learned representations are not dependent on the node orders. Therefore, regardless of the nodes’ orders, we can serialize node set \( \mathcal{V} \) into a sequence \([v_1, v_2, \ldots, v_{|\mathcal{V}|}]\). For the same graph instance, if it is fed to train/tune SEG-BERT for multiple times, the node order in the list can change arbitrarily without affecting the learning representations.

For each node in the list, e.g., \( v_i \), we can represent its raw features as a vector \( x_i = x(v_i) \in \mathbb{R}^{d_x \times 1} \), which can cover various types of information, e.g., node tags, attributes, textual descriptions and even images. Via certain embedding mappings, we can denote the embedded feature representation of \( v_i \)'s raw features as

\[
e^{(x)}_i = \text{Embed}(x_i) \in \mathbb{R}^{d_h \times 1}.
\] (1)

Depending on the input features, different approaches can be utilized to define the \( \text{Embed}(\cdot) \) function, e.g., CNN for image features, LSTM for textual features, positional embedding for tags and MLP for real-number features. In the case when the graph instance nodes have no raw attributes on nodes, a dummy zero vector will be used to fill in the embedding vector \( e^{(x)}_i \) entries by default.

To handle the graph instances without node attributes, in this paper, we will also extend the original GRAPH-BERT model by defining the node adjacency neighborhood embeddings. Meanwhile, to ensure such an embedding is node-order invariant, we will cast an artificially fixed node order on this embedding vector (which is fixed forever for the graph instance). For simplicity, we will just follow the node subscript index as the node orders in this paper. Formally, for each node \( v_i \), following the fixed artificial node order, we can denote its adjacency neighbors as a vector \( w_i = [w(i, j)]_{v_j \in \mathcal{V}} \in \mathbb{R}^{|\mathcal{V}| \times 1} \). Via several fully connected (FC) layers based mappings, we can represent the embedded representation of \( v_i \)'s adjacency neighborhood information as

\[
e^{(w)}_i = \text{FC-Embed}(w_i) \in \mathbb{R}^{d_h \times 1}.
\] (2)

The degrees of nodes can illustrate their basic proper-
ties (Chung et al. 2003; Bondy 1976), and according to (Lovász 1996) for the Markov chain or random walk on graphs, their final stationary distribution will be proportional to the nodes’ degrees. Node degree is also a node-order invariant actually. Formally, we can represent the degree of a node in the graph instance as \( D(v_i) \in \mathbb{N} \), and its embedding can be represented as

\[
e_{i}^{(d)} = \text{Position-Embed}(D(v_i)) = \left[ \sin \left( \frac{D(v_i)}{10000 \pi} \right), \cos \left( \frac{D(v_i)}{10000 \pi} \right) \right]_{l=0}^{\frac{d_h}{2}},
\]

where \( e_{i}^{(d)} \in \mathbb{R}^{d_h \times 1} \) and the vector index \( l \) will iterate through the vector to compute the entry values based on the \( \sin(\cdot) \) and \( \cos(\cdot) \) functions.

In addition to the node raw feature embedding, node adjacency neighborhood embedding and node degree embedding, we will also include the nodes’ Weisfeiler-Lehman role embedding vector in this paper, which effectively denotes the nodes’ global roles in the input graph. Nodes’ Weisfeiler-Lehman code is node-order-invariant, which denotes a positional property of the nodes actually. Formally, given a node \( v_i \) in the input graph instance, we can denote its pre-computed WL code as \( \text{WL}(v_i) \in \mathbb{N} \), whose corresponding embeddings can be denoted as

\[
e_{i}^{(r)} = \text{Position-Embed}(\text{WL}(v_i)) \in \mathbb{R}^{d_h \times 1}.
\]

\[\text{SEG-BERT doesn’t include the relative positional embedding and relative hop distance embedding used in (Zhang et al. 2020), as there exist no target node for the graph instances studied in this paper. Based on the above descriptions, we can represent the initially computed input embedding vectors of node \( v_i \) in graph \( G \) as}\]

\[
h_{i}^{(0)} = \text{sum}(e_{i}^{(x)}, e_{i}^{(w)}, e_{i}^{(d)}, e_{i}^{(r)}) \in \mathbb{R}^{d_h \times 1}.
\]

### 4.2. Graph Instance Size Unification Strategies

Different from (Zhang et al. 2020), where the sampled subgraphs all have the identical size, the graph instance input usually have different number of nodes instead. To handle such a problem, we design SEG-BERT with an instance size unification component in this paper. Formally, we can denote the input portal size of SEG-BERT used in this paper as \( k \), i.e., it can take the initial input embedding vectors of \( k \) nodes at a time. Depending on the input graph instance sizes, SEG-BERT will define the parameter \( k \) and handle the graph instances with different strategies:

- **Full-Input Strategy**: The input portal size \( k \) of SEG-BERT is defined as the largest graph instance size in the dataset, and dummy node padding will be used to expand all graph instances to \( k \) nodes (zero padding for the connections, raw attributes and other tags).

- **Padding/Pruning Strategy**: The input portal size \( k \) of SEG-BERT is assigned with a value slightly above the graph instance average size. For the graph instance input with less than \( k \) nodes, dummy node padding (zero padding) is used to expand the graph to \( k \) nodes; whereas for larger input graph instances, a \( k \)-node sub-graph (e.g., the first \( k \) nodes) will be extracted from them and the remaining nodes will be pruned.

- **Segment Shifting Strategy**: A fixed input portal size \( k \) will be pre-specified, which can be a very small number. For the graph instance input with node set \( \mathcal{V} \), the nodes will be divided into \( \left\lceil \frac{|\mathcal{V}|}{k} \right\rceil \) segments, and dummy node padding will be used for the last segment if necessary. SEG-BERT will shift along the segments to learn all the nodes representations in the graph instances.

Generally, the full-input strategy will use all the input graph nodes for representation learning, but for a small-graph set with a few number of extremely large graph instance(s), a very large \( k \) will be used in SEG-BERT, which may introduce unnecessary high time costs. The padding/pruning strategy balances the parameter \( k \) among all the graph instances and can learn effective representations in an efficient way, but it may have information loss for the pruned parts of some graph instances. Meanwhile, the segment shifting strategy can balance between the full-input strategy and padding/pruning strategy, which fuses the graph instance global information for representation learning with a small model input portal. More experimental tests of such different graph instance size unification strategies will also be explored with experiments on real-world benchmark datasets to be introduced in Section 6.

### 4.3. Segmented Graph-Transformer

To learn the graph instance representations, we will introduce the segmented graph-transformer in this part to update such nodes’ representations iteratively with \( D \) layers. Formally, based on the above descriptions, we can denote the current segment as \( s_j = [v_{j,1}, v_{j,2}, \cdots, v_{j,k}] \), whose initial input embeddings can be denoted as \( H_{j}^{(0)} = [h_{j,1}^{(0)}, h_{j,2}^{(0)}, \cdots, h_{j,k}^{(0)}]^T \in \mathbb{R}^{k \times d_h} \) and \( h_{j,l}^{(0)} \in \mathbb{R}^{d_h \times 1} \) is defined in Equation 6. Depending on the different unification strategies adopted, the segment can denote all the graph nodes, pruned/padded node subset or the node segments, respectively. For the segmented graph-transformer, it will update the segment representations iteratively for each layer \( l \in \{1, \cdots, D\} \) according to the following
equation:

\[ H^{(l)}_j = \text{G-Transformer}(H^{(l-1)}_j), \]
\[ = \text{Transformer}(H^{(l-1)}_j) + \text{G-Res}(H^{(l-1)}_j, X_j), \quad (6) \]

where Transformer(·) and G-Res(·) denote the transformer \cite{Vaswani2017} and graph residual terms \cite{ZhangM19}, respectively. Here, we need to add some remarks on the graph residual terms used in the model. Different from \cite{Zhang2020}, which integrates the target node residual terms to all the nodes in the batch, (1) for the graph instances without any node attributes, the nodes adjacency neighborhood vectors will be used to compute the residual terms, and (2) we compute the graph residual term with the whole graph instance instead in this paper (not merely for the target node).

Such a process will iterate through all the segments of the input graph instance nodes, and the finally learned node representations can be denoted as \( H^{(D)} = [h_1^{(D)}, h_2^{(D)}, \ldots, h_{|V|}^{(D)}]^{\top} \in \mathbb{R}^{|V| \times d_h} \). In this paper, for presentation simplicity, we just assume all the hidden layers are of the same dimension \( d_h \). Considering that we focus on learning the representations of the entire graph instance in this paper, \textsc{Seg-BERT} will integrate such node representations together to define the fused graph instance representation vector as follows:

\[ z = \text{Fusion}(H^{(D)}) = \frac{1}{|V|} \sum_{i=1}^{|V|} h_i^{(D)}. \quad (7) \]

Both vector \( z \) and matrix \( H^{(D)} \) will be outputted to the down-stream application tasks for the model training/tuning and graph instance representation learning, which will be introduced in the follow-up section in detail.

5. \textsc{Seg-BERT} Learning

In this part, we will focus on the pre-training and fine-tuning of \textsc{Seg-BERT} with several concrete graph instance learning tasks. To be more specific, we will introduce two unsupervised pre-training tasks to learn \textsc{Seg-BERT} based on node raw attribute reconstruction and graph structure recovery, which ensure the learned node and graph representations can capture both the raw attribute and structure information in the graph instances. After that, we will introduce one supervised tasks to fine-tune \textsc{Seg-BERT} for the graph instance classification, which will cast extra refinements on the learned node and graph representations.

5.1. Unsupervised Pre-Training

The pre-training tasks used in this paper will enable \textsc{Seg-BERT} to effectively capture both the node raw attributes and graph instance structures in the learned nodes and graph representation vectors. In the case where the graph instances have no node raw attributes, only graph structure recovery will be used for the pre-training. Formally, based on the learned representations \( H^{(D)} \) of all the nodes in the graph instance, e.g., \( G \), with several fully connected layers, we can project such latent representations to their raw features. Furthermore, by comparing the reconstructed feature matrix against the ground-truth raw features and minimizing their differences, we will be able to pre-train the \textsc{Seg-BERT} model with all the graph instances. Furthermore, based on the nodes learned representations, we can also compute the closeness (e.g., cosine similarity of the representation vectors) of the node pairs in the graph. Furthermore, compared against the ground-truth graph connection weight matrix, we can define the introduced loss term for graph structure recovery of all the graph instances to pre-train the \textsc{Seg-BERT} model.

5.2. Transfer and Fine-Tuning

When applying the pre-trained \textsc{Seg-BERT} and the learned representations in new application tasks, e.g., graph instance classification, necessary fine-tuning can be needed. Formally, we can denote the batch of labeled graph instance as \( T = \{(G_i, y_i)_i\} \), where \( y_i \) denotes the label vector of graph instance \( G_i \). Based on the fused representation vector \( z_i \) learned for graph instance \( G_i \in G \), we can further project it to the label vector with several fully connected layers together with the softmax normalization function,
Table 1: Experimental results of different comparison methods. For the results not reported in the previous works, we mark the corresponding entries with ‘*’ in the table. The entries are the accuracy scores (mean±std) achieved by the baseline methods with the 10 folds. For SEG-BERT (padding/pruning, none) and SEG-BERT (padding/pruning, raw), they denote SEG-BERT with the padding/pruning strategy and different graph residual terms (raw vs none). At the last row on SEG-BERT*, we show the best results obtained by SEG-BERT with all these three unification strategies.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>IMDB-B</th>
<th>IMDB-M</th>
<th>COLLAB</th>
<th>MUTAG</th>
<th>PROTEINS</th>
<th>PTC</th>
<th>NC11</th>
</tr>
</thead>
<tbody>
<tr>
<td># Graphs</td>
<td>1,000</td>
<td>1,500</td>
<td>5,000</td>
<td>188</td>
<td>1,113</td>
<td>344</td>
<td>4,110</td>
</tr>
<tr>
<td># Classes</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Avg. # Nodes</td>
<td>19.8</td>
<td>13.0</td>
<td>74.5</td>
<td>17.9</td>
<td>39.1</td>
<td>25.5</td>
<td>29.8</td>
</tr>
<tr>
<td>max # Nodes</td>
<td>136</td>
<td>89</td>
<td>492</td>
<td>28</td>
<td>620</td>
<td>109</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Accuracy (mean±std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WL (Shervashidze et al. 2011)</td>
<td>79.02±1.77</td>
</tr>
<tr>
<td>GK (Shervashidze et al. 2009)</td>
<td>74.68±0.49</td>
</tr>
<tr>
<td>DGK (Yanardag &amp; Vishwanathan 2015)</td>
<td>75.90±4.30</td>
</tr>
<tr>
<td>AWE (Ivanov &amp; Buneaev 2018)</td>
<td>82.19±0.18</td>
</tr>
<tr>
<td>PSCL (Niepert et al. 2016)</td>
<td>76.33±1.68</td>
</tr>
<tr>
<td>DAGCNC (Chen et al. 2019)</td>
<td>81.68±1.69</td>
</tr>
<tr>
<td>SPI-GCN (Atamna et al. 2019)</td>
<td>82.48±1.99</td>
</tr>
<tr>
<td>DGCCNN (Zhang et al. 2018)</td>
<td>82.40±1.47</td>
</tr>
<tr>
<td>GCAPS-CNN (Verma &amp; Zhang 2018)</td>
<td>82.72±2.38</td>
</tr>
<tr>
<td>CapsGNN (Zhang &amp; Chen 2019)</td>
<td>78.35±1.55</td>
</tr>
<tr>
<td>SEG-BERT (padding/pruning, none)</td>
<td>82.42±1.29</td>
</tr>
<tr>
<td>SEG-BERT (padding/pruning, raw)</td>
<td>82.42±1.29</td>
</tr>
<tr>
<td>SEG-BERT*</td>
<td>82.42±1.29</td>
</tr>
</tbody>
</table>

which can be denoted as

$$\hat{y}_i = \text{softmax} \left( FC(z_i) \right) \in \mathbb{R}^{d_y \times 1}. \quad (8)$$

Meanwhile, based on the known ground-truth label vector of graph instances in the training set, we can define the introduced loss term for the graph instance based on the cross-entropy term as follows:

$$\ell_{gc} = \sum_{(G_i, y_i) \in T} d_y \sum_j -y_i(j) \log \hat{y}_i(j). \quad (9)$$

By optimizing the above loss term, we will be able to refine SEG-BERT and the learned graph instance representations based on the application tasks specifically.

6. Experiments

To test the effectiveness of SEG-BERT, extensive experiments on real-world graph instance benchmark datasets will be done in this section. What’s more, we will also compare SEG-BERT with both the classic and state-of-the-art graph instance representation learning baseline methods to demonstrate its advantages.

Reproducibility: Both the datasets and source code used can be accessed via the github page[1]. Detailed information about the server used to run the model can be found at the footnote[2].

6.1. Dataset and Experimental Settings

Dataset Descriptions: The graph instance datasets used in this experiments include IMDB-Binary, IMDB-Multi and COLLAB, as well as MUTAG, PROTEINS, PTC and NCI1, which are all the benchmark datasets as used in [Yanardag & Vishwanathan 2015] and all the follow-up graph classification papers [Zhang & Chen 2019, Verma & Zhang 2018, Xu et al. 2018]. Among them, IMDB-Binary, IMDB-Multi and COLLAB are the social graph datasets; whereas the remaining ones are the bio-graph datasets. Basic statistical information about the datasets is also available at the top of Table[1].

Comparison Baselines: The comparison baseline methods used in this paper include (1) conventional graph kernel based methods, e.g., Weisfeiler-Lehman subtree kernel (WL) [Shervashidze et al. 2011] and graphlet count kernel (GK) [Shervashidze et al. 2009]; (2) existing deep learning based methods, e.g., Deep Graph Kernel (DGK) [Yanardag & Vishwanathan 2015] and AWE [Ivanov & Buneaev 2018]. PATCHY-SAN (PSCN) [Niepert et al. 2016];

---


[2]GPU Server: ASUS X99-E WS motherboard, Intel Core i7 CPU 6850K@3.6GHz (6 cores, 40 PCIe lanes), 3 Nvidia GeForce GTX 1080 Ti GPU (11 GB buffer each), 128 GB DDR4 memory and 128 GB SSD swap.
and (3) state-of-the-art deep learning methods, e.g., Dual Attention Graph Convolutional Network (DAGCN) (Chen et al., 2019), Simple Permutation-Invariant Graph Convolutional Network (SPI-GCN) (Atamna et al., 2019), Graph Capsule CNN (GCAPS-CNN) (Verma & Zhang, 2018) and Deep Graph CNN (DGCNN) (Zhang et al., 2018), and Capsule Graph Neural Network (CapsGNN) (Zhang & Chen, 2019). Evaluation Metric: The learning performance of these methods will be evaluated by Accuracy as the metric.

Experimental Settings: In the experiments, we will first compare SEG-BERT with the padding/pruning strategy for input size unification against the baseline methods, which can help test if a sub-structures in graph instances can capture the characteristics of the whole graph instance or not. The other two strategies will be discussed at the end of this section in detail. For the input portal size $k$, it is assigned with a value slightly larger than the graph instance average sizes of each dataset. All the graph instances in the datasets will be partitioned into train, validate, testing sets according to the ratio 8:1:1 with the 10-fold cross validation, where the validation set is used for parameter tuning and selection. Meanwhile, for fair comparisons (existing works use 9:1 train/test partition without validation set), such 10% validation set will also be used as training data as well to further fine-tune SEG-BERT after the parameter selection.

Default Model Parameter Settings: If not clearly specified, the results reported in this paper are based on the following parameter settings of SEG-BERT: input portal size: $k = 25$ (MUTAG), $k = 50$ (IMDB-Binary, IMDB-Multi, NCI1, PTC) and $k = 100$ (COLLAB, PROTEINS); hidden size: 32; attention head number: 2; hidden layer number: $D = 2$; learning rate: 0.0005 (PTC) and 0.0001 (others); weight decay: $5e^{-4}$; intermediate size: 32; hidden dropout rate: 0.5; attention dropout rate: 0.3; graph residual term: raw/none; training epoch: 500 (early stop if necessary to avoid over-fitting).

### 6.2. Graph Instance Classification Results

Model Train Records: As illustrated in Figure 2, we show the learning records (training/testing accuracy) of SEG-BERT on both the IMDB-Multi social graph and the Protein bio-graph datasets. According to the plots, graph instance classification is very different from classification task on other data types, as the model can get over-fitting easily. Similar phenomena have been observed for most comparison methods on the other datasets as well. Even though the default training epoch is 500 mentioned above, an early stop of training SEG-BERT is usually necessary. In this paper, we decide the early-stop learning epoch number based on the partitioned validation set.

**Main Results:** The main learning results of all the comparison methods are provided in Table 2. The SEG-BERT method shown here adopts the padding/pruning strategy to handle the graph data input. The residual terms adopted are indicated by the SEG-BERT method name in the parentheses. According to the evaluation results, SEG-BERT can greatly improve the learning performance on most of the benchmark datasets. For instance, the accuracy score of SEG-BERT (none) on IMDB-Binary is 75.40, which is much higher than many of the state-of-the-art baseline methods, e.g., DGCNN, GCAPS-CNN and CapsGNN. Similarly learning performance advantages have also been observed on the other datasets, except NCI1.

For the raw residual term used in SEG-BERT, for some of the datasets as shown in Table 2 e.g., MUTAG, it can improve the learning performance; whereas for the remaining ones, its effectiveness is not very significant. The main reason can be for most of the graph instances studied here they don’t have node attributes and the discrete nodes adjacency neighborhood embeddings can be very sparse, which renders the computed residual terms to be less effective for performance improvement.

**Graph Size Unification Strategy Analysis:** The analyses of the different graph size unification strategies is provided in Table 2. For SEG-BERT with the full-input strategy on COLLAB, PROTEIN and NCI1 (with more instances and large max graph sizes), the training time costs are too high (> 3 days to run the 10 folds). So, the results on these three datasets are not shown here. As illustrated in Table 2 the performance of full-input is better than padding/pruning, but it also consumes the highest time cost. The padding/pruning strategy has the lowest time costs but its performance is slightly lower than full-input and segment shifting. Meanwhile, the learning performance of segment shifting balances between full-input and...
padding/pruning, which can even out-perform full-input as it introduces far less dummy paddings in the input.

7. Conclusion

In this paper, we have introduced a new graph neural network model, namely SEG-BERT, for graph instance representation learning. With several significant modifications, the re-designed SEG-BERT has no node-order-variant inputs or function components, which can handle the node orderless property very well. SEG-BERT has an extendable architecture. For large-sized graph instance input, SEG-BERT will divide the nodes into segments, whereas all the nodes in the graph will be used for the representation learning of each segment. We have tested the effectiveness of SEG-BERT on several concrete application tasks, and the experimental results demonstrate that SEG-BERT can outperform the state-of-the-art graph instance representation learning models effectively.

References

Atamna, A., Sokolovska, N., and CRIVELLO, J.-C. SPIGCN: A Simple Permutation-Invariant Graph Convolutional Network. working paper or preprint, April 2019. URL https://hal.archives-ouvertes.fr/hal-02093451


