Segmented GRAPH-BERT for Graph Instance Modeling

Jiawei Zhang [1]

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

---

[1]IFM Lab, Department of Computation, Florida State University, Tallahassee, FL, USA.. Correspondence to: Jiawei Zhang <jiawei@ifmlab.org>.

graph instance input to fit the model configuration. SEG-BERT has a segmented architecture and introduces three strategies to unify the graph instance sizes, i.e., *full-input*, *padding/pruning* and *segment shifting*, which help feed SEG-BERT with the whole input graph instances (or graph segments) for representation learning. (c) SEG-BERT re-introduces graph links as initial inputs. Considering that some graph instances may have no node attributes, SEG-BERT re-introduces the graph links (in an artificial fixed node order) back as one part of the initial inputs for graph instance representation learning. (d) SEG-BERT uses the graph residual terms of the whole graph instance. Since we focus on learning the representations of the whole graph instances, the graph residual terms involved in SEG-BERT will be defined for all the nodes in the graph (or segments) instead of merely for the target nodes (Zhang et al., 2020).

SEG-BERT is pre-trainable in an unsupervised manner, and the pre-trained model can be further transferred to new tasks directly or with necessary fine-tuning. To be more specific, in this paper, we will explore the pre-training and fine-tuning of SEG-BERT for several graph instance studies, e.g., *node attribute reconstruction*, *graph structure recovery*, and *graph instance classification*. Such explorations will help construct the functional model pipelines for graph instance learning, and avoid the unnecessary redundant learning efforts and resource consumptions.

We summarize our contributions of this paper as follows:

- **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; Veličković 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), the inherently sequential nature precludes parallelization within training examples. To address such a problem, a brand new representation learning model solely based on attention mechanisms, i.e., the TRANSFORMER, is 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., $\mathbf{X}$) to denote matrices, and upper case calligraphic letters (e.g., $\mathcal{X}$) to denote sets or high-order tensors. Given a matrix $\mathbf{X}$, we denote $\mathbf{X}(i, :)$ and $\mathbf{X}(:, j)$ as its i_{th} row and j_{th} column, respectively. The (i_{th}, j_{th}) entry of matrix $\mathbf{X}$ can be denoted as either $\mathbf{X}(i, j)$. We use $\mathbf{X}^\top$ and $\mathbf{x}^\top$ to represent the transpose of matrix $\mathbf{X}$ and vector $\mathbf{x}$. For vector $\mathbf{x}$, we represent its L_p-norm as $\|\mathbf{x}\|_p = (\sum_i |\mathbf{x}(i)|^p)^{\frac{1}{p}}$. The Frobenius-norm of matrix $\mathbf{X}$ is represented as $\|\mathbf{X}\|_F = (\sum_{i,j} |\mathbf{X}(i,j)|^2)^{\frac{1}{2}}$. The element-wise product of vectors $\mathbf{x}$ and $\mathbf{y}$ of the same di-

mension is represented as $\mathbf{x} \otimes \mathbf{y}$, whose concatenation is represented as $\mathbf{x} \sqcup \mathbf{y}$.

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 $G = (\mathcal{V}, \mathcal{E}, w, x)$, where $\mathcal{V}$ and $\mathcal{E}$ denote the sets of nodes and links in the graph, respectively. Mapping $w : \mathcal{E} \to \mathbb{R}$ projects links in the graph to their corresponding weight. For unweighted graphs, we will have $w(e_{i,j}) = 1, \forall e_{i,j} \in \mathcal{E}$ and $w(e_{i,j}) = 0, \forall e_{i,j} \in \mathcal{V} \times \mathcal{V} \setminus \mathcal{E}$. For the nodes in the graph instance, they may also be associated with certain attributes, which can be represented by mapping $x : \mathcal{V} \to \mathcal{X}$ (here, $\mathcal{X} = \mathbb{R}^{d_x}$ 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 $\mathbf{w}_i = [w(e_{i,j})]_{v_j \in \mathcal{V}} \in \mathbb{R}^{|\mathcal{V}| \times 1}$. Meanwhile, the raw attribute vector representation of v_i can also be simplified as $\mathbf{x}_i = x(v_i)$. The size of graph instance G can be denoted as the number of involved nodes, i.e., $|\mathcal{V}|$. 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 n labeled graph instances studied in this paper as set $\mathcal{G} = \{(G_i, \mathbf{y}_i)\}_{i=1}^n$, where $\mathbf{y}_i \in \mathcal{Y}$ denotes the label vector of G_i (here, $\mathcal{Y} = \mathbb{R}^{d_y}$ 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 $f : \mathcal{G} \to \mathcal{R}^{d_h}$ 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-
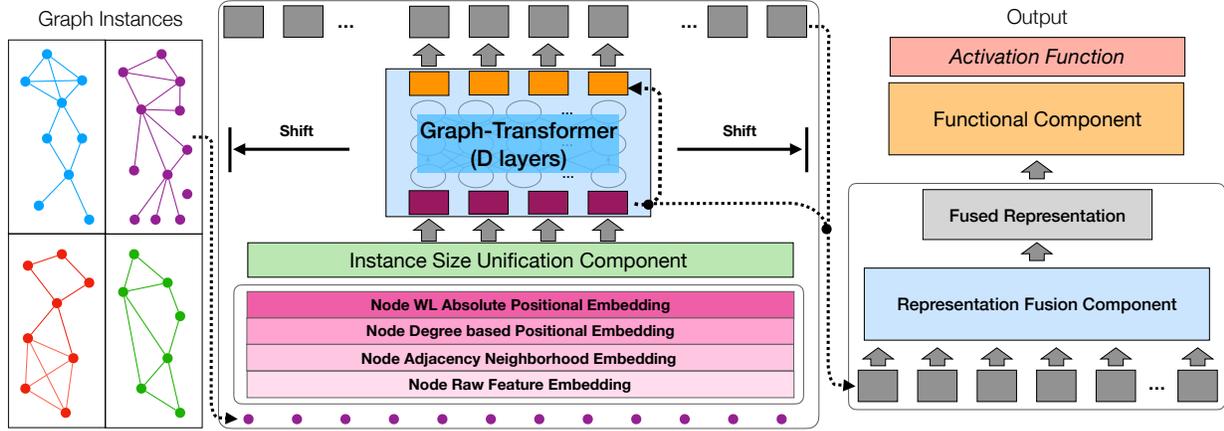
Figure 1: An Illustration of the SEG-BERT Model for Graph Instance Representation Learning.

gories of information inputs, (c) f can be effectively pretrained with the unsupervised learning tasks, and (d) representations learned by f can also be transferred to the downstream application tasks.

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

$[v_1, v_2, \cdots, v_{|\mathcal{V}|}]$. 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 $\mathbf{x}_i = x(v_i) \in \mathbb{R}^{d_x \times 1}$, 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

$$\mathbf{e}_i^{(x)} = \text{Embed}\left(\mathbf{x}_i\right) \in \mathbb{R}^{d_h \times 1}. \tag{1}$$

Depending on the input features, different approaches can be utilized to define the 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 $\mathbf{e}_i^{(x)}$ 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 $\mathbf{w}_i = [w(i,j)]_{v_j \in \mathcal{V}} \in \mathbb{R}^{|\mathcal{V}| \times 1}$. Via several fully connected (FC) layers based mappings, we can represent the embedded representation of v_i's adjacency neighborhood information as

$$\mathbf{e}_i^{(w)} = \text{FC-Embed}\left(\mathbf{w}_i\right) \in \mathbb{R}^{d_h \times 1}. \tag{2}$$

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 v_i in the graph instance as $D(v_i) \in \mathbb{N}$, and its embedding can be represented as

$$\mathbf{e}_i^{(d)} = \text{Position-Embed}\left(D(v_i)\right)$$
$$= \left[sin\left(\frac{D(v_i)}{10000^{\frac{2l}{d_h}}}\right), cos\left(\frac{D(v_j)}{10000^{\frac{2l+1}{d_h}}}\right)\right]_{l=0}^{\left\lfloor \frac{d_h}{2} \right\rfloor},$$
(3)

where $\mathbf{e}_i^{(d)} \in \mathbb{R}^{d_h \times 1}$ 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

$$\mathbf{e}_i^{(r)} = \text{Position-Embed}\left(\text{WL}(v_i)\right) \in \mathbb{R}^{d_h \times 1}. \qquad (4)$$

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

$$\mathbf{h}_i^{(0)} = \text{sum}\left(\mathbf{e}_i^{(x)}, \mathbf{e}_i^{(w)}, \mathbf{e}_i^{(d)}, \mathbf{e}_i^{(r)}\right) \in \mathbb{R}^{d_h \times 1}. \quad (5)$$

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 $\mathbf{H}_j^{(0)} = [\mathbf{h}_{j,1}^{(0)}, \mathbf{h}_{j,2}^{(0)}, \cdots, \mathbf{h}_{j,k}^{(0)}]^\top \in \mathbb{R}^{k \times d_h}$ and $\mathbf{h}_{j,i}^{(0)} \in \mathbb{R}^{d_h \times 1}$ is defined in Equation (5). 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 $\forall l \in \{1, \cdots, D\}$ according to the following

(a) Train Acc (IMDB)   (b) Test Acc (IMDB)   (c) Train Acc (Proteins)   (d) Test Acc (Proteins)
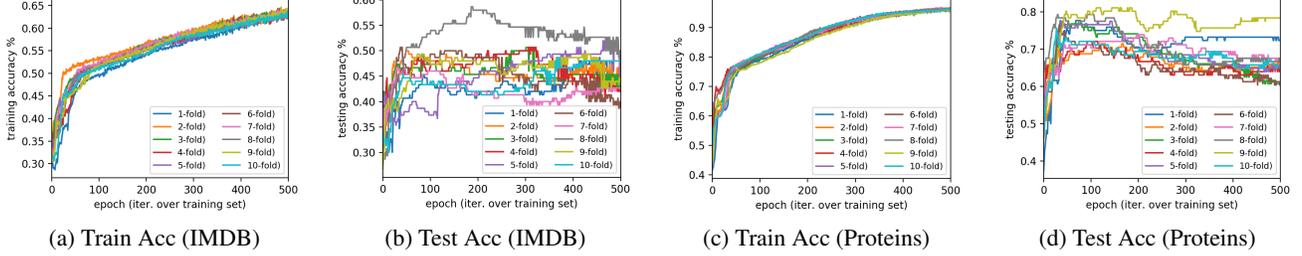
Figure 2: Learning records of SEG-BERT on the IMDB-Multi social-graph dataset and the Proteins bio-graph dataset. For the graph data with discrete structures, most of the comparison models studied in the experiments will overfit the training data easily and an early stop is usually necessary. The x axis: iteration, and the y axis: training/testing loss.

equation:

$$
\begin{aligned}
\mathbf{H}_j^{(l)} &= \text{G-Transformer}(\mathbf{H}_j^{(l-1)}), \\
&= \text{Transformer}(\mathbf{H}_j^{(l-1)}) + \text{G-Res}(\mathbf{H}_j^{(l-1)}, \mathbf{X}_j),
\end{aligned}
\tag{6}
$$

where Transformer$(\cdot)$ and G-Res$(\cdot)$ denote the transformer (Vaswani et al., 2017) and graph residual terms (Zhang & Meng, 2019), respectively. Here, we need to add some remarks on the graph residual terms used in the model. Different from (Zhang et al., 2020), 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 $\mathbf{H}^{(D)} = [\mathbf{h}_1^{(D)}, \mathbf{h}_2^{(D)}, \cdots, \mathbf{h}_{|\mathcal{V}|}^{(D)}]^\top \in \mathbb{R}^{|\mathcal{V}| \times d_h}$. 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, SEG-BERT will integrate such node representations together to define the fused graph instance representation vector as follows:

$$
\mathbf{z} = \text{Fusion}\left(\mathbf{H}^{(D)}\right) = \frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} \mathbf{h}_1^{(D)}.
\tag{7}
$$

Both vector $\mathbf{z}$ and matrix $\mathbf{H}^{(D)}$ 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. SEG-BERT Learning

In this part, we will focus on the pre-training and fine-tuning of SEG-BERT with several concrete graph instance learning tasks. To be more specific, we will introduce two unsupervised pre-training tasks to learn 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 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 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 $\mathbf{H}^{(D)}$ 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 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 SEG-BERT model.

5.2. Transfer and Fine-Tuning

When applying the pre-trained 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 $\mathcal{T} = \{(G_i, \mathbf{y}_i)\}_i$, where $\mathbf{y}_i$ denotes the label vector of graph instance G_i. Based on the fused representation vector $\mathbf{z}_i$ learned for graph instance $G_i \in \mathcal{G}$, 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.

Datasets	IMDB-B	IMDB-M	COLLAB	MUTAG	PROTEINS	PTC	NCI1
# Graphs	1,000	1,500	5,000	188	1,113	344	4,110
# Classes	2	3	3	2	2	3	2
Avg. # Nodes	19.8	13.0	74.5	17.9	39.1	25.5	29.8
max # Nodes	136	89	492	28	620	109	111
Methods	Accuracy (mean±std)						
WL (Shervashidze et al., 2011)	73.40±4.63	49.33±4.75	**79.02±1.77**	82.05±0.36	74.68±0.49	59.90±4.30	**82.19±0.18**
GK (Shervashidze et al., 2009)	65.87±0.98	43.89±0.38	72.84±0.28	81.58±2.11	71.67±0.55	57.26±1.41	62.28±0.29
DGK (Yanardag & Vishwanathan, 2015)	66.96±0.56	44.55±0.52	73.09±0.25	87.44±2.72	75.68±0.54	60.08±2.55	**80.31±0.46**
AWE (Ivanov & Burnaev, 2018)	**74.45±5.83**	**51.54±3.61**	73.93±1.94	87.87±9.76	−	−	−
PSCN (Niepert et al., 2016)	71.00±2.29	45.23±2.84	72.60±2.15	**88.95±4.37**	75.00±2.51	62.29±5.68	76.34±1.68
DAGCN (Chen et al., 2019)	−	−	−	87.22±6.10	76.33±4.3	62.88±9.61	**81.68±1.69**
SPI-GCN (Atamna et al., 2019)	60.40±4.15	44.13±4.61	−	84.40±8.14	72.06±3.18	56.41±5.71	64.11±2.37
DGCNN (Zhang et al., 2018)	70.03±0.86	47.83±0.85	73.76±0.49	85.83±1.66	75.54±0.94	58.59±2.47	74.44±0.47
GCAPS-CNN (Verma & Zhang, 2018)	71.69±3.40	48.50±4.10	77.71±2.51	−	**76.40±4.17**	66.01±5.91	**82.72±2.38**
CapsGNN (Zhang & Chen, 2019)	73.10±4.83	50.27±2.65	**79.62±0.91**	86.67±6.88	76.28±3.63	−	78.35±1.55
SEG-BERT(padding/pruning, none)	**75.40±2.29**	**52.27±1.55**	78.42±1.29	89.24±7.78	77.09±4.15	68.86±4.17	70.15±1.84
SEG-BERT(padding/pruning, raw)	**74.70±3.74**	50.60±3.03	74.90±1.78	89.80±6.71	76.28±2.91	64.84±6.77	68.10±2.55
SEG-BERT*	**77.20±3.09**	**53.40±2.12**	78.42±1.29	90.85±6.58	77.09±4.15	68.86±4.17	70.15±1.84

which can be denoted as

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

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 corss-entropy term as follows:

$$\ell_{gc} = \sum_{(G_i, \mathbf{y}_i) \in \mathcal{T}} \sum_{j=1}^{d_y} -\mathbf{y}_i(j) \log \hat{\mathbf{y}}_i(j). \quad (9)$$

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 & Burnaev, 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.

---

[1]https://github.com/jwzhanggy/SEG-BERT

Table 2: Evaluation results of different graph instance size unification strategies in SEG-BERT. No residual terms are used here, and the default epoch number is 500. The time denotes the average time cost for mode training in the 10 folds. For SEG-BERT with the *segment shifting* strategy, the default parameter k is set as 20.

Strategies	IMDB-B			IMDB-M			MUTAG			PTC		
	Accuracy	k	Time(s)	Accuracy	k	Time(s)	Accuracy	k	Time(s)	Accuracy	k	Time(s)
Full-Input	**76.90±1.76**	136	2808.70	**53.33±2.53**	89	2397.42	**90.85±6.58**	28	55.73	68.01±4.23	109	700.98
Padding/Pruning	75.40±2.29	50	**1312.42**	52.27±1.55	50	**1654.66**	89.24±7.78	25	**55.19**	**68.86±4.17**	50	**223.47**
Segment Shifting	**77.20±3.09**	20	1525.97	**53.40±2.12**	20	1730.08	90.29±7.74	20	88.40	66.54±4.18	20	295.83

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 of all the comparison methods are provided in Table 1. 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 1, 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 introduce far less dummy paddings in the input.

7. Conclusion

In this paper, we have introduce 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. SPI-GCN: A Simple Permutation-Invariant Graph Convolutional Network. working paper or preprint, April 2019. URL https://hal.archives-ouvertes.fr/hal-02093451.

Bondy, J. A. *Graph Theory With Applications*. Elsevier Science Ltd., GBR, 1976. ISBN 0444194517.

Chen, F., Pan, S., Jiang, J., Huo, H., and Long, G. DAGCN: dual attention graph convolutional networks. *CoRR*, abs/1904.02278, 2019. URL http://arxiv.org/abs/1904.02278.

Chung, F., Lu, L., and Vu, V. Spectra of random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 100(11):6313–6318, 2003. ISSN 0027-8424. doi: 10.1073/pnas.0937490100. URL https://www.pnas.org/content/100/11/6313.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3844–3852. Curran Associates, Inc., 2016.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Hammond, D. K., Vandergheynst, P., and Gribonval, R. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129?150, Mar 2011. ISSN 1063-5203. doi: 10.1016/j.acha.2010.04.005. URL http://dx.doi.org/10.1016/j.acha.2010.04.005.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Comput.*, 9(8), November 1997.

Ivanov, S. and Burnaev, E. Anonymous walk embeddings. *CoRR*, abs/1805.11921, 2018. URL http://arxiv.org/abs/1805.11921.

Jiang, J., Cui, Z., Xu, C., and Yang, J. Gaussian-induced convolution for graphs. *CoRR*, abs/1811.04393, 2018. URL http://arxiv.org/abs/1811.04393.

Kim, Y. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL https://www.aclweb.org/anthology/D14-1181.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

Kriege, N. M., Giscard, P.-L., and Wilson, R. On valid optimal assignment kernels and applications to graph classification. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 1623–1631. Curran Associates, Inc., 2016.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized BERT pre-training approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Lovász, L. Random walks on graphs: A survey. In Miklós, D., Sós, V. T., and Szőnyi, T. (eds.), *Combinatorics, Paul Erdős is Eighty*, volume 2, pp. 353–398. János Bolyai Mathematical Society, Budapest, 1996.

Mallea, M. D. G., Meltzer, P., and Bentley, P. J. Capsule neural networks for graph classification using explicit tensorial graph representations. *CoRR*, abs/1902.08399, 2019. URL http://arxiv.org/abs/1902.08399.

Meltzer, P., Mallea, M. D. G., and Bentley, P. J. Pinet: A permutation invariant graph neural network for graph classification. *CoRR*, abs/1905.03046, 2019. URL http://arxiv.org/abs/1905.03046.

Meng, L. and Zhang, J. Isonn: Isomorphic neural network for graph representation learning and classification. *CoRR*, abs/1907.09495, 2019. URL http://arxiv.org/abs/1907.09495.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., and Jaiswal, S. graph2vec: Learning distributed representations of graphs. *CoRR*, abs/1707.05005, 2017. URL http://arxiv.org/abs/1707.05005.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. *CoRR*, abs/1605.05273, 2016. URL http://arxiv.org/abs/1605.05273.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Ranjan, E., Sanyal, S., and Talukdar, P. P. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. *arXiv preprint arXiv:1911.07979*, 2019.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. Efficient graphlet kernels for large graph comparison. In van Dyk, D. and Welling, M. (eds.), *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pp. 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR. URL http://proceedings.mlr.press/v5/shervashidze09a.html.

Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12(null): 2539?2561, November 2011. ISSN 1532-4435.

Sun, Y., Wang, S., Li, Y., Feng, S., Tian, H., Wu, H., and Wang, H. ERNIE 2.0: A continual pre-training framework for language understanding. *CoRR*, abs/1907.12412, 2019. URL http://arxiv.org/abs/1907.12412.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

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

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018. URL http://arxiv.org/abs/1810.00826.

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ?15, pp. 1365?1374, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. doi: 10.1145/2783258.2783417. URL https://doi.org/10.1145/2783258.2783417.

Zhang, J. Graph Neural Networks for Small Graph and Giant Network Representation Learning: An Overview. *arXiv e-prints*, art. arXiv:1908.00187, Jul 2019.

Zhang, J. and Meng, L. Gresnet: Graph residual network for reviving deep gnns from suspended animation. *ArXiv, abs/1909.05729*, 2019.

Zhang, J., Zhang, H., Sun, L., and Xia, C. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

Zhang, X. and Chen, L. Capsule graph neural network. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Byl8BnRcYm.