GRAPH-BERT: Only Attention is Needed for Learning Graph Representations

Jiawei Zhang[⋆] , **Haopeng Zhang**[⋆] , **Li Sun**[¶] , **Congying Xia**[†]

[⋆]IFM Lab, Florida State University, Tallahassee, FL, USA

[¶]Institute for Data Science, Tsinghua University, Beijing, China

[†]University of Illinois at Chicago, IL, USA.

{ jiawei, haopeng }@ifmlab.org, l.sun@bupt.edu.cn, cxia8@uic.edu

Abstract

The dominant graph neural networks (GNNs) overrely on the graph links, several serious performance problems with which have been witnessed already, e.g., *suspended animation problem* and *over-smoothing problem*. What's more, the inherently inter-connected nature precludes parallelization within the graph, which becomes critical for large-sized graph, as memory constraints limit batching across the nodes. In this paper, we will introduce a new graph neural network, namely GRAPH-BERT (Graph based BERT), solely based on the attention mechanism without any graph convolution or aggregation operators. Instead of feeding GRAPH-BERT with the complete large input graph, we propose to train GRAPH-BERT with sampled linkless subgraphs within their local contexts. GRAPH-BERT can be learned effectively in a standalone mode. Meanwhile, a pre-trained GRAPH-BERT can also be transferred to other application tasks directly or with necessary fine-tuning if any supervised label information or certain application oriented objective is available. We have tested the effectiveness of GRAPH-BERT on several graph benchmark datasets. Based the pre-trained GRAPH-BERT with the *node attribute reconstruction* and *structure recovery* tasks, we further fine-tune GRAPH-BERT on *node classification* and *graph clustering* tasks specifically. The experimental results have demonstrated that GRAPH-BERT can out-perform the existing GNNs in both the learning effectiveness and efficiency.

1 Introduction

Graph provides a unified representation for many inter-connected data in the real-world, which can model both the diverse attribute information of the node entities and the extensive connections among these nodes. For instance, the human brain imaging data, online social media and bio-medical molecules can all be represented as graphs, i.e., the brain graph [Meng and Zhang, 2019], social graph [Ugander *et al.*, 2011] and protein-protein-interaction (PPI) graph [Jin *et al.*, 2018], respectively. Traditional machine learning models can hardly be applied to the graph data directly, which usually take the feature vectors as the inputs. Viewed in such a perspective, learning the representations of the graph structured data is an important research task.

In recent years, great efforts have been devoted to designing new graph neural networks (GNNs) for effective graph representation learning. Besides the network embedding models, e.g., node2vec [Grover and Leskovec, 2016] and deepwalk [Perozzi *et al.*, 2014a], the recent graph neural networks, e.g., GCN [Kipf and Welling, 2016], GAT [Veličković *et al.*, 2018] and LOOPYNET [Zhang, 2018], are also becoming much more important, which can further refine the learned representations for specific application tasks, e.g., node classification. Meanwhile, most of these existing graph representation learning models are still based on the graph structures, i.e., the links among the nodes. Via necessary neighborhood information aggregation or convolutional operators along the links, nodes' representations learned by such approaches can preserve the graph structure information.

However, several serious learning performance problem, e.g., suspended animation problem [Zhang and Meng, 2019] and over-smoothing problem [Li *et al.*, 2018], with the existing GNN models have also been witnessed in recent years. According to [Zhang and Meng, 2019], for the GNNs based on the approximated graph convolutional operators [Hammond *et al.*, 2011], as the model architecture goes deeper and reaches certain limit, the model will not respond to the training data and suffers from the suspended animation problem. Meanwhile, the node representations obtained by such deep models tend to be over-smoothed and also become indistinguishable [Li *et al.*, 2018]. Both of these two problems greatly hinder the applications of GNNs for deep graph representation learning tasks. What's more, the inherently inter-connected nature precludes parallelization within the graph, which becomes critical for large-sized graph input, as memory constraints limit batching across the nodes. It still remains an open problem by this context so far.

To address the above problems, in this paper, we will propose a new graph neural network model, namely GRAPH-BERT (Graph based BERT). Model GRAPH-BERT will be trained with sampled nodes together with their context (which are called linkless subgraphs in this paper) batches from the input large-sized graph data. Distinct from the existing GNN models, in the representation learning process, GRAPH-BERT utilizes no links in such sampled batches, which will be

purely based on the attention mechanisms instead [Vaswani *et al.*, 2017; Devlin *et al.*, 2018]. Therefore, GRAPH-BERT can get rid of the aforementioned learning effectiveness and efficiency problems with existing GNN models promisingly.

What's more, compared with computer vision [He *et al.*, 2018] and natural language processing [Devlin *et al.*, 2018], graph neural network pre-training and fine-tuning are still not common practice by this context so far. The main obstacles that prevent such operations can be due to the diverse input graph structures and the extensive connections among the nodes. Also the diverse learning task objectives also prevents the transfer of GNNs across different tasks. Since GRAPH-BERT doesn't really rely on the graph links at all, in this paper, we will investigate the transfer of pre-trained GRAPH-BERT on new learning tasks and new graph datasets (with necessary fine-tuning), which will also help construct the functional pipeline of models in graph learning.

We summarize our contributions of this paper as follows:

- **New GNN Model**: In this paper, we introduce a new GNN model GRAPH-BERT for graph data representation learning. GRAPH-BERT doesn't rely on the graph structures for representation learning and can effectively address the learning effectiveness problems, i.e., suspended animation problem and over-smoothing problem. Also GRAPH-BERT is trainable with sampled linkless subgraphs (i.e., target node with context), which is more efficient than existing GNNs constructed for the complete input graph.
- **Unsupervised Pre-Training**: Given the input unlabeled graph, we will pre-train GRAPH-BERT based on to two common tasks in graph studies, i.e., node attribute reconstruction and structure recovery. Node attribute recovery ensures the learned node representations can capture the input attribute information; whereas structure recovery can further ensure GRAPH-BERT learned with linkless subgraphs can still maintain both the graph local and global structure properties.
- **Fine-Tuning and Transfer**: Depending on the specific application task objectives, the GRAPH-BERT model can be further fine-tuned to adapt the learned representations to the task requirements, e.g., node classification and graph clustering. Meanwhile, GRAPH-BERT trained on one graph data can also be transferred and applied to other graph data as well, which allows the construction of functional pipelines for graph learning.

The remaining parts of this paper are organized as follows. We will introduce the related work in Section 2. Detailed information about the GRAPH-BERT model will be introduced in Section 3, whereas the pre-training and fine-tuning of GRAPH-BERT will be introduced in Section 4 in detail. The effectiveness of GRAPH-BERT will be tested in Section 5. Finally, we will conclude this paper in Section 6.

2 Related Work

To make this paper self-contained, we will introduce some related topics here on GNNs, TRANSFORMER and BERT.

Graph Neural Network: Representative examples of GNNs proposed by present include GCN [Kipf and Welling, 2016], GraphSAGE [Hamilton *et al.*, 2017] and LOOPYNET [Zhang, 2018], based on which various extended models, e.g., GAT [Veličković *et al.*, 2018], and other variants [Sun *et al.*, 2019; Klicpera *et al.*, 2018], have been introduced as well. As mentioned above, GCN, GAT and their variants are all based on the approximated graph convolutional operator [Hammond *et al.*, 2011], which may lead to the suspended animation problem [Zhang and Meng, 2019] and over-smoothing problem [Li *et al.*, 2018] for deep model architectures. Theoretic analyses of the reasons are provided in [Li *et al.*, 2018; Zhang and Meng, 2019; Gürel *et al.*, 2019]. To handle such problems, [Zhang and Meng, 2019] proposes a method based on graph residual learning; [Li *et al.*, 2018] proposes to adopt residual/dense connections and dilated convolutions into the GCN architecture. Several other works [Sun *et al.*, 2019; Huang and Carley, 2019] seek to involve the recurrent network for deep node representation learning instead.

BERT and TRANSFORMER: In NLP, the dominant sequence transduction models are based on complex recurrent [Hochreiter and Schmidhuber, 1997; Chung *et al.*, 2014] or convolutional neural networks [Kim, 2014]. However, the inherently sequential nature precludes parallelization within training examples. Therefore, in [Vaswani *et al.*, 2017], the authors propose a new simple network architecture, the TRANSFORMER, based solely on attention mechanisms, dispensing 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. In recent years, TRANSFORMER and BERT based learning approaches have been used extensively in various learning tasks [Dai *et al.*, 2019; Lan *et al.*, 2019; Shang *et al.*, 2019]. In this paper, we are the first to extend the BERT based model for graph representation learning.

Due to the limited space, only a brief introduction can be provided here. Readers may also refer to page[1] and page[2] for more information on the state-of-the-art work on these topics.

3 Method

In this section, we will introduce the detailed information about the GRAPH-BERT model. As illustrated in Figure 1, GRAPH-BERT involves several parts: (1) linkless subgraph batching, (2) node input embedding, (3) graph transformer based encoder, (4) representation fusion, and (5) the functional component. The representations learned by the the graph transformer model will be aggregated as the result for the target nodes. In this section, we will introduce these key parts in great detail, whereas the pre-training and fine-tuning of GRAPH-BERT will be introduced in the following section.

3.1 Linkless Subgraph Batching

Prior to talking about the subgraph batching method, we would like to present the problem settings first. Formally, we can denote the input graph data as $G = (\mathcal{V}, \mathcal{E}, w, x, y)$, where $\mathcal{V}$ and $\mathcal{E}$ denote the sets of nodes and links in graph G. Mapping $w : \mathcal{E} \rightarrow \mathbb{R}$ projects links to their weight; whereas mappings $x : \mathcal{V} \rightarrow \mathcal{X}$ and $y : \mathcal{V} \rightarrow \mathcal{Y}$ can project the nodes to their raw features and labels instead. The above term defines a general graph concept. If the studied G is unweighted, we will have $w(v_i, v_j) = 1, \forall (v_i, v_j) \in \mathcal{E}$; otherwise, we have

[1] https://paperswithcode.com/area/graphs

[2] https://paperswithcode.com/area/natural-language-processing

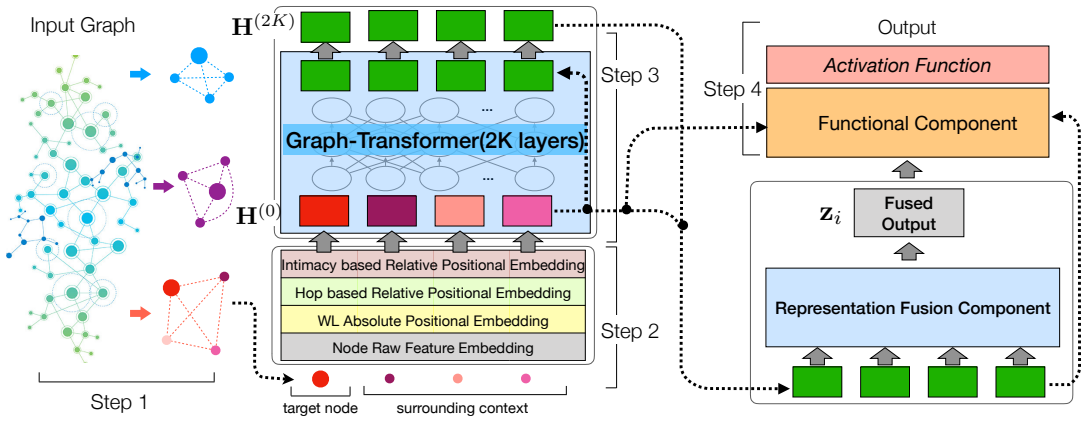


Figure 1: Architecture of the GRAPH-BERT Model. (Part 1: linkless subgraph batching; Part 2: node input vector embeddings; Part 3: graph transformer based encoder; Part 4: representation fusion; Part 5: functional component. Depending on the target application task, the function component will generate different output. In the sampled subgraphs, it covers both the target node and the surrounding context nodes.)

$w(v_i, v_j) = 0$. Notations $\mathcal{X}$ and $\mathcal{Y}$ denote feature space and label space, respectively. In this paper, we can simply denote $\mathcal{X} = \mathbb{R}^{d_x}$ and $\mathcal{Y} = \mathbb{R}^{d_y}$ (d_x and d_y are the dimensions of raw feature vector and label vector). For node v_i, we can also simplify its feature and label vectors as $\mathbf{x}_i = x(v_i) \in \mathbb{R}^{d_x}$ and $\mathbf{y}_i = y(v_i) \in \mathbb{R}^{d_y}$. The GRAPH-BERT model pre-training doesn't require any label supervision information actually, but partial of the labels will be used for the fine-tuning application task on node classification to be introduced later.

Instead of working on the complete graph G, GRAPH-BERT will be trained with linkless subgraph batches sampled from the input graph instead. It will effectively enable the learning of GRAPH-BERT to parallelize (even though we will not study parallel computing of GRAPH-BERT in this paper) on extremely large-sized graphs that the existing graph neural networks cannot handle. Different approaches can be adopted here to sample the subgraphs [Zhang *et al.*, 2018] from the input graph. However, to control the randomness involved in the sampling process, in this paper, we propose to introduce the *top-k intimacy* sampling approach instead. Such a sampling algorithm works based on the graph intimacy matrix $\mathbf{S} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, where entry $\mathbf{S}(i,j)$ measures the intimacy score between nodes v_i and v_j.

There exists different metrics to measure the intimacy scores among the nodes within the graph, e.g., Jaccard's coefficienty [Jaccard, 1901], Adamic/Adar [Adamic and Adar, 2003], Katz [Katz, 1953]. In this paper, we define matrix $\mathbf{S}$ based on the pagerank algorithm, which can be denoted as

$$\mathbf{S} = \alpha \cdot \left(\mathbf{I} - (1-\alpha)\cdot \bar{\mathbf{A}}\right)^{-1}, \quad (1)$$

where factor $\alpha \in [0,1]$ (which is usually set as 0.15). Term $\bar{\mathbf{A}} = \mathbf{A}\mathbf{D}^{-1}$ denotes the colum-normalized adjacency matrix. In its representation, $\mathbf{A}$ is the adjacency matrix of the input graph, and $\mathbf{D}$ is its corresponding diagonal matrix with $\mathbf{D}(i,i) = \sum_j \mathbf{A}(i,j)$ on its diagonal.

Formally, for any target node $v_i \in \mathcal{V}$ in the input graph, based on the intimacy matrix $\mathbf{S}$, we can define its learning context as follows:

DEFINITION 1. *(Node Context): Given an input graph G and its intimacy matrix $\mathbf{S}$, for node v_i in the graph, we define its learning context as set $\Gamma(v_i) = \{v_j | v_j \in \mathcal{V} \setminus \{v_i\} \wedge \mathbf{S}(i,j) \geq \theta_i\}$. Here, the term θ_i defines the minimum intimacy score threshold for nodes to involve in v_i's context.*

Here, we may need to add a remark: for all the nodes in v_i' learning context $\Gamma(v_i)$, they can cover both local neighbor of v_i as well as the nodes which are far away. In this paper, we define the threshold θ_i as the k_{th} entry of sorted$(\mathbf{S}(i,:))$, i.e., $\Gamma(v_i)$ covers the *top-k* intimate nodes of v_i in graph G.

Based on the node context concept, we can also represent the set of sampled graph batches as set $\mathcal{G} = \{g_1, g_2, \cdots, g_{|\mathcal{V}|}\}$, and g_i denotes the subgraph sampled for v_i (as the target node). Formally, g_i can be represented as $g_i = (\mathcal{V}_i, \varnothing)$, where the node set $\mathcal{V}_i = \{v_i\} \cup \Gamma(v_i)$ covers both v_i and its context nodes and the link set is null. For large-sized input graphs, set $\mathcal{G}$ can further be decomposed into several mini-batches, i.e., $\mathcal{B} \in \mathcal{G}$, which will be fed to train the GRAPH-BERT model.

3.2 Node Input Vector Embeddings

Different from image and text data, where the pixels and words/chars have their inherent orders, nodes in graphs are orderless [Meng and Zhang, 2019]. The GRAPH-BERT model to be learned in this paper doesn't require any node orders of the input sampled subgraph actually. Meanwhile, to simplify the presentations, we still propose to serialize the input subgraph nodes into certain ordered list instead. Formally, for all the nodes $\mathcal{V}_i$ in the sampled linkless subgraph $g_i \in \mathcal{B}$, we can denote them as a node list $[v_i, v_{i,1}, \cdots, v_{i,k}]$, where $v_{i,j}$ will be placed ahead of $v_{i,m}$ if $\mathbf{S}(i,j) > \mathbf{S}(i,m), \forall v_{i,j}, v_{i,m} \in \mathcal{V}_i$. For the remaining of this subsection, we will follow the identical node orders as indicated in the above list by default for their input vector embeddings.

Raw Feature Vector Embedding

The input vector embeddings to be fed to the graph-transformer model actually cover four components: (1) raw feature vector embedding, (2) Weisfeiler-Lehman absolute role embedding, (3) intimacy based relative positional embedding, and (4) hop based relative distance embedding, respectively. Formally, for each node $v_j \in \mathcal{V}_i$ in the subgraph g_i, we can embed its raw feature vector into a shared feature space (of the same dimension d_h) with the remaining embedding vectors, which can be denoted as

$$\mathbf{e}_j^{(x)} = \text{Embed}\left(\mathbf{x}_j; \mathbf{W}_x, \mathbf{b}_x\right), \text{ and } \mathbf{e}_j^{(x)} \in \mathbb{R}^{d_h \times 1}. \quad (2)$$

Depending on the input raw features properties, different models can be used to define the Embed$(\cdot; \mathbf{W}_x, \mathbf{b}_x)$ function ($\mathbf{W}_x, \mathbf{b}_x$ denote the involved variables). For instance, CNN can be used if $\mathbf{x}_j$ denotes images; if $\mathbf{x}_j$ denotes texts,

LSTM/BERT can be applied; and simple fully connected layers can also be used for simple attribute inputs.

Weisfeiler-Lehman Absolute Role Embedding

The Weisfeiler-Lehman (WL) algorithm can label the nodes according to their structural roles in the graph data, where the nodes with the identical roles will be labeled with the same code (e.g., integer strings or node colors). Formally, for node $v_j \in \mathcal{V}_i$ in the sampled subgraph, we can denote its WL code as $\text{WL}(v_j) \in \mathbb{N}$, which can pre-computed based on the complete graph and is invariant for different sampled subgraphs. In this paper, we adopt the embedding approach proposed in [Vaswani *et al.*, 2017; Niepert *et al.*, 2016] and define the nodes WL absolute role embedding vector as

$$\begin{aligned}\mathbf{e}_j^{(r)} &= \text{Position-Embed}\left(\text{WL}(v_j)\right)\\ &= \left[sin\left(\frac{\text{WL}(v_j)}{10000^{\frac{2l}{d_h}}}\right), cos\left(\frac{\text{WL}(v_j)}{10000^{\frac{2l+1}{d_h}}}\right)\right]_{l=0}^{\lfloor \frac{d_h}{2} \rfloor},\end{aligned} \tag{3}$$

where $\mathbf{e}_j^{(r)} \in \mathbb{R}^{d_h \times 1}$. The entry index l iterates throughout all the entries in the above vector to compute the entry values with $sin(\cdot)$ and $cos(\cdot)$ functions for the input node based on its WL code.

Intimacy based Relative Positional Embedding

The WL based role embeddings can capture the global node role information in the representations. Here, we will introduce a relative positional embedding to extract the local information in the subgraph based on the placement orders of the serialized node list introduced at the beginning of this subsection. Formally, based on that serialized node list, we can denote the position of $v_j \in \mathcal{V}_i$ as $P(v_j)$. We know that $P(v_i) = 0$ by default and nodes closer to v_i will have a small positional index. Furthermore, $P(\cdot)$ is a variant position index metric. For the identical node v_j, its positional index $P(v_j)$ will also be different for different sampled subgraphs.

Formally, for node v_j, we can also extract its intimacy based relative positional embedding with the Position-Embed$(\cdot)$ function defined above as follows:

$$\mathbf{e}_j^{(p)} = \text{Position-Embed}\left(\text{P}(v_j)\right) \in \mathbb{R}^{d_h \times 1}, \tag{4}$$

which is quite close to the positional embedding in [Vaswani *et al.*, 2017] for the relative positions in the word sequence.

Hop based Relative Distance Embedding

The hop based relative distance embedding can be treated as a balance between the absolute role embedding (for global information) and intimacy based relative positional embedding (for local information). Formally, for node $v_j \in \mathcal{V}_i$ in the subgraph g_i, we can denote its relative distance in hops to v_i in the original input graph as $H(v_j; v_i)$, which can be used to define its embedding vector as

$$\mathbf{e}_j^{(d)} = \text{Position-Embed}\left(\text{H}(v_j; v_i)\right) \in \mathbb{R}^{d_h \times 1}. \tag{5}$$

It it easy to observe that vector $\mathbf{e}_j^{(d)}$ will be variant for the identical node v_j in different subgraphs.

3.3 Graph Transformer based Encoder

Based on the computed embedding vectors defined above, we will be able to aggregate them together to define the initial input vectors for nodes, e.g., v_j, in the subgraph g_i as follows:

$$\mathbf{h}_j^{(0)} = \text{Aggregate}\left(\mathbf{e}_j^{(x)}, \mathbf{e}_j^{(r)}, \mathbf{e}_j^{(p)}, \mathbf{e}_j^{(d)}\right). \tag{6}$$

In this paper, we simply define the aggregation function as the vector summation. Furthermore, the initial input vectors for all the nodes in g_i can organized into a matrix $\mathbf{H}^{(0)} = [\mathbf{h}_i^{(0)}, \mathbf{h}_{i,1}^{(0)}, \cdots, \mathbf{h}_{i,k}^{(0)}]^\top \in \mathbb{R}^{(k+1)\times d_h}$. The graph-transformer based encoder to be introduced below will update the nodes' representations with $2K$ layers, and the output by the l_{th} layer can be denoted as

$$\begin{aligned}\mathbf{H}^{(l)} &= \text{G-Transformer}\left(\mathbf{H}^{(l-1)}\right)\\ &= \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_h}}\right)\mathbf{V} + \text{G-Res}\left(\mathbf{H}^{(l-1)}, \mathbf{X}_i, G\right),\end{aligned} \tag{7}$$

where

$$\begin{cases}\mathbf{Q} &= \mathbf{H}^{(l-1)}\mathbf{W}_Q^{(l)},\\ \mathbf{K} &= \mathbf{H}^{(l-1)}\mathbf{W}_K^{(l)},\\ \mathbf{V} &= \mathbf{H}^{(l-1)}\mathbf{W}_V^{(l)}.\end{cases} \tag{8}$$

In the above equations, $\mathbf{W}_Q^{(l)}, \mathbf{W}_K^{(l)}, \mathbf{W}_K^{(l)} \in \mathbb{R}^{d_h \times d_h}$ denote the involved variables. Notation G-Res $\left(\mathbf{H}^{(l-1)}, \mathbf{X}_i, G\right)$ represents the graph residual term introduced in [Zhang and Meng, 2019], and $\mathbf{X}_i \in \mathbb{R}^{(k+1)\times d_x}$ is the raw features of all nodes in the subgraph g_i. Also different from conventional residual learning, we will add the residual terms computed for the target node v_i to the hidden state vectors of all the nodes in the subgraph batch. Based on the graph-transformer function defined above, we can represent the representation learning process of GRAPH-BERT as the following equations:

$$\begin{cases}\mathbf{H}^{(0)} = [\mathbf{h}_i^{(0)}, \mathbf{h}_{i,1}^{(0)}, \cdots, \mathbf{h}_{i,k}^{(0)}]^\top,\\ \mathbf{H}^{(l)} = \text{G-Transformer}\left(\mathbf{H}^{(l-1)}\right), \forall l \in \{1, 2, \cdots, 2K\},\\ \mathbf{z}_i \quad = \text{Fusion}\left(\mathbf{H}^{(2K)}\right).\end{cases} \tag{9}$$

Different from the application of conventional transformer model on NLP problems, which aims at learning the representations of all the input tokens. In this paper, we aim to apply the graph-transformer to get the representations of the target node only. In the above equation, function Fusion $(\cdot)$ will average the representations of all the nodes in input list, which defines the state of the target v_i, i.e., $\mathbf{z}_i \in \mathbb{R}^{d_h \times 1}$. Both vector $\mathbf{z}_i$ and matrix $\mathbf{H}^{(2K)}$ will be outputted to the following functional component attached to GRAPH-BERT. Depending on the application tasks, the functional component and output will be different. We will show more detailed representations for the output in the following section on GRAPH-BERT pre-training and fine-tuning.

4 GRAPH-BERT Learning

In this section, we will introduce the pre-training and fine-tuning of the GRAPH-BERT for the graph data representation learning. We propose to pre-train GRAPH-BERT with two tasks: (1) node attribute reconstruction, and (2) graph structure recovery. Meanwhile, depending on the objective application tasks, e.g., (1) node classification and (2) graph clustering as studied in this paper, GRAPH-BERT can be further fine-tuned to adapt both the model and the learned node representations accordingly to the new tasks.

4.1 Pre-training

The node raw attribute reconstruction task focuses on capturing the node attribute information in the learned representations, whereas the graph structure recovery task focuses more on the graph connection information instead.

Task #1: Node Raw Attribute Reconstruction

Formally, for the target node v_i in the sampled subgraph g_i, we have its learned representation by GRAPH-BERT to be $\mathbf{z}_i$. Via the fully connected layer (together with the activation function layer if necessary), we can denote the reconstructed raw attributes for node v_i based on $\mathbf{z}_i$ as $\hat{\mathbf{x}}_i = \text{FC}(\mathbf{z}_i; \mathbf{W}_{fc}, \mathbf{b}_{fc})$. To ensure the learned representations can capture the node raw attribute information, compared against the node raw features, e.g., $\mathbf{x}_i$ for v_i, we can define the node raw attribute reconstruction based loss term as follows:

$$\ell_1 = \sum_{g_i \in \mathcal{B}} \|\mathbf{m}_i \otimes (\mathbf{x}_i - \hat{\mathbf{x}}_i)\|_2 . \quad (10)$$

Considering that the nodes input raw attributes can be very sparse, to avoid trivial solutions (e.g, $\mathbf{0}$) for $\hat{\mathbf{x}}_i$) and maintain the non-zero attributes, in the above loss equation, we introduce the mask matrix $\mathbf{m}_i \in \{1, \beta\}^{d_x \times 1}$, where entry $\mathbf{M}_i(m) = \beta$ iff $\mathbf{x}_i(m) \neq 0$ ($\beta > 1$).

Task #2: Graph Structure Recovery

Meanwhile, for all the nodes v_i in the complete input graph, we can denote their ultimately learned representations as set $\{\mathbf{z}_i\}_{v_i \in \mathcal{V}}$. To ensure such representation vectors can also capture the graph structure information, the graph structure recovery task is also used as a pre-training task. Formally, for any two nodes v_i and v_j, based on their learned representations, we can denote the inferred connection label vector between them as $\hat{\mathbf{y}}_{i,j} = \sigma(\text{FC}(\mathbf{z}_i \sqcup \mathbf{z}_j; \mathbf{W}_{fc}, \mathbf{b}_{fc}))$, where operator $\sqcup$ denotes the concatenation of vectors and $\hat{\mathbf{y}}_{i,j} \in \mathbb{R}^{2\times 1}$. Compared against the ground truth label vector $\mathbf{y}_{i,j}(m)$, we can define the introduced loss function as follows:

$$\ell_2 = \sum_{v_i, v_j \in \mathcal{V} \wedge v_i \neq v_j} \sum_{m=1}^{2} -\mathbf{y}_{i,j}(m) \log \hat{\mathbf{y}}_{i,j}(m), \quad (11)$$

where $\mathbf{y}_{i,j} = [1, 0]$ if $(v_i, v_j) \in \mathcal{E}$, otherwise $\mathbf{y}_{i,j} = [0, 1]$. Considering that the graph we have is usually very sparse, i.e., there exist far more unconnected node pairs than the connected ones, we will sample a subset of the unconnected node pairs to maintain a class balanced learning setting. By integrating the above two different learning tasks together with variable regularization terms, we will be able to define the pre-training objective function, minimization of which will help pre-train GRAPH-BERT effectively.

4.2 Model Transfer and Fine-tuning

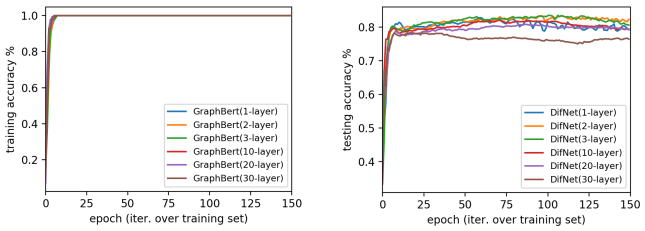
In applying the learned GRAPH-BERT into new learning tasks, the learned graph representations can be either fed into the new tasks directly or with necessary adjustment, i.e., fine-tuning. In this part, we can take the *node classification* and *graph clustering* tasks as the examples, where *graph clustering* can use the learned representations directly but fine-tuning will be necessary for the *node classification* task.

Task # 1: Node Classification

Based on the nodes learned representations, e.g., $\mathbf{z}_i$ for v_i, we can denote the inferred label for the node via the functional component as $\hat{\mathbf{y}}_i = \text{softmax}(\text{FC}(\mathbf{z}_i; \mathbf{W}_{fc}, \mathbf{b}_{fc}))$. Compared against the node true labels $\mathbf{y}_i$, we will be able to define the introduced node classification loss term as

$$\ell_{nc} = \sum_{v_i \in \mathcal{V}} \sum_{m=1}^{d_y} -\mathbf{y}_i(m) \log \hat{\mathbf{y}}_i(m). \quad (12)$$

By re-training these stacked fully connected layers together with GRAPH-BERT (whose variables are initialized with the values we learned via the pre-training step), we will be able to use the learned representations to infer node class labels.



(a) Training Accuracy (b) Testing Accuracy

Figure 2: The learning performance of GRAPH-BERT with 1-layer, ..., 3-layer, and 10-layer, ⋯, 30-layer on the Cora dataset. The x axis denotes the iterations over the whole training set. The y axes denote the training and testing accuracy, respectively.

Task # 2: Graph Clustering

Meanwhile, for the graph clustering task, the main objective is to partition nodes in the graph into several different clusters, e.g., $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_l\}$ (l is a hyper-parameter pre-specified in advance). For each objective cluster, e.g., $\mathcal{C}_j \in \mathcal{C}$, we can denote its center as a variable vector $\boldsymbol{\mu}_j = \sum_{v_i \in \mathcal{C}_j} \mathbf{z}_i \in \mathbb{R}^{d_h}$. For the graph clustering tasks, the main objective is to group similar nodes into the same community, whereas the different nodes will be partitioned into different clusters instead. Therefore, the objective function of graph clustering can be defined as follows:

$$\min_{\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_l} \min_{\mathcal{C}} \sum_{j=1}^{l} \sum_{v_i \in \mathcal{C}_j} \|\mathbf{z}_i - \boldsymbol{\mu}_j\|_2 . \quad (13)$$

The above objective function involves multiple variables to learn concurrently, which can be trained with the EM algorithm much more effectively instead of error backpropagation. Therefore, instead of re-training the above graph clustering model together with GRAPH-BERT, we will only take the learned node representations as the node feature input for training the graph clustering model instead.

5 Experiments

To test the effectiveness of GRAPH-BERT in learning the graph representations, in this section, we will provide extensive experimental results of GRAPH-BERT on three real-world benchmark graph datasets, i.e., Cora, Citeseer and Pubmed, respectively.

We are still finalizing and tuning the GRAPH-BERT model at present. We have the node classification results of GRAPH-BERT on Cora and Pubmed ready, but encounter some problems on Citeseer. So, we didn't show the results in the previous version submitted to arXiv. We received many emails and messages from people who are interested in this work, so we are happy to release the source code together with the current results we obtained with the community. As we are polishing GRAPH-BERT, the architecture and the source code of GRAPH-BERT can still be subject to changes in the follow-up versions.

5.1 Dataset and Learning Settings

The graph benchmark datasets used in the experiments include Cora, Citeseer and Pubmed [Yang *et al.*, 2016], which are used in most of the recent state-of-the-art graph neural network research works [Kipf and Welling, 2016; Veličković *et al.*, 2018; Zhang and Meng, 2019]. Based on the input graph data, we will first compute the node intimacy scores, based on which subgraph batches will be sampled subject to the subgraph size $k \in \{1, 2, \cdots, 10, 15, 20, \cdots, 50\}$. In addition, we will also compute the node pairwise hop distance and WL node roles. By minimizing the node raw feature reconstruction loss and graph structure recovery loss, GRAPH-BERT can be effectively pre-trained, whose learned variables will be transferred to the follow-up node classification and graph clustering tasks with/without fine-tuning.

Reproducibility. Both the datasets and source code used can be accessed via link[3]. Detailed information about the server used to run the model can be found at the footnote[4].

Default Parameter Settings. The results reported in this paper are based on the following parameter settings of GRAPH-BERT: *subgraph size*: k=7 (Cora) and k=25 (Pubmed); *hidden size*: 32; *attention head number*: 2; *hidden layer number*: 2; *learning rate*: 0.01 (Cora) and 0.001 (Citeseer and Pubmed); *weight decay*: $5e^{-4}$; *intermediate size*: 32; *hidden dropout rate*: 0.5; *attention dropout rate*: 0.3; *graph residual term*: graph-raw.

5.2 Node Classification Experimental Results

Based on the learned variables of GRAPH-BERT, we further load them into a new GRAPH-BERT based model for node classification. As introduced in Section 4.2, we will stack several fully connected layers (and a softmax layer) on the output layer of GRAPH-BERT to infer the multi-class labels of the nodes. Here, we will follows the identical train/validation/test set partitions used in the existing graph neural network papers [Yang *et al.*, 2016] for fair comparisons. For the new model, we further fine-tune the new model variables with less than 10 epochs with the training data (mainly about the fully connected layer based functional component), which will be used to infer the labels of nodes in the testing set.

Learning Convergence of Deep GRAPH-BERT

In Figure 2, we illustrate the training records of GRAPH-BERT for node classification on the Cora dataset, where the depth of GRAPH-BERT varies with value in $\{1, 2, 3, 10, 20, 30\}$. According to the plots, GRAPH-BERT can converge very fast (with less than 10 epochs) on the training set. What's more, as the model depth increases, GRAPH-BERT will not suffer from the suspended animation problem. Even the very deep GRAPH-BERT (30 layers) can still respond effectively to the training data and achieve good learning performance.

[3]https://github.com/jwzhanggy/Graph-Bert

[4]GPU Server: ASUS X99-E WS motherboard, Intel Core i7 CPU 6850K@3.6GHz (6 cores), 3 Nvidia GeForce GTX 1080 Ti GPU (11 GB buffer each), 128 GB DDR4 memory and 128 GB SSD swap. For the deep models which cannot fit in the GPU memory, we run them with CPU instead.

Table 1: Learning Performance of GRAPH-BERT Compared Against Existing Baseline Methods on Node Classification. The results of GRAPH-BERT reported here denotes the best observed scores obtained with $k \in \{1, 2, \cdots, 10, 15, 20, \cdots, 50\}$. For the GRESNET method, it corresponds to GRESNET(GCN, raw) with GCN as the base mode and raw residual term.

Methods	Datasets (Accuracy)		
	Cora	**Citeseer**	**Pubmed**
LP ([Zhu *et al.*, 2003])	0.680	0.453	0.630
ICA ([Lu and Getoor, 2003])	0.751	0.691	0.739
ManiReg ([Belkin *et al.*, 2006])	0.595	0.601	0.707
SemiEmb ([Weston *et al.*, 2008])	0.590	0.596	0.711
DeepWalk ([Perozzi *et al.*, 2014b])	0.672	0.432	0.653
Planetoid ([Yang *et al.*, 2016])	0.757	0.647	0.772
MoNet ([Monti *et al.*, 2016])	0.817	-	0.788
GCN ([Kipf and Welling, 2016])	0.815	0.703	0.790
GAT ([Veličković *et al.*, 2018])	0.830	0.725	0.790
LOOPYNET ([Zhang, 2018])	0.826	0.716	0.792
GRAPH-BERT	**0.843**	**-**	**0.793**

Table 2: Analysis of parameter k on Cora for model performance (testing accuracy and testing loss) and total time cost.

k	**Cora Dataset**		
	Test Accuracy	**Test Loss**	**Total Time Cost** (s)
1	0.804	0.791	**30.81**
2	0.806	0.708	41.21
3	0.819	0.663	45.93
4	0.818	0.690	50.97
5	0.824	0.636	56.83
6	0.834	0.625	65.34
7	**0.843**	**0.620**	70.15
8	0.828	0.653	76.49
9	0.814	0.679	84.17
10	0.819	0.653	92.60
20	0.819	0.666	163.20
30	0.801	0.710	227.52
40	0.768	0.805	288.61
50	0.759	0.833	382.54

Main Results

The learning results of GRAPH-BERT on node classification is provided in Table 1. According to the achieved scores, we observe that GRAPH-BERT can achieve much better performance than these state-of-the-art graph neural network baseline approaches with a big improvement.

Parameter Analysis of Subgraph Size k

As illustrated in Table 2, we provide the learning results of GRAPH-BERT with different subgraph sizes, i.e., parameter k. According to the results, parameter k affects the learning performance of GRAPH-BERT a lot, since it defines how many of nearby nodes will be used to define the nodes contexts. For the Cora dataset, we observe that the learning performance of GRAPH-BERT improves steadily as k increases from 1 to 7. After that, as k further increases, the performance of GRAPH-BERT will degrade dramatically. For the good scores with k = 1, partial contributions come from the graph residual terms involved in GRAPH-BERT. The time cost of GRAPH-BERT increases as k goes larger, which is minor actually compared with existing graph neural network models like GAT. Similar results can be observed for the other two

Table 3: Learning performance of GRAPH-BERT with different initial embedding inputs (To show the performance difference, we didn't add any graph residual terms here).

Methods		Datasets (Accuracy & Model Depth)		
Models	Embedding	**Cora**	**Citeseer**	**Pubmed**
GRAPH-BERT	raw feature	0.795	-	0.780
	wl role	0.457	-	0.443
	position	0.323	-	0.395
	hop distance	0.307	-	0.445
	all	**0.804**	-	**0.786**

Table 4: Best performance of GRAPH-BERT with different graph residual terms.

Methods		Datasets (Accuracy & Model Depth)		
Base Models	Residuals	**Cora**	**Citeseer**	**Pubmed**
GRAPH-BERT	none	0.804	-	0.786
	raw	0.817	**-**	0.786
	graph-raw	**0.843**	-	**0.793**

datasets.

Node Information Embeddings

As shown in Table 3, we provide the learning performance of GRAPH-BERT on the three datasets, which takes different initial embeddings as the input. According to the results, using the *Weisfeiler-Lehman absolute role embedding*, *intimacy based relative positional embedding*, and *hop based relative distance embedding* vectors along, GRAPH-BERT cannot work well actually, which indicates that the raw feature input coveys very important information. Meanwhile, by incorporating such complementary embeddings into the raw feature embedding, the model can achieve better performance than using the raw feature embedding along.

Graph Residual Terms

In Table 4, we also provide the learning results of GRAPH-BERT with different graph residual terms. According to the scores, GRAPH-BERT with graph-raw residual term can outperform the other two, which is also consistent with the experimental observations on these different residual terms as reported in [Zhang and Meng, 2019].

6 Conclusion and Future Works

In this paper, we have introduced the new GRAPH-BERT model for graph representation learning. Different from existing GNNs, GRAPH-BERT doesn't rely on the approximated graph convolutional operator and is learnable without any supervision information. Therefore, GRAPH-BERT works well in deep architectures and will not suffer from the common problems with other GNNs. Based on a batch of linkless subgraphs sampled from the original graph data, GRAPH-BERT can effectively learn the representations of the target node with the extended graph-transformer layers introduced in this paper. GRAPH-BERT can serve as the graph representation learning component in graph learning pipeline. The pre-trained GRAPH-BERT can be transferred and applied to address new tasks either directly or with necessary fine-tuning.

Potential Future Works: (1) Node-wise parameter k: In this paper, we set a common k for all the nodes in subgraph batching. However, in the real-world, such a setting is not realist, and different nodes can have a local context formed by various sized subsets of nearby neighbors. According the experimental tests, parameter k has a big impact on the learning performance of GRAPH-BERT, and an automatic selection of node-wise parameter k is an important direction. (2) Subgraph batching: In this paper, to avoid introducing randomness, we propose to sample the fixed context (determined by the node intimacy scores) for all the nodes. New subgraph batching methods can be explored to achieve the subgraph samples for learning GRAPH-BERT. (3) Subgraph initial embeddings: In this paper, we introduce four initial embeddings for the subgraphs (i.e., raw feature, WL role, position and hop based distance) and propose to aggregate them together by adding them together. Both new initial subgraph embedding and new aggregation methods will be explored to further improve GRAPH-BERT. (4) Target node representation: Based on the outputted representations, we propose to compute the target node representation by computing the average of all the nodes in subgraphs in this paper. This component may also require more explorations to try some new approaches, e.g., attention based aggregation or pooling based methods. (5) Parallel and Distributed Learning: Based on the sampled subgraph batches, the node connections have been broken down already, the learning process of GRAPH-BERT can be easily parallelized or can be done with distributed computing platforms. Due to the limited computational facilities, we cannot test that part in this paper, and we will try to further explore the distributed GRAPH-BERT as our future work. (6) Model Transfer across Data: In this paper, we transfer and apply the pre-trained GRAPH-BERT to address other tasks on the same dataset. Meanwhile, pre-trained model transfer between different datasets may still require more explorations.

References

[Adamic and Adar, 2003] Eytan Adamic and Lada A. Adar. Friends and neighbors on the web. (3):211–230, July 2003.

[Belkin *et al.*, 2006] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *J. Mach. Learn. Res.*, 7:2399?2434, December 2006.

[Chung *et al.*, 2014] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[Dai *et al.*, 2019] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[Gürel *et al.*, 2019] Nezihe Merve Gürel, Hansheng Ren, Yujing Wang, Hui Xue, Yaming Yang, and Ce Zhang. An anatomy of graph neural networks going deep via the lens of mutual information: Exponential decay vs. full preservation. *ArXiv*, abs/1910.04499, 2019.

[Hamilton *et al.*, 2017] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[Hammond *et al.*, 2011] David K. Hammond, Pierre Vandergheynst, and Remi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129?150, Mar 2011.

[He *et al.*, 2018] Kaiming He, Ross B. Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *CoRR*, abs/1811.08883, 2018.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8), November 1997.

[Huang and Carley, 2019] Binxuan Huang and Kathleen M. Carley. Inductive graph representation learning with recurrent graph neural networks. *CoRR*, abs/1904.08035, 2019.

[Jaccard, 1901] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[Jin *et al.*, 2018] Wengong Jin, Regina Barzilay, and Tommi S. Jaakkola. Junction tree variational autoencoder for molecular graph generation. *CoRR*, abs/1802.04364, 2018.

[Katz, 1953] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, Mar 1953.

[Kim, 2014] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[Kipf and Welling, 2016] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[Klicpera *et al.*, 2018] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Personalized embedding propagation: Combining neural networks on graphs with personalized pagerank. *CoRR*, abs/1810.05997, 2018.

[Lan *et al.*, 2019] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019.

[Li *et al.*, 2018] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *CoRR*, abs/1801.07606, 2018.

[Lu and Getoor, 2003] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 496?503. AAAI Press, 2003.

[Meng and Zhang, 2019] Lin Meng and Jiawei Zhang. Isonn: Isomorphic neural network for graph representation learning and classification. *CoRR*, abs/1907.09495, 2019.

[Monti *et al.*, 2016] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *CoRR*, abs/1611.08402, 2016.

[Niepert *et al.*, 2016] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. *CoRR*, abs/1605.05273, 2016.

[Perozzi *et al.*, 2014a] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710, New York, NY, USA, 2014. ACM.

[Perozzi *et al.*, 2014b] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014.

[Shang *et al.*, 2019] Junyuan Shang, Tengfei Ma, Cao Xiao, and Jimeng Sun. Pre-training of graph augmented transformers for medication recommendation. *CoRR*, abs/1906.00346, 2019.

[Sun *et al.*, 2019] Ke Sun, Zhouchen Lin, and Zhanxing Zhu. Adagcn: Adaboosting graph convolutional networks into deep models, 2019.

[Ugander *et al.*, 2011] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[Weston *et al.*, 2008] Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning*, ICML'08, page 1168?1175, New York, NY, USA, 2008. Association for Computing Machinery.

[Yang *et al.*, 2016] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *CoRR*, abs/1603.08861, 2016.

[Zhang and Meng, 2019] Jiawei Zhang and Lin Meng. Gresnet: Graph residual network for reviving deep gnns from suspended animation. *ArXiv*, abs/1909.05729, 2019.

[Zhang *et al.*, 2018] Jiawei Zhang, Limeng Cui, and Fisher B. Gouza. SEGEN: sample-ensemble genetic evolutional network model. *CoRR*, abs/1803.08631, 2018.

[Zhang, 2018] Jiawei Zhang. Deep loopy neural network model for graph structured data representation learning. *CoRR*, abs/1805.07504, 2018.

[Zhu *et al.*, 2003] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 912?919. AAAI Press, 2003.