Learning Adaptive Node Embeddings across Graphs

Gaoyang Guo, Chaokun Wang, *Member, IEEE*, Bencheng Yan, Yunkai Lou, Hao Feng, Junchao Zhu, Jun Chen, Fei He, and Philip S. Yu, *Fellow, IEEE*

Abstract—Recently, learning embeddings of nodes in graphs has attracted increasing research attention. There are two main kinds of graph embedding methods, i.e., transductive embedding methods and inductive embedding methods. The former focuses on directly optimizing the embedding vectors, and the latter tries to learn a mapping function for the given nodes and features. However, little work has focused on applying the learned model from one graph to another, which is a pervasive idea in Computer Vision or Natural Language Processing. Although some of the graph neural networks (GNNs) present a similar motivation, none of them considers both the structure bias and the feature bias between graphs. In this paper, we present a novel graph embedding problem called Adaptive Task (AT), and propose a unified framework for the adaptive task, which introduces two types of alignment to learn adaptive node embeddings across graphs. Then, based on the proposed framework, a novel Graph Adaptive Embedding network (GraphAE) is designed to address the adaptive task. Furthermore, we extend GraphAE to a multi-graph version to consider a more complex adaptive situation. The extensive experimental results demonstrate that our model significantly outperforms the state-of-the-art methods, and also show that our framework can make a great improvement over a number of existing GNNs.

Index Terms—Graph Embedding, Adaptive Task, Graph Bias

1 Introduction

Graph (a.k.a. network) data analysis (abbr. GDA) has been receiving more and more attention because of the great expressive power of graphs. Recently, there has been a surge of interest in the problem of graph embedding (GE), which is a new direction of GDA with machine learning methods [1], [2], [3], [4], [5]. The goal of GE is to extract high-level features of nodes as well as their neighbors and embed nodes as points in a low-dimensional vector space. Then, the relationship among nodes in the learned space can reflect the structure relationship in the original graph. Meanwhile, it has been proved that the resultant vectors, i.e., the embedding vectors of nodes in graphs, are extremely useful for many applications, such as node classification [4], [5], [6].

Most existing GE work can be classified into two categories: the transductive embedding and the inductive embedding. (1) **Transductive Embedding** (TE). TE methods need to access the information of test data during training. For example, DeepWalk [4] and Node2vec [6] (always followed by a classifier in supervised tasks) adopt a random walk strategy. They optimize the embedding vectors by modeling the co-occurrence probability of a node pair in the same path. In this way, they can capture the similarity between nodes. (2) **Inductive Embedding** (IE). IE methods aim to learn a mapping function and do not need to access the information of test data during training. It means that, given the attributes and neighborhoods of a node v, the learned function can infer the embedding vector of v no matter whether v is present in the training set or not. For example, Graph Neural Network (GNN) approaches (e.g., GraphSAGE [3] and GAT [1]) adopt the message propagation to reveal the relationship among nodes, and learn the embedding patterns. The general idea of GNNs is that each node iteratively aggregates the features of its neighbors and combines them with its own features. Theoretically, a k-layer stacked (k iterations) GNN model can capture the structure and attribute information within the node's k-hop structure. In this way, GNN can be taken as a well-learned mapping function to infer node embeddings.

However, in real-world applications, graphs coming from different times or sources are ubiquitous, and these graphs may have different distributions. Clearly, it is a general embedding situation, where we always want to learn some knowledge from known graphs (also called source graphs) and use the knowledge to deal with new graphs (also called target graphs) that have different graph distributions.

Concretely, the distributions to characterize a graph usually consist of the structure distribution and the feature distribution. (1) The **structure distribution** of a graph is a probability distribution that captures the inherent properties of the topology of the graph. Generally speaking, there are mainly two categories of structure distributions in the existing literature: the degree distribution [7], [8], and the subgraph distribution [9], [10]. Specifically, the degree distribution is used as the structure distribution of the graph in our paper since the degree distribution is easily understood and has been applied in a number of research studies as a typical structure distribution of graphs [11], [12]. For ex-

- *Gaoyang Guo, Chaokun Wang, Bencheng Yan, Yunkai Lou, Hao Feng, Junchao Zhu, and Fei He are with the School of Software, Tsinghua University, Beijing 100084, P. R. China. E-mail: {ggy16, ybc17, lyk18, fh20, zhu-jc17}@mails.tsinghua.edu.cn, {chaokun, hefei}@tsinghua.edu.cn.*
- *Jun Chen is from Baidu Inc. E-mail: chenjun22@baidu.com.*
- *Philip S. Yu is now with Department of Computer Science, University of Illinois at Chicago, IL 60607, USA. Email: psyu@uic.edu.*

(Corresponding author: Chaokun Wang.)

ample, degree distributions in real networks often conform to power-law distributions [7]. (2) The **feature distribution** of a graph is a probability distribution that captures the inherent properties of the semantics of the graph [13], [14]. Specifically, each node in a graph can be represented by a semantic feature vector, which can be composed of either hand-crafted features (such as the item attributes in an e-commerce network) or machine-generated features (such as the features generated by SVD decomposition in a citation network). The feature distribution just represents the probability distribution that each node's semantic feature vector follows.

Thus, graph bias naturally exists between source graphs and target graphs with different graph distributions. Correspondingly, graph bias consists of structure bias and feature bias. (1) **Structure bias** refers to the discrepancy of structure distributions between source graphs and target graphs. (2) **Feature bias** refers to the discrepancy of feature distributions between source graphs and target graphs.

Please note that in the fields of computer vision and natural language processing, image distribution/text distribution only refers to the feature distributions of image/text data (e.g., pixel value distribution for image data and bag-of-words distribution for text data), where there is no connection relationship among image/text data. Different with image/text data, nodes are connected via edges in graph data. Therefore, graph distribution additionally considers connection relationship between nodes (i.e., structure distribution) besides feature distribution. Thus, data bias for image/text data only refers to feature bias, while graph bias refers to both feature bias and structure bias.

We study the aforementioned embedding situation where we need to learn embeddings of source graphs and use the learned knowledge to infer embeddings of target graphs when the graph bias exists. For example, considering e-commerce item-item networks, user behaviors often vary a lot from a normal day to a shopping carnival. Making use of the network constructed in a normal day (i.e., source graph) to help predict or analyze the user behaviors in the carnival (i.e., target graph) is urgently needed for e-commerce companies [15]. Another example is the citation network in data mining conferences. Hot topics in the field of data mining are changing over time (e.g., SVM for the past and Deep Learning for now). The learned knowledge from a network constructed by the papers published in the early twenty-first century (i.e., source graph) can be used to analyze the network constructed by the papers published recently (i.e., target graph). The direct way to solve the above problem is fine-tuning on the target graph. However, a significant amount of labeled data from the target graph is needed for fine-tuning, which is not easy to be obtained for many applications [16]. Hence, adaption learning is required to reduce the graph bias between source and target graphs.

In this paper, we propose a novel problem called **A**daptive **T**ask (AT) (See Section 3 for its definition), and present an **A**daptive **E**mbedding (AE) method to address the adaptive task.

Example 1. As shown in Figure 1(a), Graph A and Graph B represent two e-commerce networks constructed from a normal day and a shopping carnival, respectively, where each node refers to one item, the number in a node denotes the unique item id, and an edge between two items means the two items are viewed by the same user within 30 minutes. Obviously, Graph A and Graph B have graph bias, i.e., there exist both structure bias (B is denser than A since it is from a shopping carnival) and feature bias (A and B have different sets of items with different features) between graph A and graph B. The nodes in Graph A are labeled with their item categories, i.e., rectangle for "Electronics" and circle for "Non-Electronics". The nodes in Graph B are unlabeled and need to be predicted. The goal of AT is to utilize the labeled Graph A to infer the embeddings of items in

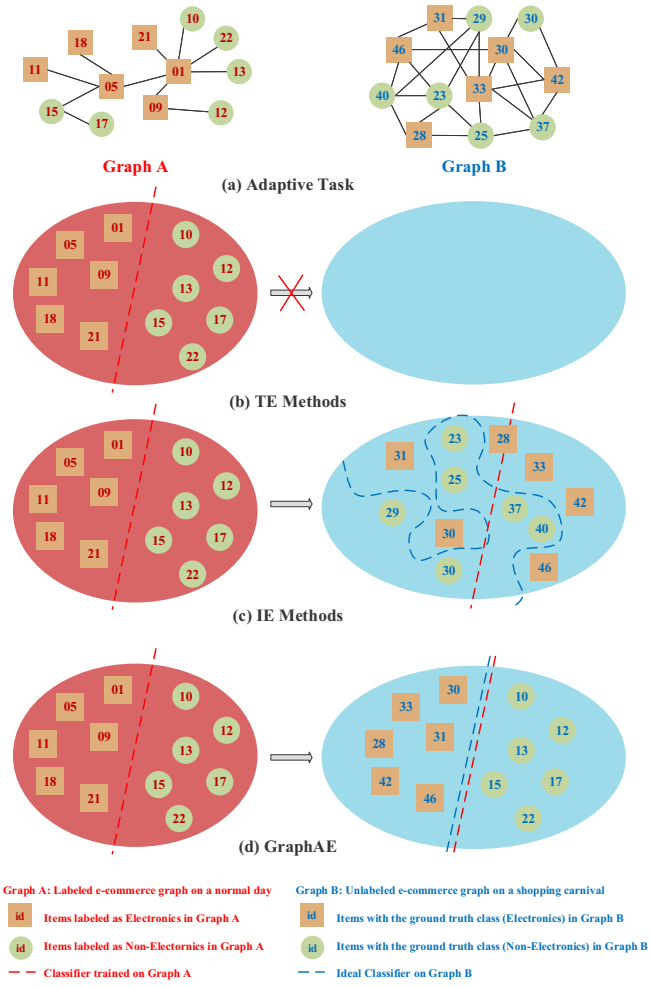


Fig. 1. Examples of different methods applied to the adaptive task. (a) The knowledge learned from labeled e-commerce Graph A is utilized to infer the items' embeddings and predict the items' categories in unlabeled Graph B. The items' embeddings learned by different methods are plotted into a 2-D space. (b) TE methods cannot infer the item's embeddings of Graph B, since these methods directly optimize the embedding vectors rather than learn patterns (i.e., mapping functions which can be used to infer new graphs) from Graph A. (c) The IE methods cannot infer satisfactory items' embeddings of Graph B since there exists graph bias between Graphs A and B. The classifier (the red dashed line) trained on labeled Graph A fails to give a satisfactory prediction on unlabeled Graph B compared with the ideal classifier (the blue dashed line). (d) Our approach (GraphAE) considers the graph bias and eliminates the discrepancy of graph distributions between Graphs A and B, so that the classifier trained on Graph A can readily predict items' categories in Graph B.

Graph B and predict their categories, even if these two graphs have graph bias.

Example 2. As shown in Figure 1(b), the learned embeddings of items in Graph A by TE methods are plotted into a 2-D space, and can be well used to classify the labels of items, i.e., "Electronics" or "Non-Electronics". But there is no access to infer the embeddings of items in Graph B and predict their categories.

Example 2 shows that for TE methods, the node embeddings in the source graph are directly optimized. Therefore, there is no way to infer the node embeddings and predict their labels in the target graph.

Example 3. As shown in Figure 1(c), IE methods can be trained on Graph A and infer the items' embeddings of Graph B. However, due to the graph bias between Graphs A and B, the inferred items' embeddings of Graph B are confused. Specifically, on the one hand, inconsistent embedding distributions between Graphs A and B indicate the patterns learned from Graph A are not well used to infer Graph B. On the other hand, the classifier (the red dashed line) trained on labeled Graph A fails to give a satisfactory prediction for unlabeled Graph B compared with the ideal classifier (the blue dashed line), which leads to poor classification accuracy.

Example 3 shows that, though IE methods (such as GraphSAGE [3] and GAT [1]) can infer the node embeddings in target graphs by mapping functions, they cannot learn adaptive embeddings when the graph bias exists.

Hence, both TE and IE methods cannot well solve the AE task considering the following two major challenges. (1) **Graph Bias**. The discrepancy of distributions between source graphs and target graphs is ubiquitous in practice. Such graph bias leads to a more difficult situation for graph embedding, where the patterns learned from source graphs may not be suitable to be used in the target graph. (2) **Limited Information**. Limited label information from target graphs inhibits the model from inferring the node embeddings of target graphs. Therefore, a more careful design is needed to utilize the label information from known graphs.

Present work. In this paper, we propose a unified framework for the adaptive task, where most existing GNNs can be taken as special cases of the proposed framework. Then, based on our framework, we propose a **Graph A**daptive **E**mbedding (GraphAE) network to address the adaptive task. As shown in Figure 1(d), GraphAE can eliminate the discrepancy between the embedding distribution of the source graph (i.e., Graph A) and that of the target graph (i.e., Graph B). Then, the learned patterns from Graph A can be safely applied in Graph B (i.e., the red dash) to predict the labels of items in Graph B. Furthermore, we try to learn knowledge from multiple source graphs to help analyze the target graph, and propose a multi-graph version of GraphAE called MGraphAE. Besides, due to the flexibility of our framework, we successfully enhance the performance of existing GNNs with the help of the alignment introduced in our framework.

The main contributions of this paper are summarized as follows.

- A new graph embedding task (i.e., adaptive task) is presented, which is a more general situation in the graph embedding. Moreover, a unified framework to address the adaptive task is proposed.
- Based on our framework, a novel Graph Adaptive Embedding (GraphAE) network is proposed to reduce the graph bias and learn adaptive node embeddings across graphs.
- A multi-graph version of GraphAE called MGraphAE is further proposed. By utilizing the knowledge learned from multiple source graphs, MGraphAE can improve the performance of GraphAE in most cases.
- Experimental results demonstrate that our model outperforms the state-of-the-art methods, and show that existing GNN models can be enhanced by our framework (resulting in an average 5.63% gain in accuracy on the adaptive tasks).

2 Related Work

2.1 Transductive Embedding Methods

Transductive embedding methods focus on the transductive task, i.e., requiring that all nodes in the graph (including the test nodes) are present during training [4], [6], [17], [18], [19], [20]. DeepWalk [4] borrows the idea of the language model from natural language processing and generates truncated random walks. Then the model takes the generated random walks into the skip-gram model and optimizes the embedding vectors of nodes according to the relationship among nodes in one random walk. Node2vec [6] extends the idea of DeepWalk, and introduces two parameters to control the random walk, i.e., proposing a strategy that combines depth-first sampling (DFS) and breadth-first sampling (BFS). In this way, Node2vec can adaptively allow the random walk to go further, and capture more complex structures. Please note that these unsupervised node embedding algorithms (e.g., DeepWalk and Node2vec) are always followed by a classifier (e.g. a softmax classifier) in supervised tasks.

Although these transductive embedding methods can learn good embeddings for one known graph, the optimized model cannot infer embeddings for any new graph. Such property may be a bottleneck of these methods. Actually, a great number of users always want to learn patterns from known graphs, and then use the patterns to infer new graphs.

2.2 Inductive Embedding Methods

Inductive embedding methods focus on the inductive task, i.e., trying to learn a mapping function from known graphs [1], [2], [3], [21], [22], [23]. The learned function can not only give the node embeddings of known graphs, but also infer the node embeddings of new graphs. The most popular inductive embedding methods are graph neural networks (GNNs). GCN [2] is a pioneer work that naturally gathers the neighbors' information by the normalized graph Laplacians. GraphSAGE [3] extends the idea of GCN, and proposes several types of aggregation strategies. FastGCN [22] proposes a layer-wise node sampling strategy to train

the GCN model effectively. GAT [1] employs multi-head based self-attention to filter out important information in neighbors and integrates neighbors' information by different weights. GeniePath [24] further improves GAT by using the attention mechanism to choose the receptive fields of graph neural networks with the help of breadth and depth explorations.

Although these inductive embedding methods can infer the embeddings of nodes across graphs, none of them considers the data bias between graphs, including the structure bias and the feature bias. Ignoring such graph bias and directly inferring the embeddings of nodes in a new graph may introduce some noises and lead to unsatisfactory embeddings. In this paper, we propose a unified framework considering the graph bias to address the adaptive task.

2.3 Domain Adaptive Embedding Methods

There has been some work focusing on domain adaptive embedding methods. Lee et al. [25] and Qiu et al. [26] pre-train graph neural networks on known graphs and fine-tune them on new graphs. AdaGCN [27] designs a domain adaptation component to reduce the distribution discrepancy between different domains. UDA-GCN [28] utilizes an attention mechanism to combine the representations of a dual graph convolutional network that is followed by a domain classifier loss to make the representations domain-invariant.

Although these domain adaptive embedding methods consider the domain discrepancy between graphs, on the one hand, the GNN pre-training methods still need some label information of new graphs for fine-tuning, which do not apply to our task where target graphs have no label information. On the other hand, the existing domain adaptive GNNs only consider reducing the feature bias between graphs in the last layer of GNN, while our proposed method considers reducing both the structure bias and the feature bias between graphs in all layers of GNN.

3 Notations and Problem Formulation

Definition 1 (Structure Distribution). *Given a graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ where $\mathcal{V}$ denotes the node set and $\mathcal{E}\subseteq\mathcal{V}\times\mathcal{V}$ denotes the edge set, the structure distribution $\mathcal{P}_g$ of $\mathcal{G}$ is the probability distribution that a type of structural element $\mathbf{e}$ of $\mathcal{G}$ follows, i.e., $\mathbf{e}\sim\mathcal{P}_g(\mathbf{e})$, where the structural element $\mathbf{e}$ can be a degree value or a subgraph pattern. We have $\mathcal{P}_g(\mathbf{e})=\mathcal{N}_\mathbf{e}/|\mathcal{N}|$, where $\mathcal{N}_\mathbf{e}$ denotes the number of the occurrence of $\mathbf{e}$ in $\mathcal{G}$, and $\mathcal{N}$ denotes the number of all the structural elements in $\mathcal{G}$.*

Specifically, we use the degree value as the specific structural element of $\mathcal{G}$ in this paper. That is to say, the structure distribution $\mathcal{P}_g$ of $\mathcal{G}$ in this paper is the probability distribution from which the degree value $\mathbf{d}$ of each node $v\in\mathcal{V}$ is drawn, i.e., $\mathbf{d}\sim\mathcal{P}_g(\mathbf{d})$. We have $\mathcal{P}_g(\mathbf{d})=\mathcal{N}_\mathbf{d}/|\mathcal{N}|$, where $\mathcal{N}_\mathbf{d}$ denotes the number of nodes having $\mathbf{d}$ neighborhoods in $\mathcal{G}$, and $|\mathcal{N}|$ denotes the node number of $\mathcal{G}$.

Definition 2 (Feature Distribution). *Given a graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ where each node $v\in\mathcal{V}$ is associated with a feature vector $\mathbf{x}$, the feature distribution $\mathcal{P}_f$ is the probability distribution that $\mathbf{x}$ follows, i.e., $\mathbf{x}\sim\mathcal{P}_f(\mathbf{x})$.*

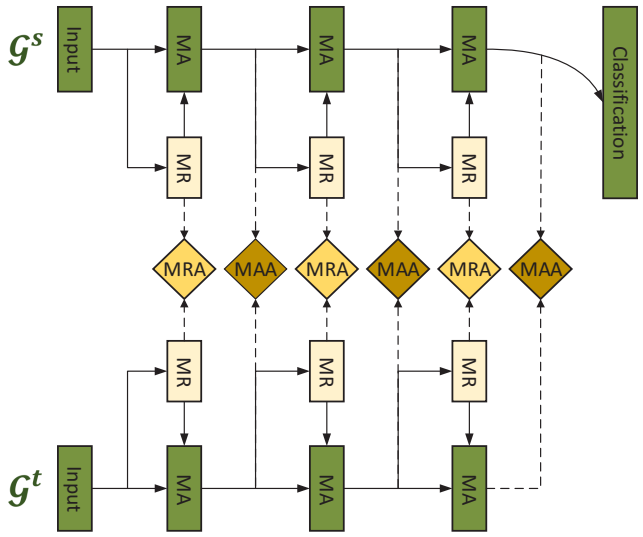


Fig. 2. The Unified Framework for Adaptive Task.

In this paper, we focus on a challenging task called Adaptive Task, which is defined in Definition 3.

Definition 3 (Adaptive Task). *Given a source graph $\mathcal{G}^s=(\mathcal{V}^s,\mathcal{E}^s)$ with its structure distribution $\mathcal{P}_g^s$ and its feature distribution $\mathcal{P}_f^s$, and a target graph $\mathcal{G}^t=(\mathcal{V}^t,\mathcal{E}^t)$ with its structure distribution $\mathcal{P}_g^t$ and its feature distribution $\mathcal{P}_f^t$, there is the graph bias between $\mathcal{G}^s$ and $\mathcal{G}^t$, i.e., $\mathcal{P}_g^s\neq\mathcal{P}_g^t$ and $\mathcal{P}_f^s\neq\mathcal{P}_f^t$. Meanwhile, the nodes in $\mathcal{G}^s$ are labeled and the nodes in $\mathcal{G}^t$ are unlabeled. The goal of the adaptive task is to infer the node embeddings of $\mathcal{G}^t$ and predict the node labels of $\mathcal{G}^t$ only using the supervision from $\mathcal{G}^s$.*

4 Unified Framework for Adaptive Task

In this section, we propose a unified framework for the adaptive task. As shown in Figure 2, there are two branches in our framework, i.e., one for the source graph $\mathcal{G}^s$ and the other for the target graph $\mathcal{G}^t$. Each branch is designed by the single graph embedding composed by Message Router (MR) and Message Aggregator (MA) (see Section 4.1). The parameters are shared in these two branches. To learn adaptive embeddings across graphs, two across-graph alignment strategies (Message Routing Alignment (MRA) and Message Aggregating Alignment (MAA)) are designed (see Section 4.2). Both the supervised information (introduced by the node labels of the source graph) and the unsupervised information (introduced by the two alignment strategies) are used to train the network.

4.1 Single Graph Embedding

Based on the graph neural network model [29], [30], [31], we conduct the single graph embedding. Intuitively, it is a message propagation process, where the embedding vectors of nodes are taken as the messages for their neighbors. Hence, the embedding vectors of nodes can be iteratively updated by their neighbors. In this way, after convergence, the nodes' embeddings can naturally capture the graph's structure and feature information.

4.1.1 Message Router

Message router (MR) is designed to capture the structure distribution of a graph. MR is instantiated as a neural network component. The goal of the message router is that, for an edge (u, v), it decides how much the message can be sent from node u to node v. Intuitively, message routing influences the flow of message propagation. Specifically, for a given edge (u, v), the message router can be represented as:

$$r_i = g_{rt}(h_i) \tag{1}$$

$$\alpha_{v,u} = g_r\Big(r_v, r_u, \{r_p : p \in N(v)\}\Big) \tag{2}$$

where h_i refers to the current embedding vector of node i, g_{rt} is a neural network layer which transforms the node embedding h_i into the routing space, r_i is the routing level embedding of node i, and g_r is a routing algorithm. For the given edge (u, v), the inputs of g_r are the routing level embeddings of node u, node v, and nodes $N(v)$ ($N(v)$ denotes the neighbor nodes of node v). Then, the output $\alpha_{v,u} \in R$ of g_r is taken as the routing weight that dynamically controls the message propagation from node u to node v.

In this way, MR implicitly embeds the structure information of the source graph $\mathcal{G}^s$ and that of the target graph $\mathcal{G}^t$ into the same routing space since the message routing weight $\alpha_{v,u}$ is influenced by node v's degree value. The routing level embedding distribution of $\mathcal{G}^s$ and that of $\mathcal{G}^t$ in the routing space reflect the structure distributions of $\mathcal{G}^s$ and $\mathcal{G}^t$ to a certain extent, respectively.

4.1.2 Message Aggregator

Message Aggregator (MA) is designed to capture the feature distribution of a graph. MA is instantiated as another neural network component, where for each node v, it aggregates the messages of v's neighbors. Specifically, for a given node v, we define the message aggregator as

$$m_i = g_{mt}(h_i) \tag{3}$$

$$h_v^* = g_m\left(\textstyle\sum_{u \in N(v)} \alpha_{v,u} m_u, h_v\right) \tag{4}$$

where h_i is the current embedding vector of node i, g_{mt} is a neural network layer which transforms the node embedding h_i into the aggregation space, m_i is the aggregation level embedding of node i, and g_m is an aggregation algorithm. For the given node v, the inputs of g_m contain two parts. One is the current embedding vector of node v (i.e., h_v), which is used to preserve the information of center node v. The other is the weighted average vector of the aggregation level embeddings from $N(v)$, where the weight $\alpha_{v,u}$ is calculated by the message router (introduced in Section 4.1.1). The output h_v^* is taken as the updated embedding (feature) of node v.

In this way, MA implicitly embeds the feature information of the source graph $\mathcal{G}^s$ and that of the target graph $\mathcal{G}^t$ into the same aggregation space since the updated embedding h_v^* is influenced by current features of node v and its neighbors. The updated embedding distribution of $\mathcal{G}^s$ and that of $\mathcal{G}^t$ in the aggregation space reflect the feature distributions of $\mathcal{G}^s$ and $\mathcal{G}^t$ to a certain extent, respectively.

Note that although g_{rt} and g_{mt} seem to have similar inputs and outputs, it is worth designing two separate neural network layers since they capture different information of a graph — g_{rt} captures the structure information in the routing space and g_{mt} captures the feature information in the aggregation space.

4.1.3 Stacked Message Propagation

As introduced above, with the help of the message router and the message aggregator, the embedding of a node can be updated by its neighbors' embeddings, which captures the 1-hop (neighbor) structure information. Since the high-order structure information is helpful to enrich the embedding, a stacked L-layer framework is designed to explore the high-order structure. Specifically, at the l-th layer, the message router can be expressed as

$$r_i^{(l)} = g_{rt}^{(l)}\left(h_i^{(l-1)}\right) \tag{5}$$

$$\alpha_{v,u}^{(l)} = g_r^{(l)}\left(r_v^{(l)}, r_u^{(l)}, \left\{r_p^{(l)} : p \in N(v)\right\}\right) \tag{6}$$

where $h_v^{(0)}$ is initialized by the node feature vector, i.e., $h_v^{(0)} = x_v$.

Analogously, the message aggregator can be rewritten as

$$m_i^{(l)} = g_{mt}^{(l)}\left(h_i^{(l-1)}\right) \tag{7}$$

$$h_v^{(l)} = g_m^{(l)}\left(\textstyle\sum_{u \in N(v)} \alpha_{v,u}^{(l)} m_u^{(l)}, h_v^{(l-1)}\right) \tag{8}$$

where the output $h_v^{(l)}$ of $g_m^{(l)}$ is taken as the l-hop-aware node embedding, which can be taken as the input of the message router at the $(l+1)$-th layer.

Finally, the output embedding from the last layer is taken as the learned node embedding. In this way, the L-layer graph neural network can capture the L-hop structure.

4.2 Alignment across Graphs

As introduced in Section 4.1, there are two main components during the embedding learning, i.e., the message router and the message aggregator. To eliminate the graph bias and learn adaptive node embeddings, we design an alignment mechanism in these two phases. On the one hand, the proposed alignment can encourage the embedding distributions of the source graph and the target graph to be more similar to tackle the first challenge in the adaptive task, i.e., graph bias. On the other hand, the unlabeled target nodes' information can be incorporated into the alignment. We can take this alignment information as an auxiliary signal in the training phase to deal with the second challenge, i.e., limited information.

4.2.1 Message Routing Alignment

Message Routing Alignment (MRA) is designed to reduce the structure bias between graphs. The structure distribution mainly affects the message propagation in a graph. For example, the source graph is a dense graph and the target graph is a sparse one, i.e., they have different degree distributions. In the source graph, the messages spread quickly and the global structure is more likely to be explored since the connection is dense. In the target graph, the messages are more likely to spread in the local structures since the connection is sparse. To apply the patterns learned from the dense source graph to the sparse target graph well, we need to eliminate the discrepancy between the

structure distribution of the source graph and that of the target graph, i.e., structure bias. As discussed in Section 4.1.1, the distribution of routing level embedding r_i in MR implicitly captures the structure distribution of a graph in the routing space. Hence, to align the structure distributions, we focus on reducing the discrepancy between the routing level embedding distribution of the source graph and that of the target graph.

Specifically, we define $r_i^{(l)s}$ as the routing level embedding of node i in the source graph at the l-th layer and $r_j^{(l)t}$ as the routing level embedding of node j in the target graph at the l-th layer. $P_r^{(l)s}$ and $P_r^{(l)t}$ are the distributions of the routing level embeddings $r_i^{(l)s}$ and $r_j^{(l)t}$, respectively. To eliminate the discrepancy between $P_r^{(l)s}$ and $P_r^{(l)t}$, we design two loss terms in MRA: distance loss and adversary loss.

Distance Loss in MRA. We define a distance loss $d_{mra}^{(l)}(P_r^{(l)s}, P_r^{(l)t})$ for MRA in Equation 9. $d_{mra}^{(l)}(P_r^{(l)s}, P_r^{(l)t})$ measures the distance between $P_r^{(l)s}$ and $P_r^{(l)t}$ by calculating the distance between the expected values of the routing level embeddings $r_i^{(l)s}$ and $r_j^{(l)t}$ sampled from $P_r^{(l)s}$ and $P_r^{(l)t}$. Please note $d_{mra}^{(l)}(P_r^{(l)s}, P_r^{(l)t})$ can be applied in each layer of the model.

$$d_{mra}^{(l)}(P_r^{(l)s}, P_r^{(l)t}) = \left\| E_{r_i^{(l)s} \sim P_r^{(l)s}} \left[r_i^{(l)s} \right] - E_{r_j^{(l)t} \sim P_r^{(l)t}} \left[r_j^{(l)t} \right] \right\| \tag{9}$$

Intuitively, if we minimize the distance $d_{mra}^{(l)}(P_r^{(l)s}, P_r^{(l)t})$, the routing level embedding distributions of the source graph and the target graph will be encouraged to be similar, and thus structure bias can be reduced.

Adversary Loss in MRA. Inspired by the idea of adversarial learning applied in privacy protection [32], [33], [34] and domain adaptation for image data [35], we design an adversary loss $A_{mra}(P_r^{(l)s}, P_r^{(l)t})$ for MRA in Equation 10 to further reduce the structure bias. D_{mra} is a graph discriminator (instantiated as a logistic regression classifier) that aims to distinguish whether a routing level embedding (i.e., $r_i^{(l)s}$ or $r_j^{(l)t}$) belongs to the source graph or the target graph. $A_{mra}(P_r^{(l)s}, P_r^{(l)t})$ is defined by the negative of binary cross-entropy loss, where the label for $r_i^{(l)s}$ is 1 and that for $r_j^{(l)t}$ is 0.

$$\begin{aligned} A_{mra}^{(l)}(P_r^{(l)s}, P_r^{(l)t}) =& E_{r_i^{(l)s} \sim P_r^{(l)s}} \log \left[D_{mra} \left(r_i^{(l)s} \right) \right] \\ &+ E_{r_j^{(l)t} \sim P_r^{(l)t}} \log \left[1 - D_{mra} \left(r_j^{(l)t} \right) \right] \end{aligned} \tag{10}$$

To reduce the discrepancy between $P_r^{(l)s}$ and $P_r^{(l)t}$, we conduct adversarial learning on D_{mra} and the message router (MR), which is a minimax optimization problem on $A_{mra}(P_r^{(l)s}, P_r^{(l)t})$. Specifically, D_{mra} is learned by maximizing $A_{mra}(P_r^{(l)s}, P_r^{(l)t})$ to distinguish the source graph from the target graph that a routing level embedding (i.e., $r_i^{(l)s}$ or $r_j^{(l)t}$) belongs to, while MR is learned by minimizing $A_{mra}(P_r^{(l)s}, P_r^{(l)t})$ to confuse D_{mra}. This optimization process can help MR generate adaptive routing level embeddings that follow similar distributions on the source graph, i.e., $P_r^{(l)s}$, and the target graph, i.e., $P_r^{(l)t}$, and thus structure bias can be reduced.

4.2.2 Message Aggregating Alignment

Message Aggregating Alignment (MAA) is designed to reduce the feature bias between graphs. The feature distribution characterizes inherent properties of nodes and may vary from one graph to another. When the feature bias exists, the patterns learned from the source graph may not be suitable to be used in the target graph. Thus, we need to eliminate the discrepancy between the feature distribution of the source graph and that of the target graph. As discussed in Section 4.1.2, the distribution of updated embeddings h_v^* in MA implicitly captures the structure distribution of a graph. Hence, to align the feature distributions, we focus on reducing the discrepancy between the updated embedding distribution of the source graph and that of the target graph.

Specifically, we define $h_i^{(l)s}$ as the updated embedding of node i in the source graph at the l-th layer and $h_j^{(l)t}$ as the updated embedding of node j in the target graph at the l-th layer. $P_a^{(l)s}$ and $P_a^{(l)t}$ are the distributions of the updated embeddings $h_i^{(l)s}$ and $h_j^{(l)t}$, respectively. Similar to the design of MRA, to eliminate the discrepancy between $P_a^{(l)s}$ and $P_a^{(l)t}$, we also design two loss terms in MAA: distance loss and adversary loss.

Distance Loss in MAA. We define a distance loss $d_{maa}^{(l)}(P_a^{(l)s}, P_a^{(l)t})$ for MAA in Equation 11. $d_{maa}^{(l)}(P_a^{(l)s}, P_a^{(l)t})$ measures the distance between $P_a^{(l)s}$ and $P_a^{(l)t}$ by calculating the distance between the expected values of the updated embeddings $h_i^{(l)s}$ and $h_j^{(l)t}$ sampled from $P_a^{(l)s}$ and $P_a^{(l)t}$. Please note $d_{maa}^{(l)}(P_a^{(l)s}, P_a^{(l)t})$ can be applied in each layer of the model.

$$d_{maa}^{(l)}(P_a^{(l)s}, P_a^{(l)t}) = \left\| E_{h_i^{(l)s} \sim P_a^{(l)s}} \left[h_i^{(l)s} \right] - E_{h_i^{(l)t} \sim P_a^{(l)t}} \left[h_i^{(l)t} \right] \right\| \tag{11}$$

Intuitively, if we minimize the distance $d_{maa}^{(l)}(P_a^{(l)s}, P_a^{(l)t})$, the updated embedding distributions of the source graph and the target graph will be encouraged to be similar, and thus feature bias can be reduced.

Adversary Loss in MAA. Similar to MRA, we design an adversary loss $A_{maa}(P_a^{(l)s}, P_a^{(l)t})$ for MAA in Equation 12 to further reduce the feature bias. D_{maa} is a graph discriminator (instantiated as a logistic regression classifier) that aims to distinguish whether an updated embedding (i.e., $h_i^{(l)s}$ or $h_j^{(l)t}$) belongs to the source graph or the target graph. $A_{maa}(P_a^{(l)s}, P_a^{(l)t})$ is defined by the negative of binary cross-entropy loss, where the label for $h_i^{(l)s}$ is 1 and that for $h_j^{(l)t}$ is 0.

$$\begin{aligned} A_{maa}^{(l)}(P_a^{(l)s}, P_a^{(l)t}) =& E_{h_i^{(l)s} \sim P_a^{(l)s}} \log \left[D_{maa} \left(h_i^{(l)s} \right) \right] \\ &+ E_{h_j^{(l)t} \sim P_a^{(l)t}} \log \left[1 - D_{maa} \left(h_j^{(l)t} \right) \right] \end{aligned} \tag{12}$$

To reduce the discrepancy between $P_a^{(l)s}$ and $P_a^{(l)t}$, we conduct adversarial learning on D_{maa} and the message aggregator (MA), which is a minimax optimization problem on $A_{maa}(P_a^{(l)s}, P_a^{(l)t})$. Specifically, D_{maa} is learned by maximizing $A_{maa}(P_a^{(l)s}, P_a^{(l)t})$ to distinguish the source graph from the target graph that an updated embedding (i.e., $h_i^{(l)s}$ or $h_j^{(l)t}$) belongs to, while MR is learned by minimizing

$A_{maa}(P_a^{(l)s}, P_a^{(l)t})$ to confuse D_{maa}. This optimization process can help MA generate adaptive updated embeddings that follow similar distributions on the source graph, i.e., $P_a^{(l)s}$, and the target graph, i.e., $P_a^{(l)t}$, and thus feature bias can be reduced.

4.3 Objective Function

The objective function of the unified framework is composed of two parts, i.e., supervised loss and unsupervised loss. The supervised loss is defined as the cross-entropy error over all labeled nodes in the source graph, i.e.,

$$\begin{aligned} z_i &= softmax\left(W_z h_i^{(L)}\right) \\ \mathcal{L}_s &= -\textstyle\sum_{i\in\mathcal{V}^s} y_i^\top \ln z_i \end{aligned} \tag{13}$$

where W_z is a weight matrix, $h_i^{(L)}$ is the learned embedding of node i (i.e., the updated embedding from the last layer), z_i is the predicted probability over all classes, y_i is the ground-truth label denoted by a one-hot vector, and $\mathcal{V}^s$ refers to the set of all labeled nodes in the source graph.

Unsupervised loss refers to the distance loss and adversary loss in MRA and MAA over all layers, i.e.,

$$\mathcal{L}_{mra}=\textstyle\sum_{l=1}^{L}\left[d_{mra}^{(l)}(P_r^{(l)s}, P_r^{(l)t})+A_{mra}^{(l)}(P_r^{(l)s}, P_r^{(l)t})\right] \tag{14}$$

$$\mathcal{L}_{maa}=\textstyle\sum_{l=1}^{L}\left[d_{maa}^{(l)}(P_a^{(l)s}, P_a^{(l)t})+A_{maa}^{(l)}(P_a^{(l)s}, P_a^{(l)t})\right] \tag{15}$$

Hence, the final objective function is presented as

$$\min_{\substack{MR\\MA}}\max_{\substack{D_{mra}\\D_{maa}}} \mathcal{L} = \mathcal{L}_s + \gamma\mathcal{L}_{mra} + \lambda\mathcal{L}_{maa} \tag{16}$$

where γ and λ are penalty parameters. The model is trained in a minimax optimization manner, while $\mathcal{L}$ is minimized over MR and MA but maximized over D_{mra} and D_{maa}.

5 GraphAE

In this section, we bring forward the implementation of the proposed framework, i.e., GraphAE. Furthermore, we analyze the time cost of GraphAE.

5.1 Implementation of the Unified Framework

So far, we have presented a unified framework for the adaptive task. This framework can be implemented and extended in various ways. In this subsection, we propose the GraphAE network, which is an implementation of the proposed unified framework to address the adaptive task. The details of GraphAE are explicated as follows.

Message Router. Firstly, a weight matrix with an activation function is used to transform the embedding into the routing space. Then, an attention mechanism is used to implement the routing algorithm.

$$\begin{aligned} r_i^{(l)} &= \sigma\left(W_{rt}^{(l)} h_i^{(l-1)}\right) \\ \alpha_{v,u}^{(l)} &= \frac{exp\left(\sigma\left(a^{(l)\top}\left[r_v^{(l)}||r_u^{(l)}\right]\right)\right)}{\sum_{p\in N(v)} exp\left(\sigma\left(a^{(l)\top}\left[r_v^{(l)}||r_p^{(l)}\right]\right)\right)} \end{aligned} \tag{17}$$

where $W_{rt}^{(l)}$ is the weight matrix, $a^{(l)}$ is a weight vector, σ is the activation function, and $||$ refers to the concatenation operation.

Message Aggregator. The message aggregator is implemented as

$$\begin{aligned} m_i^{(l)} &= \sigma\left(W_{mt}^{(l)} h_i^{(l-1)}\right) \\ h_v^{(l)} &= \sigma\left(W_m^{(l)}\left[\left(\textstyle\sum_{u\in N(v)} \alpha_{v,u}^{(l)} m_u^{(l)}\right)||h_v^{(l-1)}\right]\right) \end{aligned} \tag{18}$$

where $W_{mt}^{(l)}$ and $W_m^{(l)}$ are weight matrices.

Message Routing Alignment & Message Aggregating Alignment. The main idea of these two alignment strategies is to eliminate the structure bias and the feature bias between the source graph and the target graph.

For the specific design of the distance loss, we explore the idea of the multiple kernel variant of MMD (MK-MMD) [36], which calculates the distance between the expected values of two embedding vectors x^s and x^t from two distributions P^s and P^t in the kernel Hilbert space $\mathcal{H}_k$ endowed with a characteristic kernel k. The squared formulation of MK-MMD is defined as

$$\begin{aligned} d_k^2(P^s, P^t) &\triangleq \left|\left|E_{P^s}\left[\phi(x^s)\right] - E_{P^t}\left[\phi(x^t)\right]\right|\right|_{\mathcal{H}_k}^2 \\ \left\langle\phi(x^s), \phi(x^t)\right\rangle &= k\left(x^s, x^t\right) = \textstyle\sum_{i=1}^{I}\beta_i k_i\left(x^s, x^t\right) \\ s.t.\ \beta_i &\geq 0, \textstyle\sum_{i=1}^{I}\beta_i = 1 \end{aligned} \tag{19}$$

where ϕ is defined as the feature map, $k(x^s, x^t)$ denotes the convex combination of I positive semi-definite (PSD) kernels, and β_i is the coefficient of the i-th PSD kernel. The most important property of MK-MMD is that $P^s = P^t$ *iff* $d_k^2(P^s, P^t) = 0$. Then, the specific distance losses in MRA and MAA can be expressed as

$$d_{mra}^{(l)}(R^{(l)s}, R^{(l)t}) = d_k^2\left(R^{(l)s}, R^{(l)t}\right) \tag{20}$$

$$d_{maa}^{(l)}(H^{(l)s}, H^{(l)t}) = d_k^2\left(H^{(l)s}, H^{(l)t}\right) \tag{21}$$

where $R^{(l)s} = \{r_i^{(l)s}, \forall i \in V^s\}$, $R^{(l)t} = \{r_j^{(l)t}, \forall j \in V^t\}$, $H^{(l)s}=\{h_i^{(l)s}, \forall i \in V^s\}$, and $H^{(l)t}=\{h_j^{(l)t}, \forall j \in V^t\}$.

For the specific design of the adversary loss, we explore the idea of the conditional adversarial mechanism [35], which conditions the graph discriminators D_{mra} and D_{maa} on the softmax prediction z_i (see Equation 13). Then the specific adversary losses in MRA and MAA can be expressed as

$$\begin{aligned} A_{mra}^{(l)}(R^{(l)s}, R^{(l)t}) =& E\log\left[D_{mra}\left(r_i^{(l)s}||z_i^s\right)\right] \\ &+ E\log\left[1 - D_{mra}\left(r_j^{(l)t}||z_j^t\right)\right] \end{aligned} \tag{22}$$

$$\begin{aligned} A_{maa}^{(l)}(H^{(l)s}, H^{(l)t}) =& E\log\left[D_{maa}\left(h_i^{(l)s}||z_i^s\right)\right] \\ &+ E\log\left[1 - D_{maa}\left(h_j^{(l)t}||z_j^t\right)\right] \end{aligned} \tag{23}$$

where z_i^s is the softmax prediction of node i in the source graph, z_j^t is the softmax prediction of node j in the target graph, $||$ is the concatenation operation, $D_{mra}\left(r_i^{(l)s}||z_i^s\right)$ outputs the conditional probability that $r_i^{(l)s}$ belongs to the source graph conditioned on z_i^s, and $D_{mra}\left(r_j^{(l)t}||z_j^t\right)$, $D_{maa}\left(h_i^{(l)s}||z_i^s\right)$, $D_{maa}\left(h_j^{(l)t}||z_j^t\right)$ express similar conditional probabilities.

In practice, a mini-batch strategy is usually adopted to train the graph neural network. In each training step, we feed a batch [3] (i.e., one batch B^s for the source graph and another batch B^t for the target graph) of nodes into GraphAE rather than the whole graph. Then, the distance loss and the adversary loss in MRA and MAA can be approximated by the nodes in the batches B^s and B^t, i.e., $R^{(l)s}$=$\{r_v^{(l)s}, \forall v \in B^s\}$, $H^{(l)s}$=$\{h_v^{(l)s}, \forall v \in B^s\}$, $R^{(l)t}$=$\{r_v^{(l)t}, \forall v \in B^t\}$, and $H^{(l)t}$=$\{h_v^{(l)t}, \forall v \in B^t\}$.

5.2 Complexity Analysis

The time cost of GraphAE mainly comes from four parts, i.e., MR, MA, MRA, and MAA. Considering a batch-style training, at the l-th layer, MR has the computational complexity of $O(BI^{(l)}F^{(l)} + E_B I^{(l)} A^{(l)})$, where B is the batch size ($B = |B^s| = |B^t|$), $I^{(l)}$ is the number of input features at the l-th layer, $F^{(l)}$ is the number of the transformed features at the l-th layer, $A^{(l)}$ is the number of the attention features, and E_B is the number of edges among the nodes in the current batch. MA has the computational complexity of $O(BI^{(l)}F^{(l)} + BI^{(l+1)}(F^{(l)} + I^{(l)}))$. Both MRA and MAA have the computational complexity of $O(B^2F^{(l)} + BF^{(l)})$. Thus, the total time cost for two input batches, i.e., B^s and B^t, is $O(2\sum_{l=1}^{L}(2BI^{(l)}F^{(l)} + E_B I^{(l)} A^{(l)} + BI^{(l+1)}(F^{(l)} + I^{(l)}) + B^2F^{(l)} + BF^{(l)}))$, where L is the number of layers.

6 MGraphAE

In Section 5, we introduce a graph adaptive embedding method that focuses on learning knowledge from a single source graph. However, in practice, it is common that we have multiple source graphs. For instance, considering Example 1 introduced in Section 1, there may be several e-commerce networks that are constructed in different days. The presented unified framework in Section 4 can be easily extended to deal with this new situation. Specifically, several new branches (Each of them is similar with the branch for the single source graph $\mathcal{G}^s$ in Figure 2) are added for the multiple source graphs, and the branch for the target graph $\mathcal{G}^t$ in Figure 2 remains unchanged. Due to the limited space, we omit the diagram of this extended framework. Furthermore, we propose a new model called MGraphAE based on the extended framework, which learns knowledge from multiple source graphs to analyze a single target graph. It can also be regarded as a multi-graph version of GraphAE. The corresponding task is called the **M-Adaptive Task**.

The two major challenges, i.e., the graph bias and limited information, still exist in this situation. Similar to GraphAE, we can also adopt the proposed two alignment strategies to address these challenges. The difference is that the alignment is applied for multiple pairs of graphs rather than one pair of graphs.

Specifically, we consider the graph bias between each of the source graphs and the target graph. Then, the message routing alignment in MGraphAE can be rewritten as

$$\frac{1}{N}\sum_{i=1}^{N}\left[d_{mra}^{(l)}(R_i^{(l)s}, R^{(l)t}) + A_{mra}^{(l)}(R_i^{(l)s}, R^{(l)t})\right] \quad (24)$$

where N refers to the number of the source graphs, $R_i^{(l)s}$ refers to the routing level embedding set of nodes from the i-th source graph and $R^{(l)t}$ is the routing level embedding set of nodes from the target graph. Analogously, the message aggregating alignment in MGraphAE can be rewritten as

$$\frac{1}{N}\sum_{i}^{N}\left[d_{maa}^{(l)}(H_i^{(l)s}, H^{(l)t}) + A_{maa}^{(l)}(H_i^{(l)s}, H^{(l)t})\right] \quad (25)$$

where N refers to the number of the source graphs, $H_i^{(l)s}$ refers to the feature level embedding set of nodes from the i-th source graph and $H^{(l)t}$ is the feature level embedding set of nodes from the target graph. By reducing the graph bias of all the pairs of graphs, we can learn adaptive node embeddings, which carefully consider both the structure bias and the feature bias between different source graphs and the target graph.

7 Experiments

In this section, we investigate the potential benefits of our proposed model against a number of state-of-the-art models. Specifically, we address the following questions:

- **Q1** How does GraphAE perform compared with the state-of-the-art models in addressing the adaptive task?
- **Q2** Does MGraphAE still work on the m-adaptive task?
- **Q3** How much (if any) improvement does the alignment strategies (i.e., MRA and MAA) provide?
- **Q4** Can our framework be incorporated with the state-of-the-art GNN models to enhance their performance?
- **Q5** How do the penalty parameters λ and γ affect the performance of our model?

7.1 Data Sets

7.1.1 Ali-CVR

Ali-CVR is a public data set[1] gathered from the real-world browsing logs of the recommender system in Taobao, the largest online retail platform in the world. Each log entry records the behavior (e.g., purchase) of one user on the browsed items (goods). To evaluate the proposed model, we construct three item-based graphs from Ali-CVR.

- **Ali-Normal (AN)** is a graph constructed by user logs (behaviors) in one normal day. Each node in the graph refers to one item. Each item has 40 features including the item brand, the item price, the item sales, and so on. If two items are viewed by the same user within 30 minutes, there is one edge between these two items. Each item belongs to one of 31 classes including computer hardware, clothes, books, and so on. We take the item category (e.g., electronics or clothes) as the label information.
- **Ali-11 (A11)** is constructed on a special day called the shopping carnival. The number of user logs increases tremendously. Hence, although this graph is

1. https://tianchi.aliyun.com/dataset/dataDetail?dataId=58&lang=en-us

TABLE 1
Data Sets Information

Data sets	#Nodes	#Edges	#Classes	#Features
AN	46,800	946,175	31	40
A11	57,493	2,155,835	31	40
AB11	51,785	1,422,281	31	40
D50	960	1,015	21	256
D15	2,244	2,407	21	256
D60	5,443	6,386	21	256

constructed in the same way as AN, both the graph structure and the node features (e.g., the view count for each item in one day) of A11 are different from those of AN.

- **Ali-Before-11 (AB11)** is constructed on another special day, i.e., the day before the shopping carnival. Obviously, the distribution of AB11 is also different from that of AN or A11.

The statistics of the three graphs from Ali-CVR are summarized in Table 1. As introduced in Section 1, we firstly apply the proposed model in the e-commerce scene to analyze the effectiveness of GraphAE. Specifically, we evaluate our model across six adaptive tasks (source graph $\rightarrow$ target graph), i.e., **AN** $\rightarrow$ **A11**, **AN** $\rightarrow$ **AB11**, **AB11** $\rightarrow$ **AN**, **AB11** $\rightarrow$ **A11**, **A11** $\rightarrow$ **AN**, and **A11** $\rightarrow$ **AB11**. For simplification, we define these six tasks as A1, A2, A3, A4, A5, and A6, respectively. For the m-adaptive task which considers multiple source graphs, there are three m-adaptive tasks ([source graph1, source graph2, ...] $\rightarrow$ target graph), i.e., [**AN**, **AB11**] $\rightarrow$ **A11**, [**AN**, **A11**] $\rightarrow$ **AB11**, and [**A11**, **AB11**] $\rightarrow$ **AN**. For simplification, we define these three tasks as MA1, MA2, and MA3, respectively.

7.1.2 DBLP-Citation

DBLP-Citation is another public data set [2] gathered from DBLP. The data set includes 21 different journals or conferences including TKDE, TKDD, VLDB, and so on. Each record in this data set contains the meta-data of one paper such as the abstract, the authors, the year, the venue, and the title. To fully evaluate the proposed model, we also construct three paper-based graphs from DBLP-Citation.

- **DBLP-9500 (D50)** is constructed from the papers published in 21 different journals or conferences from 1995 to 2000 in DBLP-Citation. We take each paper as a node associated with a label that refers to the journal or the conference where the paper is published. The edges represent the citation relationship between papers. To extract the original attributes of each node, we first calculate the TF-IDF features of the title and then conduct SVD on the TF-IDF features. We take the vector generated by SVD as the original attributes of each node.
- **DBLP-0105 (D15)** is also constructed from the papers published in 21 different journals or conferences. The only difference between D15 and D50 is that all of the papers in D15 are published from 2001 to 2005. This means, although both of these two graphs are constructed in a similar way, the different research interests (e.g., different hot topics) lead to totally different distributions between D50 and D15.
- **DBLP-0610 (D60)** is constructed from another period, i.e., from 2006 to 2010. Obviously, the distribution of this graph is also different from that of D50 or D15.

The statistics of the three graphs derived from DBLP-Citation are also shown in Table 1. As analyzed in Section 1, apart from the e-commerce scene, we also conduct experiments on the citation networks to give an exhaustive understanding of the proposed model. Specifically, we conduct another six adaptive tasks, i.e., **D50** $\rightarrow$ **D15**, **D50** $\rightarrow$ **D60**, **D15** $\rightarrow$ **D50**, **D15** $\rightarrow$ **D60**, **D60** $\rightarrow$ **D50**, and **D60** $\rightarrow$ **D15**. For simplification, we define these six tasks as D1, D2, D3, D4, D5, and D6, respectively. Similarly, for the m-adaptive task which considers multiple source graphs, we also conduct three m-adaptive tasks, i.e., [**D50**, **D15**] $\rightarrow$ **D60**, [**D60**, **D15**] $\rightarrow$ **D50**, and [**D50**, **D60**] $\rightarrow$ **D15**. For simplification, we define these three tasks as MD1, MD2, and MD3, respectively.

7.2 Baselines

To fully evaluate the proposed models, we consider the state-of-the-art baselines as follows for performance comparisons.

For the transductive embedding methods, as mentioned in Section 1, these methods cannot be applied to the adaptive task. Hence, we do not consider these methods as baselines in the experiments.

For the inductive embedding methods, most of them learn a mapping function from the source graph, and can directly infer the embeddings of nodes in the target graph. Thus, we train these models on the source graph and apply the learned models to the target graph. The following inductive embedding methods are taken as our baselines:

- **GCN** [2] is a graph convolution-based classification model, which further simplifies the filtering operations introduced in ChebNet [37] by only using the first-order neighbors. It means that the model extracts the 1-localized information for each node in each convolution layer, and then the high-order relational features can be captured by stacking multiple convolution layers.
- **GraphSAGE** [3] extends GCN to a more general architecture, and introduces aggregation and combination functions to learn node embeddings. Four different aggregation strategies are proposed to aggregate neighbors' information, i.e., Mean, GCN, Pool, and LSTM.
  – *Mean*. The mean strategy takes the element-wise mean of the vectors of neighbors.
  – *GCN*. The GCN strategy aggregates neighbors' information in the GCN [2] manner, i.e., the element-wise averaging of the vectors of neighbors and the center node.
  – *Pool*. The element-wise max pooling is applied to aggregate the information of neighbors.
  – *LSTM*. The neighbors' information is aggregated in a sequential manner.

2. https://www.aminer.cn/citation

For simplicity, we use GS-Mean, GS-GCN, GS-Pool, and GS-LSTM to denote four GraphSAGE variants with the aggregation strategies of Mean, GCN, Pool, and LSTM, respectively.

- **GAT** [1] considers that different nodes have different degrees, and implicitly specifies different weights for different neighbor nodes by employing the multi-head self-attention strategy [38].
- **GeniePath** [24] improves GAT by using an attention mechanism to choose the receptive fields of graph neural networks with the help of breadth and depth explorations.

For the domain adaptive embedding methods, we choose one GNN pre-training method and two domain adaptive GNNs as our baselines:

- **SCNN** [25] adopts the strategies of pre-training and fine-tuning to learn embeddings across graphs.
- **AdaGCN** [27] uses a domain adaptation component to reduce the distribution divergence between different domains.
- **UDA-GCN** [28] utilizes an attention mechanism to combine the representations of a dual graph convolutional network that is followed by a domain classifier loss to make the representations domain-invariant.

We must point out that, some label information of the target graph is still needed to train SCNN, which does not meet the adaptive task setting (the label information of the target graph is unavailable). Specifically, we feed five nodes of the target graph per class into SCNN to fine-tune it. Although leaking target label information to SCNN is unfair to other baselines and our model, we still try to take SCNN as a baseline to give a detailed comparison with GraphAE and MGraphAE.

Furthermore, following the work in [1], [3], we also make comparisons with **MLP**. It directly feeds the original node features into a multilayer perceptron (MLP) classifier. This means MLP takes each node as a separate sample and does not consider the graph structure at all.

Settings. To have a fair comparison, we always use two layers for all models, and the embedding dimension is 128. We set $\beta = 1/I$ and use a family of I=5 Gaussian kernels $\{k_i(x_a, x_b) = e^{-||x_a - x_b||^2/\mu_i}\}_{i=1}^{I}$ in MK-MMD by varying bandwidth μ_i between $2^{-8}\mu$ and $2^8\mu$ where μ is set to the median pairwise distance on the training data [39]. We set $\gamma = 1$ and $\lambda = 1$ for all data sets. All models are trained using the Adam optimizer [40] with an initial learning rate of 0.01. For the six adaptive tasks on Ali-CVR, we take 5000 nodes from the source graph as the validation nodes for all methods. For the other six adaptive tasks on DBLP-Citation, we take 10% of the nodes from the source graph as the validation nodes for all methods. For SCNN, during fine-tuning on the target graph, we take five nodes per class from the target graph as the validation nodes. We use an early stopping strategy on the validation accuracy with a patience of 20 epochs.

7.3 Results for Adaptive Tasks

For the adaptive tasks, we report the node classification accuracy in the target graphs. Tables 2 and 3 compare the performance of GraphAE with the state-of-the-art baselines. These results provide positive answers to our question **Q1**:

TABLE 2
The Results for Adaptive Tasks on Ali-CVR.

Method	A1	A2	A3	A4	A5	A6	average
MLP	26.84	27.58	26.63	25.50	25.59	23.99	26.02
GS-Mean	33.00	37.17	34.33	30.31	34.04	27.84	32.78
GS-Pool	25.18	22.07	29.41	23.73	27.34	23.23	25.16
GS-LSTM	39.73	46.53	40.47	31.83	33.30	27.41	36.55
GS-GCN	32.01	35.28	32.80	28.29	32.24	26.78	31.23
GCN	29.59	30.06	23.72	31.13	30.61	27.25	28.73
GAT	37.52	44.16	38.38	33.83	39.73	31.03	37.44
GeniePath	33.14	40.10	39.29	34.99	37.93	37.71	37.19
SCNN	35.79	35.62	38.32	35.88	38.08	36.40	36.68
AdaGCN	42.19	51.89	49.08	40.92	42.31	43.50	44.98
UDA-GCN	38.21	41.42	42.82	38.84	41.42	40.97	40.61
GraphAE	**48.92**	**52.49**	**51.68**	**47.34**	**49.55**	**49.60**	**49.93**

1) For the baseline MLP, GraphAE achieves a significant improvement on all twelve adaptive tasks. Specifically, GraphAE achieves average gains of 23.91% and 8.66% over MLP on Ali-CVR and DBLP-Citation, respectively.

2) For the inductive embedding baselines, we observe that GraphAE obtains the highest performance and achieves the state-of-the-art results on all twelve adaptive tasks. Compared with the best inductive embedding baseline, GraphAE achieves the gains by an average of 12.49% and 2.92% on Ali-CVR and DBLP-Citation, respectively.

3) Please note that SCNN is the only method to utilize the label information of target graphs, while GraphAE does not use the supervision from target graphs. Nonetheless, GraphAE still achieves better results than SCNN except one case (average gains of 13.25% and 2.85% on Ali-CVR and DBLP-Citation, respectively). Furthermore, we notice the bad case for GraphAE (i.e., D2), where SCNN achieves the best result (i.e., 22.14%) that is higher than the result of GraphAE (14.91%). The reasons are: (1) The graph D50 is extremely different from D60 (D60 contains about six times more nodes than D50), which makes it more difficult to learn useful patterns from such a small graph D50 and to infer the node embeddings of such a big and different graph D60. Actually, all of the baselines except SCNN achieve low accuracy in this case. (2) Due to the small and extremely different source graph D50, a little additional supervision (label information) from the target graph (D60) can significantly improve the performance. (3) Furthermore, we conduct additional experiments, i.e., providing some label information of the target graph for GraphAE as SCNN does. In this setting, GraphAE obtains 30.15% on the task D2 which is significantly better than SCNN. It further shows the label information from D60 guarantees the good performance of SCNN.

7.4 Results for M-Adaptive Tasks

In this subsection, we evaluate the performance of the multi-graph version of GraphAE, i.e., MGraphAE. We conduct experiments on tasks MA1, MA2, MA3, MD1, MD2, and MD3. The accuracy of the node classification in the target graph is reported in Table 4.

Compared with all the baselines, MGraphAE obtains the best results on all the six m-adaptive tasks. Specifically,

TABLE 3
The Results for Adaptive Tasks on DBLP-Citation.

Method	D1	D2	D3	D4	D5	D6	average
MLP	22.82	12.31	35.83	29.62	27.40	34.05	27.00
GS-Mean	26.74	14.09	43.75	34.67	37.29	39.88	32.74
GS-Pool	25.32	11.78	38.73	31.84	36.02	36.86	30.09
GS-LSTM	25.76	14.02	40.10	33.47	37.40	38.86	31.60
GS-GCN	25.72	12.83	41.44	35.41	40.19	39.04	32.44
GCN	24.91	13.32	29.38	27.83	33.23	32.35	26.84
GAT	25.80	13.85	42.19	34.03	36.15	38.24	31.71
GeniePath	25.27	13.63	41.35	32.52	28.96	36.50	29.71
SCNN	25.83	**22.14**	40.49	33.16	37.12	38.14	32.81
AdaGCN	26.11	13.94	37.57	31.26	29.02	40.33	29.71
UDA-GCN	26.71	13.19	44.09	34.42	35.32	39.81	32.26
GraphAE	**29.26**	14.91	**46.82**	**38.25**	**42.82**	**41.91**	**35.66**

TABLE 4
The Results for M-Adaptive Tasks.

Method	MA1	MA2	MA3	MD1	MD2	MD3	average
MLP	24.36	23.54	22.58	30.34	34.76	35.38	28.49
GS-Mean	41.01	33.34	44.27	32.72	40.72	41.24	38.88
GS-Pool	29.23	26.08	33.24	30.58	41.06	41.47	33.61
GS-LSTM	35.73	44.15	46.45	32.43	43.19	41.03	40.50
GS-GCN	36.73	29.71	40.29	32.75	43.34	41.66	37.41
GCN	27.90	27.28	31.44	27.31	36.33	38.51	31.46
GAT	40.87	39.35	46.57	29.10	42.91	39.63	39.74
GeniePath	38.82	40.02	42.62	29.52	40.19	38.29	38.24
SCNN	34.18	32.49	36.60	28.26	33.52	34.10	33.19
AdaGCN	42.90	51.93	48.98	30.83	40.02	40.21	42.48
UDA-GCN	39.71	42.17	43.09	32.48	45.22	39.90	40.43
MGraphAE	**51.10**	**52.62**	**52.27**	**36.15**	**48.12**	**42.60**	**47.14**

MGraphAE achieves the average gains from 4.66% to 18.65% over all baselines. These results answer the question **Q2**.

Furthermore, we try to analyze the influence of the number of source graphs, i.e., compare the results among Tables 2, 3, and 4. Specifically, for the same target graph, we compare the performance from a single source graph with that from multiple source graphs. Some observations can be summarized as follows: (1) In most cases, considering multiple source graphs (i.e., MGraphAE) could achieve better performance than considering only a single source graph (i.e., GraphAE). For example, all the three tasks A3, A5, and MA3 are conducted to infer the embeddings of nodes in the same target graph AN. MGraphAE (in the m-adaptive task MA3) achieves a better performance than GraphAE (in the adaptive tasks A3 and A5) and has gains of 0.59% and 2.72%, respectively. Similarly, in the m-adaptive tasks MA1, MA2, MD2, and MD3, MGraphAE all achieves better performance than GraphAE in corresponding adaptive tasks. This can be explained as MGraphAE could utilize more knowledge from multiple source graphs than GraphAE to infer node embeddings of the target graph. (2) We also notice that in individual cases, introducing more source graphs may not improve the performance. For example, MD1 and D4 are both conducted to infer the node embeddings of the same target graph D60. The performance of MGraphAE (in the m-adaptive task MD1) is worse than the performance of GraphAE (in the adaptive task D4). One of the main reasons may be that the structure of the graph D50 (one of the source graphs in MD1) is extremely different from D60 (the target graph in MD1), which is also the reason for the bad case in task D2 as said before. In this case, considering the patterns learned from the source graph D50 will damage the performance.

Clearly, given a practical situation, the selection of GraphAE or MGraphAE is one of interesting problems. There is an empirical approach to this problem based on our experience. In detail, for a certain graph, the ratio of its size (calculated by the sum of the node number and the edge number) to the target graph size, denoted as δ, could be used to decide whether this graph should be considered as an additional source graph. MGraphAE should be used to consider this graph as an additional source graph if δ is greater than a predefined threshold δ_0; otherwise, this graph should not be added into source graphs. δ_0 is set to 0.2 empirically in our experiments. For example, for GraphAE on A1 (**AN** $\rightarrow$ **A11**), δ of AB11 is greater than 0.2. Then MGraphAE should be used to consider AB11 as an additional source graph (i.e., [**AN**, **AB11**] $\rightarrow$ **A11**), which leads to a better performance (compare the performance between MA1 and A1). As a contrast, for GraphAE on D4 (**D15** $\rightarrow$ **D60**), δ of D50 is less than 0.2. Adding D50 into source graphs (i.e., [**D15**, **D50**] $\rightarrow$ **D60**) by using MGraphAE will lead to a worse performance (compare the performance between MD1 and D4). In this case, we should choose GraphAE rather than MGraphAE. Obviously, we can assign other values to δ_0 in other studies.

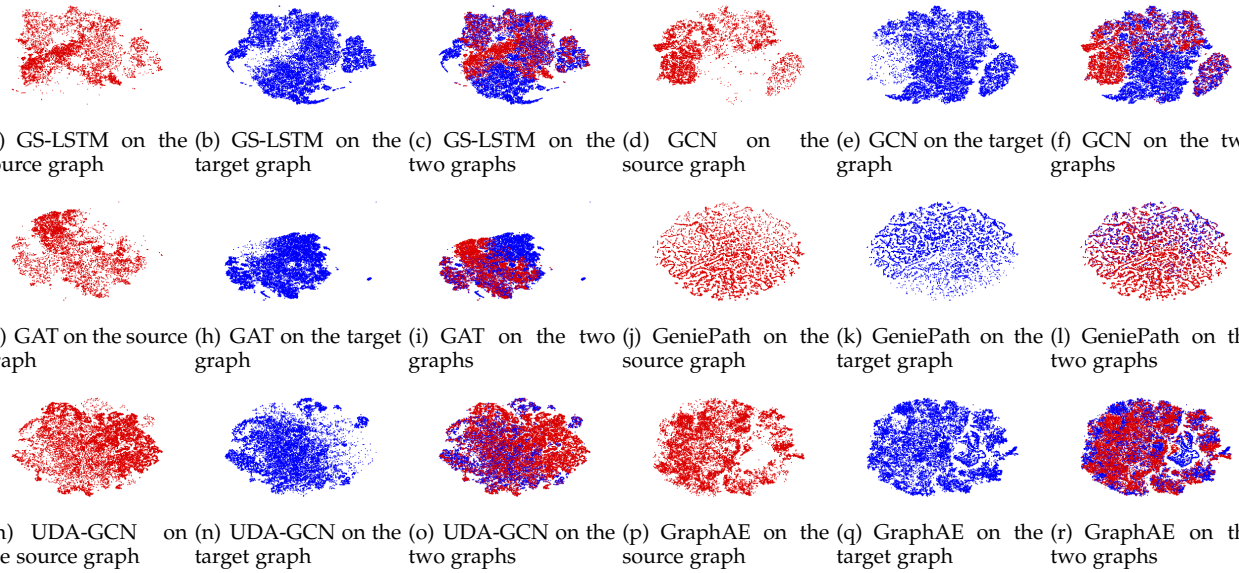
7.5 Visualization

Based on the results obtained in Section 7.3, in this subsection, we try to further analyze the reasons for the good performance of our proposed method. To fully understand the limitation of the existing work and the intuition of our proposed method, we visualize the node embeddings (learned by each method) of the source graph and the target graph, respectively. Specifically, the learned embeddings are plotted in a 2-D space by t-SNE [41]. Due to the limited space, we only show the results obtained on the adaptive task A6 (other tasks show similar results). In addition, we take GS-LSTM as the representative of four GraphSAGE variants since GS-LSTM achieves the best performance among them and we only show the visualization results of GS-LSTM. As shown in Figure 3, for each method, the learned embeddings from the source graph and the target graph are plotted in two sub-figures separately (i.e., red points for the source graph and blue points for the target graph). Moreover, to compare the learned embeddings of the source graph and the target graph more intuitively, we plot these two sets of embeddings in a same sub-figure.

From Figure 3, we make the following observations: (1) For the baselines, the embedding distributions of the source graph and the target graph are very different (e.g., in Figure 3(c), many blue points and red points do not coincide with each other). (2) For GraphAE, the embedding distributions of the source graph and the target graph are similar (in Figure 3(r), the blue points and the red points are plotted in similar positions).

These observations can explain the superior performance of GraphAE over the baselines as follows:

- For the baselines, the embedding distribution bias still exists between the source graph and the target



(a) GS-LSTM on the source graph (b) GS-LSTM on the target graph (c) GS-LSTM on the two graphs (d) GCN on the source graph (e) GCN on the target graph (f) GCN on the two graphs

(g) GAT on the source graph (h) GAT on the target graph (i) GAT on the two graphs (j) GeniePath on the source graph (k) GeniePath on the target graph (l) GeniePath on the two graphs

(m) UDA-GCN on the source graph (n) UDA-GCN on the target graph (o) UDA-GCN on the two graphs (p) GraphAE on the source graph (q) GraphAE on the target graph (r) GraphAE on the two graphs

Fig. 3. The results of visualization. For each method, we plot three sub-figures (i.e., one for the source graph, one for the target graph, and one for the two graphs). The red points refer to the nodes in the source graph, and the blue points refer to the nodes in the target graph.

TABLE 5
The Results for Ablation Tasks (Gains Refers to the improvement over GraphAE-None).

AT	GraphAE-None	GraphAE-MRA	GraphAE-MAA	GraphAE
A1	40.91	45.22	46.21	**48.92**
A2	44.26	45.75	46.68	**52.49**
A3	37.67	43.81	46.41	**51.68**
A4	33.3	40.73	45.73	**47.34**
A5	38.85	42.53	45.01	**49.55**
A6	30.37	42.19	44.27	**49.60**
average	37.56	43.37	45.72	**49.93**
gains	NA	+5.81	+8.16	**+12.37**

graph. It indicates that the patterns learned from the source graph are not well used to infer the node embeddings of the target graph, which leads to poor node classification accuracy on the target graph.

- For GraphAE, the embedding distribution bias between the two graphs is almost eliminated by the proposed alignment strategies. It indicates that the patterns learned from the source graph can be well utilized to infer the node embeddings of the target graph. Thus, GraphAE has high node classification accuracy on the target graph.

7.6 Analysis of Ablation Tasks

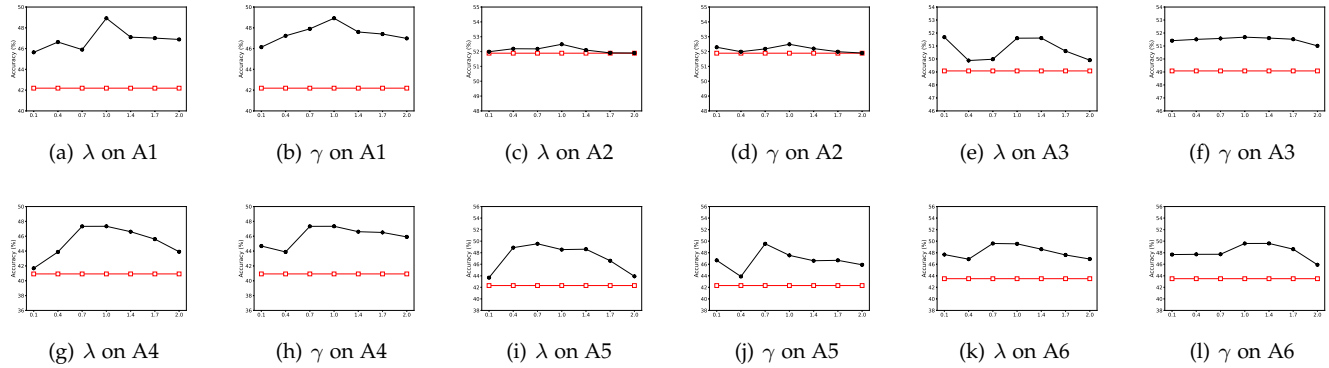
Although GraphAE shows the best performance against all baselines (analyzed in Sections 7.3 and 7.5), we still lack knowledge about the importance of each component in GraphAE (i.e., the question **Q3**).

In this subsection, to answer **Q3**, we investigate the contributions of the two alignment strategies to GraphAE by ablation study. Specifically, we design three variants of GraphAE: (1) **GraphAE-MRA**, which only adopts the alignment of MRA in GraphAE. (2) **GraphAE-MAA**, which only adopts the alignment of MAA in GraphAE. (3) **GraphAE-None**, which has no alignment in GraphAE.

The node classification accuracy of GraphAE and the three GraphAE variants are shown in Table 5 (Due to the limited space, we only show the results on Ali-CVR. A similar conclusion can be made on DBLP-Citation). We observe that compared with GraphAE-None, GraphAE-MRA, GraphAE-MAA, and GraphAE achieve the average gains of 5.81%, 8.16%, and 12.37%, respectively. It suggests that both alignment strategies (i.e., MRA and MAA) contribute to the alleviation of the data bias across graphs. Meanwhile, GraphAE-MAA performs better than GraphAE-MRA. One of the possible reasons is that MAA is a more direct alignment strategy to adjust embeddings. Moreover, the combination of MAA and MRA (i.e., GraphAE) achieves the best results most of the time, which further demonstrates the effectiveness of the alignment strategies.

7.7 Analysis of Incorporation Tasks

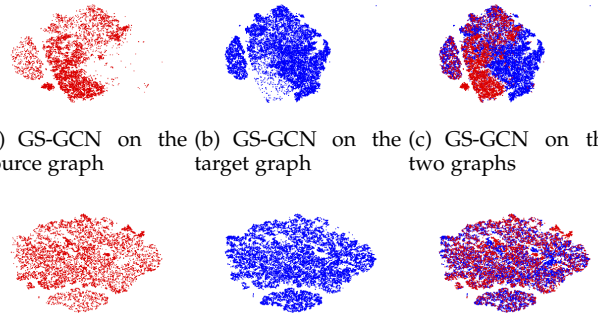
Actually, GraphAE is a general model to settle the graph bias problem. The idea of GraphAE can also be applied to existing GNN models. Based on our framework, existing GNNs can be incorporated with our alignment strategies. In this subsection, we analyze the improvement of GNNs by integrating our framework into them to address the adaptive task. Specifically, MAA can be incorporated into GCN and GraphSAGE. For GAT, both MAA and MRA can be adopted. The results of the baselines and their variants mentioned above are shown in Figure 6 (Note that "N" refers to the original baselines and "Y" refers to the corresponding variants incorporated with our alignment strategies). These results can support answering **Q4** (Due to the limited space, we only report the results on Ali-CVR. The results on DBLP-Citation are similar). We observe that the proposed framework significantly enhances the performance of all the baselines most of the time with an average gain of 5.63%.



(a) λ on A1 (b) γ on A1 (c) λ on A2 (d) γ on A2 (e) λ on A3 (f) γ on A3

(g) λ on A4 (h) γ on A4 (i) λ on A5 (j) γ on A5 (k) λ on A6 (l) γ on A6

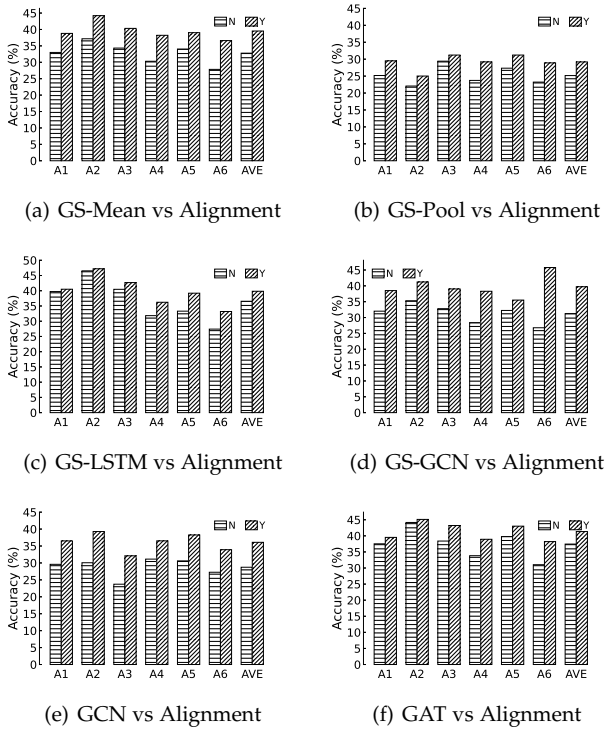
Fig. 4. Parameter sensitivity on Ali-CVR for λ and γ. The black line in each sub-figure refers to the results obtained by GraphAE, and the red line in each sub-figure refers to the best result among baselines.



(a) GS-GCN on the source graph (b) GS-GCN on the target graph (c) GS-GCN on the two graphs

(d) GS-GCN-Y on the source graph (e) GS-GCN-Y on the target graph (f) GS-GCN-Y on the two graphs

Fig. 5. The visualization of the incorporation task. The top three sub-figures show the visualization results of GS-GCN. The bottom three sub-figures show the visualization results of the corresponding variant incorporated with our alignment strategies (denoted as GS-GCN-Y).



(a) GS-Mean vs Alignment (b) GS-Pool vs Alignment

(c) GS-LSTM vs Alignment (d) GS-GCN vs Alignment

(e) GCN vs Alignment (f) GAT vs Alignment

Fig. 6. The results of the incorporation tasks. Note that "N" in each sub-figure refers to the original baselines, and "Y" refers to the corresponding variants incorporated with our alignment strategies. For simplification, we denote the average accuracy over the six adaptive tasks as "AVE".

To further show the effectiveness of the proposed alignment strategies and answer **Q4**, we again conduct the visualization experiments on the baselines (similar to the visualization experiments introduced in Section 7.5). Specifically, we first plot the learned embeddings by the original baselines into 2-D spaces (the top three sub-figures of Figure 5). Then, we incorporate the original baselines with the proposed alignment strategies and show their visualization results in the bottom three sub-figures of Figure 5. Due to the limited space, we only report the results obtained by GS-GCN and its corresponding variant (denoted as GS-GCN-Y) on the adaptive task A6 (Similar results can be obtained by other baselines on other adaptive tasks). From Figure 5, we observe that the embedding distributions of the source graph and the target graph learned by the original baseline are very different, which leads to poor classification accuracy (shown in Table 2 and Table 3). When the baseline is incorporated with our framework, the embedding distributions of the source graph and the target graph are similar, which leads to better classification accuracy (shown in Figure 6(d)).

7.8 Parameter Sensitivity

In this subsection, we investigate the influences of the parameters λ and γ to answer **Q5**. The former shows the effect of MAA, and the latter shows the effect of MRA. Specifically, when evaluating one of the penalty parameters of GraphAE, we fix another parameter to 1. Furthermore, we provide the best classification accuracy achieved by the baselines (plotted with the red lines). Figure 4 gives an illustration of the variation of performance as λ or $\gamma \in \{0.1, 0.4, 0.7, 1.0, 1.4, 1.7, 2.0\}$ for six adaptive tasks on Ali-CVR (The results on DBLP-Citation have similar trends).

From Figure 4, we can observe that although the performance of GraphAE varies with different values of λ or γ, GraphAE achieves higher accuracy than the best accuracy among baselines most of the time. Furthermore, when the value of λ or γ becomes too large, GraphAE always obtains lower accuracy. The reason is that too much penalty leads

to a large weight on the unsupervised loss, and thus weakens the contribution of the supervised loss, which directly influences the classification accuracy. The results indicate that a good trade-off between the supervised loss and the unsupervised loss can enhance the adaptive embedding learning across graphs and lead to good performances (e.g., λ=1.0 and γ=1.0 are proper choices in our experiments).

8 Conclusion

In this paper, we focus on learning adaptive node embeddings across graphs, which is called AT. We propose a unified framework for AT. Then, based on the proposed framework, GraphAE is designed to address this problem. By minimizing the discrepancy between the graph distribution of the source graph and that of the target graph, GraphAE can learn adaptive node embeddings. Furthermore, we extend the proposed model to a multi-graph version (MGraphAE). We apply the proposed models (i.e., GraphAE and MGraphAE) to two kinds of data sets, i.e., e-commerce data (Ali-CVR) and citation data (DBLP-Citation). We conduct twelve adaptive tasks and six m-adaptive tasks. Both GraphAE and MGraphAE achieve the best performance compared with all the state-of-the-art baselines. The exhaustive experimental results demonstrate the effectiveness of the proposed model. More kinds of structure distributions of graphs, e.g., subgraph distributions, will be considered in the future.

Acknowledgments

Chaokun Wang is supported in part by the National Natural Science Foundation of China (No. 61872207) and Baidu Inc. Fei He is supported in part by the National Natural Science Foundation of China (No. 62072267, No. 62021002). Philip S. Yu is supported in part by NSF under grants III-1763325, III-1909323, III-2106758, and SaTC-1930941.

References

[1] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *International conference on learning representations*, 2018.

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

[3] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st international conference on neural information processing systems*, 2017, p. 1025–1035.

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

[5] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

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

[7] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical review E*, vol. 78, no. 4, p. 046110, 2008.

[8] C. Wang, B. Wang, B. Huang, S. Song, and Z. Li, "Fastsgg: efficient social graph generation using a degree distribution generation model," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 564–575.

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

[10] X. Ma, J. Wang, H. Chen, and G. Song, "Improving graph neural networks with structural adaptive receptive fields," in *Proceedings of the Web Conference 2021*, 2021, pp. 2438–2447.

[11] M. Hay, C. Li, G. Miklau, and D. Jensen, "Accurate estimation of the degree distribution of private networks," in *2009 Ninth IEEE International Conference on Data Mining*, 2009, pp. 169–178.

[12] B. Nettasinghe and V. Krishnamurthy, ""what do your friends think?": Efficient polling methods for networks using friendship paradox," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 3, pp. 1291–1305, 2019.

[13] B. Rozemberczki and R. Sarkar, "Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, 2020, pp. 1325–1334.

[14] L. Wang, C. Huang, W. Ma, X. Cao, and S. Vosoughi, "Graph embedding via diffusion-wavelets-based node feature distribution characterization," in *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, 2021, pp. 3478–3482.

[15] X. Xu, Q. Li, L. Peng, T.-L. Hsia, C.-J. Huang, and J.-H. Wu, "The impact of informational incentives and social influence on consumer behavior during alibaba's online shopping carnival," *Computers in Human Behavior*, vol. 76, pp. 245–254, 2017.

[16] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," *arXiv preprint arXiv:1412.3474*, 2014.

[17] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 385–394.

[18] C. Shi, B. Hu, W. X. Zhao, and S. Y. Philip, "Heterogeneous information network embedding for recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 2, pp. 357–370, 2018.

[19] J. Kim, T. Kim, S. Kim, and C. D. Yoo, "Edge-labeling graph neural network for few-shot learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11–20.

[20] L. Xu, J. Cao, X. Wei, and S. Y. Philip, "Network embedding via coupled kernelized multi-dimensional array factorization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 12, pp. 2414–2425, 2019.

[21] H. Zhu, F. Feng, X. He, X. Wang, Y. Li, K. Zheng, and Y. Zhang, "Bilinear graph neural network with neighbor interactions," in *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, 2020.

[22] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

[23] Y. Weng, X. Chen, L. Chen, and L. Wei, "Gain: Graph attention & interaction network for inductive semi-supervised learning over large-scale graphs," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2020.

[24] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi, "Geniepath: Graph neural networks with adaptive receptive paths," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4424–4431.

[25] J. Lee, H. Kim, J. Lee, and S. Yoon, "Transfer learning for deep learning on graph-structured data," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[26] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, "Gcc: Graph contrastive coding for graph neural network pre-training," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020, pp. 1150–1160.

[27] Q. Dai, X. Shen, X.-M. Wu, and D. Wang, "Network transfer learning via adversarial domain adaptation with graph convolution," *arXiv preprint arXiv:1909.01541*, 2019.

[28] M. Wu, S. Pan, C. Zhou, X. Chang, and X. Zhu, "Unsupervised domain adaptive graph convolutional networks," in *Proceedings of the web conference 2020*, 2020, pp. 1457–1467.

[29] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[30] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 249–270, 2022.
[31] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.
[32] Z. Cai, Z. Xiong, H. Xu, P. Wang, W. Li, and Y. Pan, "Generative adversarial networks: A survey toward private and secure applications," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–38, 2021.
[33] K. Li, G. Luo, Y. Ye, W. Li, S. Ji, and Z. Cai, "Adversarial privacy-preserving graph embedding against inference attack," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6904–6915, 2020.
[34] K. Li, G. Lu, G. Luo, and Z. Cai, "Seed-free graph deanonymiztiation with adversarial learning," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, 2020, pp. 745–754.
[35] M. Long, Z. Cao, J. Wang, and M. I. Jordan, "Conditional adversarial domain adaptation," in *Proceedings of the 32nd international conference on neural information processing systems*, 2018, p. 1647–1657.
[36] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *International conference on machine learning*. PMLR, 2015, pp. 97–105.
[37] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proceedings of the 30th international conference on neural information processing systems*, 2016, pp. 3844–3852.
[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st international conference on neural information processing systems*, 2017, p. 6000–6010.
[39] A. Gretton, B. Sriperumbudur, D. Sejdinovic, H. Strathmann, S. Balakrishnan, M. Pontil, and K. Fukumizu, "Optimal kernel choice for large-scale two-sample tests," in *Proceedings of the 25th international conference on neural information processing systems*, 2012, pp. 1205–1213.
[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[41] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

Gaoyang Guo received the BEng degree in software engineering from the Harbin Institute of Technology, China, in 2016. He is currently working toward the PhD degree in software engineering in the School of Software, Tsinghua University. His major research interests include social networks and graph neural networks. learning.

Chaokun Wang (Member, IEEE) received the PhD degree in computer software and theory from the Harbin Institute of Technology, China, in 2005, and is an associate professor in the School of Software at Tsinghua University currently. He has published more than 100 refereed papers, received four best paper awards at international conferences and owned 21 patents. His current research interests include social network analysis, graph data management, and big data systems.

Bencheng Yan received the BEng degree in software engineering from Northeastern University, and the MSc degree in software engineering from Tsinghua University, in 2017 and 2020, respectively. His major research interests include graph neural networks and recommendation systems.

Yunkai Lou received the BEng degree in software engineering from Tsinghua University, in 2018. He is currently working toward the PhD degree in the School of Software, Tsinghua University. His major research interests include graph algorithms, graph query languages, and graph database systems.

Hao Feng received the BEng degree in software engineering from Tsinghua University, in 2020. He is currently working toward the PhD degree in the School of Software, Tsinghua University. His major research interests include social networks and graph neural networks.

Junchao Zhu received the BEng degree in software engineering from Tsinghua University, in 2017. He is now working toward the PhD degree in the School of Software, Tsinghua University. His major research interests include social network analysis and data mining in graphs.

Jun Chen is currently a research scientist and the algorithm lead in Intelligent Healthcare Unit, Baidu Inc. He received his Ph.D. degree and B.S. degree in software engineering both from Tsinghua University, Beijing, China. He has authored research papers in major conferences and journals including IEEE TIP, IEEE TKDE, ACL, AAAI, IJCAI, VLDB, ACM Multimedia, KAIS and spj Health Data Science. His research interests include machine learning, natural language processing, recommender system and deep learning for healthcare.

Fei He received the B.S. degree in computer science and technology from National University of Defense Technology in 2002, and the Ph.D. degree in computer science and technology from Tsinghua University in 2008. He is currently an Associate Professor in the School of Software at Tsinghua University, Beijing, China. His research interests include formal verification and program analysis.

Philip S. Yu (Fellow, IEEE) received the BS degree in electrical engineering from National Taiwan University, Taipei, Taiwan, the MS and PhD degrees in electrical engineering from Stanford University, Stanford, California, and the MBA degree from New York University, New York. He is currently a Distinguished Professor in the Department of Computer Science at the University of Illinois Chicago, Chicago, IL, USA, where he also holds the Wexler Chair in Information and Technology. His current research interests include big data, data mining, social network, privacy preserving data publishing, data streams, database systems, and Internet applications and technologies. He is a fellow of ACM.