

Bipartite Graph Neural Networks for Efficient Node Representation Learning

Chaoyang He^{*1}, Tian Xie^{*1}, Yu Rong², Wenbing Huang²,
Yanfang Li¹, Junzhou Huang², Xiang Ren¹, Cyrus Shahabi¹

¹ University of Southern California, ² Tencent AI Lab

{chaoyang.he, xiet, yangfang.li, xiangren, shahabi}@usc.edu, yu.rong@hotmail.com, hwenbing@126.com, jzhuang@uta.edu

Abstract

Existing Graph Neural Networks (GNNs) mainly focus on general structures, while the specific architecture on bipartite graphs—a crucial practical data form that consists of two distinct domains of nodes—is seldom studied. In this paper, we propose Bipartite Graph Neural Network (BGNN), a novel model that is domain-consistent, unsupervised, and efficient. At its core, BGNN utilizes the proposed Inter-domain Message Passing (IDMP) for message aggregation and Intra-domain Alignment (IDA) towards information fusion over domains, both of which are trained without requiring any supervision. Moreover, we formulate a multi-layer BGNN in a cascaded manner to enable multi-hop relation modeling while enjoying promising efficiency in training. Extensive experiments on several datasets of varying scales verify the effectiveness of BGNN compared to other counterparts. Particularly for the experiment on a large-scale bipartite graph dataset, the scalability of our BGNN is validated in terms of fast training speed and low memory cost.

1 Introduction

Graphs that characterize relations among data arise in various domains including drug discovery (You et al., 2018; Jin, Barzilay, and Jaakkola, 2018), social networks analysis (Wang, Cui, and Zhu, 2016; Qiu et al., 2018), and visual understanding (Yang et al., 2018; Wan et al., 2018) to name a few. Amongst their varying forms, bipartite graphs belong to one type of the most central structures. A bipartite graph (depicted in Figure 1) is a graph whose vertices are divided into two independent components such that every edge connects nodes from one component to the other. This specific type of structures is demanded to explore in many real applications—taking the E-commerce recommendation system as an example, users and products correspond to two distinct components, and how to take benefit of the buying correlations between them plays an important role in accurate and effective recommendation services (Linden, Smith, and York, 2003).

Early works on graphical structure data can be dated back to the research of graph embedding

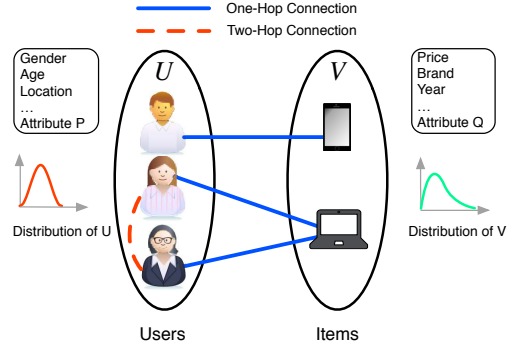


Figure 1: A Bipartite Graph in the E-commerce system.

(Perozzi, Al-Rfou, and Skiena, 2014; Grover and Leskovec, 2016), where graph topology and node relations are embedded as vector space. In light of the rapid advances of deep learning (LeCun, Bengio, and Hinton, 2015), current research attention has been paid to the application of deep neural networks on graphs, giving rise to a novel and also powerful kind of models, dubbed as Graph Neural Networks (GNNs) (Gori, Monfardini, and Scarselli, 2005; Scarselli et al., 2009). To their essential characteristics, GNNs recursively update each node’s feature through aggregation (or message passing) of its neighbors, by which the patterns of graph topology and node features are both captured. While GNNs have exhibited tremendous progress, applying them to the particular vein, *i.e.* bipartite graphs, is never studied previously and requires delicate developments. The main challenge is that the features in two groups of bipartite graphs follow different distributions with different dimensions (for example, in Figure 1, users and items have quite distinct characteristics). Therefore, it is insufficient to depict the consistency and correlation between two domains if we pack them as a global graph and leverage neighbor message passing as usual GNN does. One can extract two different sub-graphs for each domain by regarding two-hop neighbors as one-hop homogeneous connections within the same domain (note that each node and its two-hop neighbors are in the same group). Obviously, such a separation scheme is unable to conduct information

^{*}Equal Contribution

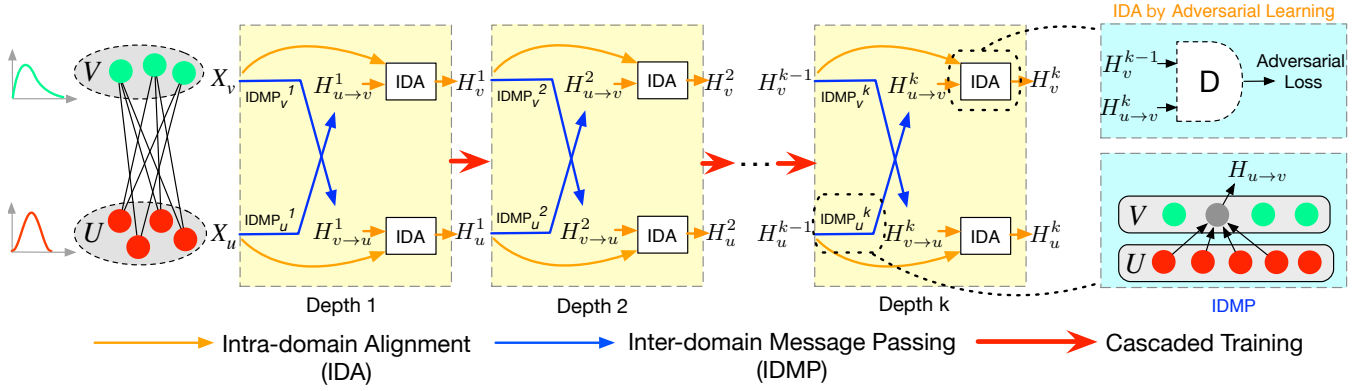


Figure 2: Illustration of BGNN. Given the inputs of two domains X_u and X_v , we obtain their unsupervised-embedded representations as H_u and H_v via inter-domain message passing and inter-domain distribution alignment. To enable multi-hop neighbor information aggregation, we stack multiple layers to formulate a deep BGNN whose layers are trained in a cascaded manner.

fusion across domains.

In this paper, we propose the Bipartite Graph Neural Network (BGNN) that is domain-consistent, unsupervised, and efficient. BGNN consists of two central operations: *inter-domain message passing* and *intra-domain alignment*, as illustrated in Figure 2. Suppose the two domains to be U and V , respectively. In each layer (depth) of BGNN, we formulate two kinds of information flows, one from U to V and the other from V to U , each of which is equipped with different weight filter. In this way, we attain inter-domain representation for each node. To enable domain fusion, one may concatenate the input feature of each node with its corresponding inter-domain representation as an enhanced output. Nevertheless, such a simple concatenation requires supervised signals for the training, which is not allowable in practice. Instead, we propose an intra-domain alignment trick to minimize the divergence between input features and inter-domain representations, by using adversarial models (Goodfellow et al., 2014; Pan et al., 2018), a kind of tool that has been applied successfully for distribution matching. In contrast to feature concatenation, our alignment trick is unsupervised and more practical for broader cases.

Another benefit of the intra-domain alignment loss is that it allows for *cascaded training*. By saying cascaded training, we mean the training of the upper layer (depth $k+1$ in Figure 2) begins only after the lower one (depth k in Figure 2) has already been trained. This cascaded training is possible for BGNN since we can utilize the intra-domain alignment loss for unsupervised training layer by layer. Clearly, the cascaded training is more memory-efficient than the conventional end-to-end training paradigm, because it does not need to restore all intermediate activation maps of all neural layers. Besides, by cascaded training, the domain shift (*i.e.* the discrepancy between two domain features, which always exists during the early training phase) in lower layers will not be passed to higher ones; while in end-to-end training this kind of errors will accumulate as the depth increases.

We contrast the performance of our algorithm against several unsupervised graph learning counter-parts: Node2Vec (Grover and Leskovec, 2016), VGAE (Kipf and Welling, 2016), Graph-

SAGE (Hamilton, Ying, and Leskovec, 2017), and AS-GCN (Huang et al., 2018). For the evaluation, we apply a large-scale social network from Tencent Platform, and also construct three synthesized datasets based on the citation networks Cora, Citeseer, and PubMed (Sen et al., 2008). Upon all benchmarks, our method exhibits more expressive representation learning ability, higher classification accuracy, faster training speed, and lower memory cost than all compared models.

To sum up, our contributions are listed as follows.

- To the best of our knowledge, we are the first to apply graph neural networks (GNNs) for bipartite graph representation learning.
- We propose a novel model termed as bipartite graph neural network (BGNN), which is expressive, unsupervised and resource-economic.
- We experimentally evaluate the effectiveness of our method against state of the arts on several public benchmarks as well as one large-scale bipartite graph dataset.

2 Related work

Graph representation learning (also termed as graph embedding or network embedding) is closely related to our work. It can be generally classified into two groups: random walks-based methods, and graph convolutional networks (GCN)-based methods. DeepWalk (Perozzi, Al-Rfou, and Skiena, 2014) and Node2vec (Grover and Leskovec, 2016) are representative random walk-based methods to model homogeneous graphs. Some follow-up works embed 1st-order, 2nd-order and even high-order proximities between vertices on homogeneous graphs, such as LINE (Tang et al., 2015), GraRep (Cao, Lu, and Xu, 2015), SDNE (Wang, Cui, and Zhu, 2016), DVNE (Zhu et al., 2018). However, these methods are developed for embedding homogeneous graph. Thus, when applying them to bipartite graph embedding, the distinct vertex feature information is ignored. Although we can view the bipartite graph as a special type of heterogeneous network and then apply heterogeneous graph embedding methods, they are still suboptimal. For example, Metapath2vec++

(Dong, Chawla, and Swami, 2017) ignores the strength of the relations between vertices and treats the explicit and implicit relations as equally. While BiNE (Gao et al.) is a random walk-based method customized for bipartite graph embedding, it lacks evaluation on comparison with GCN-based methods, and more significantly, its drawback on extending to large-scale graphs largely hinders its application in practice. To handle the complexity of graph data, Graph Convolutional Networks (GCN)-based methods have a rapid development recently. Among them, GCN (Kipf and Welling, 2017), VGAE (Kipf and Welling, 2016), GraphSAGE (Hamilton, Ying, and Leskovec, 2017) show state-of-the-art performance in many graph representation learning applications. In general, these methods perform a convolution by aggregating the neighbor nodes' information so that each node can learn a relationship between the nodes in the entire graph. However, the weakness for GCN-based methods is that same node attributes should be assumed. Our work is also relevant to those GCN-based methods that deal with the scalability issue on large-scale graphs, like the node-wise sampling method GraphSAGE (Hamilton, Ying, and Leskovec, 2017) and the layer-wise sampling method AS-GCN (Huang et al., 2018). Differ from these methods, our BGNN has better scale performance by proposing a cascaded training architecture.

3 Proposed model: BGNN

This section introduces Bipartite Graph Neural Networks (BGNN), a general extension of GNN to bipartite graphs. We propose two novel techniques for unsupervised domain information fusion, including inter-domain message passing and intra-domain alignment. We finally present the efficient version of BGNN to enable large-scale graph learning. The overall implementation framework is also summarized.

Bipartite Graphs Let $G = (U, V, E)$ be a bipartite network, where U and V denote the set of the two domains of vertices (nodes). u_i and v_j denote the i -th and j -th vertex in U and V , respectively, where $i = 1, 2, \dots, M$ and $j = 1, 2, \dots, N$. There are only inter-domain edges which are defined as $E \subseteq U \times V$. e_{ij} represents the edge between u_i and v_j . The adjacent matrix for set U is $B_u \in \mathbb{R}^{M \times N}$ and $B_v \in \mathbb{R}^{N \times M}$ for set V . $B_{u(i,j)} = 1$ if $e_{ij} \in E$, and $B_{u(i,j)} = 0$ if $e_{ij} \notin E$. The features of two sets of nodes can be formulated as X_u and X_v , respectively, where $X_u \in \mathbb{R}^{M \times P}$ is a feature matrix with $x_{u(i)} \in \mathbb{R}^P$ representing the feature vector of node u_i , and $X_v \in \mathbb{R}^{N \times Q}$ is similarly defined.

Overall Framework Here we introduce the overall framework of our model for better readability. In general, *bipartite graph representation learning* is to learn the embedding representations $H_u \in \mathbb{R}^{P'}$ and $H_v \in \mathbb{R}^{Q'}$ for nodes in group U and V , respectively. Let f_{emb} be a general bipartite graph embedding model with parameters θ , then the representation of H_u and H_v is defined as follow:

$$H_u, H_v = f_{emb}(X_u, B_u, X_v, B_v; \theta) \quad (1)$$

GNN-based methods (Kipf and Welling, 2017; Hamilton, Ying, and Leskovec, 2017) define neural propagation for representation learning on general graphs. When customized to bipartite graphs, the intra-domain connections are absent, so Eq. (1) is divided into two terms as

$$H_{v \rightarrow u} = f_u(X_v, B_u) \quad (2)$$

$$H_{u \rightarrow v} = f_v(X_u, B_v) \quad (3)$$

In this paper, we term this inter-domain message passing (IDMP) which follows the blue color information flow in Figure 2. More details are provided in § 3.1.

To encourage domain information fusion, we further propose an intra-domain alignment (IDA, the orange color information flow in Figure 2) to enhance the inter-domain representations of Eq. (2)-(3). We arrive at

$$Loss_u = L_{adv}(H_{v \rightarrow u}, X_u) \quad (4)$$

$$Loss_v = L_{adv}(H_{u \rightarrow v}, X_v) \quad (5)$$

where L_{adv} is specified as an adversarial loss. We provide more explanations in § 3.2. After the training by minimizing Eq. (4)-(5), we obtain the representations as H_u and H_v for two domains. This alignment, from the other perspective of understanding, is able to combine two distinct distributions in an unsupervised manner. To be specific, the representation $H_{v \rightarrow u}$ is a function of X_v , and it has also been driven towards the distribution of X_u by minimizing the alignment loss, hence the aligned H_u contains information from both X_v and X_u .

The embedding H_u (resp. H_v) merely captures one-hop topology structure of B_u (resp. B_v) as well as feature information from X_u and X_v . As presented in previous works (Kipf and Welling, 2017; Hamilton, Ying, and Leskovec, 2017), the one-hop aggregation is insufficient to characterize diverse graph structures; hence a multi-hop mechanism (or equivalently a deep network) is in demand. Other than leveraging typical end-to-end networks of multiple layers, this paper develops a cascaded architecture to drive multi-hop message passing. We will detail how the cascaded framework is formulated in § 3.3.

3.1 Inter-Domain Message Passing (IDMP)

Formally, the adjacent matrix of bipartite graph is

$$A = \begin{pmatrix} 0_{u,u} & B_u \\ B_v & 0_{v,v} \end{pmatrix} \quad (6)$$

For stability, we normalize B_u as $\hat{B}_u = I + D_u^{-\frac{1}{2}} B_u D_u^{-\frac{1}{2}}$, where D_u is the degree matrix of B_u . Similar normalization is done for B_v . The IDMP process is defined as

$$\begin{cases} H_{v \rightarrow u}^{(k)} = \sigma(\hat{B}_u H_v^{(k)} W_u^{(k)}) \\ H_{u \rightarrow v}^{(k)} = \sigma(\hat{B}_v H_u^{(k)} W_v^{(k)}) \end{cases} \quad (7)$$

where as mentioned in Eq. (2)-(3), $H_{v \rightarrow u}^k \in \mathbb{R}^{M \times Q'}$ (resp. $H_{u \rightarrow v}^{(k)} \in \mathbb{R}^{N \times P'}$) are hidden features of the nodes in set

U (resp. V) aggregated from the features in V (resp. U), k indicates the depth index (note that when $k = 0$, $H_u^{(0)} = X_u$, $H_v^{(0)} = X_v$ are actually input features).

As we can see from Eq.7, there are two important characteristics that differ IDMP from conventional GCNs: 1. IDMP only performs aggregation on each node’s neighbor nodes without involving the node itself, while conventional GCN methods usually consider the self-loop computation; 2. The propagation is only one-hop-neighbor aware. The first characteristic motivates us to further design intra-domain alignment to take the self-input features into account, while the second one leads to our design on the cascaded training approach which enables multi-hop modeling and supports efficient training.

3.2 Intra-Domain Alignment (IDA)

We introduce IDA from the perspective of domain U . As mentioned previously in Eq. (4), we design two types of alignment losses to align $H_{v \rightarrow u}$ to H_u . Our first alignment is to employ the work (Goodfellow et al., 2014; Ganin et al., 2015) to the graph representation learning domain. A *discriminator* is trained to discriminate between vectors randomly sampled from $H_{v \rightarrow u}$ and H_u . In another way, IDMP (Inter-Domain Message Passing) is trained as *generator* to prevent the discriminator from making accurate predictions. As a result, this is a two-player min-max game, where the discriminator aims to maximize the ability to identify two distinct feature representations, and IDMP aims to prevent the discriminator from doing so by aligning the encoded representations $H_{v \rightarrow u}$ (source) to H_u (target). After training, they will reach a Nash equilibrium so that the output of IDMP can embed information from two distinct feature spaces (learned representation H_u).

Discriminator objective We define the parameters of the IDA discriminator as θ and the IDMP generator as ϕ . We denote $P_{\theta, \phi}(\text{source} = 1|z)$ as the probability that input vector z is from the source domain $H_{v \rightarrow u}$. As opposed to this, $\text{source} = 0$ means z is from the target domain H_u . The discriminator loss function is as follows:

$$L_D(\theta|\phi) = - \sum_{i=1}^M \log P_{\theta, \phi}(\text{source} = 0|h_{u(i)}) - \sum_{i=1}^N \log P_{\theta, \phi}(\text{source} = 1|h_{v \rightarrow u(i)}) \quad (8)$$

Generator objective In generative setting, IDMP is trained so that the discriminator is unable to distinguish between the intra-domain features:

$$L_G(\phi|\theta) = - \sum_{i=1}^N \log P_{\theta, \phi}(\text{source} = 0|h_{v \rightarrow u(i)}) \quad (9)$$

During training, for every input sample, the discriminator and the generator are trained successively with gradient updates to optimize two networks, respectively. We call our model as BGNN-Adv when using this approach as IDA.

Another intuitive approach to do intra-domain alignment is to project the IDMP output and features to the same dimensions with a multi-layer perceptron (MLP). We term our

model as BGNN-MLP when using MLP as IDA. This approach is relatively straight-forward, but we can compare it with BGNN-Adv to judge whether adversarial learning could align domain effectively. In our experiment section, we regard BGNN-MLP as a baseline method for comparison. Formally, we define the loss function for one set U as

$$\mathcal{L}_u = \|\text{MLP}(H_{v \rightarrow u}^{(i)}) - H_u^{(i-1)}\|_F \quad (10)$$

which is symmetric for set V . The multi-layer perceptron takes the IDMP output as its input and minimizes with original features in a Frobenius norm. In the experiment section, we show that on the downstream classification task, even this simple loss can gain improvement upon conventional GCN-based methods. BGNN-Adv is even better than BGNN-MLP in terms of model accuracy, which empirically proves that the adversarial approach can embed more information to boost graph embedding performance.

3.3 Cascaded Architecture: Towards Efficient BGNN

In this section, we present the cascaded architecture design for our proposed BGNN model. In Figure 3(a), we depict a detailed diagram to illustrate our cascaded training process to compare with the conventional end-to-end training paradigm. We regard the previous IDMP with IDA as one depth. Each depth (layer) is trained one-hop embedding in E epochs at a time. Then its final learned embedding is used as the input for the later depth training. This is in contrast to the conventional end-to-end training paradigm which propagates through multiple depths for a fixed number of hops to train the final embedding (Shown in Figure 3(b)).

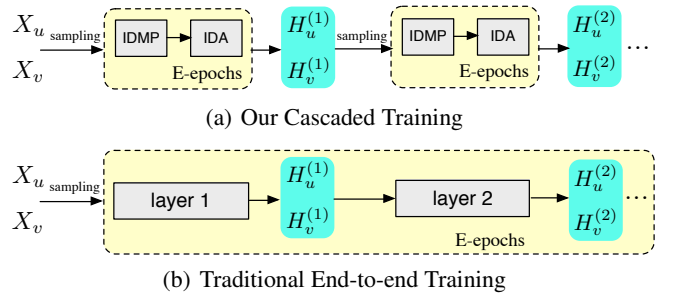


Figure 3: Comparison of cascaded training with traditional end-to-end training.

This cascaded architecture can embed multi-hops information for bipartite graph like GCN-based methods performing in general graphs. In addition to this, system-wise advantages further strengthen our design choice. In general, cascaded training is more memory-efficient and also costs less training time. This is attributed to our four findings confirmed by experiments:

Only one depth training is alive. In Figure 3(a), each depth in cascaded architecture takes the final embedding from the previous depth as input. This indicates we can destroy model instance (release unused memory) in previous depth and only keep one depth training alive in the entire

training process. One may argue that multi-hop topology information can also be preserved by end-to-end multi-depth training paradigm, but it significantly increases the memory cost and training time on large-scale bipartite graphs.

No uncontrollable neighborhood expansion. On a large-scale graph, the neighborhood expansion of each node layer by layer will quickly demand a high memory cost and lead to low computation speed. To deal with this problem, node-wise sampling like GraphSAGE (Hamilton, Ying, and Leskovec, 2017) directly samples neighbor nodes, while layer-wise sampling method like AS-GCN (Huang et al., 2018) uses adaptive sampling to fix the number of nodes in each layer further. Compared to these methods, since the propagation of our architecture only happens one-hop, mini-batch sampling can be applied without neighborhood expansion, which speeds up the training and reduces memory cost.

Faster convergence speed. We also found that normally our model requires only three epochs on a large-scale dataset to achieve the best result. Some sampling methods, like the layer-wise sampling AS-GCN method, can speed up the training time for each epoch, but it requires much more training epochs than our proposed architecture.

More robust in hyper-parameter tuning. Our cascaded architecture is robust so that it can be easily trained without too much hyper-parameter tuning effort. By cascaded training, the domain shift (i.e., the discrepancy between two domain features, which always exists during the early training phase) in lower depths will not be passed to higher ones; while in end-to-end training this kind of error will accumulate as the depth increases.

In our experiment, we compared the training time and memory cost with other baselines. It proves our design choice for efficient training.

3.4 Algorithm

We summarize our overall implementation framework for BGNN model in Algorithm 1 which is consistent to Figure 3(a). The processes for set U and V are symmetric. Each step in the outmost loop proceeds as follows, where k represents the current layer and $H_u^{(k)}, H_v^{(k)}$ are hidden representations in layer k . For every epoch, sampling is conducted on these hidden representations to get mini-batch as input. After several epochs of training, the embedding representations of depth k can be learned and saved for $k+1$ layer training. The k th layer model instance and unused memory are released. The finally representations can be extracted in the last layer K , which then can be used for the downstream graph analytic tasks. The time complexity per epoch for BGNN is fixed at $O(|E|)$ ($|E|$ denotes the number of edges), where there is no neighborhood expansion along with layer (depth) in traditional end-to-end training.

4 Experiments

We design our experiments with the goals of (i) providing a rigorous comparison of the graph representation performance between our BGNN model and the state-of-art baselines, (ii) verifying the effectiveness of the cascaded archi-

Algorithm 1: BGNN algorithm

Input: Graph $G(U, V, E)$; input features $\{X_u, X_v\}$

Output: Node representations Z_u and Z_v

$H_u^0 \leftarrow X_u; H_v^0 \leftarrow X_v$

for $k = 1, \dots, K$ **do**

for e in epochs **do**

 Sampling batches $(h_u^{(k)}, h_v^{(k)})$ from

$(H_u^{(k)}, H_v^{(k)})$

for $h_u^{(k)}, h_v^{(k)}$ as batches of input **do**

$h_u^{(k+1)} \leftarrow \text{IDA}^{(k)}(h_u^{(k)}, \underbrace{\text{IDMP}^{(k)}(h_v^{(k)})}_{h_{v \rightarrow u}^{(k)}});$

$h_v^{(k+1)} \leftarrow \text{IDA}^{(k)}(h_v^{(k)}, \underbrace{\text{IDMP}^{(k)}(h_u^{(k)})}_{h_{u \rightarrow v}^{(k)}});$

end

end

 Save($H_u^{(k+1)}, H_v^{(k+1)}$) for $(k+1)$ th depth training

 Release($\text{IDMP}^{(k)}, \text{IDA}^{(k)}, H_u^k, H_v^k$)

end

$Z_u \leftarrow H_u^K; Z_v \leftarrow H_v^K$

ture, (iii) evaluating the BGNN efficiency of space and time complexity on a large-scale dataset.

Dataset		Tencent	Cora	Citeseer	PubMed
#Edges		991,734	1,802	1,000	18,782
#Nodes	U	619,030	734	613	13,424
	V	90,044	877	510	3,435
#Features	U	8	1,433	3,703	400
	V	16	1,000	3,000	500
#Classes	U	2	7	6	3
	V	N/A	6	6	3

Table 1: Dataset statistics

Dataset The statistics and distributions of our datasets are summarized in Table 1 and appendix. **Tencent**—is a large-scale real world social network represented by a bipartite graph. Nodes in set U are social network users and nodes in set V are social communities (E.g., some social network users who share the same interest in electrical products may join the same shopping community). Both users and communities are described by dense off-the-shelf feature vectors. The edge connection between two sets indicates that the user belongs to the community. Another type of dataset is the *synthetic* bipartite graphs dataset generated from citation network datasets: **Cora**, **Citeseer** and **PubMed** (Sen et al., 2008). Documents and citation links between them are treated as nodes and undirected edges, respectively. We summarize our synthetic method here: for Cora and Citeseer, we randomly divide the nodes into two sets and remove all the edge connections between nodes which belong to the same set. The node features in two sets are also chosen to be different. As for PubMed, we manu-

ally select nodes with a low degree to construct a sparse bipartite graph. These four datasets cover graphs from small, medium to large-scale size, and from the long tail (Tencent, Cora, Citeseer) to balanced degree distribution (PubMed), which plays a comprehensive role in evaluating model effectiveness.

4.1 Experimental Settings

For our adversarial learning BGNN, we use hyperbolic function as the non-linear activation function in the graph convolution network. The dropout and L2 regularization are applied to each layer to prevent overfitting. During training, we use mini-batch to reduce the memory and computation cost for large-scale data set as discussed before. We found the optimal batch size for all four data set is near 500, and it only requires around 3 epochs on each data set to converge to the best result quickly. More details about parameter settings for each dataset are shown in the appendix.

Baselines We mainly compare our BGNN algorithm against four unsupervised node embedding baselines:

- **Raw features:** This indicates a naive classification model that learns from nodes’ raw features, without using any graph structure information incorporated.
- **Node2Vec** (Grover and Leskovec, 2016): This approach is an extension of Word2Vec (Mikolov et al., 2013) on graph, which learns a feature representations by simulating a biased random walks on the graph. We run Node2Vec on the bipartite graph and then concatenate the node embeddings with their own features.
- **VGAE** (Kipf and Welling, 2016): This method is based on variational auto-encoder, where GCN is used as an encoder and a simple inner product as a decoder to embed the nodes into low-dimensional feature space.
- **GraphSAGE-MEAN, GraphSAGE-GCN** (Hamilton, Ying, and Leskovec, 2017; Kipf and Welling, 2017): We implement two types of aggregator functions: GCN and MEAN aggregator. Node-wise sampling is used to address the scalability issue.
- **BGNN-MLP:** We also implement the straight-forward MLP to do intra-domain alignment as one baseline.

Due to the inconsistency of nodes feature dimensions in two bipartite sets, direct applying GCN is impractical. To make the comparison available, we reconstruct the bipartite graph into two subgraphs, where each of them only contains nodes from one set with their two-hops connection through the opposite set, as shown in Figure 1. Through this conversion, GCN based methods can be implemented on two sets, each with the same feature dimensions respectively, but still contains the original connectivity information in bipartite graphs. For each baseline model, we follow the open-source implementation from the authors’ original paper. In order to provide a fair comparison, we also tune the hyper-parameters for every baseline and report the best results among them.

Experiments are conducted on a GPU server with 8 Tesla V100 cards. We describe more details about our infrastructure in the appendix.

4.2 Results

To get a faster training process, instead of training two sets simultaneously in one layer, we alternatively select one set in each layer to train the model. During each layer training, we wait until the loss functions converge to continue to the next layer step. The adversarial training loss function

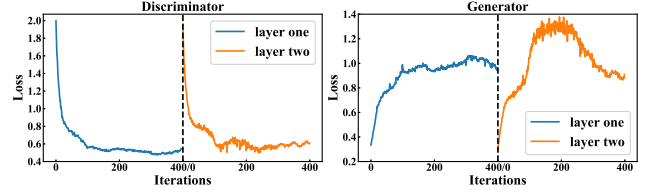


Figure 4: Adversarial training loss of cascaded architecture. The x axis denotes iteration numbers in each layer.

of BGNN is shown in Figure 4. We only plot the training losses of layer one and two. Clearly, the generator and discriminator losses show that the output of IDMP is doing an intra-domain alignment in an adversarial way.

We further evaluate our BGNN results on a classification downstream task. For binary classification on the Tencent dataset, we report F_1 scores. For other multi-classification tasks, we use both micro and macro-averaged F_1 scores.

Performance comparison From Table 2 we can see BGNN achieves the best performance on bipartite graph representation learning. BGNN-Adv surpasses other methods on both large and small data sets, which suggests the effectiveness of BGNN to capture both inter-domain and intra-domain information. Although BGNN-MLP does not work better than BGNN-Adv, it still outperforms the baselines. Particularly on the Tencent large-scale bipartite graph, BGNN-Adv achieves a 29% gain over the raw feature baseline. In the PubMed dataset, due to the balanced degree distribution, even incorporate the graph structure information into the model can only improve a small fraction. However, BGNN still gets the best result among other baselines, which verifies that BGNN also has better compatibility to embed more information on the long-tail dataset and balanced dataset.

4.3 Effect of Layer Numbers

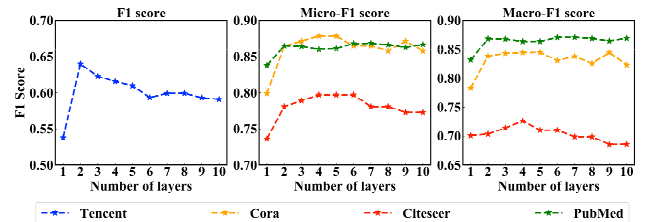


Figure 5: Influence of training depth (number of layers) on downstream classification task. The x axis denotes the number of BGNN layers and the y axis is the F_1 -score .

	Tencent	Cora		Citeseer		PubMed	
Methods	F_1	Micro F_1	Macro F_1	Micro F_1	Macro F_1	Micro F_1	Macro F_1
Raw features	0.497	0.789	0.758	0.707	0.621	0.838	0.843
Node2Vec	0.610	0.810	0.780	0.724	0.627	0.834	0.839
VGAE	N/A	0.782	0.754	0.732	0.645	0.823	0.828
GraphSAGE-GCN	0.529	0.782	0.763	0.715	0.627	0.838	0.843
GraphSAGE-MEAN	0.580	0.823	0.801	0.748	0.665	0.838	0.843
BGNN-MLP	0.582 ± 0.001	0.802 ± 0.008	0.784 ± 0.775	0.756 ± 0.008	0.662 ± 0.014	0.846 ± 0.001	0.842 ± 0.001
BGNN-Adv	0.622 ± 0.017	0.859 ± 0.005	0.831 ± 0.007	0.768 ± 0.004	0.698 ± 0.005	0.857 ± 0.005	0.860 ± 0.005
% gain over feat.	25%	9%	9%	9%	12%	2%	2%

Table 2: Prediction results for the four datasets (F_1 scores). Results for BGNN unsupervised nodes embedding are shown.

We also evaluate the performance with different number of layers in BGNN on four datasets. Figure 5 shows the results of F_1 score of downstream classification task along with the increase of number of layers. We observe the following phenomena:

- Particularly, one layer BGNN means that there is no cascaded architecture during learning; only one-time optimization is performed. Clearly, without cascaded architecture, the one layer BGNN has relatively lower performance on all datasets. Besides, one layer BGNN only embeds one-hop neighbor connection information in the final embeddings.
- BGNN can achieve better performance by increasing the number of layers. This benefit is due to the highly contrasting average degree in two sets of nodes, which cause a large number of two-hop connections for particular sparse connectivity set. However, one layer BGNN will completely ignore this relation.
- With more than two layers, the improvement is not as significant as the first two layers. The reason is in the bipartite graph setting, nodes in two sets normally represent entirely different entities (e.g., user and community). So multi-hop connections (from the opposite set) of one node has less correlation between each other. As we see, the accuracy will sometimes slightly drop when the network becomes deeper. Perhaps the use of trained parameters from previous layers will solve this problem to a certain extent. We remain this as our future exploration.

4.4 Towards Large-scale Bipartite Graphs

In order to compare scalability of BGNN against the baselines, we measure the training time and memory cost of algorithms running on Tencent large-scale dataset. We also add adaptive layer-wise sampling GCN (AS-GCN) into comparison, a method which efficiently solves the large-scale graph problem for GCN. Clearly, in Figure 6, BGNN greatly outperforms others in terms of both space and time requirements. This final result is due to our experimental observations beforehand: (1) BGNN does not need to load the entire graph into memory; only one mini-batch is needed. However, all other methods first require to fill the graph into memory, which is showing as the unstable increase of memory cost at the training start; (2) BGNN does not require sampling procedure in each graph convolutional layer,

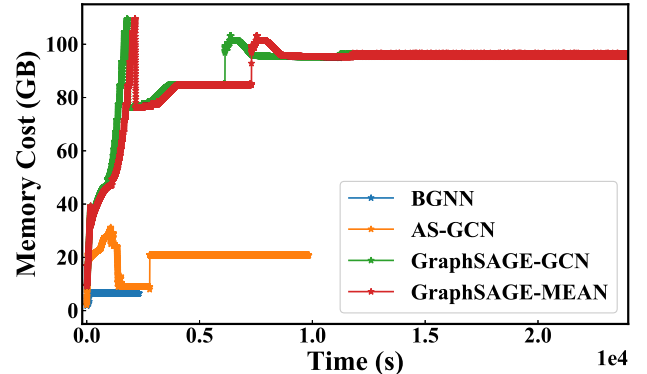


Figure 6: Memory cost and training time on Tencent data with related best parameters. The x axis denotes the wall-clock time in second, whereas the y axis is the memory cost. The short blue line of BGNN and orange line of AS-GCN mean the training has finished, whereas the training time of GraphSAGE is too long to be shown.

which is an extra time-consuming procedure during training; (3) The unique unsupervised learning loss in BGNN based on the adversarial learning between inter-domain message and intra-domain features does not need further computation process. For example, in GraphSAGE, the unsupervised loss is based on random walks where it will increase significantly with the graph size. More nodes and longer walk length are needed to maintain high performance, which requires even longer training time and larger memory cost. More discussion is covered in § 3.3.

5 Conclusion

In this paper, we develop the BGNN model for bipartite graph node representation learning. BGNN is expressive, unsupervised and resource-economic. We proposed Inter-domain Message Passing (IDMP) as the encoder and Intra-domain Alignment (IDA) by adversarial learning to address the node feature inconsistency issue in bipartite graphs. We further design the cascaded architecture to capture the multi-hop relationship in local bipartite structure as well as to improve the scalability. The extensive experiments confirm the effectiveness and efficiency of our models.

References

- Cao, S.; Lu, W.; and Xu, Q. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 891–900. ACM.
- Dong, Y.; Chawla, N. V.; and Swami, A. 2017. meta-path2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 135–144. ACM.
- Ganin, Y.; Ustinova, E.; Ajakan, H.; Germain, P.; Larochelle, H.; Laviolette, F.; Marchand, M.; and Lempitsky, V. 2015. Domain-adversarial training of neural networks.
- Gao, M.; Chen, L.; He, X.; and Zhou, A. BiNE: Bipartite network embedding. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval - SIGIR '18*, 715–724. ACM Press.
- Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*. arXiv: 1406.2661.
- Gori, M.; Monfardini, G.; and Scarselli, F. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, 729–734. Montreal, Que., Canada: IEEE.
- Grover, A., and Leskovec, J. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 855–864. San Francisco, California, USA: ACM Press.
- Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. *arXiv:1706.02216 [cs, stat]*. arXiv: 1706.02216.
- Huang, W.; Zhang, T.; Rong, Y.; and Huang, J. 2018. Adaptive sampling towards fast graph representation learning. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc. 4558–4567.
- Jin, W.; Barzilay, R.; and Jaakkola, T. 2018. Junction Tree Variational Autoencoder for Molecular Graph Generation. *arXiv:1802.04364 [cs, stat]*. arXiv: 1802.04364.
- Kipf, T. N., and Welling, M. 2016. Variational Graph Auto-Encoders. *arXiv:1611.07308 [cs, stat]*. arXiv: 1611.07308.
- Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks.
- LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521:436.
- Linden, G.; Smith, B.; and York, J. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* 7(1):76–80.
- Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.
- Pan, S.; Hu, R.; Long, G.; Jiang, J.; Yao, L.; and Zhang, C. 2018. Adversarially Regularized Graph Autoencoder for Graph Embedding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2609–2615. Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization.
- Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: Online Learning of Social Representations. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14* 701–710. arXiv: 1403.6652.
- Qiu, J.; Tang, J.; Ma, H.; Dong, Y.; Wang, K.; and Tang, J. 2018. DeepInf: Social Influence Prediction with Deep Learning. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '18* 2110–2119. arXiv: 1807.05560.
- Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20(1):61–80.
- Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective Classification in Network Data. *AI Magazine* 29(3):93.
- Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. LINE: Large-scale Information Network Embedding. *Proceedings of the 24th International Conference on World Wide Web - WWW '15* 1067–1077. arXiv: 1503.03578.
- Wan, H.; Luo, Y.; Peng, B.; and Zheng, W.-S. 2018. Representation Learning for Scene Graph Completion via Jointly Structural and Visual Embedding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 949–956. Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization.
- Wang, D.; Cui, P.; and Zhu, W. 2016. Structural Deep Network Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 1225–1234. San Francisco, California, USA: ACM Press.
- Yang, J.; Lu, J.; Lee, S.; Batra, D.; and Parikh, D. 2018. Graph R-CNN for Scene Graph Generation. In Ferrari, V.; Hebert, M.; Sminchisescu, C.; and Weiss, Y., eds., *Computer Vision – ECCV 2018*, volume 11205. Cham: Springer International Publishing. 690–706.
- You, J.; Liu, B.; Ying, Z.; Pande, V.; and Leskovec, J. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc. 6410–6421.
- Zhu, D.; Cui, P.; Wang, D.; and Zhu, W. 2018. Deep variational network embedding in wasserstein space. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2827–2836. ACM.

6 Appendix

This supplementary material provides source code for reproducibility, more details of the dataset, hyper-parameter settings, more experiment results, the infrastructure, and the future extension to large-scale bipartite graph system.

A Source Code for Reproducibility

Source code. For experiment results reproducibility, we store this paper’s source code at <https://tinyurl.com/BGNN>. Since we may refactor our code for further research, we maintain the original version of our code in this URL. We also provide the data that we use in this paper for running experiments. Besides the BGNN model, we also provide baseline codes we use in our experiments. Each model’s code is organized in an independent directory. In order to help reproduce our results efficiently, in the *README.md* file at the root directory, we organize a table of scripts for training procedure.

B More Experimental Evaluations

In this section, we provide more information related to our paper, including detailed analysis of datasets and models implementation details.

B.1 Datasets

Data Preprocessing (Citation Networks). For citation network dataset, we process the Cora, CiteSeer and PubMed datasets similarly. We treat the original graph as an undirected graph. First, we divide the paper documents of each class into two equal size subsets. Then, we combine the first half of all classes into U group and the second half into V group. We remove some of the features of papers in the V group to introduce heterogeneity between U and V . Lastly, we only keep edges that connected a paper in U group and a paper in V group and remove all other edges to make the graph bipartite. All the nodes that are isolated are removed.

Data Preprocessing (Social Networks). As for Tencent dataset, it is already a bipartite graph, with one set represents users, and another set represents the social communities that users joined in. As for data preprocessing, the format keeps the same as the citation networks datasets to simplify the data loading process.

Data Distribution Visualization. Distribution of our datasets is shown in Figure 7. The four datasets own distinct degree distribution. All Tencent, Citeseer and Cora datasets have a long tail exponential distribution. But Tencent dataset is more imbalance than others, which contains more multi-hops connection information. On the other hand, PubMed does not have long-tail distribution, but every node has almost the same number of edge connections.

Insight from the dataset. Due to the large-scale Tencent dataset, directly applying graph convolutional networks is impractical and impossible. The neighborhood expansion along with layer depth induces large computation time and memory cost. Even fill the entire graph into the model requires a large memory cost. So in order to deal with this problem, we come up with a new learning way called cascaded architecture. Instead of the end-to-end training which optimizing the loss after multiply layers, we transfer the

optimization procedure in between two layers. The output from the previous layer is the input to the next layer. By doing this, only one-hop neighbor are being calculated at each time but with multi-hops information embedded. The experiments prove the effectiveness of cascaded training architecture.

Furthermore, the average degree of nodes in users set is 1.6 and 11.0 in communities set. This sparse connection of users motivates us to design the cascaded architecture with deeper layers in BGNN. Since only implementing one side aggregation will significantly lose the structure information from multi-hops connections. For example, the average one-hop edge for users is 1.6, but each user has around $1.6 \times 11 \approx 18$ two-hops connections on average. With deeper BGNN layers, multi-hops information is able to be embedded into final representations. The experiments result also prove the effectiveness of multi-layers BGNN model.

B.2 Model Implementation Details

Logistic regression. In order to evaluate our model output embedding performance, we use logistic regression to predict the nodes’ label. We use the logistic *SGDClassifier* from scikit-learn Python package. We split the nodes into 80 percentage for training and the rest for testing (30% for validation).

Model Implementation. We use the code of baselines published by the author of the original paper. We summarize the baseline code we use in Table 3. We follow the parameter settings in their original papers and fine-tuned on our bipartite datasets. The Node2Vec is a high-performance version (C++), so its running time is comparable to ours. Since all the baselines are not designed for heterogeneous bipartite graph, in order to make a fair comparison with our models, we first transform the bipartite graph into a simple connected graph. We multiply the incidence matrix with its transpose to extract all two-hops connection. Since it is a bipartite graph, the two-hops connection of one set will only contain nodes in the exact same set. Through this simple transformation, the graph becomes to a single homogeneous graph, and all the baselines can achieve on it.

Hyper-Parameters. We use a grid-search to tune our model on every dataset to find the best hyperparameters. Here, we list all the final hyperparameters of BGNN for different datasets.

As for epochs, we first search in a wide range and find that with small epochs size will achieve better performance. This also proves the reason why our model requires less training time. The BGNN-MLP model contains two dense layers with rectified activation layer and dropout layer in between. The output of the decoder is aligned in the range $[-1, 1]$ using hyperbolic tangent, which is the same distribution as the input features. As for BGNN-Adv model, the discriminator also contains two dense layers but with leaky rectified activation layer, which can avoid sparse gradient problem.

B.3 More Experimental Results on Large-Scale Dataset

Training loss. Here we further show the training loss of BGNN-MLP model versus iterations on Pubmed dataset

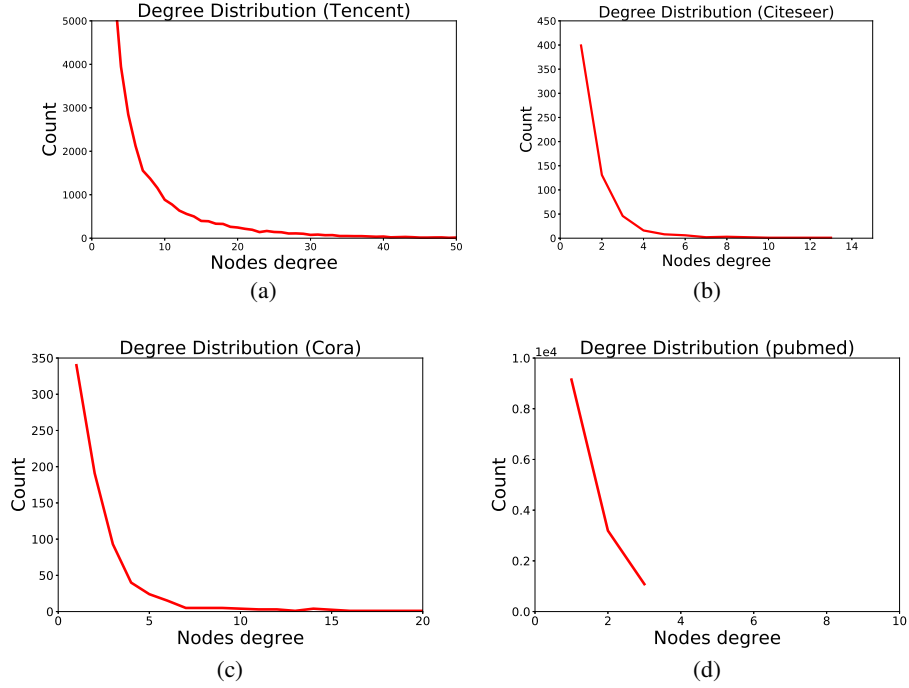


Figure 7: Node degree distributions on four datasets

Table 3: Reference for baselines code

Baseline	Code link
Node2Vec (high performance version)	https://github.com/snap-stanford/snap
VGAE	https://github.com/tkipf/gae
GraphSage	https://github.com/williamleif/GraphSAGE
GCN	https://github.com/williamleif/GraphSAGE
AS-GCN	https://github.com/huangwb/AS-GCN

Table 4: Hyperparameters for BGNN on four datasets

	Hyperparameters	Tencent	Citeseer	Cora	PubMed
BGNN-Adv	batch size	600	400	400	700
	epochs	2	4	2	3
	learning rate	0.0004	0.0004	0.0004	0.0004
	weight decay	0.0005	0.001	0.001	0.0005
	dropout	0.4	0.35	0.35	0.35
	encoder output dimensions	16	16	24	24
BGNN-MLP	batch size	500	64	128	128
	epochs	3	3	5	3
	learning rate	0.0003	0.001	0.001	0.0001
	weight decay	0.001	0.0005	0.0008	0.005
	dropout	0.4	0.2	0.2	0.2
	encoder output dimensions	24	48	48	48
	decoder hidden dimensions	16	16	16	16

in Figure 8. The reason is that PubMed is a medium-size dataset with balanced degree distribution, so it is a great dataset to illustrate. The loss function converges after around

500 iterations. The detail parameter settings can be found in table 4.

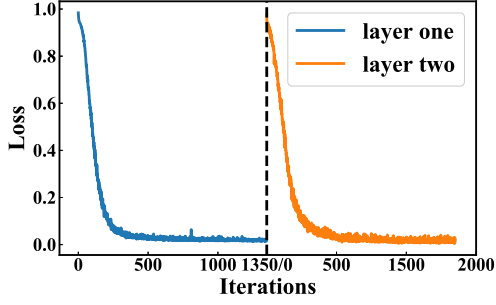


Figure 8: BGNN-MLP training loss on Pubmed.

C More Details of the Cascaded Training

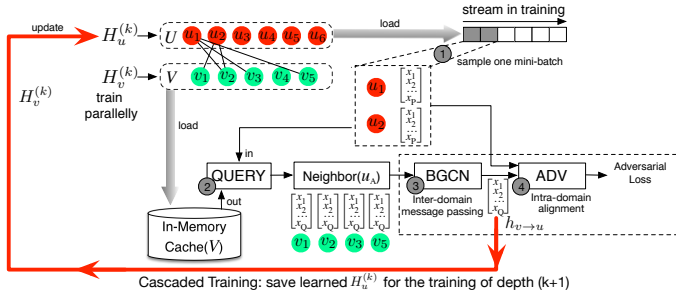


Figure 9: Cascaded Training Pipeline

We depict a detailed diagram to illustrate our cascaded training process from the perspective of set U , shown in Figure 9. The step order is shown as a circle within a number. In step 1, we sample a mini-batch of node feature vectors from group U (E.g., $h_{u(a)}$ and $h_{u(b)}$). In step 2, the QUERY operation takes the sampled node vectors as input and queries their neighbor node vectors from the opposite set V (E.g., the queried neighbor vectors are $h_{v(a)}$, $h_{v(b)}$, $h_{v(c)}$, and $h_{v(f)}$). The inter-domain message passing is in step 3 where neighbor vectors are aggregated to $h_{v \rightarrow u}$. Then in step 4, the intra-domain alignment taking sampled node vectors in U and their neighbor node vectors as input is trained with an adversarial loss. After iterating all mini-batches with multiple epochs, the learned H_u is saved for $(k+1)$ th depth cascaded training. This pipeline is consistent with Algorithm 1.