TinyGNN: Learning Efficient Graph Neural Networks

Bencheng Yan, Chaokun Wang, Gaoyang Guo, Yunkai Lou
School of Software, Tsinghua University, Beijing 100084, China
{ybc17, ggy16, louyk18}@mails.tsinghua.edu.cn, chaokun@tsinghua.edu.cn

ABSTRACT

Recently, Graph Neural Networks (GNNs) arouse a lot of research interest and achieve great success in dealing with graph-based data. The basic idea of GNNs is to aggregate neighbor information iteratively. After k iterations, a k-layer GNN can capture nodes' k-hop local structure. In this way, a deeper GNN can access much more neighbor information leading to better performance. However, when a GNN goes deeper, the exponential expansion of neighborhoods incurs expensive computations in batched training and inference. This takes the deeper GNN away from many applications, e.g., real-time systems. In this paper, we try to learn a small GNN (called *TinyGNN*), which can achieve high performance and infer the node representation in a short time. However, since a small GNN cannot explore as much local structure as a deeper GNN does, there exists a neighbor information gap between the deeper GNN and the small GNN. To address this problem, we leverage peer node information to model the local structure explicitly and adopt a neighbor distillation strategy to learn local structure knowledge from a deeper GNN implicitly. Extensive experimental results demonstrate that *TinyGNN* is empirically effective and achieves similar or even better performance compared with the deeper GNNs. Meanwhile, *TinyGNN* gains a 7.73×–126.59× speed-up on inference over all data sets.

CCS CONCEPTS

• **Information systems** → **Data mining**; *Data analytics*; • **Human-centered computing** → **Social network analysis**; *Social networks*; • **Computing methodologies** → **Neural networks**; **Inductive logic learning**; **Learning latent representations**; *Supervised learning by classification.*

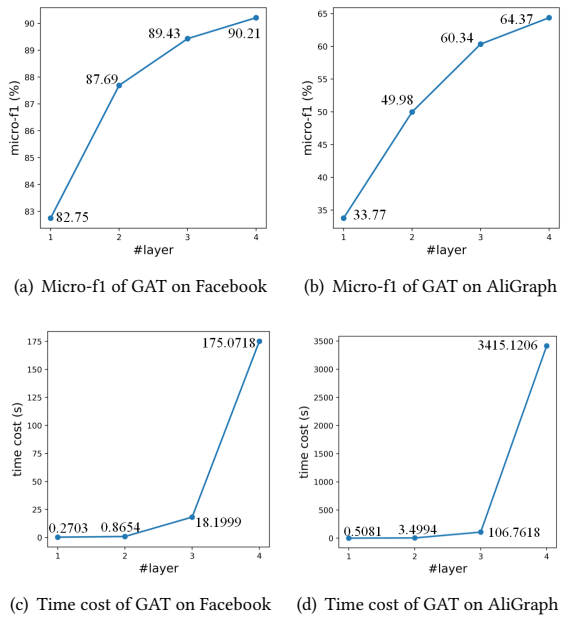
KEYWORDS

TinyGNN, Peer-Aware Module, Neighbor Distillation

ACM Reference Format:
Bencheng Yan, Chaokun Wang, Gaoyang Guo, Yunkai Lou. 2020. *TinyGNN*: Learning Efficient Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403236

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403236



(a) Micro-f1 of GAT on Facebook (b) Micro-f1 of GAT on AliGraph

(c) Time cost of GAT on Facebook (d) Time cost of GAT on AliGraph

Figure 1: An example of the GNN performance on Facebook and AliGraph.

1 INTRODUCTION

Graph structure data is ubiquitous in the real world, such as social networks [7, 16], citation networks [8] and graph-based molecules [21]. Analysis of graph data can help us to understand the relationship among objects in the graph and extract useful information. Recently, many works have been focusing on analyzing graph-based problems by leveraging the Graph Neural Networks (GNNs). The goal of GNNs is to learn a representation vector for each node in the graph. Then the learned vectors can be applied for many graph-based applications, such as link prediction, node classification, and recommender systems [7, 16, 17, 19, 22].

The general idea of GNNs is to aggregate neighbor information for each node iteratively. For example, GCN [12] aggregates neighbors by the weighted average of their information. GraphSAGE [8] proposes many types of aggregation functions, including mean aggregation, LSTM aggregation, and pooling aggregation. GAT [21] employs self-attention [20] to adaptively aggregate neighbor information according to the importance of neighbors for the center node. After k iterations of aggregation, the learned representation vectors of nodes can capture nodes' k-hop network structure. It means a deeper GNN can better characterize the local structure and is more likely to learn a better node representation [12, 24, 27]. An example is shown in Figure 1 (a)(b). The performance of node classification of an example GNN (i.e., GAT) is significantly improved when increasing the number of GNN layers. The details of

the graphs (Facebook and AliGraph) can be found in Section 5.1. Note, a deeper GNN cannot always achieve better performance due to the over-smoothing problem [14, 24] (see Section 5.6).

Although deeper GNNs usually can access much more neighbor information leading to better performance, the recursive expansion of neighborhoods across layers incurs expensive computations in batched training and inference. An example is presented in Figure 1 (c)(d). When the GNN goes deeper, the time cost of inference increases almost exponentially. Specifically, the time cost of a 4-layer GNN is nearly 6721× and 648× times longer than the time cost of a 1-layer GNN on AliGraph and Facebook, respectively. It indicates that a deeper GNN can hardly be applied to real applications, especially for real-time systems. Actually, most of the existing works [3, 13] only adopt a 1-layer GNN in applications.

As discussed above, there is a dilemma that a deeper GNN can give high performance, while the requirement of the efficiency tends to develop a small GNN. In the present paper, we focus on training a new small GNN which can well characterize the local structure of each node and achieve similar or even better performance compared with an existing deeper GNN. The challenge is that there exists a huge neighbor information gap between a small GNN and a deeper GNN. For example, assuming each node in a graph has an average of 10 neighbor nodes, the number of neighbors for each node in a 4-layer GNN is 1000× (containing repeating nodes) times larger than the number of neighbors in a 1-layer GNN. Such an information gap leads to a performance gap among GNNs. To address this problem, we propose the peer-aware module (Section 4.1) and the neighbor distillation strategy (Section 4.2) to learn as much local structure as possible. For the former, it leverages the peer node information to model the local structure explicitly. For the latter, it adopts a distillation strategy to implicitly model the local structure.

The main contributions of this paper are summarized as follows: (1) A small, efficient GNN (called *TinyGNN*) is proposed, which can achieve high performance and infer the node representation in a short time. (2) To address the neighbor information gap between a small GNN and a deeper GNN, we present Peer-Aware Module (PAM) and Neighbor Distillation Strategy (NDS) to explicitly and implicitly model the local structure, respectively. (3) Extensive experimental results demonstrate that *TinyGNN* can achieve similar or even better performance compared with a deeper GNN, and is 7.73× to 126.59× faster on inference over all data sets.

The remainder of the paper is organized as follows. Section 2 reviews related works. Section 3 presents the preliminaries. Section 4 proposes *TinyGNN*. Section 5 reports the experiments. Section 6 concludes this paper.

2 RELATED WORK

In this section, we discuss some works related to our paper, including Graph Neural Networks and Knowledge Distillation.

2.1 Graph Neural Networks

Recently, there has been a surge of interest in Graph Neural Networks (GNNs) approaches to representation learning of graphs. GCN [12] is a pioneer work, which aims at the semi-supervised node classification task on graphs. In this model, the authors truncate the Chebyshev polynomial in graph filtering [2, 5] to first-order, and naturally gather the neighbor information by the normalized graph Laplacians. GraphSAGE [8] extends the idea of GCN, and proposes an aggregation-based representation learning method. Based on the framework proposed in GraphSAGE, GAT [21] employs multi-head based self-attention to filter out unimportant information in neighbors and integrates neighbor information by different weights. GraphAE [25] tries to learn adaptive node representation across graphs. RECT [23] considers the imbalanced situation in the node representation.

Besides, there are some works also focusing on the efficiency problem in GNNs. Since the key factor of this problem is recursive neighborhood expansion, most of these works try to design a more efficiency strategy to obtain neighbors. Ying et al. [26] alleviate this problem by creating a subgraph which only contains the center node and its neighbors, and then map-reduce based strategy is used to further improve the efficiency. Chen et al. [4] propose a layer-wise importance sampling to explore neighbors which can avoid the exponential neighbor increasing. Huang et al. [10] propose a layer-wise adaptive sampling to take the nodes in the upper layer into consideration. Bojchevski et al. [1] introduce the pagerank-based value into neighborhood expansion, and incorporate multi-hop neighborhood information in a single (non-recursive) step. Actually, most of them try to solve this problem by designing a new neighbors exploring strategy. In our paper, we present a new direction to address this problem, i.e., training tiny graph neural networks. Furthermore, the proposed peer-aware module and neighbor distillation strategy are general methods which can be easily incorporated into existing GNNs (including the above efficiency-focused GNNs).

2.2 Knowledge distillation

Knowledge distillation [9] aims to compress the knowledge in a network with a large set of parameters into a compact and fast-to-execute model. The key idea is to train a small network (called the student network) by imitating the soft targets (or class probabilities) generated by the deeper model (called the teacher network). There are several variants of this technique. For example, Romero et al. [18] take the intermediate representations learned by the teacher network as additional hints to improve the training process and the final performance of the student network. Huang et al. [11] regard the knowledge distillation process as the feature distribution matching, and adopt the idea of Maximum Mean Discrepancy (MMD) used in domain adaption [6, 15]. Inspired by knowledge distillation, we propose a neighbor distillation strategy (Section 4.2) to implicitly learn the local structure of each node from a teacher network.

3 PRELIMINARIES

3.1 Notation

Let $G = (V, E)$ be a graph, where V is the node set, and E is the edge set. Each node $v \in V$ has a feature vector x_v and an associated label y_v. We define $N(v)$ as the 1-hop neighbor node set of node $v \in V$. Further, we define the nodes sampled from $N(v)$ of node v in a GNN layer as peer nodes (see Figure 2 (b), the nodes having the same color are the peer nodes to each other).

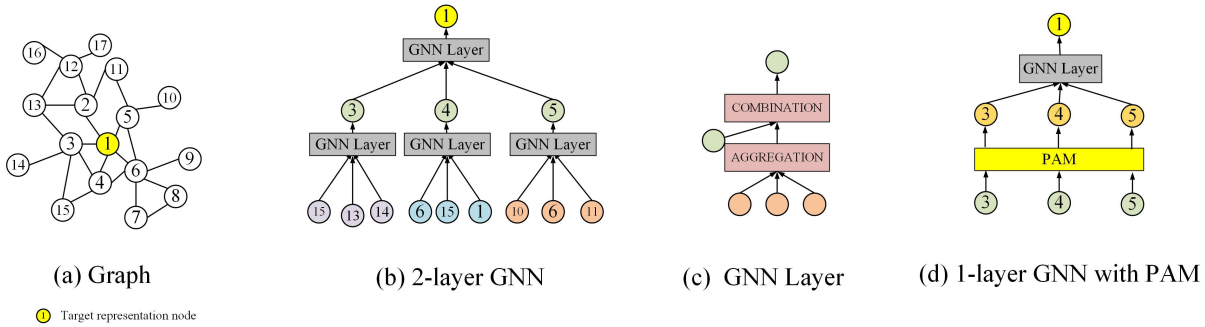


Figure 2: (a) An example of Graph, where the yellow node 1 refers to the target node to be inferred. (b) An example of a 2-layer GNN. The output is the representation vector of the target node. The nodes having the same color are the peer nodes to each other. (c) An example of a GNN layer where aggregation function is first applied to aggregate neighbor information, and then the combination function is used to update node representations. (d) An example of a 1-layer GNN with PAM, where the peer nodes are firstly fed into the peer-aware module, and then the updated representations of peer nodes are applied to the GNN layer.

3.2 Graph Neural Networks

GNNs aim to learn node representation mapping function by using the graph structure information, node features, and node labels. The main idea of GNNs can be summarized into two processes, i.e., AGGREGATION and COMBINATION (see Figure 2 (c)). AGGREGATION (e.g., mean-aggregation and max-aggregation [8]) is a process where each node aggregates the information (features) from its neighbors. Then, we can iteratively update each node representation by combining the current node representation and its neighbor information in COMBINATION. Hence, after l iterations of propagation, the node representation can capture l-hop graph structure information. An example of a 2-layer GNN is shown in Figure 2 (b). To obtain the representation of the target node 1, the GNN iteratively aggregates neighbor information from the lower layer.

Formally, at the l-th layer, a GNN can be expressed as

$$h_{N(v)}^{(l)} = AGGREGATION(\{h_u^{(l-1)} : u \in N(v)\}) \quad (1)$$

$$h_v^{(l)} = COMBINATION(h_{N(v)}^{(l)}, h_v^{(l-1)}) \quad (2)$$

where $h_v^{(l-1)}$ is the representation of node v at the $(l-1)$-th layer, which captures the $(l-1)$-hop structure information, and $h_{N(v)}^{(l)}$ can be taken as the integrated representation from neighbors. The initialization representation $h_v^0 = x_v$.

3.3 Problem Description

Given a set of training nodes, the goal of our paper is to learn a small graph neural network, such that the learned small GNN achieves similar or even better performance compared with a deeper GNN trained by the same training node set, with a much lower computational cost in inference.

4 METHOD

In this section, we introduce the proposed method. The overall framework of *TinyGNN* is shown in Figure 3. Generally speaking, to bridge the neighbor information gap between the deeper GNN and the small GNN, we design a peer-aware module (Section 4.1) and a neighbor distillation strategy (Section 4.2). The former is a neural network layer which explicitly models the neighbor information and can be easily added into a standard GNN. The latter is a Teacher-Student framework, which implicitly models the neighbor information.

Table 1: Statistics of direct link rate among peer nodes.

Data Sets	Facebook	Chameleon	Squirrel	AliGraph
rate (%)	32.81	42.94	46.91	20.69

Specifically, we randomly sample its neighbors from the target node to its neighbors for both the teacher GNN and the student GNN, respectively. Then, the attributes of the sampled neighbors are fed into the teacher GNN and the student GNN. For the teacher GNN, it adopts a standard graph neural network structure (Section 3), and the output of the final layer is the label prediction vector of the target node. For the student GNN, apart from the standard GNN layer, a peer-aware module (Section 4.1) is added before the GNN layer, and the output is also a label prediction vector. The final loss is from two parts (Section 4.2). One is the cross-entropy loss between the prediction vector produced by the student GNN and the ground-truth label. The other is the soft cross-entropy loss between the prediction vectors produced by the teacher GNN and the student GNN. Furthermore, the overall algorithm is presented in Section 4.3, and the computational complexity is analyzed in Section 4.4.

4.1 Peer-Aware Module

As introduced in Section 3, a node (e.g., node 3 in Figure 2 (b)) in the upper layer aggregates its neighbor information (e.g., nodes 13, 14 and 15 in Figure 2 (b)) from the lower layer, and there is no communication among peer nodes (e.g., peer nodes 4 and 5 are not involved in the representation updating of node 3). Actually, there is a close connection among peer nodes in the graph. At least, all these peer nodes are reachable in two hops through their common neighbor in the upper layer (e.g., nodes 3, 4, and 5 can reach each other in two hops by the common neighbor node 1). Further, we count the rate of direct connection among peer nodes in different real data sets (details of the data sets can be found in Section 5.1).

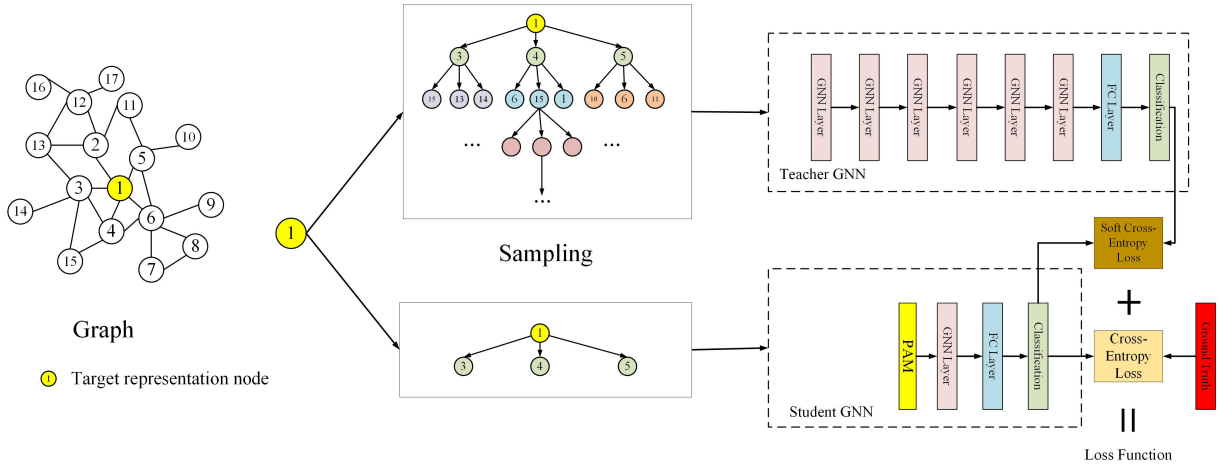


Figure 3: The proposed model, *TinyGNN*. Overall, it is a teacher-student framework. The teacher GNN is a standard deep graph neural network. The student GNN is a smaller graph neural network. Besides, before each GNN layer of the student GNN, a peer-aware module is applied. Firstly, a sampling strategy is used to sample neighbors from the node 1 for the teacher GNN and the student GNN, respectively. Then, the label prediction of the teacher GNN is taken as a soft label for the student GNN. The student GNN is trained by the soft label from the teacher GNN and the ground-truth label.

Table 1 shows that an average of 20.69% to 46.91% peer nodes are directly connected. It means a large number of peer nodes are neighbors, and the neighbor information from the lower layer can be directly profiled by peer nodes. Inspired by this observation, we propose a novel module called Peer-Aware Module (PAM), which utilizes peer node information to address the problem that a small GNN cannot access much more neighbor information from the lower layer.

Given the d-dimensional representation set of K peer nodes $\{h_i\}_{i=1}^{K}$, Peer-Aware Module updates each node representation in the set by considering other peer nodes.

$$h_i^* = \sum_j \alpha_{i,j} \cdot (W_v \cdot h_j) \tag{3}$$

$h_i^* \in R^d$ is an updated representation for each peer node. It is a weighted sum of representations from other peer nodes, linearly transformed by $W_v \in R^{d*d}$. The weight $\alpha_{i,j}$ indicates the implicit connection between peer nodes i and j. It is computed as

$$\alpha_{i,j} = \frac{exp(a_{i,j})}{\sum_k exp(a_{i,k})} \tag{4}$$

$$a_{i,j} = \frac{dot(W_a h_j, W_a h_i)}{\sqrt{d}} \tag{5}$$

Where W_a is a weight matrix. It projects the original features h_i and h_j into a subspace to measure how well they match. *dot* refers to the dot product. d is the scaling factor. Note, when updating the node representation, the information of the node itself is also considered, i.e., $j \in i \cup otherpeernodes$. Actually, PAM adopts self-attention [20] to update representation of neighbor nodes.

The Peer-Aware Module enables each node to explore the local structure explicitly by peer nodes without aggregating the information from lower layer nodes. This kind of property helps the small GNN to avoid heavy computation induced by iteratively aggregating neighbors from the lower layer.

We notice that GAT [21] also adopts self-attention to aggregate node information. However, the idea of PAM and GAT is totally different. Self-attention used in PAM is to aggregate the peer nodes (e.g., in Figure 2 (d), self-attention is applied on peer nodes 3, 4 and 5 in the same layer), while self-attention used in GAT is to aggregate the neighbor nodes from the lower layer (e.g., in Figure 2 (b), neighbor information of nodes 13, 14 and 15 is aggregated to nodes 3). Actually, PAM explores new relations among nodes in the same layer. While GAT simply follows the standard GNN layer, i.e., nodes in the upper layer aggregate neighbor information from the lower layer.

Peer-Aware Module for Graph Neural Network Layer. Peer-Aware Module has the same input and output dimension. Hence it can be regarded as a basic building block to be used in-place within any graph network architecture. Peer-Aware Module can be applied before GNN layers (see Figure 2 (d)). The neighbors of center node 1 (i.e., peer nodes 3, 4, and 5) are first fed into PAM. Their representations are enriched by other peer nodes. Then the peer-aware representations are taken as the input representations of a GNN layer.

Taken Figure 2 (b)(d) as an example, a small GNN can be benefited from (1) **Low computation**. Compared with a 2-layer GNN (a deeper model), a 1-layer GNN with PAM is still a small model which can avoid exponential explosion problem. Hence, there is no much time cost in the inference phase. (2) **Exploring more local structure**. Compared with a 2-layer GNN (a deeper model), although nodes 3, 4, and 5 in a 1-layer GNN with PAM cannot obtain rich neighbor information from the lower layer, the local structure information can be characterized from other peer nodes by PAM. Hence the high performance of a small GNN can be achieved. The effectiveness of PAM is validated in Table 3 and Section 5.3.

4.2 Neighbor Distillation Strategy

Although PAM addresses the inability of a small GNN to explore neighbors explicitly, local structure information is limited by peer

nodes. A lot of neighbor information is still ignored. To allow the small GNN can still leverage amount of neighbor information, inspired by Knowledge Distillation (KD) [9], we propose a Neighbor Distillation Strategy (NDS). It is a **Teacher-Student** network, wherein the teacher GNN has access to more neighbors than the student GNN. By imitating the behaviors of the teacher GNN (a deeper GNN), abundant neighbor knowledge can be captured in the student GNN (a small GNN) implicitly.

4.2.1 Teacher. The teacher network is a standard graph neural network. This teacher network explores rich neighbor information by adopting a deeper GNN.

Assuming $\{x_i, y_i\}_{i=1}^N$ are N training samples, where x_i is the i-th node representation vectors for GNN, and y_i is the corresponding ground-truth label. The parameters of the teacher network can be learned using a standard multi-label classification loss L_{CE}^t, which is a sum of the cross-entropy loss between the true label y_i and prediction $\widehat{y_i^t}$ from the teacher network.

$$L_{CE}^t = \sum_{i=1}^{N} y_i \log(\widehat{y_i^t}) + (1 - y_i) \log(1 - \widehat{y_i^t}) \tag{6}$$

4.2.2 Student. The student network is a small GNN with Peer-Aware Module (Section 4.1). Compared with the teacher network, the student network lacks neighbor information. To transfer the neighbor knowledge from the teacher network to the student network, a neighbor distillation loss [9] is adopted to encourage the student model to imitate the teacher's behavior. Specifically, we penalize the soft cross-entropy loss between the student network's logits against the teacher's logits:

$$L_{ND} = -softmax(z_t/T) \cdot \log softmax(z_s/T) \tag{7}$$

where z_t and z_s are the logit vectors predicted by the teacher and student, respectively. T is the temperature used in KD, which controls how much to rely on the teacher's soft predictions. A higher temperature produces a more diverse probability distribution over classes [9]. Besides distillation loss, the supervised information for the student network can also come from the node label prediction, where a cross-entropy loss is included for model training:

$$L_{CE}^s = \sum_{i=1}^{N} y_i \log(\widehat{y_i^s}) + (1 - y_i) \log(1 - \widehat{y_i^s}) \tag{8}$$

where $\widehat{y_i^s}$ is the prediction from the student network. Thus, the final objective function for neighbor distillation can be formulated as:

$$L_{Student} = L_{CE}^s + \alpha L_{ND} \tag{9}$$

where α is the hyper-parameter that balances the importance of the classification loss and the neighbor distillation loss.

4.3 Training

The overall training algorithm is summarized in Algorithm 1. First, we train the teacher GNN on the training set V^t by the label classification loss (Line 3-4). Then, we fix the parameters of the learned teacher GNN (Line 5). Next, we train the student GNN. Specifically, in each iteration, we sample a batch of nodes from the training set V^t (Line 9). At each layer l, the representations of neighbors are fed into the peer-aware module, and their representations are updated by peer nodes (Line 14). Then, a standard GNN layer (including aggregation and combination) is applied (Line 15-16). After the forward propagation (Lines 9-17), we calculate the soft cross-entropy loss L_{ND} (Line 18) and the student classification loss L_{CE}^s (Line 19). Then, back-propagation is carried out to update the parameters (Line 20). Finally, after converging, we take the learned student GNN as *TinyGNN* to be used for inference (Line 22).

Algorithm 1: The Algorithm of *TinyGNN*.

Input: Graph $G = (V, E)$; Node features $\{x_v^t, \forall v \in V^t\}$; Training node set $V^t = \{v, \forall v \in V\}$; Teacher layer number L_t; Student layer number L_s; Temperature T; Neighbor distillation loss weight α;
Output: the learned student GNN (*TinyGNN*);

1 set $h_v^{(0)} = x_v, \forall v \in V^t$;
2 // Teacher GNN per-train;
3 Initializing the parameters of the teacher GNN;
4 Training the teacher GNN by the supervised label information from V^t, according to Eq 6;
5 Fixing the parameters of the learned teacher GNN;
6 // Student GNN learning;
7 Initializing the parameters of student GNN;
8 **while** *not converge* **do**
9 Sampling node batch B from V^t;
10 Initializing $L_{CE}^s = 0$;
11 Initializing $L_{ND} = 0$;
12 Obtaining the logit vectors z_t predicted by feeding the batch node features into the teacher GNN;
13 **for** *l=1,...,*L_s **do**
14 Learning peer node information, i.e., updating current peer nodes representation according to Eq 3;
15 Aggregating neighbor information from the updated peer nodes representation, according to Eq 1;
16 Updating current center node representation by combining the aggregated neighbor information and center node representation, according to Eq 2;
17 **end**
18 Adding the soft cross-entropy loss between the logits of teacher and student GNN, according Eq. 7 to L_{ND};
19 Adding the student classification loss according Eq. 9 to L_{CE}^s;
20 Back-propagation and update parameters in the student GNN;
21 **end**
22 return the learned student GNN (*TinyGNN*).

4.4 Complexity Analysis

TinyGNN is a general GNN model. The aggregation (Eq 1) and the combination (Eq 2) can vary. Hence, in this section, we take the aggregation and combination used in GAT [21] as an example to analyze the complexity of *TinyGNN*. In the training phase, the time cost comes from two parts, i.e., the teacher GNN and the student GNN. For the teacher GNN, at the l-th layer, the computational complexity of the attention calculation is $O(N^{(l)} I^{(l)} F^{(l)} + E_N^{(l)} (F^{(l)})^2)$, where $N^{(l)}$ is the number of neighbor nodes at the l-layer, $I^{(l)}$ is the dimension of input features at the l-layer, $F^{(l)}$ is the dimension of the transformed feature at the l-th layer, and $E_N^{(l)}$ is the number of edges among the nodes at the $(l-1)$-th and the l-th

Table 2: Data Sets Information.

Data Sets	#Nodes	#Edges	#Class
Facebook	22,470	171,002	4
Chameleon	2,277	31,421	10
Squirrel	5,201	198,493	6
AliGraph	46,800	946,175	7

layer. Then, the computational complexity of the l-th GNN layer is $T^{(l)}_{layer} = O(N^{(l)}I^{(l)}F^{(l)} + E^{(l)}_N(F^{(l)})^2 + N^{(l)}I^{(l)}F^{(l)} + N^{(l)}I^{(l+1)}(F^{(l)} + I^{(l)}))$. The overall computational complexity of the teacher GNN is $T_{teacher} = \sum_l^{L_t} T^{(l)}_{layer}$, where L_t is the layer number of the teacher GNN. For the student GNN, apart from the time cost from the GNN layer, the computational complexity of PAM at the l-th layer is $T^{(l)}_{PAM} = O(P^{(l)}I^{(l)}F^{(l)} + P^{(l)}(P^{(l)}-1)(F^{(l)})^2)$, where $P^{(l)}$ is the number of peer nodes at the l-th layer. Then the overall computational complexity of the student GNN is $T_{student} = \sum_l^{L_s}(T^{(l)}_{layer} + T^{(l)}_{PAM})$, where L_s is the layer number of the student GNN. Hence, the time cost of training phase is $T_{teacher}+T_{student}$. As for the inference phase, since only the student GNN is used to infer the node representation, the time cost is $T_{student}$.

5 EXPERIMENTS

In this section, firstly, data sets used in the experiments are introduced. Then, baselines and training details are presented. At last, the experimental results are reported and discussed.

5.1 data sets

We evaluate the performance of our method on four real-world graphs: **Facebook** (http://snap.stanford.edu/data/) is a web graph. Nodes represent official Facebook pages while the links are mutual likes between sites. Node labels refer to the page from different categories. Node features are extracted from the site descriptions. **Chameleon** (http://snap.stanford.edu/data/) is a Wikipedia page-page network collected on a specific topic: chameleons. Nodes are articles from the English Wikipedia. Edges are mutual links that exist between pairs of sites. Node features describe the presence of nouns appearing in the articles. Node labels refer to the monthly traffic-level of the corresponding pages. **Squirrel** (http://snap.stanford.edu/data/) is another Wikipedia page-page network collected on a specific topic: squirrels. This graph is constructed in the same way as Chameleon. **AliGraph** (https://tianchi.aliyun.com/dataset/dataDetail?dataId=58&lang=en-us) is a graph constructed from real-world traffic logs of the recommender system in Taobao, the largest online retail platform in the world. Nodes refer to items. If two items are viewed by the same user within 30 minutes, there will be one edge between these two items. Each item (node) contains 36 features. We take the item category as the label information. Each item has only one category, and there is a total of 7 categories.

The statistics of the data sets are summarized in Table 2.

5.2 Baselines and Training Details

In the following experiments, we use GAT [21] as an example to investigate the proposed method for Graph Neural Networks.

Baselines used in our paper are listed as follows:

- GNN_k refers to a k-layer GNN, which is directly trained on a set of training nodes in a graph.
- GNN_k*-PAM* refers to a k-layer GNN with Peer-Aware Module (Section 4.1).
- GNN_k*-NDS* refers to a k-layer GNN with Neighbor Distillation Strategy (Section 4.2), i.e., GNN_k*-NDS* is trained with the help of another teacher network.
- $TinyGNN_k$ is the proposed method, referring to a k-layer GNN with PAM and NDS.

In order to reduce the hyper-parameter search space, we fix the node representation dimension as 16, the batch size as 64, and the learning rate as 0.001 for all the tasks. We set the number of sampled neighbors in each layer for the graphs Facebook, Chameleon, and Squirrel as 10, and for AliGraph as 20. We set the temperature T as $\{1, 3, 5, 8, 10\}$, $\alpha = \{0.1, 0.5, 1, 2, 5\}$ and perform grid search over T, α. For all graphs, we split 10% nodes as the training nodes and 45% nodes as the validation nodes and the rest 45% as the test nodes. We use an early stopping strategy on both the model loss and accuracy score on the validation nodes, with the patience of 20 epochs. All the experiments are performed on a single GTX 1080ti GPU.

5.3 Node Classification Task

We evaluate *TinyGNN* on node classification. In neighbor distillation strategy, GNN_3 is taken as a teacher network which is pre-trained on the same training set used in the training phase of the student network. We take the 1-layer GNN and the 2-layer GNN as the student network, respectively. To have a comprehensive comparison, the results of the *TinyGNN* variants which are only trained with NDS or PAM are also reported. Besides, we also compare the base model results of GNN_3, GNN_2 and GNN_1. Results are summarized in Table 3.

Comparisons with the base GNN_1 (or GNN_2): We first compare the results between *TinyGNN* and *GNN* (Base) in the same layer. The proposed model *TinyGNN* performs best on all the tasks among all GNNs in the same layer. Specially, the improvements of $TinyGNN_1$ is from 1.81% to 8.57% on Micro-f1, and 1.76% to 7.51% on Macro-f1 over all data sets. The gains of $TinyGNN_2$ is from 1.71% to 11.14% on Micro-f1, and 1.83% to 10.72% on Macro-f1 over all data sets. It indicates that, although both $TinyGNN_k$ and GNN_k ($k = \{1, 2\}$) have the same number of GNN layers, $TinyGNN_k$ can better characterize the local structure of each node, and learn a better node representation.

Comparisons with the teacher GNN_3: Then we compare the results between *TinyGNN* and GNN_3 (Teacher). We can observe that: (1) Although the number of neighbors for GNN_3 (Teacher) is 100× larger than $TinyGNN_1$, $TinyGNN_1$ can still obtain comparable performance with GNN_3 (Teacher) in most data sets. Especially for Chameleon, $TinyGNN_1$ outperforms the teacher network and gains 0.59% and 0.3% on Micro-f1 and Macro-f1, respectively. For Squirrel, $TinyGNN_1$ outperforms the teacher network and gains 1.41% and 1.59% on Micro-f1 and Macro-f1, respectively. Further, we also notice that there is still a gap between $TinyGNN_1$ and GNN_3 (Teacher) on AliGraph. It shows that abundant neighbor information is important in representing the node structure. (2) When we add additional one GNN layer for $TinyGNN_1$ (i.e., $TinyGNN_2$), although the number neighbors of GNN_3 (Teacher) is still 10× larger than $TinyGNN_2$,

Table 3: Results for node classification. The best results for one- or two-layer GNNs are in bold, respectively. GNN_3 (Teacher) is the teacher GNN. GNN_1 (Base) and GNN_2 (Base) are direct trained on each dataset without using NDS and PAM.

Model	Facebook		Chameleon		Squirrel		AliGraph	
	Micro-f1(%)	Macro-f1(%)	Micro-f1(%)	Macro-f1(%)	Micro-f1(%)	Macro-f1(%)	Micro-f1(%)	Macro-f1(%)
GNN_3 (Teacher)	89.43	88.69	39.12	38.16	30.68	30.41	60.34	53.75
GNN_2 (Base)	87.69	86.83	38.73	36.92	30.47	30.08	49.98	43.45
GNN_1 (Base)	82.75	81.65	36.88	35.79	29.83	29.24	33.77	25.22
GNN_1-*NDS*	83.32	82.11	37.46	36.42	30.73	30.38	35.30	24.88
GNN_1-*PAM*	83.15	81.79	38.05	36.56	31.32	31.08	40.19	32.05
TinyGNN$_1$	**84.56**	**83.41**	**39.71**	**38.46**	**32.09**	**32.00**	**42.34**	**32.37**
GNN_2-*NDS*	88.90	88.15	39.51	38.21	30.98	30.52	54.72	46.97
GNN_2-*PAM*	88.56	87.72	40.98	39.62	32.69	31.95	57.81	50.40
TinyGNN$_2$	**89.40**	**88.66**	**41.17**	**40.01**	**33.33**	**33.17**	**61.12**	**54.17**

Table 4: The inference time for each model on different data sets.

model	Facebook	Chameleon	Squirrel	AliGraph
GNN_3 (Teacher)	18.1999 (1×)	1.2715 (1×)	3.2022 (1×)	106.7618 (1×)
GNN_2 (Base)	0.8654 (21.03×)	0.0856 (14.85×)	0.164 (19.53×)	3.4994 (30.51×)
GNN_1 (Base)	0.2703 (67.33×)	0.03 (40.50×)	0.0447 (67.13×)	0.5081 (210.12×)
TinyGNN$_1$	0.4314 (42.19×)	0.0332 (38.30×)	0.0897 (35.70×)	0.8434 (126.59×)
TinyGNN$_2$	2.3542 (7.73×)	0.1059 (12.00×)	0.3112 (10.29×)	5.0072 (21.32×)

TinyGNN$_2$ outperforms GNN_3 (Teacher) on almost all data sets except for Facebook. It indicates that *TinyGNN* can improve the performance with the help of modeling both implicit and explicit information.

Ablation analysis: We further investigate the performance gain from Peer-Aware Module (PAM) and Neighbor Distillation Strategy (NDS). Compared with GNN_1, GNN_1-*PAM* achieves better performance in all data sets, and GNN_1-*NDS* outperforms GNN_1 on almost all the data sets except for AliGraph in Macro-f1. Similar conclusion can be found in the 2-layer GNNs (i.e., GNN_2-*PAM* and GNN_1-*NDS*). The results indicate that both of PAM and NDS are crucial for the proposed *TinyGNN*. Then, the combination (i.e., *TinyGNN*) can achieve the best performance among all GNNs in the same layer. Further, GNN_1-*PAM* (or GNN_2-*PAM*) performs better than GNN_1-*NDS* (or GNN_2-*NDS*) on almost all data sets except for Facebook. It shows modeling the local structure (PAM) explicitly can be more effective than modeling (NDS) implicitly in node representations.

5.4 Analysis of Model Efficiency

We have demonstrated that *TinyGNN* can effectively learn node representation without performance sacrifice. In this section, we further analyze the efficiency of *TinyGNN* on inference-time speed-up. Specifically, we conduct experiments on all samples of the testing set in each dataset. The results are shown in Table 4. It shows that: (1) The inference time cost of *GNN* achieves an almost exponential speed-up when the number of layers increases. The reason is that the number of sampled neighbor nodes grows exponentially, which leads to heavy computation. (2) Compared with the teacher (deeper) GNN_3, *TinyGNN*$_1$ makes a significant improvement and achieves 35.70× to 126.59× times speed-up over all data sets. For *TinyGNN*$_2$, it achieves 7.73× to 21.32× times speed-up over all data sets. This brings the possibility for GNN to be applied in a wider range of fields, especially for a real-time system. (3) Considering the large time cost intruded by increasing layers, *TinyGNN* achieves a similar time cost with *GNN* (base) with the same number of layers. It indicates that the additional time cost from PAM of *TinyGNN* is tiny. Note the NDS is only used in the training phase. The time cost of NDS does not affect the inference phase.

5.5 Analysis of Convergence

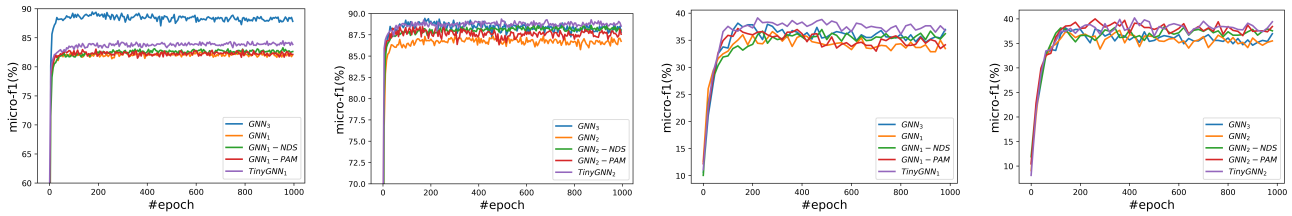
In this subsection, we conduct some experiments to analyze the convergence of the proposed model on the node classification task. For the space limitation, we only report the Micro-f1 of each GNN on different data sets within 1000 epoch (similar conclusions can be found in the Macro-f1). The results are shown in Figure 4. Each subfigure refers to the results of 1-layer or 2-layer GNNs in a dataset. Besides, we also report the results of the teacher GNN in each subfigure. We can observe that: (1) In most cases, all the three *TinyGNN* variants (including $GNN-NDS$, $GNN-PAM$ and *TinyGNN*) are faster to be saturated than GNN (Base) with the same layer, and achieve better results on Micro-f1. (2) Compared with the teacher network (GNN_3), *TinyGNN*$_2$ still can converge faster, and improve performance, which demonstrates the effectiveness of *TinyGNN*.

5.6 Different Teacher GNNs vs. Different Student GNNs

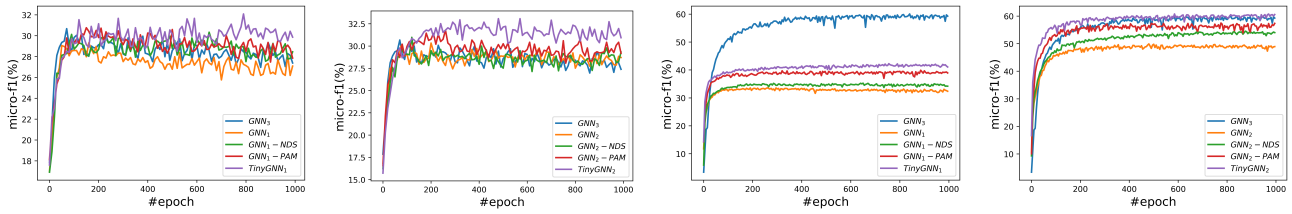
We further conduct additional experiments to evaluate the effectiveness of different teacher GNNs for different student GNNs. Specifically, we take GNN_4, GNN_3, and GNN_2 as teacher GNNs, respectively. Meanwhile, *TinyGNN*$_3$, *TinyGNN*$_2$, and *TinyGNN*$_1$ are considered as student GNNs. The results are reported in Table 5.

Comparisons with different Teachers: First, we compare the performance of the same student GNN with different teacher GNNs. From Table 5, some observations are summarized as follows:

(1) **Does a better teacher help?** In general, no matter which teacher GNN is used, the learned *TinyGNN* always can outperform



(a) Micro-f1 of 1-layer GNNs on Facebook (b) Micro-f1 of 2-layer GNNs on Facebook (c) Micro-f1 of 1-layer GNNs on Chameleon (d) Micro-f1 of 2-layer GNNs on Chameleon



(e) Micro-f1 of 1-layer GNNs on Squirrel (f) Micro-f1 of 2-layer GNNs on Squirrel (g) Micro-f1 of 1-layer GNNs on AliGraph (h) Micro-f1 of 2-layer GNNs on AliGraph

Figure 4: The Micro-f1 of GNNs in different epoch on different data sets.

Table 5: Performance comparison with different teacher and different student GNNs. The best results for GNNs with the same layer number are in bold.

		Facebook		Chameleon		Squirrel		AliGraph	
Base		Micro-f1(%)	Macro-f1(%)	Micro-f1(%)	Macro-f1(%)	Micro-f1(%)	Macro-f1(%)	Micro-f1(%)	Macro-f1(%)
GNN_1		82.75	81.65	36.88	35.79	29.83	29.24	33.77	25.22
GNN_2		87.69	86.83	38.73	36.92	30.47	30.08	49.98	43.45
GNN_3		89.43	88.69	39.12	38.16	30.68	30.41	60.34	53.75
GNN_4		90.21	89.53	38.73	36.57	29.70	28.94	64.37	58.35
Teacher	Student								
GNN_4	$TinyGNN_1$	**84.99**	**83.75**	39.22	37.72	**32.86**	**32.48**	**42.67**	**32.39**
GNN_3	$TinyGNN_1$	84.56	83.41	**39.71**	**38.46**	32.09	32.00	42.34	32.37
GNN_2	$TinyGNN_1$	83.04	81.73	38.24	36.12	32.22	31.66	41.43	31.85
GNN_4	$TinyGNN_2$	**89.89**	**89.24**	40.68	39.22	32.44	32.29	**61.28**	**54.26**
GNN_3	$TinyGNN_2$	89.40	88.66	**41.17**	**40.01**	**33.33**	**33.17**	61.12	54.17
GNN_4	$TinyGNN_3$	**90.83**	**90.25**	**41.46**	**40.46**	**31.58**	**31.40**	**67.07**	**60.83**

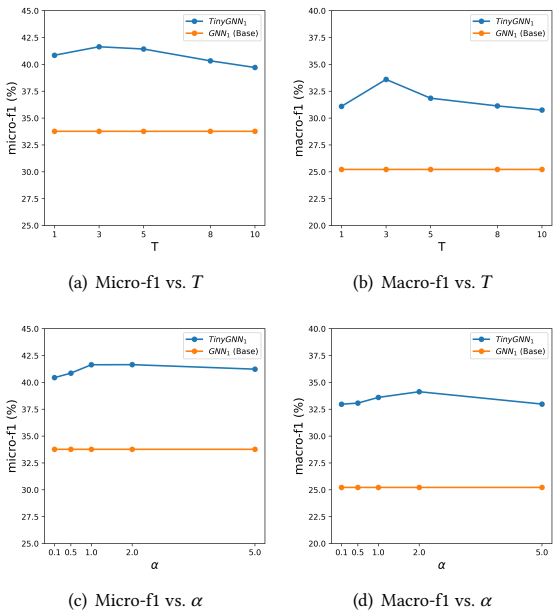
the GNN (Base) with the same number of GNN layers. If one teacher GNN is much better than another teacher GNN, the corresponding student GNN can get better performance in most cases, e.g., GNN_k ($k = 3, 4$) outperforms GNN_2 on Facebook and AliGraph, the corresponding student $TinyGNN_1$ can be further improved. We also note that sometimes *TinyGNN* cannot achieve better performance with a better teacher, e.g., GNN_4 outperforms GNN_3 on Squirrel, while $TinyGNN_1$ trained by GNN_3 is better than that trained by GNN_4. It indicates the selection of the teacher GNN not only depends on the teacher performance, but also depends on the dataset.

(2) **Does a deeper teacher help?** A deeper teacher GNNs also does not always obtain better performance. Sometimes, due to the over-smoothing problem [14, 24], the performance of GNNs drops with the number GNN layer increasing (e.g., the performance of GNN_4 is worse than GNN_3 performance on Chameleon and Squirrel). Further, a deeper GNNs also need heavy computations on training phrase. Hence a teacher GNN with a suitable layer number is recommended.

Comparisons with different students: We further analyze the performance of different students with the same teacher. It shows that a deeper student GNN can significantly outperform the smaller student GNN in almost all data sets except for Squirrel. When applying the student GNN into a real system, the inference time is an important factor to be considered. According to Table 4, the inference time grows nearly exponentially with the increasing of layers. Thus, only 1-layer GNNs are always considered to be learned.

5.7 Parameter Sensitivity

In this section, we carry out the experimental analysis of several important parameters (including T and α) involved in *TinyGNN*. Since all the four data sets have a similar conclusion, we only report the results on AliGraph. Here, we take a 2-layer GNN as the teacher GNN (GNN_2) and 1-layer *TinyGNN* as a student GNN ($TinyGNN_1$). Besides, we also plot the results of GNN_1 as the base results. The results are reported in Figure 5.



(a) Micro-f1 vs. T (b) Macro-f1 vs. T

(c) Micro-f1 vs. α (d) Macro-f1 vs. α

Figure 5: Parameters analysis on AliGraph for T and α.

The temperature parameter T: The predictions of the teacher GNN are smoothed by the temperature T, which controls how much to rely on the teacher knowledge. We test the performance of $TinyGNN_1$ with $T = \{1, 3, 5, 8, 10\}$, respectively. It shows that: (1) Compared with GNN_1 (Base), $TinyGNN_1$ is still much better than GNN_1 with different T. It demonstrates the effectiveness of *TinyGNN*. (2) When the value of T becomes larger, the performance drops slightly. The reason may be that the predictions of the teacher are over smoothed by large T. Hence, a suitable value of T (e.g., $T = 3$) can achieve better performance.

The balance parameter α: Here, we experimentally analyze the effect of parameter α on the results. In general, the balance parameter α allows *TinyGNN* to take the teacher knowledge into consideration. We conduct some experiments with $\alpha = \{0.1, 0.5, 1, 2, 5\}$. We observe that $TinyGNN_1$ can outperforms GNN_1 (Base) in different setting, and a suitable value α (e.g., $\alpha = 1$) can further improve the performance.

6 CONCLUSION

In this paper, we focus on learning an efficient graph neural network (called *TinyGNN*), which can achieve similar or better performance achieved in a deeper GNN while inferring the node representation in a short time. The challenge is that unlike a deeper GNN, a small GNN has no access to much more neighbor information limiting the learning ability of the GNN. To address this problem, we first propose a peer-aware module, which utilizes the information from peer nodes to model the local structure explicitly. Then a neighbor distillation strategy is designed to transfer the structure knowledge from a deeper GNN implicitly. In this way, *TinyGNN* can effectively characterize the local structure and learn a better node representation. *TinyGNN* is empirically effective and achieves better performance than the deeper GNN, while being 7.73× to 126.59× speed-up on inference over all data sets.

7 ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (No. 61872207) and Baidu Inc. Chaokun Wang is the corresponding author.

REFERENCES

[1] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Martin Blais, Amol Kapoor, Michal Lukasik, and Stephan Günnemann. 2019. Is pagerank all you need for scalable graph neural networks?. In *ACM KDD, MLG Workshop*.
[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *ICLR*.
[3] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Representation Learning for Attributed Multiplex Heterogeneous Network. In *KDD*. 1358–1368.
[4] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *ICLR*.
[5] Michael Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 3844–3852.
[6] Arthur Gretton, Dino Sejdinovic, Heiko Strathmann, Sivaraman Balakrishnan, Massimiliano Pontil, Kenji Fukumizu, and Bharath K Sriperumbudur. 2012. Optimal kernel choice for large-scale two-sample tests. In *NIPS*. 1205–1213.
[7] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864.
[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.
[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. In *arXiv preprint arXiv:1503.02531*.
[10] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. In *NIPS*. 4558–4567.
[11] Zehao Huang and Naiyan Wang. 2017. Like what you like: Knowledge distill via neuron selectivity transfer. In *arXiv preprint arXiv:1707.01219*.
[12] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[13] Feng Li, Zhenrui Chen, Pengjie Wang, Yi Ren, Di Zhang, and Xiaoyu Zhu. 2019. Graph Intention Network for Click-through Rate Prediction in Sponsored Search. In *SIGIR*. 961–964.
[14] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*. 3538–3545.
[15] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I Jordan. 2015. Learning transferable features with deep adaptation networks. In *ICML*. 97–105.
[16] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. ACM, 701–710.
[17] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *KDD*. ACM, 385–394.
[18] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. Fitnets: Hints for thin deep nets. In *ICLR*.
[19] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.
[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*. 5998–6008.
[21] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[22] Changping Wang, Chaokun Wang, Zheng Wang, Xiaojun Ye, and Philip S Yu. 2020. Edge2vec: Edge-based Social Network Embedding. In *TKDD*. ACM.
[23] Zheng Wang, Xiaojun Ye, Chaokun Wang, Jian Cui, and Philip Yu. 2020. Network Embedding with Completely-imbalanced Labels. In *IEEE Transactions on Knowledge and Data Engineering*.
[24] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*. 5449–5458.
[25] Bencheng Yan and Chaokun Wang. 2020. GraphAE: Adaptive Embedding across Graphs. In *ICDE*. IEEE, 1958–1961.
[26] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*. 974–983.
[27] Yizhou Zhang, Yun Xiong, Xiangnan Kong, Shanshan Li, Jinhong Mi, and Yangyong Zhu. 2018. Deep Collective Classification in Heterogeneous Information Networks. In *WWW*. 399–408.