SAMPLING BEFORE TRAINING: RETHINKING THE EFFECT OF EDGES IN THE PROCESS OF TRAINING GRAPH NEURAL NETWORKS CONFERENCE SUBMISSIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNN) demonstrate excellent performance on many graph-based tasks; however, they also impose a heavy computational burden when trained on a large-scale graph. Although various sampling methods have been proposed to speed up training GNN by shrinking the scale of the graph **during training**, they become unavailable if we need to perform sampling **before training**. In this paper, we quantify the importance of every edge for training in the graph with the extra information they convey in addition to the node features, as inspired by a manifold learning algorithm called *diffusion map*. Based on this calculation, we propose **Graph Diffusion Sampling** (GDS), a simple but effective sampling method for shrinking the size of the edge set before training. GDS prefers to sample edges with high importance, and edges dropped by GDS will never be used in the training procedure. We empirically show that GDS preserves the edges crucial for training in a variety of models (GCN, GraphSAGE, GAT, and JKNet). Compared to training on the full graph, GDS can guarantee the performance of the model while only samples a small fraction of the edges.

1 INTRODUCTION

Nowadays, graph-structured data has proliferated in many important applications for its ability to efficiently represent complex networks such as social networks (Backstrom & Leskovec, 2011), brain networks (Lee et al., 2017), and transaction networks (Liu et al., 2018). How to mine useful information from graphs has become a crucial problem. Denote A graph dataset as $\mathscr{G} = (\mathscr{V}, \mathscr{E}, \mathbf{X})$, where $\mathscr{V}$ is the set of nodes, $\mathscr{E}$ is the edge set describing how nodes are connected, and $\mathbf{X}$ is a matrix collecting the feature vectors of every node. Graph neural networks (GNN) have become increasingly popular tools for this challenge. By training a GNN model on a graph dataset, we can obtain node embeddings that can be utilized for the downstream tasks such as node classification. GNN models such as Graph Convolution Network (GCN) (Kipf & Welling, 2016), GraphSAGE (Hamilton et al., 2017), Graph Attention Network (GAT) (Veličković et al., 2017) and JKNet (Xu et al., 2018) have shown excellent efficiency in learning graph representations.

However, these models are inefficient when the graph is large-scale. For a node in the graph, calculating its embedding need aggregate embeddings of its neighbors by the aggregator specified by the model. Therefore, the number of nodes we need for embedding this node increases exponentially with the number of layers. For attention-based models such as GAT, another complexity is introduced by the computation of pairwise attention weights between connected nodes, which involves the softmax operator. The above two operations cause a significant increase in the computational when the degree (the number of neighbors of each node) is high. This problem is even worse when executing backpropagation on the GNN during training.

To overcome the drawback of high computational cost, a number of sampling methods are proposed (Liu et al., 2021). Computational complexity can be controlled by training on a subgraph of $\mathscr{G}$ which is more sparse than the original graph. By sampling neighbors of every node, for instance, we can limit the degree of nodes, thereby lifting the computational burden of neighbor aggregation and the attention calculation. Various sampling methods are applied to training GNN, including node-wise sampling (sampling the neighbor set of every node each time we calculate the aggregation of its neighbors) (Ying et al., 2018; Oh et al., 2019; Liu et al., 2020; Srinivasa et al., 2020), layer-wise sampling (sampling a subgraph from the training graph at every layer of the GNN model) (Chen et al., 2018; Zou et al., 2019; Cong et al., 2020; Huang et al., 2018) and subgraph sampling (sampling the training graph at every epoch during training) (Rong et al., 2019; Chiang et al., 2019; Zeng et al., 2019). The three kinds of methods mentioned above are ordered by the granularity of sampling operation from high to low.

Although the aforementioned sampling methods enhance the performance of various GNN models, they mainly aim to choose a series of mini-batches from $\mathscr{G}$ during training, and demand us to store the whole dataset. If we want to shrink the size of the dataset **before training**, these methods cannot help us to select the proper part of the graph, since this time the information of the part we do not select is unavailable for training. How can we select the part of $\mathcal{G}$ before training to shrink the computational cost of training and guarantee the performance of the model at the same time? To answer this question, we must know which part of $\mathcal{G}$ is crucial for training GNN models.

The overarching goal of this paper is to answer the above question from the perspective of edge by finding the edges which are crucial for training and selecting them as a subset set of $\mathcal{E}$ to make $\mathcal{G}$ more sparse before training GNN. We propose an efficient sampling algorithm called **Graph Diffusion Sampling** (GDS) which calculates the importance of every edge and samples the edges with high importance for training. GDS only need some simple computations on the original dataset $\mathcal{G}$ and it guarantees the performance of different types of models in both transductive and inductive tasks. In general, the model will perform worse when trained on a smaller dataset with the same training procedure, however, we empirically show that models can even perform better when trained on a subgraph sampled by GDS than trained on a full graph. Inspired by a classical manifold learning method called *diffusion map* (Nadler et al., 2005), the importance of edges can be measured by the extra information they bring relative to the feature matrix $\mathbf{X}$. Distinguished from existing sampling methods that design sampling strategies for variance reduction (Chen et al., 2018; Cong et al., 2020; Huang et al., 2018) or preserving graph spectral information (Srinivasa et al., 2020), the extra information is the only criterion of GDS to select edges from $\mathcal{E}$.

The main contributions of this study are as follows: (1) We analyze and quantify the importance of every edge for training GNN. (2) We propose the GDS algorithm for shrinking the edge set $\mathcal{E}$ before training which has a time and space complexity linear to the number of edges. (3) We empirically show that GDS does guarantee the performance of various baseline models on both transductive and inductive node classification tasks even when only a small fraction of the edges is sampled.

2 RELATED WORKS

2.1 GRAPH REPRESENTATION LEARNING

Research on graph representation has gained increasing attention in recent years (Chen et al., 2020b). By training graph neural networks, we obtain low-dimensional embeddings of nodes, edges or whole graphs which can be used as the input for downstream tasks such as node classification, edge classification and link prediction. There have been a variety of GNN models for graph representation learning which have achieved excellent performance on mining and learning graph-structured data. DeepWalk (Perozzi et al., 2014) embeds nodes of the graph by implementing a skip-gram model on random walks which is regarded as sequences of nodes (Mikolov et al., 2013). Graph Convolution Networks (GCN) (Mikolov et al., 2013) introduces a convolution operator induced by the graph Laplacian which aggregates the embeddings of neighbors of every node. GraphSAGE (Hamilton et al., 2017) extends GCN by allowing different kinds of aggregator functions. GraphSAGE alters the formula of graph convolution so that it can be applied to inductive tasks. Graph Attention Networks (GAT) (Veličković et al., 2017) makes use of an attention mechanism (Vaswani et al., 2017) to adaptively adjust the weight distribution of neighbor aggregations for every node. Jumping Knowledge Networks (JKNet) (Xu et al., 2018) enables the embedding of nodes using a structure-aware strategy. The above models, however, can significantly increase the computational cost when the size of the graph is large without a proper sampling method.

2.2 SAMPLING METHODS FOR TRAINING GRAPH NEURAL NETWORKS

Depending on the granularity of the sampling operation, we can divide the existing sampling methods acting on homogeneous graphs into three categories: node-wise sampling, layer-wise sampling and subgraph sampling (Liu et al., 2021). Here we summarize some representative sampling methods for each category.

2.2.1 NODE-WISE SAMPLING METHODS

In GraphSAGE, we can uniformly sample a part of neighbors for node embeddings every time we execute the neighbor-aggregating operation for every node. By controlling the size of every sample set, we can alleviate the computational burden which increases dramatically as the number of layers increases. Oh et al. (2019) extends this method by treating the neighbor selection problem as a reinforcement learning problem. In this work, a non-linear regressor is trained by reinforcement learning for calculating a real-valued importance measure of a neighborhood. Liu et al. (2020) also applies reinforcement learning for sampling with an objective to reduce the variance. Motivated by the graph sparsification theory (Spielman & Srivastava, 2011), Srinivasa et al. (2020) proposes a neighbor sampling algorithm called FastGAT. FastGAT implements an importance sampling to the neighbors of every node based on the effective resistances before computing attention to reduce the burden of computation. According to Spielman & Srivastava (2011), the effective resistances of edges indicate how important each edge is for preserving the spectral information of the graph.

2.2.2 LAYER-WISE SAMPLING METHODS

FastGCN (Chen et al., 2018) samples a certain number of nodes in each layer of GCN model, where the sampling probability distribution minimizes the variance caused by sampling. AS-GCN (Huang et al., 2018) introduces an adaptive sampling strategy that is implemented sequentially from layer to layer in GCN model. The adaptive part of the sampling method reduces the variance of the sampling. MVS-GNN (Cong et al., 2020) also aims to reduce the sampling variance. MVS-GNN uses history embeddings of the previous training epoch to approximate the embeddings of the new epoch. The deviation caused by this approximation is decomposed into the embedding approximation variance and the stochastic gradient variance. The former is negligibly small whereas the latter is reduced by optimizing the sampling probability distribution.

2.2.3 SUBGRAPH SAMPLING METHODS

DropEgde (Rong et al., 2019) uniformly samples a certain number of edges from the graph for training GNN at every epoch of the training procedure. Theoretically, DropEdge can fix the issues of over-smoothing (Li et al., 2018) and over-fitting. GraphSAINT (Zeng et al., 2019) generates a series of subgraphs by implementing importance sampling on the graph for training GCN. Before training, GraphSAINT calculates the normalization coefficients of every node and edge based on their frequency of occurrence in the previously sampled subgraph . The normalization coefficients include aggregator normalization and loss normalization that are used to reduce the bias of the forward propagation and the loss function respectively. Cluster-GCN (Chiang et al., 2019) is an improved GCN model which utilizes a graph clustering algorithm called Metis (Karypis & Kumar, 1998) to partition the graph into several clusters. For every epoch of training, Cluster-GCN randomly chooses a certain number of clusters and trains the GCN on the induced subgraph of the union of the chosen clusters. The partition operation limits the embedding of every node on only a small number of clusters and avoids neighborhood expansion. It also preserves the clustering and community structure of the graph so that the performance of GCN is guaranteed.

3 PROBLEM DEFINITION

Let $\mathscr{G} = (\mathscr{V}, \mathscr{E}, \mathbf{X})$ be a directed, unweighted and attributed graph without self-loop, where $\mathscr{V}$ is the node set, $\mathscr{E} \subset \mathscr{V} \times \mathscr{V}$ is the edge set, and $\mathbf{X} = (X_u)_{u \in \mathscr{V}} \in \mathbb{R}^{n \times |\mathscr{V}|}$ is a matrix consisting of all nodes' n-dimensional feature vectors. Given the labels of a part of the nodes, we learn the embedding of the nodes by training a model on $\mathscr{G}$ through supervised learning. To reduce the cost in both space and time for training the model, we sample a subset $\hat{\mathscr{E}}$ from the edge set $\mathscr{E}$. Then we train the model on the sampled dataset $\hat{\mathscr{G}} = (\mathscr{V}, \hat{\mathscr{E}}, \mathbf{X})$. Noticed that we sample $\mathscr{E}$ once and for all, in contrast to many other sampling methods that sample $\mathscr{E}$ multiples times to get a series of mini-batches for training the model in different epochs.

Goal. Our goal is to derive a sampling method so that the model still perform well after trained on $\hat{\mathscr{G}}$ compared to the model trained on $\mathscr{G}$, even when the sample ratio $\frac{|\hat{\mathscr{E}}|}{|\mathscr{E}|}$ is low.

4 QUANTIFYING EXTRA INFORMATION BROUGHT BY THE GRAPH

The key idea of our technique is to identify which edges are more important for training the model by evaluating the extra information they bring relative to the feature of nodes.

The attributed graph $\mathscr{G}$ can be divided into two parts: the **feature part** $(\mathscr{V}, \mathbf{X})$ and the **graph part** $(\mathscr{V}, \mathscr{E})$. Ignoring the graph part, there are various ways to learn the nodes embedding. For instance, by exploiting manifold learning methods on $\mathbf{X}$, we can discover the low-dimensional manifold structure so that embed feature vectors of nodes into this low-dimension space. In this way, however, we lose useful information brought by the graph. The graph structure regularizes the model so that the embedding outputs of two connected nodes tend to be close with each other (Belkin et al., 2006). Therefore, graph structure provides us with extra information that reshapes original manifold structure of X.

4.1 DIFFUSION MAP

To quantify this extra information, we exploit a classical manifold learning method called the *diffusion map* (Nadler et al., 2005) which is introduced as follows. For a dataset with feature matrix $\mathbf{X}$, diffusion map builds a Markov random field by defining transition probability distributions on every node. First, we introduce the Gaussian kernel $K(\cdot,\cdot)$

$$K(x,y) = \exp\left(-\frac{\|x-y\|^2}{\varepsilon}\right), x, y \in \mathbb{R}^n,$$

where ε is a positive constant. This is a popular kernel function which can be used to measure the similarity between two real vectors. Let the similarity between two nodes u and v as $s_{u,v} = K(X_u, X_v)$, and define the similarity matrix $S = (s_{u,v})_{u,v \in \mathscr{V}}$, which collects the similarities of all pairs of nodes. S can be interpreted as a weighted graph denoted as $G_{\mathbf{X}}$. To control the cost of computation on the matrix in the subsequent process, we introduce receptive fields for every node. Given a node u, its receptive field denoted as $\mathscr{F}_u$ is a subset of $\mathscr{V}$, and we redefine the similarity between u and other nodes in $\mathscr{V}$ as

$$s_{u,v} = \begin{cases} K(X_u, X_v), & v \in \mathscr{F}_u \\ 0, & v \notin \mathscr{F}_u \end{cases}.$$

In this way, the corresponding similarity matrix S and the graph $G_{\mathbf{X}}$ are more sparse. In the diffusion map, $\mathscr{F}_u$ is either the k-nearest neighbors of u or the nodes inside the r-ball of u, where r is a positive parameter. We apply the latter so that G_X is symmetric and assume that r is not too small so that G_X is connected.

Given a node u with a feature vector X_u, the probability of transitioning from u to another node v with a feature vector X_V is calculated as: $p(v \mid u) = \frac{s_{u,v}}{z_u}$, where $Z_u = \sum_{v \in \mathscr{F}_u} s_{u,v}$ is the normalization term at the node u. Obviously, $\mathscr{F}_u$ is the set of nodes reachable from u within one step and when $v \notin \mathscr{F}_u$, the transition probability $p(v \mid u)$ is zero. Collecting the transition probability between all pairs of nodes, we get the (one-step) probability transition matrix M, which determines a Markov random field denote as M. Since G_X is connected by definition, this Markov random field is irreducible. Define the stationary distribution as $\mathbf{q} = (\mathbf{q}_v)_{v \in \mathscr{V}}$ satisfying $\mathbf{q}^T M = \mathbf{q}$. If a Markov random field is irreducible and has finite states, then there exists a unique stationary distribution $\mathbf{q}$, and $\mathbf{q}_v$ is strictly positive for all $v \in \mathscr{V}$. By some calculations, we have $\mathbf{q} = \frac{1}{\sum_{v \in \mathscr{V}} z_v}(z_v)_{v \in \mathscr{V}}$ satisfies $\mathbf{q}^T M = \mathbf{q}$. Therefore, $\mathbf{q}$ is the stationary distribution of M. By the property of Markov chain, the t-step probability transition matrix is M^t. The elements of M^t are the t-step transition probability written as $p_t(v \mid u)$. Then we define the *diffusion distance* between two nodes u and u' as

$$D_t^2(u,u') = \sum_{v \in \mathscr{V}} \frac{(p_t(v \mid u) - p_t(v \mid u'))^2}{\mathbf{q}_v},$$

which serves as a measure of similarity between two points in the observation space. Next, we approximate the diffusion distance by some matrix theories. Define a diagonal matrix $D = (d_{u,v})$ with $d_{u,u} = Z_u$. Then we have the relation: $M = D^{-1}S$. Since S is symmetric, $W = D^{-\frac{1}{2}} S D^{-\frac{1}{2}}$ is also symmetric. There exist an orthogonal matrix U and diagonal matrix Λ such that $W = U \Lambda U^T$. And then, we have $M = D^{-\frac{1}{2}} U \Lambda U^T D^{\frac{1}{2}}$. Denote $D^{\frac{1}{2}} U = \Phi = (\Phi_u)_{u \in \mathscr{V}}$ and $D^{-\frac{1}{2}} U = \Psi = (\Psi_u)_{u \in \mathscr{V}}$, where Φ_u and Ψ_u are the column vectors of Φ and Ψ respectively. Then Φ_u is the left eigenvectors of M, and Ψ_u is the right eigenvectors of M. Without loss of generality, let $\Phi_0 = \mathbf{q}$. Then, it can be proved that (Nadler et al., 2005)

$$D_t^2(u,u') = \sum_{k=1}^{|\mathscr{V}|} \lambda_k^{2t} (\Psi_{u,k} - \Psi_{u',k})^2,$$

where λ_k is the eigenvalue of M related to Ψ_k and Φ_k. We approximate the $D_t^2(u,u')$ by its first m terms and define the *diffusion map* $\Psi_t^m(u) = (\lambda_k^t \Psi_{k,u})_{k=2}^{m+1}$, which maps every node to a m-dimensional vector. This map approximatively preserves the diffusion distances between different pairs of nodes and therefore preserves the essential structure of the dataset. When $m \ll n$, this map is served as a low-dimensional embedding of feature vectors.

4.2 CALCULATING THE EXTRA INFORMATION

Back to the graph dataset $\mathscr{G}$, we already have the graph structure $(\mathscr{V}, \mathscr{E})$, which can induce another Markov random field $\mathbf{M}$ with corresponding diffusion distance $\mathbf{D}_t^2(u,u')$ and diffusion map $\Psi_t'^m$, if we define a transition probability distribution for every $u \in \mathscr{V}$ as

$$\mathbf{p}_u(v) = \begin{cases} \frac{1}{d_u^{out}}, & v \in \mathscr{N}_u \\ 0, & v \notin \mathscr{N}_u \end{cases}.$$

Normally, the diffusion map obtained from $(\mathscr{V}, \mathscr{E})$ is different from the original diffusion map Ψ_t^m, and this difference provides information about how to twist Ψ_t^m to fit the graph structure. If we get a diffusion map identical to the one we get from $G_{\mathbf{X}}$, however, $(\mathscr{V}, \mathscr{E})$ does not provide any extra information for us to adjust Ψ_t^m. Motivated by this observation, we measure the extra information brought by $(\mathscr{V}, \mathscr{E})$s by evaluating the difference between $\Psi_t'^m$ and Ψ_t^m.

Since the direct comparison between two diffusion maps involves calculating eigenvalues of the whole graph and this calculation is unscalable when the dataset has a large number of nodes or edges, we compare the difference between the two Markov random fields instead. Given two distributions p_1 and p_2, the Kullback–Leibler divergence of them is define as

$$KL(p_1 \mid\mid p_2) = \mathbb{E}_{x \sim p_1}\left[\log \frac{p_1(x)}{p_2(x)}\right].$$

For $u \in \mathscr{V}$, we compute the Kullback–Leibler divergence between the transition probability distribution $\mathbf{p}_u$ induced by the graph part and the transition probability distribution p_u induced by the feature part:

$$KL(\mathbf{p}_u \mid\mid p_u) = \sum_{v \in \mathscr{N}_u^{\text{out}}} \frac{1}{d_u^{out}} \log \frac{\frac{1}{d_u^{out}}}{\frac{K(X_u,X_v)}{z_u}} = \frac{1}{d_u^{out}} \sum_{v \in \mathscr{N}_u^{\text{out}}} (\log \frac{z_u}{d_u^{out}} - \log K(X_u,X_v)).$$

Noticed that we have made the assumption that $\mathscr{N}_u \subset \mathscr{F}_u$, since if there exist $v \in \mathscr{F}_u \cap \mathscr{N}_u^c$, $KL(p_u \,||\, \mathbf{p}_u)$ is infinite. We define the extra information of the whole graph part as

$$I_G = \sum_{u \in \mathscr{V}} KL(\mathbf{p}_u \,||\, p_u) = \sum_{u \in \mathscr{V}} \frac{1}{d_u^{out}} \sum_{v \in \mathscr{N}_u^{\text{out}}} (\log \frac{z_u}{d_u^{out}} - \log K(X_u, X_v)).$$

Theorem 1. *If $I_G = 0$, then diffusion distances induced by both $(\mathscr{V}, \mathbf{X})$ and $(\mathscr{V}, \mathscr{E})$ are the same, namely*

$$D_t^2(u,u') = \mathbf{D}_t^2(u,u'), \forall t \in \mathbb{Z}^+, \forall u, u' \in \mathscr{V}$$

Proof. If $I_G = 0$, then $KL(p_u \,||\, \mathbf{p}_u) = 0$ for all $u \in \mathscr{V}$ by the non-negativity of Kullback–Leibler divergence. Then p_u is identical to $\mathbf{p}_u$ by the property of Kullback–Leibler divergence. Therefore, M and $\mathbf{M}$ are the same, and then the diffusion distances induced by them are identical. □

5 EVALUATING IMPORTANCE OF EDGES

5.1 COMPLEXITY ISSUE

We have quantified the extra information brought by the graph part of the data set as I_G. Given the budget of sampling number of edges E_b, one of the natural ways to utilize this indicator for sampling edge set is to solve the constrained optimization problem

$$\begin{cases} \max_{\hat{\mathscr{E}} \subset \mathscr{E}} I_{\hat{\mathscr{G}}}, & \hat{\mathscr{G}} = (\mathscr{V}, \hat{\mathscr{E}}, \mathbf{X}) \\ \quad \text{subject to} & \left|\hat{\mathscr{E}}\right| \leq E_b \end{cases},$$

where $\hat{\mathscr{E}}$ is the set of edges sampled from $\mathscr{E}$, and $I_{\hat{\mathscr{G}}}$ is the extra information of the sampled subgraph. This is an NP-hard problem since it involves subset selection. One observation is that we can decompose this problem into $|\mathscr{V}|$ optimization problems if we know the exact degree distribution of the solution. Denote $\hat{d}_u^{out}$ as the out-degree of node $u \in \mathscr{V}$ in $\hat{\mathscr{G}}$ which is one of the solutions, then the optimization problem is decoupled as

$$\begin{cases} \max_{\hat{\mathscr{N}}_u \subset \mathscr{N}_u} KL(p_u \,||\, \mathbf{p}_u) = \frac{1}{\hat{d}_u^{out}} \sum_{v \in \hat{\mathscr{N}}_u^{\text{out}}} (\log \frac{z_u}{\hat{d}_u^{out}} - \log K(X_u, X_v)) \\ \quad \text{subject to} \quad \left|\hat{\mathscr{N}}_u^{\text{out}}\right| = \hat{d}_u^{out} \end{cases}, \forall u \in \mathscr{V}.$$

In this way, the complexity of optimization decreases from $O(2^{|\mathscr{E}|})$ to $O(\sum_{u \in \mathscr{V}} 2^{|\hat{d}_u^{out}|})$. However, the cost of solving these problems is still unacceptable when there are many nodes with high degree.

To overcome the problem of complexity, we propose the edge replacement method to assign every edge an importance indicator.

5.2 EDGE REPLACEMENT

To evaluate the importance of a given edge (u,v), we replace it by self-loop (u,u) and calculate the corresponding extra information $I_G{}'$. This replacement causes the variation of I_G

$$\Delta I_G = I_G{}' - I_G = \frac{1}{d_u^{out}} \log K(u,v) - \frac{1}{d_u^{out}} \log K(u,u).$$

Since $K(u,u) \geq K(u,v)$, $\Delta I_G \leq 0$. This replacement will always reduce the I_G, i.e., shrinking the extra information brought by the graph part. We define $I_{u,v} = -\Delta I_G$ as the importance of edge (u,v). The larger the $I_{u,v}$, the more extra information will decrease if we replace (u,v) by (u,u). Therefore, $I_{u,v}$ can be regarded as the relative extra information of (u,v) relative to (u,u), and the more $I_{u,v}$ is, the more preference we have to sample (u,v). By the definition, we can compute $I_{u,v}$ as following

$$\begin{aligned} I_{u,v} &= -\frac{1}{d_u^{out}} \log K(u,v) + \frac{1}{d_u^{out}} \log K(u,u) \\ &= \frac{1}{d_u^{out} \varepsilon} \|X_u - X_v\|^2 - \frac{1}{d_u^{out} \varepsilon} \|X_u - X_u\|^2 = \frac{1}{d_u^{out} \varepsilon} \|X_u - X_v\|^2 . \end{aligned}$$

5.3 SYMMETRIZATION

Although edges in $\mathscr{G}$ are direct, the effect of edge (u,v) in the training process can be bilateral. Take GAT as an instance, for every node $u \in \mathscr{V}$, we calculate the attention weights $\alpha_{u,v}, v \in \mathscr{N}_u$ as

$$\alpha_{u,v} = \frac{\exp\{\sigma(\mathbf{a}^T [W\mathbf{h}_u \,||\, W\mathbf{h}_v])\}}{\sum_{k \in \mathscr{N}_u} \exp\{\sigma(\mathbf{a}^T [W\mathbf{h}_u \,||\, W\mathbf{h}_k])\}},$$

where $\sigma(\cdot)$ is a non-linear activation function, $[\cdot \,||\, \cdot]$ is the concatenation operation, $\mathbf{a}, W$ are learnable parameters, and h_u, h_v are the embeddings of u and v respectively at the current layer. As long as one of (u,v)

and (v,u) exists, during the backward propagation, both h_u and h_v will received feedback from $\alpha_{u,v}$ or $\alpha_{v,u}$. Therefore, either (u,v) or (v,u) can effect embeddings of both u and v.

To take into consideration the bilateral effect of edges, we symmetrize $I_{u,v}$ by considering reverses of all edges in $\mathscr{E}$

$$I_{u,v}^{sym} = I'_{u,v} + I'_{v,u} = \frac{1}{d_u \varepsilon} \|X_u - X_v\|^2 + \frac{1}{d_v \varepsilon} \|X_v - X_u\|^2 = (\frac{1}{d_u \varepsilon} + \frac{1}{d_v \varepsilon}) \|X_u - X_v\|^2,$$

where $I'_{u,v}$ and $I'_{v,u}$ are the importance of (u,v) and (v,u) respectively calculated after we add reversed edge for each edge in the original graph, making $(\mathscr{V}, \mathscr{E})$ as a symmetric graph, and d_u is the degree of u.

5.4 Preventing Degeneration

Another issue is the degeneration of the importance $I_{u,v}^{sym}$. In some graph datasets, there may exist pairs of neighbor nodes that have the close or even the same features, and this will make the importance of corresponding edges vanish, regardless of different topological environments around those edges. To prevent this problem, we add a term to $I_{u,v}^{sym}$:

$$\mathbf{I}_{u,v} = I_{u,v}^{sym} + \tilde{I}_{u,v}^{sym} = (\frac{1}{d_u \varepsilon} + \frac{1}{d_v \varepsilon})(\|X_u - X_v\|^2 + \bar{D}^2),$$

where $\bar{X}$ is the mean vector of all the features, and $\tilde{I}_{u,v}^{sym}$ is the symmetrized importance of $I_{u,v}$ calculated after replacing $\|X_u - X_v\|, (u,v) \in \mathscr{E}$ by the average distance to the mean vector: $\bar{D} = \sqrt{\frac{1}{|\mathscr{V}|} \sum_{k \in V} \|X_k - \bar{X}\|^2}$. Noticed that $\bar{D}^2$ is also the conditional expectation of the squared distance between features of two nodes sampled independently from the empirical distribution $\mathscr{N}(\mu, \Sigma)$, where $\mu = \frac{1}{|\mathscr{V}|} \sum_{k \in V} X_k$ and $\Sigma = \text{diag}\{\frac{1}{|\mathscr{V}|} \sum_{k \in V} (X_k^{(i)} - \mu^{(i)})^2, i = 1, ..., n\}$:

$$\mathbb{E}_{(X,Y) \sim \mathscr{N}(\mu,\Sigma) \otimes \mathscr{N}(\mu,\Sigma)} \left[\|X - Y\|^2 \middle| X \right] = \frac{1}{|\mathscr{V}|} \sum_{k \in V} \|X_k - \bar{X}\|^2 = \bar{D}^2.$$

When we need to calculate $\mathbf{I}_{u,v}$, $\bar{D}$ can be exactly calculated or estimated by sampling a small batch of sample points. We can also compute $\bar{D}$ directly if we have prior knowledge about the distribution of features $\mathbf{X}$.

6 Our Algorithm: GDS

In this section, we present our sampling algorithm (GDS) **Graph Diffusion Sampling** based on the importance indicator $\mathbf{I}_{u,v}$. GDS applys independent edge sampling: for every edge $(u,v) \in \mathscr{E}$, we assign a sampling probability $p_{u,v}$ according to $\mathbf{I}_{u,v}$. To control the behavior of the algorithm, we introduce three parameters $p_1, p_2, q \in [0,1]$. We assign a probability p_2 to the edges with the top-q importance and a probability p_1 to other edges. The algorithm is shown in Algo. 1. Noticed that since we only need to compare the relative magnitude of the importance, we can ignore the parameter ε when computing $\mathbf{I}_{u,v}$.

Algorithm 1 Graph Diffusion Sampling

```
1: Input: The training data set $\mathscr{G} = (\mathscr{V}, \mathscr{E}, \mathbf{X})$, the average distance $\bar{D}$; sampling parameters: $p_1$, $p_2$, $q$.
2: Output: The sample set $\hat{\mathscr{E}}$.
3: $\hat{\mathscr{E}} \leftarrow \emptyset$
4: for all edge $(u,v) \in \mathscr{E}$ do
5:     $\mathbf{I}_{u,v} \leftarrow (\frac{1}{d_u} + \frac{1}{d_v})(\|X_u - X_v\|^2 + \bar{D}^2)$
6: $\theta \leftarrow$ the $(1-q)$-quantile of $\{\mathbf{I}_{u,v}, (u,v) \in \mathscr{E}\}$
7: for all edge $(u,v) \in \mathscr{E}$ do
8:     if $\mathbf{I}_{u,v} \geq \theta$ then
9:         push the edge $(u,v)$ into the sample set $\hat{\mathscr{E}}$ with probability $p_2$.
10:    else
11:        push the edge $(u,v)$ into the sample set $\hat{\mathscr{E}}$ with probability $p_1$.
```

By adjusting p_1, p_2 and q, we can control the expected sample size. Denote the sample ratio $\rho = \frac{|\hat{\mathscr{E}}|}{|\mathscr{E}|}$. Since every edge is sampled independently, we have:

$$\mathbb{E}[\rho] = p_1(1-q) + p_2 q.$$

When $p_1 = p_2$, GDS uniformly samples every edge with probability p_1. When $p_1 = 0, p_2 = 1$, GDS simply selects edges with top-q importance without randomness. The role of p_1 and p_2 is to make the sampling soft so that every edge has chance to be sampled.

Dataset	**Type**	**Nodes**	**Edges**	**Classes**	**Node Features**
Citeseer	transductive	3327	4732	6	3703
Cora	transductive	2708	5429	7	1433
Pubmed	transductive	19717	44338	3	500
CoraFull	inductive	19793	126842	70	8710
Amazon	inductive	13752	491722	10	767
Coauther	inductive	18333	163788	15	6805
Gamble	inductive	28042	312826	2	6394

Table 1: **Datasets description.**

Dataset	**Model**	**Full Graph**	**US(50%)**	**GDS (50%)**
Cora	GCN	**79.94 (0.5)**	78.84 (1.0)	78.93 (0.4)
	GraphSAGE	76.54 (7.3)	76.02 (6.2)	**76.63 (5.0)**
	GAT	**78.91 (1.3)**	76.94 (1.7)	77.80 (1.0)
	JKNet	**78.87 (1.2)**	77.67 (2.0)	77.88(1.6)
Citeseer	GCN	**70.89 (0.5)**	69.25 (1.4)	69.30 (0.5)
	GraphSAGE	67.57 (5.8)	67.15 (4.5)	**67.99 (3.8)**
	GAT	**68.96 (0.7)**	67.23 (2.3)	68.71 (0.1)
	JKNet	64.75(1.3)	64.90(2.4)	**67.84 (1.7)**
Pubmed	GCN	79.28 (0.4)	76.07 (1.4)	**79.58 (0.1)**
	GraphSAGE	76.56 (6.5)	73.71 (7.1)	**77.43 (6.2)**
	GAT	77.57 (0.8)	75.23 (1.7)	**78.28 (0.7)**
	JKNet	**76.66 (0.9)**	75.13 (1.8)	76.57 (1.3)

Table 2: **US means uniform sampling. We show the accuracies in % with standard deviation (over 100 independent experiments) and set the font of best results in bold.**

6.1 Complexity analysis

We analyze the time and space complexity of GDS. During the sampling procedure, time is mainly spent on calculating importance for every edge. Noticed that for an edge $(u,v) \in \mathscr{E}$, we only need d_u, d_v and $\|X_u - X_v\|^2$ for computing $\mathbf{I}_{u,v}$. Therefore, the time complexity of GDS is $O(|\mathscr{E}|)$. Since the calculations of importance are independent for different edges, it is easy to speedup GDS by parallelization on edges.

During the sampling, the intermediate results we need to store are d_u, $\|X_u - X_v\|^2$ and $\mathbf{I}_{u,v}$, where $u \in \mathscr{V}, (u,v) \in \mathscr{E}$. Altogether there are $|\mathscr{V}| + 2|\mathscr{E}|$ scalars. Therefore, the extra cost of storage during the sampling is negligible comparing to the cost of storing $\mathscr{G}$. As we mentioned in the previous section, the final storage, i.e., the size of sample set $\hat{\mathscr{E}}$ can be controlled by the sampling parameters directly. Therefore, the final space complexity is $O(\rho\,|\mathscr{E}|)$, where the expectation of ρ is determined by the sampling parameters.

7 Experiments

7.1 Experimental setup

Dataset. We evaluate the performance of GDS on six public benchmarks and one industrial dataset called *Gamble*. All the public datasets can be downloaded directly from DGL[1]. We put more details about datasets in the supplementary material. Tasks in Citeseer (Giles et al., 1998), Cora (McCallum et al., 2000), and Pubmed (Sen et al., 2008) are transductive meaning that all the nodes are used for training, while tasks in CoraFull (Bojchevski & Günnemann, 2017), Amazon (McAuley et al., 2015), Coauthor[2], and Gamble are inductive meaning that nodes for testing are unseen when training. The description of these datasets is summarized in Table 1.

Model. We train four baseline graph embedding methods: GCN, GraphSAGE, GAT, and JKNet. We have already introduced these models in Sec. 2.

Implementations. We implement all models in PyTorch and Deep Graph Library (Wang et al., 2019) with Adam optimizer (Kingma & Ba, 2014).

Goal. We design our experiments for two goals: (i) verifying the effect of GDS for both transductive tasks and inductive tasks under various models and sample ratio, (ii) providing some guides for tuning sampling parameters for different tasks and sample ratio.

7.2 Results

7.2.1 The effect of GDS on transductive tasks

We compare the accuracy of the models on test sets after training the models on the full graph, subgraph sampled by uniform sampling, and subgraph sampled by GDS, where uniform sampling means that we sample every edge from $\mathscr{E}$ independently with the same probability. Noticed that we only implement sampling methods on training sets, not on validating or test sets. Table 2 summarizes the results on three citation datasets, where we train a two-layer GCN, GraphSAGE (applying the GCN aggregator), GAT, and JKNet

[1]https://docs.dgl.ai/api/python/dgl.data.htmlnode-prediction-datasets

[2]https://www.microsoft.com/en-us/research/project/microsoft-academic-graph/

respectively for the node classification task. In these experiments, we always set $p_1 = 0, p_2 = 1$ and $q = 0.5$. We calculate the mean and standard deviation over 100 independent experiments for every combination of models and datasets. Experiments in which the sampling ratio is lower than 50% are shown in the supplementary material for the limit of space.

We have the following observations: (1) In all the situations, GDS has a better performance than uniform sampling. (2) GDS can even outperform full graph training in some cases. (3) The standard deviation of the model accuracy after training on the subgraph sampled by GDS is always the minimum compared to other sampling methods, and this means that after sampling by GDS, training is more robust.

7.2.2 THE EFFECT OF GDS ON INDUCTIVE TASKS

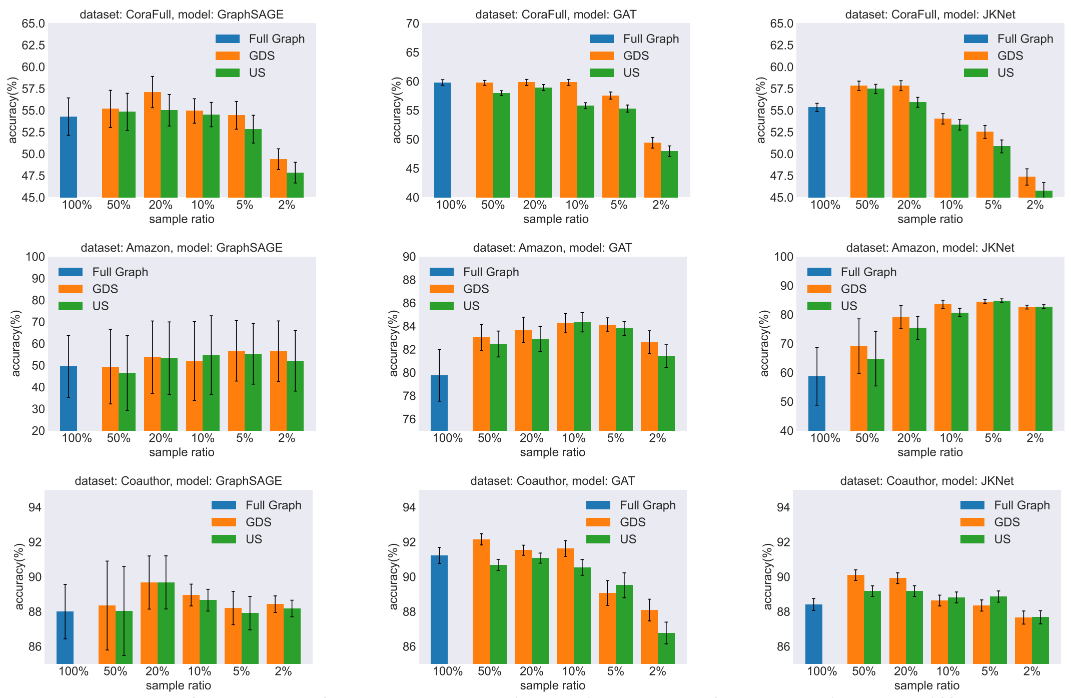

Figure 1: **The performance of models on the inductive tasks after sampling by different sampling methods under different sample ratio**.

In this section, we show the experimental results on the inductive tasks. We split every dataset by 70%, 15%, and 15% for training, validation, and test respectively. For CoraFull, Amazon, and Coauthor datasets, we train two-layer GraphSAGE, GAT, and JKNet models on the training set which is sampled by GDS and uniform sampling with varying sample ratios from 2% to 100%. For every dataset, model, and sampling method, we calculate the mean accuracy of the models on test sets with its standard deviation over 100 independent experiments. The results are shown in Fig. 1 and the detailed results are shown in the supplementary material. Fig 2 summarizes the results of the Gamble dataset, where we train a two-layer GAT model. The parameter settings of GDS for tests on the inductive tasks are shown in Table 3. It's observed that GDS consistently outperforms uniform sampling in most situations. Interestingly, both GDS and uniform sampling can have better performance than the full graph training. This phenomenon will be discussed later.

7.2.3 THE PARAMETER SETTINGS

Noticed that we set p_1, p_2, and q so that GDS is soft for selecting edges in the case of inductive tasks, different from the transductive situation. This is because, for the inductive tasks, the model only sees a part of the whole graph when training, and if graph properties such as degree distribution of the training graph are too different from the test graph due to the GDS sampling (which tends to sample edges whose endpoints have less degree), it may be more difficult for the generalization of the model on the test graph. The soft GDS sampling can be regarded as the mixed of the hard GDS and uniform sampling method, and it reduces the gap of the degree distributions between the training graph and test graph. For example, to make the expectation of the sample ratio closed to 2%, we can set the soft one $p_1 = 0.01, p_2 = 0.03, q = 0.5$ or the hard one $p_1 = 0, p_2 = 1, q = 0.98$. For the GAT model on the Amazon dataset, the soft one can achieve the mean accuracy of 82.61%, while the hard one only reaches 77.43%.

For transductive tasks, if the sample ratio is too low, we also need to make GDS soft, otherwise, there will be many isolated nodes (especially for the graph with fewer edges) which deteriorate the embeddings of nodes. For example, for GAT model on the Citeseer dataset, the soft setting $p_1 = 0.1, p_2 = 0.3, q = 0.5$ achieve the mean accuracy of 63.92%, while the hard setting $p_1 = 0, p_2 = 1, q = 0.8$ only reach 47.52%. More details are discussed in the supplementary material.

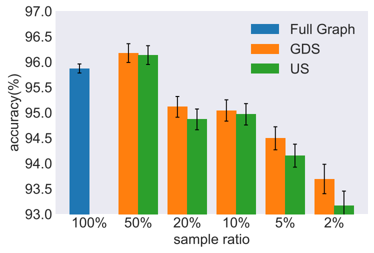

Figure 2: **The accuracy of GAT on the Gamble dataset**

sample ratio	p_1	p_2	q
50%	0.3	0.7	0.5
20%	0.1	0.3	0.5
10%	0.05	0.15	0.5
5%	0.03	0.07	0.5
2%	0.01	0.03	0.5

Table 3: **Parameter settings under different sample ratios for inductive tasks**

7.2.4 WHY MODELS CAN PERFORM BETTER WHEN TRAINING ON SAMPLED SUBGRAPHS THAN ON FULL GRAPHS

It is observe that models trained on the subgraphs can perform better than those trained on the full graph. We explain this phenomenon as follows. We first introduce the over-smoothing issue (Chen et al., 2020a). Over-smoothing is one of the main obstacles for training deep Graph Neural Networks, which make the representations of adjacent nodes converge to a stationary point when the model goes with an infinite number of layers. Therefore, the model tends to mix adjacent nodes and fails to classify nodes of different classes if they are in the same connected component of $\mathscr{G}$. Rong et al. (2019) proves that graph sparsification reduces the convergence speed of over-smoothing or relieve the information loss caused by it.

GDS makes the graph more sparse mainly by dropping the edges that are not necessary for training. In this way, GDS alleviates the over-smoothing without losing important information for training. Uniform sampling can also alleviate the over-smoothing, however, may drop edges which is important for training. Therefore, GDS can perform better than both full graph training and uniform sampling.

When there are a large number of redundant edges in the graph, uniform sampling can also sparsify the graph without losing crucial information for training, since in this case there is more probability to drop an edge which is unnecessary for training. Therefore, uniform sampling can beat full graph training in datasets where redundant edges are pervasive. Consequently, the comparison between training on full graph and subgraph sampled by uniform sampling indicates how redundant the edge set is for a given dataset.

8 DISCUSSION

Extensions. First, we can calculate the extra information of various graphlets such as triangles, four-node cycles (), cliques (), and stars () in the graph, so that we will know what kind of structures is more essential for training. Second, in the case where features are on the edges instead of nodes, we can simply aggregate (execute mean operator, for instance) the features of neighbor edges of each node and generate its feature before applying GDS. The choice of the aggregator needs further study.

Imitations. GDS works well on static graphs as the experiments shown in the last section, however, it may have little effect on dynamical graphs, where there is a timestamp on every edge. GDS does not consider the temporal information of edges (such as the order of edges on the timeline) which may be crucial for tasks in this situation (Rossi et al., 2020; Wang et al., 2021). One possible remedy is to develop a stream sampling (Ahmed et al., 2013) version of GDS. A stream sampling method takes an array of edges $\{e_n, n \in \mathbb{Z}_+\}$ ordered by their timestamps (also referred as a graph stream) as its input. At the timestep n, the method receives the edge e_n, and makes a decision to store e_n or ignore it. When GDS acts as a stream sampling method, it can make the decision of whether to sample the coming edge based on an estimated extra information of it. By applying techniques such as adaptive strategy (Ahmed & Duffield, 2019) and waiting room (Shin, 2017), we can observe temporal dependencies in edge, which may be served as the temporal information utilized by GDS. In this way, GDS is also applicable to the online learning tasks where the graph comes as a graph stream.

9 CONCLUSION

In this paper, we propose Graph Diffusion Sampling (GDS), an edge sampling algorithm for training graph neural networks (GNN). GDS calculates the importance of every edge by quantifying its extra information relative to the features information and samples the edges with high importance. Our experiments show that different types of GNN models can perform better on the subgraph sampled by GDS compared to training on the subgraph sampled by uniformly sampling and even training on the full graph. In further study, we will develop the GDS algorithm by exploring more delicate ways of quantifying extra information of a graph.

REFERENCES

Nesreen K Ahmed and Nick Duffield. Adaptive shrinkage estimation for streaming graphs. *arXiv preprint arXiv:1908.01087*, 2019.

Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. Network sampling: From static to streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(2):1–56, 2013.

Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 635–644, 2011.

Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(11), 2006.

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3438–3445, 2020a.

Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. Graph representation learning: A survey. *APSIPA Transactions on Signal and Information Processing*, 9, 2020b.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1393–1403, 2020.

C Lee Giles, Kurt D Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pp. 89–98, 1998.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 4563–4572, Red Hook, NY, USA, 2018. Curran Associates Inc.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

John Boaz Lee, Xiangnan Kong, Yihan Bao, and Constance Moore. Identifying deep contrasting networks from time series data: Application to brain network analysis. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 543–551. SIAM, 2017.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. Sampling methods for efficient training of graph convolutional networks: A survey. *arXiv preprint arXiv:2103.05872*, 2021.

Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 2077–2085, 2018.

Ziqi Liu, Zhengwei Wu, Zhiqiang Zhang, Jun Zhou, Shuang Yang, Le Song, and Yuan Qi. Bandit samplers for training graph neural networks. *arXiv preprint arXiv:2006.05806*, 2020.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pp. 43–52, 2015.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Boaz Nadler, Stephane Lafon, Ronald R Coifman, and Ioannis G Kevrekidis. Diffusion maps, spectral clustering and eigenfunctions of fokker-planck operators. *arXiv preprint math/0506090*, 2005.

Jihun Oh, Kyunghyun Cho, and Joan Bruna. Advancing graphsage with a data-driven node sampling. *arXiv preprint arXiv:1904.12935*, 2019.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Kijung Shin. Wrs: Waiting room sampling for accurate triangle counting in real graph streams (supplementary document), 2017.

Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

Rakshith S Srinivasa, Cao Xiao, Lucas Glass, Justin Romberg, and Jimeng Sun. Fast graph attention networks using effective resistance based graph sparsification. *arXiv preprint arXiv:2006.08796*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, et al. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 2628–2638, 2021.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462. PMLR, 2018.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983, 2018.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *arXiv preprint arXiv:1911.07323*, 2019.