Chapter 6
Graph Neural Networks: Scalability

Hehuan Ma, Yu Rong, and Junzhou Huang

Abstract Over the past decade, Graph Neural Networks have achieved remarkable success in modeling complex graph data. Nowadays, graph data is increasing exponentially in both magnitude and volume, e.g., a social network can be constituted by billions of users and relationships. Such circumstance leads to a crucial question, how to properly extend the scalability of Graph Neural Networks? There remain two major challenges while scaling the original implementation of GNN to large graphs. First, most of the GNN models usually compute the entire adjacency matrix and node embeddings of the graph, which demands a huge memory space. Second, training GNN requires recursively updating each node in the graph, which becomes infeasible and ineffective for large graphs. Current studies propose to tackle these obstacles mainly from three sampling paradigms: node-wise sampling, which is executed based on the target nodes in the graph; layer-wise sampling, which is implemented on the convolutional layers; and graph-wise sampling, which constructs sub-graphs for the model inference. In this chapter, we will introduce several representative research accordingly.

Hehuan Ma
Department of CSE, University of Texas at Arlington, e-mail: hehuan.ma@mavs.uta.edu

Yu Rong
Tencent AI Lab, e-mail: yu.rong@hotmail.com

Junzhou Huang
Department of CSE, University of Texas at Arlington, e-mail: jzhuang@uta.edu

6.1 Introduction

Graph Neural Network (GNN) has gained increasing popularity and obtained remarkable achievement in many fields, including social network (Freeman, 2000; Perozzi et al, 2014; Hamilton et al, 2017b; Kipf and Welling, 2017b), bioinformatics (Gilmer et al, 2017; Yang et al, 2019b; Ma et al, 2020a), knowledge graph (Liben-Nowell and Kleinberg, 2007; Hamaguchi et al, 2017; Schlichtkrull et al, 2018), etc. GNN models are powerful to capture accurate graph structure information as well as the underlying connections and interactions between nodes (Li et al, 2016b; Veličković et al, 2018; Xu et al, 2018a, 2019d). Generally, GNN models are constructed based on the features of the nodes and edges, as well as the adjacency matrix of the whole graph. However, since the graph data is growing rapidly nowadays, the graph size is increasing exponentially too. Recently published graph benchmark datasets, Open Graph Benchmark (OGB), collects several commonly used datasets for machine learning on graphs (Weihua Hu, 2020). Table 6.1 is the statistics of the datasets about node classification tasks. As observed, large-scale dataset *ogbn-papers100M* contains over one hundred million nodes and one billion edges. Even the relatively small dataset *ogbn-arxiv* still consists of fairly large nodes and edges.

Table 6.1: The statistics of node classification datasets from OGB (Weihua Hu, 2020).

Scale	Name	Number of Nodes	Number of Edges
Large	ogbn-papers100M	111,059,956	1,615,685,872
Medium	ogbn-products	2,449,029	61,859,140
Medium	ogbn-proteins	132,534	39,561,252
Medium	ogbn-mag	1,939,743	21,111,007
Small	ogbn-arxiv	169,343	1,166,243

For such large graphs, the original implementation of GNN is not suitable. There are two main obstacles, 1) large memory requirement, and 2) inefficient gradient update. First, most of the GNN models need to store the entire adjacent matrices and the feature matrices in the memory, which demand huge memory consumption. Moreover, the memory may not be adequate for handling very large graphs. Therefore, GNN cannot be applied on large graphs directly. Second, during the training phase of most GNN models, the gradient of each node is updated in every iteration, which is inefficient and infeasible for large graphs. Such scenario is similar with the gradient descent versus stochastic gradient descent, while the gradient descent may take too long to converge on large dataset, and stochastic gradient is introduced to speed up the process towards an optimum.

In order to tackle these obstacles, recent studies propose to design proper sampling algorithms on large graphs to reduce the computational cost as well as increase the scalability. In this chapter, we categorize different sampling methods according to the underlying algorithms, and introduce typical works accordingly.

6.2 Preliminary

We first briefly introduce some concepts and notations that are used in this chapter. Given a graph $\mathscr{G}(\mathscr{V},\mathscr{E})$, $|\mathscr{V}| = n$ denotes the set of n nodes and $|\mathscr{E}| = m$ denotes a set of m edges. Node $u \in \mathscr{N}(v)$ is the neighborhood of node v, where $v \in \mathscr{V}$, and $(u,v) \in \mathscr{E}$. The vanilla GNN architecture can be summarized as:

$$\mathbf{h}^{(l+1)} = \sigma\left(A\mathbf{h}^{(l)}W^{(l)}\right),$$

where A is the normalized adjacency matrix, $\mathbf{h}^{(l)}$ represents the embedding of the node in the graph for layer/depth l, $W^{(l)}$ is the weight matrix of the neural network, and σ denotes the activation function.

For large-scaled graph learning, the problem is often referred as the node classification, where each node v is associated with a label y, and the goal is to learn from the graph and predict the labels of unseen nodes.

6.3 Sampling Paradigms

The concept of sampling aims at selecting a partition of all the samples to represent the entire sample distribution. Therefore, the sampling algorithm on large graphs refers to the approach that uses partial graph instead of the full graph to address target problems. In this chapter, we categorize different sampling algorithms into three major groups, which are node-wise sampling, layer-wise sampling and graph-wise sampling.

Node-wise sampling plays a dominant role during the early stage of implementing GCN on large graphs, such as *Graph SAmple and aggreGatE* (GraphSAGE) (Hamilton et al, 2017b) and *Variance Reduction Graph Convolutional Networks* (VR-GCN) (Chen et al, 2018d). Later, layer-wise sampling algorithms are proposed to address the neighborhood expansion problem occurred during node-wise sampling, e.g., *Fast Learning Graph Convolutional Networks* (FastGCN) (Chen et al, 2018c) and *Adaptive Sampling Graph Convolutional Networks* (ASGCN) (Huang et al, 2018). Moreover, graph-wise sampling paradigms are designed to further improve the efficiency and scalability, e.g., *Cluster Graph Convolutional Networks* (Cluster-GCN) (Chiang et al, 2019) and *Graph SAmpling based INductive learning meThod* (GraphSAINT) (Zeng et al, 2020a). Fig. 6.1 illustrates a comparison between three sampling paradigms. In the node-wise sampling, the nodes are sampled based on the target node in the graph. While in the layer-wise sampling, the nodes are sampled based on the convolutional layers in the GNN

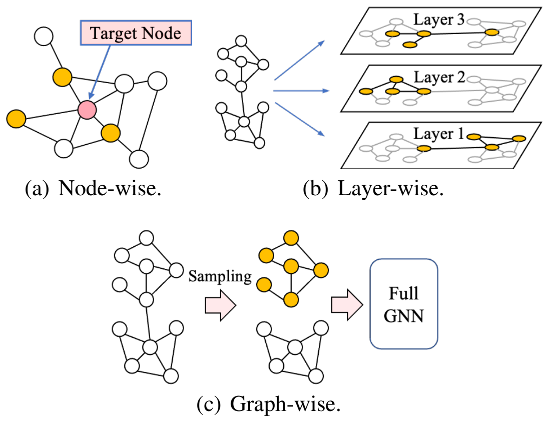


Fig. 6.1: Three sampling paradigms toward large-scale GNNs.

models. For the graph-wise sampling, the sub-graphs are sampled from the original graph, and used for the model inference.

According to these paradigms, two main issues should be addressed while constructing large-scale GNNs: 1) *how to design efficient sampling algorithms?* and 2) *how to guarantee the sampling quality?* In recent years, a lot of works have studied about how to construct large-scale GNNs and how to address the above issues properly. Fig. 6.2 displays a timeline of certain representative works in this area from the year 2017 until recent. Each work will be introduced accordingly in this chapter.

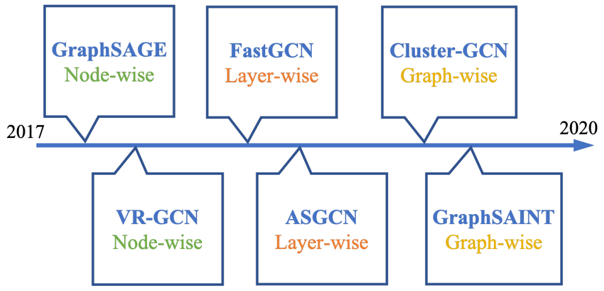


Fig. 6.2: Timeline of leading research work toward large-scale GNNs.

Other than these major sampling paradigms, more recent works have attempted to improve the scalability of large graphs from various perspectives as well. For example, heterogeneous graph has attracted more and more attention with regards to the rapid growth of data. Large graphs not only include millions of nodes but also various data types. How to train GNNs on such large graphs has become a new domain of interest. Li et al (2019a) proposes a GCN-based Anti-Spam (GAS) model to detect spams by considering both homogeneous and heterogeneous graphs. Zhang et al (2019b) designs a random walk sampling method based on all types of nodes. Hu et al (2020e) employs the transformer architecture to learn the mutual attention between nodes, and sample the nodes according to different node types.

6.3.1 Node-wise Sampling

Rather than use all the nodes in the graph, the first approach selects certain nodes through various sampling algorithms to construct large-scale GNNs. GraphSAGE (Hamilton et al, 2017b) and VR-GCN (Chen et al, 2018d) are two pivotal studies that utilize such a method.

6.3.1.1 GraphSAGE

At the early stage of GNN development, most work target at the transductive learning on a fixed-size graph (Kipf and Welling, 2017b, 2016), while the inductive setting is more practical in many cases. Yang et al (2016b) develops an inductive learning on graph embeddings, and GraphSAGE Hamilton et al (2017b) extends the study on large graphs. The overall architecture is illustrated in Fig. 6.3.

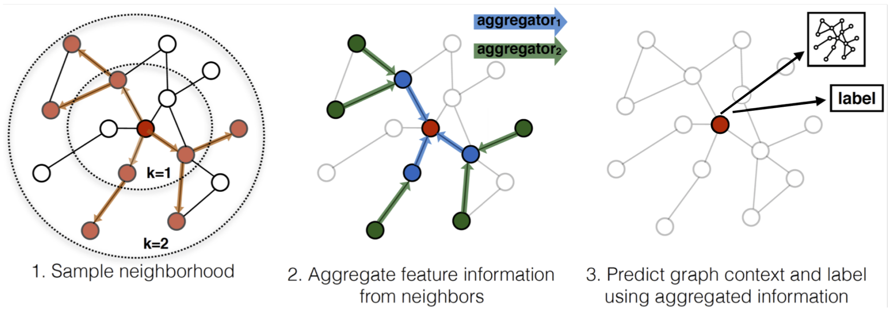

k=1
k=2
1. Sample neighborhood
aggregator$_1$
aggregator$_2$
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Fig. 6.3: Overview of the GraphSAGE architecture. Step 1: sample the neighborhoods of the target node; step 2: aggregate feature information from the neighbors; step 3: utilize the aggregated information to predict the graph context or label. Figure excerpted from (Hamilton et al, 2017b).

GraphSAGE can be viewed as an extension of the original Graph Convolutional Network (GCN) (Kipf and Welling, 2017b). The first extension is the generalized aggregator function. Given $\mathcal{G}(\mathcal{V},\mathcal{E})$, $\mathcal{N}(v)$ is the neighborhood of v, $\mathbf{h}$ is the representation of the node, the embedding generation at the current $(l+1)$-th depth from the target node $v \in \mathcal{V}$ can be formulated as,

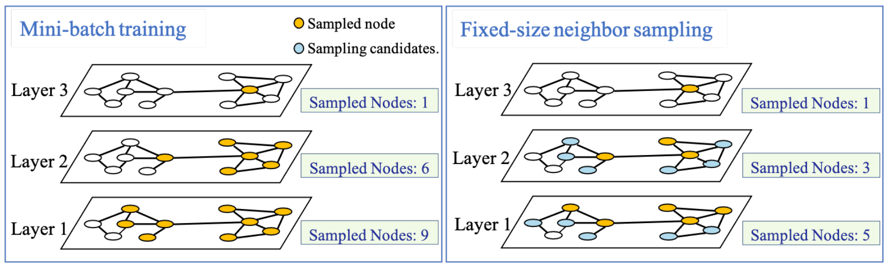
$$\mathbf{h}_{\mathscr{N}(v)}^{(l+1)} = \text{AGGREGATE}_{\,l}\left(\left\{\mathbf{h}_u^{(l)}, \forall u \in \mathscr{N}(v)\right\}\right),$$

Different from the original mean aggregator in GCN, GraphSAGE proposes LSTM aggregator and Pooling aggregator to aggregate the information from the neighbors. The second extension is that GraphSAGE applies the concatenation function to combine the information of target node and neighborhoods instead of the summation function:

$$\mathbf{h}_v^{(l+1)} = \sigma\left(W^{(l+1)} \cdot \text{CONCAT}\left(\mathbf{h}_v^{(l)}, \mathbf{h}_{\mathscr{N}(v)}^{(l+1)}\right)\right),$$

where $W^{(l+1)}$ are the weight matrices, and σ is the activation function.

In order to make GNN suitable for the large-scale graphs, GraphSAGE introduces the mini-batch training strategy to reduce the computation cost during the training phase. Specifically, in each training iteration, only the nodes that are used by computing the representations in the batch are considered, which significantly reduces the number of sampled nodes. Take layer 2 in Fig. 6.4(a) as an example, unlike the full-batch training which takes all 11 nodes into consideration, only 6 nodes are involved for mini-batch training. However, the simple implementation of mini-batch training strategy suffers the neighborhood expansion problem. As shown in layer 1 of Fig. 6.4(a), most of the nodes are sampled since the number of sampled nodes grows exponentially if all the neighbors are sampled at each layer. Thus, all the nodes are selected eventually if the model contains many layers.



(a) Mini-batch training, Batch Size=1 (b) Fix-size neighbor sampling, S=2

Fig. 6.4: Visual comparison between mini-batch training and fixed-size neighbor sampling.

To further improve the training efficiency and eliminate the neighborhood expansion problem, GraphSAGE adopts fixed-size neighbor sampling strategy. In specific, a fixed-size set of neighbor nodes are sampled for each layer for computing, instead of using the entire neighborhood sets. For example, one can set the fixed-size set as two nodes, which is illustrated in Fig. 6.4(b), the yellow nodes represent the sampled nodes, and the blue nodes are the candidate nodes. It is observed that the number of sampled nodes is significantly reduced, especially for layer 1.

In summary, GraphSAGE is the first to consider inductive representation learning on large graphs. It introduces a generalized aggregator, the mini-batch training, and fixed-size neighbor sampling algorithm to accelerate the training process. However, fixed-size neighbor sampling strategy can not totally avoid the neighborhood expansion problem. Also, there is no theoretical guarantees for the sampling quality.

6.3.1.2 VR-GCN

In order to further reduce the size of the sampled nodes, as well as conduct a comprehensive theoretical analysis, VR-GCN (Chen et al, 2018d) proposes a Control Variate Based Estimator. It only samples an arbitrarily small size of the neighbor nodes by employing historical activations of the nodes. Fig. 6.5 compares the receptive field of one target node using different sampling strategies. For the implementation of the original GCN (Kipf and Welling, 2017b), the number of sampled nodes is increased exponentially with the number of layers. With neighbor sampling, the size of the receptive field is reduced randomly according to the preset sampling number. Compared with them, VR-GCN utilizes the historical node activations as a control variate to keep the receptive field small scaled.

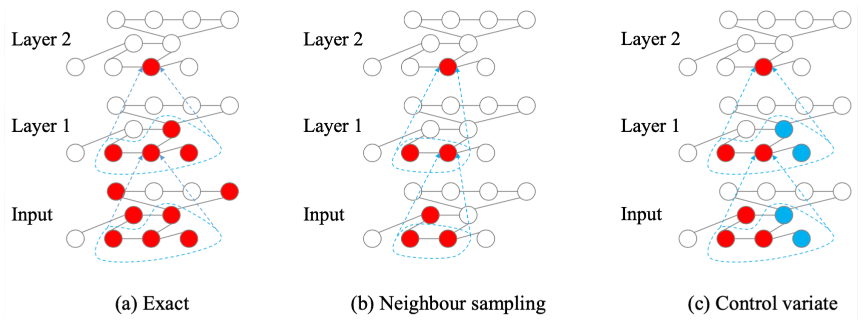


Fig. 6.5: Illustration of the receptive field of a single node utilizing different sampling strategies with a two-layer graph convolutional neural network. The red circle represents the latest activation, and the blue circle indicates the historical activation. Figure excerpted from (Chen et al, 2018d).

The neighbor sampling (NS) algorithm proposed by GraphSAGE (Hamilton et al, 2017b) can be formulated as:

$$\mathrm{NS}_v^{(l)} := R \sum_{u \in \hat{\mathcal{N}}^{(l)}(v)} A_{vu} \mathbf{h}_u^{(l)}, \quad R = \mathcal{N}(v)/d^{(l)},$$

where $\mathcal{N}(v)$ represents the neighbor set of node v, $d^{(l)}$ is the sampled size of the neighbor nodes at layer l, $\hat{\mathcal{N}}^{(l)}(v) \subset \mathcal{N}(v)$ is the sampled neighbor set of node v at layer l, and A represents the normalized adjacency matrix. Such a method has been proved to be a biased sampling, and would cause larger variance. The detailed proof can be found in (Chen et al, 2018d). Such properties result in a larger sample size $\hat{\mathcal{N}}^{(l)}(v) \subset \mathcal{N}(v)$.

To address these issues, VR-GCN proposes **C**ontrol **V**ariate Based Estimator (CV Sampler) to maintain all the historical hidden embedding $\bar{\mathbf{h}}_v^{(l)}$ of every participated node. For a better estimation, since the difference between $\bar{\mathbf{h}}_v^{(l)}$ and $\mathbf{h}_v^{(l)}$ shall be small if the model weights do not change too fast. CV Sampler is capable of reducing the variance and obtaining a smaller sample size $\hat{n}^{(l)}(v)$ eventually. Thus, the feed-forward layer of VR-GCN can be defined as,

$$H^{(l+1)} = \sigma\left(A^{(l)}\left(H^{(l+1)} - \bar{H}^{(l)}\right) + A\bar{H}^{(l)}\right)W^{(l)}.$$

$A^{(l)}$ is the sampled normalized adjacency matrix at layer l, $\bar{H}^{(l)} = \{\bar{\mathbf{h}}_1^{(l)}, \cdots, \bar{\mathbf{h}}_n^{(l)}\}$ is the stack of the historical hidden embedding $\bar{\mathbf{h}}^{(l)}$, $H^{(l+1)} = \{\mathbf{h}_1^{(l+1)}, \cdots, \mathbf{h}_n^{(l+1)}\}$ is the embedding of the graph nodes in the $(l+1)$-th layer, and $W^{(l)}$ is the learnable weights matrix. In such a manner, the sampled size of $A^{(l)}$ is greatly reduced compared with GraphSAGE by utilizing the historical hidden embedding $\bar{\mathbf{h}}^{(l)}$, which introduces a more efficient computing method. Moreover, VR-GCN also studies how to apply the Control Variate Estimator on the dropout model. More details can be found in the paper.

In summary, VR-GCN first analyzes the variance reduction on node-wise sampling, and successfully reduces the size of the samples. However, the trade-off is that the additional memory consumption for storing the historical hidden embeddings would be very large. Also, the limitation of applying GNNs on large-scale graphs is that it is not realistic to store the full adjacent matrices or the feature matrices. In VR-GCN, the historical hidden embeddings storage actually increases the memory cost, which is not helping from this perspective.

6.3.2 Layer-wise Sampling

Since node-wise sampling can only alleviate but not completely solve the neighborhood expansion problem, layer-wise sampling has been studied to address this obstacle.

6.3.2.1 FastGCN

In order to solve the neighborhood expansion problem, FastGCN (Chen et al, 2018c) first proposes to understand the GNN from the functional generalization perspective. The authors point out that training algorithms such as stochastic gradient descent are implemented according to the additivity of the loss function for independent data samples. However, GNN models generally lack sample loss independence. To solve this problem, FastGCN converts the common graph convolution view to an integral transform view by introducing a probability measure for each node. Fig. 6.6 shows the conversion between the traditional graph convolution view and the integral transform view. In the graph convolution view, a fixed number of nodes are sampled in a bootstrapping manner in each layer, and are connected if there is a connection exists. Each convolutional layer is responsible for integrating the node embeddings. The integral transform view is visualized according to the probability measure, and the integral transform (demonstrated in the yellow triangle form) is used to calculate the embedding function in the next layer. More details can be found in (Chen et al, 2018c).

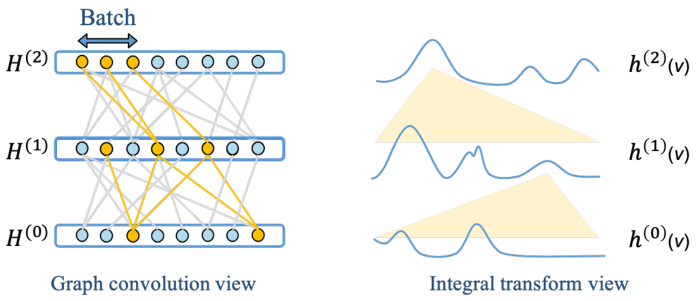


Fig. 6.6: Two views of GCN. The circles represent the nodes in the graph, while the yellow circles indicate the sampled nodes. The lines represent the connection between nodes.

Formally, given a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, an inductive graph $\mathcal{G}'$ with respect to a possibility space $(\mathcal{V}',F,p)$ is constructed. In specific, $\mathcal{V}'$ denotes the sample space of nodes which are iid samples. The probability measure p defines a sampling distribution, and F can be any event space, e.g., $F = 2^{\mathcal{V}'}$. Take node v and u with same probability measure p, $g\left(\mathbf{h}^{(K)}(v)\right)$ as the gradient of the final embedding of node v, and E as the expectation function, the functional generalization is formulated as,

$$L = \mathrm{E}_{v\sim p}\left[g\left(\mathbf{h}^{(K)}(v)\right)\right] = \int g\left(\mathbf{h}^{(K)}(v)\right) dp(v).$$

Moreover, consider sampling t_l iid samples $u_1^{(l)},\ldots,u_{t_l}^{(l)} \sim p$ for each layer l, $l = 0,\ldots,K-1$, a layer-wise estimation of the loss function is admitted as,

$$L_{t_0,t_1,\ldots,t_K} := \frac{1}{t_K}\sum_{i=1}^{t_K} g\left(\mathbf{h}_{t_K}^{(K)}\left(u_i^{(K)}\right)\right),$$

which proves that FastGCN samples a fixed number of nodes at each layer.

Furthermore, in order to reduce the sampling variance, FastGCN adopts the importance sampling with respect to the weights in the normalized adjacency matrix.

$$q(u) = \|A(:,u)\|^2 / \sum_{u' \in \mathscr{V}} \left\|A\left(:,u'\right)\right\|^2, \quad u \in \mathscr{V}, \tag{6.1}$$

where A is the normalized adjacency matrix of the graph. Detailed proofs can be found in (Chen et al, 2018c). According to Equation 6.1, the entire sampling process is independent for each layer, and the sampling probability keeps the same.

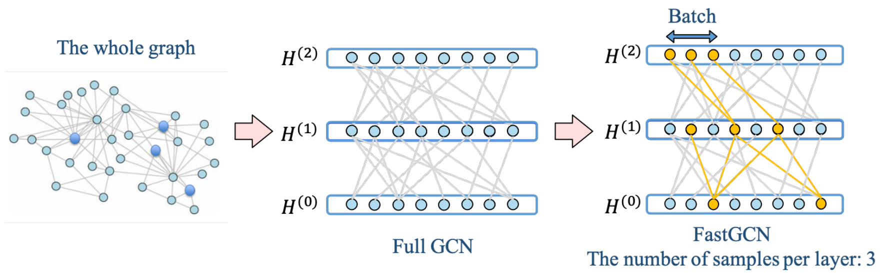


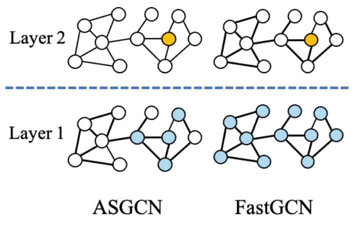
Fig. 6.7: Comparison between full GCN and FastGCN.

Compared with GraphSAGE (Hamilton et al, 2017b), FastGCN is much less computational costly. Assume t_l neighbor nodes are samples for layer l, the neighborhood expansion size is at most the sum of the t_l's for FastGCN, while could be up to the product of the t_l's for GraphSAGE. Fig. 6.7 illustrates the sampling difference between Full GCN and FastGCN. In full GCN, the connections are very sparse so that it has to compute and update all the gradients, while FastGCN only samples a fixed number of samples at each layer. Therefore, the computational cost is greatly decreased. On the other hand, FastGCN still retains most of the information according to the importance sampling method. The fixed number of nodes are randomly sampled in each training iteration, thus every node and the corresponding connections could be selected and fit into the model if the training time is long enough. Therefore, the information of the entire graph is generally retained.

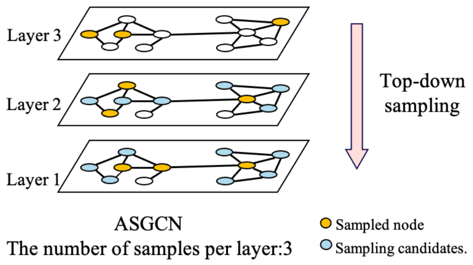
In summary, FastGCN solves the neighborhood expansion problem according to the fixed-size layer sampling. Meanwhile, this sample strategy has a quality guarantee. However, since FastGCN samples each layer independently, it failed to capture the between-layer correlations, which leads to a performance compromise.

6.3.2.2 ASGCN

To better capture the between-layer correlations, ASGCN (Huang et al, 2018) proposes an adaptive layer-wise sampling strategy. In specific, the sampling probability of lower layers depends on the upper ones. As shown in Fig. 8(a), ASGCN only samples nodes from the neighbors of the sampled node (yellow node) to obtain the better between-layer correlations, while FastGCN utilizes the importance sampling among all the nodes.



(a) ASGCN vs. FastGCN.



(b) Top-down sampling of ASGCN.

Fig. 6.8: A demonstration of the sampling strategies used in ASGCN.

Meanwhile, the sampling process of ASGCN is performed in a top-down manner. As shown in Fig. 8(b), the sampling process is first conducted in the output layer, which is the layer 3. Next, the participated nodes of the intermediate layer are sampled according to the results of the output layer. Such a sampling strategy captures dense connections between layers.

The sampling probability of lower layers depends on the upper ones. Take Fig. 6.9 as an illustration, $p(u_j \mid v_i)$ is the probability of sampling node u_j given node v_i, v_i refers to node i in the (l+1)-th layer while u_j denotes node j in the l-th layer, n' represents the sampled node number in every layer while n is the number of all the nodes in the graph, $q(u_j \mid v_1, \cdots, v_{n'})$ is the probability of sampling u_j given all the nodes in the current layer, and $\hat{a}(v_i, u_j)$ represents the entry of node v_i and u_j in the re-normalized adjacency matrix $\hat{A}$. The sampling probability $q(u_j)$ can be written as,

$$q(u_j) = \frac{p(u_j \mid v_i)}{q(u_j \mid v_1 \ldots v_{n'})}$$

$$p(u_j \mid v_i) = \frac{\hat{a}(v_i, u_j)}{\mathcal{N}(v_i)}, \quad \mathcal{N}(v_i) = \sum_{j=1}^{n} \hat{a}(v_i, u_j).$$

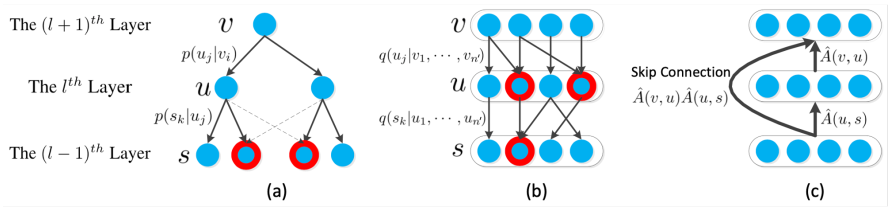


Fig. 6.9: Network construction example: (a) node-wise sampling; (b) layer-wise sampling; (c) skip connection implementation. Figure excerpted from (Huang et al, 2018).

To further reduce the sampling variance, ASGCN introduces the explicit variance reduction to optimize the sampling variance as the final objective. Consider $\boldsymbol{x}(u_j)$ as the node feature of node u_j, the optimal sampling probability $q^*(u_j)$ can be formulated as,

$$q^*(u_j) = \frac{\sum_{i=1}^{n'} p(u_j \mid v_i) \left| g(\boldsymbol{x}(u_j)) \right|}{\sum_{j=1}^{n} \sum_{i=1}^{n'} p(u_j \mid v_i) \left| g(\boldsymbol{x}(v_j)) \right|}, \quad g(\boldsymbol{x}(u_j)) = W_g \boldsymbol{x}(u_j). \tag{6.2}$$

However, simply utilizing the sampler given by Equation 6.2 is not sufficient to secure a minimal variance. Thus, ASGCN designs a hybrid loss by adding the variance to the classification loss $\mathscr{L}_c$, as shown in Equation 6.3. In such a manner, the variance can be trained to achieve the minimal status.

$$\mathscr{L} = \frac{1}{n'} \sum_{i=1}^{n'} \mathscr{L}_c(y_i, \bar{y}(\hat{\boldsymbol{\mu}}_q(v_i))) + \lambda \operatorname{Var}_q(\hat{\boldsymbol{\mu}}_q(v_i)), \tag{6.3}$$

where y_i is the ground-truth label, $\hat{\boldsymbol{\mu}}_q(v_i)$ represents the output hidden embeddings of node v_i, and $\bar{y}(\hat{\boldsymbol{\mu}}_q(v_i))$ is the prediction. λ is involved as a trade-off parameter. The variance reduction term $\lambda \operatorname{Var}_q(\hat{\boldsymbol{\mu}}_q(v_i))$ can also be viewed as a regularization according to the sampled instances.

ASGCN also proposes a skip connection method to obtain the information across distant nodes. As shown in Fig. 6.9 (c), the nodes in the (l-1)-*th* layer theoretically preserve the second-order proximity (Tang et al, 2015b), which are the 2-hop neighbors for the nodes in the (l+1)-*th* layer. The sampled nodes will include both 1-hop and 2-hop neighbors by adding a skip connection between the (l-1)-*th* layer and the (l+1)-*th* layer, which captures the information between distant nodes and facilitates the model training.

In summary, by introducing the adaptive sampling strategy, ASGCN has gained better performance as well as equips a better variance control. However, it also brings in the additional dependence during sampling. Take FastGCN as an example, it can perform parallel sampling to accelerate the sampling process since each layer is sampled independently. While in ASGCN, the sampling process is dependent to the upper layer, thus parallel processing is not applicable.

6.3.3 Graph-wise Sampling

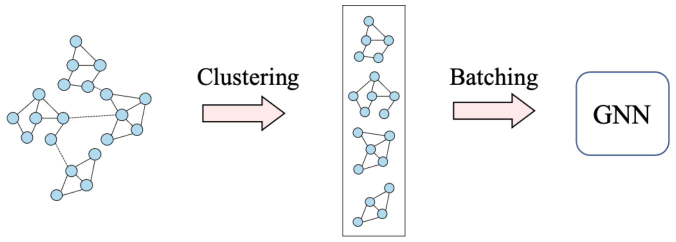


Fig. 6.10: An illustration of graph-wise sampling on large-scale graph.

Other than layer-wise sampling, the graph-wise sampling strategy is introduced recently to accomplish efficient training on large-scale graphs. As shown in Fig. 6.10, a whole graph can be sampled into several sub-graphs and fit into the GNN models, in order to reduce the computational cost.

6.3.3.1 Cluster-GCN

Cluster-GCN (Chiang et al, 2019) first proposes to extract small graph clusters based on efficient graph clustering algorithms. The intuition is that the mini-batch algorithm is correlated with the number of links between nodes in one batch. Hence, Cluster-GCN constructs mini-batch on the sub-graph level, while previous studies usually construct mini-batch based on the nodes.

Cluster-GCN extracts small clusters based on the following clustering algorithms. A graph $\mathscr{G}(\mathscr{V},\mathscr{E})$ can be devided into c portions by grouping its nodes, where $\mathscr{V}=[\mathscr{V}_1,\cdots\mathscr{V}_c]$. The extracted sub-graphs can be defined as,

$$\bar{\mathscr{G}}=[\mathscr{G}_1,\cdots,\mathscr{G}_c]=[\{\mathscr{V}_1,\mathscr{E}_1\},\cdots,\{\mathscr{V}_c,\mathscr{E}_c\}].$$

$(\mathscr{V}_t,\mathscr{E}_t)$ represents the nodes and the links within the t-th portion, $t\in(1,c)$. And the re-ordered adjacency matrix can be written as,

$$A=\bar{A}+\Delta=\begin{bmatrix}A_{11}&\cdots&A_{1c}\\ \vdots&\ddots&\vdots\\ A_{c1}&\cdots&A_{cc}\end{bmatrix};\quad \bar{A}=\begin{bmatrix}A_{11}&\cdots&0\\ \vdots&\ddots&\vdots\\ 0&\cdots&A_{cc}\end{bmatrix},\Delta=\begin{bmatrix}0&\cdots&A_{1c}\\ \vdots&\ddots&\vdots\\ A_{c1}&\cdots&0\end{bmatrix}.$$

Different graph clustering algorithms can be used to partition the graph by enabling more links between nodes within the cluster. The motivation of considering sub-graph as a batch also follows the nature of graphs, which is that neighbors usually stay closely with each other.

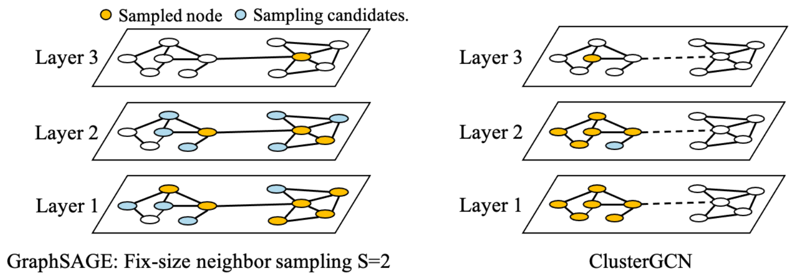


Fig. 6.11: Comparison between GraphSAGE and Cluster-GCN. In Cluster-GCN, it only samples the nodes in each sub-graph.

Obviously, this strategy can avoid the neighbor expansion problem since it only samples the nodes in the clusters, as shown in Fig. 6.11. For Cluster-GCN, since there is no connection between the sub-graphs, the nodes in other sub-graphs will not be sampled when the layer increases. In such a manner, the sampling process establishes a neighbor expansion control by sampling over the sub-graphs, while in layer-wise sampling the neighbor expansion control is implemented by fixing the neighbor sampling size.

However, there still remain two concerns with the vanilla Cluster-GCN. The first one is that the links between sub-graphs are dismissed, which may fail to capture important correlations. The second issue is that the clustering algorithm may change the original distribution of the dataset and introduce some bias. To address these concerns, the authors propose stochastic multiple partitions scheme to randomly combine clusters to a batch. In specific, the graph is first clustered into p sub-graphs; then in each epoch training, a new batch is formed by randomly combine q clusters ($q < p$), and the interactions between clusters are included too. Fig. 6.12 visualized an example when q equals to 2. As observed, the new batch is formed by 2 random clusters, along with the retained connections between the clusters.

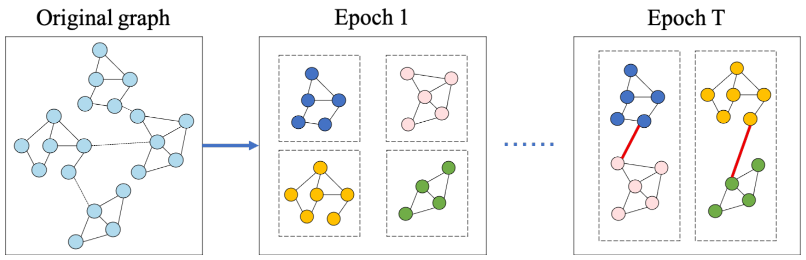
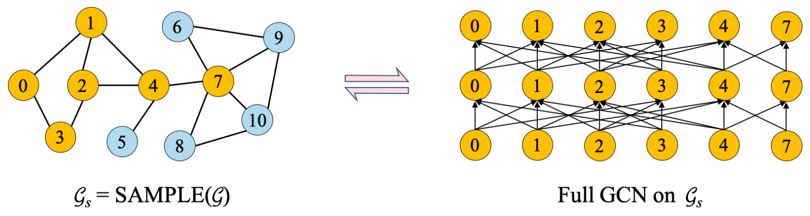


Fig. 6.12: An illustration of stochastic multiple partitions scheme.

In summary, Cluster-GCN is a practical solution based on the sub-graph batching. It has good performance and good memory usage, and can alleviate the neighborhood expansion problem in traditional mini-batch training. However, Cluster-GCN does not analyze the sampling quality, e.g., the bias and variance of this sampling strategy. In addition, the performance is highly correlated to the clustering algorithm.

6.3.3.2 GraphSAINT

Instead of using clustering algorithms to generate the sub-graphs which may bring in certain bias or noise, GraphSAINT (Zeng et al, 2020a) proposes to directly sample a sub-graph for mini-batch training according to sub-graph sampler, and employ a full GCN on the sub-graph to generate the node embeddings as well as back-propagate the loss for each node. As shown in Fig. 6.13, sub-graph $\mathscr{G}_s$ is constructed from the original graph $\mathscr{G}$ with Nodes 0, 1, 2, 3, 4, 7 included. Next, a full GCN is applied on these 6 nodes along with the corresponding connections.



0
1
2
3
4
5
6
7
8
9
10
$\mathcal{G}_s$ = SAMPLE($\mathcal{G}$)
Full GCN on $\mathcal{G}_s$

Fig. 6.13: An illustration of GraphSAINT training algorithm. The yellow circle indicates the sampled node.

GraphSAINT introduces three sub-graph sampler constructions to form the sub-graphs, which are node sampler, edge sampler and random walk sampler (Fig. 6.14). Given graph $\mathscr{G}(\mathscr{V},\mathscr{E})$, node $v \in \mathscr{V}$, edge $(u,v) \in \mathscr{E}$, the node sampler randomly samples $\mathscr{V}_s$ nodes from $\mathscr{V}$. The edge sampler selects the sub-graph based on the probability of edges in the original graph $\mathscr{G}$. The random walk sampler picks node pairs according to the probability that there exists L hops paths from node u to v.

Moreover, GraphSAINT provides comprehensive theoretical analysis on how to control the bias and variance of the sampler. First, it proposes loss normalization and aggregation normalization to eliminate the sampling bias.

$$\text{Loss normalization:} \quad \mathscr{L}_{\text{batch}} = \sum_{v \in \mathscr{G}_s} L_v/\lambda_v, \quad \lambda_v = |\mathscr{V}|p_v$$

$$\text{Aggregation normalization:} \quad a(u,v) = p_{u,v}/p_v$$

where p_v is the probability of a node $v \in \mathcal{V}$ being sampled, $p_{u,v}$ is the probability of an edge $(u,v) \in \mathcal{E}$ being sampled, L_v represents the loss of v in the output layer. Second, GraphSAINT also proposes to minimize the sampling variance by adjusting the edge sampling probability by:

$$p_{u,v} \propto 1/d_u + 1/d_v.$$

The extensive experiments demonstrate the effectiveness and efficiency of GraphSAINT, and prove that GraphSAINT converges fast as well as achieves superior performance.

In summary, GraphSAINT proposes a highly flexible and extensible framework including the graph sampler strategies and the GNN architectures, as well as achieves good performance on both accuracy and speed.

6.3.3.3 Overall Comparison of Different Models

Table 6.2 compares and summarizes the characteristics of previously mentioned models. *Paradigm* indicates the different sampling paradigms, and *Model* defers to the proposed method in each paper. *Sampling Strategy* shows the sampling theory, and *Variance Reduction* denotes whether such analysis is conducted in the paper. *Solved Problem* represents the problem that proposed model has addressed, and *Characteristic* extracts the features of the model.

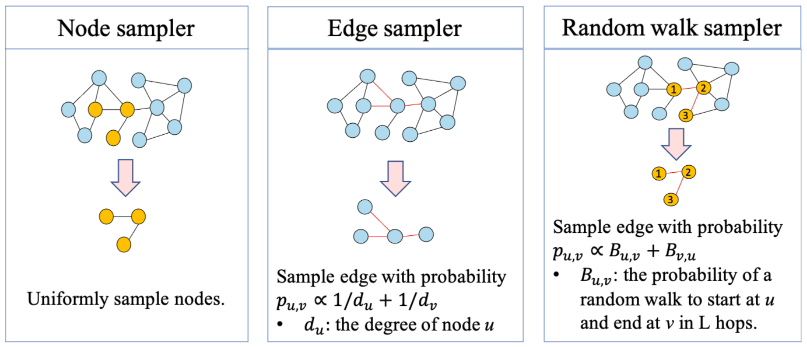

Node sampler
Uniformly sample nodes.
Edge sampler
Sample edge with probability
$p_{u,v} \propto 1/d_u + 1/d_v$
• d_u: the degree of node u
Random walk sampler
1
2
3
Sample edge with probability
$p_{u,v} \propto B_{u,v} + B_{v,u}$
• $B_{u,v}$: the probability of a random walk to start at u and end at v in L hops.

Fig. 6.14: An illustration of different samplers.

Table 6.2: The comparison between different models.

Paradigm	Model	Sampling Strategy	Variance Reduction	Solved Problem	Characteristics
Node-wise Sampling	GraphSAGE (Hamilton et al, 2017b)	Random	×	Inductive learning	Mini-batch training, reduce neighborhood expansion.
	VR-GCN (Chen et al, 2018d)	Random	✓	Neighborhood expansion	Historical activations.
Layer-wise Sampling	FastGCN (Chen et al, 2018c)	Importance	✓	Neighborhood expansion	Integral transform view.
	ASGCN (Huang et al, 2018)	Importance	✓	Between-layer correlation	Explicit variance reduction, skip connection.
Graph-wise Sampling	Cluster-GCN (Chiang et al, 2019)	Random	✓	Graph batching	Mini-batch on sub-graph.
	GraphSAINT (Zeng et al, 2020a)	Edge Probability	✓	Neighborhood expansion	Variance and bias control.

6.4 Applications of Large-scale Graph Neural Networks on Recommendation Systems

Deploying large-scale neural networks in academia has achieved remarkable success. Other than the theoretical study on *how to expand the GNNs on large graphs*, another crucial problem is *how to embed the algorithms into industrial applications*. One of the most conventional applications that demand tremendous data is the recommendation systems, which learn the user preferences and make predictions for what the users may interest in. Traditional recommendation algorithms like collaborative filtering are mainly designed according to the user-item interactions(Goldberg et al, 1992; Koren et al, 2009; Koren, 2009; He et al, 2017b). Such methods are not capable of the explosive increased web-scale data due to the extreme sparsity. Recently, graph-based deep learning algorithms have gained significant achievements on improving the prediction performance of recommendation systems by modeling the graph structures of web-scale data (Zhang et al, 2019b; Shi et al, 2018a; Wang et al, 2018b). Therefore, utilizing large-scale GNNs for recommendation has become a trend in industry (Ying et al, 2018b; Zhao et al, 2019b; Wang et al, 2020d; Jin et al, 2020b).

Recommendation systems can be typically categorized into two fields: item-item recommendation and user-item recommendation. The former one aims to find the similar items based on a user's historical interactions; while the later one directly predicts the user's preferred items by learning the user behaviors. In this chapter, we briefly introduce notable recommendation systems that are implemented on large graphs for each field.

6.4.1 Item-item Recommendation

PinSage (Ying et al, 2018b) is one of the successful applications in the early stage of utilizing large-scale GNNs on item-item recommendation systems, which is deployed on Pinterest[1]. Pinterest is a social media application that shares and discovers various content. The users mark their interested content with pins and organize them on the boards. When the users browse the website, Pinterest recommends the potentially interesting content for them. By the year 2018, the Pinterest graph contains 2 billion pins, 1 billion boards, and over 18 billion edges between pins and boards.

In order to scale the training model on such a large graph, Ying et al (2018b) proposes PinSage, a random-walk-based GCN, to implement node-wise sampling on Pinterest graph. In specific, a short random walk is used to select a fixed-number neighborhood of the target node. Fig. 6.15 demonstrates the overall architecture of PinSage. Take node A as an example, a 2-depth convolution is constructed to generate the node embedding $\mathbf{h}_A^{(2)}$. The embedding vector $\mathbf{h}_{\mathscr{N}(A)}^{(1)}$ of node A's neighbors are aggregated by node B, C, and D. Similar process is established to get the 1-hop neighbors' embedding $\mathbf{h}_B^{(1)}$, $\mathbf{h}_C^{(1)}$, and $\mathbf{h}_D^{(1)}$. An illustration of all participated nodes for each node from the input graph is shown at the bottom of Fig. 6.15. In addition, a L1-normalization is computed to sort the neighbors by their importance (Eksombatchai et al, 2018), and a curriculum training strategy is used to further improve the prediction performance by feeding harder-and-harder examples.

A series of comprehensive experiments that are conducted on Pinterest data, e.g., offline experiments, production A/B tests and user studies, have demonstrated the effectiveness of the proposed method. Moreover, with the adoption of highly efficient MapReduce inference pipeline, the entire process on the whole graph can be finished within one day.

6.4.2 User-item Recommendation

Unlike item-item recommendation, user-item recommendation systems is more complex since it aims at predicting the user's behaviors. Moreover, there remains more auxiliary information between users and users, items and items, and users and items, which leads to a heterogeneous graph problem. As shown in Fig. 6.16, there are various properties of the edges between user-user and item-item, which cannot be considered as one simple relation, e.g., user *searches* a word or *visits* a shop should be considered with different impacts.

[1] `https://www.pinterest.com/`

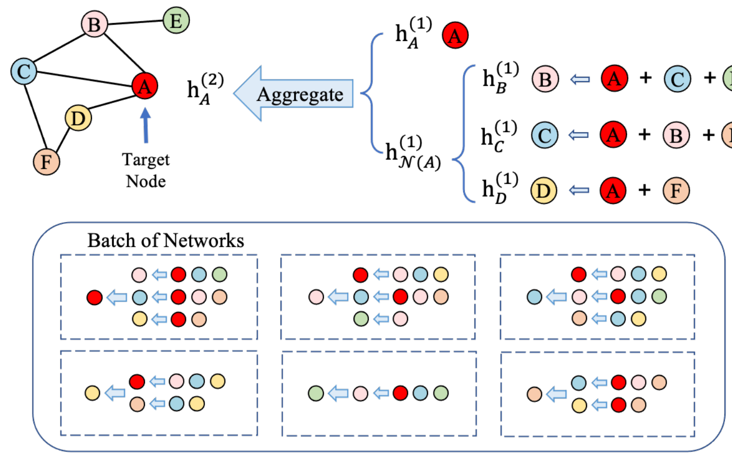


Fig. 6.15: Overview of PinSage architecture. Colored nodes are applied to illustrate the construction of graph convolutions.

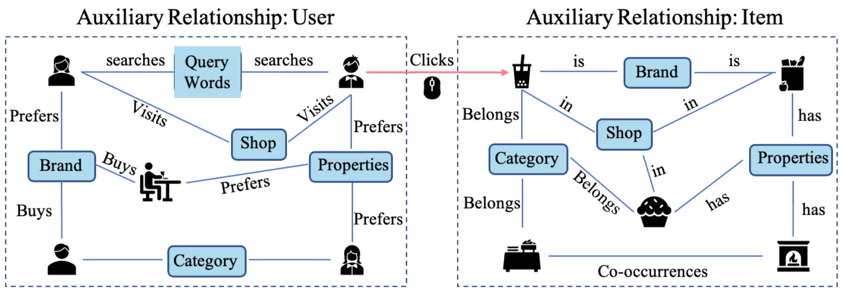


Fig. 6.16: Examples of heterogeneous auxiliary relationships on e-commerce websites.

IntentGC (Zhao et al, 2019b) proposes a GCN-based framework for large-scale user-item recommendation on e-commerce data. It explores the explicit user preferences as well as the abundant auxiliary information by graph convolutions and make predictions. E-commerce data such as Amazon contains billions of users and items, while the diverse relationships bring in more complexity. Thus, the graph structure gets larger and more complicated. Moreover, due to the sparsity of user-item graph network, sampling methods like GraphSAGE may result in a very huge sub-graph. In order to train efficient graph convolutions, IntentGC designs a faster graph convolution mechanism to boost the training, named as IntentNet.

As shown in Fig. 6.17, the bit-wise operation illustrates the traditional way of node embedding construction in GNN. In specific, consider node v as the target node, the embedding vector $\mathbf{h}_v^{(l+1)}$ is generated by concatenating the neighborhoods'

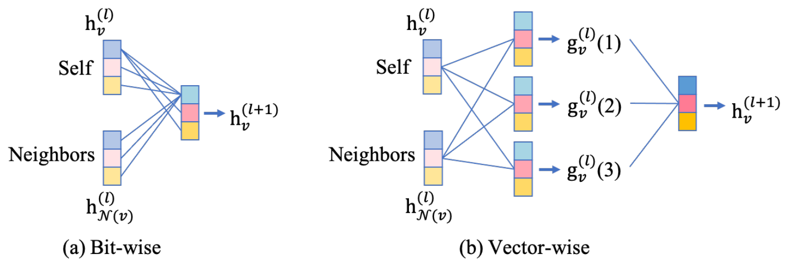


Fig. 6.17: Comparison between bit-wise and vector-wise graph convolution.

embeddings $\mathbf{h}_{\mathcal{N}(v)}^{(l)}$ and the target itself $\mathbf{h}_v^{(l)}$. Such an operation is able to capture two types of information: the interactions between target node and its neighborhoods; and the interactions between different dimensions of the embedding space. However, in user-item networks, learning the information between different feature dimensions may be less informative and unnecessary. Therefore, IntentNet designs a vector-wise convolution operation as follows:

$$\mathbf{g}_v^{(l)}(i) = \sigma\left(W_v^{(l)}(i,1)\cdot\mathbf{h}_v^{(l)} + W_v^{(l)}(i,2)\cdot\mathbf{h}_{\mathcal{N}(v)}^{(l)}\right),$$
$$\mathbf{h}_v^{(l+1)} = \sigma\left(\textstyle\sum_{i=1}^{L}\theta_i^{(l)}\cdot\mathbf{g}_v^{(l)}(i)\right),$$

where $W_v^{(l)}(i,1)$ and $W_v^{(l)}(i,2)$ are the associated weight matrices for the *i-th* local filter. $\mathbf{g}_v^{(l)}(i)$ represents the operation that learns the interactions between the target node and its neighbor nodes in a vector-wise manner. Another vector-wise layer is applied to gather the final embedding vector of the target node for the next convolutional layer. Moreover, the output vector of the last convolutional layer is fed into a three-layer fully-connected network to further learn the node-level combinatory features. Such an operation significantly promotes the training efficiency and reduces the time complexity.

Extensive experiments are conducted on Taobao and Amazon datasets, which contain millions to billions of users and items. IntentGC outperforms other baseline methods, as well as reduces the training time for about two days compared with GraphSAGE.

6.5 Future Directions

Overall, in recent years, the scalability of GNNs has been extensively studied and has achieved fruitful results. Fig. 6.18 summarizes the development towards large-scale GNNs.

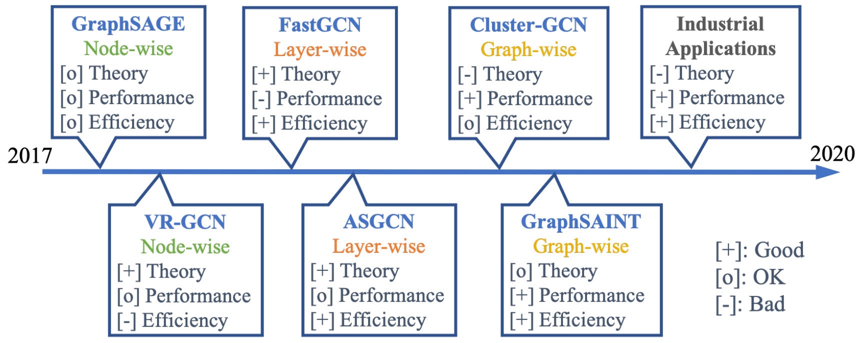


Fig. 6.18: Overall performance comparison of introduced work on large-scale GNNs.

GraphSAGE is the first to propose sampling on the graph instead of computing on the whole graph. VR-GCN designs another node sampling algorithm and provides a comprehensive theoretical analysis, but the efficiency is still limited. FastGCN and ASGCN propose to sample over layers, and both prove the efficiency with detailed analysis. Cluster-GCN first partitions the graph into sub-graphs to eliminate the neighborhood expansion problem, and boosts the performance of several benchmarks. GraphSAINT further improves the graph-wise sampling algorithm to achieve the state-of-the-art classification performance over commonly used benchmark datasets. Various industrial applications prove the effectiveness and practicability of large-scale GNNs in the real world.

However, many new open problems arise, e.g., how to balance the trade-off between variance and bias during sampling; how to deal with complex graph types such as heterogeneous/dynamic graphs; how to properly design models over complex GNN architectures. Studies toward such directions would improve the development of large-scale GNNs.

Editor's Notes: For graphs of large scale or with rapid expansibility, such as dynamic graph (chapter 15) and heterogeneous graph (chapter 16), the scalability characterization of GNNs is of vital importance to determine whether the algorithm is superior in practice. For example, graph sampling strategy is especially necessary to ensure computational efficiency in industrial scenarios, such as recommender system (chapter 19) and urban intelligence (chapter 27). With the increasing complexity and scale of the real problem, the limitation in scalability has been considered almost everywhere in the study of GNNs. Researchers devoted to graph embedding (chapter 2), graph structure learning (chapter 14) and self-supervised learning (chapter 18) put forward very remarkable works to overcome it.