1. 算法的相关指标

对于不同的采样算法，有三个方面的指标：

1. 精度
   > 高精度实际对应的就是最小化的经验误差, 经验误差可以分为bias和variance项。这里假设噪声为0
   >
   > $$E(f;D) = bias^2(\boldsymbol{x}) + var(\boldsymbol{x}) + \varepsilon^2 , \qquad (2.42)$$
   >
   > 也就是说, 泛化误差可分解为偏差、方差与噪声之和.
2. 时间
   - 每轮的时间
   - 算法的收敛性
   > 算法的收敛性这里也认为与之相关参考
3. 内存

在GraphSAGE, VRGCN, FastGCN, AS-GCN, LADIES, ClusterGCN, GraphSAINT, MVS-GNN论文中，都认为采样算法相比full-batch所带来的variance是影响精度和算法收敛性的原因。所以各自的重点都在做variance reduction.

2. 采样算法概述

按采样方式，可以将采样算法分为三类（LADIES, ClusterGCN, GraphSAINT, MVS-GNN），其中node-wise和layer-wise采样出的子图层之间的结构是不一样的，subgraph采样的子图每层之间的连接是一样的

1. node-wise:
   - GraphSAGE: 每层每个节点均匀采样出k个邻居
   - VRGCN: 认为每次参数的更新变化很小，存在以下关系$h_v^{l+1} = \sum_{w \in \mathcal{N}(v)} (\hat{h}_w^l + \triangle h_w^{(l)}), \hat{h}_w^l$可以使用历史数据，这里只需要计算$\triangle h_w^{(l)}$, 所以可以取很小的k值;
2. layer-wise:
   - FastGCN: 将$h_v^{l+1} = \sum_{w \in \mathcal{N}(v)} h_w^l$转化为积分形式，因此可以认为进行importance sampling; 考虑最小化variance, 在参数更新上加上了因子的设计
   - AS-GCN: 根据FastGCN的采样公式，试图最小化variance，然后将其中某一不可计算项设置为独立项，重新设计了loss计算公式; 考虑了加入skip connection的影响
   - LADIES: 基于Layer Dependent进行importance sampling
3. subgraph:

- ClusterGCN: 提出embedding utilization是影响计算效率的关键因素，认为聚类是提高embedding utilization的方法，采用了聚类算法进行了sampling; 为了减少variance, 采用的方法是合并不同的K个块，重新添加边
- GraphSAINT: 使用aggergater和weight update中引入参数来解决了subgraph带来的bias问题；针对variance问题，通过度数的概率设计了node sampler, edge sampler, random walk base sampler三种采样算法

现有的采样算法总的模型都是以GCN为模型进行设计，其中

$$P^l = \tilde{A}, H^{l+1} = P^l H^l W^l$$

这里P^l是将A预处理得到的结果。

算法	github地址
GraphSAGE	https://github.com/williamleif/GraphSAGE
VRGCN	https://github.com/thu-ml/stochastic_gcn
FastGCN	https://github.com/matenure/FastGCN
ASGCN	https://github.com/huangwb/AS-GCN
LADIES	https://github.com/acbull/LADIES
ClusterGCN	https://github.com/google-research/google-research/tree/master/cluster_gcn
GraphSAINT	https://github.com/GraphSAINT/GraphSAINT

2.1 GraphSAGE

a. 背景介绍

从内容推荐到识别蛋白质功能，大图形中节点的低维嵌入已被证明是非常有用的。但是，大多数现有方法要求图形中的所有节点在进行嵌入训练时都存在;这些以前的方法本质上是转导的，不会自然地泛化到看不见的节点。在这里，我们介绍 GraphSAGE，一个通用的归纳框架，它利用节点要素信息（例如文本属性）来有效地为以前不可见的数据生成节点嵌入。我们学习的不是为每个节点进行单个嵌入，而是通过采样和聚合节点的本地邻域要素来生成嵌入的函数。我们的算法在三个归纳节点分类基准上优于基准线。

b. 算法思想

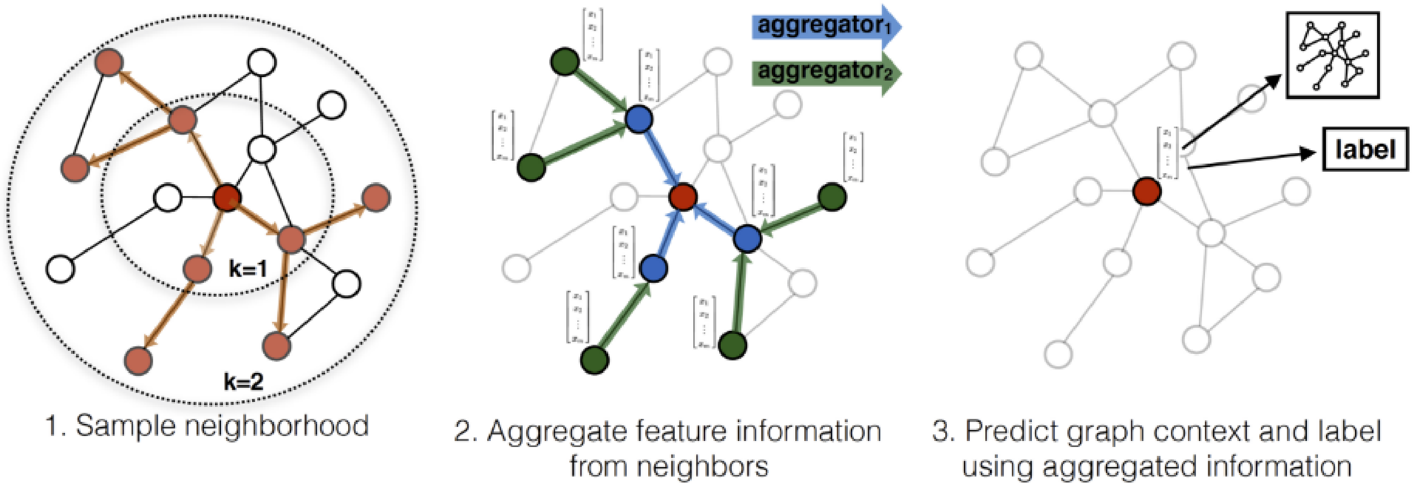

k=1
k=2
1. Sample neighborhood
aggregator1
aggregator2
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

c. 算法伪代码

Algorithm 2: GraphSAGE minibatch forward propagation algorithm

```
Input  : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$;
         input features $\{\mathbf{x}_v, \forall v \in \mathcal{B}\}$;
         depth $K$; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$;
         non-linearity $\sigma$;
         differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$;
         neighborhood sampling functions, $\mathcal{N}_k : v \rightarrow 2^{\mathcal{V}}, \forall k \in \{1, ..., K\}$
Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{B}$
1  $\mathcal{B}^K \leftarrow \mathcal{B}$;
2  for $k = K...1$ do
3      $B^{k-1} \leftarrow \mathcal{B}^k$ ;
4      for $u \in \mathcal{B}^k$ do
5          $\mathcal{B}^{k-1} \leftarrow \mathcal{B}^{k-1} \cup \mathcal{N}_k(u)$;
6      end
7  end
8  $\mathbf{h}_u^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{B}^0$ ;
9  for $k = 1...K$ do
10     for $u \in \mathcal{B}^k$ do
11         $\mathbf{h}_{\mathcal{N}(u)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_{u'}^{k-1}, \forall u' \in \mathcal{N}_k(u)\})$;
12         $\mathbf{h}_u^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_u^{k-1}, \mathbf{h}_{\mathcal{N}(u)}^k)\right)$;
13         $\mathbf{h}_u^k \leftarrow \mathbf{h}_u^k / \|\mathbf{h}_u^k\|_2$;
14     end
15 end
16 $\mathbf{z}_u \leftarrow \mathbf{h}_u^K, \forall u \in \mathcal{B}$
```

d. 实验结果

Table 1: Prediction results for the three datasets (micro-averaged F1 scores). Results for unsupervised and fully supervised GraphSAGE are shown. Analogous trends hold for macro-averaged scores.

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

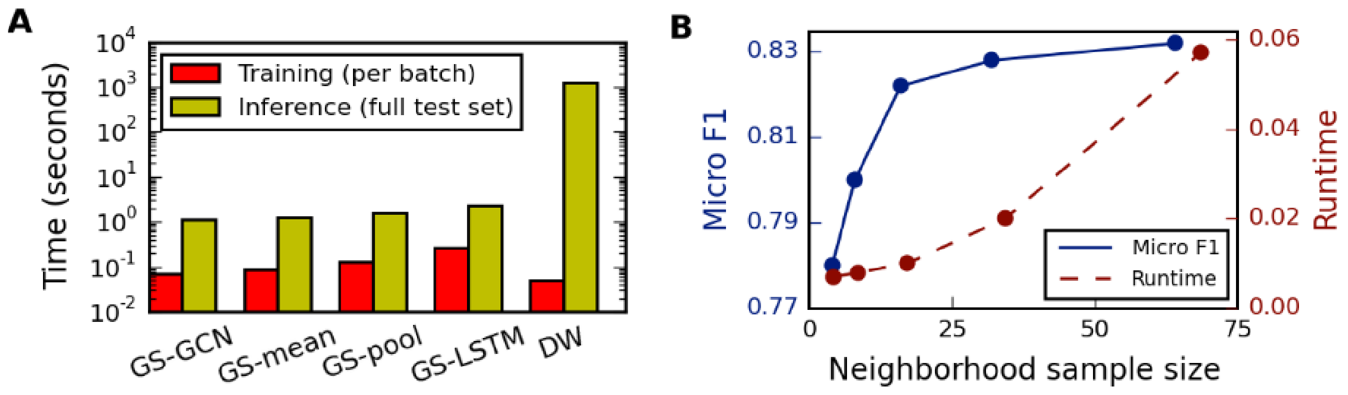


Figure 2: **A**: Timing experiments on Reddit data, with training batches of size 512 and inference on the full test set (79,534 nodes). **B**: Model performance with respect to the size of the sampled neighborhood, where the "neighborhood sample size" refers to the number of neighbors sampled at each depth for $K = 2$ with $S_1 = S_2$ (on the citation data using GraphSAGE-mean).

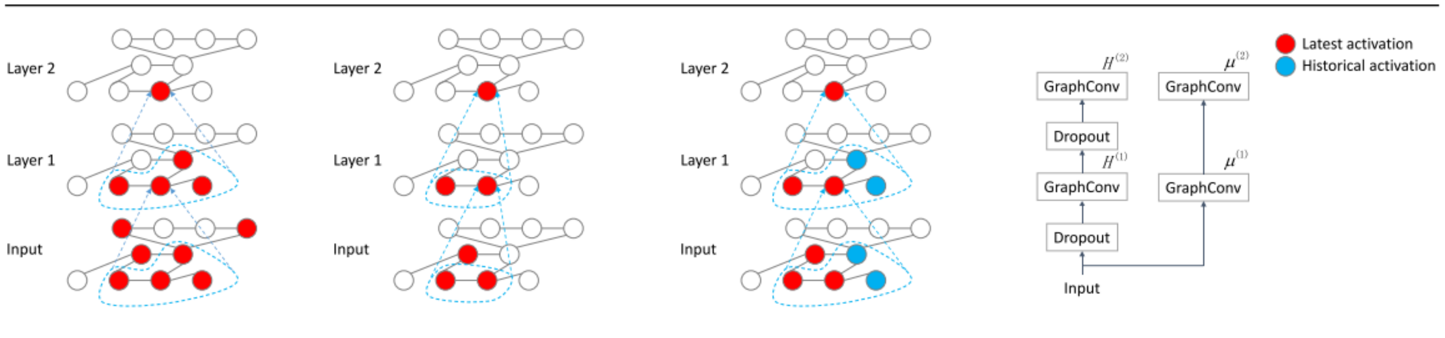
2.2 VRGCN

a. 背景介绍

GCN之前的工作都在降采样邻居的数目，但是这些算法没有保证算法的收敛性，而且他们每个节点的感受野仍然在非常大的范围。
本文提出了基于control variate控制的随机近似算法。我们利用节点的历史激活作为控制变量。我们表明，虽然 NS 估计器的方差取决于激活的幅度，但我们的算法的方差仅取决于激活与其历史值之间的差值。此外，我们的算法带来了新的理论保证。在测试时，我们的算法提供精确和零方差预测，在训练时，我们的算法收敛到 GCN 的局部最佳值，而不考虑相邻采样大小 D^l。通过每个节点仅对两个邻域进行采样，从而可以显著降低时间复杂性，但仍保留模型的质量。

最终在六个数据集上的结果显示，本文的方法能够减小NS在相同感受野的梯度的bias和variance.并且，仅仅通过使用$D^l = 2$邻居，本文的算法与精确算法一样达到了通用的预测表现，同时也表现了很好的收敛性。在Reddit数据集上，训练时间比GCN, GraphSAGE, FastGCN快了7倍以上

b. 算法原理

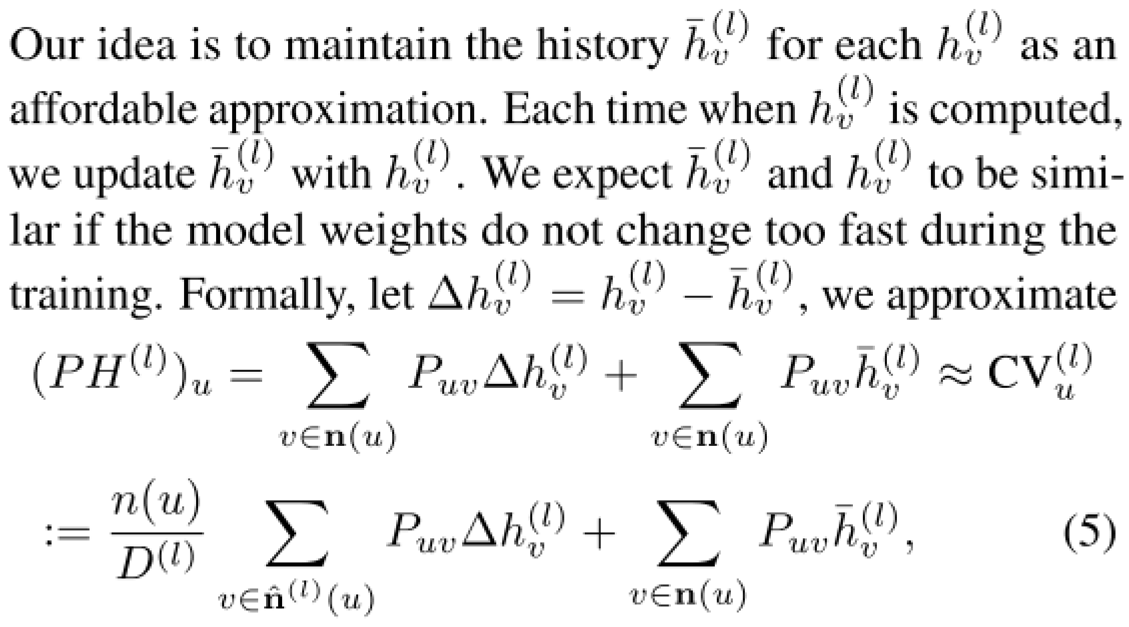


(a) Exact (b) Neighbour sampling (c) Control variate (d) CVD network

Figure 1. Two-layer graph convolutional networks, and the receptive field of a single vertex.

> - Excat: full-batch训练
> - Neigbor sampling: GraphSAGE邻居采样
> - Control variate: 论文提出的采样方法1
> - CVD network(control variate for dropout): 论文提出的采样方法2

control variate机制

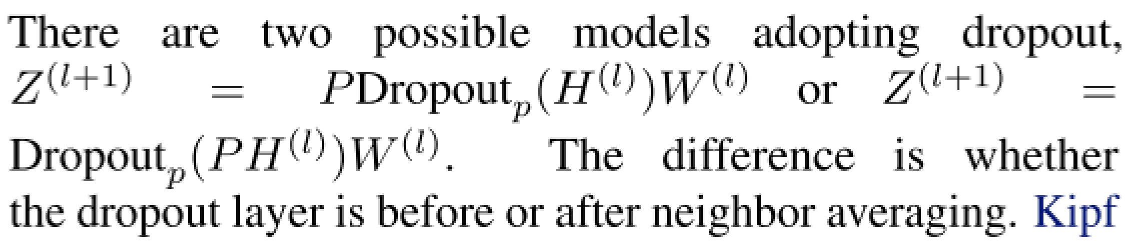


Our idea is to maintain the history $\bar{h}_v^{(l)}$ for each $h_v^{(l)}$ as an affordable approximation. Each time when $h_v^{(l)}$ is computed, we update $\bar{h}_v^{(l)}$ with $h_v^{(l)}$. We expect $\bar{h}_v^{(l)}$ and $h_v^{(l)}$ to be similar if the model weights do not change too fast during the training. Formally, let $\Delta h_v^{(l)} = h_v^{(l)} - \bar{h}_v^{(l)}$, we approximate

$$(PH^{(l)})_u = \sum_{v\in\mathbf{n}(u)} P_{uv}\Delta h_v^{(l)} + \sum_{v\in\mathbf{n}(u)} P_{uv}\bar{h}_v^{(l)} \approx \mathrm{CV}_u^{(l)}$$

$$:= \frac{n(u)}{D^{(l)}} \sum_{v\in\hat{\mathbf{n}}^{(l)}(u)} P_{uv}\Delta h_v^{(l)} + \sum_{v\in\mathbf{n}(u)} P_{uv}\bar{h}_v^{(l)}, \qquad (5)$$

preprocessing strategy(dropout)



There are two possible models adopting dropout, $Z^{(l+1)} = P\mathrm{Dropout}_p(H^{(l)})W^{(l)}$ or $Z^{(l+1)} = \mathrm{Dropout}_p(PH^{(l)})W^{(l)}$. The difference is whether the dropout layer is before or after neighbor averaging. Kipf

> 这篇论文中使用了后者

模型

Esti.	VNS	VD
Exact	0	$S_u^{(l)}$
NS	$(P_{uv_1}\mu_{v_1}^{(l)} - P_{uv_2}\mu_{v_2}^{(l)})^2$	$\frac{n(u)}{D^{(l)}} S_u^{(l)}$
CV	$(P_{uv_1}\Delta\mu_{v_1}^{(l)} - P_{uv_2}\Delta\mu_{v_2}^{(l)})^2$	$\left(3 + \frac{n(u)}{D^{(l)}}\right) S_u^{(l)}$
CVD	$(P_{uv_1}\Delta\mu_{v_1}^{(l)} - P_{uv_2}\Delta\mu_{v_2}^{(l)})^2$	$S_u^{(l)}$

Table 2. Variance of different estimators. To save space we omit $\frac{C_u^{(l)}}{2D^{(l)}} \sum_{v_1, v_2 \in \mathbf{n}(u)}$ before all the VNS terms.

c. 算法伪代码

Algorithm 1 Constructing the receptive fields and random propagation matrices.

```
$\mathbf{r}^{(L)} \leftarrow \mathcal{V}_B$
**for** layer $l \leftarrow L-1$ to $0$ **do**
    $\mathbf{r}^{(l)} \leftarrow \varnothing$
    $\hat{P}^{(l)} \leftarrow \mathbf{0}$
    **for** each node $u \in \mathbf{r}^{(l+1)}$ **do**
        $\mathbf{r}^{(l)} \leftarrow \mathbf{r}^{(l)} \cup \{u\}$
        $\hat{P}_{uu}^{(l)} \leftarrow \hat{P}_{uu}^{(l)} + P_{uu} n(u)/D^{(l)}$
        **for** $D^{(l)} - 1$ random neighbors $v \in \mathbf{n}(u)$ **do**
            $\mathbf{r}^{(l)} \leftarrow \mathbf{r}^{(l)} \cup \{v\}$
            $\hat{P}_{uv}^{(l)} \leftarrow \hat{P}_{uv}^{(l)} + P_{uv} n(u)/D^{(l)}$
        **end for**
    **end for**
**end for**
```

Algorithm 2 Training with the CV algorithm

```
for each minibatch $\mathcal{V}_B \subset \mathcal{V}$ do
    Compute the receptive fields $\mathbf{r}^{(l)}$ and stochastic propagation matrices $\hat{P}^{(l)}$
as Alg. 1.
    (Forward propgation)
    for each layer $l \leftarrow 0$ to $L-1$ do
        $Z^{(l+1)} \leftarrow \left(\hat{P}^{(l)}(H^{(l)} - \bar{H}^{(l)} + P\bar{H}^{(l)}\right) W^{(l)}$
        $H^{(l+1)} \leftarrow \sigma(Z^{(l+1)})$
    end for
    Compute the loss $\mathcal{L} = \frac{1}{|\mathcal{V}_B|}\sum_{v\in\mathcal{V}_B} f(y_v, Z_v^{(L)})$
    (Backward propagation)
    $W \leftarrow W - \gamma_i \nabla_W \mathcal{L}$
    (Update historical activations)
    for each layer $l \leftarrow 0$ to $L-1$ do
        for each node $v \in \mathbf{r}^{(l)}$ do
            $\bar{h}_v^{(l)} \leftarrow h_v^{(l)}$
        end for
    end for
end for
```

Algorithm 3 Training with the CVD algorithm

```
for each minibatch $\mathcal{V}_B \subset \mathcal{V}$ do
    Compute the receptive fields $\mathbf{r}^{(l)}$ and stochastic propagation matrices $\hat{P}^{(l)}$ as Alg. 1.
    (Forward propgation)
    for each layer $l \leftarrow 0$ to $L-1$ do
        $U \leftarrow \left(\bar{P}^{(l)}(H^{(l)} - \mu^{(l)}) + \hat{P}^{(l)}(\mu^{(l)} - \bar{\mu}^{(l)}) + P\bar{H}^{(l)}\right)$
        $H^{(l+1)} \leftarrow \sigma(\text{Dropout}_p(U)W^{(l)})$
        $\mu^{(l+1)} \leftarrow \sigma(UW^{(l)})$
    end for
    Compute the loss $\mathcal{L} = \frac{1}{|\mathcal{V}_B|}\sum_{v\in\mathcal{V}_B} f(y_v, H_v^{(L)})$
    (Backward propagation)
    $W \leftarrow W - \gamma_i \nabla_W \mathcal{L}$
    (Update historical activations)
    for each layer $l \leftarrow 0$ to $L-1$ do
        for each node $v \in \mathbf{r}^{(l)}$ do
            $\bar{h}_v^{(l)} \leftarrow h_v^{(l)}$
        end for
    end for
end for
```

d. 实验结果

Dataset	M0	M1	M1+PP
Citeseer	$70.8 \pm .1$	$70.9 \pm .2$	$70.9 \pm .2$
Cora	$81.7 \pm .5$	$82.0 \pm .8$	$81.9 \pm .7$
PubMed	$79.0 \pm .4$	$78.7 \pm .3$	$78.9 \pm .5$
NELL	-	64.9 ± 1.7	64.2 ± 4.6
PPI	$97.9 \pm .04$	$97.8 \pm .05$	$97.6 \pm .09$
Reddit	$96.2 \pm .04$	$96.3 \pm .07$	$96.3 \pm .04$

Table 3. Testing accuracy of different algorithms and models after fixed number of epochs. Our implementation does not support M0 on NELL so the result is not reported.

> M0表示$Z^{l+1} = PDropout_p H^{(l)} W^{(l)}$, M1表示$Z^{l+1} = Dropout_p PH^{(l)} W^{(l)}$, M1+PP表示sampling $D^l = 20$个邻居和预处理$PH^{(0)}$

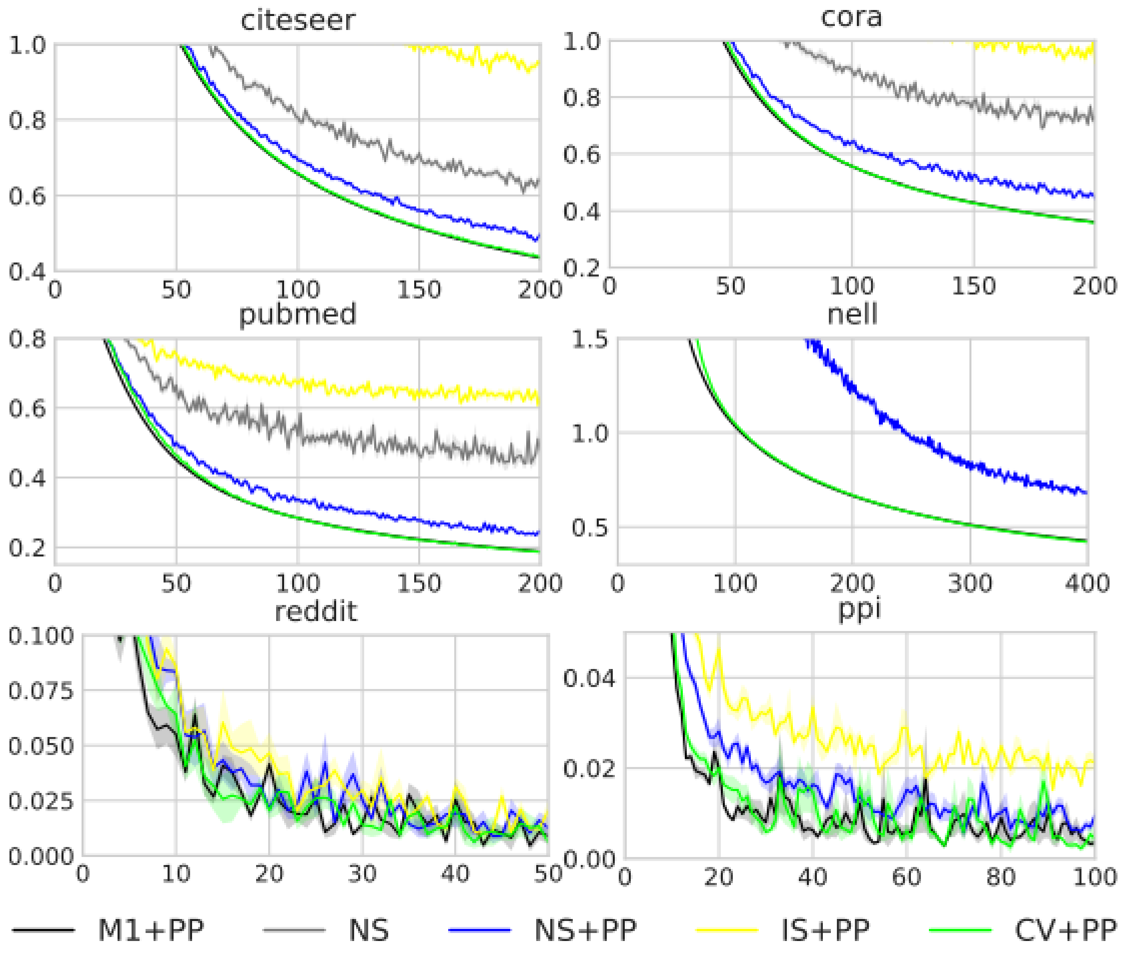


Figure 2. Comparison of training loss with respect to number of epochs without dropout. The CV+PP curve overlaps with the Exact curve in the first four datasets. The training loss of NS and IS+PP are not shown on some datasets because they are too high.

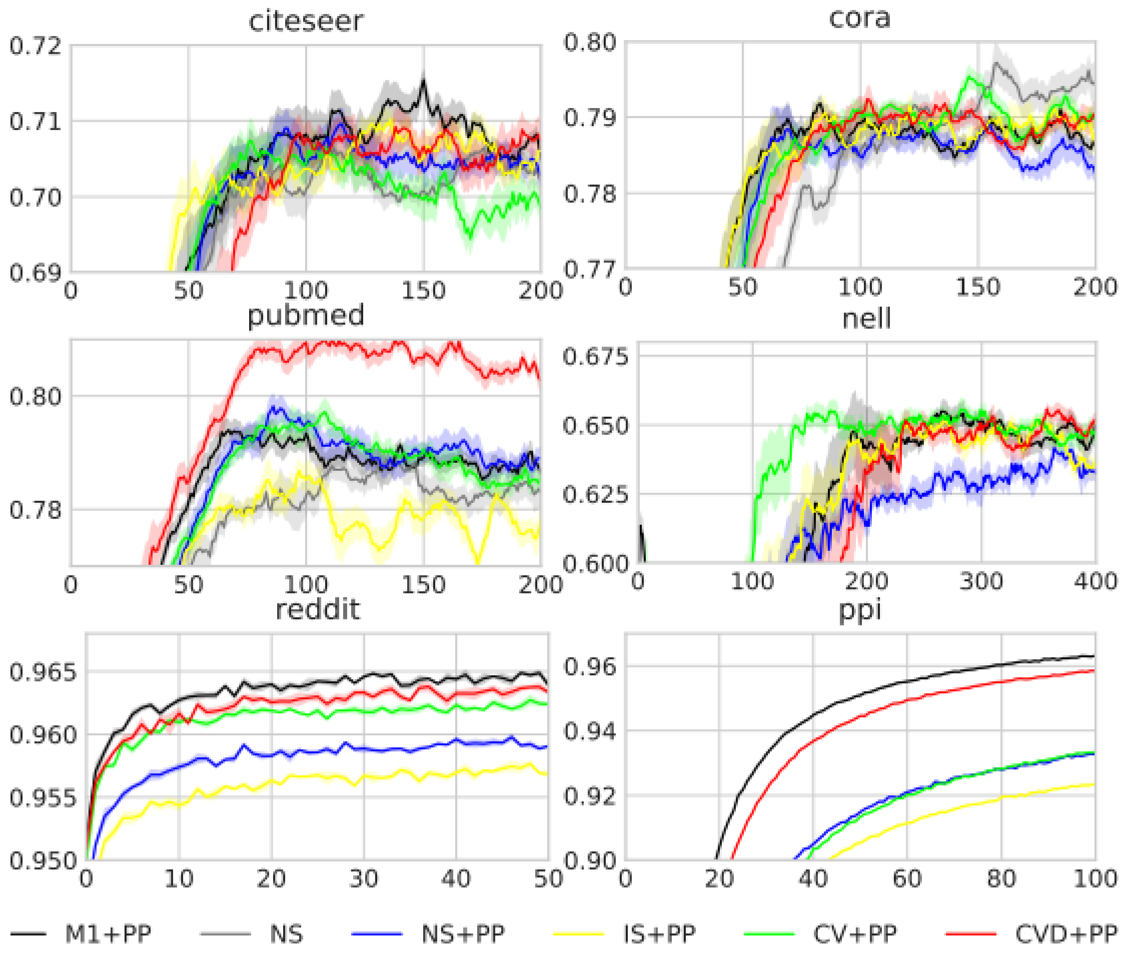


Figure 3. Comparison of validation accuracy with respect to number of epochs. NS converges to 0.94 on the Reddit dataset and 0.6 on the PPI dataset.

2.3 FastGCN

GCN面临的一个更严重的挑战是，跨层邻域的递归扩展在批处理训练中会产生昂贵的计算。特别是对于密集图形和电源法图，单个顶点的邻域扩展会快速填满图形的很大一部分。然后，通常的小型批量培训将涉及每个批次的大量数据，即使批量较小。因此，可扩展性是GCN适用于大型密集图形的紧迫问题。为了解决这两个挑战，我们建议从不同的角度查看图形卷积，并将其解释为在概率度量下嵌入函数的积分变换。这种观点为归纳学习提供了一个原则性机制，从损耗的公式到梯度的随机版本。具体来说，我们解释图形顶点是某种概率分布的iid样本，并将损耗和每个卷积层写入顶点嵌入函数的积分。然后，通过蒙特卡罗近似值计算积分，该近似值定义了样品损耗和样本梯度。可以进一步更改采样分布（如重要性采样），以减少近似方差。
本文提出的采样方法FastGCN，不仅摆脱了对测试数据的依赖，而且产生了可控制的每个批次计算成本。在撰写本文时，我们注意到一份新出版的作品《GraphSAGE》（Hamilton等人，2017年），该作品还建议使用采样来减少GCN的计算占用空间。我们的采样方案更经济，在梯度计算中节省了大量成本。

实验结果表明，FastGCN的每个批次计算速度比GraphSAGE快一个数量级以上，而分类精度则具有高度可比性。

a. 算法原理

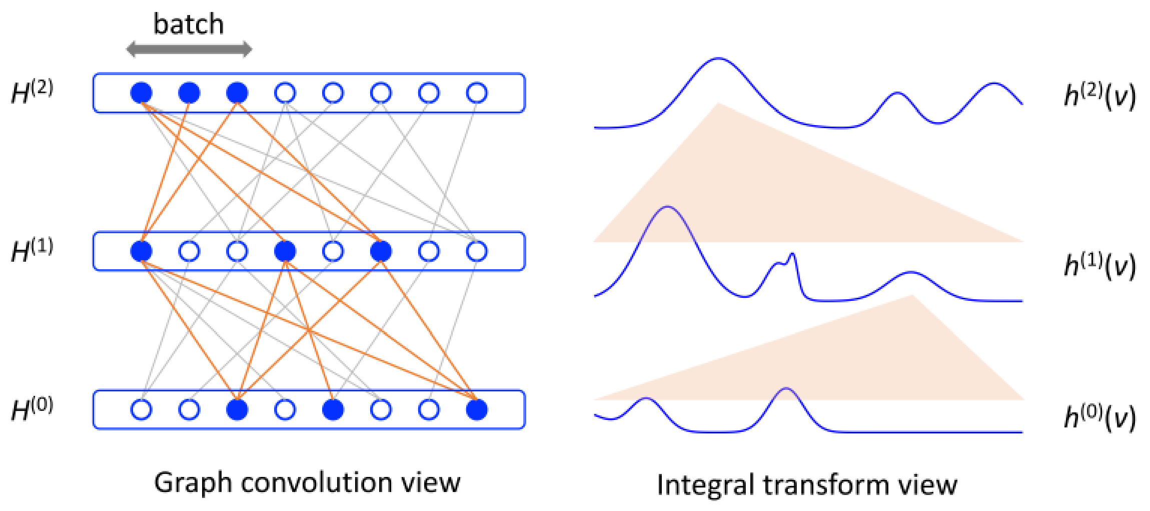


Figure 1: Two views of GCN. On the left (graph convolution view), each circle represents a graph vertex. On two consecutive rows, a circle i is connected (in gray line) with circle j if the two corresponding vertices in the graph are connected. A convolution layer uses the graph connectivity structure to mix the vertex features/embeddings. On the right (integral transform view), the embedding function in the next layer is an integral transform (illustrated by the orange fanout shape) of the one in the previous layer. For the proposed method, all integrals (including the loss function) are evaluated by using Monte Carlo sampling. Correspondingly in the graph view, vertices are subsampled in a bootstrapping manner in each layer to approximate the convolution. The sampled portions are collectively denoted by the solid blue circles and the orange lines.

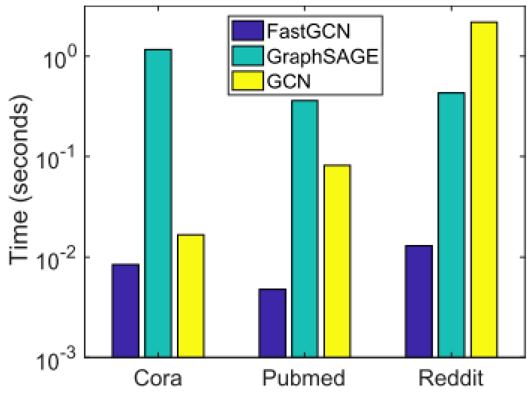
b. 算法流程

Algorithm 2 FastGCN batched training (one epoch), improved version

```
1: For each vertex $u$, compute sampling probability $q(u) \propto \|\hat{A}(:,u)\|^2$
2: for each batch do
3:     For each layer $l$, sample $t_l$ vertices $u_1^{(l)}, \dots, u_{t_l}^{(l)}$ according to distribution $q$
4:     for each layer $l$ do                ▷ Compute batch gradient $\nabla L_{\text{batch}}$
5:         If $v$ is sampled in the next layer,
               $\nabla \tilde{H}^{(l+1)}(v,:) \leftarrow \frac{1}{t_l}\sum_{j=1}^{t_l} \frac{\hat{A}(v,u_j^{(l)})}{q(u_j^{(l)})} \nabla\left\{H^{(l)}(u_j^{(l)},:)W^{(l)}\right\}$
6:     end for
7:     $W \leftarrow W - \eta\nabla L_{\text{batch}}$                ▷ SGD step
8: end for
```

> 这里$\hat{A} = P$, 红圈为variance reduction的操作

c. 实验结果



Micro F1 Score

	Cora	Pubmed	Reddit
FastGCN	0.850	0.880	0.937
GraphSAGE-GCN	0.829	0.849	0.923
GraphSAGE-mean	0.822	0.888	0.946
GCN (batched)	0.851	0.867	0.930
GCN (original)	0.865	0.875	NA

Figure 3: Per-batch training time in seconds (left) and prediction accuracy (right). For timing, GraphSAGE refers to GraphSAGE-GCN in Hamilton et al. (2017). The timings of using other aggregators, such as GraphSAGE-mean, are similar. GCN refers to using batched learning, as opposed to the original version that is nonbatched; for more details of the implementation, see the appendix. The nonbatched version of GCN runs out of memory on the large graph Reddit. The sample sizes for FastGCN are 400, 100, and 400, respectively for the three data sets.

Table 3: Further comparison of per-batch training time (in seconds) with new implementation of GraphSAGE for small graphs. The new implementation is in PyTorch whereas the rest are in TensorFlow.

	Cora	Pubmed	Reddit
FastGCN	0.0084	0.0047	0.0129
GraphSAGE-GCN (old impl)	1.1630	0.3579	0.4260
GraphSAGE-GCN (new impl)	0.0380	0.3989	NA
GCN (batched)	0.0166	0.0815	2.1731

2.4 AS-GCN

a. 背景介绍

GNNs最明显的一个挑战就是scalability, 因为计算卷积往往需要扩层领域的递归扩展，这在计算上需要大量的内存占用，即使是单个节点，领域扩展也可能将整个图包含进来。所以传统的mini-batch训练也不能加快卷积计算，因为每个batch会涉及到大量的顶点，即使batch size很小的情况下。

为了避免over-expansion问题，我们通过加速了GCNs的训练通过控制每层采样的邻居的大小。我们建立网络结构的方法为top-down, 即lower layer基于upper layer采样得到。
这样的layer-wise采样方法有两个有点：第一，我们可以重用采样到的邻居的信息；第二，这样可以固定每层的大小从而避免over-expansionentity

本文方法的核心是设计合理的sampler, 一个优化的目标就是最小化resulting variance. 不幸地，在本文设计的top-down sampling和bottom-up propagation的网络中，the optimal sampler to minimize the

variance 是不可计算的。为了解决这个问题，本文将一个不可计算的部分替代为了self-dependent function, 然后add the variance to the loss function.
其次，我们探索了如何enable efficient message passing across distant nodes. 现有的方法是resort random walkds来产生邻居的various steps, 囊荷整合到多跳邻居中。在本文中提出了一个新的机制, 在l-1层和l+1层之间skip connection，这个链接重用了l-1层的节点作为l+1层的2跳邻居。

本文的贡献：

1. 提出了新的layer-wise的采样方法, between-layer information是共享的，the size of the sampling nodes是可以控制的
2. layer-wise sampling的采样算法是adptive,可以在训练过程中进行训练
3. 提出了一个有效的两层之间的skip connection。
   本文的实验在Cora, Citeseer, Pubmed和Reddit上都取得了很好的分类精度和收敛速度

b. 算法原理

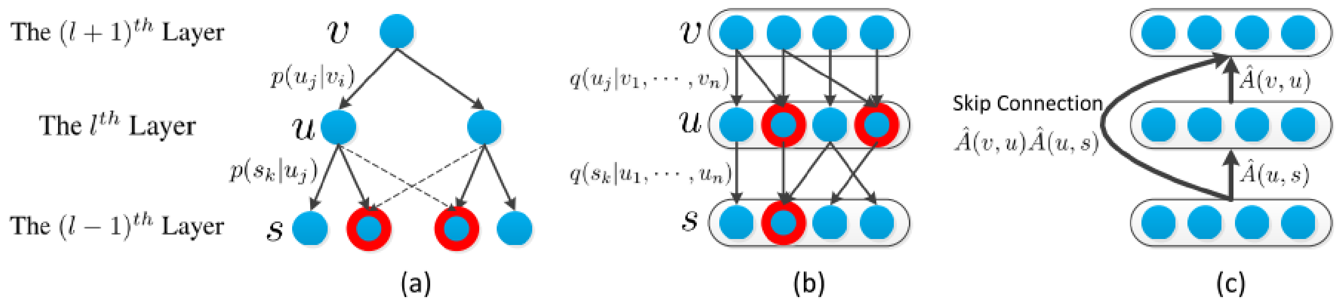


Figure 1: Network construction by different methods: (a) the node-wise sampling approach; (b) the layer-wise sampling method; (c) the model considering the skip-connection. To illustrate the effectiveness of the layer-wise sampling, we assume that the nodes denoted by the red circle in (a) and (b) have at least two parents in the upper layer. In the node-wise sampling, the neighborhoods of each parent are not seen by other parents, hence the connections between the neighborhoods and other parents are unused. In contrast, for the layer-wise strategy, all neighborhoods are shared by nodes in the parent layer, thus all between-layer connections are utilized.

Explicit Variance Reduction

Layer-Wise Sampling. We equivalently transform Eq. (2) to the following form by applying importance sampling, *i.e.*,

$$h^{(l+1)}(v_i) \quad = \quad \sigma_{W^{(l)}}(N(v_i)\mathbb{E}_{q(u_j|v_1,\cdots,v_n)}[\frac{p(u_j|v_i)}{q(u_j|v_1,\cdots,v_n)}h^{(l)}(u_j)]), \tag{4}$$

where $q(u_j|v_1,\cdots,v_n)$ is defined as the probability of sampling u_j given all the nodes of the current layer (*i.e.*, $v_1,\cdots,v_n$). Similarly, we can speed up Eq. (4) by approximating the expectation with the Monte-Carlo mean, namely, computing $h^{(l+1)}(v_i) = \sigma_{W^{(l)}}(N(v_i)\hat{\mu}_q(v_i))$ with

$$\hat{\mu}_q(v_i) \quad = \quad \frac{1}{n}\sum\nolimits_{j=1}^{n}\frac{p(\hat{u}_j|v_i)}{q(\hat{u}_j|v_1,\cdots,v_n)}h^{(l)}(\hat{u}_j), \quad \hat{u}_j \sim q(\hat{u}_j|v_1,\cdots,v_n). \tag{5}$$

We term the sampling in Eq. (5) as the *layer-wise sampling* strategy. As opposed to the node-wise method in Eq. (3) where the nodes $\{\hat{u}_j\}_{j=1}^{n}$ are generated for each parent v_i independently, the sampling in Eq. (5) is required to be performed only once. Besides, in the node-wise sampling, the neighborhoods of each node are not visible to other parents; while for the layer-wise sampling all sampling nodes $\{\hat{u}_j\}_{j=1}^{n}$ are shared by all nodes of the current layer. This sharing property is able to enhance the message passing at utmost. More importantly, the size of each layer is fixed to n, and the total number of sampling nodes only grows linearly with the network depth.

According to the derivations of importance sampling in [23], we immediately conclude that

Proposition 1. *The variance of the estimator $\hat{\mu}_q(v_i)$ in Eq. (5) is given by*

$$\mathrm{Var}_q(\hat{\mu}_q(v_i)) \quad = \quad \frac{1}{n}\mathbb{E}_{q(u_j)}[\frac{(p(u_j|v_i)|h^{(l)}(u_j)| - \mu_q(v_i)q(u_j))^2}{q^2(u_j)}]. \tag{6}$$

The optimal sampler to minimize the variance $\mathrm{Var}_{q(u_j)}(\hat{\mu}_q(v_i))$ in Eq. (6) is given by

$$q^*(u_j) \quad = \quad \frac{p(u_j|v_i)|h^{(l)}(u_j)|}{\sum_{j=1}^{N}p(u_j|v_i)|h^{(l)}(u_j)|}. \tag{7}$$

To alleviate this chicken-and-egg dilemma, we learn a self-dependent function of each node to determine its importance for the sampling. Let $g(x(u_j))$ be the self-dependent function computed based on the node feature $x(u_j)$. Replacing the hidden function in Eq. (7) with $g(x(u_j))$ arrives at

$$q^*(u_j) \quad = \quad \frac{p(u_j|v_i)|g(x(u_j))|}{\sum_{j=1}^{N}p(u_j|v_i)|g(x(u_j))|}, \tag{8}$$

The sampler by Eq. (8) is node-wise and varies for different v_i. To make it applicable for the layer-wise sampling, we summarize the computations over all nodes $\{v_i\}_{i=1}^{n}$, thus we attain

$$q^*(u_j) \quad = \quad \frac{\sum_{i=1}^{n}p(u_j|v_i)|g(x(u_j))|}{\sum_{j=1}^{N}\sum_{i=1}^{n}p(u_j|v_i)|g(x(v_j))|}. \tag{9}$$

In this paper, we define $g(x(u_j))$ as a linear function *i.e.* $g(x(u_j)) = W_g x(u_j)$ parameterized by the matrix $W_g \in \mathbb{R}^{1\times D}$. Computing the sampler in Eq. (9) is efficient, since computing $p(u_j|v_i)$ (*i.e.* the adjacent value) and the self-dependent function $g(x(u_j))$ is fast.

loss

$$\mathcal{L} = \frac{1}{n}\sum\nolimits_{i=1}^{n} \mathcal{L}_c(y_i, \bar{y}(\hat{\mu}_q(v_i))) + \lambda \mathrm{Var}_q(\hat{\mu}_q(v_i))), \tag{10}$$

Preserving Second-Order Proximities by Skip Connections

Instead of learning a free $W_{skip}^{(l-1)}$ in Eq. (11), we decompose it to be

$$W_{skip}^{(l-1)} = W^{(l-1)}W^{(l)}, \tag{13}$$

where $W^{(l)}$ and $W^{(l-1)}$ are the filters of the l-th and $(l-1)$-th layers in original network, respectively. The output of skip-connection will be added to the GCN layer (Eq.(1)) before nonlinearity.

By the skip connection, the second-order proximity is maintained without extra 2-hop sampling. Besides, the skip connection allows the information to pass between two distant layers thus enabling more efficient back-propagation and model training.

The key idea of the skip connection is to reuse the nodes of the $(l-1)$-th layer to preserve the second-order proximity (see the definition in [7]). For the $(l+1)$-th layer, the nodes of the $(l-1)$-th layer are actually the 2-hop neighborhoods. If we further add a skip connection from the $(l-1)$-th to the $(l+1)$-th layer, as illustrated in Figure 2 (c), the aggregation will involve both the 1-hop and 2-hop neighborhoods. The calculations along the skip connection are formulated as

$$h_{skip}^{(l+1)}(v_i) = \sum\nolimits_{j=1}^{n} \hat{a}_{skip}(v_i, s_j)h^{(l-1)}(s_j)W_{skip}^{(l-1)}, \quad i = 1, \cdots, n, \tag{11}$$

where $s = \{s_j\}_{j=1}^{n}$ denote the nodes in the $(l-1)$-th layer. Due to the 2-hop distance between v_i and s_j, the weight $\hat{a}_{skip}(v_i, s_j)$ is supposed to be the element of $\hat{A}^2$. Here, to avoid the full computation of $\hat{A}^2$, we estimate the weight with the sampled nodes of the l-th layer, *i.e.*,

$$\hat{a}_{skip}(v_i, s_j) \approx \sum\nolimits_{k=1}^{n} \hat{a}(v_i, u_k)\hat{a}(u_k, s_j). \tag{12}$$

c. 实验结果

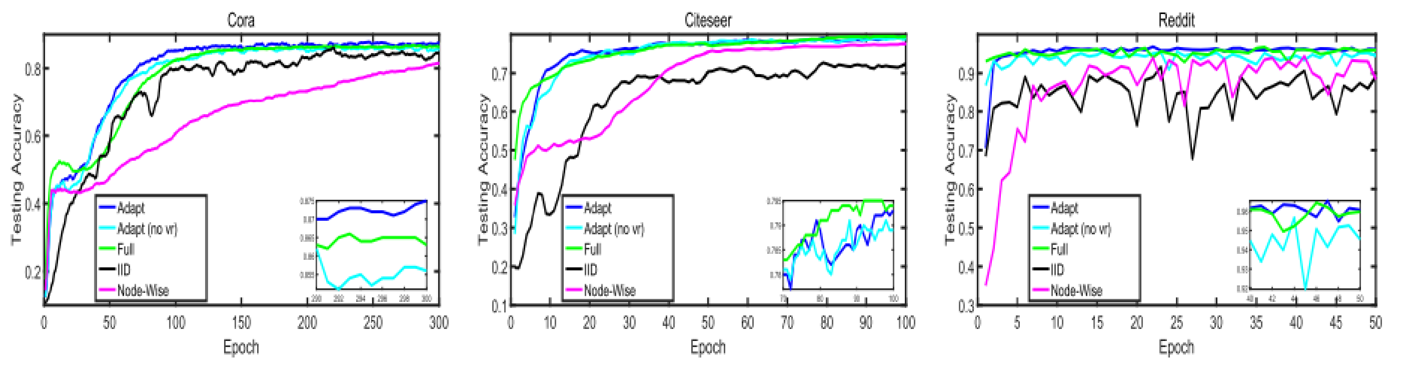
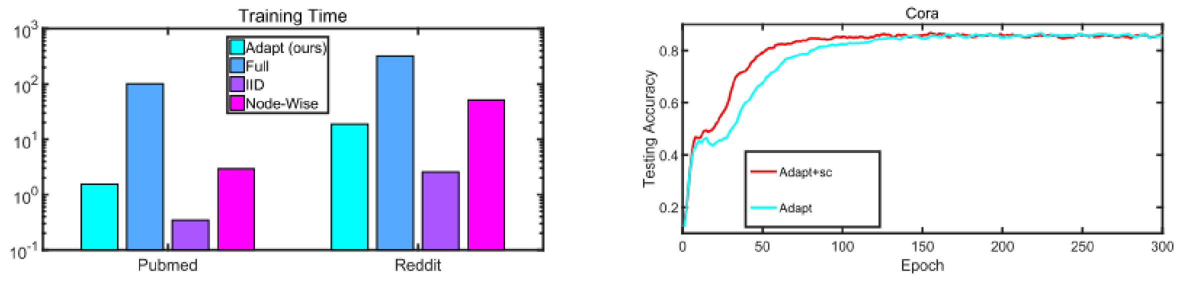


Figure 2: The accuracy curves of test data on Cora, Citeseer and Reddit. Here, one training epoch means a complete pass of all training samples.

Table 1: Accuracy Comparisons with state-of-the-art methods.

Methods	Cora	Citeseer	Pubmed	Reddit
KLED [25]	0.8229	-	0.8228	-
2-hop DCNN [18]	0.8677	-	0.8976	-
FastGCN [21]	0.8500	0.7760	0.8800	0.9370
GraphSAGE[3]	0.8220	0.7140	0.8710	0.9432
Full	0.8664 ± 0.0011	0.7934 ± 0.0026	0.9022 ± 0.0008	0.9568 ± 0.0069
IID	0.8506 ± 0.0048	0.7387 ± 0.0078	0.8200 ± 0.0114	0.8611 ± 0.0437
Node-Wise	0.8202 ± 0.0133	0.7734 ± 0.0081	0.9002 ± 0.0017	0.9449 ± 0.0026
Adapt (no vr)	0.8588 ± 0.0062	0.7942 ± 0.0022	0.9060 ± 0.0024	0.9501 ± 0.0047
Adapt	$\mathbf{0.8744 \pm 0.0034}$	$\mathbf{0.7966 \pm 0.0018}$	$\mathbf{0.9060 \pm 0.0016}$	$\mathbf{0.9627 \pm 0.0032}$



(a) (b)

Figure 3: (a) Training time per epoch on Pubmed and Reddit. (b) Accuracy curves of testing data on Cora for our Adapt method and its variant by adding skip connections.

2.5 LADIES

a. 背景介绍

为了解决full-batch训练带来的computation dependency的指数级增长，现有两个方向来解决这个问题：

1. node-wise neighbor sampling, 比如GraphSAGE, 通过uniform sampling出固定数目的邻居。这种方法会带来冗余计算[11]的问题，尽管两个邻居可能采样的是相同的邻居，这些邻居的表示会被计算

两次，并且冗余计算会随着层数变化而增加; VRGCN通过利用variance reduction技术提高了采样的复杂度，ClusterGCN将采样的邻居限制在了dense subgraph;

2. layer-wise importance-sampling方法。FastGCN基于每个顶点的度数计算了采样的概率，然后每层采样了固定数目的节点; 然而，在这种采样方法中，每一层的采样概率是独立的，这导致两个连续层之间不一定相互连接，从而导致采样得到的邻接矩阵异常稀疏。为了解决inter-layer correlation和reduce the estimation variance, huang[11]提出了adaptive和trainable采样方法，最终取得了很好的精度，但是[11]的方法没有很好的理论和实际证明。

综合上面采样方法的优缺点，我们认为一个理想的采样算法应该包括如下的特征:

1. layer-wise, the neigbor nodes可以被考虑到计算下一层的表示，并且没有冗余
2. neighbor-dependent, 采样到的邻接矩阵应该稠密，不能丢失太多的训练信息
3. importance samling方法应该被采样来reduce the sampling variance和accelerate convergence. 基于以上，本文提出一个新的采样算法，LAyer-Dependene ImportancE-Sampling(LADIES)

算法的流程如下

1. 对于当前层l, 基于上一层l+1采样，从图中采样它们的所有邻居，构建二分图
2. 根据顶点的阶数计算采样的概率来reduce sampling variance
3. 基于概率采样固定数目的邻居
4. 构建最后的邻接矩阵，用于训练

本文的贡献：

1. 提出了LADIES算法，避免了接受域指数级增长的问题和采样子图之间的连接性
2. 理论上证明了该算法拥有更好的内存和时间复杂度，相比于之前的算法
3. 在benchmark adtasets上再running time和test accuracy都取得了非常好的精度

b. 算法原理

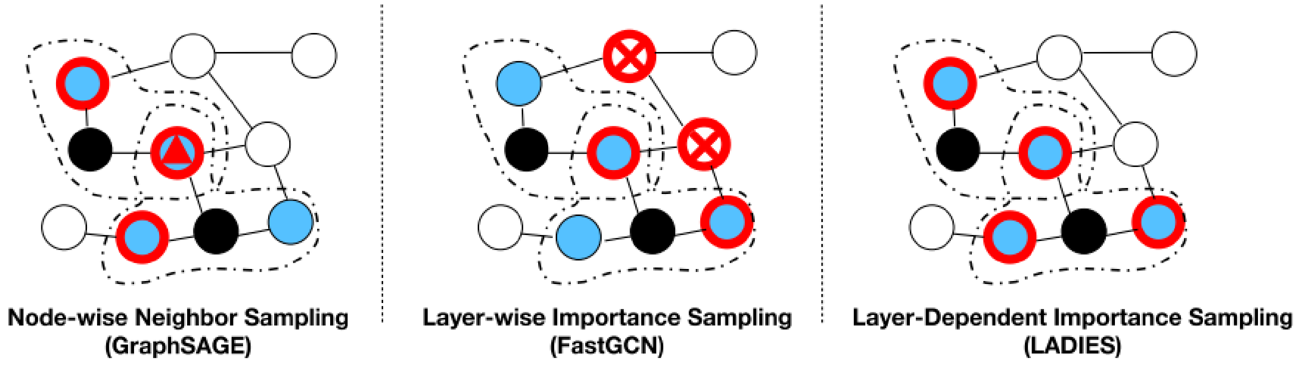

Node-wise Neighbor Sampling (GraphSAGE)
Layer-wise Importance Sampling (FastGCN)
Layer-Dependent Importance Sampling (LADIES)

Figure 1: An illustration of the sampling process of GraphSage, FastGCN, and our proposed LADIES. Black nodes denote the nodes in the upper layer, blue nodes in the dashed circle are their neighbors, and node with the red frame is the sampled nodes. As is shown in the figure, GraphSAGE will redundantly sample a neighboring node twice, denoted by the red triangle, while FastGCN will sample nodes outside of the neighborhood. Our proposed LADIES can avoid these two problems.

Notations, Summary of Complexity and Variance

Table 1: Summary of Notations

$\mathcal{G} = (\mathcal{V}, \mathbf{A})$, $\|\mathbf{A}\|_0$	$\mathcal{G}$ denotes a graph consist of a set of nodes $\mathcal{V}$ with node number $\|\mathcal{V}\|$, $\mathbf{A}$ is the adjacency matrix, and $\|\mathbf{A}\|_0$ denotes the number of non-zero entries in $\mathbf{A}$.
$\mathbf{M}_{i,*}, \mathbf{M}_{*,j}$ $\mathbf{M}_{i,j}$	$\mathbf{M}_{i,*}$ is the i-th row of matrix M, $\mathbf{M}_{*,j}$ is the j-th column of matrix M, and $\mathbf{M}_{i,j}$ is the entry at the position (i, j) of matrix M.
$\tilde{\mathbf{A}}, \tilde{\mathbf{D}}, \mathbf{P}$	$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\tilde{\mathbf{D}}$ is a diagonal matrix satisfying $\tilde{\mathbf{D}}_{i,i} = \sum_j \tilde{\mathbf{A}}_{i,j}$, and $\mathbf{P} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is the normalized Laplacian matrix.
$l, \sigma(\cdot), \mathbf{H}^{(l)}, \mathbf{Z}^{(l)}, \mathbf{W}^{(l)}$	l denotes the GCN layer index, $\sigma(\cdot)$ denotes the activation function, $\mathbf{H}^{(l)} = \sigma(\mathbf{Z}^{(l)})$ denotes the embedding matrix at layer l, $\mathbf{H}^{(0)} = \mathbf{X}$, $\mathbf{Z}^{(l)} = \mathbf{P}\mathbf{H}^{(l-1)}\mathbf{W}^{(l-1)}$ is the intermediate embedding matrix, and $\mathbf{W}^{(l)}$ denotes the trainable weight matrix at layer l.
L, K	L is the total number of layers in GCN, and K is the dimension of embedding vectors (for simplicity, assume it is the same across all layers).
b, s_{node}, s_{layer}	For batch-wise sampling, b denotes the batch size, s_{node} is the number of sampled neighbors per node for node-wise sampling, and s_{layer} is number of sampled nodes per layer for layer-wise sampling.

Table 2: Summary of Complexity and Variance [6]. Here ϕ denotes the upper bound of the ℓ_2 norm of embedding vector, $\Delta\phi$ denotes the bound of the norm of the difference between the embedding and its history, D denotes the average degree, b denotes the batch size, and $\bar{V}(b)$ denotes the average number of nodes which are connected to the nodes sampled in the training batch.

Methods	Memory Complexity	Time Complexity	Variance
Full-Batch [12]	$\mathcal{O}(L\|\mathcal{V}\|K + LK^2)$	$\mathcal{O}(L\|\mathbf{A}\|_0 K + L\|\mathcal{V}\|K^2)$	0
GraphSage [9]	$\mathcal{O}(bKs_{node}^{L-1} + LK^2)$	$\mathcal{O}(bKs_{node}^{L} + bK^2 s_{node}^{L-1})$	$\mathcal{O}\big(D\phi\|\mathbf{P}\|_F^2/(\|\mathcal{V}\|s_{node})\big)$
VR-GCN [3]	$\mathcal{O}(L\|\mathcal{V}\|K + LK^2)$	$\mathcal{O}(bDKs_{node}^{L-1} + bK^2 s_{node}^{L-1})$	$\mathcal{O}\big(D\Delta\phi\|\mathbf{P}\|_F^2/(\|\mathcal{V}\|s_{node})\big)$
FastGCN [4]	$\mathcal{O}(LKs_{layer} + LK^2)$	$\mathcal{O}(LKs_{layer}^2 + LK^2 s_{layer})$	$\mathcal{O}(\phi\|\mathbf{P}\|_F^2/s_{layer})$
LADIES	$\mathcal{O}(LKs_{layer} + LK^2)$	$\mathcal{O}(LKs_{layer}^2 + LK^2 s_{layer})$	$\mathcal{O}\big(\phi\|\mathbf{P}\|_F^2\bar{V}(b)/(\|\mathcal{V}\|s_{layer})\big)$

c. 算法流程

Algorithm 1 Sampling Procedure of LADIES

```
Require: Normalized Laplacian Matrix $\mathbf{P}$; Batch Size $b$, Sample Number $n$;
1: Randomly sample a batch of $b$ output nodes as $\mathbf{Q}^L$
2: for $l = L$ to 1 do
3:     Get layer-dependent laplacian matrix $\mathbf{Q}^{(l)}\mathbf{P}$. Calculate sampling probability for each node
       using $p_i^{(l-1)} \leftarrow \frac{\|\mathbf{Q}^{(l)}\mathbf{P}_{*,i}\|_2^2}{\|\mathbf{Q}^{(l)}\mathbf{P}\|_F^2}$, and organize them into a random diagonal matrix $S^{(l-1)}$.
4:     Sample $n$ nodes in $l-1$ layer using $p^{(l-1)}$. The sampled nodes formulate $\mathbf{Q}^{(l-1)}$
5:     Reconstruct sampled laplacian matrix between sampled nodes in layer $l-1$ and $l$ by
       $\tilde{\mathbf{P}}^{(l-1)} \leftarrow \mathbf{Q}^{(l)}\mathbf{P}\mathbf{S}^{(l-1)}\mathbf{Q}^{(l-1)\top}$, then normalize it by $\tilde{\mathbf{P}}^{(l)} \leftarrow \mathbf{D}_{\tilde{\mathbf{P}}^{(l)}}^{-1}\tilde{\mathbf{P}}^{(l)}$.
6: end for
7: return Modified Laplacian Matrices $\{\tilde{\mathbf{P}}^{(l)}\}_{l=1,\dots,L}$ and Sampled Node at Input Layer $\mathbf{Q}^0$;
```

d. 实验结果

Table 3: Comparison of LADIES with original GCN (Full-Batch), GraphSage (Neighborhood Sampling) and FastGCN (Important Sampling), in terms of accuracy, time, memory and convergence. Training 5-layer GCNs on different node classification datasets (node number is below the dataset name). Results show that LADIES can achieve the best accuracy with lower time and memory cost.

Dataset	Sample Method	F1-Score(%)	Total Time(s)	Mem(MB)	Batch Time(ms)	Batch Num
Cora (2708)	Full-Batch	76.5 ± 1.4	1.19 ± 0.82	30.72	15.75 ± 0.52	80.8 ± 51.7
	GraphSage (5)	75.2 ± 1.5	6.77 ± 4.94	471.39	78.42 ± 0.87	65.2 ± 52.1
	FastGCN (64)	25.1 ± 8.4	0.55 ± 0.65	3.13	9.22 ± 0.20	63.2 ± 71.2
	FastGCN (512)	78.0 ± 2.1	4.70 ± 1.35	7.33	10.08 ± 0.29	487 ± 147
	LADIES (64)	77.6 ± 1.4	4.19 ± 1.16	**3.13**	9.68 ± 0.48	436 ± 118.4
	LADIES (512)	**78.3 ±1.6**	**0.72 ± 0.39**	7.35	9.77 ± 0.28	75.6 ± 37.0
Citeseer (3327)	Full-Batch	62.3 ± 3.1	0.61 ± 0.70	68.13	15.77 ± 0.58	40.6 ± 22.8
	GraphSage (5)	59.4 ± 0.9	4.51 ± 3.68	595.71	53.14 ± 1.90	57.2 ± 42.1
	FastGCN (64)	19.2 ± 2.7	0.53 ± 0.48	5.89	8.88 ± 0.40	64.0 ± 57.0
	FastGCN (512)	44.6 ± 10.8	4.34 ± 1.73	13.97	10.41 ± 0.51	386 ± 167
	FastGCN (1024)	63.5 ± 1.8	2.24 ± 1.01	23.24	10.54 ± 0.27	223 ± 98.6
	LADIES (64)	**65.0 ± .1.4**	2.17 ± 0.65	**5.89**	9.60 ± 0.39	232 ± 66.8
	LADIES (512)	64.3 ± 2.4	**0.41 ± 0.22**	13.92	10.32 ± 0.23	37.6 ± 11.9
Pubmed (19717)	Full-Batch	71.9 ± 1.9	4.80 ± 1.53	137.93	44.69 ± 0.57	102 ± 33.4
	GraphSage (5)	70.1 ± 1.4	5.53 ± 2.57	453.58	44.73 ± 0.30	74.8 ± 31.7
	FastGCN (64)	38.5 ± 6.9	0.40 ± 0.69	1.92	7.42 ± 0.16	58.8 ± 94.8
	FastGCN (512)	39.3 ± 9.2	**0.44 ± 0.61**	4.53	10.06 ± 0.41	44.8 ± 55.0
	FastGCN (8192)	74.4 ± 0.8	3.47 ± 1.16	49.41	17.84 ± 0.33	195 ± 56.9
	LADIES (64)	**76.8 ± 0.8**	2.57 ± 0.72	**1.92**	9.43 ± 0.47	277 ± 82.2
	LADIES (512)	75.9 ± 1.1	2.27 ± 1.17	4.39	10.43 ± 0.36	245 ± 84.5
Reddit (232965)	Full-Batch	91.6 ± 1.6	474.3 ± 84.4	2370.48	1564 ± 3.41	179 ± 75.5
	GraphSage (5)	92.1 ± 1.1	13.12 ± 2.84	1234.63	121.47 ± 0.72	81.5 ± 42.3
	FastGCN (64)	27.8 ± 12.6	2.06 ± 1.29	3.75	7.85 ± 0.72	57.4 ± 43.7
	FastGCN (512)	17.5 ± 16.7	**0.31 ± 0.41**	6.91	10.01 ± 0.31	32.1 ± 72.3
	FastGCN (8192)	89.5 ± 1.2	5.63 ± 2.12	74.28	16.57 ± 0.58	278 ± 51.2
	LADIES (64)	83.5 ± 0.9	5.62 ± 1.58	**3.75**	9.42 ± 0.48	453 ± 88.2
	LADIES (512)	**92.8 ± 1.6**	6.87 ± 1.17	7.26	10.87 ± 0.63	393 ± 74.4

- **Accuracy**: The micro F1-score of the test data at the convergence point. We calculate it using the full-batch version to get the most accurate inference (only care about training).
- **Total Running Time (s)**: The total training time (exclude validation) before convergence point.
- **Memory (MB)**: Total memory costs of model parameters and all hidden representations of a batch.
- **Batch Time and Num**: Time cost to run a batch and the total batch number before convergence.

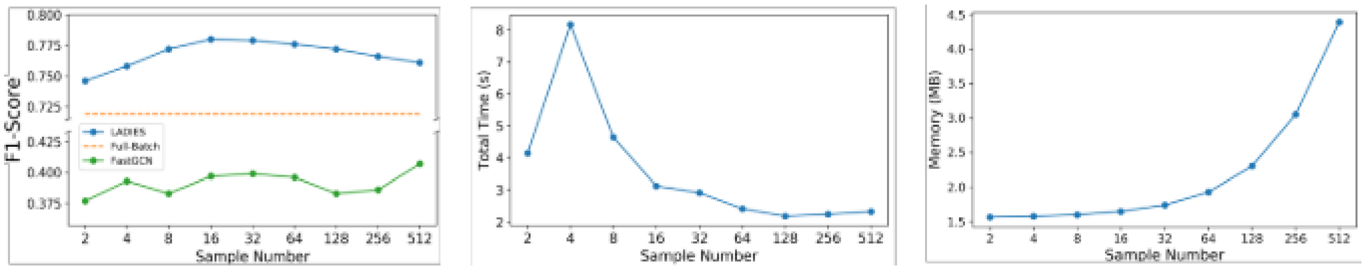
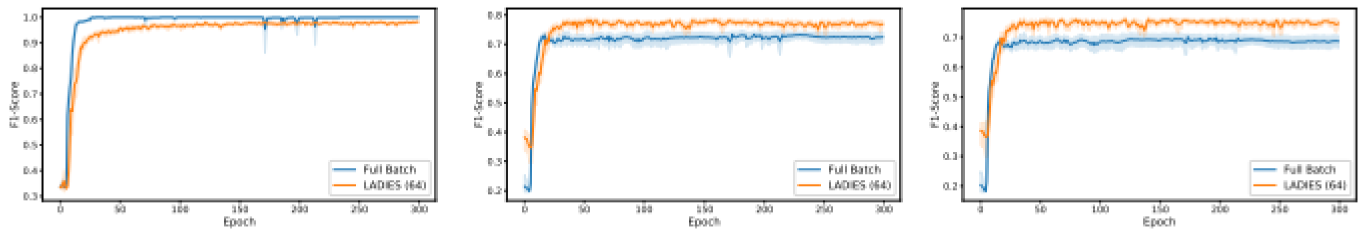


Figure 2: F1-score, total time and memory cost at convergence point for training PubMed, when we choose different sampling numbers of our method. Results show that LADIES can achieve good generalization accuracy (F1-score = 77.6) even with a small sampling number as 16, while FastGCN cannot converge (only reach F1-score = 39.3) with a large sampling number as 512.



(a) Training (60 data samples) (b) Validation (500 data samples) (c) Testing (1000 data samples)

Figure 3: Experiments on the PubMed dataset. We plot the F1-score of both full-batch GCN and LADIES every epoch on (a) Training dataset (b) Validation dataset and (c) Testing dataset.

2.6 ClusterGCN

a. 背景介绍

对于一个图，GCN采用图卷积运算逐层地获取节点的embedding：在每一层，要获取一个节点的embedding，需要通过采集相邻节点的embedding，然后进行一层或几层线性变换和非线性激活。最后一层embedding将用于一些最终任务。例如，在节点分类问题中，最后一层embedding被传递给分类器来预测节点标签，从而可以对GCN的参数进行端到端的训练。

由于GCN中的图卷积运算（operator）需要利用图中节点之间的交互来传播embeddings，这使得训练变得相当具有挑战性。不像其他神经网络，训练损失可以在每个样本上完美地分解为单独的项（decomposed into individual terms），GCN中的损失项(例如单个节点上的分类损失)依赖于大量的其他节点，尤其是当GCN变深时。由于节点依赖性，GCN的训练非常慢，需要大量的内存——反向传播需要将计算图上的所有embeddings存储在GPU内存中。

现有算法的缺陷

为了证明开发可扩展的GCN训练算法的必要性，文中首先讨论了现有方法的优缺点，包括：内存需求、每个epoch的时间、每个epoch收敛速度。

这三个因素是评估训练算法的关键。注意，内存需求直接限制了算法的可扩展性，后两个因素结合在一起将决定训练速度。在接下来的讨论中，用N为图中的节点数，F为embedding的维数，L为分析经典GCN训练算法的层数。

GCN的第一篇论文提出了全批次梯度下降（Full-batch gradient descent）。要计算整个梯度，它需要存储所有中间embeddings，导致O(NFL)内存需求，这是不可扩展的。
GraphSAGE中提出了Mini-batch SGD。它可以减少内存需求，并在每个epoch执行多次更新，从而加快了收敛速度。然而，由于邻居扩展问题，mini-batch SGD在计算L层单个节点的损失时引入了大量的计算开销。
VR-GCN提出采用variance减少技术来减小邻域采样节点的大小。但它需要将所有节点的所有中间的embeddings存储在内存中，从而导致O(NFL)内存需求。

作者定义了“Embedding utilization”的概念来表达计算效率。如果节点i在第l层的embedding在计算第l+1层的embeddings时被重用了u次，那么就说相应的的embedding utilization是u。

下表中总结了现有GCN训练算法相应的时间和空间复杂度。显然，所有基于SGD的算法的复杂度都和层数呈指数级关系。对于VR-GCN，即使r很小，也会产生超出GPU内存容量的巨大空间复杂度。

本文提出的的Cluster-GCN算法，它实现了两全其美的效果：即每个epoch和full gradient descent具有相同的时间复杂度， 同时又能与朴素GD具有相同的空间复杂度。

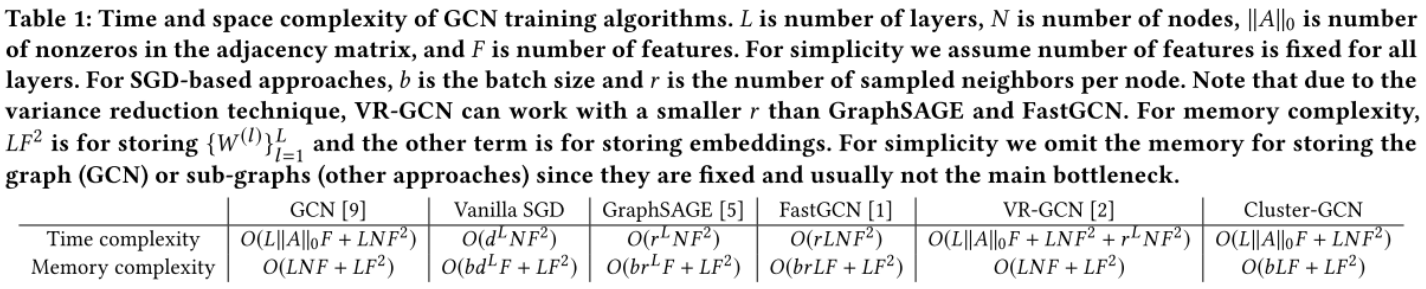


Table 1: Time and space complexity of GCN training algorithms. L is number of layers, N is number of nodes, $\|A\|_0$ is number of nonzeros in the adjacency matrix, and F is number of features. For simplicity we assume number of features is fixed for all layers. For SGD-based approaches, b is the batch size and r is the number of sampled neighbors per node. Note that due to the variance reduction technique, VR-GCN can work with a smaller r than GraphSAGE and FastGCN. For memory complexity, LF^2 is for storing $\{W^{(l)}\}_{l=1}^{L}$ and the other term is for storing embeddings. For simplicity we omit the memory for storing the graph (GCN) or sub-graphs (other approaches) since they are fixed and usually not the main bottleneck.

	GCN [9]	Vanilla SGD	GraphSAGE [5]	FastGCN [1]	VR-GCN [2]	Cluster-GCN
Time complexity	$O(L\|A\|_0 F + LNF^2)$	$O(d^L NF^2)$	$O(r^L NF^2)$	$O(rLNF^2)$	$O(L\|A\|_0 F + LNF^2 + r^L NF^2)$	$O(L\|A\|_0 F + LNF^2)$
Memory complexity	$O(LNF + LF^2)$	$O(bd^L F + LF^2)$	$O(br^L F + LF^2)$	$O(brLF + LF^2)$	$O(LNF + LF^2)$	$O(bLF + LF^2)$

b. 算法原理

文中的Cluster-GCN技术是由以下问题驱动的：在mini-batch SGD更新中，我们可以设计一个batch和相应的计算子图来最大限度地提高embedding utilization吗？文中使用了图聚类算法来划分图。图聚类的方法，旨在在图中的顶点上构建分区，使簇内连接远大于簇间连接，从而更好地捕获聚类和社区结构。

下图展示了两种不同的节点分区策略：随机分区和clustering分区。可以看到，cluster-GCN可以避免大量的邻域搜索，并且集中在每个簇中的邻居上。作者使用随机分割和Metis聚类方法将图分成10个部分。然后使用一个分区作为一个batch来执行SGD更新。在相同的时间段内，使用聚类划分可以获得更高的精度。这表明使用图聚类是很重要的，分区不应该随机形成。

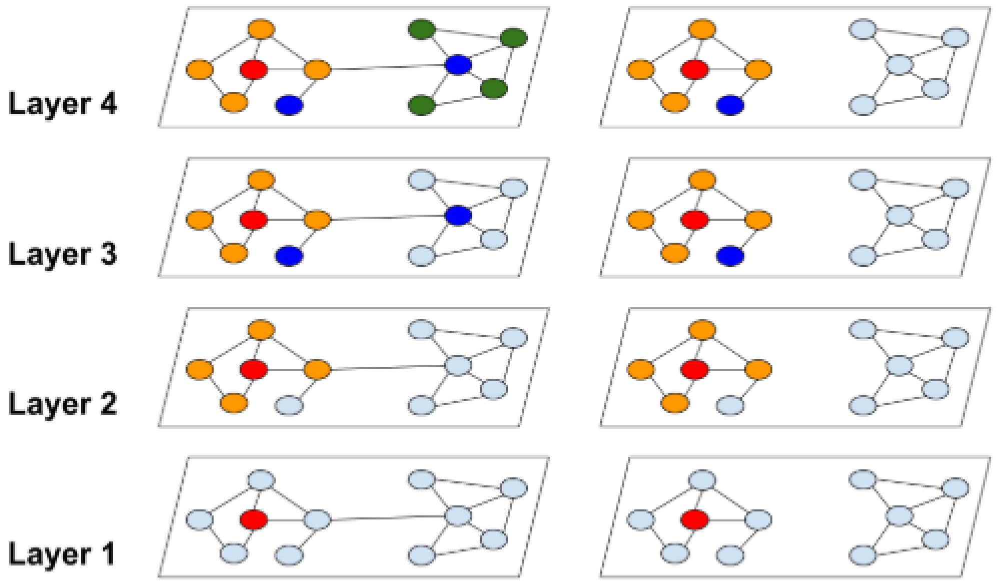


Figure 1: The neighborhood expansion difference between traditional graph convolution and our proposed cluster approach. The red node is the starting node for neighborhood nodes expansion. Traditional graph convolution suffers from exponential neighborhood expansion, while our method can avoid expensive neighborhood expansion.

随机多聚类

尽管朴素Cluster-GCN实现了良好的时间和空间复杂度，但仍然存在两个潜在问题：

- 图被分割后，一些连接被删除。因此，性能可能会受到影响。
- 图聚类算法往往将相似的节点聚集在一起，因此聚类的分布可能不同于原始数据集，从而导致在执行SGD更新时对 full gradient的估计有偏差。

为了解决上述问题，文中提出了一种随机多聚类方法，在簇接之间进行合并，并减少batch间的差异（variance）。作者首先用一个较大的p把图分割成p个簇V1,...,Vp，然后对于SGD的更新重新构建一个batch B，而不是只考虑一个簇。随机地选择q个簇，定义为t1,...,tq ,并把它们的节点包含到这个batch B中。此外，在选择的簇之间的连接也被添加回去。作者在Reddit数据集上进行了一个实验，证明了该方法的有效性。

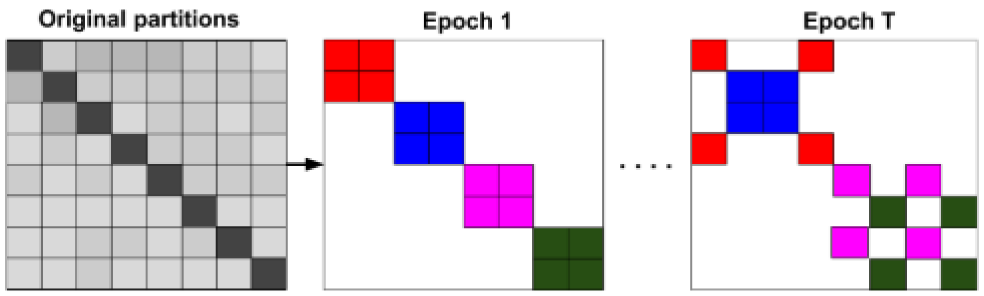


Figure 3: The proposed stochastic multiple partitions scheme. In each epoch, we randomly sample q clusters ($q = 2$ is used in this example) and their between-cluster links to form a new batch. Same color blocks are in the same batch.

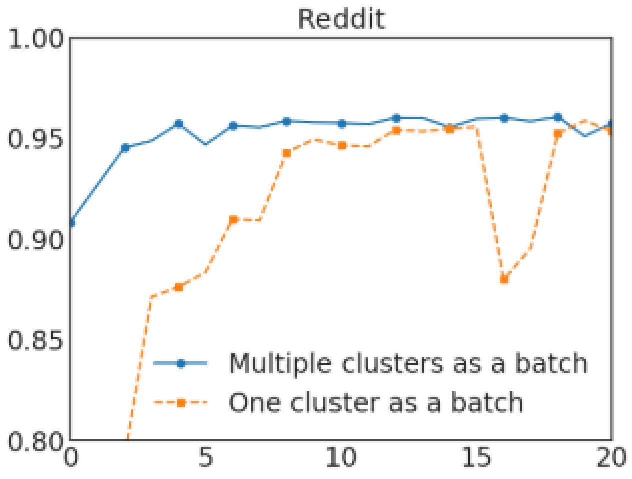
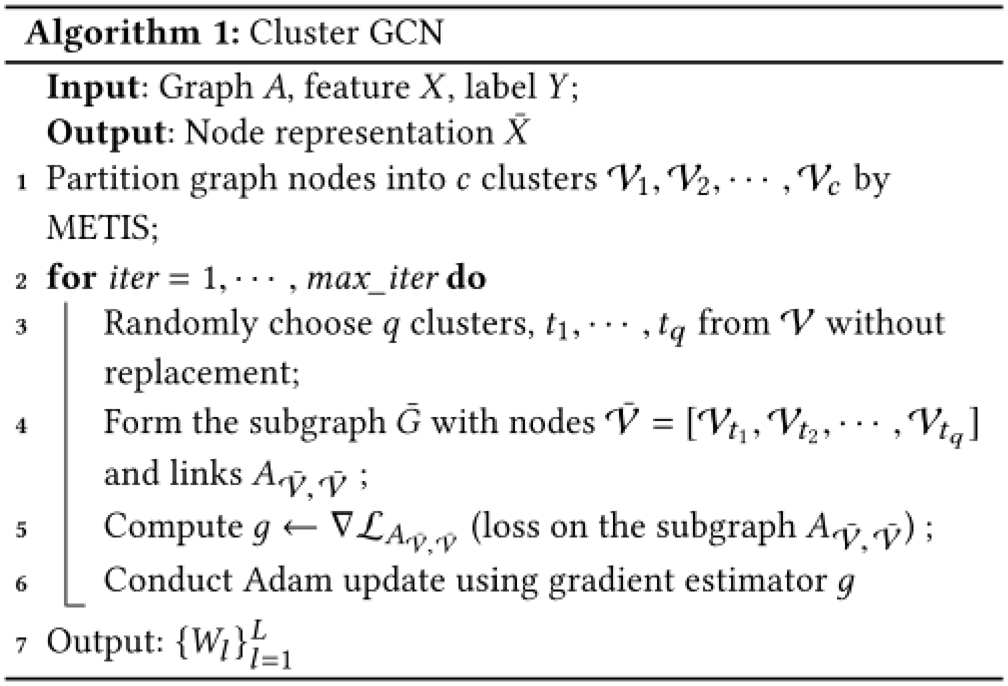


Figure 4: Comparisons of choosing one cluster versus multiple clusters. The former uses 300 partitions. The latter uses 1500 and randomly select 5 to form one batch. We present epoch (x-axis) versus F1 score (y-axis).

c. 算法流程



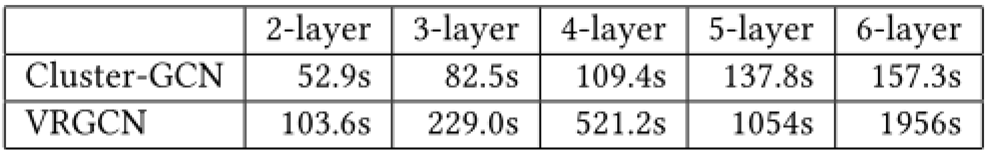
```
Algorithm 1: Cluster GCN
  Input: Graph $A$, feature $X$, label $Y$;
  Output: Node representation $\bar{X}$
1 Partition graph nodes into $c$ clusters $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_c$ by
  METIS;
2 for $iter = 1, \cdots, max\_iter$ do
3     Randomly choose $q$ clusters, $t_1, \cdots, t_q$ from $\mathcal{V}$ without
      replacement;
4     Form the subgraph $\bar{G}$ with nodes $\bar{\mathcal{V}} = [\mathcal{V}_{t_1}, \mathcal{V}_{t_2}, \cdots, \mathcal{V}_{t_q}]$
      and links $A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}$ ;
5     Compute $g \leftarrow \nabla \mathcal{L}_{A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}}$ (loss on the subgraph $A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}$) ;
6     Conduct Adam update using gradient estimator $g$
7 Output: $\{W_l\}_{l=1}^{L}$
```

> 此文中还提到了一些变形公式，比如$X^{l+1} = \delta(AX^lW^l) + X^l$, $X^{l+1} = \delta((A + I)X^lW^l)$

d. 实验结果

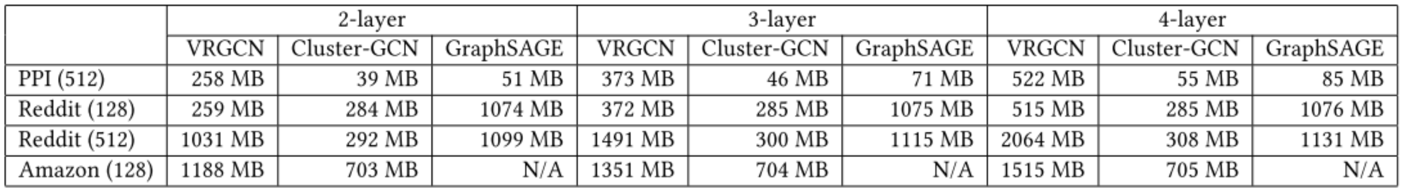
文中评估了所提出的针对四个公共数据集的多标签和多类分类两个任务的GCN训练方法，数据集统计如表3所示。Reddit数据集是迄今为止为GCN所看到的最大的公共数据集，为了测试GCN训练算法在大规模数据上的可扩展性，作者基于Amazon co-purchase network构建了一个更大的图Amazon2M，包含超过200万个节点和6100万条边。

Table 9: Comparisons of running time when using different numbers of GCN layers. We use PPI and run both methods for 200 epochs.



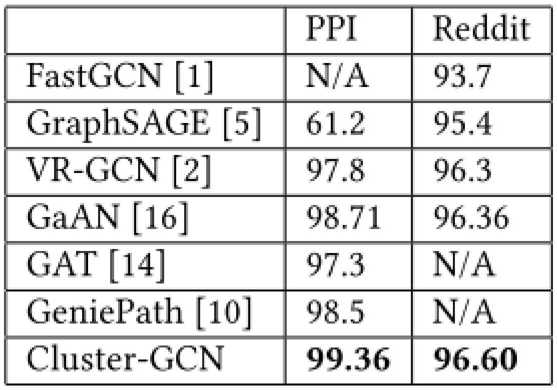
	2-layer	3-layer	4-layer	5-layer	6-layer
Cluster-GCN	52.9s	82.5s	109.4s	137.8s	157.3s
VRGCN	103.6s	229.0s	521.2s	1054s	1956s

Table 5: Comparisons of memory usages on different datasets. Numbers in the brackets indicate the size of hidden units used in the model.



	2-layer			3-layer			4-layer		
	VRGCN	Cluster-GCN	GraphSAGE	VRGCN	Cluster-GCN	GraphSAGE	VRGCN	Cluster-GCN	GraphSAGE
PPI (512)	258 MB	39 MB	51 MB	373 MB	46 MB	71 MB	522 MB	55 MB	85 MB
Reddit (128)	259 MB	284 MB	1074 MB	372 MB	285 MB	1075 MB	515 MB	285 MB	1076 MB
Reddit (512)	1031 MB	292 MB	1099 MB	1491 MB	300 MB	1115 MB	2064 MB	308 MB	1131 MB
Amazon (128)	1188 MB	703 MB	N/A	1351 MB	704 MB	N/A	1515 MB	705 MB	N/A

Table 10: State-of-the-art performance of testing accuracy reported in recent papers.



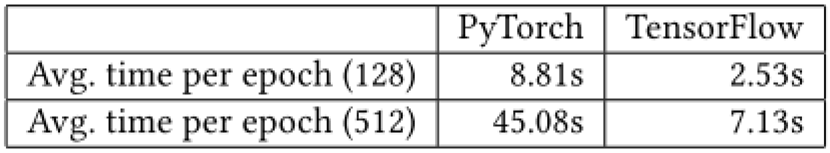
	PPI	Reddit
FastGCN [1]	N/A	93.7
GraphSAGE [5]	61.2	95.4
VR-GCN [2]	97.8	96.3
GaAN [16]	98.71	96.36
GAT [14]	97.3	N/A
GeniePath [10]	98.5	N/A
Cluster-GCN	**99.36**	**96.60**

作者比较了不同层次GCNs的VRGCN在训练时间、内存使用和测试准确度(F1分数)方面的差异。从表中可以看出

- 训练两层时VRGCN比Cluster-GCN快，但是当增加一层网络，却慢于实现相似准确率的Cluster-GCN；
  在内存使用方面，VRGCN比Cluster-GCN使用更多的内存(对于三层的情况5倍多)。
- 当训练4层GCN的时候VRGCN将被耗尽，然而Cluster-GCN当增加层数的时候并不需要增加太多的内存，并且Cluster-GCN对于这个数据集训练4层的GCN将实现最高的准确率。

Tensorflow vs PyTorch

Table 6: Benchmarking on the Sparse Tensor operations in PyTorch and TensorFlow. A network with two linear layers is used and the timing includes forward and backward operations. Numbers in the brackets indicate the size of hidden units in the first layer. Amazon data is used.



	PyTorch	TensorFlow
Avg. time per epoch (128)	8.81s	2.53s
Avg. time per epoch (512)	45.08s	7.13s

2.7 GraphSAINT

a. 背景介绍

传统GNN挑战：邻居爆炸（Neighbor Explosion）

邻居爆炸：

GNN会不断地聚合图中相邻节点的信息，则L-层的GNN中每个目标节点都需要聚合原图中L-hop以内的所有节点信息。在大图中，邻节点的个数可以随着L指数级增长。

邻点爆炸式增长，使得GNN的minibatch训练极具挑战性。
受显存的因素限制，GNN在不损失精度的前提下，难以高效地被拓展到大于两层的模型（否则指数爆炸）。

现有方法：图采样

Layer-wise Sampling：
邻居爆炸：在矩阵采样多层时，假设每层采样n个邻居，则会导致n^2级别的节点扩充速度。
领接矩阵稀疏：在矩阵采样的过程中，会导致邻接矩阵稀疏化，丢失一些本来存在的边。
时间耗费高：每一层卷积都要采样，这就导致计算的时间耗费。
前人工作提出了众多基于采样邻居节点的方法，并利用多类型的聚合函数提高模型的表达能力。大部分现有工作都在探索如何通过对GNN每层的节点进行采样，以降低训练成本。

Graph-wise Sampling：
从原图上采样一个子图下来，在这个子图上做局部的、完全展开的GCN。
解决了邻居采样的指数化问题，而且可以对采下来的子图进行直接的并行化，就大大的改进了效率。
即，可以在preprocess阶段提前采样图，并且可以进行mini batch的加速。
但是这样的采样往往会丢失一些信息。

属于Graph sampling。
从原图中采样子图，在子图上使用GCN学习。
基于极其轻量级的子图采样算法，同时实现了在准确率和复杂度上的显著提升。
提出了适用于大图以及深层网络的，通用的训练框架。
在标准的Reddit数据集上，以100倍的训练时间提速，提高了1%以上的准确率。在这里插入图片描述

b. 算法原理

将全图进行多次采样，在得到的sub-graph上进行全GCN，然后将多个sub-graph的信息融合起来。

本文的采样是基于节点的连通性：

1. 相互影响较大的节点应在同一子图中采样，intra sub-graph的节点是具有强联系的，但这就引入了采样的bias。理想的SAMPLE要求衡量节点连接的联合信息以及属性。但这种算法可能具有很高的复杂度，所以，为简单起见，从图连接性角度定义“影响力”，并设计基于拓扑的采样器。
2. 为了消除保留图连通性特征的采样器引入的偏差，作者引入自行设计的归一化技术，以消除偏差，使得估计量无偏。归一化系数通过pre-processing估计出来。
3. 每条边的采样概率均不可忽略，使得神经网络能够探索整个特征和标签空间。
4. 考虑variance reduction（降低采样方差），旨在得到最小方差的边采样概率。

重点是估计每个节点、边、子图的采样概率。

- 节点采样概率：
- 边采样概率:
- 子图采样概率：

> 这篇文章的采样sub-graph的概念应该是介于AS-GCN的“有条件的采样全图”和ClusterGraph的“将图聚类后采样”之间。由于采样的存在，FastGCN, AS-GCN等在不同的layer计算的是不同的图结构，而GraphSAINT, ClusterGraph可以看做在不同layer都对同一个graph进行特征提取。

c. 实验结果

F1-micro score

deep model

total_time

Effect of Batch Size

2.8 MVS-GNN

a. 背景介绍

在大规模图上进行GNNs的训练仍然是一个很大的问题，这主要是因为图中节点之间的相互依赖。在GNNs中，节点表示通过聚集邻居上一层的表示得到。由此，跟其他神经网络的最终的输出和梯度由数据分解得到不一样，GNNs的embedding由其邻居的embedding递归得到，dependency由此随着层的数量指数级上升，即neigbor explosion。为了缓解这个问题, 有mini-batch方法提出，比如node-wise sampling, layer-wise sampling, subgraph sampling。

尽管实证结果展示上面的算法可以将GNN训练扩展到大规模数据集，但是这些采样方法仍然会产生high variance, 从而破坏收敛性和繁华性。为了减小采样方法的variance, 我们可以增加每层的mini-batch的大小或者应用adaptive sampling方法(gradient information of representations). 计算和内存要求会对采样节点的大小进行限制。

在importance sampling, adpative sampling方法中，关键的想法是利用优化期间的梯度信息采样训练数据来减小无偏随机梯度的方差。近期，不同的adaptive sampling方法，importance sampling, adaptive importance sampling, gradient-based sampling, safe adaptive sampling, bandit sampling. 尽管这些采样方法通过SGD保证了训练的结果，但是这些方法的概括却不是很直接。

本文认为由于训练目标的复合结构，stochastic gradient是full-batch的biased estimation, 因此可以分为两个部分：embedding approximation variance(因为每层采样到的是邻居的子集)和stochastic gradient variance(因为这里用mini-batch的梯度进行估计full-batch的梯度)。并且，stochastic gradient的bias和

embedding approximation variance成正比，随着stochastic gradient variance减小为0，stochastic gradient趋向于五篇。

本文所提出的minminval variance sampling schema即MVS-GNN, 利用了优化过程中的动态信息来采样节点信息，包含了两个方面的内容：1. explicit embedding variance by utilizing the history of embdding of nodes 2. gradient-based minimal variance sampling by utilizaing the norm of gradient of nodes and solving an optimization problem.

实验结果表明，相比于其他采样算法，MVS-GNN在非常小的mini-batchs下达到了很高的收敛性和精度。

b. 算法原理

decoupled variance reduction

> 1. embedding approximation variance, 因为只有一小部分邻居被采样
> 2. stochastic gradient variance, 因为只有一个mini-batch的数据区估计full-batch的梯度

下表为不同采样算法的embedding approximation variance

作者这里的思路借用了VRGNN的做法，使用两层循环：

1. 外层大Batch：
2. 内层小Batch：小Batch从大Batch中采样，

Gradient-based minimal variance sampling
求解一下最优化问题：

结果：

c. 算法流程

d. 实验结果

the effect if mini-batch size

effectiveness of variance reduction

evaluation of total time

> - T_{Sample}表示构建20 mini-batchs for traning所花的时间
> - T_{train}表示运行20 mini-batchs所有的时间
> - T_{Dists}表示importance sampling所花的时间

evaluation on small mini-batch size

evaluation on small mini-batch size

参考文献

- [GraphSAGE]Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. NIPS, 2017-Decem(Nips), 1025–1035. Retrieved from http://arxiv.org/abs/1706.02216
- [VRGCN]Chen, J., Zhu, J., & Song, L. (2018). Stochastic training of graph convolutional networks with variance reduction. 35th International Conference on Machine Learning, ICML 2018, 3, 1503–1532.
- [FastGCN]Chen, J., Ma, T., & Xiao, C. (2018). FastGCN: Fast learning with graph convolu-tional networks via importance sampling. ICLR, 1–15.
- [AS-GCN]Huang, W., Zhang, T., Rong, Y., & Huang, J. (2018). Adaptive sampling towards fast graph representation learning. Advances in Neural Information Processing Systems, 2018-Decem(NeurIPS), 4558–4567.
- [LADIES]Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., & Gu, Q. (2019). Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. (NeurIPS).
- [ClusterGCN]Chiang, W. L., Li, Y., Liu, X., Bengio, S., Si, S., & Hsieh, C. J. (2019). Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. SIGKDD, 257–266. https://doi.org/10.1145/3292500.3330925
- [GraphSAINT]Zeng, H., Srivastava, A., Prasanna, V., Zhou, H., & Kannan, R. (2020). GraphSAINT : GRAPH SAMPLING BASED INDUCTIVE LEARNING METHOD. (2018).
- [MVS-GNN]Cong, W., Forsati, R., Kandemir, M., & Mahdavi, M. (2020). Minimal Variance Sampling with Provable Guarantees for Fast Training of Graph Neural Networks. Retrieved from http://arxiv.org/abs/2006.13866