Performance-Adaptive Sampling Strategy Towards Fast and Accurate Graph Neural Networks

Minji Yoon∗, Théophile Gervet∗, Baoxu Shi†, Sufeng Niu†, Qi He‡, Jaewon Yang†
∗Carnegie Mellon University, Pittsburgh, PA, USA
†LinkedIn Corporation, Sunnyvale, CA, USA
{minjiy, tgervet}@andrew.cmu.edu, {dashi, sniu, qhe, jeyang}@linkedin.com

**ABSTRACT**

The main challenge of adapting Graph convolutional networks (GCNs) to large-scale graphs is the scalability issue due to the uncontrollable neighborhood expansion in the aggregation stage. Several sampling algorithms have been proposed to limit the neighborhood expansion. However, these algorithms focus on minimizing the variance in sampling to approximate the original aggregation. This leads to two critical problems: 1) low accuracy because the sampling policy is agnostic to the performance of the target task, and 2) vulnerability to noise or adversarial attacks on the graph.

In this paper, we propose a performance-adaptive sampling strategy PASS that samples neighbors informative for a target task. PASS optimizes directly towards task performance, as opposed to variance reduction. PASS trains a sampling policy by propagating gradients of the task performance loss through GCNs and the non-differentiable sampling operation. We dissect the back-propagation process and analyze how PASS learns from the gradients which neighbors are informative and assigned high sampling probabilities. In our extensive experiments, PASS outperforms state-of-the-art sampling methods by up to 10% accuracy on public benchmarks and up to 53% accuracy in the presence of adversarial attacks.

**CCS CONCEPTS**
- Information systems → Data mining.

**KEYWORDS**

Graph Neural Networks; Sampling policy

**ACM Reference Format:**

1 **INTRODUCTION**

Graph convolutional networks (GCNs) [12] have garnered considerable attention as a powerful deep learning tool for representation learning of graph data [2, 19]. For instance, GCNs demonstrate state-of-the-art performance on node classification [5], link prediction [16], and graph property prediction tasks [7]. Motivated by convolutional neural networks, GCNs aggregate information from a node’s neighbors analogously to how convolution filters process text or image data [9, 13].

The main challenge of adapting GCNs to large-scale graphs is that GCNs expand neighbors recursively in the aggregation operations, leading to high computation and memory footprints. For instance, given a graph whose average degree is $d$, $L$-layer GCNs access $d^L$ neighbors per node on average. If the graph is dense or has many high degree nodes, GCNs need to aggregate a huge number of neighbors for most of the training/test examples. The only way to alleviate this neighbor explosion problem is to sample a fixed number of neighbors in the aggregation operation, thereby regulating the computation time and memory usage [8].

Most samplers minimize the variance in sampling to approximate the original aggregation of the full neighborhood [4, 10, 14, 23]. These sampling policies learn neighbors helpful for variance reduction, not neighbors informative for the target task’s performance. Thus, those variance reduction-oriented samplers suffer from two critical problems: 1) low accuracy because the sampling policy is agnostic to the performance, and 2) vulnerability to noise or adversarial attacks on the graph because the sampling policy cannot distinguish relevant neighbors from irrelevant ones or true neighbors from adversarially added fake neighbors.

Then what is the optimal sampling policy for GCNs? To answer this question, we come back to the motivation of the aggregation operation. In GCNs, each node aggregates its neighbors’ embeddings assuming that neighbors are informative for the target task. We extend this motivation to the sampling policy and sample neighbors informative for the target task. In other words, we aim for a sampler that maximizes the target task’s performance instead of minimizing sampling variance.

Here we propose PASS, a performance-adaptive sampling strategy that optimizes a sampling policy directly for task performance. PASS trains the sampling policy based on gradients of the performance loss passed through the GCN. To receive the gradients from the GCN, we need to pass them through the sampling operations, which is non-differentiable. To address this, PASS borrows the log derivative trick commonly used in the reinforcement learning community to train stochastic policies [15, 20]. PASS optimizes the sampling policy jointly with the GCN to minimize the task performance loss, resulting in a considerable performance improvement.

Graph attention networks (GATs) [21] share the same objective of learning the importance of neighbors. They select neighbors through an attention mechanism trained by back-propagating gradients of the performance loss. This mechanism was originally...
In this section, we briefly review graph convolutional networks (GCNs) then describe how sampling operations operate and solve the scalability issue in GCNs.

Notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph with $N$ nodes $v_i \in \mathcal{V}$ and edges $(v_i, v_j) \in \mathcal{E}$. Denote an adjacency matrix $A = (a(v_i, v_j)) \in \mathbb{R}^{N \times N}$ and a feature matrix $H(0) \in \mathbb{R}^{N \times D(0)}$ where $h_i^{(0)}$ denotes the $D(0)$-dimensional feature vector of node $v_i$. Table 1 gives a list of symbols and definitions.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{G} = (\mathcal{V}, \mathcal{E})$</td>
<td>input graph with nodes $v_i \in \mathcal{V}$ and edges $(v_i, v_j) \in \mathcal{E}$</td>
</tr>
<tr>
<td>$L$</td>
<td>number of layers in GCN model</td>
</tr>
<tr>
<td>$D(l)$</td>
<td>dimension of the $l$-th hidden layer where $l = 0, 1, \ldots, L$</td>
</tr>
<tr>
<td>$H(l)$</td>
<td>$N \times D(0)$ input node feature matrix</td>
</tr>
<tr>
<td>$W(l)$</td>
<td>$D(l) \times D(l+1)$ transformation matrix at the $l$-th layer where $l = 0, \ldots, L - 1$</td>
</tr>
<tr>
<td>$\alpha(\cdot)$</td>
<td>nonlinear activation function</td>
</tr>
<tr>
<td>$\alpha_{W(l)}(\cdot)$</td>
<td>abbreviation of $\alpha(W(l) \cdot \cdot)$</td>
</tr>
<tr>
<td>$p(j</td>
<td>i)$</td>
</tr>
<tr>
<td>$q(j</td>
<td>i)$</td>
</tr>
</tbody>
</table>

The GCN models stack layers of first-order spectral filters followed by a nonlinear activation functions to learn node embeddings. When $h_i^{(l)}$ denotes the hidden embeddings of node $v_i$ in the $l$-th layer, the simple and general form of GCNs is as follows [4]:

$$h_i^{(l+1)} = \alpha \left( \frac{1}{N(l)} \sum_{j \neq i} a(v_i, v_j) h_j^{(l)} W(l) \right)$$

where $a(v_i, v_j)$ is set to 1 when there is an edge from $v_i$ to $v_j$, otherwise 0. $N(l) = \sum_{i=1}^{N} a(v_i, v_j)$ is the degree of node $v_i$; $\alpha(\cdot)$ is a nonlinear function; $W(l) \in \mathbb{R}^{D(l) \times D(l+1)}$ is the learnable transformation matrix in the $l$-th layer with $D(l)$ denoting the hidden dimension at the $l$-th layer.

**Sampling operation in GCN.** GCNs require the full expansion of neighborhoods across layers, leading to high computation and memory costs. To circumvent this issue, sampling operations are added to GCNs to regulate the size of neighborhood. We first recast Equation 1 as follows:

$$h_i^{(l+1)} = \alpha_{W(l)}(\mathbb{E}_{j \neq i} a(v_i, v_j) h_j^{(l)})$$

where we combine the transformation matrix $W(l)$ into the activation function $\alpha_{W(l)}(\cdot)$ for concision; $p(j|i)$ defines

```latex
\begin{align}
\alpha_{W(l)}(\cdot) & \text{ abbreviation of } \alpha(W(l) \cdot \cdot) \\
p(j|i) & \text{ probability of sampling node } v_j \text{ given node } v_i \\
q(j|i) & \text{ approximation of } p(j|i)
\end{align}
```

Figure 1: PASS learns which neighbors are informative for the job industry classification task on the LinkedIn member-to-member network. (a) Given Member A from the “Computer software” industry, PASS learns high sampling probabilities for Members B, C, and D from similar industries but low probabilities for Members E and F from different industries. (b) Given Member G from the “Hospital & health care” industry, PASS assigns a low sampling probability to Member I, who has an unrelated career as a ‘Program Analyst’ although he works in the same industry. This shows PASS is able to determine that the attributes of Member I are different from Member G’s and thus not informative. For space efficiency, we show part of neighbors; thus, the sum of sampling probabilities does not sum to 1. See Section 6 for details.
the probability of sampling \( v_j \) given \( v_i \). Then we approximate the expectation by Monte-Carlo sampling as follows:

\[
    h_l^{(l+1)} = a_W h_l^{(l)} + \sum_{j \in P_l(i)} \frac{1}{k} \sum_{j \in P_l(i)} h_j^{(l)} = \sum_{j \in P_l(i)} \frac{1}{k} \sum_{j \in P_l(i)} h_j^{(l)} \quad l = 0, \ldots, L - 1
\]

where \( k \) is the number of sampled neighbors for each node. Now, we regulate the size of neighborhood using \( k \).

**Scalability solution.** Equations 1 and 3 describe the computation at the \( l \)-th layer in the original GCN and GCN with sampling, respectively. In Equation 1, the numbers of nodes that participate in the \( l \)-th and \((l + 1)\)-th layers are both up to \( O(N) \), resulting in a time complexity of \( O(|E|D_lD_{l+1}(l+1)) \) where \( |E| \) denotes the number of edges. On the other hand, we can regulate the number of nodes engaged at each layer in the GCN with sampling. When we set the number of nodes sampled for the \( l \)-th and \((l + 1)\)-th layers to \( k \), the number of edges engaged in Equation 3 is up to \( O(k^2) \), leading to a time complexity of \( O(k^2D_lD_{l+1}(l+1)) \). With \( k \ll N \), sampling solves the scalability issue in the GCN successfully [10, 14].

Table 2: PASS out-features competitors: comparison of our proposed PASS and existing sampling methods for GCNs.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Importance</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Learnability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Performance adaptiveness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

3 RELATED WORK

The sampling algorithms for GCNs broadly fall into two categories: node-wise sampling and layer-wise sampling.

- **Node-Wise Sampling.** The sampling distribution \( q(j|i) \) is defined as a probability of sampling node \( v_j \) given a source node \( v_i \). In node-wise sampling, each node samples \( k \) neighbors from its sampling distribution, then the total number of nodes in the \( l \)-th layer becomes \( O(k^l) \). GraphSage [8] is one of the most well-known node-wise sampling method with the uniform sampling distribution \( q(j|i) = \frac{1}{N(i)} \). GCN-BS [14] introduces a variance reduced sampler based on multi-armed bandits. GCN-BS defines an individual sampling probability \( q(j|i) \) for each edge and trains them toward minimum sampling variance.

- **Layer-Wise Sampling.** To alleviate the exponential neighbor expansion \( O(k^l) \) of the node-wise samplers, layer-wise samplers define the sampling distribution \( q(j|i_1, \ldots, i_n) \) as a probability of sampling node \( v_j \) given a set of nodes \( \{v_k\}_{k=1}^n \). Each layer samples \( k \) neighbors from their sampling distribution \( q(j|i_1, \ldots, i_n) \), then the number of sampled nodes in each layer becomes \( O(k) \). FastGCN [4] defines \( q(j|i_1, \ldots, i_n) \) proportional to the degree of the target node \( v_j \), thus every layer has independent-identical-distributions. LADIES [23] adopts the same iid as FastGCN but limits the sampling domain to the neighborhood of the sampler layer. AS-GCN [10] parameterizes the sampling distributions \( q(j|i_1, i_2, \ldots, i_n) \) with a learnable linear function. While the layer-wise samplers successfully regulate the neighbor expansion, they suffer from sparse connection problems — some nodes fail to sample any neighbors while other nodes sample their neighbors repeatedly in a given layer.

**Learnable Sampling Policy.** GraphSage [8], FastGCN [4], and LADIES [23] use the heuristic sampling probability distributions (e.g., proportional to degrees of nodes). GCN-BS [14] and AS-GCN [10] train their sampling distributions towards minimum sampling variance. They compute the optimal sampling probability model with the minimum variance theoretically, then update their sampling models towards the optimal variance.

Our proposed PASS is a learnable node-wise sampler. Table 2 compares PASS with existing sampling methods.

4 PROPOSED METHOD

What is the optimal sampling policy for GCNs? To answer this question, we come back to the motivation of the aggregation operation in GCNs. The aggregation operation intends to complement node embeddings with neighbors’ embeddings on the assumption that neighbors are informative for the target task. We extend this motivation to the sampling policy and sample neighbors informative for the target task. In other words, we train a sampler that directly maximizes the GCN performance.

The key idea behind our approach is that we learn a sampling policy by propagating gradients of the GCN performance loss through the non-differentiable sampling operation. We first describe a learnable sampling policy function and how it operates in the GCN (i.e., forward propagation) in Section 4.1. We then describe how to learn the parameters of the sampling policy by back-propagating gradients through the sampling operation in Section 4.2. Finally, we present the overall algorithm and discuss implementation considerations in Section 4.3.

4.1 Sampling Policy

Fig. 2 shows an overview of PASS. In the forward pass, PASS samples neighbors with its parameterized policy (Fig. 2(a)), then propagates their embeddings through the GCN (Fig. 2(b)). In this section, we introduce our parameterized sampling policy \( q(l)(j|i) \) that estimates the probability of sampling node \( v_j \) given node \( v_i \) at the \( l \)-th layer.

The policy \( q(l)(j|i) \) is composed of two methodologies, importance \( q_{imp}(l)(j|i) \) and random sampling \( q_{rand}(l)(j|i) \) as follows:

\[
    q_{imp}(l)(j|i) = \langle W_s \cdot h_l^{(l)} \rangle 
\]

\[
    q_{rand}(l)(j|i) = \frac{1}{N(i)} 
\]

\[
    q(l)(j|i) = a_s \cdot q_{imp}(l)(j|i), \quad q_{rand}(l)(j|i) \]

\[
    q(l)(j|i) = q(l)(j|i) \sum_{k=1}^{N(i)} q(l)(k|i) \]

where \( W_s \in \mathbb{R}^{D_{l} \times D_{l}} \) is a transformation matrix with \( D_{l} \) denoting the hidden dimension in the sampling policy and \( D_{l} \) denoting the hidden dimension of the \( l \)-th layer; \( h_l^{(l)} \) is the hidden embedding of node \( v_i \) at the \( l \)-th layer; \( N(i) \) is the degree of node \( v_i \); \( a_s \in \mathbb{R}^{1 \times 2} \) is an attention vector; and \( q_{imp}(l)(j|i) \) is normalized to sum to 1. \( W_s \) and \( a_s \) are learnable parameters of our sampling policy, which will be updated toward performance improvement.

We describe each component in the sampling policy.
Importance Sampling. The first term \( q_{imp}(j|i) \) computes the intermediate score of sampling \( v_j \) given node \( v_i \) in the \( l \)-th layer, corresponding to importance sampling. We first map the hidden embeddings \( h_i(l) \) and \( h_j(l) \) into the \( D^{(l)} \)-dimension through the transformation matrix \( W_s \), then compute the similarity between these two embeddings by dot product. We describe the intuition behind this dot product-based importance sampling in Section 5.

Random Sampling. The second term \( q_{rand}(j|i) \) assigns the same sampling probability to each node in the neighborhood. When a graph is well-clustered, nodes are connected with all informative neighbors. Then random sampling becomes effective since its randomness helps aggregate diverse neighbors, thus preventing the GCN from overfitting. By capitalizing on both importance and random samplings, our sampling policy better generalizes across various graphs. We show how random sampling complements importance sampling experimentally in Section 6.

Attention of Sampling. The attention \( a_s \) regulates the trade-off between importance sampling \( q_{imp}(j|i) \) and random sampling \( q_{rand}(j|i) \). \( a_s \) learns which sampling methodology is more effective on a given task. We initialize \( a_s \) with higher attention to the random sampling than the importance sampling and allow the model to examine a broad scope of neighbors at first.

While our sampling policy \( q^{(l)}(j|i) \) assigns a distinct sampling probability to each edge at each layer, it shares the parameters \((W_s, a_s)\) across all edges and all layers. This parameter sharing helps our model generalize and prevents the sampling policy from overfitting to the training set.

### 4.2 Training the Sampling Policy

As shown in Fig. 2(c), after a forward pass with sampling, the GCN computes the performance loss (e.g., cross-entropy for node classification) then back-propagates gradients of the loss. Next, we describe how the gradients of the loss pass through the non-differentiable sampling operation to update our sampling policy.

When \( \theta \) denotes parameters \((W, a_s)\) in our sampling policy \( q^{(l)}(j|i) \), we can write the sampling operation with \( q^{(l)}(j|i) \) as follows:

\[
h^{(l+1)}_i = a_{W(l)}[\mathbb{E}_{j \sim q^{(l)}(j|i)}[h^{(l)}_j]], \quad l = 0, \ldots, L - 1
\]

Before being fed as input to the GCN transformation, \( a_{W(l)} \), the hidden embeddings go through a non-differentiable expectation under the sampling policy, which is non-differentiable. To pass gradients of the loss through the expectation, we apply the log derivative trick [22], widely used in reinforcement learning to compute gradients of stochastic policies. Then the gradient \( \nabla_{\theta} L \) of the loss \( L \) w.r.t. the sampling policy \( q^{(l)}(j|i) \) is computed as follows:

**Theorem 4.1.** Given the loss \( L \) and the hidden embedding \( h^{(l)}_i \) of node \( v_i \) at the \( l \)-th layer, the gradient of \( L \) w.r.t. the parameter \( \theta \) of the sampling policy \( q^{(l)}(j|i) \) is computed as follows:

\[
\nabla_{\theta} L = \frac{dL}{dh_i^{(l+1)}} a_{W(l)}[\mathbb{E}_{j \sim q^{(l)}(j|i)}[\nabla_{\theta} \log q^{(l)}_\theta(j|i)h_j^{(l)}]]
\]

**Proof.** By the chain rule, \( \frac{dL}{d\theta} \) is decomposed as follows:

\[
\frac{dL}{d\theta} = \frac{dL}{dh_i^{(l+1)}} \frac{dh_i^{(l+1)}}{d\theta}
\]

We compute the gradient of \( h_i^{(l+1)} \) w.r.t. \( \theta \) as follows:

\[
\frac{d h_i^{(l+1)}}{d\theta} = \nabla_{\theta} a_{W(l)}[\mathbb{E}_{j \sim q^{(l)}(j|i)}[h_j^{(l)}]]
\]

\[
= a_{W(l)}(\nabla_{\theta} q^{(l)}_\theta(j|i)h_j^{(l)}) + a_{W(l)}[\sum_{k=0}^N q^{(l)}_\theta(u_k|i)h_{u_k}]
\]

\[
= a_{W(l)}[\sum_{k=0}^N \nabla_{\theta} q^{(l)}_\theta(u_k|i)h_{u_k}]
\]

\[
= a_{W(l)}[\sum_{k=0}^N \nabla_{\theta} q^{(l)}_\theta(u_k|i)\nabla_{\theta} \log q^{(l)}_\theta(u_k|i)h_{u_k}]
\]

\[
= a_{W(l)}[\sum_{k=0}^N \nabla_{\theta} q^{(l)}_\theta(u_k|i)\nabla_{\theta} \log q^{(l)}_\theta(u_k|i)h_{u_k}]
\]

where the nodes \( \{u_k\}_{k=0}^N \) are neighbors of \( v_i \). The log derivative trick leverages the property of the logarithm \( \nabla_{\theta} \log q_{\theta} = \nabla_{\theta} q_{\theta}/q_{\theta} \) to transform the sum into an expectation under \( q_{\theta} \) that we can sample is applied in the fourth equation.

In Theorem 4.1, we describe the gradient of the loss w.r.t the sampling policy of a single edge (i.e., sampling probability \( q^{(l)}_\theta(j|i) \) of node \( j \) given node \( i \)). In the implementation, we average the gradients \( \nabla_{\theta} L \) passed through all edges. Also, to show how the gradients w.r.t. the sampling policy is passed through GCN parameters \((\sigma_W)\)
After computing the loss, we do backpropagation and update parameters. Algorithms 1 and 2 describe how we train graph convolutional networks with our sampling policy. Our algorithm’s framework is more efficiently, we omit how the gradients pass through the ReLU. Except for the ReLU condition ($x > 0$), there is no difference in the final result.

Based on Theorem 4.1, we pass the gradients of the GCN performance loss to the sampling policy through the non-differentiable sampling operation and optimize the sampling policy for the GCN performance.

### 4.3 Algorithm

Algorithms 1 and 2 describe how we train graph convolutional networks with our sampling policy. Our algorithm’s framework is composed of three steps: 1) sampling, 2) feedforward propagation, and 3) backpropagation.

In the sampling process, we define a computation graph. The computation graph is a $L$-layer network composed of nodes and edges participating in a minibatch. We generate the computation graph using our sampling policy $q^{(L)}(j|i)$ in a top-down manner ($L : L \rightarrow 1$). When a minibatch of size $b$ is given, the $b$ nodes are located at the $L$-th layer; each node samples $k$ neighbors following the sampling policy $q^{(L)}(j|i)$; the sampled $kb$ nodes are located at the $(L-1)$-th layer; each node samples $k$ neighbors following the sampling policy $q^{(L-1)}(j|i)$; the sampled $k^2b$ nodes are located at the $(L-2)$-th layer; repeat until the 1-st layer.

After acquiring the computation graph, we do feedforward propagation in a bottom-up manner ($l : 1 \rightarrow L$), i.e., iteratively aggregate neighboring embeddings and pass them through transformations. After computing the loss, we do backpropagation and update parameters using gradients of the loss in a top-down manner ($l : L \rightarrow 1$).

In the backpropagation phase, we update the parameters of both the GCN and the sampling policy. In practice, we find that the gradients from the 1-st layer are sufficient to successfully update the sampling policy. We repeat the whole process with each minibatch.

#### 4.3.1 Implementation

In the backpropagation phase, PASS uses the log derivative trick [22] to pass gradients of the loss from the GCN to the sampling policy through an expectation operation. In reinforcement learning, the log derivative trick is used to compute the gradient of the expectation of a scalar function (e.g., a reward function) [15, 22]. However, our model applies the log derivative trick to compute the gradient of an expectation of vectors (matrices for a batch implementation) located in the middle of the neural network. The implementation of the log derivative trick in this context requires hand-crafted backpropagation. If we were to brute-force the implementation, we would need to compute $dL/dh_i^{(l)}$, compute $dh_i^{(l)}/d\theta$ using the log derivative trick, multiply them to output $dL/d\theta$, and finally update $\theta$ manually using $dL/d\theta$.

Here, we introduce an additional SUB-LOSS trick that allows us to leverage the backpropagation mechanisms built in deep learning frameworks (e.g., PyTorch, TensorFlow).

#### Theorem 4.2 (SUB-LOSS trick)

Given $\theta \in \mathbb{R}^{D(x)}$, a hidden embedding $h(\theta) \in \mathbb{R}^{D(l)}$, and a loss $L(h) \in \mathbb{R}$, the gradient of the loss $L$ w.r.t. $\theta$ is presented as follows:

$$\frac{dL}{d\theta} = \frac{dL}{dh} \cdot h$$

with an assumption $\frac{dL}{dh} = 0$.

**Proof.** Proofs are given in Appendix A.1.\)

With the SUB-LOSS trick, we compute an auxiliary loss $L_{aux} = dL/dh_i^{(l)} \cdot h_i^{(l)}$ and simply call the backpropagation function of our deep learning framework on $L_{aux}$ to compute the gradient w.r.t. $\theta$. More details are in Appendix A.1.

### 5 THEORETICAL FOUNDATION

In this section, we dissect the back-propagation process of PASS and analyze how PASS learns whether a neighbor is informative for the target task from gradients of the performance loss (i.e., why it assigns a certain sampling probability to the neighbor).

GCNs train their parameters to move the node embeddings $h_i^{(l)}$ in the direction that minimizes the performance loss $L$, i.e., the gradient $-dL/dh_i^{(l)}$. PASS promotes this process by sampling neighbors whose embeddings are aligned with the gradient $-dL/dh_i^{(l)}$. When $h_i^{(l)}$ is aggregated with the embedding $h_j^{(l)}$ of a sampled neighbor aligned with the gradient, it moves in the direction that reduces the loss $L$.

In other words, PASS decides a neighbor node $v_j$ is informative when its embedding $h_j^{(l)}$ is aligned with the gradient $-dL/dh_i^{(l)}$. In Fig.3, PASS considers $v_3$ more informative than $v_5$ since $h_3^{(l)}$ is better aligned with $-dL/dh_5^{(l)}$, thereby helping $h_3^{(l)}$ move towards loss reduction. In Theorem 5.1, we show how PASS measures the
The embedding of $v_2$ after aggregating with $v_3$.

The current embedding of $v_2$.

The direction in which $h_i^{(l)}$ should move to decrease the performance loss $L$. 

 alignment between $-d\mathcal{L}/dh_i^{(l)}$ and $h_j^{(l)}$ and how it increases the sampling probability $q(j|i)$ in proportion to this alignment.

**Theorem 5.1.** Given a source node $v_i$ and its neighbor node $v_j$, **PASS** increases a sampling probability $q(j|i)$ in proportion to the dot product of $-d\mathcal{L}/dh_i^{(l)}$ and $h_j^{(l)}$.

**Proof.** Let $z_i^{(l)} = \mathbb{E}_{j \sim q(j|i)} [h_j^{(l)}]$ denote an intermediate hidden embedding of node $v_i$ at the $l$-th layer after the aggregation operation. By the chain rule, $-d\mathcal{L}/dq(j|i)$ decomposes into $\left(-d\mathcal{L}/dz_i^{(l)}\right) \cdot \left(dz_i^{(l)}/dq(j|i)\right)$. The first component $-d\mathcal{L}/dz_i^{(l)}$ is the direction $z_i^{(l)}$ needs to move towards to decrease the loss $\mathcal{L}$. The second component $dz_i^{(l)}/dq(j|i)$ is computed as follows:

$$\frac{dz_i^{(l)}}{dq(j|i)} = \frac{d}{dq(j|i)} \mathbb{E}_{k \sim q(k|i)} [h_k^{(l)}] = \frac{d}{dq(j|i)} \sum_{k=1}^{N_{out}} q(k|i)h_{uk} = \frac{d}{dq(j|i)} \left(\sum_{k=1}^{N_{out}} (q(k|i)h_{uk}) = h_{j}^{(l)}\right)$$

where the nodes $\{u_k\}_{k=1}^{N_{out}}$ are neighbors of $v_i$. In the third equation, $\mathbb{E}_{k \sim q(k|i)} q(k|i)h_{uk}$ is zero for nodes $u_k \neq j$. Then $-d\mathcal{L}/dq(j|i)$ is presented as follows:

$$-\frac{d\mathcal{L}}{dq(j|i)} = \left(-\frac{d\mathcal{L}}{dz_i^{(l)}}\right) \cdot \frac{dz_i^{(l)}}{dq(j|i)} = \left(-\frac{d\mathcal{L}}{dz_i^{(l)}}\right) \cdot h_j^{(l)} \tag{9}$$

Since $d\mathcal{L}/dz_i^{(l)} = d\mathcal{L}/dh_i^{(l)} - d\mathcal{L}/dh_i^{(l)}$, and $h_i^{(l)}$ is decided by the dot product of $-d\mathcal{L}/dh_i^{(l)}$ and $h_j^{(l)}$. When $-d\mathcal{L}/dh_i^{(l)}$ and $h_j^{(l)}$ have similar directions, $-d\mathcal{L}/dq(j|i)$ becomes large and the probability $q(j|i)$ is updated to increase by the gradient descent. ■

In Theorem 5.1, **PASS** estimates the alignment between $-d\mathcal{L}/dh_i^{(l)}$ and $h_j^{(l)}$ from their dot product. However, the dot product as a measure of alignment prefers $h_j^{(l)}$ with a large L1 norm. To prevent this issue, we normalize $h_j^{(l)}$ in our experiments.

This reasoning process leads to two important considerations. First, it crystallizes our understanding of the aggregation operation in GCNs. The aggregation operation enables a node’s embedding to move towards its neighbors to reduce the performance loss. Second, this reasoning process shows the benefits of jointly optimizing the GCN and the sampling policy. Optimizing the sampling policy for task performance allows an embedding to choose which neighbors to move towards, leading to the minimum loss more efficiently.

### 5.1 Design of Sampling Policy

In Equation 9, $-d\mathcal{L}/dq(j|i)$ measures alignment/similarity between $-d\mathcal{L}/dz_i^{(l)}$ and $h_j^{(l)}$ by a dot product. We choose the same similarity measurer, the dot product, to estimate the importance of neighbors in our sampling policy (Equation 4). When we choose another similarity measurer, for instance, a concatenation-based measurer $a(v_i, v_j) = \alpha \cdot [W \cdot h_i^{(l)} | W \cdot h_j^{(l)}]$ used in graph attention networks (GAT), we observe up to 28% drop in accuracy (more details in Section 6). This shows a careful design of the sampling policy has a large impact on performance.

### 6 EXPERIMENTS

In this section, we evaluate the performance of **PASS** compared to state-of-the-art sampling algorithms on GCNs.

#### 6.1 Experimental setting

We compare the performance of **PASS** and other sampling algorithms on semi-supervised node classification tasks. All experiments were conducted on the same p2.xlarge Amazon EC2 instance. **Datasets.** We use seven public datasets — three citation networks (Cora, Citeseer, and Pubmed) [17], two co-purchase graphs (Amazon Computers and Amazon Photo) [18], and two co-authorship graphs (MS CS and MS Physics) [18]. In addition, we also evaluate on a subset of LinkedIn social networks where nodes are alumni from a US university, and edges are connections between them. We use members’ latest job title and their industry as labels. We split 50%/10%/40% of the datasets into the training/validation/test sets, respectively. We report their statistics in Table 3. **Baselines.** We compare **PASS** with four state-of-the-art sampling methods: GraphSage [8], FastGCN [4], AS-GCN [10], and GCN-Bs [14]; and one attention method: GAT [21]. For fair comparison, all methods share the same network structure, two-hidden-layer GCN with all hidden dimensions set to 64. Please refer to Appendix A.2 for more details. **Unified time complexity bound.** With the batch size set to 64, layer-wise sampling methods (FastGCN, AS-GCN) sample 64 nodes per layer. For a fair comparison, node-wise sampling methods

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Features</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>2,485</td>
<td>5,069</td>
<td>1,433</td>
<td>7</td>
</tr>
<tr>
<td>Citeseer</td>
<td>2,110</td>
<td>3,668</td>
<td>3,703</td>
<td>6</td>
</tr>
<tr>
<td>Pubmed</td>
<td>19,717</td>
<td>44,324</td>
<td>500</td>
<td>3</td>
</tr>
<tr>
<td>AmazonC</td>
<td>13,381</td>
<td>245,778</td>
<td>767</td>
<td>10</td>
</tr>
<tr>
<td>AmazonP</td>
<td>7,487</td>
<td>119,043</td>
<td>745</td>
<td>8</td>
</tr>
<tr>
<td>MsCS</td>
<td>18,333</td>
<td>81,894</td>
<td>6,805</td>
<td>15</td>
</tr>
<tr>
<td>MsPhysics</td>
<td>34,493</td>
<td>247,962</td>
<td>8,415</td>
<td>5</td>
</tr>
<tr>
<td>LinkedIn</td>
<td>39K+</td>
<td>1.7M+</td>
<td>20+</td>
<td>(industry) ~150 (title) ~8,000</td>
</tr>
</tbody>
</table>
We measure the accuracy of each sampling algorithm on the node classification tasks. In Table 4, our proposed PASS shows the highest accuracy among all baselines across all datasets. Layer-wise methods (FastGCN, AS-GCN) show lower accuracy than node-wise methods (GraphSage, GCN-BS, PASS).

Layer-wise samplers define the probability of sampling node \( v_j \) given a set of source nodes \( \{ v_k \}_{k=1}^{l_n} \), as \( q(j|v_1, \ldots, v_{l_n}) \). Since they sample from a union pool of each source node’s neighborhood, there is no guarantee for each source node to fairly sample their neighbors. Moreover, a node’s sampling probability in a layer-wise sampler is proportional to its degree, which follows a power-law distribution [6]. If a source node \( v_i \)’s neighbors all have smaller degrees than neighbors of other source nodes, none of \( v_i \)’s neighbors are likely to be sampled, and \( v_i \) fails to aggregate any neighbor information. This results in sparse connections between layers and poor performance for layer-wise samplers.

Among node-wise sampling methods, PASS outperforms GraphSage and GCN-BS. One interesting result is that GraphSage, which just samples neighbor randomly, still shows good performance among carefully-designed sampling algorithms. The seven public datasets are well-clustered; thus there is not much room to be improved by importance sampling. In the following Section, we show when the graphs have noise (e.g., random connections between different communities), GraphSage plummets in accuracy. GCN-BS shows higher accuracy than other baselines but lower than our method. While PASS learns a shared sampling policy across all edges with the performance loss, GCN-BS trains individual sampling probabilities for each edge with variance reduction loss. This result presents the effectiveness of the parameter sharing and the performance loss of PASS.

### 6.3 Robustness

To examine the robustness of sampling algorithms, we inject noise into graphs. We investigate two different noise scenarios: 1) fake connections among existing nodes, and 2) fake neighbors with random feature vectors. These two scenarios are common in real-world graphs. The first ‘fake connection’ scenario simulates connections made by mistake or unfit for purpose (e.g., connections between family members in a job search platform). The second scenario simulates fake accounts with random attributes used for fraudulent activities. For each node, we generate five true neighbors and five fake neighbors for each scenario. We keep the rest of the experimental setting as in Section 6.1.

Table 5 shows that PASS consistently has high accuracy across all scenarios, while the performance of all other methods plummets. The sparse connection problems of layer-wise sampling methods (FastGCN, AS-GCN) become worse with graph noise. Node-wise sampling methods also show much lower accuracy than on the original graphs (Table 4). GraphSage gives the same sampling probability to true neighbors and fake neighbors, resulting in a sharp drop in accuracy. GCN-BS is likely to sample high-degree or dense-feature nodes, which help stabilize the sampling variance, regardless of their relationship with the source node. Thus GCN-BS fails to distinguish fake neighbors from true neighbors. On the other hand, PASS learns which neighbors are informative or fake from gradients of the performance loss (Theorem 5.1). These results show that the optimization of the sampling policy toward performance brings robustness to graph noise.
Table 6: Comparison with GATs: PASS is scalable across all datasets while GATs run out of memory on Pubmed, Amazon Computer, MS CS, and MS Physics datasets. We run PASS with both 1 and 5 sampled neighbors, trading-off speed for accuracy. On the few datasets where GATs are applicable, PASS (5) shows comparable or higher accuracy as GATs with considerably shorter training and test times.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>GATs (1)</th>
<th>PASS (1)</th>
<th>GATs (5)</th>
<th>PASS (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>0.850</td>
<td>0.821</td>
<td>0.847</td>
<td></td>
</tr>
<tr>
<td>Citeseer</td>
<td>0.744</td>
<td>0.715</td>
<td>0.735</td>
<td></td>
</tr>
<tr>
<td>Pubmed</td>
<td>-</td>
<td>0.858</td>
<td>0.871</td>
<td></td>
</tr>
<tr>
<td>AmazonC</td>
<td>-</td>
<td>0.757</td>
<td>0.886</td>
<td></td>
</tr>
<tr>
<td>AmazonP</td>
<td>0.905</td>
<td>0.855</td>
<td>0.944</td>
<td></td>
</tr>
<tr>
<td>MS CS</td>
<td>-</td>
<td>0.884</td>
<td>0.918</td>
<td></td>
</tr>
<tr>
<td>MS Physics</td>
<td>-</td>
<td>0.934</td>
<td>0.952</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>GATs (1)</th>
<th>PASS (1)</th>
<th>GATs (5)</th>
<th>PASS (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td></td>
<td></td>
<td>189.670</td>
<td>9.459</td>
</tr>
<tr>
<td>Citeseer</td>
<td></td>
<td></td>
<td>404.904</td>
<td>13.962</td>
</tr>
<tr>
<td>Pubmed</td>
<td></td>
<td></td>
<td>87.660</td>
<td>94.918</td>
</tr>
<tr>
<td>AmazonC</td>
<td></td>
<td></td>
<td>52.060</td>
<td>184.522</td>
</tr>
<tr>
<td>AmazonP</td>
<td></td>
<td></td>
<td>1689.690</td>
<td>101.840</td>
</tr>
<tr>
<td>MS CS</td>
<td></td>
<td></td>
<td>439.378</td>
<td>507.816</td>
</tr>
<tr>
<td>MS Physics</td>
<td></td>
<td></td>
<td>439.378</td>
<td>507.816</td>
</tr>
</tbody>
</table>

Table 7: Ablation study: Our dot-product-based importance sampling $q_{\text{imp}}$ outperforms the GAT-version importance sampling mechanism. Random sampling $q_{\text{rand}}$ complements importance sampling $q_{\text{imp}}$.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>GAT-version</th>
<th>$q_{\text{imp}}$</th>
<th>$q_{\text{imp}} + q_{\text{rand}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>0.574</td>
<td>0.779</td>
<td>0.821</td>
</tr>
<tr>
<td>Citeseer</td>
<td>0.456</td>
<td>0.706</td>
<td>0.715</td>
</tr>
<tr>
<td>Pubmed</td>
<td>0.606</td>
<td>0.862</td>
<td>0.858</td>
</tr>
<tr>
<td>AmazonC</td>
<td>0.482</td>
<td>0.746</td>
<td>0.757</td>
</tr>
<tr>
<td>AmazonP</td>
<td>0.575</td>
<td>0.854</td>
<td>0.855</td>
</tr>
<tr>
<td>MS CS</td>
<td>0.661</td>
<td>0.883</td>
<td>0.884</td>
</tr>
<tr>
<td>MsPhysics</td>
<td>0.683</td>
<td>0.933</td>
<td>0.934</td>
</tr>
</tbody>
</table>

6.4 Convergence & Variance

In this section, we analyze the convergence and variance of sampling algorithms across epochs. We train each algorithm 5 times and plot the mean and standard deviation. PASS shows the highest accuracy by a significant margin (+5.5%) with a slightly higher variance (0.5 – 1.2%) than baselines (0.1 – 0.6%). Static algorithms, including GraphSage and FastGCN, show low variance since their sampling policy is decided heuristically and fixed. Learnable algorithms, including AS-GCN and GCN-BS, show low variance because their sampling policy is optimized for variance reduction. On the other hand, PASS optimizes for performance improvement. PASS explores neighbors to find the most informative, leading to slightly higher variance and much higher accuracy than baselines that exploit the neighbors that minimize variance.

6.5 Comparison with GAT

In this section, we compare PASS with GATs. PASS and GATs share the same objective of learning the importance of neighbors, respectively through sampling probabilities and attention scores. While PASS solves the scalability issues of GCNs with sampling, GATs suffer from high computation and memory footprints. To investigate their scalability, we train GATs and PASS on a GPU with 16GB of memory. We run PASS with both 1 and 5 sampled neighbors (denoted as PASS-1 and PASS-5), trading-off speed for accuracy.

Table 6 shows PASS is scalable across all datasets while GAT runs out of memory on the Pubmed, Amazon Computer, MS CS, and MS Physics datasets. On the few datasets where GATs are applicable (Cora, Citeseer, and Amazon Photo), PASS-1 shows up to ×60 shorter training time and ×8 shorter test time than GAT while having 5% lower accuracy. When sampling more neighbors to increase accuracy at the price of speed, PASS-5 shows comparable or higher accuracy as GATs while maintaining shorter training and test times. On the Amazon Photo dataset, where neighbors have 20 neighbors on average, PASS-5 shows 5% higher accuracy than GATs while only sampling 5 neighbors when GATs consider the full neighborhood. This shows PASS is scalable and learns neighbors informative for performance improvement.

6.6 Ablation Study

In this section, we examine the effectiveness of importance and random sampling in PASS. We compare the performance of our dot-product-based importance sampling with the importance sampling mechanism introduced in GATs, presented as

$$q_{\text{GAT}}^{(l)}(j|i) = a : [W \cdot h_{j}^{(l)}][W \cdot h_{j}^{(l)}]$$

where $W$ and $a$ are a trainable $(D^{(s)} \times D^{(l)})$ matrix and a $(1 \times 2D^{(s)})$ vector, respectively.

Table 7 shows our dot-product-based importance sampling outperforms the GATs version by up to 27.9% accuracy. The addition of random sampling improves accuracy by up to another 4.2% accuracy as the noise helps aggregate diverse neighbors (i.e., exploration), preventing the GCN from overfitting.
which neighbors are informative for the task. Given Member A from the "Computer software" industry, PASS learns high sampling probabilities for Members B, C, and D from similar industries but low probabilities for Members E and F from different industries. Given Member G from the "Hospital & health care" industry, PASS assigns a low sampling probability to Member I, who has an unrelated career as a "Program Analyst" although he works in the same industry. This shows PASS is able to determine that the attributes of Member I are different from Member G’s and thus not informative. These case studies show the effectiveness of PASS at identifying informative neighbors on real-world graphs. Additional case studies on the Cora and Amazon photo datasets are in Appendix A.

6.7 Case Study

Figure 1 shows a case study where PASS is used to classify the job industry of nodes in the LinkedIn social network. PASS learns which neighbors are informative for the task. Given Member A from the "Computer software" industry, PASS learns high sampling probabilities for Members B, C, and D from similar industries but low probabilities for Members E and F from different industries. Given Member G from the "Hospital & health care" industry, PASS assigns a low sampling probability to Member I, who has an unrelated career as a "Program Analyst" although he works in the same industry. This shows PASS is able to determine that the attributes of Member I are different from Member G’s and thus not informative. These case studies show the effectiveness of PASS at identifying informative neighbors on real-world graphs. Additional case studies on the Cora and Amazon photo datasets are in Appendix A.

6.8 Visualization of PASS

In Section 5, we saw that PASS decides whether a neighbor $v_j$ is informative based on the alignment between its embedding $h_{l-1}^{(j)}$ and the gradient $-d\mathcal{L}/dh_{l-1}^{(i)}$ of the loss w.r.t. the source node $v_i$. Figure 5 shows the hidden-layer embeddings projected to 2D via t-SNE [3]. Numbers denote the increase/decrease in sampling probabilities. The neighbors in the red area, which are close to the gradient (the red cross), see an increase in their sampling probabilities. Conversely, the neighbors in the blue area, which are far from the gradient, see a decrease in their sampling probabilities. This result shows our theoretical foundation holds on real-world datasets.

7 CONCLUSION

In this paper, we propose a novel sampling algorithm PASS for graph convolutional networks. Our main contributions are:

- **Performance-adaptive sampler**: PASS samples neighbors informative for the task performance.
- **Effectiveness**: PASS outperforms state-of-the-art samplers, being up to 10.4% more accurate.
- **Robustness**: PASS shows up to 53.1% higher accuracy than the baselines in the presence of adversarial attacks.
- **Theoretical foundation**: PASS explains why a neighbor is considered informative and assigned a high sampling probability.

Future works include learning an edit imputation policy and combining it with our proposed edge sampling policy to improve the overall performance of graph neural networks.

REFERENCES

A APPENDIX

A.1 Proof of SUB-LOSS trick

Our model applies the log derivative trick to compute the gradient of an expectation of vectors (matrices for a batch implementation) located in the middle of the neural network. The implementation of the log derivative trick in this context requires hand-crafted backpropagation. Here, we introduce an additional SUB-LOSS trick that allows us to leverage the backpropagation mechanisms built in deep learning frameworks (e.g., PyTorch, TensorFlow).

**Theorem A.1 (SUB-LOSS trick).** Given \( \theta \in \mathbb{R}^{D^{(s)}} \), a hidden embedding \( h(\theta) \in \mathbb{R}^{D^{(l)}} \), and a loss \( L(h) \in \mathbb{R} \), the gradient of the loss \( L \) with respect to \( \theta \) is presented as follows:

\[
\frac{dL}{d\theta} = \frac{dL}{dh} \cdot \frac{dL}{d\theta} \cdot h
\]

with an assumption \( \frac{dL}{d\theta} \cdot \frac{dL}{dh} = 0 \).

**Proof.** By the chain rule, \( \frac{dL}{d\theta} \) is decomposed as follows:

\[
\frac{dL}{d\theta} = \frac{dL}{dh} \cdot \frac{dL}{d\theta}
\]

where \( \frac{dL}{dh} \) is an \((1 \times D^{(l)})\) matrix and \( \frac{dL}{d\theta} \) is a \((D^{(l)} \times D^{(s)})\) matrix. The \( i \)-th component of \( \frac{dL}{d\theta} \) is presented as follows:

\[
\frac{dL}{d\theta_i} = \sum_{j=0}^{L_{aux}} \frac{dL}{dh_j} \cdot \frac{dL}{d\theta_i}
\]

The dot product of \( \frac{dL}{dh} \) and \( h \) is presented as follows:

\[
\frac{dL}{dh} \cdot h = \frac{dL}{dh_0} \cdot h_0 + \frac{dL}{dh_1} \cdot h_1 + \cdots + \frac{dL}{dh_{(L^{(l)}-1)}} \cdot h_{(L^{(l)}-1)}
\]

where \( \frac{dL}{dh} \cdot h \) is a scalar value. With an assumption \( \frac{dL}{d\theta_i} \cdot \frac{dL}{dh} = 0 \), the gradient of \( \frac{dL}{dh} \cdot h \) with respect to \( \theta_i \) is presented as follows:

\[
\frac{dL}{d\theta_i} \cdot \frac{dL}{dh} = \frac{dL}{dh_0} \cdot \frac{dL}{d\theta_i} + \frac{dL}{dh_1} \cdot \frac{dL}{d\theta_i} + \cdots + \frac{dL}{dh_{(L^{(l)}-1)}} \cdot \frac{dL}{d\theta_i}
\]

\[
\frac{dL}{d\theta_i} = \sum_{j=0}^{L_{aux}} \frac{dL}{dh_j} \cdot \frac{dL}{d\theta_i}
\]

Then \( \frac{dL}{d\theta_i} = \frac{dL}{dh} \cdot h \) for every \( 0 \leq i < D^{(s)} \). This shows \( \frac{dL}{d\theta} \) is equal to \( \frac{dL}{dh} \cdot h \).

With the SUB-LOSS trick, we compute an auxiliary loss \( L_{aux} = dL/dh_i \cdot h_i \) and simply call the backpropagation function of our deep learning framework on \( L_{aux} \) to compute the gradient w.r.t. \( \theta \).

A.2 Experimental Setting

**Hyper-parameters.** We use the Adam optimizer [11] and tune each baseline with a grid search on each dataset. Most baselines perform best on most datasets with a learning rate of 0.01, weight decay of \( 5 \times 10^{-4} \). We report the average performance across 5 runs for each experiment.

**Baselines.** We refer to the following websites when implementing the baseline models:

- **FastGCN:** https://github.com/matenure/FastGCN
- **AS-GCN:** https://github.com/huangwb/AS-GCN
- **GraphSAGE:** https://github.com/williamleif/GraphSAGE
- **GCN-BS:** https://github.com/xavierzw/ogb-geniepath-bs
- **GAT:** https://github.com/PetarV-/GAT

A.3 Case Study

In Figure 6(a), we find PASS distinguishes informative neighbors (same labels) from less informative ones (different labels). The node 464 with label 0 has a high sampling probability (0.27) for the node 2240 with label 0 while low probabilities (0.09) for the nodes 1940 and 1892 with different labels. In Figure 6(b), the node 643 with label 1 gives a high sampling probability (0.21) to the node 6572 with different label 4. The node 6572 has a high sampling probability (0.20) for the neighbor node 5454 with label 1; thus, the node 6572 contains information of label 1. In the two-layer GCNs, the node 643 aggregates the node 5454 through the node 6572 and supplements its embedding with another label 1 node.

A.4 Different sample numbers

In Section 6, we sample one neighbor per node for a fair comparison between layer-wise samplers and node-wise samplers. With the batch size set to 64, both the layer-wise and node-wise samplers samples 64 nodes in total for each layer. Under the same time/memory efficiency, the node-wise samplers outperform the layer-wise sampler in accuracy. Here, we compare the performance of the node-wise samplers with larger numbers of samples \( k > 1 \).

In Table 8, PASS shows higher or similar accuracy with its competitors. The accuracy gap between PASS and its competitors is smaller than when the sampling number is 1. The large sample number
Table 8: Node-wise samplers with large numbers of samples.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#sample</th>
<th>GraphSage</th>
<th>GCN-BS</th>
<th>PASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>3</td>
<td>0.845</td>
<td>0.840</td>
<td>0.844</td>
</tr>
<tr>
<td>Citeseer</td>
<td>3</td>
<td>0.740</td>
<td>0.708</td>
<td>0.735</td>
</tr>
<tr>
<td>Pubmed</td>
<td>3</td>
<td>0.839</td>
<td>0.877</td>
<td>0.874</td>
</tr>
<tr>
<td>AmazonC</td>
<td>5</td>
<td>0.844</td>
<td>0.880</td>
<td>0.889</td>
</tr>
<tr>
<td>AmazonC</td>
<td>10</td>
<td>0.862</td>
<td>0.898</td>
<td>0.885</td>
</tr>
<tr>
<td>AmazonP</td>
<td>5</td>
<td>0.900</td>
<td>0.919</td>
<td>0.942</td>
</tr>
<tr>
<td>AmazonP</td>
<td>10</td>
<td>0.923</td>
<td>0.937</td>
<td>0.945</td>
</tr>
<tr>
<td>MsCS</td>
<td>3</td>
<td>0.862</td>
<td>0.909</td>
<td>0.912</td>
</tr>
</tbody>
</table>

allows the informative neighbors sampled at some point by GraphSage and GCN-BS. Thus the accuracy of GraphSage and GCN-BS could catch up with our accuracy, not surpass ours. In addition, the accuracy is saturated around 0.88 and 0.94 from the sampling number 5 on the AmazonC and AmazonP datasets, respectively. This shows the number of informative neighbors is under 5. Thus sampling neighbors more than 5 does not bring further increase in accuracy. These results show that our experimental setting with one sample per node is more effective at comparing the performance of the sampling algorithms.