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:
Minji Yoon[1], Théophile Gervet[1], Baoxu Shi[2], Sufeng Niu[2], Qi He[2], Jaewon Yang[2] . 2021. Performance-Adaptive Sampling Strategy Towards Fast and Accurate Graph Neural Networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467284

1 INTRODUCTION

Graph convolutional networks (GCN) [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].

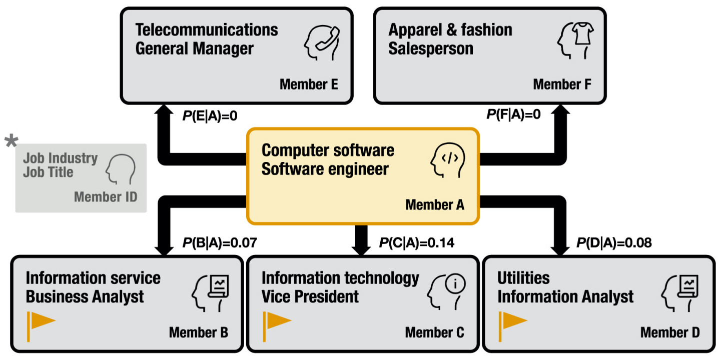
Most samplers minimize the variance in sampling to approximate the original aggregation of the full neighborhood [4, 10, 14, 23]. These sampling policies learn neighbors helpful for variance reduction, not neighbors informative for the target task's performance. Thus, those variance reduction-oriented 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.

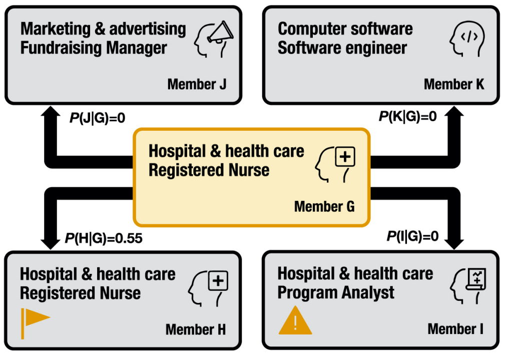
Here we propose PASS, a performance-adaptive sampling strategy that optimizes a sampling policy directly for task performance. PASS trains the sampling policy based on gradients of the performance loss passed through the GCN. To receive the gradients from the GCN, we need to pass them through the sampling 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

KDD '21, August 14–18, 2021, Virtual Event, Singapore.
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8332-5/21/08.
https://doi.org/10.1145/3447548.3467284



(a) Neighborhood for a software engineer member



(b) Neighborhood for a registered nurse member

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.

designed as a continuous approximation of the non-differentiable hard selection (i.e., sampling) operation [1, 21]. However, GATs suffer from the same scalability issues as GCNs. Since sampling is inevitable in large scale graphs, we embed the informative neighbor selection directly in the sampler, instead of approximating it downstream with an attention mechanism. In our experiments, we show how PASS not only alleviates scalability issues of GATs but also shows higher performance.

Another advantage of PASS compared to previous sampling-based methods is that we provide theoretical foundations on how sampling policy is updated to optimize the task performance. While other samplers present the back-propagation algorithm to learn the sampling policy as a black box, PASS cracks it open. We present a transparent reasoning process on how PASS learns whether a neighbor is informative from the back-propagated gradients and why it assigns a certain sampling probability to a neighbor.

Through extensive experiments on seven public benchmarks and one LinkedIn production dataset, we demonstrate the superior performance of PASS over existing sampling algorithms. We also present various case studies examining the effectiveness of PASS on real-world datasets (Figure 1). Our main contributions are:

- **Performance-adaptiveness:** PASS learns a sampling policy that 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 presents a transparent reasoning process on how it learns whether a neighbor is informative.

2 PRELIMINARIES

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 1: Commonly used notation.

Symbol	Definition
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	input graph with nodes $v_i \in \mathcal{V}$ and edges $(v_i, v_j) \in \mathcal{E}$
L	number of layers in GCN model
$D^{(l)}$	dimension of the l-th hidden layer where $l = 0, 1, \cdots, L$
$H^{(0)}$	$N \times D^{(0)}$ input node feature matrix
$H^{(l)}$	$N \times D^{(l)}$ hidden embeddings at the l-th layer
$W^{(l)}$	$D^{(l)} \times D^{(l+1)}$ transformation matrix at the l-th layer where $l = 0, \cdots, L-1$
$\alpha(\cdot)$	nonlinear activation function
$\alpha_{W^{(l)}}(\cdot)$	abbreviation of $\alpha(W^{(l)} \cdot \cdot)$
$p(j\|i)$	probability of sampling node v_j given node v_i
$q(j\|i)$	approximation of $p(j\|i)$

GCN. 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]:

$$h_i^{(l+1)} = \alpha\Big(\frac{1}{N(i)} \sum_{j=1}^{N} a(v_i, v_j) h_j^{(l)} W^{(l)}\Big), \quad l = 0, \ldots, L-1 \tag{1}$$

where $a(v_i, v_j)$ is set to 1 when there is an edge from v_i to v_j, otherwise 0. $N(i) = \sum_{j=1}^{N} a(v_i, v_j)$ 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:

$$h_i^{(l+1)} = \alpha_{W^{(l)}}(\mathbb{E}_{j \sim p(j|i)}[h_j^{(l)}]), \quad l = 0, \ldots, L-1 \tag{2}$$

where we combine the transformation matrix $W^{(l)}$ into the activation function $\alpha_{W^{(l)}}(\cdot)$ for concision; $p(j|i) = \frac{a(v_i, v_j)}{N(i)}$ defines the probability of sampling v_j given v_i. Then we approximate the expectation by Monte-Carlo sampling as follows:

$$h_i^{(l+1)} = \alpha_{W^{(l)}}(\frac{1}{k}\sum_{j\sim p(j|i)}^{k} h_j^{(l)}), \quad l = 0, \dots, L-1 \tag{3}$$

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 $O(|\mathcal{E}|D^{(l)}D^{(l+1)})$ where $|\mathcal{E}|$ 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 $O(k^2D^{(l)}D^{(l+1)})$. 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.

Property \ Method	GraphSage [8]	FastGCN [4]	LADIES [23]	AS-GCN [10]	GCN-BS [14]	**PASS**
Importance Sampling		✓	✓	✓	✓	✓
Learnability				✓	✓	✓
Performance adaptiveness						✓

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 $q(j|i_1, \cdots, i_n)$ as a probability of sampling node v_j given a set of nodes $\{v_k\}_{k=i_1}^{i_n}$. Each layer samples k neighbors from their sampling distribution $q(j|i_1, \cdots, i_n)$, then the number of sampled nodes in each layer becomes $O(k)$. FastGCN [4] defines $q(j|i_1, \cdots, i_n)$ 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 $q(j|i_1, i_2, \dots, i_n)$ 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 sampling 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:

$$q_{imp}^{(l)}(j|i) = (\mathbf{W}_s \cdot h_i^{(l)}) \cdot (\mathbf{W}_s \cdot h_j^{(l)}) \tag{4}$$

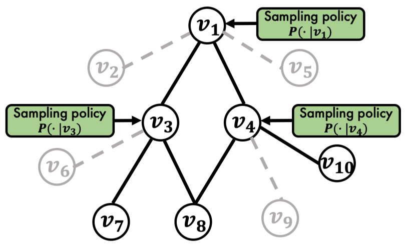
$$q_{rand}^{(l)}(j|i) = \frac{1}{N(i)} \tag{5}$$

$$\tilde{q}^{(l)}(j|i) = a_s \cdot [q_{imp}^{(l)}(j|i), \quad q_{rand}^{(l)}(j|i)] \tag{6}$$

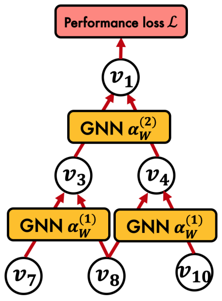
$$q^{(l)}(j|i) = \tilde{q}^{(l)}(j|i) / \sum_{k=1}^{N(i)} \tilde{q}^{(l)}(k|i) \tag{7}$$

where $\mathbf{W}_s \in \mathbb{R}^{D^{(s)} \times D^{(l)}}$ is a transformation matrix with $D^{(s)}$ denoting the hidden dimension in the sampling policy and $D^{(l)}$ denoting the hidden dimension of the l-th layer; $h_i^{(l)}$ 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 $q^{(l)}(\cdot|i)$ is normalized to sum to 1. $\mathbf{W}_s$ and a_s are learnable parameters of our sampling policy, which will be updated toward performance improvement.

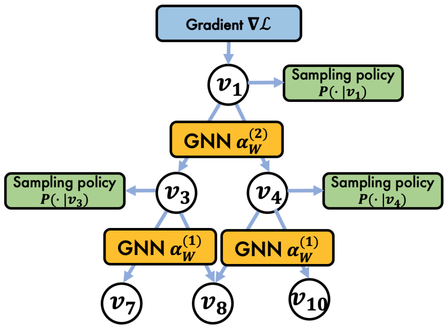
We describe each component in the sampling policy.



(a) Sampling process



(b) Forward propagation



(c) Backward propagation

Figure 2: PASS is composed of three steps: 1) sampling, 2) feedforward propagation, and 3) backpropagation. In the backpropagation process, the GCN and the sampling policy are optimized jointly to minimize the GCN performance loss.

Importance Sampling. The first term $q^{(l)}_{imp}(j|i)$ 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 $h^{(l)}_i$ and $h^{(l)}_j$ into the $D^{(s)}$-dimension through the transformation matrix $\mathbf{W}_s$, 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 $q^{(l)}_{rand}(j|i)$ assigns the same sampling probability to each node 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 $q^{(l)}_{imp}(j|i)$ and random sampling $q^{(l)}_{rand}(j|i)$. 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 $(\mathbf{W}_s, a_s)$ 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 θ denotes parameters $(\mathbf{W}_s, a_s)$ in our sampling policy $q^{(l)}_\theta$, we can write the sampling operation with $q^{(l)}_\theta(j|i)$ as follows:

$$h^{(l+1)}_i = \alpha_{W^{(l)}}(\mathbb{E}_{j\sim q^{(l)}_\theta(j|i)}[h^{(l)}_j]), \quad l = 0, \dots, L-1 \tag{8}$$

Before being fed as input to the GCN transformation, $\alpha_{W^{(l)}}$, 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 $\nabla_\theta \mathcal{L}$ of the loss $\mathcal{L}$ w.r.t. the sampling policy $q^{(l)}_\theta(j|i)$ is computed as follows:

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

$$\nabla_\theta \mathcal{L} = \frac{d\mathcal{L}}{dh^{(l+1)}_i}\alpha_{W^{(l)}}\mathbb{E}_{j\sim q^{(l)}_\theta(j|i)}[\nabla_\theta \log q^{(l)}_\theta(j|i) h^{(l)}_j]$$

Proof. By the chain rule, $\frac{d\mathcal{L}}{d\theta}$ is decomposed as follows:

$$\frac{d\mathcal{L}}{d\theta} = \frac{d\mathcal{L}}{dh^{(l+1)}_i}\frac{dh^{(l+1)}_i}{d\theta}$$

We compute the gradient of $h^{(l+1)}_i$ w.r.t. θ as follows:

$$\begin{aligned}\frac{dh^{(l+1)}_i}{d\theta} &= \nabla_\theta \alpha_{W^{(l)}}(\mathbb{E}_{j\sim q^{(l)}_\theta(j|i)}[h^{(l)}_j]) \\ &= \alpha_{W^{(l)}}(\nabla_\theta \sum_{k=0}^{N(i)} q^{(l)}_\theta(u_k|i) h^{(l)}_{u_k}) \\ &= \alpha_{W^{(l)}}(\sum_{k=0}^{N(i)} \nabla_\theta q^{(l)}_\theta(u_k|i) h^{(l)}_{u_k}) \\ &= \alpha_{W^{(l)}}(\sum_{k=0}^{N(i)} q^{(l)}_\theta(u_k|i) \nabla_\theta \log q^{(l)}_\theta(u_k|i) h^{(l)}_{u_k}) \\ &= \alpha_{W^{(l)}}(\mathbb{E}_{j\sim q^{(l)}_\theta(j|i)}[\nabla_\theta \log q^{(l)}_\theta(j|i) h^{(l)}_j])\end{aligned}$$

where the nodes $\{u_k\}_{k=1}^{N(i)}$ are neighbors of v_i. The log derivative trick leveraging the property of the logarithm $\nabla_\theta \log q_\theta = \nabla_\theta q_\theta / q_\theta$ to tranform the sum into an expectation under q_θ 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 $\nabla_\theta \mathcal{L}$ passed through all edges. Also, to show how the gradients w.r.t. the sampling policy is passed through GCN parameters (σ_W)

Algorithm 1: One minibatch in PASS

```
Require: a minibatch of labeled nodes: $\{v_i, y_i\}_{i=1}^{b}$, sample number: $k$,
    GCN model: $\{W^{(l)}\}_{l=1}^{L-1}$, sampling policy: $q^{(l)}(j|i)$
Ensure: updated GCN model and sampling policy
 1: $\mathcal{G}_{comp} = Sampler(\{v_i\}_{i=1}^{b}, q^{(l)}(j|i), k)$
 2: for $l$ from 1 to $L-1$ do
 3:     for $v_i$ in $\mathcal{G}_{comp}[l+1]$ do
 4:         $Neighbor(v_i)$ = neighbor nodes of $v_i$ in $\mathcal{G}_{comp}[l]$
 5:         $h_i^{(l+1)} = \alpha(\sum_{j \in Neighbor(v_i)} h_j^{(l)} W^{(l)})$
 6:     end for
 7: end for
 8: $\mathcal{L}$ = loss($\{h_i^{(L)}, y_i\}_{i=1}^{b}$)
 9: for $l$ from $L-1$ to 1 do
10:     update $W^{(l)}$ using $\nabla_{W^{(l)}} \mathcal{L}$
11:     update $q^{(l)}(j|i)$ using $\nabla_{q^{(l)}(j|i)} \mathcal{L}$
12: end for
13: return $\{W^{(l)}\}_{l=1}^{L-1}$ and $q^{(l)}(j|i)$
```

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 ($l : L \longrightarrow 1$). 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 ($l : 1 \longrightarrow L$), 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 ($l : L \longrightarrow 1$). 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

Algorithm 2: Sampler

```
Require: a minibatch of nodes: $\{v_i\}_{i=1}^{b}$, sampling policy: $q^{(l)}(j|i)$,
    sample number: $k$
Ensure: computation graph $\mathcal{G}_{comp}$
 1: $\mathcal{G}_{comp}[L] = \{v_i\}_{i=0}^{b}$
 2: for $l = L-1$ to 1 do
 3:     for $v_i$ in $\mathcal{G}_{comp}[l+1]$ do
 4:         $\mathcal{G}_{comp}[l] = \mathcal{G}_{comp}[l] + \{v_{u_j}\}_{j=1}^{k} \sim q^{(l)}(u_j|i)$
 5:     end for
 6: end for
 7: return $\mathcal{G}_{comp}$
```

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 $d\mathcal{L}/dh_i^{(l)}$, compute $dh_i^{(l)}/d\theta$ using the log derivative trick, multiply them to output $d\mathcal{L}/d\theta$, and finally update θ manually using $d\mathcal{L}/d\theta$.

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

$$\frac{d\mathcal{L}}{d\theta} = \frac{d}{d\theta}(\frac{d\mathcal{L}}{dh} \cdot h)$$

with an assumption $\frac{d}{d\theta}(\frac{d\mathcal{L}}{dh}) = 0$.

PROOF. Proofs are given in Appendix A.1 ∎

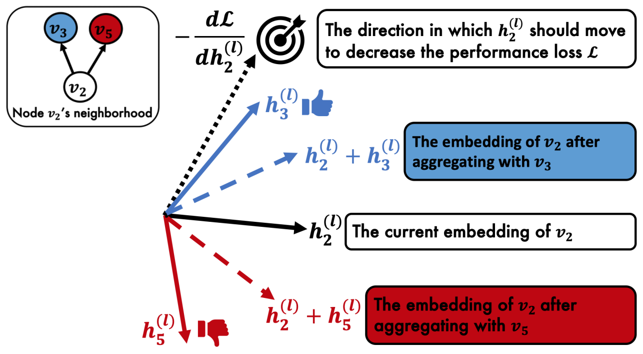
With the SUB-LOSS trick, we compute an auxiliary loss $\mathcal{L}_{aux} = d\mathcal{L}/dh_i^{(l)} \cdot h_i^{(l)}$ and simply call the backpropagation function of our deep learning framework on $\mathcal{L}_{aux}$ to compute the gradient w.r.t. θ. More details are in Appendix A.1.

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 $\mathcal{L}$, i.e., the gradient $-d\mathcal{L}/dh_i^{(l)}$. PASS promotes this process by sampling neighbors whose embeddings are aligned with the gradient $-d\mathcal{L}/dh_i^{(l)}$. 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 $\mathcal{L}$.

In other words, PASS decides a neighbor node v_j is informative when its embedding $h_j^{(l)}$ is aligned with the gradient $-d\mathcal{L}/dh_i^{(l)}$. In Fig.3, PASS considers v_3 more informative than v_5 since $h_3^{(l)}$ is better aligned with $-d\mathcal{L}/dh_2^{(l)}$, thereby helping $h_2^{(l)}$ move towards loss reduction. In Theorem 5.1, we show how PASS measures the



v_3
v_5
v_2
Node v_2's neighborhood
$-\frac{d\mathcal{L}}{dh_2^{(l)}}$
The direction in which $h_2^{(l)}$ should move to decrease the performance loss $\mathcal{L}$
$h_3^{(l)}$
$h_2^{(l)} + h_3^{(l)}$
The embedding of v_2 after aggregating with v_3
$h_2^{(l)}$
The current embedding of v_2
$h_2^{(l)} + h_5^{(l)}$
The embedding of v_2 after aggregating with v_5
$h_5^{(l)}$

Figure 3: Interpretation of why PASS assigns higher sampling probability to node v_3 than v_5 given source node v_2: node v_3's embedding $h_3^{(l)}$ helps v_2's embedding $h_2^{(l)}$ move in the direction $-d\mathcal{L}/dh_2^{(l)}$ that decreases the performance loss $\mathcal{L}$ while aggregating the embedding of node v_5 would move v_2 in the opposite direction.

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

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

$$\begin{aligned}\frac{dz_i^{(l)}}{dq^{(l)}(j|i)} &= \frac{d}{dq^{(l)}(j|i)}\mathbb{E}_{k\sim q^{(l)}(k|i)}[h_k^{(l)}]\\ &= \frac{d}{dq^{(l)}(j|i)}\sum_{k=0}^{N(i)} q^{(l)}(u_k|i)h_{u_k}^{(l)}\\ &= \frac{d}{dq^{(l)}(j|i)}(q^{(l)}(j|i)h_j^{(l)}) = h_j^{(l)}\end{aligned}$$

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

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

Since $d\mathcal{L}/dz_i^{(l)} = d\mathcal{L}/dh_i^{(l)}$, $-d\mathcal{L}/dq^{(l)}(j|i)$ 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, $-d\mathcal{L}/dq^{(l)}(j|i)$ becomes large and the probability $q^{(l)}(j|i)$ 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.

Table 3: Dataset statistics: LinkedIn dataset on member networks has two labels, member industry and job title.

Dataset	Nodes	Edges	Features	Labels
Cora	2,485	5,069	1,433	7
Citeseer	2,110	3,668	3,703	6
Pubmed	19,717	44,324	500	3
AmazonC	13,381	245,778	767	10
AmazonP	7,487	119,043	745	8
MsCS	18,333	81,894	6,805	15
MsPhysics	34,493	247,962	8,415	5
LinkedIn	39K+	1.7M+	20+	(industry) ~150 (title) ~8,000

5.1 Design of Sampling Policy

In Equation 9, $-d\mathcal{L}/dq^{(l)}(j|i)$ 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 $a(v_i, v_j) = a\cdot[\mathbf{W}\cdot h_i^{(l)}||\mathbf{W}\cdot h_j^{(l)}]$ 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: GraghSage [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 4: Effectiveness test: PASS outperforms all baselines up to 10.4% on the benchmark datasets and up to 10.2% on our production datasets (LnkIndustry, LnkTitle). Results on the benchmark datasets are presented in precision. Results on our production datasets are presented in percentage point (pp) with respect to GraphSage (random sampling). A higher precision/percentage point is better.

Method	Cora	Citeseer	Pubmed	AmazonC	AmazonP	MsCS	MsPhysics	LnkIndustry	LnkTitle
FastGCN	0.582	0.496	0.569	0.480	0.542	0.520	0.638	-4.2pp	-2.0pp
AS-GCN	0.462	0.387	0.502	0.419	0.480	0.403	0.516	-7.1pp	-0.6pp
GraphSage	0.788	0.698	0.792	0.707	0.787	0.766	0.875	0.0pp	0.0pp
GCN-BS	0.788	0.693	0.809	0.736	0.800	0.780	0.887	1.8pp	0.7pp
PASS	**0.821**	**0.715**	**0.858**	**0.757**	**0.855**	**0.884**	**0.934**	**10.2pp**	**1.3pp**

Table 5: Robustness test: PASS maintains high accuracy in various graph noise scenarios, while the accuracy of all other baselines plummets with noise. PASS is effective not only in sampling informative neighbors but also in removing irrelevant neighbors.

	Fake connections among existing nodes						Fake neighbors with random feature vectors					
Method	Cora	Citeseer	Pubmed	AmazonC	AmazonP	MsCS	Cora	Citeseer	Pubmed	AmazonC	AmazonP	MsCS
FastGCN	0.293	0.254	0.416	0.300	0.307	0.292	0.597	0.513	0.614	0.502	0.566	0.563
AS-GCN	0.229	0.171	0.334	0.206	0.167	0.176	0.233	0.152	0.379	0.271	0.169	0.252
GraphSage	0.312	0.261	0.439	0.376	0.306	0.262	0.282	0.269	0.459	0.264	0.264	0.248
GCN-BS	0.320	0.265	0.457	0.387	0.305	0.264	0.571	0.493	0.681	0.639	0.686	0.622
PASS	**0.658**	**0.603**	**0.811**	**0.669**	**0.698**	**0.822**	**0.722**	**0.681**	**0.761**	**0.672**	**0.783**	**0.667**

(GraghSage, GCN-BS, and PASS) sample one neighbor per node, thus sampling 64 nodes per layer in total. You can find the results with larger numbers of samples in Appendix A.4.

Evaluation with Sampling. Previous works [4, 10, 14] sample during training but not during testing: they compute node embeddings on the test set by aggregating full neighborhoods. This setting is unrealistic: the prohibitive time and memory costs from the full neighborhood expansion that prompted us to sample during training are also issues during testing. **In this work, we sample both during training and testing.** This results in a significant drop in accuracy for certain baselines, especially layer-wise samplers.

6.2 Effectiveness

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 $\{v_k\}_{k=i_1}^{i_n}$ as $q(j|i_1, \cdots, i_n)$. 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.

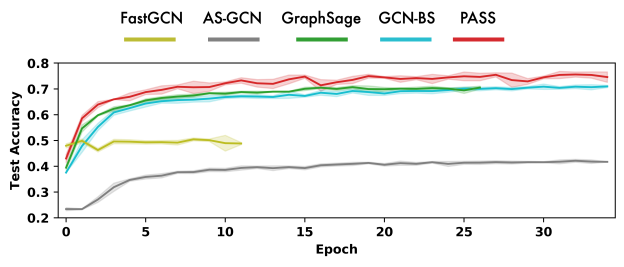
6.3 Robustness

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.

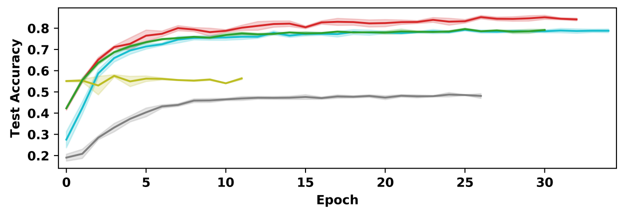
Table 5 shows that PASS consistently has high accuracy across all scenarios, while the performance of all other methods plummets. The sparse connection problems of layer-wise sampling methods (FastGCN, AS-GCN) become worse with graph noise. Node-wise sampling methods also show much lower accuracy than 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 time.

	Accuracy			Training time (s)			Test time (s)		
Dataset	GATs	PASS (1)	PASS (5)	GATs	PASS (1)	PASS (5)	GATs	PASS (1)	PASS (5)
Cora	**0.850**	0.821	**0.847**	189.670	9.459	7.226	0.122	0.022	0.033
Citeseer	**0.744**	0.715	0.735	404.904	13.962	13.225	0.175	0.043	0.069
Pubmed	-	0.858	**0.871**	-	87.660	94.918	-	0.612	1.510
AmazonC	-	0.757	**0.886**	-	52.060	184.522	-	0.256	1.218
AmazonP	0.905	0.855	**0.944**	1869.690	30.060	68.134	0.709	0.094	0.338
MS CS	-	0.884	**0.918**	-	101.840	142.099	-	0.811	3.113
MS Physics	-	0.934	**0.952**	-	439.378	507.816	-	4.162	8.445



(a) Amazon Computer



(b) Amazon Photo

Figure 4: The convergence on test set in terms of epochs.

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.

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

Dataset	GAT-version	q_{imp}	$q_{imp} + q_{rand}$
Cora	0.574	0.779	0.821
Citeseer	0.456	0.706	0.715
Pubmed	0.606	0.862	0.858
AmazonC	0.482	0.746	0.757
AmazonP	0.575	0.854	0.855
MsCS	0.661	0.883	0.884
MsPhysics	0.683	0.933	0.934

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

$$q^{(l)}_{GAT}(j|i) = a \cdot [\mathbf{W} \cdot h^{(l)}_i || \mathbf{W} \cdot h^{(l)}_j]$$

where $\mathbf{W}$ 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.

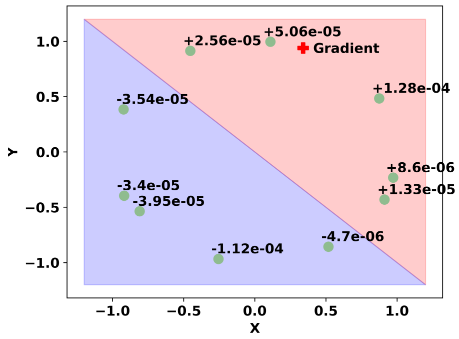


Figure 5: The hidden-layer embeddings of a neighborhood in the Amazon Computer dataset (visualized by t-SNE [3]). The red cross denotes the gradient of the loss w.r.t the source node and green points denote the embeddings of neighbor nodes. Numbers denote the increase/decrease in sampling probabilities. PASS increases sampling probabilities for neighbors in the red area, close to the gradient, while decreasing probabilities for the neighbors in the blue zone, which are far from the gradients.

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 $h_j^{(l)}$ and the gradient $-d\mathcal{L}/dh_i^{(l)}$ 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 edge imputation policy and combining it with our proposed edge sampling policy to improve the overall performance of graph neural networks.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[3] Andreas Buja, Dianne Cook, and Deborah F Swayne. 1996. Interactive high-dimensional data visualization. *Journal of computational and graphical statistics* 5, 1 (1996), 78–99.

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

[5] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning.*

[6] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. 1999. On power-law relationships of the internet topology. *ACM SIGCOMM computer communication review* 29, 4 (1999), 251–262.

[7] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. 2017. Protein interface prediction using graph convolutional networks. In *Advances in neural information processing systems.* 6530–6539.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems.*

[9] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems.* 2042–2050.

[10] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. In *Advances in neural information processing systems.* 4558–4567.

[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems.* 1097–1105.

[14] Ziqi Liu, Zhengwei Wu, Zhiqiang Zhang, Jun Zhou, Shuang Yang, Le Song, and Yuan Qi. 2020. Bandit Samplers for Training Graph Neural Networks. *arXiv preprint arXiv:2006.05806* (2020).

[15] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. 2019. Monte carlo gradient estimation in machine learning. *arXiv preprint arXiv:1906.10652* (2019).

[16] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference.* Springer, 593–607.

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

[18] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[19] Baoxu Shi, Jaewon Yang, Tim Weninger, Jing How, and Qi He. 2019. Representation Learning in Heterogeneous Professional Social Networks with Ambiguous Social Connections. In *IEEE BigData.*

[20] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

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

[22] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[23] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. In *Advances in Neural Information Processing Systems.* 11249–11259.

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 $\mathcal{L}(h) \in \mathbb{R}$, the gradient of the loss $\mathcal{L}$ with respect to θ is presented as follows:*

$$\frac{d\mathcal{L}}{d\theta} = \frac{d}{d\theta}\left(\frac{d\mathcal{L}}{dh}\cdot h\right)$$

with an assumption $\frac{d}{d\theta}\left(\frac{d\mathcal{L}}{dh}\right) = 0$.

Proof. By the chain rule, $\frac{d\mathcal{L}}{d\theta}$ is decomposed as follows:

$$\frac{d\mathcal{L}}{d\theta} = \frac{d\mathcal{L}}{dh}\cdot\frac{dh}{d\theta}$$

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

$$\frac{d\mathcal{L}}{d\theta_i} = \sum_{j=0}^{D^{(l)}} \frac{d\mathcal{L}}{dh_j}\cdot\frac{dh_j}{d\theta_i}$$

The dot product of $\frac{d\mathcal{L}}{dh}$ and h is presented as follows:

$$\frac{d\mathcal{L}}{dh}\cdot h = \frac{d\mathcal{L}}{dh_0}h_0 + \frac{d\mathcal{L}}{dh_1}h_1 + \cdots + \frac{d\mathcal{L}}{dh_{(D^{(l)}-1)}}h_{(D^{(l)}-1)}$$

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

$$\begin{aligned}\frac{d}{d\theta_i}\left(\frac{d\mathcal{L}}{dh}\cdot h\right) &= \frac{d\mathcal{L}}{dh_0}\frac{dh_0}{d\theta_i} + \frac{d\mathcal{L}}{dh_1}\frac{dh_1}{d\theta_i} + \cdots + \frac{d\mathcal{L}}{dh_{(D^{(l)}-1)}}\frac{dh_{(D^{(l)}-1)}}{d\theta_i} \\ &= \sum_{j=0}^{D^{(l)}} \frac{d\mathcal{L}}{dh_j}\frac{dh_j}{d\theta_i} \\ &= \frac{d\mathcal{L}}{d\theta_i}\end{aligned}$$

Then $\frac{d\mathcal{L}}{d\theta_i} = \frac{d}{d\theta_i}\left(\frac{d\mathcal{L}}{dh}\cdot h\right)$ for every $0 \le i < D^{(s)}$. This shows $\frac{d\mathcal{L}}{d\theta}$ is equal to $\frac{d}{d\theta}\left(\frac{d\mathcal{L}}{dh}\cdot h\right)$. ■

With the SUB-LOSS trick, we compute an auxiliary loss $\mathcal{L}_{aux} = d\mathcal{L}/dh_i^{(l)}\cdot h_i^{(l)}$ and simply call the backpropagation function of our deep learning framework on $\mathcal{L}_{aux}$ to compute the gradient w.r.t. θ.

A.2 Experimental Setting

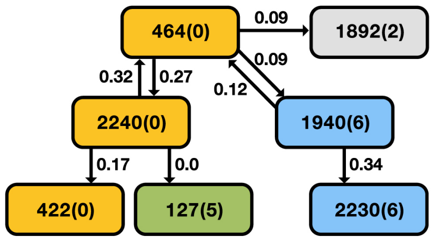
Hyper-parameters. We use the Adam optimizer [11] and tune each baseline with a grid search on each dataset. Most baselines perform best on most datasets with a learning rate of 0.01, weight decay of 5×10^{-4}. We report the average performance across 5 runs for each experiment.

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

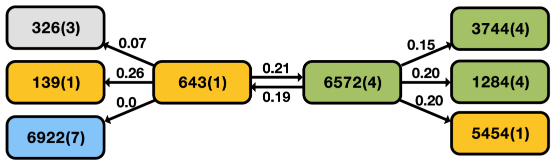
- **FastGCN:** https://github.com/matenure/FastGCN
- **AS-GCN:** https://github.com/huangwb/AS-GCN
- **GraphSage:** https://github.com/williamleif/GraphSAGE
- **GCN-BS:** https://github.com/xavierzw/ogb-geniepath-bs
- **GAT:** https://github.com/PetarV-/GAT

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) Nodes and subset of neighbors from the Cora dataset



(b) Nodes and subset of neighbors from the Amazon Photo dataset

Figure 6: PASS learns which neighbors are informative or not. The numbers in nodes denote node ids and labels. The numbers in edges denote sampling probabilities computed by PASS.

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.

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

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.