On the Importance of Sampling in Learning Graph Convolutional Networks: Convergence Analysis and Variance Reduction

Weilin Cong* Mehrdad Mahdavi†

Abstract

Graph Convolutional Networks (GCNs) have achieved impressive empirical success across a wide variety of graph-related applications. Despite their great success, training GCNs on large graphs suffers from computational and memory issues. A potential path to circumvent these obstacles is sampling-based methods, where at each layer a subset of nodes are sampled. Although recent studies have empirically demonstrated the effectiveness of sampling-based methods, however, these methods face a memory barrier, as large mini-batches are required, and lack theoretical convergence guarantees under realistic settings. In this paper, we describe and analyze a general **doubly variance reduction** schema that can accelerate any sampling method under the memory budget. The motivating impetus for the proposed schema is a careful analysis of variance of sampling methods where it is shown that the induced variance can be decomposed into *function approximation variance* (zeroth-order variance) and *layer-gradient variance* (first-order variance) during forward and backward propagation, respectively. We theoretically analyze the convergence of proposed schema and show that it enjoys an $\mathcal{O}(1/T)$ convergence rate. We complement our theoretical results by integrating the proposed schema in different sampling methods and applying to different large-scale real-world graphs.

1 Introduction

In the past few years, graph convolutional networks (GCNs) have achieved great success in many graph-related applications, such as semi-supervised node classification [1], supervised graph classification [2], protein interface prediction [3], and knowledge graph [4]. However, most works on GCNs focused on relatively small graphs, scaling GCNs for large-scale graph is non-trivial, even by conducting mini-batch training. Due to the dependency of the nodes in the graph, we still need to consider a large-receptive field to calculate the representation of each node in the mini-batch, and the receptive field grows exponentially with respect to the number of layers. To alleviate this issue, sampling-based methods, such as node-wise sampling [5–7], layer-wise sampling [8–10], and subgraph sampling [11, 12] are proposed for mini-batch GCN training.

Although empirical results show that sampling-based methods can scale GCN training to large graphs, these methods suffer from a few key issues. First, the theoretical understanding of sampling-based methods is still lacking. Second, the sampling strategies are only based on the structure of the graph via either uniform or importance sampling based strategies over layers or nodes, and thereby do not adapt to the information of the evolving parameters during optimization (representation of nodes or gradient information). As a result, these methods usually introduce high variance in training which significantly degrades the convergence rate and leads to a poor generalization (test) error.

To reduce the variance of mini-batches, a simple solution is to utilize adaptive importance sampling strategies to constantly re-evaluate the relative importance of nodes during training (e.g., current gradient or representation of nodes). The main drawback of this method is that it is computationally inadmissible to compute the optimal adaptive sampling distribution as it requires the computation of the full gradient in

*Department of Computer Science and Engineering, The Pennsylvania State University. Email: wxc272@psu.edu.
†Department of Computer Science and Engineering, The Pennsylvania State University. Email: mzm616@psu.edu.

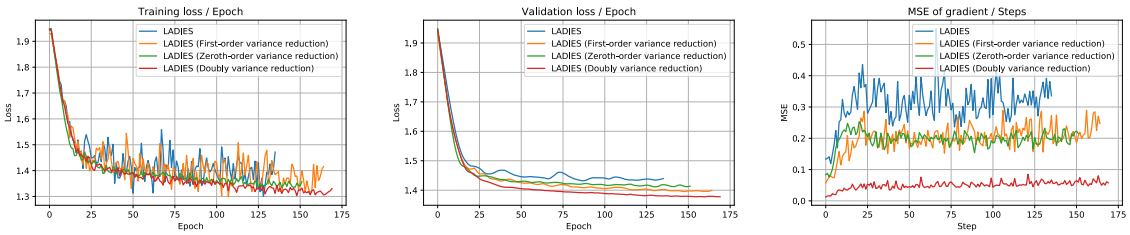

Figure 1: The effect of doubly variance reduction on training loss, validation loss, and mean-square error (MSE) of gradient on `Flickr` dataset using layer-wise sampling method `LADIES` proposed in [10].

each iteration. This necessitates developing alternative solutions that can efficiently be computed and that come with theoretical guarantees.

In this paper, we develop a novel doubly variance reduction schema that can be applied to any sampling strategy to significantly reduce its induced variance and enjoys the beneficial properties of adaptive importance sampling methods. The main motivation behind the proposed schema stems from our theoretical analysis of the variance of sampling methods in training GCNs. Specifically, we show that due to the composite structure of training objective, any sampling strategy introduces two types of variance in estimating the stochastic gradients: function approximation variance (zeroth-order variance) which results from estimating representations or embeddings and layer-gradient variance (first-order variance) which results from gradient estimation. In Figure 1, we exhibit the performance of proposed schema when utilized in the sampling strategy introduced in [10]. The plots show that applying our proposal can lead to a significant reduction in variance; hence faster convergence rate and better test accuracy. We can also see that both zeroth-order and first-order methods are equally important and demonstrate significant improvement when applied jointly (i.e, doubly variance reduction).

Contributions. We summarize the contributions as follows:

- We provide the theoretical analysis of sampling-based mini-batch methods for training GCNs and provide a non-asymptotic convergence rate under a weaker assumption about stochastic gradients. We note that the analysis in [13] is asymptotic and achieves an $\mathcal{O}(1/\sqrt{T})$ rate under the assumption that stochastic gradients are biased but consistent, and the analysis in [7] achieves the same rate under the strong assumption that stochastic gradients are unbiased, where both assumption are not practical due to the composite structure of training objective as will be elaborated.

- We propose a generic and efficient doubly variance reduction schema, dubbed as `SGCN++`, that can be integrated with different sampling-based methods to significantly reduce the function approximation variance and stochastic gradient variance, resulting in a faster convergence rate and better generalization.

- We theoretically analyze the convergence of `SGCN++` algorithm and obtain an $\mathcal{O}(1/T)$ rate, which significantly improves the best known bound $\mathcal{O}(1/\sqrt{T})$. We note that our analysis is the first obtaining such a rate for training GCNs via sampling and is optimal.

- We empirically verify `SGCN++` through various experimental results on different large-scale real graph datasets and different sampling methods (such as `GraphSage`, `FastGCN`, and `LADIES`), where `SGCN++` demonstrate significant improvements compared to the original sampling methods. We remark that `SGCN++` can be specialized to different sampling methods, without changing the graph convolutional structure.

Organization. The remainder of this paper is organized as follows. Section 2 discusses more related works. In Section 3, we present a novel analysis of variance and convergence of sampling based methods. In Section 4, we propose a doubly variance reduction algorithm and theoretically analyze its convergence rate. Finally, we

empirically verify our theory via experiments in Section 5 and conclude the paper in Section 6. To avoid breaking the flow of our exposition, all proofs are relegated to Appendix A, B, and C. The details for different algorithms are presented in Appendix D. The additional experiments are discussed in Appendix E and F, and instructions on reproducing the results in Appendix G.

2 Additional Related Work

Training GCNs via sampling. The full-batch training of a typical GCN is employed in [1] which necessities to keep the whole graph data and intermediate nodes' representations in the memory. This is the key bottleneck that hinders the scalability of full-batch training. To overcome the issues of full-batch training, sampling-based GCN training methods are proposed to train GCNs based on mini-batch of nodes, and only aggregate the embeddings of a sampled subset of neighbors of nodes in the mini-batch. For example, the node-wise sampling method `GraphSage` [5] restricts the computation complexity by uniformly sampling a fixed number of neighbors from the previous layer nodes. However, the significant computational overhead is introduced when GCN goes deep. `VRGCN` [7] further reduces the neighborhood size and uses history activation of the previous layer to reduce variance. However, they require to perform a full-batch graph convolutional operation on history activation during each forward propagation, which is computation expensive. Another direction uses a layer-wise method, and apply importance-sampling to reduce variance. For example, `FastGCN` [8] independently sample a constant number of nodes in all layers using importance sampling. However, the sampled nodes are too sparse to achieve high accuracy. `LADIES` [10] further restrict the candidate nodes in the union of the neighborhoods of the sampled nodes in the upper layer. However, significant overhead may be incurred due to the expensive sampling algorithm. Another direction uses a subgraph sampling method. For example, `GraphSaint` [12] construct mini-batches by importance sampling, and apply normalization techniques to eliminate bias and reduce variance. However, the sampled subgraphs are usually sparse, which requires a large sampling size to guarantee the performance.

Theoretical analysis. Despite many algorithmic progresses over the years, the theoretical understanding of the convergence of sampling-based method for training GCNs is still limited. [7] provide a convergence analysis on `VRGCN` under the assumption that the stochastic gradient due to sampling is unbiased and achieved a convergence rate of $\mathcal{O}(1/\sqrt{T})$. However, the convergence analysis is specified to `VRGCN`, and the assumption is not true in practice due to the composite structure of GCN. [13] provides another convergence analysis for `FastGCN` under a strong assumption that the stochastic gradient of GCN converges to the consistent gradient exponentially fast with respect to the sample size, and result in the same convergence rate as does use unbiased ones, i.e., $\mathcal{O}(1/\sqrt{T})$. Most recently, [14] provide PAC learning-style bounds on the node embedding and gradient estimation for sampling-based GCN training. However, their analysis is restricted to neighbor sampling algorithms with uniform-sampling [5] and binary classification, whether the analysis can be extended to layer-wise non-uniform sampling or multi-class classification is still an open problem. Another direction of theoretical research focuses on analyzing the expressive power of GCN [2, 15–17], which is not the focus of this paper and we omit for brevity.

Variance reduced non-convex optimization. For non-convex optimization, the vanilla SGD has shown to yield a stochastic first-order oracle complexity of $\mathcal{O}(\epsilon^{-2})$ to attain a first-order stationary point that satisfies $\mathbb{E}[\|\nabla f(\boldsymbol{\theta})\|^2] \leq \epsilon$. Furthermore, various variance reduction methods have been proposed to reduce the variance of the gradient estimator in SGD, such as SVRG [18], SAGA [19], and SPIDER [20]. In particular, SVRG and SAGA have shown to yield an overall stochastic first-order oracle (SFO) complexity of $\mathcal{O}(n^{2/3}\epsilon^{-1})$. More recently, SPIDER [20] propose to recursively update the stochastic gradient in the inner loop, and achieve an overall $\min\{\mathcal{O}(n^{1/2}\epsilon^{-1}), \mathcal{O}(\epsilon^{-3/2})\}$. For two-level non-convex composite optimization problem, [21] propose to employ two-level stochastic updates in different step size, and achieves sample complexity of $\mathcal{O}(\epsilon^{-7/4})$ for smooth non-convex case. [22] extend SAGA to two-level composite optimization setting, and achieves sample complexity of $\mathcal{O}(n + n^{2/3}\epsilon^{-1})$. For a multi-level non-convex composite optimization problem, [23] adopts a similar approach as [21] and achieves the sample complexity of $\mathcal{O}(\epsilon^{-(L+7)/4})$ where L is the total

level of the optimization problem. [22] extends SPIDER to multi-level composite optimization and achieves the sample complexity of $\mathcal{O}(n+\sqrt{n}\epsilon^{-1})$ but adopts a learning rate of $\mathcal{O}(\epsilon/L)$ to guarantee the convergence, which prevents it from making the big process even if it is possible. Besides, the convergence analysis of [22] requires a very small per-iteration increment $\|\theta_k-\theta_{k-1}\|=\mathcal{O}(\epsilon/L)$, which is difficult to guarantee when applying it to deep neural networks, due to the fast movement of θ_k in the optimization landscape.

3 Sampling-based GCNs

In this section, we present a novel analysis of variance and convergence of sampling-based methods for stochastically training GNNs. Indeed, we show theoretically that due to the structure of optimization problem in training GCNs, the induced variance can be decomposed into function approximation variance and layer gradient variance, two key factors that affect the convergence. We then rigorously analyze the convergence rate of stochastic mini-batch algorithms for training GCNs.

Problem formulation. In this paper, we consider training GCNs under the supervised learning setting. Given an undirected graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ with $N=|\mathcal{V}|$ nodes, each node is associated with an F dimensional feature vector and $\mathbf{X}\in\mathbb{R}^{N\times F}$ denotes the feature matrix for all N nodes. The corresponding adjacency matrix $\mathbf{A}\in\mathbb{R}^{N\times N}$ is a sparse matrix with $A_{i,j}=1$ if there exist an edge between node i and node j, and $A_{i,j}=0$ otherwise. let $\mathbf{L}=\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ be the normalized Laplacian matrix, where $\mathbf{D}$ is the degree matrix. The ℓth layer of GCN is defined as

$$\mathbf{Z}^{(\ell)}=\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)},\ \mathbf{H}^{(\ell)}=\sigma(\mathbf{Z}^{(\ell)}),$$

where $\sigma(\cdot)$ is the activation function, $\mathbf{H}^{(\ell)}\in\mathbb{R}^{N\times F_\ell}$ is the embedding matrix for all N nodes at layer ℓ, and $\mathbf{W}^{(\ell)}\in\mathbb{R}^{F_\ell\times F_{\ell+1}}$ is the weight matrix at the ℓth GCN layer. The weight matrices are updated by minimizing $\mathcal{L}(\boldsymbol{\theta})=\frac{1}{N}\sum_{i\in\mathcal{V}}\phi(\boldsymbol{z}_i^{(L)},y_i)$, where $\boldsymbol{\theta}=\{\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(L)}\}$ is the stacked parameters that specifies the entire model, $\phi(\cdot,\cdot)$ is the specified loss function, $\boldsymbol{z}_i^{(L)}$ denotes the ith row of node embedding matrix $\mathbf{Z}^{(L)}$ with the ground truth label y_i. The gradient is computed over all the nodes by $\frac{1}{N}\sum_{i\in\mathcal{V}}\nabla\phi(\boldsymbol{z}_i^{(L)},y_i)$.

When the graph is large, the computational complexity of forward and backward propagation could be very high. One practical solution to alleviate this issue is to sample a mini-batch $\mathcal{V}_\mathcal{B}$ of size $B=|\mathcal{V}_\mathcal{B}|$ of nodes and a sparser normalized Laplacian matrix $\mathbf{L}^{(\ell)}$ from $\mathbf{L}$ for the ℓth layer, then perform forward and backward propagation only based on the sampled Laplacian matrix. We refer to sampling based methods as `SGCN`. More specifically, the ℓth layer of `SGCN` is defined as

$$\mathbf{Z}^{(\ell)}=\mathbf{L}^{(\ell)}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)},\ \mathbf{H}^{(\ell)}=\sigma(\mathbf{Z}^{(\ell)}),$$

and the gradient is computed over all the nodes in the mini-batch by $\frac{1}{B}\sum_{i\in\mathcal{V}_\mathcal{B}}\nabla\phi(\boldsymbol{z}_i^{(L)},y_i)$.

Optimization problem. For the ease of presentation, we write the above L-layer `SGCN` as a multi-level composite stochastic optimization problems of the form

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} f(\boldsymbol{\theta})=\mathbb{E}_{\xi_L}\Big[f_{\xi_L}^{(L)}\Big(\mathbb{E}_{\xi_{L-1}}\big[f_{\xi_{L-1}}^{(L-1)}\big(\ldots\mathbb{E}_{\xi_1}[f_{\xi_1}^{(1)}(\boldsymbol{\theta})]\ldots\big)\big]\Big)\Big],$$

where $\boldsymbol{\theta}=\{\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(L)}\}$, ξ_ℓ are the set of nodes `SGCN` sampled at the ℓth layer. For example, $f^{(\ell)}(\boldsymbol{\theta})=\sigma(\mathbf{L}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)})$ and $f_{\xi_\ell}^{(\ell)}(\boldsymbol{\theta})=\sigma(\mathbf{L}^{(\ell)}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)})$. The algorithm for `SGCN` is detailed in Algorithm 1. We denote the deterministic function by

$$F^{(\ell)}(\boldsymbol{\theta}_t)=f^{(\ell)}\circ f^{(\ell-1)}\circ\ldots\circ f^{(1)}(\boldsymbol{\theta}_t),\ \ell\in[L],$$

where $[L]$ denote the set of numbers $\{1,\ldots,L\}$ for brevity. Applying the chain rule, the full gradient at step t of stochastic gradient descent (SGD) is calculated as

$$\nabla F(\boldsymbol{\theta}_t)=\nabla f^{(1)}(\boldsymbol{\theta}_t)\prod_{\ell=2}^{L}\nabla f^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t)).$$

Algorithm 1 SGCN

Input: initial weight $\boldsymbol{\theta}_1 = \{\mathbf{W}_1^{(\ell)}\}_{\ell=1}^L$, step size $\eta > 0$, layer-wise sample sizes $\{S_\ell\}_{\ell=1}^L$.
for $t = 1, \ldots, T$ **do**
  Set $\mathbf{H}_t^{(0)}$ as the node feature matrix $\mathbf{X}$
  for $\ell = 1, \ldots, L$ **do**
    Randomly sample batch $\mathcal{B}_\ell$ of size $S_\ell = |\mathcal{B}_\ell|$
    $\mathbf{H}_t^{(\ell)} = f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_t^{(\ell-1)}; \mathbf{W}_t^{(\ell)})$
    $\mathbf{D}_t^{(\ell)} = \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_t^{(\ell-1)}; \mathbf{W}_t^{(\ell)})$
  end for
  Calculate gradient $\mathbf{g}_t = \mathbf{D}_t^{(1)} \mathbf{D}_t^{(2)} \ldots \mathbf{D}_t^{(L)}$
  Update weights $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{g}_t$
end for
Output: $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}_k$, where $k = \arg\min_t \mathbb{E}[\nabla f(\boldsymbol{\theta}_t)]$

To approximate the above gradient, we need to construct estimators for function value approximation,

$$\mathbf{H}_t^{(1)} \approx F^{(1)}(\boldsymbol{\theta}_t), \mathbf{H}_t^{(\ell-1)} \approx F^{(\ell-1)}(\boldsymbol{\theta}_t),$$

and estimators for layer gradient approximation,

$$\mathbf{D}_t^{(1)} \approx \nabla f^{(1)}(\boldsymbol{\theta}_t), \mathbf{D}_t^{(\ell)} \approx \nabla f^{(\ell)}(\mathbf{H}_t^{(\ell-1)}),$$

Then, the stochastic gradient is calculated by

$$\mathbf{g}_t = \mathbf{D}_t^{(1)} \mathbf{D}_t^{(2)} \ldots \mathbf{D}_t^{(L)},$$

which can be used to update the GCN parameters.

Variance analysis. For theoretical analysis, we make the following assumptions on the functions $f^{(1)}(\cdot), \ldots, f^{(L)}(\cdot)$.

Assumption 1. *For the function $f^{(\ell)}(\cdot)$ we assume (a) For each $\ell \in [L]$ and each realization of ξ_ℓ, the mapping $f_{\xi_\ell}^{(\ell)}$ is ρ_ℓ-Lipschitz and its gradient $\nabla f_{\xi_\ell}^{(\ell)}$ is L_ℓ-Lipschitz; (b) For each $\ell \in [L-1]$, there exists δ_ℓ such that $\mathbb{E}_{\xi_\ell}\left[\|f_{\xi_\ell}^{(\ell)}(y) - f^{(\ell)}(y)\|^2\right] \leq \delta_\ell^2$; (c) For each $\ell \in [L]$, there exists σ_ℓ such that $\mathbb{E}_{\xi_\ell}\left[\|\nabla f_{\xi_\ell}^{(\ell)}(y) - \nabla f^{(\ell)}(y)\|^2\right] \leq \sigma_\ell^2$.*

As a result of this assumption, we know the gradient of function $f(\boldsymbol{\theta})$ are L_f-Lipschitz continuous, and the Lipschitz constants are given in the following lemma.

Lemma 1. *Under Assumption 1, the gradient of composite function $\nabla f(\cdot)$ is Lipschitz continuous with constant*

$$L_f = \sum_{\ell=1}^{L} L_i \left(\prod_{i=1}^{\ell-1} \rho_i^2\right) \left(\prod_{i=\ell+1}^{L} \rho_i\right).$$

Key challenges. The key challenge in theoretically or empirically analyzing SGCN is its multi-level composite structure which leads to two types of variance, i.e., function approximation variance and layer gradient variance. We highlight this by analyzing the mean-square error (MSE) of the stochastic gradient of SGCN in the following lemma.

Lemma 2. *Suppose Assumption 1 hold. The mean-square error of stochastic gradient $\mathbb{E}[\|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2]$ can be bounded by function approximation variance $\mathbb{V}^{(\ell)} := \mathbb{E}[\|f_{\xi_\ell}^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t)) - F^{(\ell)}(\boldsymbol{\theta}_t)\|^2]$ and layer gradient*

variance $\mathbb{G}^{(\ell)} := \mathbb{E}[\|\mathbf{D}_t^{(\ell)} - \nabla f^{(\ell)}(\mathbf{H}_t^{(\ell-1)})\|^2]$ *as follows:*

$$\mathbb{E}[\|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2] \leq 2L \sum_{\ell=1}^{L} \left(\prod_{i=1}^{\ell-1} \rho_i^2\right) \left(\prod_{i=\ell+1}^{L} \rho_i^2\right) \cdot \mathbb{G}_\ell + 2L \sum_{\ell=2}^{L} \left(\prod_{i=1}^{\ell-1} \rho_i^2\right) \left(\prod_{i=\ell+1}^{L} \rho_i^2\right) L_\ell^2 \cdot \ell \sum_{i=1}^{\ell} \left(\prod_{j=i+1}^{\ell} \rho_j^2 \cdot \mathbb{V}_j\right)$$

Note that the function approximation variance and layer gradient variance exist because a sparser normalized Laplacian matrix $\mathbf{L}^{(\ell)}$ is sampled from the full normalized Laplacian matrix $\mathbf{L}$ to calculate the approximation embedding matrix $\mathbf{H}^{(\ell)}$ at each layer. The key difference between them is that the function approximation variance happens during forward propagation, while the layer gradient variance happens during backward propagation.

Convergence analysis. The following theorem establishes the convergence rate of `SGCN` in Algorithm 1.

Theorem 1. *Suppose Assumption 1 hold and apply* ***SGCN*** *in Algorithm 1 with learning rate chosen as*

$$\eta = \min\left\{\frac{1}{L_f}, \sqrt{\frac{\mathbb{E}[f(\boldsymbol{\theta}_1)] - \mathbb{E}[f(\boldsymbol{\theta}_*)]}{\Delta L_f T}}\right\},$$

where L_f *denotes the Lipschitz constant of gradient and* Δ *denotes the upper-bound of MSE of stochastic gradient*

$$\Delta = 2L \sum_{\ell=1}^{L} \left(\prod_{i=1}^{\ell-1} \rho_i^2\right) \left(\prod_{i=\ell+1}^{L} \rho_i^2\right) \cdot \mathbb{G}_\ell + 2L \sum_{\ell=2}^{L} \left(\prod_{i=1}^{\ell-1} \rho_i^2\right) \left(\prod_{i=\ell+1}^{L} \rho_i^2\right) L_\ell^2 \cdot \ell \sum_{i=1}^{\ell} \left(\prod_{j=i+1}^{\ell} \rho_j^2 \cdot \mathbb{V}_j\right)$$

Then, the corresponding output $\tilde{\boldsymbol{\theta}}$ *satisfy* $\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2] \leq \epsilon$ *provided that the total number* T *of iterations satisfies*

$$T \geq \mathcal{O}(\Delta/\epsilon^2)$$

Proof. Proof can be found in Appendix A. □

Remark 1. *Suppose there are* N *nodes in the graph* $\mathcal{G}$*. Let* $|\mathcal{B}_L| = b$ *and* $|\mathcal{B}_\ell| = N$ *for all* $\ell \in [L-1]$*. Then,* ***SGCN*** *is the same as performing SGD with mini-batch size* b*, which has a convergence rate of* $\mathcal{O}(1/\sqrt{T})$ *that is independent of the number of layers* L*.*

Remark 2. *Existing* ***SGCN*** *algorithms, such as* ***GraphSage*** *[5],* ***FastGCN*** *[8],* ***LADIES*** *[10], and* ***VRGCN*** *[7], propose to reduce the function approximation variance* $\mathbb{V}^{(\ell)}$ *by increasing the neighbors size, applying importance sampling, and using history activations, respectively. Therefore, these algorithms suffer the same amount of layer gradient variance* $\mathbb{G}^{(\ell)}$*, which can be only reduced by increasing the mini-batch size of* ***SGCN****.*

We highlight the difference of function approximation variance $\mathbb{V}^{(\ell)}$ of different sampling based methods in Table 1, and defer the detailed derivation to Appendix B. Note that one of the main factors that affect the function approximation variance is the number of neighbors used to estimate node embeddings. Therefore, clearly `GraphSage` should have the lowest function approximation variance since the graph is guaranteed to be densely connected, and the variance of `GraphSage` is in the order of $\mathcal{O}(D/s)$, where D is the average degree, s is the number of neighbors sampled. Let N_ℓ as the number of nodes sampled at the ℓth layer in layer-wise sampling. `FastGCN` perform layer-wise sampling by sampling a subset of nodes independently for each GCN layer using importance sampling, the variance of `FastGCN` is in the order of $\mathcal{O}(N^2/N_\ell^2)$, which is highly dependent on the ratio of the number of sampled nodes to the total number of points. `LADIES` restricts

Table 1: Summary of function approximation variance $\mathbb{V}_\ell$. Denote ++ as applying doubly variance reduction to the original algorithm. Here D denote the average degree of nodes, s denote the size of sampled neighbors, N_ℓ denote the size of sampled nodes in ℓth layer, γ_ℓ denote the upper-bound of $\|\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)}\|_2$ and $\bar{\gamma}_\ell$ denote the upper-bound of $\|\mathbf{H}^{(\ell-1)}(\mathbf{W}_k^{(\ell)} - \mathbf{W}_{k-1}^{(\ell)})\|_2$ for any $k = 2, \ldots, K$, where K is the maximum number of inner-loop steps. We use $\mathcal{O}(\cdot)$ hide constants that remain the same between different algorithms.

Method	GraphSage	FastGCN	LADIES	GraphSage++	FastGCN++	LADIES++
$\mathbb{V}^{(\ell)}$	$\mathcal{O}(D\gamma_\ell^2/s)$	$\mathcal{O}(N^2\gamma_\ell^2/N_\ell^2)$	$\mathcal{O}(N\gamma_\ell^2/N_\ell)$	$\mathcal{O}(KD\bar{\gamma}_\ell^2/s)$	$\mathcal{O}(KN^2\bar{\gamma}_\ell^2/N_\ell^2)$	$\mathcal{O}(KN\bar{\gamma}_\ell^2/N_\ell)$

Algorithm 2 SGCN++

Input: initial point $\bar{\boldsymbol{\theta}}_1 = \{\bar{\mathbf{W}}^{(\ell)}\}_{\ell=1}^L$, step size $\eta > 0$, control factors $\{\omega_\ell\}_{\ell=1}^L$, layer-wise sample sizes $\{S_\ell, S'_\ell\}_{\ell=1}^L$, maximum inner-loop steps K.
for $t = 1, \ldots, T$ **do**
  Set $\boldsymbol{\theta}_t \leftarrow \bar{\boldsymbol{\theta}}_t$
  for $\ell = 1, \ldots, L$ **do**
    Randomly sample batch $\mathcal{B}_\ell$ of nodes of size S_ℓ.
    Run Eq. 1 and Eq. 3
  end for
  Calculate gradient $\nabla f(\boldsymbol{\theta}_1) = \mathbf{D}_1^{(1)}\mathbf{D}_1^{(2)}\ldots\mathbf{D}_1^{(L)}$, then update weight $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_1 - \eta\nabla f(\boldsymbol{\theta}_1)$
  for $k = 2, \ldots, K$ **do**
    for $\ell = 1, \ldots, L$ **do**
      Randomly sample batch $\mathcal{B}_\ell$ of nodes of size S'_ℓ.
      Run Eq. 2 and Eq. 4
      if Condition in Eq. 5 is true **then**
        Break and start another outer-loop
      end if
    end for
  end for
  Calculate gradient $\mathbf{g}(\boldsymbol{\theta}_k) = \mathbf{D}_k^{(1)}\mathbf{D}_k^{(2)}\ldots\mathbf{D}_k^{(L)}$, then update weight $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\mathbf{g}(\boldsymbol{\theta}_k)$
  Set $\bar{\boldsymbol{\theta}}_{t+1} = \boldsymbol{\theta}_{K+1}$
end for
Output: $\tilde{\boldsymbol{\theta}}$ is randomly chosen from all solutions

the candidate nodes in the union of the neighborhoods of the sampled nodes in the upper layer. This can reduce the dependence on the ratio of the number of sampled nodes to the total number of points to some extent, and reduce the variance to the order of $\mathcal{O}(N/N_\ell)$. We leave the discussion on $\mathbb{V}^{(\ell)}$ for doubly variance reduction to the next section.

4 Doubly Variance Reduction

Our proposed doubly variance reduction algorithm stems from the observation which is presented in the previous section. We note that the analysis in this section applies to different graph neural network variants, including graph convolutional networks (GCNs), graph attention networks (GCNs), gated graph neural networks (GGNNs). But for the ease of presentation, we focus on GCNs.

From the convergence analysis in Theorem 1, we observe that there are two types of variances: the function approximation variance $\mathbb{V}^{(\ell)}$ and the layer gradient variance $\mathbb{G}^{(\ell)}$. This motivate us to propose a doubly variance reduction algorithm, SGCN++, that reduces $\mathbb{V}^{(\ell)}$ by zeroth-order variance reduction and $\mathbb{G}^{(\ell)}$ by first-order variance reduction. We detailed SGCN++ in Algorithm 2.

The main idea of our variance reduction algorithm is based on the property of L^2 martingales, stated in

Lemma 3, which is recently used in [20] for stochastic variance reduced optimization.

Lemma 3. *Suppose there exist an consecutive mini-batch estimator generate sequence* $\{\mathbf{H}_1, \ldots, \mathbf{H}_k\}$ *using a recursion:*

$$\mathbf{H}_1 = \frac{1}{B_1} \sum_{i \in \mathcal{B}_1} f_i(\mathbf{W}_1)$$

$$\mathbf{H}_k = \mathbf{H}_{k-1} + \frac{1}{B_k} \sum_{i \in \mathcal{B}_k} (f_i(\mathbf{W}_k) - f_i(\mathbf{W}_{k-1})), \ k \geq 2$$

then the MSE of the estimator can be bounded as

$$\mathbb{E}[\|\mathbf{H}_k - f(\mathbf{W}_k)\|^2] \leq \mathbb{E}[\|\mathbf{H}_1 - f(\mathbf{W}_1)\|^2] + \sum_{k=2}^{T} \frac{1}{B_k} \mathbb{E}[\|f(\mathbf{W}_k) - f(\mathbf{W}_{k-1})\|^2]$$

Zeroth-order variance reduction. At the beginning of each inner-loop, we run a full-batch GCN

$$\mathbf{H}_1^{(\ell)} = f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_1^{(\ell-1)}; \mathbf{W}_1^{(\ell)}), \tag{1}$$

and save the node embeddings of all nodes $\mathbf{H}_1^{(\ell)}$ into the memory. Then at the kth step, we use $\mathbf{H}_{k-1}^{(\ell)} - f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_{k-1}^{(\ell-1)}; \mathbf{W}_{k-1}^{(\ell)})$ for zeroth-order variance reduction, and calculated the function value estimation of $\mathbf{H}_k^{(\ell)}$ by

$$\mathbf{H}_k^{(\ell)} = \mathbf{H}_{k-1}^{(\ell)} + f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_k^{(\ell-1)}; \mathbf{W}_k^{(\ell)}) - f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_{k-1}^{(\ell-1)}; \mathbf{W}_{k-1}^{(\ell)}) \tag{2}$$

Notice that if we increase the size of mini-batch $\mathcal{B}_\ell$ to the full-batch, we have $\mathbf{H}_k^{(\ell)} = f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_k^{(\ell-1)}; \mathbf{W}_k^{(\ell)})$. In addition, we keep track of the ℓ_2 norm of the difference between $\mathbf{H}_1^{(\ell)}$ and $\mathbf{H}_k^{(\ell)}$, and start another outer-loop if the difference is too large.

We compare the function approximation variance $\mathbb{V}^{(\ell)}$ of `SGCN` with `SGCN++` in Table 1, and we defer the detailed calculation to Appendix B. Taking advantage of the properties of L^2 martingale sequence, we can reduce the function approximation variance of `SGCN++` to $K\bar{\gamma}_\ell^2/\gamma_\ell^2$ times the function approximation variance of `SGCN`, where K is the maximum number of inner-loop steps, γ_ℓ is the upper-bound of $\|\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)}\|_2$ and $\bar{\gamma}_\ell$ is the upper-bound of $\|\mathbf{H}^{(\ell-1)}(\mathbf{W}_k^{(\ell)} - \mathbf{W}_{k-1}^{(\ell)})\|_2$ for $k = 2, \ldots, K$. Notice that, as $t \to \infty$ in Algorithm 4, we have $\bar{\gamma}_\ell \to 0$. Therefore, as long as we choose K such that $K < \gamma_\ell^2/\bar{\gamma}_\ell^2$, we can guarantee a zeroth-order variance reduction.

First-order variance reduction. Similar to the zeroth-order variance reduction, we run a full-batch GCN at the beginning of each inner-loop

$$\mathbf{D}_1^{(\ell)} = \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_1^{(\ell-1)}; \mathbf{W}_1^{(\ell)}), \tag{3}$$

and save the full-batch gradient $\mathbf{D}_1^{(\ell)}$ into the memory. Then at the kth step, we use the weight from $(k-1)$th step to calculate the stochastic gradient $\nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_k^{(\ell-1)}; \mathbf{W}_k^{(\ell)})$ for first-order variance reduction, and calculate an unbiased layer gradient estimator of $\mathbf{D}_k^{(\ell)}$ by

$$\mathbf{D}_k^{(\ell)} = \mathbf{D}_{k-1}^{(\ell)} + \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_k^{(\ell-1)}; \mathbf{W}_k^{(\ell)}) - \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_{k-1}^{(\ell-1)}; \mathbf{W}_{k-1}^{(\ell)}) \tag{4}$$

Besides, we keep track of the ℓ_2 norm of the difference between $\mathbf{D}_1^{(\ell)}$ and $\mathbf{D}_k^{(\ell)}$, and start another outer-loop if the difference is too large.

Key challenges. The key challenges of applying stochastic variance reduction optimization problem to the deep neural network are due to the staleness. In particular, the variance of stochastic variance reduction optimizer is directly depends on how similar the function value (gradient) is between the snapshot point $\boldsymbol{\theta}_1$ and current iterate $\boldsymbol{\theta}_k$. If the $\boldsymbol{\theta}_k$ evolves too quickly through the optimization landscape, the snapshot point will be too out-of-date to provide meaningful variance reduction. In addition, the function value (gradient) in the larger model changes more rapidly. To alleviate this issue, we monitor the staleness by keep tracking the ℓ_2 norm of the distance between $\mathbf{H}_k^{(\ell)}$ and $\mathbf{H}_1^{(\ell)}$, between $\mathbf{D}_k^{(\ell)}$ and $\mathbf{D}_1^{(\ell)}$, and start another outer-loop if the condition in Eq 5 holds:

$$\|\mathbf{H}_k^{(\ell)} - \mathbf{H}_1^{(\ell)}\| \geq \omega_\ell \|\mathbf{H}_1^{(\ell)}\|, \|\mathbf{D}_k^{(\ell)} - \mathbf{D}_1^{(\ell)}\| \geq \omega_\ell \|\mathbf{D}_1^{(\ell)}\| \tag{5}$$

In addition, due to the multi-level composition structure, two types of stochastic variance, i.e., function approximation variance $\mathbb{V}^{(\ell)}$ and layer gradient variance $\mathbb{G}^{(\ell)}$, makes directly applying first-order only variance reduction, e.g., SVRG, SAGA, and SARAH, not practical. We empirically show the ineffectiveness of applying zeroth-order only and first-order only variance reduction in Section 5.

Convergence results. We have the following general theorem for `SGCN++` described in Algorithm 2. The analysis is based on the assumptions in Assumption 1.

Theorem 2. *Suppose there are N nodes in the graph. Let Assumption 1 hold and apply Algorithm 2 with parameters $S = N, S' = \sqrt{N}, K = L_f^2\sqrt{N}/\phi$ and step-size $\eta = \frac{1}{2L_f}$, where ϕ is a constant defined in Lemma 19. Then, the corresponding output $\tilde{\boldsymbol{\theta}}$ satisfies $\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2] \leq \epsilon$ provided that the total number TK of iterations satisfies*

$$TK \geq \mathcal{O}\Big(\frac{L_f(f(\boldsymbol{\theta}_1) - f(\boldsymbol{\theta}_*))}{\epsilon}\Big).$$

and the resulting SFO complexity is $\mathcal{O}(\sqrt{n}\epsilon^{-1} + n)$.

Proof. Proof can be found in Appendix C. □

Remark 3. *The algorithm proposed in Algorithm 2 shares the same spirit as the multi-level variance reduction algorithm proposed in [24] (Algorithm 3), and we enjoy the same convergence rate as [24] Theorem 5.4. Here we highlight the difference between Algorithm 2 and [24]. Recall that the main challenge of variance reduction algorithms is staleness control [25]. [24] control the staleness by using a learning rate of $\eta = \epsilon/L_f$, which is proportional to the expected error of ϵ, to make sure the history gradient is not too staled. However, using a tiny learning rate is not practical for the non-convex learning problem, and will prevent the algorithm from making big process even if it is possible. Instead, we use the idea of early stopping in Algorithm 2 by monitoring the staleness of history gradient, and start another outer-loop if the condition is triggered. By doing so, we can choose a large learning rate, and even use Adam [26] as the base optimizer for weight update, which enjoys a significant speedup on convergence.*

Remark 4. *Since both Algorithm 2 and VRGCN [7] use zeroth-order variance reduction, it is worth contrasting these methods. At each iteration in VRGCN, a giant Laplacian matrix $\mathbf{L}$ is applied on to the history activations $\mathbf{H}_{k-1}^{(\ell-1)}$ to calculate $\mathbf{L}\mathbf{H}_{k-1}^{(\ell-1)}\mathbf{W}_k^{(\ell)}$ as a variance reduction estimator*

$$\mathbf{H}_k^{(\ell)} = \sigma((\mathbf{L}^{(\ell)}(\mathbf{H}_{k-1}^{(\ell-1)} - \mathbf{H}_{k-1}^{(\ell-1)}) + \mathbf{L}\mathbf{H}_{k-1}^{(\ell-1)})\mathbf{W}_k^{(\ell)})$$

This process introduces a significant computational complexity. In addition, since different Laplacian matrix $\mathbf{L}$ and $\mathbf{L}^{(\ell)}$ are applied before the activation function, VRGCN requires a change of graph convolutional structure, which makes it hard to generalize on different `SGCN`s. Instead, we applies the same Laplacian matrix $\mathbf{L}^{(\ell)}$ on $\mathbf{H}_k^{(\ell)}$ and $\mathbf{H}_{k-1}^{(\ell)}$,

$$\mathbf{H}_k^{(\ell)} = \mathbf{H}_{k-1}^{(\ell)} + \sigma(\mathbf{L}^{(\ell)}\mathbf{H}_k^{(\ell-1)}\mathbf{W}_k^{(\ell)}) - \sigma(\mathbf{L}^{(\ell)}\mathbf{H}_{k-1}^{(\ell-1)}\mathbf{W}_{k-1}^{(\ell)}),$$

which is more computational efficient compared against VRGCN and easier to extend to different `SGCN` variants.

Table 2: Summary of dataset statistics. **m** stands for **m**ulti-class classification, and **s** stands for **s**ingle-class.

Dataset	Nodes	Edges	Degree	Feature	Classes	Train/Val/Test
Pubmed	19,717	44,338	3	500	3(**s**)	92%/ 3%/ 5%
Flickr	89,250	899,756	10	500	7 (**s**)	50%/25%/25%
Reddit	232,965	11,606,919	50	602	41(**s**)	66%/10%/24%
PPI	14,755	225,270	15	50	121(**m**)	66%/12%/22%

5 Experiments

In this section, we conduct experiments to examine the variance and convergence of `SGCN` and `SGCN++` on large-scale node classification datasets.

Table 3: Comparison of the accuracy (micro F1-score) and training time (seconds) of `SGCN` variants (including `GraphSage`, `FastGCN`, `LADIES`, and `Exact`) and their zeroth-order, first-order, and doubly variance reduction (denoted by `++`) on GCN training.

Method		Pubmed		Flickr		Reddit		PPI	
Exact	SGCN	0.867	35s	0.516	165s	0.948	2,365s	0.519	69s
	SGCN++ (1-st)	**0.875**	27s	**0.530**	95s	**0.951**	1,369s	**0.653**	182s
GraphSage	SGCN	0.859	32s	0.514	60s	0.943	1,037s	0.470	50s
	SGCN++ (0-th)	0.861	60s	0.514	92s	0.949	1,264s	0.508	69s
	SGCN++ (1-st)	0.862	27s	**0.520**	173s	0.899	671s	0.578	82s
	SGCN++ (doubly)	**0.862**	41s	0.516	135s	**0.949**	961s	**0.589**	99s
LADIES	SGCN	0.870	29s	0.499	164s	0.941	2,937s	0.454	23s
	SGCN++ (0-th)	0.878	54s	0.505	192s	0.940	924s	0.525	40s
	SGCN++ (1-st)	**0.880**	24s	0.515	66s	0.941	661s	0.506	41s
	SGCN++ (doubly)	0.876	34s	**0.521**	147s	**0.949**	1,165s	**0.545**	65s
FastGCN	SGCN	0.706	62s	0.423	233s	0.206	2,459s	0.453	31s
	SGCN++ (0-th)	**0.874**	54s	0.489	143s	**0.928**	1,594s	0.522	82s
	SGCN++ (1-st)	0.873	29s	0.483	56s	0.899	695s	0.494	37s
	SGCN++ (doubly)	0.873	73s	**0.501**	269s	0.898	943s	**0.526**	49s

5.1 Experiment setup

Experiments are under supervised learning setting. We evaluate on the following real-world datasets. (a) **Pubmed**: classifying academic papers based on citations; (b) **Flickr**: classifying types of images based on descriptions and common properties of online images; (c) **Reddit**: classifying communities of online posts based on user comments; (d) **PPI**: classifying protein functions based on the interactions of human tissue proteins. Dataset statistic details are summarized in Table 2.

To make the fair comparison only on the sampling and variance reduction part, we modified the PyTorch implementation of GCN [1] to add `LADIES` [10], `FastGCN` [8], `GraphSage` [5] and `Exact` sampling mechanism. Then, we implement `SGCN++` on the top of each sampling method to illustrate how doubly variance reduction help for GCN training. Other than doubly variance reduction, we introduce zeroth-order only and first-order only variance reduction as baselines. Details can be found in Appendix D. By default, we train 2-layer GCNs with hidden state dimension as 256. We choose 5 neighbors to be sampled for `GraphSage` sampling, 512 nodes to be sampled for `FastGCN` and `LADIES` sampling. For `SGCN++`, we choose inner-loop steps $K = 10$ and early stop parameter $\omega = 2e - 3$. Historical node embeddings in `SGCN++` are stored in CPU memory by default, however, it is also possible to store them on an external hard drive if facing a memory budget. We update the model with a batch side of 512 and Adam optimizer with a learning rate of 0.01. Note that `Exact`

sampling is a variant of `GraphSage` with zero *function approximation variance*, which is designed to explicitly illustrate the effectiveness of first-order variance reduction.

For all the methods and datasets, we conduct training for 3 times and take the average of the evaluation results. Each time we run 200 epochs, all methods terminate when the validation loss does not decrease a threshold (0.01) for 200 iterations. We choose the model with the lowest validation error as the convergence point. We use the following metrics to evaluate the effectiveness of sampling methods: (a) **F1-score**: The micro F1-score of the test data at the convergence point calculated by full-batch; (b) **Running time**: The total training time for before convergence point; (c) **Loss**: The loss curve during training and testing phase; (d) **Mean-square error of gradient**: The mean-square error of stochastic gradient during training.

5.2 Experiment Results

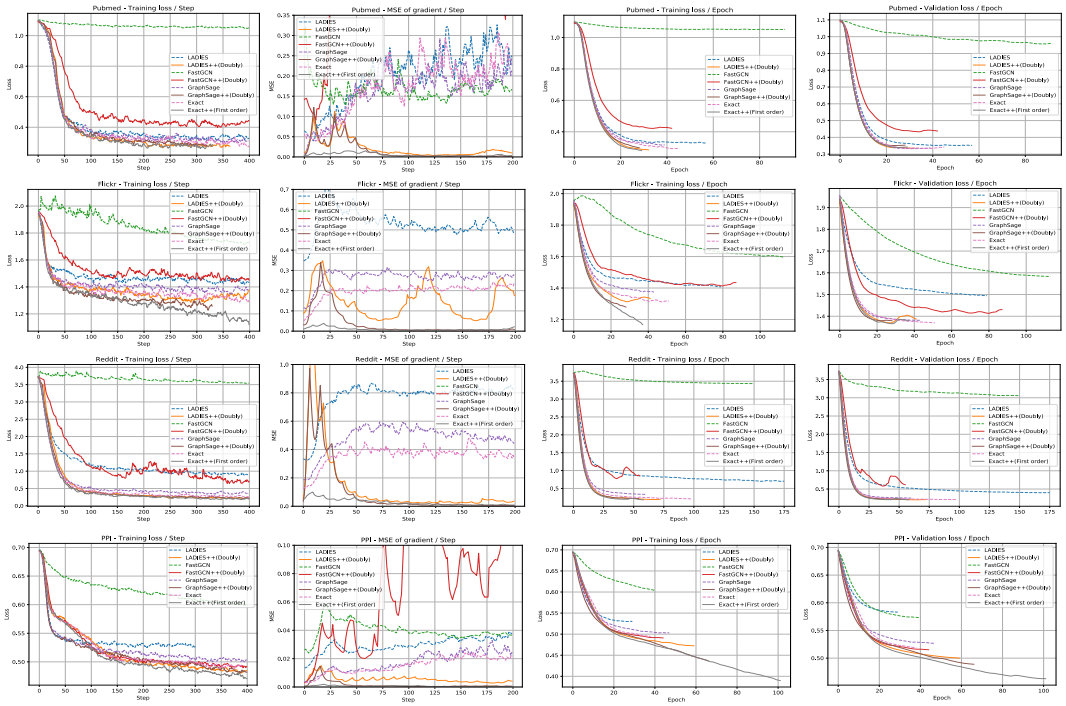

Figure 2: Comparison of training loss, validation loss, and mean-square error (MSE) of gradient of `SGCN`s (including `GraphSage`, `FastGCN`, `LADIES`, and `Exact` sampling) and doubly variance reduction training method `SGCN++` on `Pubmed`, `Flickr`, `Reddit`, and `PPI` dataset. Results show that `SGCN++` can significantly reduce the MSE of gradient and accelerate the convergence.

Comparison of variance of different `SGCN`s. We empirically evaluate the function approximation variance and layer gradient variance of different `SGCN`s by comparing the mean-square error (MSE) of the stochastic gradient. Recall that the MSE of the stochastic gradient is a weighted sum of function approximation variance and layer gradient variance. Besides, all `SGCN` variants suffer the same amount of layer gradient variance. As shown in Figure 2, `Exact` has the lowest MSE because it has zero function approximation variance. Due to layer-wise independence sampling, the sampled computation graph of `FastGCN` is sparse which results in the highest variance. `LADIES` restrict to sample from the union of the neighborhoods of the sampled nodes in the upper layer and `GraphSage` restrict a fixed number of neighbors to be sampled. `LADIES` can be seen as an importance sampling version of `GraphSage` with fewer neighbors sampled. Empirically, importance

sampling can select many neighbors for one node and no neighbor for another, in which case, LADIES has a higher function approximation variance than GraphSage. The results match the conclusion of variance we derived in Table 1.

Relation between variance and convergence. We empirically demonstrate the relationship between variance and convergence. As shown in Figure 2, the method with a smaller variance enjoys a faster convergence during training, which matches the conclusion of Theorem 1. Moreover, a lower generalization error is obtained when variance is low.

Effectiveness of variance reduction. Figure 2 shows the comparison of training loss, testing loss, and mean-square error of the stochastic gradient. Clearly, SGCN++ achieves significantly lower mean-square error of stochastic gradient and faster convergence during training. Table 3 shows the comparison the accuracy of SGCNs and their zeroth-order, first-order, and doubly variance reduction variants. The total training time before the convergence point is also provided. Clearly, variance reduction can significantly improve the accuracy and reduce the training time. For example, when training on Reddit dataset, the time required by doubly variance reduction SGCN++ is only 39% of the time required by SGCN. The results matches the conclusion of Theorem 2, which states that the training with SGCN++ converge to a local optimum faster than SGCN.

Importance of doubly variance reduction. Due to the multi-level composite structure, as we have shown in Section 3, there are two types of variance that slow down the convergence of SGCN, i.e., function approximation variance and layer gradient variance. To emphasize the importance of applying doubly variance reduction comparing to zeroth-order only and first-order only variance reduction, we compare the training loss, testing loss, and mean-square error of stochastic gradient of applying zeroth-order only, first-order only, and doubly variance reduction. Due to the space limit, figures can be found in Appendix F. We observe that doubly variance reduction can always guarantee a significant performance improvement in terms of convergence speed and accuracy, while zeroth-order only variance reduction if effective only when SGCNs suffer a large function approximation variance, e.g., FastGCN, and first-order only variance might worsen the performance when the function approximation variance is large, e.g., FastGCN.

6 Conclusion

In this work, we develop a theoretical framework for analyzing the convergence of sampling based mini-batch GCNs training (such as GraphSage, FastGCN, and LADIES) and show that function approximation variance in forward propogation and layer gradient variance in backward propagation are the two key factors that slow down the convergence of these methods and necessities larger mini-batch sizes. To overcome this issue, we propose a novel doubly variance reduction method and theoretically analyzed its convergence. Experimental results on benchmark datasets demonstrate the effectiveness of proposed schema to reduce the variance to achieve better generalization.

References

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

[2] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[3] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *NIPS*, 2017.

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

[5] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

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

[7] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

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

[9] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Advances in Neural Information Processing Systems*, pages 4558–4567, 2018.

[10] Yewen Wang Song Jiang Yizhou Sun Quanquan Gu Difan Zou, Ziniu Hu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *openreview*, 2019.

[11] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. *arXiv preprint arXiv:1905.07953*, 2019.

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

[13] Jie Chen and Ronny Luss. Stochastic gradient descent with biased but consistent gradient estimators. *arXiv preprint arXiv:1807.11880*, 2018.

[14] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Constant time graph neural networks, 2020.

[15] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[16] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

[17] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in {gnn}s. In *International Conference on Learning Representations*, 2020.

[18] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczos, and Alex Smola. Stochastic variance reduction for nonconvex optimization. In *International conference on machine learning*, pages 314–323, 2016.

[19] Zeyuan Allen-Zhu and Elad Hazan. Variance reduction for faster non-convex optimization. In *International conference on machine learning*, pages 699–707, 2016.

[20] Cong Fang, Chris Junchi Li, Zhouchen Lin, and Tong Zhang. Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. In *Advances in Neural Information Processing Systems*, pages 689–699, 2018.

[21] Mengdi Wang, Ethan X Fang, and Han Liu. Stochastic compositional gradient descent: algorithms for minimizing compositions of expected-value functions. *Mathematical Programming*, 161(1-2):419–449, 2017.

[22] Junyu Zhang and Lin Xiao. A composite randomized incremental gradient method. In *International Conference on Machine Learning*, pages 7454–7462, 2019.

[23] Shuoguang Yang, Mengdi Wang, and Ethan X Fang. Multilevel stochastic gradient methods for nested composition optimization. *SIAM Journal on Optimization*, 29(1):616–659, 2019.

[24] Junyu Zhang and Lin Xiao. Multi-level composite stochastic optimization via nested variance reduction. *arXiv preprint arXiv:1908.11468*, 2019.

[25] Aaron Defazio and Léon Bottou. On the ineffectiveness of variance reduced optimization for deep learning. In *Advances in Neural Information Processing Systems*, pages 1753–1763, 2019.

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

Organization The appendix is organized as follows: In Section A, we provide the proof of Theorem 1. In Section B, we provide the variance analysis of sampling-based GCN training methods. In Section C, we provide the proof of Theorem 2. In Section D, we provide detailed description of different variants of the proposed algorithm used in experiments (i.e., zero-order only, first-order only, and doubly variance reduction). In Section E and F, we provide detailed information on the experiment setup and detailed discussion of additional experiments that omitted from main text due to lack of space. For reproducible research, in Section G, we provide detailed instructions on reproducing the experimental results reported in this paper.

A Proof of Theorem 1

The proof of Theorem 1 is fairly straightforward and follows from by now standard stochastic non-convex optimization, with the key difference that unlike vanilla SGD where the variance is usually assumed to be fixed, in GCNs, due to nested structure of objective function, the variance varies from sampling strategy to another that needs to be carefully investigated as we theoretically analyze in Section B.

The proof of Theorem 1 is based on the following lemmas.

The following lemma gives the Lipschitz continuous constant of the gradient of composite function $f(\cdot)$, where the Lipschitz continuous constant plays an important role in choosing learning rate and the final convergence rate.

Lemma 4. *Under Assumption 1, the gradient of composite function $\nabla f(\cdot)$ is Lipschitz continuous with constant*

$$L_f = \sum_{\ell=1}^{L} L_i \left(\prod_{i=1}^{\ell-1} \rho_i^2\right) \left(\prod_{i=\ell+1}^{L} \rho_i\right).$$

The following lemma shows that the mean-square error of the stochastic gradient of sampling-based GCN training methods is a linear combination of two types of stochastic variance: function approximation variance and layer gradient variance.

Lemma 5. *Suppose Assumption 1 hold. The mean-square error of stochastic gradient $\mathbb{E}[\|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2]$ can be bounded by function approximation variance $\mathbb{V}^{(\ell)} := \mathbb{E}[\|f_{\xi_\ell}^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t)) - F^{(\ell)}(\boldsymbol{\theta}_t)\|^2]$ and layer gradient variance $\mathbb{G}^{(\ell)} := \mathbb{E}[\|\mathbf{D}_t^{(\ell)} - \nabla f^{(\ell)}(\mathbf{H}_t^{(\ell-1)})\|^2]$ as follows:*

$$\begin{aligned}\mathbb{E}[\|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2] \le{}& 2L \sum_{\ell=1}^{L} \left(\prod_{i=1}^{\ell-1} \rho_i^2\right) \left(\prod_{i=\ell+1}^{L} \rho_i^2\right) \cdot \mathbb{G}_\ell \\ &+ 2L \sum_{\ell=2}^{L} \left(\prod_{i=1}^{\ell-1} \rho_i^2\right) \left(\prod_{i=\ell+1}^{L} \rho_i^2\right) L_\ell^2 \cdot \ell \sum_{i=1}^{\ell} \left(\prod_{j=i+1}^{\ell} \rho_j^2 \cdot \mathbb{V}_j\right)\end{aligned}$$

The following lemma provides an upper-bound on the function approximation variance.

Lemma 6. *Suppose Assumption 1 hold and at each step t let $\mathcal{B}_\ell$ be a subset that samples $S_\ell = |\mathcal{B}_\ell|$ elements in $[n]$ with replacement, and let the stochastic estimator $f_{\mathcal{B}_\ell}^{(\ell)}(\cdot) = \frac{1}{S_\ell} \sum_{\ell \in \mathcal{B}_\ell} f_\ell^{(\ell)}(\cdot)$. Then we can bound the function approximation variance $\mathbb{E}[\|f_{\xi_\ell}^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t)) - F^{(\ell)}(\boldsymbol{\theta}_t)\|^2]$ as*

$$\mathbb{E}[\|f_{\xi_\ell}^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t)) - F^{(\ell)}(\boldsymbol{\theta}_t)\|^2] \le \frac{\delta_\ell^2}{S_\ell}$$

The following lemma provides an upper-bound on the layer gradient variance.

Lemma 7. *Suppose Assumption 1 hold and at each step t let $\mathcal{B}_\ell$ be a subset that samples $S_\ell = |\mathcal{B}_\ell|$ elements in $[n]$ with replacement, and let the stochastic estimator $\nabla f_{\mathcal{B}_\ell}^{(\ell)}(\cdot) = \frac{1}{S_\ell} \sum_{\ell \in \mathcal{B}_\ell} \nabla f_\ell^{(\ell)}(\cdot)$. Then we can bound the layer gradient variance $\mathbb{E}[\|\mathbf{D}_t^{(\ell)} - \nabla f_t^{(\ell)}(\mathbf{H}_t^{(\ell-1)})\|^2]$ as*

$$\mathbb{E}[\|\mathbf{D}_t^{(\ell)} - \nabla f_t^{(\ell)}(\mathbf{H}_t^{(\ell-1)})\|^2] \le \frac{\sigma_\ell^2}{S_\ell}$$

Equipped with above results, we now turn to providing the convergence of sampling based SGD for training GCNs. As the theorem demonstrates, the mean-square error of stochastic gradient is the key factor in convergence and leads to slow convergence rate when the variance is high.

Lemma 8. *Suppose non-convex function $f(\boldsymbol{\theta})$ has L_f-Lipschitz continuous gradient and its stochastic gradient variance is bounded by $\mathbb{E}[\|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2] \leq \Delta$ where $\Delta > 0$ is constant that depends on the sampling strategy. Set step size $\eta = \min\left\{\frac{1}{L_f}, \sqrt{\frac{\mathbb{E}[f(\boldsymbol{\theta}_1)] - \mathbb{E}[f(\boldsymbol{\theta}_*)]}{\Delta L_f T}}\right\}$ and let $\tilde{\boldsymbol{\theta}} = \min_t \mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t)\|]$, then we have*

$$\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2] \leq \mathcal{O}(\sqrt{\Delta/T}).$$

Theorem 3 (General Version of Theorem 1)**.** *Suppose Assumption 1 holds and apply* `SGCN` *in Algorithm 1 with parameters*

$$\eta = \min\left\{\frac{1}{L_f}, \sqrt{\frac{\mathbb{E}[f(\boldsymbol{\theta}_1)] - \mathbb{E}[f(\boldsymbol{\theta}_*)]}{\Delta L_f T}}\right\}$$

where Δ and L_f are defined as

$$\Delta = 2L\sum_{\ell=1}^{L}\left(\prod_{i=1}^{\ell-1}\rho_i^2\right)\left(\prod_{i=\ell+1}^{L}\rho_i^2\right)\cdot\frac{\sigma_\ell^2}{S_\ell} + 2L\sum_{\ell=2}^{L}\left(\prod_{i=1}^{\ell-1}\rho_i^2\right)\left(\prod_{i=\ell+1}^{L}\rho_i^2\right)L_\ell^2\cdot\ell\sum_{i=1}^{\ell}\left(\prod_{j=i+1}^{\ell}\rho_j^2\frac{\delta_j^2}{S_j}\right)$$

and

$$L_f = \sum_{\ell=1}^{L} L_i\left(\prod_{i=1}^{\ell-1}\rho_i^2\right)\left(\prod_{i=\ell+1}^{L}\rho_i\right)$$

Then, the corresponding output $\tilde{\boldsymbol{\theta}}$ satisfies $\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2] \leq \epsilon$ provided that the total number T of iterations satisfies

$$T \geq \mathcal{O}(\Delta/\epsilon^2)$$

Proof of Theorem 3. Combining Lemmas 5, 6, and 7 yields

$$\begin{aligned}
&\mathbb{E}[\|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2] \\
&\leq 2L\sum_{\ell=1}^{L}\left(\prod_{i=1}^{\ell-1}\rho_i^2\right)\left(\prod_{i=\ell+1}^{L}\rho_i^2\right)\cdot\frac{\sigma_\ell^2}{S_\ell} + 2L\sum_{\ell=2}^{L}\left(\prod_{i=1}^{\ell-1}\rho_i^2\right)\left(\prod_{i=\ell+1}^{L}\rho_i^2\right)L_\ell^2\cdot\ell\sum_{i=1}^{\ell}\left(\prod_{j=i+1}^{\ell}\rho_j^2\frac{\delta_j^2}{S_j}\right)
\end{aligned}$$

By Lemma 4, we have

$$L_f = \sum_{\ell=1}^{L} L_i\left(\prod_{i=1}^{\ell-1}\rho_i^2\right)\left(\prod_{i=\ell+1}^{L}\rho_i\right)$$

From Lemma 8, we know that by using step size $\eta = \min\left\{\frac{1}{L_f}, \sqrt{\frac{\mathbb{E}[f(\boldsymbol{\theta}_1)] - \mathbb{E}[f(\boldsymbol{\theta}_*)]}{\Delta L_f T}}\right\}$ we have

$$\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2] \leq \mathcal{O}(\sqrt{\Delta/T})$$

□

A.1 Proof of Lemma 4

Proof. Let $F^{(\ell)}(\cdot) = f^{(\ell)} \circ \ldots \circ f^{(1)}(\cdot)$ denote the deterministic function value at the ℓth layer. Under the Assumption 1, for any $\mathbf{X}$ and $\mathbf{Y}$ we have

$$\begin{aligned}
\|F^{(\ell)}(\mathbf{X}) - F^{(\ell)}(\mathbf{Y})\| &\leq \|f^{(\ell)} \circ \ldots \circ f^{(1)}(\mathbf{X}) - f^{(\ell)} \circ \ldots \circ f^{(1)}(\mathbf{Y})\| \\
&\leq \rho_\ell\|f^{(\ell-1)} \circ \ldots \circ f^{(1)}(\mathbf{X}) - f^{(\ell-1)} \circ \ldots \circ f^{(1)}(\mathbf{Y})\| \\
&\leq \prod_{i=1}^{\ell}\rho_i\|\mathbf{X} - \mathbf{Y}\|.
\end{aligned}$$

Therefore $f(\cdot) = f^{(L)} \circ \ldots \circ f^{(1)}(\cdot)$ is ρ_f-Lipschitz continuous with $\rho_f = \prod_{\ell=1}^{L} \rho_\ell$. For its gradient $\nabla f(\cdot) = \nabla f^{(L)}(F^{(L-1)}(\cdot)) \ldots \nabla f^{(1)}(\cdot)$ we have

$$
\begin{aligned}
&\|\nabla f(\mathbf{X}) - \nabla f(\mathbf{Y})\| \\
&= \|\nabla f^{(L)}(F^{(L-1)}(\mathbf{X}))\nabla f^{(L-1)}(F^{(L-2)}(\mathbf{X})) \ldots \nabla f^{(1)}(\mathbf{X}) - \nabla f^{(L)}(F^{(L-1)}(\mathbf{Y}))\nabla f^{(L-1)}(F^{(L-2)}(\mathbf{Y})) \ldots \nabla f^{(1)}(\mathbf{Y})\| \\
&\leq \|\nabla f^{(L)}(F^{(L-1)}(\mathbf{X}))\nabla f^{(L-1)}(F^{(L-2)}(\mathbf{X})) \ldots \nabla f^{(1)}(\mathbf{X}) - \nabla f^{(L)}(F^{(L-1)}(\mathbf{X}))\nabla f^{(L-1)}(F^{(L-2)}(\mathbf{X})) \ldots \nabla f^{(1)}(\mathbf{Y})\| \\
&+ \|\nabla f^{(L)}(F^{(L-1)}(\mathbf{X}))\nabla f^{(L-1)}(F^{(L-2)}(\mathbf{X})) \ldots \nabla f^{(1)}(\mathbf{Y}) - \nabla f^{(L)}(F^{(L-1)}(\mathbf{X}))\nabla f^{(L-1)}(F^{(L-2)}(\mathbf{Y})) \ldots \nabla f^{(1)}(\mathbf{Y})\| \\
&+ \ldots \\
&+ \|\nabla f^{(L)}(F^{(L-1)}(\mathbf{X}))\nabla f^{(L-1)}(F^{(L-2)}(\mathbf{Y})) \ldots \nabla f^{(1)}(\mathbf{Y}) - \nabla f^{(L)}(F^{(L-1)}(\mathbf{Y}))\nabla f^{(L-1)}(F^{(L-2)}(\mathbf{Y})) \ldots \nabla f^{(1)}(\mathbf{Y})\| \\
&= L_1 \prod_{\ell \neq 1} \rho_\ell \|\mathbf{X} - \mathbf{Y}\| + L_2 \prod_{\ell \neq 2} \rho_\ell \|F^{(1)}(\mathbf{X}) - F^{(1)}(\mathbf{Y})\| + \ldots + L_L \prod_{\ell \neq L} \rho_\ell \|F^{(L-1)}(\mathbf{X}) - F^{(L-1)}(\mathbf{Y})\|.
\end{aligned}
$$

Following the fact that the Lipschitz constant of $F^{(\ell)}(\cdot)$ is $\prod_{i=1}^{\ell} \rho_i$, we have

$$
\|\nabla f(\mathbf{X}) - \nabla f(\mathbf{Y})\| \leq \left(\sum_{\ell=1}^{L} L_\ell \left(\prod_{i=1}^{\ell-1} \rho_i^2 \right) \left(\prod_{i=\ell+1}^{L} \rho_i \right) \right) \|\mathbf{X} - \mathbf{Y}\|,
$$

which gives the expression for L_f as stated. □

A.2 Proof of Lemma 5

Proof. By definition, we can bound $\mathbb{E}[\|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2]$ by adding and subtracting intermediate terms inside such that each adjacent pair of products differ at most in one factor as follows:

$$
\begin{aligned}
&\mathbb{E}[\|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2] \\
&= \mathbb{E}[\|\mathbf{D}_t^{(1)}\mathbf{D}_t^{(2)} \ldots \mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)} - \nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t)) \ldots \nabla f^{(L-1)}(F^{(L-2)}(\boldsymbol{\theta}_t))\nabla f^{(L)}(F^{(L-1)}(\boldsymbol{\theta}_t))\|^2] \\
&\leq (2L-1)\Big(\mathbb{E}[\|\mathbf{D}_t^{(1)}\mathbf{D}_t^{(2)} \ldots \mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)} - \nabla f^{(1)}(\boldsymbol{\theta}_t)\mathbf{D}_t^{(2)} \ldots \mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)}\|^2] \\
&\quad + \mathbb{E}[\|\nabla f^{(1)}(\boldsymbol{\theta}_t)\mathbf{D}_t^{(2)} \ldots \mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)} - \nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(\mathbf{H}_t^{(1)}) \ldots \mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)}\|^2] \\
&\quad + \mathbb{E}[\|\nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(\mathbf{H}_t^{(1)}) \ldots \mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)} - \nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t)) \ldots \mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)}\|^2] + \ldots \\
&\quad + \mathbb{E}[\|\nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t)) \ldots \nabla f^{(L-1)}(F^{(L-2)}(\boldsymbol{\theta}_t))\mathbf{D}_t^{(L)} \\
&\qquad - \nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t)) \ldots \nabla f^{(L-1)}(F^{(L-2)}(\boldsymbol{\theta}_t))\nabla f^{(L)}(\mathbf{H}_t^{(L-1)})\|^2] \\
&\quad + \mathbb{E}[\|\nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t)) \ldots \nabla f^{(L-1)}(F^{(L-2)}(\boldsymbol{\theta}_t))\nabla f^{(L)}(\mathbf{H}_t^{(L-1)}) \\
&\qquad - \nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t)) \ldots \nabla f^{(L-1)}(F^{(L-2)}(\boldsymbol{\theta}_t))\nabla f^{(L)}(F^{(L-1)}(\boldsymbol{\theta}_t))\|^2]\Big) \\
&\leq (2L-1)\sum_{\ell=1}^{L} \mathbb{E}\Big[\Big(\prod_{i=1}^{\ell-1} \|\nabla f^{(i)}(F^{(i-1)}(\boldsymbol{\theta}_t))\|^2\Big)\Big(\prod_{i=\ell+1}^{L} \|\mathbf{D}_t^{(i)}\|^2\Big) \\
&\qquad \Big(\|\mathbf{D}_t^{(\ell)} - \nabla f^{(\ell)}(\mathbf{H}_t^{(\ell-1)})\|^2 + \mathbb{1}_{\ell \geq 2}\|\nabla f^{(\ell)}(\mathbf{H}_t^{(\ell-1)}) - \nabla f^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t))\|^2\Big)\Big] \\
&\leq 2L\sum_{\ell=1}^{L} \mathbb{E}\Big[\Big(\prod_{i=1}^{\ell-1} \rho_i^2\Big)\Big(\prod_{i=\ell+1}^{L} \rho_i^2\Big)\Big(\|\mathbf{D}_t^{(\ell)} - \nabla f^{(\ell)}(\mathbf{H}_t^{(\ell-1)})\|^2 + \mathbb{1}_{\ell \geq 2}\|\nabla f^{(\ell)}(\mathbf{H}_t^{(\ell-1)}) - \nabla f^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t))\|^2\Big)\Big] \\
&\leq 2L\sum_{\ell=1}^{L} \Big(\prod_{i=1}^{\ell-1} \rho_i^2\Big)\Big(\prod_{i=\ell+1}^{L} \rho_i^2\Big) \cdot \mathbb{E}[\|\mathbf{D}_t^{(\ell)} - \nabla f^{(\ell)}(\mathbf{H}_t^{(\ell-1)})\|^2] \\
&\quad + 2L\sum_{\ell=2}^{L} \Big(\prod_{i=1}^{\ell-1} \rho_i^2\Big)\Big(\prod_{i=\ell+1}^{L} \rho_i^2\Big) L_\ell^2 \cdot \mathbb{E}[\|\mathbf{H}_t^{(\ell-1)} - F^{(\ell-1)}(\boldsymbol{\theta}_t)\|^2],
\end{aligned}
$$

where the last inequality is due to Assumption 1.

According to the Algorithm 1, $\mathbf{H}_t^{(\ell)}$ is calculated as $\mathbf{H}_t^{(\ell)} = f_{\xi_\ell}^{(\ell)} \circ \ldots \circ f_{\xi_1}^{(1)}(\boldsymbol{\theta}_t)$ while $F^{(\ell)}(\boldsymbol{\theta}_t)$ is calculated as $F^{(\ell)}(\boldsymbol{\theta}_t) = f^{(\ell)} \circ \ldots \circ f^{(1)}(\boldsymbol{\theta}_t)$. Similarly, we can bound $\mathbb{E}[\|\mathbf{H}_t^{(\ell)} - F^{(\ell)}(\boldsymbol{\theta}_t)\|^2]$ by adding and subtracting intermediate terms inside the such that each adjacent pair of products differ at most in one factor. For the ease of presentation, let $\boldsymbol{\theta}_t := F^{(0)}(\boldsymbol{\theta}_t)$ and $\prod_{j=\ell+1}^{\ell} \rho_j = 1$, we have

$$
\begin{aligned}
\mathbb{E}[\|\mathbf{H}_t^{(\ell)} - F^{(\ell)}(\boldsymbol{\theta}_t)\|^2] &= \mathbb{E}[\|f_{\xi_\ell}^{(\ell)} \circ f_{\xi_{\ell-1}}^{(\ell-1)} \circ \ldots \circ f_{\xi_1}^{(1)}(\boldsymbol{\theta}_t) - f^{(\ell)} \circ f^{(\ell-1)} \circ \ldots \circ f^{(1)}(\boldsymbol{\theta}_t)\|^2] \\
&\leq \ell \Big(\mathbb{E}[\|f_{\xi_\ell}^{(\ell)} \circ f_{\xi_{\ell-1}}^{(\ell-1)} \circ \ldots \circ f_{\xi_1}^{(1)}(\boldsymbol{\theta}_t) - f_{\xi_\ell}^{(\ell)} \circ f_{\xi_{\ell-1}}^{(\ell-1)} \circ \ldots \circ f_{\xi_2}^{(2)}(F^{(1)}(\boldsymbol{\theta}_t))\|^2] \\
&\quad + \mathbb{E}[\|f_{\xi_\ell}^{(\ell)} \circ f_{\xi_{\ell-1}}^{(\ell-1)} \circ \ldots \circ f_{\xi_2}^{(2)}(F^{(1)}(\boldsymbol{\theta}_t)) - f_{\xi_\ell}^{(\ell)} \circ f_{\xi_{\ell-1}}^{(\ell-1)} \circ \ldots \circ f_{\xi_3}^{(3)}(F^{(2)}(\boldsymbol{\theta}_t))\|^2] + \ldots \\
&\quad + \mathbb{E}[\|f_{\xi_\ell}^{(\ell)}(F_t^{(\ell-1)}) - F_t^{(\ell)}\|^2] \Big) \\
&\leq \ell \Big(\prod_{i=2}^{\ell} \rho_i^2 \mathbb{E}[\|f_{\xi_1}^{(1)}(\boldsymbol{\theta}_t) - F^{(1)}(\boldsymbol{\theta}_t)\|^2] + \prod_{i=3}^{\ell} \rho_i^2 \mathbb{E}[\|f_{\xi_2}^{(2)}(F^{(1)}(\boldsymbol{\theta}_t)) - F^{(2)}(\boldsymbol{\theta}_t)\|^2] \\
&\quad + \ldots + \mathbb{E}[\|f_{\xi_\ell}^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t)) - F^{(\ell)}(\boldsymbol{\theta}_t)\|^2] \Big) \\
&= \ell \sum_{i=1}^{\ell} \left(\prod_{j=i+1}^{\ell} \rho_j^2 \mathbb{E}[\|f_{\xi_j}^{(j)}(F^{(j-1)}(\boldsymbol{\theta}_t)) - F^{(j)}(\boldsymbol{\theta}_t)\|^2] \right)
\end{aligned}
$$

Define $\mathbb{V}^{(\ell)} = \mathbb{E}[\|f_{\xi_\ell}^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t)) - F^{(\ell)}(\boldsymbol{\theta}_t)\|^2]$ and $\mathbb{G}^{(\ell)} = \mathbb{E}[\|\mathbf{D}_t^{(\ell)} - \nabla f^{(\ell)}(\mathbf{H}_t^{(\ell-1)})\|^2$. Combining the above two inequalities yields

$$
\begin{aligned}
\mathbb{E}[\|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2] \leq 2L \sum_{\ell=1}^{L} \left(\prod_{i=1}^{\ell-1} \rho_i^2 \right) \left(\prod_{i=\ell+1}^{L} \rho_i^2 \right) \cdot \mathbb{G}^{(\ell)} \\
+ 2L \sum_{\ell=2}^{L} \left(\prod_{i=1}^{\ell-1} \rho_i^2 \right) \left(\prod_{i=\ell+1}^{L} \rho_i^2 \right) L_\ell^2 \cdot \ell \sum_{i=1}^{\ell} \left(\prod_{j=i+1}^{\ell} \rho_j^2 \cdot \mathbb{V}^{(j)} \right),
\end{aligned}
$$

where the last inequality is due to

$$
\begin{aligned}
\mathbb{E}[\|\mathbf{H}_t^{(\ell-1)} - F^{(\ell-1)}(\boldsymbol{\theta}_t)\|^2] &\leq (\ell-1) \sum_{i=1}^{\ell-1} \left(\prod_{j=i+1}^{\ell-1} \rho_j^2 \mathbb{V}^{(j)} \right) \\
&\leq \ell \sum_{i=1}^{\ell} \left(\prod_{j=i+1}^{\ell} \rho_j^2 \mathbb{V}^{(j)} \right)
\end{aligned}
$$

□

A.3 Proof of Lemma 6

Proof. According to the update rule in Algorithm 1 and Assumption 1, we can bound the function approximation variance by

$$
\mathbb{E}[\|f_{\xi_\ell}^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t)) - F^{(\ell)}(\boldsymbol{\theta}_t)\|^2] \leq \frac{\delta_\ell^2}{S_\ell}
$$

□

A.4 Proof of Lemma 7

Proof. According to the update rule in Algorithm 1 and Assumption 1, we can bound the layer gradient variance by

$$
\mathbb{E}[\|\mathbf{D}_t^{(\ell)} - \nabla f_t^{(\ell)}(\mathbf{H}_t^{(\ell-1)})\|^2] \leq \frac{\sigma_\ell^2}{S_\ell}
$$

□

A.5 Proof of Lemma 8

Proof. From the Lemma 4 that $f(\cdot)$ has Lipschitz continuous gradient with parameter L_f and the update rule $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{g}_t$ we obtain

$$\begin{aligned} f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t) &\leq \langle \nabla f(\boldsymbol{\theta}_t), \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t \rangle + \frac{L_f}{2} \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|^2 \\ &= -\eta \langle \nabla f(\boldsymbol{\theta}_t), \mathbf{g}_t \rangle + \frac{L_f}{2} \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|^2 \\ &= -\eta \langle \nabla f(\boldsymbol{\theta}_t), \nabla f(\boldsymbol{\theta}_t) - \nabla f(\boldsymbol{\theta}_t) + \mathbf{g}_t \rangle + \frac{L_f}{2} \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|^2 \\ &= -\eta \|\nabla f(\boldsymbol{\theta}_t)\|^2 - \eta \langle \nabla f(\boldsymbol{\theta}_t), \mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t) \rangle + \frac{\eta^2 L_f}{2} \|\mathbf{g}_t\|^2 \end{aligned}$$

Adding and subtracting $\nabla f(\boldsymbol{\theta}_t)$ to $\mathbf{g}_t$ gives

$$\begin{aligned} \|\mathbf{g}_t\|^2 &= \|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t) + \nabla f(\boldsymbol{\theta}_t)\|^2 \\ &= \|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2 + \|\nabla f(\boldsymbol{\theta}_t)\|^2 + 2\langle \nabla f(\boldsymbol{\theta}_t), \mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t) \rangle \end{aligned}$$

Plugging the above equality back, we get

$$\begin{aligned} f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t) \leq \left(\frac{\eta^2 L_f}{2} - \eta \right) \|\nabla f(\boldsymbol{\theta}_t)\|^2 + \left(\eta^2 L_f - \eta \right) \langle \nabla f(\boldsymbol{\theta}_t), \mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t) \rangle \\ + \frac{\eta^2 L_f}{2} \|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2 \end{aligned}$$

Using the fact that $2\langle \nabla f(\boldsymbol{\theta}_t), \mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t) \rangle \leq \|\nabla f(\boldsymbol{\theta}_t)\|^2 + \|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2$, we have

$$f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t) \leq \left(\eta^2 L_f - \frac{3}{2}\eta \right) \|\nabla f(\boldsymbol{\theta}_t)\|^2 + \left(\eta^2 L_f - \frac{1}{2}\eta \right) \|\mathbf{g}_t - \nabla f(\boldsymbol{\theta}_t)\|^2$$

Taking expectation on both side with respect to randomness in sampling mini-batch in different layers and rearranging the terms results in

$$\begin{aligned} \left(\frac{3}{2}\eta - \eta^2 L_f \right) \mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t)\|^2] &\leq \mathbb{E}[f(\boldsymbol{\theta}_t)] - \mathbb{E}[f(\boldsymbol{\theta}_{t+1})] + \left(\eta^2 L_f - \frac{1}{2}\eta \right) \Delta \\ &\leq \mathbb{E}[f(\boldsymbol{\theta}_t)] - \mathbb{E}[f(\boldsymbol{\theta}_{t+1})] + \eta^2 L_f \Delta \end{aligned}$$

Summing up above inequality for all T iterations and using the fact that $f(\mathbf{w}_*) \leq f(\boldsymbol{\theta}_{t+1})$ gives

$$\left(\frac{3}{2}\eta - \eta^2 L_f \right) \sum_{t=1}^{T} \mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t)\|^2] \leq \mathbb{E}[f(\boldsymbol{\theta}_1)] - \mathbb{E}[f(\boldsymbol{\theta}_*)] + T\eta^2 L_f \Delta$$

Dividing both side by $T\left(\frac{3}{2}\eta - \eta^2 L_f\right)$ results in

$$\begin{aligned} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t)\|^2] &\leq \left(\frac{3}{2}\eta - \eta^2 L_f \right)^{-1} \frac{(\mathbb{E}[f(\boldsymbol{\theta}_1)] - \mathbb{E}[f(\boldsymbol{\theta}_*)])}{T} + \Delta \frac{2\eta^2 L_f}{(3\eta - 2\eta^2 L_f)} \\ &= \left(3\eta - 2\eta^2 L_f \right)^{-1} \frac{2\left(\mathbb{E}[f(\boldsymbol{\theta}_1)] - \mathbb{E}[f(\boldsymbol{\theta}_*)]\right)}{T} + \Delta \frac{2\eta L_f}{(3 - 2\eta L_f)} \end{aligned}$$

By choosing $\eta = \min\left\{\frac{1}{L_f}, \sqrt{\frac{\mathbb{E}[f(\boldsymbol{\theta}_1)]-\mathbb{E}[f(\boldsymbol{\theta}_*)]}{\Delta L_f T}}\right\}$ we have

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t)\|^2] &\leq \frac{1}{\eta\,(3-2\eta L_f)}\frac{2\,(\mathbb{E}[f(\boldsymbol{\theta}_1)]-\mathbb{E}[f(\boldsymbol{\theta}_*)])}{T}+\Delta\frac{2\eta L_f}{(3-2\eta L_f)}\\ &\leq \frac{1}{\eta}\frac{2\,(\mathbb{E}[f(\boldsymbol{\theta}_1)]-\mathbb{E}[f(\boldsymbol{\theta}_*)])}{T}+2\Delta\eta L_f\\ &\leq \mathcal{O}\left(\sqrt{\Delta/T}\right)\end{aligned}$$

Since $\tilde{\boldsymbol{\theta}}$ is decided in a way that has minimum gradient, it follows

$$\mathbb{E}[\|f(\tilde{\boldsymbol{\theta}})\|^2] \leq \mathcal{O}\left(\sqrt{\Delta/T}\right)$$

which gives the bound as stated in the theorem. □

B Variance Analysis

In this section, we analyze the variance of the approximation embedding for the sampled nodes.

Recall that a single layer of GCN can be written as $\mathbf{H}^{(\ell)} = \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})$, where $\mathbf{L}$ is the normalized Laplacian matrix, $\mathbf{H}^{(\ell)}$ is the node embedding, $\mathbf{W}^{(\ell)}$ is the weight matrix in ℓth layer. Denote $\mathbf{L}^{(\ell)}$ as the induced Laplacian matrix after sampling at ℓth layer, and embedding matrix used in `SGCN` training. Denote $\mathcal{V}_\ell$ as the set of nodes sampled in ℓth layer with size $N_\ell = |\mathcal{V}_\ell|$, and $N = |\mathcal{V}|$ as the number of nodes in graph. Recall that D denotes the average neighbor size and s stands for the number of samples.

We denote $L_{i,j}$ as the element at the ith row , jth column of Laplacian matrix $\mathbf{L}$, denote $\mathbf{L}_{i,*}$ and $\mathbf{L}_{*,j}$ as the ith row and jth column of Laplacian matrix $\mathbf{L}$, respectively, and denote $\mathbf{H}_i$ as the ith row of embedding matrix $\mathbf{H}$.

In order to explicitly compare the variance of different algorithms, we make the following assumptions on the Laplacian matrices $\mathbf{L}^{(\ell)}$ and feature embedding matrices $\mathbf{H}^{(\ell)}$.

Assumption 2. *We assume there exist constant ψ such that the following holds for $i \in [N]$*

$$\|\mathbf{L}_{i,*}\|_2 \leq \psi,\ \|\mathbf{L}_{*,i}\|_2 \leq \psi$$

Assumption 3. *We assume there exist constant γ_ℓ such that the following holds for $i \in [N]$*

$$\|\mathbf{H}_i^{(\ell-1)}\mathbf{W}^{(\ell)}\|_2 \leq \gamma_\ell$$

We make the following assumption on the Lipschitz continuity of the activation function which holds for standard activation functions used in our experiments.

Assumption 4. *We assume there exist constant β such that the following holds*

$$\|\sigma(\boldsymbol{x}) - \sigma(\boldsymbol{y})\| \leq \beta\|\boldsymbol{x}-\boldsymbol{y}\|$$

We first theoretically analyse the function approximation variance for `SGCN` variants.

Lemma 9 (Function approximation variance of GraphSage [5])**.** *We assume that for each node, GraphSage randomly samples s neighbors to estimate the node embedding, then we have*

$$\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \leq \frac{D}{s}\psi^2\beta^2\gamma_\ell^2$$

Proof of Lemma 9. From Assumption 4 we have

$$\begin{aligned}
&\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \\
&\leq \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\mathbb{E}[\|\sum_{j\in\mathcal{V}_{\ell-1}} L_{i,j}^{(\ell)}\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)} - \sum_{j\in\mathcal{V}} L_{i,j}\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2] \\
&= \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\left(\frac{D}{s}\sum_{j\in\mathcal{V}}\|L_{i,j}\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2 - \|\mathbf{L}_{i,*}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2\right) \\
&\leq \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\frac{D}{s}\sum_{j\in\mathcal{V}} L_{i,j}^2\|\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2 \\
&\leq \frac{D}{s}\psi^2\beta^2\gamma_\ell^2,
\end{aligned}$$

where the last inequality hold due to Assumption 2 and Assumption 3. □

Lemma 10 (Function approximation variance of FastGCN [8])**.** *Assume that for each node, FastGCN randomly samples N_ℓ nodes at the ℓth layer to estimate the node embedding. Then, we have*

$$\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \leq \frac{N^2}{N_\ell^2}\psi^2\beta^2\gamma_\ell^2$$

Proof of Lemma 10. By Assumption 4 we have

$$\begin{aligned}
&\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \\
&\leq \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\mathbb{E}[\|\sum_{j\in\mathcal{V}_{\ell-1}} L_{i,j}^{(\ell)}\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)} - \sum_{j\in\mathcal{V}} L_{i,j}\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2] \\
&\leq \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\left(\sum_{j\in\mathcal{V}}\frac{1}{p_j^{(\ell)}}\|L_{i,j}\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2 - \|\mathbf{L}_{i,*}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2\right) \\
&\leq \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\left(\sum_{j\in\mathcal{V}}\frac{1}{p_j^{(\ell)}}L_{i,j}^2\|\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2\right)
\end{aligned}$$

By applying the importance sampling probabilities $p_j^{(\ell)} = N_\ell \cdot (\sum_{k\in\mathcal{V}} L_{k,j}^2)/(\sum_{k\in\mathcal{V}}\|\mathbf{L}_{k,*}\|^2)$, we have

$$\begin{aligned}
&\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \\
&\leq \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\left(\sum_{j\in\mathcal{V}}\frac{1}{p_j^{(\ell)}}L_{i,j}^2\|\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2\right) \\
&\leq \frac{\beta^2}{N_\ell^2}\sum_{i\in\mathcal{V}_\ell}\left(\sum_{j\in\mathcal{V}}\frac{\sum_{k\in\mathcal{V}}\|\mathbf{L}_{k,*}\|^2}{\sum_{k\in\mathcal{V}}L_{k,j}^2}L_{i,j}^2\|\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2\right) \\
&\leq \frac{\beta^2}{N_\ell^2}\left(\sum_{j\in\mathcal{V}}\sum_{k\in\mathcal{V}}\|\mathbf{L}_{k,*}\|^2\cdot\frac{\sum_{i\in\mathcal{V}_\ell}L_{i,j}^2}{\sum_{i\in\mathcal{V}}L_{i,j}^2}\cdot\|\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2\right) \\
&\leq \frac{N^2}{N_\ell^2}\psi^2\beta^2\gamma_\ell^2,
\end{aligned}$$

where the last inequality holds due to Assumption 2 and Assumption 3. □

Lemma 11 (Function approximation variance of LADIES [10]). *Assuming that for each node, LADIES randomly samples N_ℓ nodes at the ℓth layer to estimate the node embedding, we have*

$$\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \leq \frac{N}{N_\ell}\psi^2\beta^2\gamma_\ell^2$$

Proof of Lemma 11. By Assumption 4 we have

$$\begin{aligned}
&\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \\
&\leq \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\mathbb{E}[\|\sum_{j\in\mathcal{V}_{\ell-1}} L_{i,j}^{(\ell)}\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)} - \sum_{j\in\mathcal{V}} L_{i,j}\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2] \\
&\leq \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\left(\sum_{j\in\mathcal{V}}\frac{1}{p_j^{(\ell)}}\|L_{i,j}\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2 - \|\mathbf{L}_{i,*}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2\right) \\
&\leq \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\left(\sum_{j\in\mathcal{V}}\frac{1}{p_j^{(\ell)}}L_{i,j}^2\|\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2\right)
\end{aligned}$$

By applying the importance sampling probabilities $p_j^{(\ell)} = N_\ell \cdot (\sum_{k\in\mathcal{V}_\ell} L_{k,j}^2)/(\sum_{k\in\mathcal{V}_\ell}\|\mathbf{L}_{k,*}\|^2)$, we have

$$\begin{aligned}
&\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \\
&\leq \frac{\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\left(\sum_{j\in\mathcal{V}}\frac{1}{p_j^{(\ell)}}L_{i,j}^2\|\mathbf{H}_j^{(\ell)}\|^2\right) \\
&\leq \frac{\beta^2}{N_\ell^2}\sum_{i\in\mathcal{V}_\ell}\left(\sum_{j\in\mathcal{V}}\frac{\sum_{k\in\mathcal{V}_\ell}\|\mathbf{L}_{k,*}\|^2}{\sum_{k\in\mathcal{V}_\ell}L_{k,j}^2}L_{i,j}^2\|\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2\right) \\
&\leq \frac{\beta^2 N}{N_\ell^2}\left(\sum_{j\in\mathcal{V}}\sum_{k\in\mathcal{V}_\ell}\|\mathbf{L}_{k,*}\|^2\|\mathbf{H}_j^{(\ell-1)}\mathbf{W}^{(\ell)}\|^2\right) \\
&\leq \frac{N}{N_\ell}\psi^2\beta^2\gamma_\ell^2,
\end{aligned}$$

where the last inequality hold due to Assumption 2 and Assumption 3. □

To compare the variance of `SGCN++` with the variance of `SGCN`, we make the following assumption

Assumption 5. *We assume there exist constant $\bar{\gamma}_\ell$ such that the following holds for $i \in [N]$ and any time t*

$$\|\mathbf{H}_i^{(\ell-1)}(\mathbf{W}_t^{(\ell)} - \mathbf{W}_{t-1}^{(\ell)})\|_2 \leq \bar{\gamma}_\ell,$$

and when $t \to \infty$, we have $\bar{\gamma} \approx 0$.

Then, we analyse the function approximation variance of `SGCN++` for each layer independently. Suppose the inner-loop runs for at most K steps and Assumption 5 holds. Let us denote $\mathbf{W}_t^{(\ell)}$ as the weight matrix at

ℓth layer at iteration t. Using Lemma 3, we have for any $t = 2, \ldots, K$

$$\begin{aligned}\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] &\leq \frac{1}{N_\ell}\sum_{t=2}^{K}\beta^2\mathbb{E}[\|\mathbf{L}^{(\ell)}\mathbf{H}^{(\ell-1)}\mathbf{W}_t^{(\ell)} - \mathbf{L}^{(\ell)}\mathbf{H}^{(\ell-1)}\mathbf{W}_{t-1}^{(\ell)}\|^2] \\ &= \frac{1}{N_\ell}\sum_{t=2}^{K}\beta^2\mathbb{E}[\|\mathbf{L}^{(\ell)}\mathbf{H}^{(\ell-1)}(\mathbf{W}_t^{(\ell)} - \mathbf{W}_{t-1}^{(\ell)})\|^2] \\ &\leq \frac{K\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\sum_{j\in\mathcal{V}}\frac{1}{p_j^{(\ell)}}L_{i,j}^2\|\mathbf{H}_j^{(\ell-1)}(\mathbf{W}_t^{(\ell)} - \mathbf{W}_{t-1}^{(\ell)})\|^2\end{aligned} \tag{6}$$

Therefore, we have the following results on the function approximation variance of training GraphSage, FastGCN, LADIES using `SGCN++`.

Lemma 12 (Function approximation variance of GraphSage++)**.** *Assume that for each node, GraphSage randomly sample s neighbors to estimate the node embedding, then we have*

$$\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \leq \frac{KD}{s}\psi^2\beta^2\bar{\gamma}_\ell^2$$

Proof of Lemma 12. Set the $p_j^{(\ell)}$ in Equation 6 as $p_j^{(\ell)} = s/D$ and by Assumption 5 we have

$$\begin{aligned}&\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \\ &\leq \frac{K\beta^2}{N_\ell}\sum_{i\in\mathcal{V}_\ell}\sum_{j\in\mathcal{V}}\frac{D}{s}L_{i,j}^2\|\mathbf{H}_j^{(\ell-1)}(\mathbf{W}_t^{(\ell)} - \mathbf{W}_{t-1}^{(\ell)})\|^2 \\ &\leq \frac{KD}{s}\psi^2\beta^2\bar{\gamma}_\ell^2\end{aligned} \tag{7}$$

□

Lemma 13 (Function approximation variance of FastGCN++)**.** *We assume that for each node, FastGCN randomly sample N_ℓ nodes at the ℓth layer to estimate the node embedding, then we have*

$$\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \leq \frac{KN^2}{N_\ell^2}\psi^2\beta^2\bar{\gamma}_\ell^2$$

Proof of Lemma 13. Set the p_j in Equation 6 as $p_j^{(\ell)} = N_\ell \cdot (\sum_{k\in\mathcal{V}} L_{k,j}^2)/(\sum_{k\in\mathcal{V}}\|\mathbf{L}_{k,*}\|^2)$ and by Assumption 5 we have

$$\begin{aligned}&\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \\ &\leq \frac{K\beta^2}{N_\ell^2}\sum_{i\in\mathcal{V}_\ell}\sum_{j\in\mathcal{V}}\frac{\sum_{k\in\mathcal{V}}\|\mathbf{L}_{k,*}\|^2}{\sum_{k\in\mathcal{V}}L_{k,j}^2}L_{i,j}^2\|\mathbf{H}_j^{(\ell-1)}(\mathbf{W}_t^{(\ell)} - \mathbf{W}_{t-1}^{(\ell)})\|^2 \\ &\leq \frac{KN^2}{N_\ell^2}\psi^2\beta^2\bar{\gamma}_\ell^2\end{aligned} \tag{8}$$

□

Lemma 14 (Function approximation variance of LADIES++)**.** *We assume that for each node, LADIES randomly sample N_ℓ nodes at the ℓth layer to estimate the node embedding, then we have*

$$\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \leq \frac{KN}{N_\ell}\psi^2\beta^2\bar{\gamma}_\ell^2$$

Proof of Lemma 14. Set the $p_j^{(\ell)}$ in Equation 6 as $p_j^{(\ell)} = N_\ell \cdot (\sum_{k\in\mathcal{V}_\ell} L_{k,j}^2)/(\sum_{k\in\mathcal{V}_\ell} \|\mathbf{L}_{k,*}\|^2)$ and by Assumption 5 we have

$$\begin{aligned}
&\frac{1}{N_\ell}\mathbb{E}[\|\mathbf{H}^{(\ell)} - \sigma(\mathbf{L}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})\|^2] \\
&\leq \frac{K\beta^2}{N_\ell^2}\sum_{i\in\mathcal{V}_\ell}\sum_{j\in\mathcal{V}}\frac{\sum_{k\in\mathcal{V}_\ell}\|\mathbf{L}_{k,*}\|^2}{\sum_{k\in\mathcal{V}_\ell} L_{k,j}^2} L_{i,j}^2 \|\mathbf{H}_j^{(\ell-1)}(\mathbf{W}_t^{(\ell)} - \mathbf{W}_{t-1}^{(\ell)})\|^2 \qquad (9)\\
&\leq \frac{KN}{N_\ell}\psi^2\beta^2\bar{\gamma}_\ell^2
\end{aligned}$$

□

Therefore, by choosing $K \leq (\gamma_\ell^2/\bar{\gamma}_\ell^2)$ for $\ell \in [L]$, we can always guarantee a lower variance.

C Proof of Theorem 2

The proof of Theorem 2 is based on the following lemmas and propositions. We show that the proposed algorithm can significantly reduce the variance, which leads to an $\mathcal{O}(1/\epsilon)$ convergence rate.

The following lemma shows that the mean-square error of the stochastic gradient of `SGCN++` can be bounded by the linear combination of $\mathbb{E}[\|\mathbf{D}_t^{(i)} - \nabla f^{(i)}(\mathbf{H}_t^{(i-1)})\|^2]$ (which is closely related to the layer gradient variance) and $\mathbb{E}[\|\nabla f^{(i)}(\mathbf{H}_t^{(i-1)}) - \nabla f^{(i)}(F^{(i-1)}(\boldsymbol{\theta}_t))\|^2]$ (which is closely related to the function approximation variance).

Lemma 15. *Consider the inner-loop in Algorithm 2. Suppose Assumption 1 holds. Then we have:*

$$\begin{aligned}
\mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t) - \mathbf{g}_t\|^2] \leq\ & 2L\sum_{i=1}^{L}\left(\prod_{\ell=1}^{i-1}\rho_\ell^2\right)\left(\prod_{\ell=i+1}^{L}(1+\omega_\ell)^2\rho_\ell^2\right)\mathbb{E}[\|\mathbf{D}_t^{(i)} - \nabla f^{(i)}(\mathbf{H}_t^{(i-1)})\|^2] \\
&+ 2L\sum_{i=2}^{L}\left(\prod_{\ell=1}^{i-1}\rho_\ell^2\right)\left(\prod_{\ell=i+1}^{L}(1+\omega_\ell)^2\rho_\ell^2\right)\mathbb{E}[\|\nabla f^{(i)}(\mathbf{H}_t^{(i-1)}) - \nabla f^{(i)}(F^{(i-1)}(\boldsymbol{\theta}_t))\|^2]
\end{aligned}$$

The following lemma provides an upper-bound on $\mathbb{E}[\|\mathbf{D}_t^{(\ell)} - \nabla f(\mathbf{H}_t^{(\ell-1)})\|^2]$.

Lemma 16. *Consider the inner-loop of Algorithm 2. Let S_ℓ be the number of samples at the ℓth layer at step $t = 1$, and S'_ℓ the number of samples at the ℓ-th layer at step $t > 1$. Under Assumption 1 we have*

$$\mathbb{E}[\|\mathbf{D}_t^{(\ell)} - \nabla f(\mathbf{H}_t^{(\ell-1)})\|^2] \leq \frac{L_\ell^2}{S'_\ell}\left(\prod_{i=1}^{\ell-1}\rho_i^2\right)\sum_{j=1}^{t}\eta^2\mathbb{E}[\|\mathbf{g}_j\|^2]$$

The following lemma provides an upper-bound on $\mathbb{E}[\|\mathbf{H}_t^{(\ell)} - f^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t))\|^2]$.

Lemma 17. *Consider the inner-loop of Algorithm 2. Let S_ℓ be the number of samples at the ℓth layer at step $t = 1$, and S'_ℓ be the number of samples at the ℓth layer at step $t > 1$. Under Assumption 1 we have*

$$\mathbb{E}[\|\mathbf{H}_t^{(\ell)} - f^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t))\|^2] \leq \ell\cdot\left(\prod_{j=1}^{\ell}\rho_j^2\right)\left(\sum_{i=1}^{\ell}\frac{\eta^2}{S'_i}\right)\left(\sum_{j=1}^{t}\mathbb{E}[\|\mathbf{g}_j\|^2]\right)$$

The following lemma provides an upper-bound on the mean-square error of the stochastic gradient of `SGCN++`.

Lemma 18. *Let $S'_1 = \ldots = S'_L = S'$, define $\phi(i) = \phi_1(i) + \phi_2(i)$ where*

$$\phi_1(i) = 2L \sum_{i=1}^{L} \left(\prod_{\ell=1}^{i-1} \rho_\ell^2 \right) \left(\prod_{\ell=i+1}^{L} (1+\omega_\ell)^2 \rho_\ell^2 \right) L_i^2 \left(\prod_{\ell=1}^{i-1} \rho_\ell^2 \right)$$

and

$$\phi_2(i) = \mathbb{1}_{i\geq 2} 2L \sum_{i=2}^{L} \left(\prod_{\ell=1}^{i-1} \rho_\ell^2 \right) \left(\prod_{\ell=i+1}^{L} (1+\omega_\ell)^2 \rho_\ell^2 \right) \cdot i^2 \cdot \left(\prod_{\ell=1}^{i} \rho_\ell^2 \right)$$

Then we have

$$\mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t) - \mathbf{g}_t\|^2] \leq \sum_{i=1}^{t} \frac{\phi(i)}{S'} \eta^2 \mathbb{E}[\|\mathbf{g}_i\|^2]$$

The following lemma shows the connection between the mean-square error of `SGCN++` to its convergence rate.

Lemma 19. *Under Assumption 1, let $\phi = \max_{i\in[L]} \phi(i)$, if the parameters η, K and S' are chosen such that*

$$\zeta := \frac{\eta}{2} - \frac{L_f \eta^2}{2} - \frac{\eta^3 \phi K}{2S'} > 0,$$

then the output $\tilde{\boldsymbol{\theta}}$ of Algorithm 2 satisfies

$$\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2] \leq \frac{2}{TK\zeta} \left(1 + \frac{L_f^2 \phi K}{S'} \right) (\mathbb{E}[f(\boldsymbol{\theta}_1)] - \mathbb{E}[f(\boldsymbol{\theta}_*)]) .$$

Proof of Theorem 2. Based on the parameter setting in Theorem 2 as

$$S = n, S' = \sqrt{n}, \eta = \frac{1}{2L_f}, K = \frac{L_f^2 \sqrt{n}}{\phi}$$

we obtain

$$\zeta = \frac{\eta}{2} - \frac{L_f \eta^2}{2} - \frac{\eta^3 \phi K}{2S'} = \frac{1}{16 L_f} > 0.$$

Moreover, for $k = 1$ of each inner-loop, as the algorithm is designed to take the full-batch gradient we have

$$\mathbb{E}[\|\mathbf{g}_k - \nabla f(\boldsymbol{\theta}_k)\|^2 = \mathbb{E}[\|\nabla f(\boldsymbol{\theta}_k) - \nabla f(\boldsymbol{\theta}_k)\|^2] = 0$$

By Lemma 19 with $\zeta = \frac{1}{16 L_f}$ we obtain

$$\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2] \leq \frac{40 L_f}{TK} (f(\boldsymbol{\theta}_1) - f(\boldsymbol{\theta}_*)) .$$

To ensure $\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|] \leq \epsilon$, it is sufficient to show that $(\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|])^2 \leq \mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2] \leq \epsilon^2$ thanks to Jensen's inequality. Thus, we need the total number TK of iterations satisfy $\frac{40 L_f}{TK} (f(\boldsymbol{\theta}_1) - f(\boldsymbol{\theta}_*)) \leq \epsilon$, which gives

$$TK = \frac{40 L_f}{\epsilon} (f(\boldsymbol{\theta}_1) - f(\boldsymbol{\theta}_*)) .$$

Therefore, the total time complexity is given by

$$T \cdot S + K \cdot S' \leq (TK + K) \cdot \frac{S}{K} + TK \cdot S' = TK\sqrt{n} \cdot (\phi / L_f^2) + n + TK\sqrt{n} = O(\sqrt{n}\epsilon^{-1} + n),$$

which completes the proof. □

C.1 Proof of Lemma 15

Proof. As before, we can bound $\mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t) - \mathbf{g}_t\|^2]$ by adding and subtracting the intermediate terms inside such that each adjacent pair of products differ at most in one factor:

$$
\begin{aligned}
&\mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t) - \mathbf{g}_t\|^2] \\
&= \mathbb{E}[\|\mathbf{D}_t^{(1)}\mathbf{D}_t^{(2)}\dots\mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)} - \nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t))\dots\nabla f^{(L-1)}(F^{(L-2)}(\boldsymbol{\theta}_t))\nabla f^{(L)}(F^{(L-1)}(\boldsymbol{\theta}_t))\|^2] \\
&\le (2L-1)\Big(\mathbb{E}[\|\mathbf{D}_t^{(1)}\mathbf{D}_t^{(2)}\dots\mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)} - \nabla f^{(1)}(\boldsymbol{\theta}_t)\mathbf{D}_t^{(2)}\dots\mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)}\|^2] \\
&\quad + \mathbb{E}[\|\nabla f^{(1)}(\boldsymbol{\theta}_t)\mathbf{D}_t^{(2)}\dots\mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)} - \nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(\mathbf{H}_t^{(1)})\dots\mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)}\|^2] \\
&\quad + \mathbb{E}[\|\nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(\mathbf{H}_t^{(1)})\dots\mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)} - \nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t))\dots\mathbf{D}_t^{(L-1)}\mathbf{D}_t^{(L)}\|^2] + \dots \\
&\quad + \mathbb{E}[\|\nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t))\dots\nabla f^{(L-1)}(F^{(L-2)}(\boldsymbol{\theta}_t))\mathbf{D}_t^{(L)} \\
&\qquad - \nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t))\dots\nabla f^{(L-1)}(F^{(L-2)}(\boldsymbol{\theta}_t))\nabla f^{(L)}(\mathbf{H}_t^{(L-1)})\|^2] \\
&\quad + \mathbb{E}[\|\nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t))\dots\nabla f^{(L-1)}(F^{(L-2)}(\boldsymbol{\theta}_t))\nabla f^{(L)}(\mathbf{H}_t^{(L-1)}) \\
&\qquad - \nabla f^{(1)}(\boldsymbol{\theta}_t)\nabla f^{(2)}(F^{(1)}(\boldsymbol{\theta}_t))\dots\nabla f^{(L-1)}(F^{(L-2)}(\boldsymbol{\theta}_t))\nabla f^{(L)}(F^{(L-2)}(\boldsymbol{\theta}_t))\|^2]\Big) \\
&\le (2L-1)\sum_{i=1}^{L}\mathbb{E}\Big[\Big(\prod_{\ell=1}^{i-1}\|\nabla f^{(\ell)}(F^{(\ell-2)}(\boldsymbol{\theta}_t))\|^2\Big)\Big(\prod_{\ell=i+1}^{L}\|\mathbf{D}_t^{(\ell)}\|^2\Big) \\
&\qquad \Big(\|\mathbf{D}_t^{(i)} - \nabla f^{(i)}(\mathbf{H}_t^{(i-1)})\|^2 + \mathbb{1}_{i\ge 2}\|\nabla f^{(i)}(\mathbf{H}_t^{(i-1)}) - \nabla f^{(i)}(F^{(i-1)}(\boldsymbol{\theta}_t))\|^2\Big)\Big] \\
&\le 2L\sum_{i=1}^{L}\mathbb{E}\Big[\Big(\prod_{\ell=1}^{i-1}\rho_\ell^2\Big)\Big(\prod_{\ell=i+1}^{L}\|\mathbf{D}_t^{(\ell)}\|^2\Big)\Big(\|\mathbf{D}_t^{(i)} - \nabla f^{(i)}(\mathbf{H}_t^{(i-1)})\|^2 + \mathbb{1}_{i\ge 2}\|\nabla f^{(i)}(\mathbf{H}_t^{(i-1)}) - \nabla f^{(i)}(F^{(i-1)}(\boldsymbol{\theta}_t))\|^2\Big)\Big],
\end{aligned}
$$

where the last inequality follows from Assumption 1.

We note that for $\mathbf{D}_t^{(\ell)}$, by the update rule in Algorithm 2 we have $\|\mathbf{D}_t^{(\ell)}\| \le \|\mathbf{D}_t^{(\ell)} - \mathbf{D}_1^{(\ell)}\| + \|\mathbf{D}_1^{(\ell)}\| \le (1+\omega_\ell)\|\mathbf{D}_1^{\ell}\| \le (1+\omega_\ell)\rho_\ell$. Plugging it back we have

$$
\begin{aligned}
\mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t) - \mathbf{g}_t\|^2] \le 2L\sum_{i=1}^{L}\Big(\prod_{\ell=1}^{i-1}\rho_\ell^2\Big)\Big(\prod_{\ell=i+1}^{L}(1+\omega_\ell)^2\rho_\ell^2\Big)\mathbb{E}[\|\mathbf{D}_t^{(i)} - \nabla f^{(i)}(\mathbf{H}_t^{(i-1)})\|^2] \\
+ 2L\sum_{i=2}^{L}\Big(\prod_{\ell=1}^{i-1}\rho_\ell^2\Big)\Big(\prod_{\ell=i+1}^{L}(1+\omega_\ell)^2\rho_\ell^2\Big)\mathbb{E}[\|\nabla f^{(i)}(\mathbf{H}_t^{(i-1)}) - \nabla f^{(i)}(F^{(i-1)}(\boldsymbol{\theta}_t))\|^2]
\end{aligned}
$$

□

C.2 Proof of Lemma 16

Proof. Consider $\mathbf{D}_t^{(\ell)} - \nabla f(\mathbf{H}_t^{(\ell-1)})$ for t in the inner-loop. By Lemma 3 we know

$$
\begin{aligned}
\mathbb{E}[\|\mathbf{D}_t^{(\ell)} - \nabla f(\mathbf{H}_t^{(\ell-1)})\|^2] &\le \mathbb{E}[\|\mathbf{D}_1^{(\ell)} - \nabla f(\mathbf{H}_1^{(\ell-1)})\|^2] + L_\ell^2\sum_{j=2}^{t}\mathbb{E}[\|\mathbf{H}_{j+1}^{(\ell-1)} - \mathbf{H}_j^{(\ell-1)}\|^2] \\
&\le \mathbb{E}[\|\mathbf{D}_1^{(\ell)} - \nabla f(\mathbf{H}_1^{(\ell-1)})\|^2] + \frac{L_\ell^2}{S_\ell'}\Big(\prod_{i=1}^{\ell-1}\rho_i^2\Big)\sum_{j=2}^{t}\eta^2\mathbb{E}[\|\mathbf{g}_{j-1}\|^2] \\
&\le \frac{L_\ell^2}{S_\ell'}\Big(\prod_{i=1}^{\ell-1}\rho_i^2\Big)\sum_{j=1}^{t}\eta^2\mathbb{E}[\|\mathbf{g}_j\|^2]
\end{aligned}
$$

The last inequality holds because at step $t = 1$ we run a full batch and as a result $\mathbb{E}[\|\mathbf{D}_1^{(\ell)} - \nabla f(\mathbf{H}_1^{(\ell-1)})\|^2] = 0$ □

C.3 Proof of Lemma 17

Proof. Consider $\mathbf{H}_t^{(\ell)} - f^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t))$ for $t = 1, \ldots, K$ in the inner-loop. Note that since we run a full-batch at step $t = 1$, we have $\mathbb{E}[\|\mathbf{H}_1^{(\ell)} - f^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_1))\|^2] = 0$. We prove the following inequality by induction

$$\mathbb{E}[\|\mathbf{H}_t^{(\ell)} - f^{(\ell)}(F^{(\ell-1)}(\boldsymbol{\theta}_t)))\|^2] \leq \ell \cdot \left(\prod_{j=1}^{\ell} \rho_j^2\right) \left(\sum_{i=1}^{\ell} \frac{\eta^2}{S_i'}\right) \left(\sum_{j=1}^{t} \mathbb{E}[\|\mathbf{g}_j\|^2]\right)$$

For base case $\ell = 1$ by Lemma 3 we have

$$\mathbb{E}[\|\mathbf{H}_t^{(1)} - f^{(1)}(\boldsymbol{\theta}_t)\|^2] \leq \rho_1^2 \frac{\eta^2}{S_1'} \sum_{j=1}^{t} \mathbb{E}[\|\mathbf{g}_j\|^2],$$

Now we assume the inequality holds for $\ell = k$, and show the result holds for $\ell = k + 1$ as well. To this end, we have

$$\begin{aligned}
&\mathbb{E}[\|\mathbf{H}_t^{(k+1)} - f^{(k+1)}(F^{(k)}(\boldsymbol{\theta}_t))\|^2] \\
&\leq (1+k)\mathbb{E}[\|\mathbf{H}_t^{(k+1)} - f_{\xi_{k+1}}^{(k+1)}(\mathbf{H}_t^{(k)})\|^2] + (1+k^{-1})\mathbb{E}[\|f_{\xi_{k+1}}^{(k+1)}(\mathbf{H}_t^{(k)}) - f^{(k+1)}(F^{(k)}(\boldsymbol{\theta}_t))\|^2] \\
&\leq (1+k)\mathbb{E}[\|\mathbf{H}_t^{(k+1)} - f_{\xi_{k+1}}^{(k+1)}(\mathbf{H}_t^{(k)})\|^2] + (1+k^{-1})\rho_{k+1}^2\mathbb{E}[\|\mathbf{H}_t^{(k)} - F^{(k)}(\boldsymbol{\theta}_t)\|^2] \\
&\leq (1+k)\left[\left(\prod_{i=1}^{k+1}\rho_i^2\right)\frac{\eta^2}{S_{k+1}'}\sum_{j=1}^{t}\mathbb{E}[\|\mathbf{g}_j\|^2]\right] + (1+k^{-1})\rho_{k+1}^2 k\left[\left(\prod_{j=1}^{k}\rho_j^2\right)\left(\sum_{i=1}^{k}\frac{\eta^2}{S_i'}\right)\sum_{j=1}^{t}\mathbb{E}[\|\mathbf{g}_j\|^2]\right] \\
&= (k+1)\left(\prod_{j=1}^{k+1}\rho_j^2\right)\left(\sum_{i=1}^{k+1}\frac{\eta_t^2}{S_i'}\right)\left(\sum_{j=1}^{t}\mathbb{E}[\|\mathbf{g}_j\|^2]\right).
\end{aligned}$$

which shows that the inequality holds for all iterations. □

C.4 Proof of Lemma 18

Proof. Combining Lemma 15, 16, and 17 yields

$$\begin{aligned}
&\mathbb{E}[\|\nabla f(\boldsymbol{\theta}_t) - \mathbf{g}_t\|^2] \\
&\leq 2L\sum_{i=1}^{L}\left(\prod_{\ell=1}^{i-1}\rho_\ell^2\right)\left(\prod_{\ell=i+1}^{L}(1+\omega_\ell)^2\rho_\ell^2\right)\frac{L_i^2}{S_i'}\left(\prod_{\ell=1}^{i-1}\rho_\ell^2\right)\eta^2\sum_{j=1}^{t}\mathbb{E}[\|\mathbf{g}_j\|^2] \\
&\quad + 2L\sum_{i=2}^{L}\left(\prod_{\ell=1}^{i-1}\rho_\ell^2\right)\left(\prod_{\ell=i+1}^{L}(1+\omega_\ell)^2\rho_\ell^2\right)\cdot i\cdot\left(\prod_{\ell=1}^{i}\rho_\ell^2\right)\left(\sum_{\ell=1}^{i}\frac{\eta^2}{S_\ell'}\right)\left(\sum_{j=1}^{t}\mathbb{E}[\|\mathbf{g}_j\|^2]\right) \\
&= \sum_{i=1}^{t}\frac{\phi_1(i)+\phi_2(i)}{S'}\eta^2\mathbb{E}[\|\mathbf{g}_i\|^2],
\end{aligned}$$

where the last equality follows from the definition of $\phi_1(i)$ and $\phi_2(i)$. □

C.5 Proof of Lemma 19

Proof. We first consider the inner-loop for $k = 1, \ldots, K$. By the assumption that objective function f is L_f-smooth, we have

$$\begin{aligned} f(\boldsymbol{\theta}_{k+1}) &\leq f(\boldsymbol{\theta}_k) + \langle \nabla f(\boldsymbol{\theta}_k), \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k \rangle + \frac{L_f}{2} \|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\|^2 \\ &= f(\boldsymbol{\theta}_k) - \eta \langle \nabla f(\boldsymbol{\theta}_k), \mathbf{g}_k \rangle + \frac{L_f \eta^2}{2} \|\mathbf{g}_k\|^2 \\ &= f(\boldsymbol{\theta}_k) - \eta \langle \nabla f(\boldsymbol{\theta}_k) - \mathbf{g}_k, \mathbf{g}_k \rangle - \eta \|\mathbf{g}_k\|^2 + \frac{L_f \eta^2}{2} \|\mathbf{g}_k\|^2 \\ &\leq f(\boldsymbol{\theta}_k) + \frac{\eta}{2} \|\nabla f(\boldsymbol{\theta}_k) - \mathbf{g}_k\|^2 - \left(\frac{\eta}{2} - \frac{L_f \eta^2}{2} \right) \|\mathbf{g}_k\|^2 \end{aligned}$$

Taking expectation on both sides of the above inequality, and using Lemma 18 we get

$$\begin{aligned} &\mathbb{E}[f(\boldsymbol{\theta}_{k+1})] \\ &\leq \mathbb{E}[f(\boldsymbol{\theta}_k)] + \frac{\eta}{2} \mathbb{E}[\|\nabla f(\boldsymbol{\theta}_k) - \mathbf{g}_k\|^2] - \left(\frac{\eta}{2} - \frac{L_f \eta^2}{2} \right) \mathbb{E}\|\mathbf{g}_k\|^2 \\ &= \mathbb{E}[f(\boldsymbol{\theta}_k)] + \frac{\eta}{2} \epsilon_1^2 + \frac{\eta^3}{2} \sum_{i=2}^{k} \frac{\phi(i)^2}{S'} \mathbb{E}[\|\mathbf{g}_{i-1}\|^2] - \left(\frac{\eta}{2} - \frac{L_f \eta^2}{2} \right) \mathbb{E}\|\mathbf{g}_k\|^2, \end{aligned}$$

where the last inequality follows from the fact we take the full-batch gradient at the first step of each inner-loop in Algorithm 2.

Summing up above inequality for all iterations from $k = 1$ to $k = K$, by letting $\phi = \max_i \phi(i)$ and $\zeta = \frac{\eta}{2} - \frac{L_f \eta^2}{2} - \frac{\eta^3 \phi K}{2S'}$ we obtain

$$\begin{aligned} \mathbb{E}[f(\boldsymbol{\theta}_{K+1})] &\leq \mathbb{E}[f(\boldsymbol{\theta}_1)] + \sum_{k=1}^{K} \frac{\eta}{2} \epsilon_1^2 + \frac{\eta^3}{2} \sum_{k=1}^{K} \sum_{i=2}^{k} \frac{\phi^2}{S'} \mathbb{E}[\|\mathbf{g}_{i-1}\|^2] - \left(\frac{\eta}{2} - \frac{L_f \eta^2}{2} \right) \sum_{k=1}^{K} \mathbb{E}[\|\mathbf{g}_k\|^2] \\ &\leq \mathbb{E}[f(\boldsymbol{\theta}_1)] + \frac{\eta^3}{2} \sum_{k=1}^{K} \sum_{i=1}^{K} \frac{\phi^2}{S'} \mathbb{E}[\|\mathbf{g}_i\|^2] - \left(\frac{\eta}{2} - \frac{L_f \eta^2}{2} \right) \sum_{k=1}^{K} \mathbb{E}[\|\mathbf{g}_k\|^2] + \sum_{k=1}^{K} \frac{\eta}{2} \epsilon_1^2 \\ &\leq \mathbb{E}[f(\boldsymbol{\theta}_1)] + \frac{\eta^3}{2} \frac{\phi K}{S'} \sum_{i=1}^{K} \mathbb{E}[\|\mathbf{g}_i\|^2] - \left(\frac{\eta}{2} - \frac{L_f \eta^2}{2} \right) \sum_{k=1}^{K} \mathbb{E}[\|\mathbf{g}_k\|^2] + \sum_{k=1}^{K} \frac{\eta}{2} \epsilon_1^2 \\ &= \mathbb{E}[f(\boldsymbol{\theta}_1)] - \sum_{k=1}^{K} \left(\frac{\eta}{2} - \frac{L_f \eta^2}{2} - \frac{\eta^3 \phi K}{2S'} \right) \mathbb{E}[\|\mathbf{g}_k\|^2] + \sum_{k=1}^{K} \frac{\eta}{2} \epsilon_1^2 \\ &= \mathbb{E}[f(\boldsymbol{\theta}_1)] - \sum_{k=1}^{K} \left(\zeta \mathbb{E}[\|\mathbf{g}_k\|^2] - \frac{\eta}{2} \epsilon_1^2 \right) \end{aligned}$$

Now we consider outer-loop for $t = 1, \ldots, T$. From the definition $\bar{\boldsymbol{\theta}}_t = \boldsymbol{\theta}_1$ and $\bar{\boldsymbol{\theta}}_{t+1} = \boldsymbol{\theta}_{K+1}$, the above equation can be rewrite as

$$\mathbb{E}[f(\bar{\boldsymbol{\theta}}_{t+1})] \leq \mathbb{E}[f(\bar{\boldsymbol{\theta}}_t)] - \sum_{k=1}^{K} \left(\zeta \mathbb{E}\|\mathbf{g}_k\|^2 - \frac{\eta}{2} \epsilon_1^2 \right).$$

Summing over t from $t=1$ to $t=T$ yields

$$\begin{aligned}\mathbb{E}[f(\bar{\boldsymbol{\theta}}_{T+1})] &\leq \mathbb{E}[f(\bar{\boldsymbol{\theta}}_1)] - \sum_{t=1}^{T}\sum_{k=1}^{K}\left(\zeta\mathbb{E}\|\mathbf{g}_{tK+k}\|^2\right) + TK\frac{\eta}{2}\epsilon_1^2 \\ &= \mathbb{E}[f(\bar{\boldsymbol{\theta}}_1)] - \sum_{i=1}^{TK}\zeta\mathbb{E}[\|\mathbf{g}_i\|^2 + TK\frac{\eta}{2}\epsilon_1^2\end{aligned}$$

Rearranging the terms results in

$$\begin{aligned}\sum_{i=1}^{TK}\zeta\mathbb{E}[\|\mathbf{g}_i\|^2] &\leq \mathbb{E}[f(\bar{\boldsymbol{\theta}}_1)] - \mathbb{E}[f(\bar{\boldsymbol{\theta}}_{T+1})] + TK\frac{\eta}{2}\epsilon_1^2 \\ &\leq \mathbb{E}[f(\bar{\boldsymbol{\theta}}_1)] - \mathbb{E}[f(\bar{\boldsymbol{\theta}}_*)] + TK\frac{\eta}{2}\epsilon_1^2.\end{aligned}$$

We next bound $\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2]$, where $\tilde{\boldsymbol{\theta}}$ is selected uniformly at random from all steps. Observe that

$$\begin{aligned}\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2] &= \mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}}) - \mathbf{g}(\tilde{\boldsymbol{\theta}}) + \mathbf{g}(\tilde{\boldsymbol{\theta}})\|^2] \\ &\leq 2\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}}) - \mathbf{g}(\tilde{\boldsymbol{\theta}})\|^2] + 2\mathbb{E}[\|\mathbf{g}(\tilde{\boldsymbol{\theta}})\|^2]\end{aligned}$$

Consider $\mathbf{g}(\tilde{\boldsymbol{\theta}})$ as the stochastic gradient at point $\tilde{\boldsymbol{\theta}}$, we have

$$\begin{aligned}\mathbb{E}[\|\mathbf{g}(\tilde{\boldsymbol{\theta}})\|^2] &= \frac{1}{TK}\sum_{i=1}^{TK}\mathbb{E}[\|\mathbf{g}_i\|^2] \\ &\leq \frac{\mathbb{E}[f(\bar{\boldsymbol{\theta}}_1)] - \mathbb{E}[f(\bar{\boldsymbol{\theta}}_*)]}{TK\zeta} + \frac{\eta}{2\zeta}\epsilon_1^2,\end{aligned}$$

Consider $\nabla f(\tilde{\boldsymbol{\theta}}) - \mathbf{g}(\tilde{\boldsymbol{\theta}})$, let $k = 1, \ldots, K$ be the inner-loop $\tilde{\boldsymbol{\theta}}$ is selected from, we have

$$\begin{aligned}\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}}) - \mathbf{g}(\tilde{\boldsymbol{\theta}})\|^2] &\leq \mathbb{E}\left[\sum_{k=2}^{K}\frac{\phi\eta^2}{S'}\mathbb{E}[\|\mathbf{g}_{k-1}\|^2]\right] \\ &\leq \epsilon_1^2 + \mathbb{E}\left[\sum_{k=1}^{K}\frac{\phi\eta^2}{S'}\mathbb{E}\|\mathbf{g}_k\|^2\right] \\ &\leq \epsilon_1^2 + \frac{1}{T}\sum_{k=1}^{TK}\frac{\phi\eta^2}{S'}\mathbb{E}[\|\mathbf{g}_k\|^2] \\ &\leq \epsilon_1^2 + \frac{K\phi\eta^3}{2S'\zeta}\epsilon_1^2 + \frac{\phi\eta^2}{TS'\zeta}\left(\mathbb{E}[f(\bar{\boldsymbol{\theta}}_1)] - \mathbb{E}[f(\bar{\boldsymbol{\theta}}_*)]\right).\end{aligned}$$

By combining the above two inequalities, we obtain

$$\begin{aligned}\mathbb{E}[\|\nabla f(\tilde{\boldsymbol{\theta}})\|^2] &\leq \frac{2\left(\mathbb{E}[f(\bar{\boldsymbol{\theta}}_1)] - \mathbb{E}[f(\bar{\boldsymbol{\theta}}_*)]\right)}{TK\zeta} + \frac{2\phi\eta^2}{TS'\zeta}\left(\mathbb{E}[f(\bar{\boldsymbol{\theta}}_1)] - \mathbb{E}[f(\bar{\boldsymbol{\theta}}_*)]\right) + \left(\frac{\eta}{\zeta} + \frac{K\phi\eta^3}{S'\zeta} + 2\right)\epsilon_1^2 \\ &= \frac{2}{TK\zeta}\left(1 + \frac{\phi\eta^2 K}{S'}\right)\left(\mathbb{E}[f(\bar{\boldsymbol{\theta}}_1)] - \mathbb{E}[f(\bar{\boldsymbol{\theta}}_*)]\right),\end{aligned}$$

where the last equality holds as full-batch gradient descent is used for each outer-loop step, i.e., $\epsilon_1^2 = 0$. $\square$

D Algorithms

In this section, we provide a detailed description of the proposed doubly variance reduction algorithm SGCN++ and its variants: zeroth-order only SGCN++ and first-order only SGCN++.

Algorithm 3 SGCN++ (Doubly variance reduction)

Input: initial point $\bar{\boldsymbol{\theta}}_1 = \{\bar{\mathbf{W}}^{(\ell)}\}_{\ell=1}^L$, step size $\eta > 0$, control factors $\{\omega_\ell\}_{\ell=1}^L$, layer-wise sample sizes $\{S_\ell, S'_\ell\}_{\ell=1}^L$.
for $t = 1, \ldots, T$ **do**
  Set $\boldsymbol{\theta}_t \leftarrow \bar{\boldsymbol{\theta}}_t$
  for $\ell = 1, \ldots, L$ **do**
    Randomly sample batch $\mathcal{B}_\ell$ of nodes of size S_ℓ.
    $\mathbf{H}_1^{(\ell)} = f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_1^{(\ell-1)}; \mathbf{W}_1^{(\ell)})$
    $\mathbf{D}_1^{(\ell)} = \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_1^{(\ell-1)}; \mathbf{W}_1^{(\ell)})$
  end for
  Calculate gradient $\nabla f(\boldsymbol{\theta}_1) = \mathbf{D}_1^{(1)}\mathbf{D}_1^{(2)}\ldots\mathbf{D}_1^{(L)}$, then update weight $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_1 - \eta\nabla f(\boldsymbol{\theta}_1)$
  for $k = 2, \ldots, K$ **do**
    for $\ell = 1, \ldots, L$ **do**
      Randomly sample batch $\mathcal{B}_\ell$ of nodes of size S'_ℓ.
      $\mathbf{H}_k^{(\ell)} = \mathbf{H}_{k-1}^{(\ell)} + f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_k^{(\ell-1)}; \mathbf{W}_k^{(\ell)}) - f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_{k-1}^{(\ell-1)}; \mathbf{W}_{k-1}^{(\ell)})$
      $\mathbf{D}_k^{(\ell)} = \mathbf{D}_{k-1}^{(\ell)} + \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_k^{(\ell-1)}; \mathbf{W}_k^{(\ell)}) - \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_{k-1}^{(\ell-1)}; \mathbf{W}_{k-1}^{(\ell)})$
      if $\|\mathbf{H}_k^{(\ell)} - \mathbf{H}_1^{(\ell)}\| \geq \omega_\ell\|\mathbf{H}_1^{(\ell)}\|$ or $\|\mathbf{D}_k^{(\ell)} - \mathbf{D}_1^{(\ell)}\| \geq \omega_\ell\|\mathbf{D}_1^{(\ell)}\|$ **then**
        Break and start another outer-loop
      end if
    end for
  end for
  Calculate gradient $\mathbf{g}(\boldsymbol{\theta}_k) = \mathbf{D}_k^{(1)}\mathbf{D}_k^{(2)}\ldots\mathbf{D}_k^{(L)}$, then update weight
  $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\mathbf{g}(\boldsymbol{\theta}_k)$
  Set $\bar{\boldsymbol{\theta}}_{t+1} = \boldsymbol{\theta}_{K+1}$
end for
Output: $\tilde{\boldsymbol{\theta}}$ is randomly chosen from all solutions

Algorithm 4 `SGCN++` (Zeroth-order only)

Input: initial point $\bar{\boldsymbol{\theta}}_1 = \{\bar{\mathbf{W}}^{(\ell)}\}_{\ell=1}^L$, step size $\eta > 0$, control factors $\{\omega_\ell\}_{\ell=1}^L$, layer-wise sample sizes $\{S_\ell, S'_\ell\}_{\ell=1}^L$.
for $t = 1, \ldots, T$ **do**
  Set $\boldsymbol{\theta}_t \leftarrow \bar{\boldsymbol{\theta}}_t$
  for $\ell = 1, \ldots, L$ **do**
    Randomly sample batch $\mathcal{B}_\ell$ of nodes of size S_ℓ.
    $\mathbf{H}_1^{(\ell)} = f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_1^{(\ell-1)}; \mathbf{W}_1^{(\ell)})$
    $\mathbf{D}_1^{(\ell)} = \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_1^{(\ell-1)}; \mathbf{W}_1^{(\ell)})$
  end for
  Calculate gradient $\nabla f(\boldsymbol{\theta}_1) = \mathbf{D}_1^{(1)}\mathbf{D}_1^{(2)}\ldots\mathbf{D}_1^{(L)}$, then update weight $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_1 - \eta\nabla f(\boldsymbol{\theta}_1)$
  for $k = 2, \ldots, K$ **do**
    for $\ell = 1, \ldots, L$ **do**
      Randomly sample batch $\mathcal{B}_\ell$ of nodes of size S'_ℓ.
      $\mathbf{H}_k^{(\ell)} = \mathbf{H}_{k-1}^{(\ell)} + f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_k^{(\ell-1)}; \mathbf{W}_k^{(\ell)}) - f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_{k-1}^{(\ell-1)}; \mathbf{W}_{k-1}^{(\ell)})$
      $\mathbf{D}_k^{(\ell)} = \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_k^{(\ell-1)}; \mathbf{W}_k^{(\ell)})$
      if $\|\mathbf{H}_k^{(\ell)} - \mathbf{H}_1^{(\ell)}\| \geq \omega_\ell \|\mathbf{H}_1^{(\ell)}\|$ **then**
        Break and start another outer-loop
      end if
    end for
  end for
  Calculate gradient $\mathbf{g}(\boldsymbol{\theta}_k) = \mathbf{D}_k^{(1)}\mathbf{D}_k^{(2)}\ldots\mathbf{D}_k^{(L)}$, then update weight
  $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\mathbf{g}(\boldsymbol{\theta}_k)$
  Set $\bar{\boldsymbol{\theta}}_{t+1} = \boldsymbol{\theta}_{K+1}$
end for
Output: $\tilde{\boldsymbol{\theta}}$ is randomly chosen from all solutions

Algorithm 5 `SGCN++` (First-order only)

Input: initial point $\bar{\boldsymbol{\theta}}_1 = \{\bar{\mathbf{W}}^{(\ell)}\}_{\ell=1}^{L}$, step size $\eta > 0$, control factors $\{\omega_\ell\}_{\ell=1}^{L}$, layer-wise sample sizes $\{S_\ell, S'_\ell\}_{\ell=1}^{L}$.
for $t = 1, \ldots, T$ **do**
  Set $\boldsymbol{\theta}_t \leftarrow \bar{\boldsymbol{\theta}}_t$
  for $\ell = 1, \ldots, L$ **do**
    Randomly sample batch $\mathcal{B}_\ell$ of nodes of size S_ℓ.
    $\mathbf{H}_1^{(\ell)} = f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_1^{(\ell-1)}; \mathbf{W}_1^{(\ell)})$
    $\mathbf{D}_1^{(\ell)} = \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_1^{(\ell-1)}; \mathbf{W}_1^{(\ell)})$
  end for
  Calculate gradient $\nabla f(\boldsymbol{\theta}_1) = \mathbf{D}_1^{(1)}\mathbf{D}_1^{(2)}\ldots\mathbf{D}_1^{(L)}$, then update weight $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_1 - \eta\nabla f(\boldsymbol{\theta}_1)$
  for $k = 2, \ldots, K$ **do**
    for $\ell = 1, \ldots, L$ **do**
      Randomly sample batch $\mathcal{B}_\ell$ of nodes of size S'_ℓ.
      $\mathbf{H}_k^{(\ell)} = f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_k^{(\ell-1)}; \mathbf{W}_k^{(\ell)})$
      $\mathbf{D}_k^{(\ell)} = \mathbf{D}_{k-1}^{(\ell)} + \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_k^{(\ell-1)}; \mathbf{W}_k^{(\ell)}) - \nabla f_{\mathcal{B}_\ell}^{(\ell)}(\mathbf{H}_{k-1}^{(\ell-1)}; \mathbf{W}_{k-1}^{(\ell)})$
      if $\|\mathbf{D}_k^{(\ell)} - \mathbf{D}_1^{(\ell)}\| \geq \omega_\ell \|\mathbf{D}_1^{(\ell)}\|$ **then**
        Break and start another outer-loop
      end if
    end for
  end for
  Calculate gradient $\mathbf{g}(\boldsymbol{\theta}_k) = \mathbf{D}_k^{(1)}\mathbf{D}_k^{(2)}\ldots\mathbf{D}_k^{(L)}$, then update weight
  $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\mathbf{g}(\boldsymbol{\theta}_k)$
  Set $\bar{\boldsymbol{\theta}}_{t+1} = \boldsymbol{\theta}_{K+1}$
end for
Output: $\tilde{\boldsymbol{\theta}}$ is randomly chosen from all solutions

E Experimental Details

Hardware specification and environment We run our experiments on a single machine with Intel i5-7500, one NVIDIA GTX1080 GPU (8GB memory) and 32GB RAM memory. The code is written in Python 3.7. We use PyTorch 1.4 on CUDA 10.1 to train the model on GPU. During each epoch, we randomly construct 10 mini-batches in parallel using in Python multiprocessing package.

Implementation details To demonstrate the effectiveness of doubly variance reduction, we modified the PyTorch implementation of [1]GCN [1] to add `LADIES` [10], `FastGCN` [8], `GraphSage` [5] and `Exact` sampling mechanism. Then, we implement `SGCN++` on the top of each sampling method to illustrate how doubly variance reduction help for GCN training. Other than doubly variance reduction, we introduce zeroth-order only and first-order only variance reduction as baselines.

By default, we train 2-layer GCNs with hidden state dimension as 256, and use element-wise ELU as the activation function

$$\texttt{ELU}(x) = \max(0, x) + \min(0, \exp(x) - 1),$$

and use use row normalized Laplacian matrix $\mathbf{L} = \mathbf{D}^{-1}\mathbf{A}$ for GraphSage and symmetric normalized Laplacian matrix $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ for others. The detailed graph convolutional network structure is defined as

$$\begin{aligned}
&\texttt{ELU}(\texttt{GraphConvolution}(\texttt{nfeat}, \texttt{nhid} = 256)) \\
&\texttt{ELU}(\texttt{GraphConvolution}(\texttt{nhid} = 256, \texttt{nhid} = 256)) \\
&\texttt{Linear}(\texttt{nhid} = 256, \texttt{nclasses})
\end{aligned}$$

We update the model with a mini-batch size of 512 and Adam optimizer with a learning rate of 0.01. We choose 5 neighbors to be sampled for GraphSage sampling, 512 nodes to be sampled for FastGCN and LADIES sampling. The dropout ratio is set to zero for all experiments.

For `SGCN++`, historical node embeddings are first calculated on GPUs and transit to CPU memory using PyTorch command `Tensor.to(device)`. Therefore, no extra GPU memory usage are required when training with `SGCN++`. To balance the staleness of snapshot model and the computational efficiency, we choose inner-loop steps $K = 10$ and early step variable $\omega = 2e - 3$.

During training, for each epoch ($t = 1, \ldots, T$), we construct 10 mini-batches of size 512 in parallel using Python package `multiprocessing` and perform training on the sampled 10 mini-batches ($k = 1, \ldots, K$). To achieves a fair comparison of different sampling strategies in terms of sampling complexity, we implement all sampling algorithms using `numpy.random` and `scipy.sparse` package.

To compare the convergence speed of vanilla `SGCN`s with `SGCN++`, we plot three types of curves: "Loss / Step", "Loss / Epoch" and "MSE of gradient / Step".

- "Loss / Step" is the loss calculated after each iteration.
- "Loss / Epoch" is the average of loss calculated in the epoch.
- "MSE of gradient / Step" is the mean-square error of stochastic gradient after each iteration, where mean-square error is calculated as the square of ℓ_2 norm of the difference between stochastic gradient and full-batch gradient, and the full-batch gradient is calculated using the full Laplacian matrix and all the nodes.

[1] https://github.com/tkipf/pygcn

F Additional Experiments

In this section, we first highlight the difference between node-wise sampling, layer-wise sampling, and subgraph sampling algorithms, then demonstrate the effectiveness of zeroth-order, first-order, and doubly variance reduction when employed in each of these sampling strategies.

Node-wide sampling The main idea of node-wise sampling is to first sample all the nodes needed for the computation using neighbor sampling (NS), then train the GCN based on the sampled nodes. For each node in the ℓth GCN layer, NS randomly samples s of its neighbors at the $(\ell-1)$th GCN layer and formulate $\mathbf{L}^{(\ell)}$ by

$$L_{i,j}^{(\ell)} = \begin{cases} \frac{|\mathcal{N}(i)|}{s} \times L_{i,j}, & \text{if } j \in \widehat{\mathcal{N}}^{(\ell)}(i) \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

where $\mathcal{N}(i)$ is the full set of ith node neighbor, $\widehat{\mathcal{N}}^{(\ell)}(i)$ is the sampled neighbors of node i for ℓth GCN layer. GraphSage [5] follows the spirit of node-wise sampling. GraphSage [5] performs uniform node sampling on the previous layer neighbors for a fixed number of nodes to bound the mini-batch computation complexity.

Layer-wise sampling To avoid the neighbor explosion issue, layer-wise sampling is introduced that controls the size of sampled neighborhoods in each layer. For the ℓth GCN layer, layer-wise sampling methods sample a set of nodes $\mathcal{V}_\ell \subseteq \mathcal{V}$ of size s under the distribution $\boldsymbol{p}$ to approximate the Laplacian by

$$L_{i,j}^{(\ell)} = \begin{cases} \frac{1}{s \times p_j} \times L_{i,j}, & \text{if } j \in \mathcal{V}_\ell \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

Existing work FastGCN [8] and LADIES [10] follows the spirit of layer-wise sampling. FastGCN [8] perform independently node sampling for each layer. FastGCN [8] applies important sampling to reduce variance and results in a constant sample size in all layers. However, mini-batches potentially become too sparse to achieve high accuracy. LADIES [10] improves [8] by layer-dependent sampling. Based on the sampled nodes in the upper layer, they select their neighborhood nodes, construct a bipartite subgraph and computes the importance probability accordingly. Then, they sample a fixed number of nodes by the calculated probability and recursively conducts such a procedure per layer to construct the whole computation graph.

Subgraph sampling Subgraph sampling is similar to layer-wise sampling by restricting $\mathbf{L}^{(1)} = \mathbf{L}^{(2)} = \ldots = \mathbf{L}^{(L)}$. For example, GraphSaint [12] can be viewed as a special case of layer-wise sampling algorithm FastGCN by restricting the nodes sampled at the 1-st to $(L-1)$th layer the same as the nodes sampled at the Lth layer. However, GraphSaint requires a significant large mini-batch size compared to other layer-wise sampling methods. We leave this as a potential future direction to explore.

Effectiveness of zeroth-order, first-order, and doubly variance reduction To emphasize the importance of applying doubly variance reduction comparing to zeroth-order only and first-order only variance reduction, we compare the training loss, testing loss, mean-square error of stochastic gradient of applying zeroth-order only, first-order only and doubly variance reduction. Details please refer to Figure 3, Figure 4, Figure 5, and Figure 6, respectively.

We observe that doubly variance reduction can always guarantee a significant performance improvement in terms of convergence speed and accuracy, while zeroth-order only variance reduction has effect only on `SGCN`s with large function approximation variance, e.g., FastGCN. Besides, first-order only variance might worsen the performance when the function approximation variance is large, e.g., FastGCN.

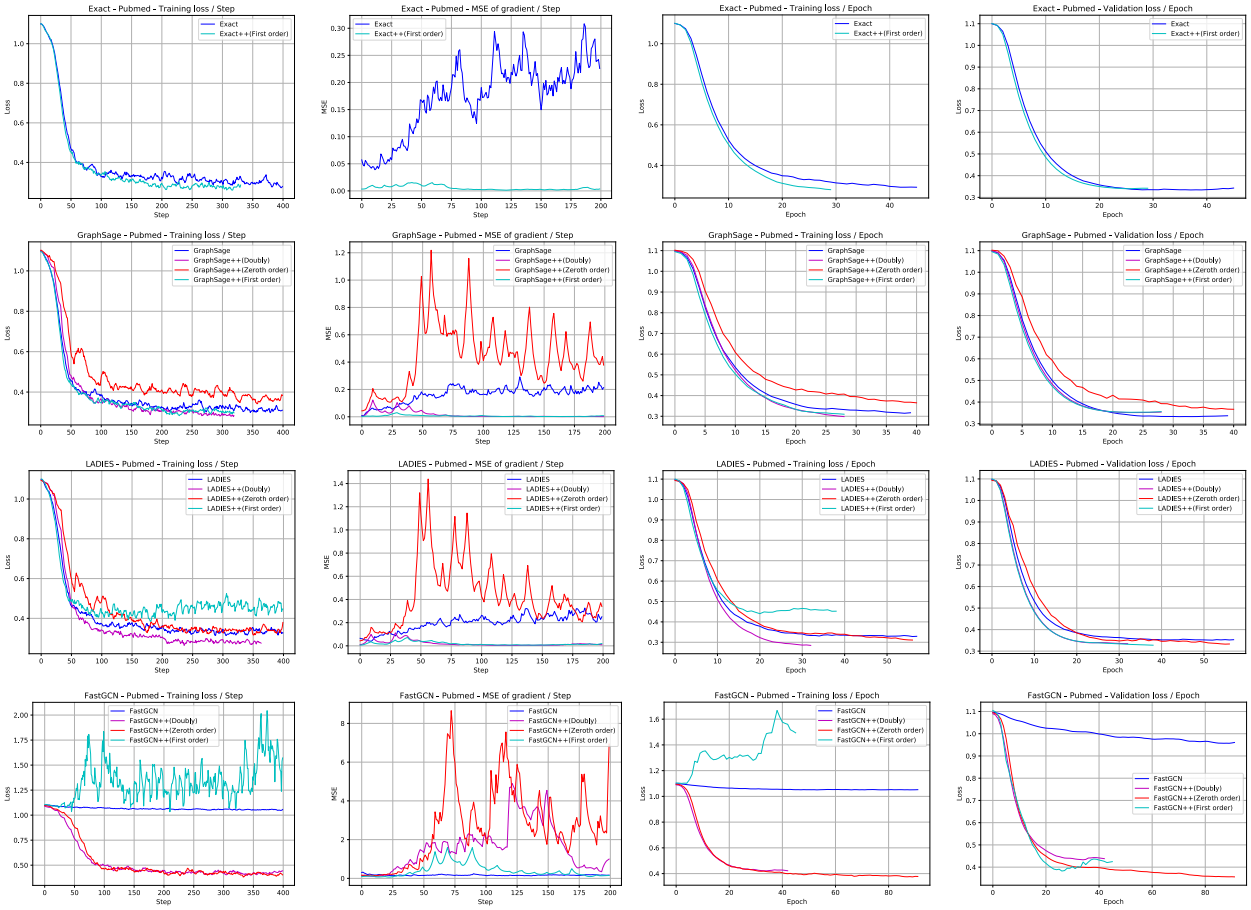

Figure 3: Comparison of training loss, testing loss, and MSE of gradient of zeroth-order only, first order only, and doubly variance reduction on `Pubmed` dataset

GPU memory usage We compare the GPU memory usage of `SGCN` and `SGCN++` in Figure 7. Where *memory allocated* is calculated by `torch.cuda.memory_allocated`, which is defined as the current GPU memory occupied by tensors in bytes for a given device and *max memory allocated* is calculated by `torch.cuda.max_memory_allocated`, which is defined as the maximum GPU memory occupied by tensors in bytes for a given device. This experiment is designed to show that neither running full-batch GCN nor saving historical activations (the output of each node after activation function) will significantly increase the computation overhead during training.

From Figure 7, we observe that since all historical activations are stored outside GPU, `SGCN++` only requires several megabytes to transit data between GPU memory to hardware stores, which can be ignored compared to the memory usage of calculation itself. On the other hand, we can conclude that the bottleneck of training sampling based GCNs is sampling rather than node embedding aggregation. Therefore, a promising future direction is developing a provable sampling algorithm with low sampling complexity.

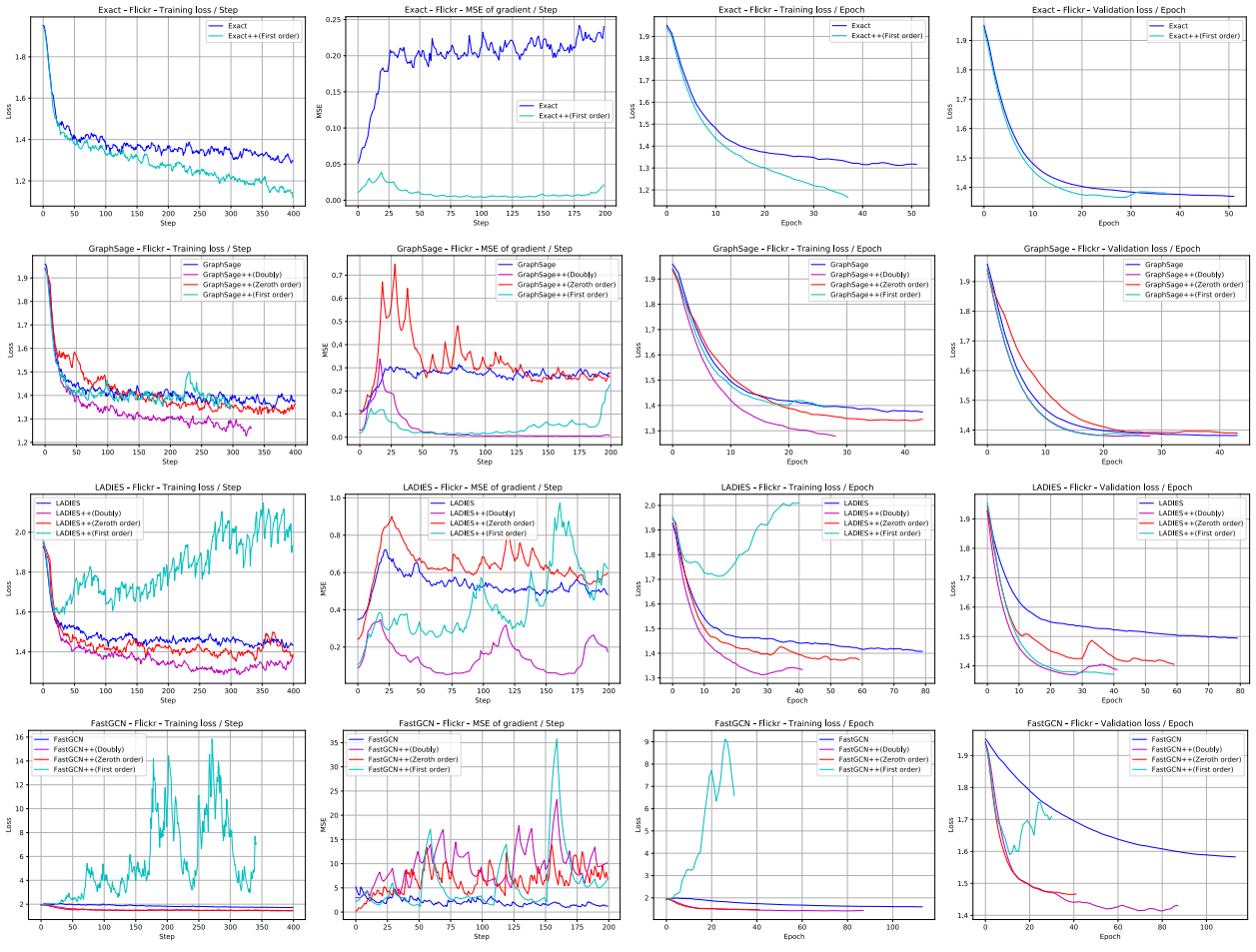

Figure 4: Comparison of training loss, testing loss, and MSE of gradient of zeroth-order only, first order only, and doubly variance reduction on `Flickr` dataset

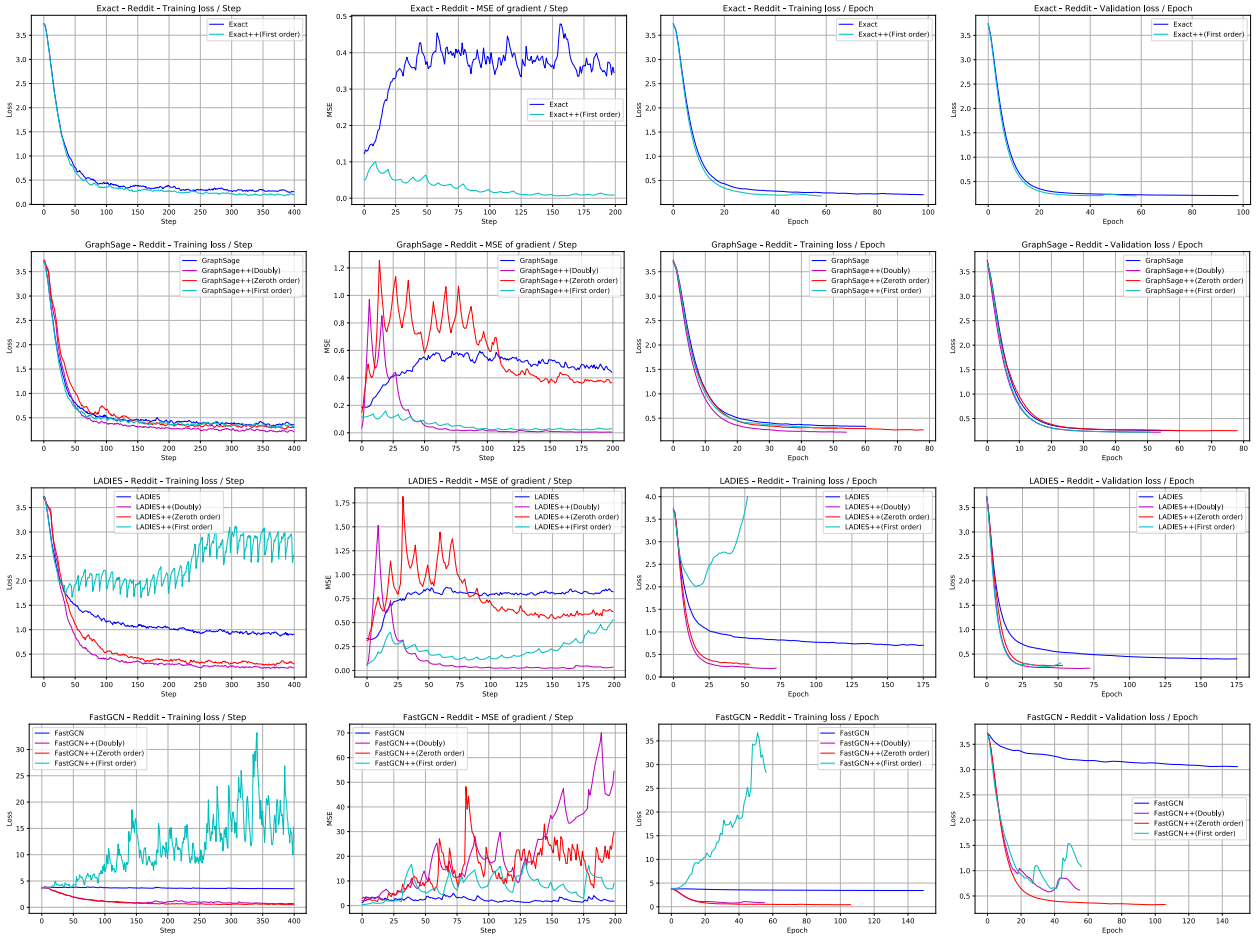

Figure 5: Comparison of training loss, testing loss, and MSE of gradient of zeroth-order only, first order only, and doubly variance reduction on `Reddit` dataset

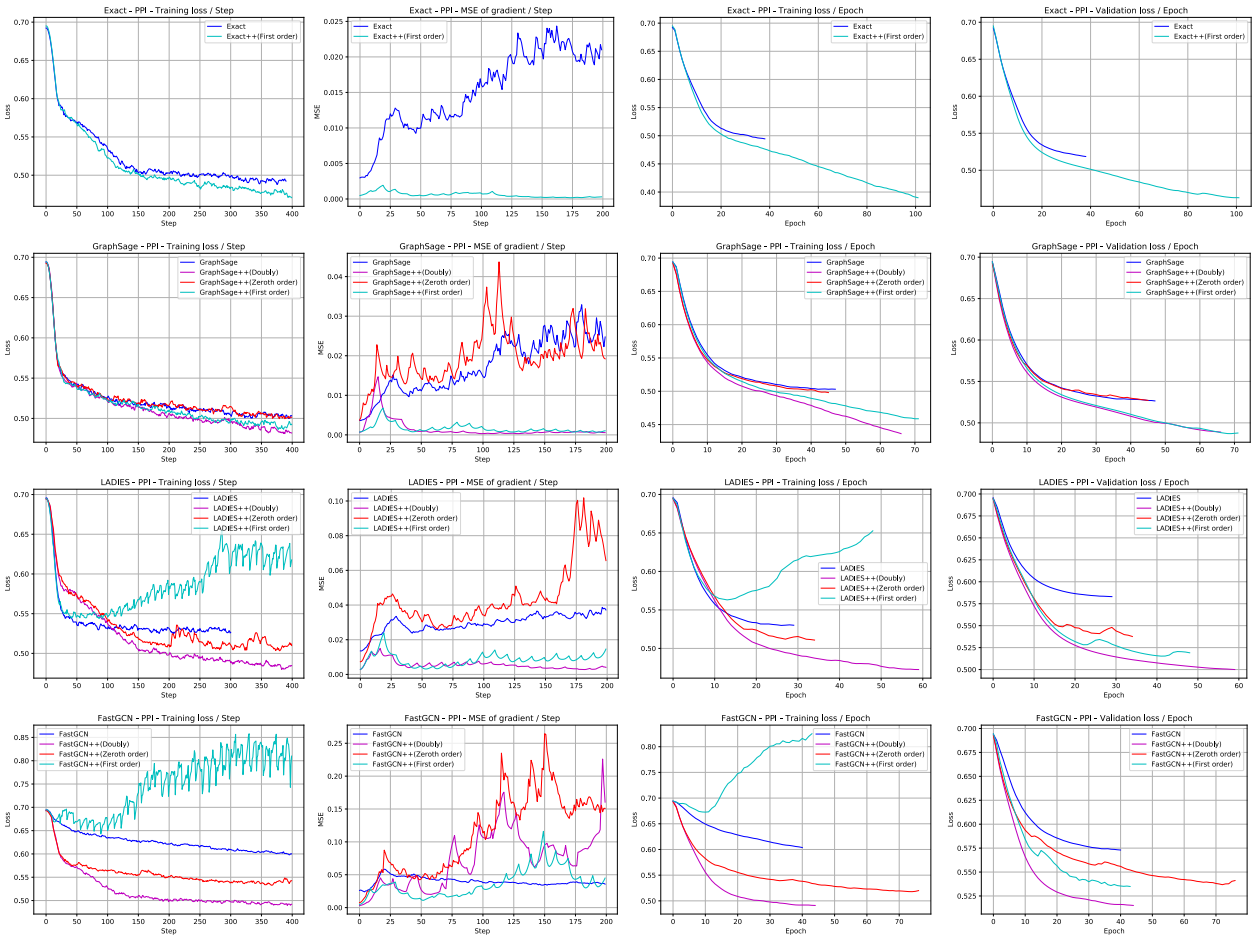

Figure 6: Comparison of training loss, testing loss, and MSE of gradient of zeroth-order only, first order only, and doubly variance reduction on `PPI` dataset

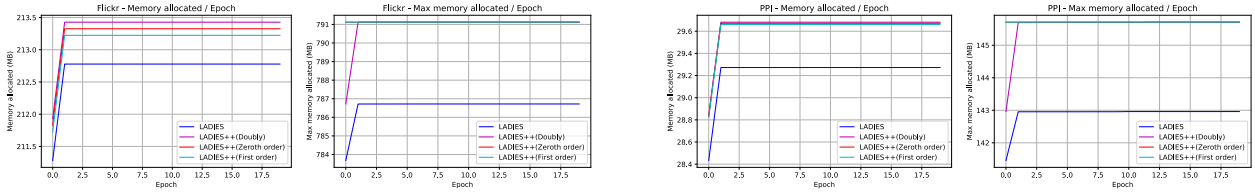

Figure 7: Comparison of GPU memory usage of `SGCN` and `SGCN++` on Flickr and PPI dataset.

G Reproducing Experiment Results

To reproduce the results reported in the paper, we create a GitHub repository https://github.com/CongWeilin/SGCN, where we provide the bash script to reproduce the experiment results and a jupyter notebook file for a quick visualization and GPU utilization calculation. It is worth noting that due to the existence of randomness, the obtained results (e.g., loss curve) may be slightly different. However, it is not difficult to find that the overall trend of loss curves and conclusions will remains the same.

This implementation is based on [2]PyTorch using Python 3. We notice that Python 2 might results in a wrong gradient update, even for vanilla `SGCN`s.

Install dependencies:

```
# create virtual environment
$ virtualenv env
$ source env/bin/activate
# install dependencies
$ pip install -r requirements.txt
```

Experiments are produced on Pubmed, Flickr, Reddit, and PPi datasets. The utilized datasets can be downloaded from [3]Google drive.

```
# create folders that save experiment results and datasets
$ mkdir ./results
$ mkdir ./data # please download the dataset and put them inside this folder
```

To reproduce the results, please run the following commands:

```
# train LADIES
$ python train_sgcn.py --dataset ppi --sample_method ladies
$ python train_sgcn.py --dataset flickr --sample_method ladies
$ python train_sgcn.py --dataset pubmed --sample_method ladies
$ python train_sgcn.py --dataset reddit --sample_method ladies

# train GraphSage
$ python train_sgcn.py --dataset ppi --sample_method graphsage
$ python train_sgcn.py --dataset flickr --sample_method graphsage
$ python train_sgcn.py --dataset pubmed --sample_method graphsage
$ python train_sgcn.py --dataset reddit --sample_method graphsage

# train FastGCN
$ python train_sgcn.py --dataset ppi --sample_method fastgcn
$ python train_sgcn.py --dataset flickr --sample_method fastgcn
$ python train_sgcn.py --dataset pubmed --sample_method fastgcn
$ python train_sgcn.py --dataset reddit --sample_method fastgcn

# train Exact
$ python train_sgcn.py --dataset ppi --sample_method exact
$ python train_sgcn.py --dataset flickr --sample_method exact
$ python train_sgcn.py --dataset pubmed --sample_method exact
$ python train_sgcn.py --dataset reddit --sample_method exact
```

[2] https://pytorch.org/
[3] https://drive.google.com/drive/folders/1wW8JwNkPbXZuv1gD4E3BAQ3CHBdqtEqm?usp=sharing