Stochastic Training of Graph Convolutional Networks with Variance Reduction

Jianfei Chen [1] **Jun Zhu** [1] **Le Song** [2 3]

Abstract

Graph convolutional networks (GCNs) are powerful deep neural networks for graph-structured data. However, GCN computes the representation of a node recursively from its neighbors, making the receptive field size grow exponentially with the number of layers. Previous attempts on reducing the receptive field size by subsampling neighbors do not have convergence guarantee, and their receptive field size per node is still in the order of hundreds. In this paper, we develop control variate based algorithms with new theoretical guarantee to converge to a local optimum of GCN regardless of the neighbor sampling size. Empirical results show that our algorithms enjoy similar convergence rate and model quality with the exact algorithm using only two neighbors per node. The running time of our algorithms on a large Reddit dataset is only one seventh of previous neighbor sampling algorithms.

1. Introduction

Graph convolution networks (GCNs) (Kipf & Welling, 2017) generalize convolutional neural networks (CNNs) (LeCun et al., 1995) to graph structured data. The "graph convolution" operation applies same linear transformation to all the neighbors of a node, followed by mean pooling and nonlinearity. By stacking multiple graph convolution layers, GCNs can learn node representations by utilizing information from distant neighbors. GCNs and their variants (Hamilton et al., 2017a; Veličković et al., 2018) have been applied to semi-supervised node classification (Kipf & Welling, 2017), inductive node embedding (Hamilton et al., 2017a), link prediction (Kipf & Welling, 2016; Berg et al., 2017) and knowledge graphs (Schlichtkrull et al., 2017), outperforming multi-layer perceptron (MLP) models that do not use the graph structure, and graph embedding approaches (Perozzi et al., 2014; Tang et al., 2015; Grover & Leskovec, 2016) that do not use node features.

However, the graph convolution operation makes GCNs difficult to be trained efficiently. The representation of a node at layer L is computed recursively by the representations of all its neighbors at layer $L-1$. Therefore, the receptive field of a single node grows exponentially with respect to the number of layers, as illustrated in Fig. 1(a), so exactly computing the stochastic gradient is expensive even for a single node. Due to the large receptive field size, Kipf & Welling (2017) propose to train GCN by a batch algorithm, which computes the representations of all the nodes altogether. However, batch algorithms cannot handle large-scale datasets because of their slow convergence and the requirement to fit the entire dataset in GPU memory.

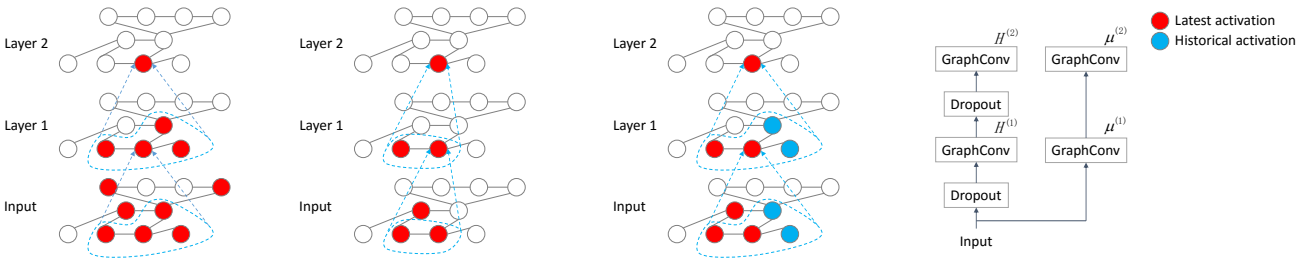
Hamilton et al. (2017a) make an initial attempt to develop stochastic training algorithms for GCNs via a scheme of neighbor sampling (NS). Instead of considering all the neighbors, they randomly subsample $D^{(l)}$ neighbors at the l-th layer. Therefore, they reduce the receptive field size to $\prod_l D^{(l)}$, as shown in Fig. 1(b). They find that for two-layer GCNs, keeping $D^{(1)} = 10$ and $D^{(2)} = 25$ neighbors can achieve comparable performance with the original model. However, there is no theoretical guarantee on the convergence of the stochastic training algorithm with NS. Moreover, the time complexity of NS is still $D^{(1)}D^{(2)} = 250$ times larger than training an MLP, which is unsatisfactory.

In this paper, we develop novel control variate-based stochastic approximation algorithms for GCN by utilizing the historical activations of nodes as a control variate. Our algorithms have new theoretical results on (1) variance reduction from the magnitude of the activation to the magnitude of the difference between current-and-historical activations; (2) exact (zero-variance) predictions at testing time; (3) convergence to a local optimum of GCN during training *regardless of the neighbor sampling size* $D^{(l)}$, with an asymptotically unbiased stochastic gradient. The theoretical properties allow us to significantly reduce the time complexity of stochastic training by sampling only $D^{(l)} = 2$ neighbors per node, yet still retain the quality of the model.

We empirically test our algorithms on six graph datasets, and the results match with the theory. Comparing with NS, our

[1]Dept. of Comp. Sci. & Tech., BNRist Center, State Key Lab for Intell. Tech. & Sys., THBI Lab, Tsinghua University, Beijing, 100084, China [2]Georgia Institute of Technology [3]Ant Financial. Correspondence to: Jun Zhu <dcszj@mail.tsinghua.edu.cn>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.



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

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

Dataset	V	E	Degree	Degree 2
Citeseer	3,327	12,431	4	15
Cora	2,708	13,264	5	37
PubMed	19,717	108,365	6	60
NELL	65,755	318,135	5	1,597
PPI	14,755	458,973	31	970
Reddit	232,965	23,446,803	101	10,858

Table 1. Number of vertexes, edges, and average number of 1-hop and 2-hop neighbors per node for each dataset. Undirected edges are counted twice and self-loops are counted once.

algorithms significantly reduce the bias and variance of the gradient. Comparing with the exact algorithm which considers all the neighbors, our algorithms with only $D^{(l)} = 2$ neighbors still get the same accuracy at testing time, and achieve similar predictive performance during training *in a comparable number of epochs*, with a much lower time complexity, while these results are not achievable by NS. On the largest Reddit dataset, the training time of our algorithm is 7 times shorter than that of the best-performing competitor among exact, neighbor sampling and importance sampling (Chen et al., 2018) algorithms.

2. Backgrounds

We briefly review graph convolutional networks (GCNs), stochastic training, neighbor sampling, and importance sampling in this section.

2.1. Graph Convolutional Networks

We present our algorithm with a GCN for semi-supervised node classification (Kipf & Welling, 2017). However, the algorithm is neither limited to the task nor the model. Our algorithm is applicable to other models including GraphSAGE-mean (Hamilton et al., 2017a) and graph attention networks (GAT) (Veličković et al., 2018), and other tasks (Kipf & Welling, 2016; Berg et al., 2017; Schlichtkrull et al., 2017; Hamilton et al., 2017b), as long as the model aggregates neighbor activations by averaging.

In the node classification task, we have an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $V = |\mathcal{V}|$ vertices and $E = |\mathcal{E}|$ edges, where each vertex v consists of a feature vector x_v and a label y_v. We observe the labels for some vertices $\mathcal{V}_\mathcal{L}$. The goal is to predict the labels for the rest vertices $\mathcal{V}_\mathcal{U} := \mathcal{V} \backslash \mathcal{V}_\mathcal{L}$. The edges are represented as a symmetric $V \times V$ adjacency matrix A, where A_{uv} is the weight of the edge between u and v, and the propagation matrix P is a normalized version of A: $\tilde{A} = A + I$, $\tilde{D}_{uu} = \sum_v \tilde{A}_{uv}$, and $P = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$. A graph convolution layer is defined as

$$Z^{(l+1)} = PH^{(l)}W^{(l)}, \quad H^{(l+1)} = \sigma(Z^{l+1}), \qquad (1)$$

where $H^{(l)}$ is the activation matrix in the l-th layer, whose each row is the activation of a graph node. $H^{(0)} = X$ is the input feature matrix, $W^{(l)}$ is a trainable weight matrix, and $\sigma(\cdot)$ is an activation function. Denote $|\cdot|$ as the cardinality of a set. The training loss is defined as

$$\mathcal{L} = \frac{1}{|\mathcal{V}_\mathcal{L}|} \sum_{v \in \mathcal{V}_\mathcal{L}} f(y_v, z_v^{(L)}), \qquad (2)$$

where $f(\cdot, \cdot)$ is a loss function. A graph convolution layer propagates information to nodes from their neighbors by computing the neighbor averaging $PH^{(l)}$. Let $\mathbf{n}(u)$ be the set of neighbors of node u, and $n(u)$ be its cardinality. The neighbor averaging of node u, $(PH^{(l)})_u = \sum_{v=1}^{V} P_{uv} h_v^{(l)} = \sum_{v \in \mathbf{n}(u)} P_{uv} h_v^{(l)}$, is a weighted sum of neighbors' activations. Then, a fully-connected layer is applied on all the nodes, with a shared weight matrix $W^{(l)}$ across all the nodes.

We denote the *receptive field* of a node u as all the activations $h_v^{(l)}$ on layer l needed for computing $z_u^{(L)}$. If the layer l is not explicitly mentioned, it is the input layer 0. Intuitively, the receptive field of node u is just all its L-hop neighbors, i.e., nodes that are reachable from u within L hops, as illustrated in Fig. 1(a). When $P = I$, GCN reduces to a multi-layer perceptron (MLP) model which does not use the graph structure. For MLP, the receptive field of a node u is just the node itself.

2.2. Stochastic Training

It is generally expensive to compute the batch gradient $\nabla \mathcal{L} = \frac{1}{|\mathcal{V}_\mathcal{L}|} \sum_{v \in \mathcal{V}_\mathcal{L}} \nabla f(y_v, z_v^{(L)})$, which involves iterat-

ing over the entire labeled set of nodes. A possible solution is to approximate the batch gradient by a stochastic gradient

$$\frac{1}{|\mathcal{V}_\mathcal{B}|}\sum_{v\in\mathcal{V}_\mathcal{B}}\nabla f(y_v, z_v^{(L)}), \tag{3}$$

where $\mathcal{V}_\mathcal{B}\subset\mathcal{V}_\mathcal{L}$ is a minibatch of labeled nodes. However, this gradient is still expensive to compute, due to the large receptive field size. For instance, as shown in Table 1, the number of 2-hop neighbors on the NELL dataset is averagely 1,597, which means in a 2-layer GCN, computing the gradient even for a single node needs $1,597/65,755\approx 2.4\%$ nodes of the entire graph.

In subsequent sections, two other stochasticity will be introduced besides the random selection of the minibatch: the random sampling of neighbors (Sec. 2.3) and the random dropout of features (Sec. 5).

2.3. Neighbor Sampling

To reduce the receptive field size, Hamilton et al. (2017a) propose a neighbor sampling (NS) algorithm. NS randomly chooses $D^{(l)}$ neighbors for each node at layer l and develops an estimator $\text{NS}_u^{(l)}$ of $(PH^{(l)})_u$ based on Monte-Carlo approximation:

$$(PH^{(l)})_u\approx \text{NS}_u^{(l)}:=\frac{n(u)}{D^{(l)}}\sum_{v\in\hat{\mathbf{n}}^{(l)}(u)}P_{uv}h_v^{(l)},$$

where $\hat{\mathbf{n}}^{(l)}(u)\subset\mathbf{n}(u)$ is a subset of $D^{(l)}$ random neighbors. Therefore, NS reduces the receptive field size from all the L-hop neighbors to the number of sampled neighbors, $\prod_{l=1}^L D^{(l)}$. We refer $\text{NS}_u^{(l)}$ as the NS estimator of $(PH^{(l)})_u$, and $(PH^{(l)})_u$ itself as the exact estimator.

Neighbor sampling can also be written in a matrix form as

$$Z^{(l+1)}=\hat{P}^{(l)}H^{(l)}W^{(l)},\quad H^{(l+1)}=\sigma(Z^{(l+1)}), \tag{4}$$

where the propagation matrix P is replaced by a sparser unbiased estimator $\hat{P}^{(l)}$, i.e., $\mathbb{E}\hat{P}^{(l)}=P$, where $\hat{P}_{uv}^{(l)}=\frac{n(u)}{D^{(l)}}P_{uv}$ if $v\in\hat{\mathbf{n}}^{(l)}(u)$, and $\hat{P}_{uv}^{(l)}=0$ otherwise. Hamilton et al. (2017a) propose to perform an approximate forward propagation as Eq. (4), and do stochastic gradient descent (SGD) with the auto-differentiation gradient. The approximated gradient has two sources of randomness: the random selection of minibatch $\mathcal{V}_\mathcal{B}\subset\mathcal{V}_\mathcal{L}$, and the random selection of neighbors.

Though $\hat{P}^{(l)}$ is an unbiased estimator of P, $\sigma(\hat{P}^{(l)}H^{(l)}W^{(l)})$ is *not* an unbiased estimator of $\sigma(PH^{(l)}W^{(l)})$, due to the non-linearity of $\sigma(\cdot)$. In the sequel, both the prediction $Z^{(L)}$ and gradient $\nabla f(y_v, z_v^{(L)})$ obtained by NS are biased, and the convergence of SGD is not guaranteed, unless the sample size $D^{(l)}$ goes to infinity. Because of the biased gradient, the sample size $D^{(l)}$ needs to be large for NS, to keep comparable predictive performance with the exact algorithm. Hamilton et al. (2017a) choose $D^{(1)}=10$ and $D^{(2)}=25$, and the receptive field size $D^{(1)}\times D^{(2)}=250$ is much larger than one, so the training is still expensive.

2.4. Importance Sampling

FastGCN (Chen et al., 2018) is another sampling-based algorithm similar as NS. Instead of sampling neighbors for each node, FastGCN directly subsample the receptive field for each layer altogether. Formally, it approximates $(PH^{(l)})_u$ with S samples $v_1,\dots,v_S\in\mathcal{V}$ as

$$(PH^{(l)})_u=V\sum_{v=1}^V\frac{1}{V}P_{uv}h_v^{(l)}\approx\frac{V}{S}\sum_{v_s\sim q(v)}P_{uv}h_{v_s}^{(l)}/q(v_s),$$

where they define the importance distribution $q(v)\propto\sum_{u=1}^V P_{uv}^2$. According to the definition of P in Sec. 2.1, we have $q(v)\propto\frac{1}{n(v)}\sum_{(u,v)\in\mathcal{E}}\frac{1}{n(u)}$. We refer to this estimator as importance sampling (IS). Chen et al. (2018) show that IS performs better than using a uniform sample distribution $q(v)\propto 1$. NS can be viewed as an IS estimator with the importance distribution $q(v)\propto\sum_{(u,v)\in\mathcal{E}}\frac{1}{n(u)}$, because each node u has probability $\frac{1}{n(u)}$ to choose the neighbor v. Though IS may have a smaller variance than NS, it still only guarantees the convergence as the sample size S goes to infinity. Empirically, we find IS to work even *worse* than NS because sometimes it can select many neighbors for one node, and no neighbor for another, in which case the activation of the latter node is just meaningless zero.

3. Control Variate Based Algorithm

We present a novel control variate based algorithm that utilizes historical activations to reduce the estimator variance.

3.1. Control Variate Based Estimator

While computing the neighbor average $\sum_{v\in\mathbf{n}(u)}P_{uv}h_v^{(l)}$, we cannot afford to evaluate all the $h_v^{(l)}$ terms because they need to be computed recursively, i.e., we again need the activations $h_w^{(l-1)}$ of all of v's neighbors w.

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

$$\begin{aligned}(PH^{(l)})_u&=\sum_{v\in\mathbf{n}(u)}P_{uv}\Delta h_v^{(l)}+\sum_{v\in\mathbf{n}(u)}P_{uv}\bar{h}_v^{(l)}\approx\text{CV}_u^{(l)}\\&:=\frac{n(u)}{D^{(l)}}\sum_{v\in\hat{\mathbf{n}}^{(l)}(u)}P_{uv}\Delta h_v^{(l)}+\sum_{v\in\mathbf{n}(u)}P_{uv}\bar{h}_v^{(l)},\end{aligned} \tag{5}$$

where we represent $h_v^{(l)}$ as the sum of $\Delta h_v^{(l)}$ and $\bar{h}_v^{(l)}$, and

we only apply Monte-Carlo approximation on the $\Delta h_v^{(l)}$ term. Averaging over all the $\bar{h}_v^{(l)}$ terms is still affordable because they do not need to be computed recursively. Since we expect $h_v^{(l)}$ and $\bar{h}_v^{(l)}$ to be close, Δh_v will be small and $\text{CV}_u^{(l)}$ should have a smaller variance than $\text{NS}_u^{(l)}$. Particularly, if the model weight is kept fixed, $\bar{h}_v^{(l)}$ should eventually equal with $h_v^{(l)}$, so that $\text{CV}_u^{(l)} = 0 + \sum_{v\in\mathbf{n}(u)} P_{uv}\bar{h}_v^{(l)} = \sum_{v\in\mathbf{n}(u)} P_{uv}h_v^{(l)} = (PH^{(l)})_u$, i.e., the estimator has zero variance. This estimator is referred as CV. We will compare the variance of NS and CV estimators in Sec. 3.2 and show that the variance of CV will be eventually zero during the training in Sec. 4. The term $\text{CV}_u^{(l)} - \text{NS}_u^{(l)} = \sum_{v\in\mathbf{n}(u)} P_{uv}\bar{h}_u^{(l)} - \frac{n(u)}{D^{(l)}}\sum_{v\in\hat{\mathbf{n}}^{(l)}(u)} P_{uv}\bar{h}_u^{(l)}$ is a *control variate* (Ripley, 2009, Chapter 5) added to the neighbor sampling estimator $\text{NS}_u^{(l)}$, to reduce its variance.

In matrix form, let $\bar{H}^{(l)}$ be the matrix formed by stacking $\bar{h}_v^{(l)}$, then CV can be written as

$$Z^{(l+1)} = \left(\hat{P}^{(l)}(H^{(l)} - \bar{H}^{(l)}) + P\bar{H}^{(l)}\right)W^{(l)}. \quad (6)$$

3.2. Variance Analysis

We analyze the variance of the estimators assuming all the features are 1-dimensional. The analysis can be extended to multiple dimensions by treating each dimension separately. We further assume that $\hat{\mathbf{n}}^{(l)}(u)$ is created by sampling $D^{(l)}$ neighbors without replacement from $\mathbf{n}(u)$. The following proposition is proven in Appendix A:

Proposition 1. *If $\hat{\mathbf{n}}^{(l)}(u)$ contains $D^{(l)}$ samples from $\mathbf{n}(u)$ without replacement, then $\text{Var}_{\hat{\mathbf{n}}^{(l)}(u)}\left[\frac{n(u)}{D^{(l)}}\sum_{v\in\hat{\mathbf{n}}^{(l)}(u)} x_v\right] = \frac{C_u^{(l)}}{2D^{(l)}}\sum_{v_1\in\mathbf{n}(u)}\sum_{v_2\in\mathbf{n}(u)}(x_{v_1}-x_{v_2})^2$, where $C_u^{(l)} = 1 - (D^{(l)}-1)/(n(u)-1)$.*

By Proposition 1, we have $\text{Var}_{\hat{\mathbf{n}}^{(l)}(u)}\left[NS_u^{(l)}\right] = \frac{C_u^{(l)}}{2D^{(l)}}\sum_{v_1\in\mathbf{n}(u)}\sum_{v_2\in\mathbf{n}(u)}(P_{uv_1}h_{v_1}^{(l)} - P_{uv_2}h_{v_2}^{(l)})^2$, in contrast, the variance of the CV estimator is $\text{Var}_{\hat{\mathbf{n}}^{(l)}(u)}\left[CV_u^{(l)}\right] = \frac{C_u^{(l)}}{2D^{(l)}}\sum_{v_1\in\mathbf{n}(u)}\sum_{v_2\in\mathbf{n}(u)}(P_{uv_1}\Delta h_{v_1}^{(l)} - P_{uv_2}\Delta h_{v_2}^{(l)})^2$, which replaces $h_v^{(l)}$ by $\Delta h_v^{(l)}$. Since $\Delta h_v^{(l)}$ is usually much smaller than $h_v^{(l)}$, the CV estimator enjoys much smaller variance than the NS estimator. Furthermore, as we will show in Sec. 4.2, $\Delta h_v^{(l)}$ converges to zero during training, so we achieve not only *variance reduction* but *variance elimination*, as the variance vanishes eventually.

3.3. Implementation Details

Training with the CV estimator is similar as with the NS estimator (Hamilton et al., 2017a). Particularly, each iteration of the algorithm involves the following steps:

Stochastic GCN with Variance Reduction

1. Randomly select a minibatch $\mathcal{V}_B \subset \mathcal{V}_L$ of nodes;
2. Build a computation graph that only contains the activations $h_v^{(l)}$ and $\bar{h}_v^{(l)}$ needed for the current minibatch;
3. Get the predictions by forward propagation as Eq. (6);
4. Get the gradients by backward propagation, and update the parameters by SGD;
5. Update the historical activations.

Step 3 and 4 are handled automatically by frameworks such as TensorFlow (Abadi et al., 2016). The computational graph at Step 2 is defined by the receptive field $\mathbf{r}^{(l)}$ and the propagation matrices $\hat{P}^{(l)}$ at each layer. The receptive field $\mathbf{r}^{(l)}$ specifies the activations $h_v^{(l)}$ of which nodes should be computed for the current minibatch, according to Eq. (6). We can construct $\mathbf{r}^{(l)}$ and $\hat{P}^{(l)}$ from top to bottom, by randomly adding $D^{(l)}$ neighbors for each node in $\mathbf{r}^{(l+1)}$, starting with $\mathbf{r}^{(L)} = \mathcal{V}_\mathcal{B}$. We assume $h_v^{(l)}$ is always needed to compute $h_v^{(l+1)}$, i.e., v is always selected as a neighbor of itself. The receptive fields are illustrated in Fig. 1(c), where red nodes are in receptive fields, whose activations $h_v^{(l)}$ are needed, and the histories $\bar{h}_v^{(l)}$ of blue nodes are also needed. Finally, in Step 5, we update $\bar{h}_v^{(l)}$ with $h_v^{(l)}$ for each $v \in \mathbf{r}^{(l)}$. We have the pseudocode for the training in Appendix D.

3.4. Time and Space Complexity

GCN has two main types of computation, namely, the sparse-dense matrix multiplication (SPMM) such as $PH^{(l)}$, and the dense-dense matrix multiplication (GEMM) such as $UW^{(l)}$. We assume that the input node feature is K-dimensional and the first hidden layer is A-dimensional.

For batch GCN, the time complexity is $O(EK)$ for SPMM and $O(VKA)$ for GEMM. For our stochastic training algorithm with control variates, the dominant SPMM computation is the average of neighbor history $P\bar{H}^{(0)}$ for the nodes in $\mathbf{r}^{(1)}$, whose size is $O(|\mathcal{V}_B|\prod_{l=2}^{L} D^{(l)})$, and each node costs $O(DK)$, where D is the average node degree. Therefore, the time complexity of SPMM is approximately $O(EK\prod_{l=2}^{L} D^{(l)})$ per epoch. The dominant GEMM computation is the first fully-connected layer on all the nodes in $\mathbf{r}^{(1)}$, whose time complexity is $O(VKA\prod_{l=2}^{L} D^{(l)})$ per epoch. Both time complexities are $\prod_{l=2}^{L} D^{(l)}$ times higher than batch GCN, where $\prod_{l=2}^{L} D^{(l)} = 2$ if we sample 2 neighbors per node and there are 2 GCN layers.

Our algorithm requires an additional $O(VLA)$ space to store historical activations. However, as implemented in our code, the history can be stored in main memory along with the data, which should be larger.

4. Theoretical Results

Besides smaller variance, CV also has stronger theoretical guarantees than NS. In this section, we present two theorems. The first states that if the model parameters are fixed, e.g., during testing, CV produces exact predictions after L epochs; and the second establishes the convergence towards a local optimum regardless of the neighbor sampling size.

In this section, we assume that the algorithm is run by epochs, where each epoch contains I iterations, and in each iteration we want to compute the stochastic gradient w.r.t. nodes in $\mathcal{V}_i$. We ensure that the activations of all nodes are computed at least one in each epoch, so that the staleness of the history is bounded. We use the subscript $_i$ for iteration number and $_{CV}$ to distinguish CV from the exact algorithm, i.e., $Z_i^{(l)}$ and $H_i^{(l)}$, W_i, and $g_i(W_i) := \frac{1}{|\mathcal{V}_i|}\sum_{v\in\mathcal{V}_i}\nabla f(y_v, z_{i,v}^{(L)})$ are the activations, model weights, and stochastic gradients obtained by the exact algorithm; and $Z_{CV,i}^{(l)}$, $H_{CV,i}^{(l)}$, and $g_{CV,i}(W_i)$ are their CV counterparts. $\nabla\mathcal{L}(W_i) = \frac{1}{|\mathcal{V}_\mathcal{L}|}\sum_{v\in\mathcal{V}_\mathcal{L}}\nabla f(y_v, z_v^{(L)})$ is the deterministic batch gradient computed by the exact algorithm. The subscript $_i$ may be omitted for the exact algorithm if W_i is a constant sequence. We let $[L] = \{0, \dots, L\}$ and $[L]_+ = \{1, \dots, L\}$.

4.1. Exact Testing

The following theorem reveals the connection between the exact predictions and the approximate predictions by CV. The proof can be found in Appendix B.

Theorem 1. *For a constant sequence of $W_i = W$ and any $i > LI$ (i.e., after L epochs), the activations computed by CV are exact, i.e., $Z_{CV,i}^{(l)} = Z^{(l)}$ for each $l \in [L]$ and $H_{CV,i}^{(l)} = H^{(l)}$ for each $l \in [L-1]$.*

Theorem 1 shows that at testing time, we can run forward propagation with CV for L epoches and get exact prediction. This outperforms NS, which cannot recover the exact prediction unless the neighbor sample size goes to infinity. Comparing with directly making exact predictions by an exact batch algorithm, CV is more scalable because it does not need to load the entire graph into memory.

4.2. Convergence Guarantee

The following theorem shows that SGD training with the approximated gradients $g_{CV,i}(W_i)$ still converges to a local optimum, regardless of the neighbor sampling size $D^{(l)}$.

Theorem 2. *Assume that (1) the activation $\sigma(\cdot)$ is ρ-Lipschitz, (2) the gradient of the cost function $\nabla_z f(y,z)$ is ρ-Lipschitz and bounded, (3) $\|g_{CV,\mathcal{V}}(W)\|_\infty$, $\|g(W)\|_\infty$, and $\|\nabla\mathcal{L}(W)\|_\infty$ are all bounded by $G > 0$ for all $\hat{P}, \mathcal{V}$ and W. (4) The loss $\mathcal{L}(W)$ is ρ-smooth, i.e., $|\mathcal{L}(W_2) - \mathcal{L}(W_1) - \langle\nabla\mathcal{L}(W_1), W_2 - W_1\rangle| \le \frac{\rho}{2}\|W_2 - W_1\|_F^2\ \forall W_1, W_2$, where $\langle A, B\rangle = tr(A^\top B)$ is the inner product of matrix A and matrix B. (5) The loss $\mathcal{L}(W) \ge \mathcal{L}_*$ is bounded below. Then, there exists $K > 0$, s.t., $\forall N > LI$, if we run SGD for $R \le N$ iterations, where R is chosen uniformly from $[N]_+$, we have*

$$\mathbb{E}_R\|\nabla\mathcal{L}(W_R)\|_F^2 \le 2\frac{\mathcal{L}(W_1) - \mathcal{L}_* + K + \rho K}{\sqrt{N}},$$

for the updates $W_{i+1} = W_i - \gamma g_{CV,i}(W_i)$ and the step size $\gamma = \min\{\frac{1}{\rho}, \frac{1}{\sqrt{N}}\}$.

Particularly, $\lim_{N\to\infty}\mathbb{E}_R\|\nabla\mathcal{L}(W_R)\|^2 = 0$. Therefore, our algorithm converges to a local optimum W where the batch gradient $\nabla\mathcal{L}(W) = 0$. The full proof is in Appendix C. For short, we show that $g_{CV,i}(W_i)$ is unbiased as $i \to \infty$, and then show that SGD with such asymptotically unbiased gradients converges to a local optimum.

Theorem 2 generalizes to graph attention networks (GAT) (Veličković et al., 2018). We leave the variance reduced stochastic estimators for GAT, and discussions on the convergence of GAT and other models in Appendix C.5.

5. Handling Dropout of Features

In this section, we consider introducing a third source of randomness, the random dropout of features (Srivastava et al., 2014), which is adopted in various GCN models as a regularization (Kipf & Welling, 2017; Veličković et al., 2018). With dropout, the GCN layer becomes $Z^{(l+1)} = M \circ (PH^{(l)})W^{(l)}$, where $M_{ij} \sim \text{Bern}(p)$ are i.i.d. Bernoulli random variables, and $\circ$ is the element-wise product. Let $\mathbb{E}_M$ be the expectation over dropout masks.

With dropout, all the activations $h_v^{(l)}$ are random variables whose randomness comes from dropout, even in the exact algorithm Eq. (1). We want to design a cheap estimator for the random variable $(PH^{(l)})_u = \sum_{v\in\mathbf{n}(u)} P_{uv}h_v^{(l)}$, based on a stochastic neighborhood $\hat{\mathbf{n}}^{(l)}(u)$. An ideal estimator should have the same distribution with $(PH^{(l)})_u$. However, such an estimator is difficult to design. Instead, we develop an estimator $\text{CVD}_u^{(l)}$ that eventually has the same mean and variance with $(PH^{(l)})_u$, i.e., $\mathbb{E}_{\hat{\mathbf{n}}^{(l)}(u)}\mathbb{E}_M\text{CVD}_u^{(l)} = \mathbb{E}_M(PH^{(l)})_u$ and $\text{Var}_{\hat{\mathbf{n}}^{(l)}(u)}\text{Var}_M\text{CVD}_u^{(l)} = \text{Var}_M(PH^{(l)})_u$.

5.1. Control Variate for Dropout

With dropout, $\Delta h_v^{(l)} = h_v^{(l)} - \bar{h}_v^{(l)}$ is not necessarily small even if $\bar{h}_v^{(l)}$ and $h_v^{(l)}$ have the same distribution. We develop another stochastic approximation algorithm, *control variate for dropout* (CVD), that works well with dropout.

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

Table 2. Variance from neighbor sampling (VNS) and variance from dropout (ND) of different estimators.

Our method is based on the weight scaling procedure (Srivastava et al., 2014) to approximately compute the mean $\mu_v^{(l)} := \mathbb{E}_M\left[h_v^{(l)}\right]$. That is, along with the dropout model, we can run a copy of the model without dropout to obtain the mean $\mu_v^{(l)}$, as illustrated in Fig. 1(d). We obtain a stochastic approximation by separating the mean and variance

$$(PH^{(l)})_u = \sum_{v\in\mathbf{n}(u)} P_{uv}(\mathring{h}_v^{(l)} + \Delta\mu_v^{(l)} + \bar{\mu}_v^{(l)}) \approx \mathrm{CVD}_u^{(l)}$$
$$:= \sqrt{R}\sum_{v\in\hat{\mathbf{n}}} P_{uv}\mathring{h}_v^{(l)} + R\sum_{v\in\hat{\mathbf{n}}} P_{uv}\Delta\mu_v^{(l)} + \sum_{v\in\mathbf{n}(u)} P_{uv}\bar{\mu}_v^{(l)},$$

where we define $\mathbf{n} = \hat{\mathbf{n}}^{(l)}(u)$, $R = n(u)/D^{(l)}$ for short, $\mathring{h}_v^{(l)} = h_v^{(l)} - \mu_v^{(l)}$, $\bar{\mu}_v^{(l)}$ is the historical mean activation, obtained by storing $\mu_v^{(l)}$ instead of $h_v^{(l)}$, and $\Delta\mu_v^{(l)} = \mu_v^{(l)} - \bar{\mu}_v^{(l)}$. We separate $h_v^{(l)}$ as three terms, the latter two terms on $\mu_v^{(l)}$ do not have the randomness from dropout, and $\mu_v^{(l)}$ are treated as if $h_v^{(l)}$ for the CV estimator. The first term has zero mean w.r.t. dropout, i.e., $\mathbb{E}_M\mathring{h}_v^{(l)} = 0$. We have $\mathbb{E}_{\hat{\mathbf{n}}^{(l)}(u)}\mathbb{E}_M\mathrm{CVD}_u^{(l)} = 0 + \sum_{v\in\mathbf{n}(u)} P_{uv}(\Delta\mu_v^{(l)} + \bar{\mu}_v^{(l)}) = \mathbb{E}_M(PH^{(l)})_u$, i.e., the estimator is unbiased, and we shall see that the estimator eventually has the correct variance if $h_v^{(l)}$'s are uncorrelated in Sec. 5.2.

5.2. Variance Analysis

We analyze the variance under the assumption that the node activations are uncorrelated, i.e., $\mathrm{Cov}_M\left[h_{v_1}^{(l)}, h_{v_2}^{(l)}\right] = 0, \forall v_1 \neq v_2$. We report the correlation between nodes empirically in Appendix G. To facilitate the analysis of the variance, we introduce two propositions proven in Appendix A . The first helps the derivation of the dropout variance; and the second implies that we can treat the variance introduced by neighbor sampling and by dropout separately.

Proposition 2. *If $\hat{\mathbf{n}}^{(l)}(u)$ contains $D^{(l)}$ samples from the set $\mathbf{n}(u)$ without replacement, $x_1, \ldots, x_V$ are random variables, $\forall v, \mathbb{E}[x_v] = 0$ and $\forall v_1 \neq v_2$, Cov $[x_{v_1}, x_{v_2}] = 0$, then Var$_{X,\hat{\mathbf{n}}^{(l)}(u)}\left[\frac{n(u)}{D^{(l)}}\sum_{v\in\hat{\mathbf{n}}^{(l)}(u)} x_v\right] = \frac{n(u)}{D^{(l)}}\sum_{v\in\mathbf{n}(u)}$ Var $[x_v]$.*

Proposition 3. *X and Y are two random variables, and $f(X,Y)$ and $g(Y)$ are two functions. If $E_X f(X,Y) = 0$, then Var$_{X,Y}[f(X,Y) + g(Y)] =$ Var$_{X,Y} f(X,Y) +$ Var$_Y g(Y)$.*

By Proposition 3, $\mathrm{Var}_{\hat{\mathbf{n}}}\mathrm{Var}_M\mathrm{CVD}_u^{(l)}$ can be written as the sum of $\mathrm{Var}_{\hat{\mathbf{n}}}\mathrm{Var}_M\left[\sqrt{R}\sum_{v\in\hat{\mathbf{n}}} P_{uv}\mathring{h}_v^{(l)}\right]$ and $\mathrm{Var}_{\hat{\mathbf{n}}}\left[R\sum_{v\in\hat{\mathbf{n}}} P_{uv}\Delta\mu_v^{(l)} + \sum_{v\in\mathbf{n}(u)} P_{uv}\bar{\mu}_v^{(l)}\right]$. We refer the first term as the variance from dropout (VD) and the second term as the variance from neighbor sampling (VNS). Ideally, VD should equal to the variance of $(PH^{(l)})_u$ and VNS should be zero. VNS can be derived by replicating the analysis in Sec. 3.2, and replacing h with μ. Let $s_v^{(l)} = \mathrm{Var}_M h_v^{(l)} = \mathrm{Var}_M\mathring{h}_v^{(l)}$, and $S_u^{(l)} = \mathrm{Var}_M(PH^{(l)})_u = \sum_{v\in\mathbf{n}(u)} P_{uv}^2 s_v^{(l)}$, By Proposition 2, VD of $\mathrm{CVD}_u^{(l)}$ is $\sum_{v\in\mathbf{n}(u)} P_{uv}^2\mathrm{Var}\left[\mathring{h}_v^{(l)}\right] = S_u^{(l)}$, wich equals with the VD of the exact estimator as desired.

We summarize the estimators and their variances in Table 2, where the derivations are in Appendix A. As in Sec. 3.2, VNS of CV and CVD depends on $\Delta\mu_v$, which converges to zero as the training progresses, while VNS of NS depends on the non-zero μ_v. On the other hand, CVD is the only estimator except the exact one that gives correct VD.

5.3. Preprocessing Strategy

There are two possible models adopting dropout, $Z^{(l+1)} = P(M \circ H^{(l)})W^{(l)}$ or $Z^{(l+1)} = M \circ (PH^{(l)})W^{(l)}$. The difference is whether the dropout layer is before or after neighbor averaging. Kipf & Welling (2017) adopt the former one, and we adopt the latter one, while the two models perform similarly in practice, as we shall see in Sec. 6.1. The advantage of the latter model is that we can preprocess $U^{(0)} = PH^{(0)} = PX$ and takes $U^{(0)}$ as the new input. In this way, the actual number of graph convolution layers is reduced by one — the first layer is merely a fully-connected layer instead of a graph convolution one. Since most GCNs only have two graph convolution layers (Kipf & Welling, 2017; Hamilton et al., 2017a), this gives a significant reduction of the receptive field size and speeds up the computation. We refer this optimization as the preprocessing strategy.

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

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

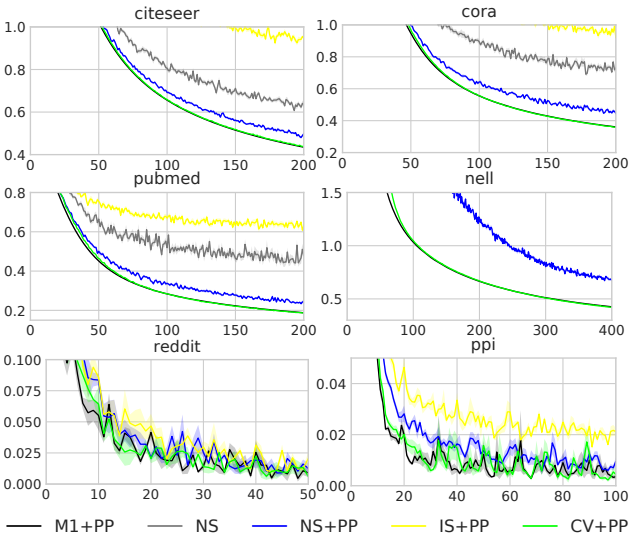


Figure 2. Comparison of training loss with respect to number of epochs without dropout. The CV+PP curve overlaps with the Exact curve in the first four datasets. The training loss of NS and IS+PP are not shown on some datasets because they are too high.

6. Experiments

We examine the variance and convergence of our algorithms empirically on six datasets, including Citeseer, Cora, PubMed and NELL from Kipf & Welling (2017) and Reddit, PPI from Hamilton et al. (2017a), with the same train / validation / test splits, as summarized in Table 1. To measure the predictive performance, we report Micro-F1 for the multi-label PPI dataset, and accuracy for all the other multi-class datasets. The model is GCN for the former 4 datasets and GraphSAGE-mean (Hamilton et al., 2017a) for the latter 2 datasets, see Appendix E for the details on the architectures. We repeat the convergence experiments 10 times on Citeseer, Cora, PubMed and NELL, and 5 times on Reddit and PPI. The experiments are done on a Titan X (Maxwell) GPU.

6.1. Impact of Preprocessing

We first examine the impact of switching the order of dropout and computing neighbor averaging in Sec. 5.3. Let M0 be the $Z^{(l+1)} = P(M \circ H^{(l)})W^{(l)}$ model by (Kipf & Welling, 2017), and M1 be our $Z^{(l+1)} = M \circ (PH^{(l)})W^{(l)}$

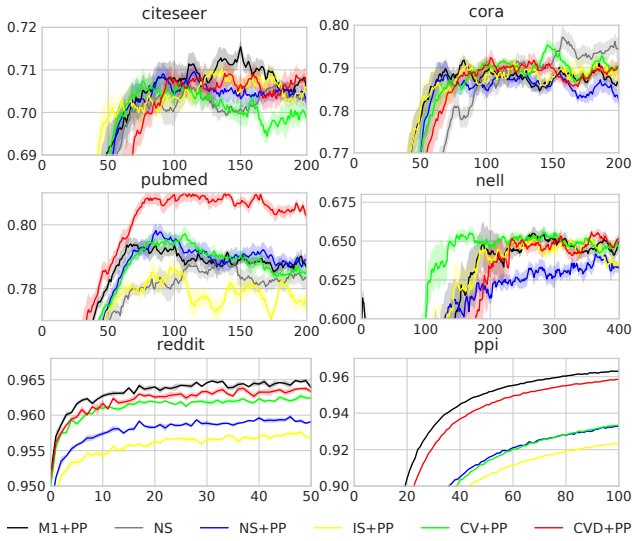


Figure 3. Comparison of validation accuracy with respect to number of epochs. NS converges to 0.94 on the Reddit dataset and 0.6 on the PPI dataset.

model, we compare three settings: M0 and M1 are exact algorithms without any neighbor sampling, and M1+PP samples a large number of $D^{(l)} = 20$ neighbors and preprocesses $PH^{(0)}$ so that the first neighbor averaging is exact. In Table 3 we can see that all the three settings performs similarly, i.e., switching the order does not affect the predictive performance. Therefore, we use the fastest M1+PP as the exact baseline in following convergence experiments.

6.2. Convergence Results

Having the M1+PP algorithm as an exact baseline, the next goal is reducing the time complexity per epoch to make it comparable with the time complexity of MLP, by setting $D^{(l)} = 2$. We cannot set $D^{(l)} = 1$ because GraphSAGE explicitly need the activation of a node itself besides the average of its neighbors. Four approximate algorithms are included for comparison: (1) NS, which adopts the NS estimator with no preprocessing. (2) NS+PP, which is same with NS but uses preprocessing. (3) CV+PP, which adopts the CV estimator and preprocessing. (4) CVD+PP, which uses the CVD estimator. All the four algorithms have similar low time complexity per epoch with $D^{(l)} = 2$, while M1+PP takes $D^{(l)} = 20$. We study how much convergence speed per epoch and model quality do these approximate algorithms sacrifice comparing with the M1+PP baseline.

We set the dropout rate as zero and plot the training loss with respect to number of epochs as Fig. 2. We can see that CV+PP can always reach the same training loss with M1+PP, while NS, NS+PP and IS+PP have higher training losses because of their biased gradients. CVD+PP is not included because it is the same with CV+PP when the dropout rate is zero. The results matches the conclusion of Theorem 2, which states that training with the CV estimator converges

Alg.	Valid. acc.	Epochs	Time (s)
M1+PP	96.0	$\mathbf{4.8 \pm .7}$	252 ± 37
NS	$94.4 \pm .01$	100	445 ± 14
NS+PP	96.0	39.8 ± 11	161 ± 47
IS+PP	$95.8 \pm .1$	50	251 ± 6
CV+PP	96.0	7.6 ± 1.6	39 ± 8
CVD+PP	96.0	6.8 ± 1.3	$\mathbf{37 \pm 7}$

Table 4. Time complexity comparison of different algorithms on the Reddit dataset.

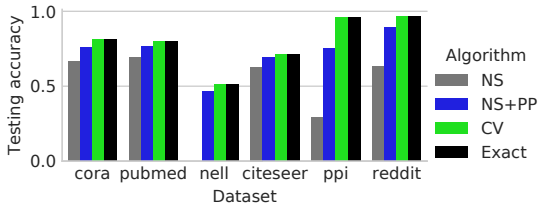


Figure 4. Comparison of the accuracy of different testing algorithms. The y-axis is Micro-F1 for PPI and accuracy otherwise.

to a local optimum of Exact, regardless of $D^{(l)}$.

Next, we turn dropout on and compare the validating accuracy obtained by the model trained with different algorithms at each epoch. Regardless of the training algorithm, the exact algorithm is used for computing predictions on the validating set. The result is shown in Fig. 3. We find that when dropout is present, CVD+PP is the only algorithm that can reach comparable validation accuracy with the exact algorithm on all datasets. Furthermore, its convergence speed with respect to the number of epochs is comparable with M1+PP, implying almost no loss of the convergence speed despite its $D^{(l)}$ is 10 times smaller. This is already the best we can expect - comparable time complexity with MLP, yet similar model quality with GCN. CVD+PP performs much better than M1+PP on the PubMed dataset, we suspect it finds a better local optimum. Meanwhile, the simpler CV+PP also reaches a comparable accuracy with M1+PP for all datasets except PPI. IS+PP works worse than NS+PP on the Reddit and PPI datasets, perhaps because sometimes nodes can have no neighbor selected, as we mentioned in Sec. 2.4. Our accuracy result for IS+PP can match the result reported by Chen et al. (2018), while their NS baseline, GraphSAGE (Hamilton et al., 2017a), does not implement the preprocessing technique in Sec. 5.3.

6.3. Further Analysis on Time Complexity, Testing Accuracy and Variance

Table 4 reports the average number of epochs and time to reach a given 96% validation accuracy on the largest Reddit dataset. Sparse and dense computations are defined in Sec. 3.4. We found that CVD+PP is about 7 times faster than M1+PP due to the significantly reduced receptive field size. Meanwhile, NS and IS+PP does not converge to the given accuracy.

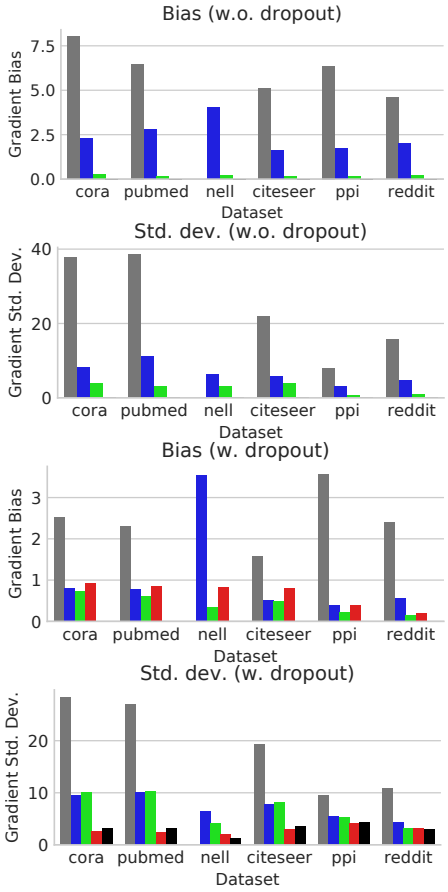


Figure 5. Bias and standard deviation of the gradient for different algorithms during training.

We compare the quality of the predictions made by different algorithms, using the *same* model trained with M1+PP in Fig. 4. As Theorem 1 states, CV reaches the same testing accuracy as the exact algorithm, while NS and NS+PP perform much worse.

Finally, we compare the average bias and variance of the gradients per dimension for first layer weights relative to the magnitude of the weights in Fig. 5. For models without dropout, the gradient of CV+PP is almost unbiased. For models with dropout, the bias and variance of CV+PP and CVD+PP are usually smaller than NS and NS+PP.

7. Conclusions

The large receptive field size of GCN hinders its fast stochastic training. In this paper, we present control variate based algorithms to reduce the receptive field size. Our algorithms can achieve comparable convergence speed with the exact algorithm even the neighbor sampling size $D^{(l)} = 2$, so that the per-epoch cost of training GCN is comparable with training MLPs. We also present strong theoretical guarantees, including exact prediction and the convergence to a local optimum. Our code is released at `https://github.com/thu-ml/stochastic_gcn`.

Acknowledgements

We thank Shuyu Cheng for his help in proofreading. This work was supported by NSFC Projects (Nos. 61620106010, 61621136008, 61332007), Beijing NSF Project (No. L172037), Tiangong Institute for Intelligent Computing, NVIDIA NVAIL Program, Siemens and Intel. L.S. was also supported in part by NSF IIS-1218749, NIH BIGDATA 1R01GM108341, NSF CAREER IIS-1350983, NSF IIS-1639792 EAGER, NSF CNS-1704701, ONR N00014-15-1-2340, Intel ISTC, NVIDIA and Amazon AWS.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

Berg, R. v. d., Kipf, T. N., and Welling, M. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

Chen, J., Ma, T., and Xiao, C. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864. ACM, 2016.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1025–1035, 2017a.

Hamilton, W. L., Ying, R., and Leskovec, J. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

LeCun, Y., Bengio, Y., et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

Ripley, B. D. *Stochastic simulation*, volume 316. John Wiley & Sons, 2009.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. v. d., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*, 2017.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *ICLR*, 2018.