Machine Learning for Graphs

GraphSAINTFAR: Revisiting GraphSAINT with a Feature-Aware Sampling Method

Arturo Abril Martinez

studentnumber: 2813498

Abstract. The work presented in this paper reproduces and extends the GraphSAINT method, a graph sampling technique for Graph Convolutional Networks. The problem context is node single and multi-class classification in the inductive setting. Three data sets (PPI, Flickr and Reddit) are used for experiments and F1 scores on their test sets are reported. An additional sampler that takes into account neighbor features to "bias" the sampling process is motivated and implemented. Additionally, a purely random sampler is introduced to act as baseline for the accuracy scores of the more elaborated samplers. The resuls obtained confirm the value of the method and highlihgt the potential of graph-based samplers in the context of GCNs.

Keywords: First keyword · Second keyword · Another keyword

1 Introduction

Graph Convolutional Networks (GCNs) have proved to be powerful models for classification tasks in Knowledge Graphs (KGs) both at the node and graph level. The scalability challenge that arises both from using large (and dense) KGs and from using deeper networks has been the main focus of a growing reasearch area during the last decade. Real-world KGs can have billions[1] of nodes. This makes necessary the use of *minibatches* of data to train the network. Furthermore, because data samples from KGs are not identically independently distributed (i.i.d.), but rather they have dependecies between each other (represented as edges in the graph), the problem of *neighbor explosion* arises: To learn the representaion of a certain node in the graph, the technique of *message passing* aggragates information from neighboring nodes for each layer in the network. As the network grows deeper (more layers), the number of neighbor embeddings that are needed to be kept in memory to update the representation of a certain node grows exponentially[2]. This can be solved by *sampling* techniques, which limit the *receptive field*[3] of nodes in the graph.
Sampling techniques are commonly subdivided into sub-groups according to what is being sampled: *node-wise* techniques sample data at the node level: for each node in the minibatch, a fraction of its neighbors is selected, and the rest is excluded from training. This is done iteratively for each layer starting from the end and going backwards. See for example [3]. *Layer-wise* methods sample data at the layer level, selecting the subset of nodes to be included layer by layer.See for example [1] or [9]. *Graph-wise* sampling methods sample subsets of the original training graph before training, and then run a full[4]-GCN on those subgraphs. This work focuses on one graph-wise sampling technique, namely `GraphSAINT`, first introduced by [8]. `GraphSAINT` proposes several sampling algorithms that aim to construct representative subgraphs of the original graph. This algorithms use topological information, this is, the structure of the graph, to make informed decisions on which nodes should be included together in a certain subgraph and which shouldn't. For the sake of simplicity, node features are disregarded when sampling, and the samplers only look at how nodes in the graph are connected to extract relevant subgraphs.

1.1 Contribution

The contribution of this work to the original `GraphSAINT` paper is two-sided: 1. a new sampling algorithm (or sampler) that does take into account node features to make informed decisions on which nodes to sample together, together with the nodes' connectivity, and 2. a purely random sampler to serve as a baseline.
In the original paper, an *edge* sampler is introduced, motivated by a variance reduction derivation. Although the reader is referred to the original article for the details, the main idea and the intuition behind the results is presented here:
Let e be the edge connecting u and v, and $\boldsymbol{b}_e^{(\ell)} = \tilde{\boldsymbol{A}}_{v,u}\tilde{\boldsymbol{x}}_u^{(\ell-1)} + \tilde{\boldsymbol{A}}_{u,v}\tilde{\boldsymbol{x}}_v^{(\ell-1)}$. The embedding of a node v in layer ℓ is defined as $\boldsymbol{x}_v^{(\ell)}$, and its estimator (using graph subsampling) will be $\zeta_v^{(\ell)}$. With

[1] For example, Wikidata has 1.57 billion nodes as of mid-2024 [7].
[2] In a graph in which all nodes have degree d, the receptive field of a given node will scale as d^L, where L is the number of layers in the network.
[3] The term *receptive field* refers to the part of the graph that a node can "see" after L iterations of message passing.
[4] Where "full" means no minibatching and no samnpling.

the goal of minimizing the variance of all estimators $\zeta_v^{(\ell)}$, the addition of all estimators ζ is defined as:

$$\zeta = \sum_{\ell} \sum_{v \in \mathcal{G}_s} \frac{\zeta_v^{(\ell)}}{p_v} = \sum_{\ell} \sum_{v,u} \frac{\tilde{\boldsymbol{A}}_{v,u}}{p_v \alpha_{u,v}} \tilde{\boldsymbol{x}}_u^{(\ell-1)} \mathbb{1}_v \mathbb{1}_{u|v} = \sum_{\ell} \sum_{e} \frac{\boldsymbol{b}_e^{(\ell)}}{p_e} \mathbb{1}_e^{(\ell)}. \tag{1}$$

and $\mathrm{Var}(\zeta)$ is to be minimized. In eq. 1, $\mathcal{G}_s$ denotes the sampled subgraph, p_v is the probability of including the node v in $\mathcal{G}_s$ when sampling, $\tilde{\boldsymbol{A}}$ is the normalized adjacency matrix, $\tilde{\boldsymbol{x}}_u^{(\ell-1)}$ is the embedding of node u at layer $\ell - 1$ already multiplied by the kernel of that layer. The indicator functions are, as expected: $\mathbb{1}_v = 1$ if $v \in \mathcal{G}_s$ and 0 o.w., $\mathbb{1}_e = 1$ if $e \in \mathcal{E}_s$[5] and 0 o.w., $\mathbb{1}_{u|v} = 1$ if $u \in \mathcal{G}_s$ given that $v \in \mathcal{G}_s$ and 0 o.w.. The expected value of these indicator functions over all possible draws of subgraphs are precisely the probabilities of the events that they represent: $\mathbb{E}(\mathbb{1}_v) = p_v$, $\mathbb{E}(\mathbb{1}_{u|v}) = p_{u,v}/p_v \equiv \alpha_{u,v}$ and $\mathbb{E}(\mathbb{1}_e^{(\ell)}) = p_e \; \forall \; \ell$[6]. As it can be easily seen, the denominators in 1 are there so that when the expected value over all possible draws of subgraphs is taken, this is $\mathbb{E}(\zeta)$, the results are the same as if one would work with the full graph.
The goal is then to choose these sampling probabilities so that $\mathrm{Var}(\zeta)$ is minimized. Focusing on edge sampling probabilities, the authors of [8] find that, after some approximations and simplifications, setting $p_e \propto \tilde{\boldsymbol{A}}_{v,u} + \tilde{\boldsymbol{A}}_{u,v} = \frac{1}{\deg(u)} + \frac{1}{\deg(v)}$ minimizes $\mathrm{Var}(\zeta)$. Intuitevely, an edge that conncects two nodes which both have few connections has high probabilities of being sampled, since this will mean that those two nodes are influential to each other.

The extension proposed in this work builds up on this derivation, adding a feature-aware term to the edge sampling probability p_e, so that $p_e \propto \alpha \cdot (\tilde{\boldsymbol{A}}_{v,u} + (\tilde{\boldsymbol{A}}_{u,v})) + (1 - \alpha) \cdot S(\boldsymbol{x}_u, \boldsymbol{x}_v)$. Where $S(\boldsymbol{x}_u, \boldsymbol{x}_v) = S(\boldsymbol{x}_v, \boldsymbol{x}_u)$ represents the cosine similarity between the nodes v and u based on their features. The use of neighbor features to learn node representations is the base of inductive learning on graphs, as presented in [3]. The idea of attending to different neighbors depending on their features is exploited in Graph Attention Networks [6]. Here, inspiration is drawn from such works to exploit node features durinig sampling.

2 Related Work

As mentioned before, several sampling techniques to efficiently train GCNs have been proposed, and can be grouped by the level at which data is being sampled, i.e, node-wise sampling, layer-wise sampling, graph-wise sampling ... GraphSAGE [3] samples a fixed number of neighbors for each node in a minibatch of the whole graph recursively for layers $\{L, L-1, ..., 1\}$. This potentially fails to avoid the neighbor explosion problem, since the receptive field of a node in the minibatch is (at worst) $\Pi_i s_i$, where s_i is the sample size for layer i. FastGCN [1] tries to mitigate this by sampling nodes at the layer level: for each layer in the GCN, a fixed number of nodes are sampled according to a probability distribution based on node degreees. If two sampled nodes happen to be connected their edge is included. Neighbor explosion is avoided, since the receptive field of a node in the minibatch is at most $\sum_i s_i$. This potentially result in very sparse layers, with sampled nodes without any conncections, making message passing difficult. LADIES [9] is introduced to solve this presented potential problems. For each layer ℓ[7], a subgraph containing the nodes of layer $\ell + 1$

[5] $\mathcal{E}_s$ is the set of edges sampled in the subgraph s.
[6] This is the same for all layers since once a subgraph is sampled, its structure remains untouched.
[7] The last layer L is formed by the nodes selected in a given minibatch, and the sampling described starts creating a subgraph between layers $L - 1$ and L.

plus their neighbours is created. The sampling probabilities of nodes in the constructed subgraph are calculated based on node degrees, and a fixed-sized sample is drawn using these probabilities. The layer-wise approach avoids neighbor explosion, while the layer-dependent importance sampling ensures conncectivity is well mantained. On the graph-wise sampling techniques, ClusterGCN [2] leverages graph clustering algorithms to decompose the adjacency matrix into c^2 submatrices (c is the number of created clusters) in the form $\boldsymbol{A} = \bar{\boldsymbol{A}} + \Delta$ where $\bar{\boldsymbol{A}}$ is a block diagonal matrix that contains the adjacency matricies for each cluster, and Δ encodes the inter-cluster conncections. By disregarding Δ they can efficiently parallelize the training of the GCN. Unlike `GraphSAINT`, ClusterGCN doesn't account for the bias towards specific clusters.

On the leveraging of features from a node's local neighborhood to learn node representations, GraphSAGE [3] first samples a fixed-sized neighborhood for each node and then applies a specific aggregator over it, so that node representations can be computed in an inductive way (so that it can work with previouly unseen data). [6] uses attention to learn which neighbors are the most important based on their features. This idea of quantifying node importance based on neighbor features is the underlying motivation for the extension proposed in this work.

3 Background

The task is to learn a representation of an un-directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ for both single class and multiple class node classification in the inductive setting, which means that features and connectinos from test nodes are unseen during training. This is done using the general framework of GCNs presented by [5] adapted to the inductive setting by [3]. Each node $v \in \mathcal{V}$ has a length-f attribute $\boldsymbol{x}_v$. As introduced above, $\boldsymbol{A}$ is the adjacency matrix and $\tilde{\boldsymbol{A}}$ is the normalized one[8]. Let the dimension of layer-ℓ input activation be $f^{(\ell)}$. The activation of node v is $\boldsymbol{x}_v^{(\ell)} \in \mathbb{R}^{f^{(\ell)}}$, and the weight matrix is $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{f^{(\ell)} \times f^{(\ell+1)}}$. Note that $\boldsymbol{x}_v = \boldsymbol{x}_v^{(1)}$.
The propagation rule of a layer is:

$$\boldsymbol{x}_v^{(\ell+1)} = \sigma\left(\sum_{u \in \mathcal{V}} \tilde{\boldsymbol{A}}_{v,u} \left(\boldsymbol{W}^{(\ell)}\right)^\top \boldsymbol{x}_u^{(\ell)}\right) \tag{2}$$

where $\tilde{\boldsymbol{A}}_{v,u}$ is a scalar, taking an element of $\tilde{\boldsymbol{A}}$. And $\sigma(\cdot)$ is the activation function (e.g., ReLU).
The way the `GraphSAINT` method works is as follows: subgraphs $(\mathcal{G}_s, \mathcal{V}_s, \mathcal{E}_s)$ are sampled from the original graph $\mathcal{G}$ making sure that 1. Nodes with high influence between each other are sampled together and 2. Every edge has non-zero probability of being sampled. This results in well-connected subgraphs with a fixed number of nodes. These sampled subgraphs form the minibatches that are used to train a GCN which layers have the general form of 2 or some variants of it (for example the GAT from [6]) using Stochastig Gradient Descent (SGD).
Importantly, to account for the bias introduced when sampling subgraphs,
`GraphSAINT` introduces normalization factors for both the neighbor aggration of eq. 2 and for the loss function. The propagation rule of a layer when using a sampled subgraph s is then:

$$\zeta_v^{(\ell+1)} = \sum_{u \in \mathcal{V}} \frac{\tilde{\boldsymbol{A}}_{v,u}}{\alpha_{u,v}} \left(\boldsymbol{W}^{(\ell)}\right)^\top \boldsymbol{x}_u^{(\ell)} \mathbb{1}_{u|v} = \sum_{u \in \mathcal{V}} \frac{\tilde{A}_{v,u}}{\alpha_{u,v}} \tilde{\boldsymbol{x}}_u^{(\ell)} \mathbb{1}_{u|v} \tag{3}$$

[8] Which can be row-wise normalized: $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A}$, column-wise normalized: $\tilde{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{D}^{-1}$ or symmetrically normalized: $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ where $\boldsymbol{D}$ represents the degree matrix.

Where, as previously mentioned, $\alpha_{u,v}$ is set equal to $p_{u,v}/p_v$ so that, on expectation over all possible sampled subgraphs, $\zeta_v^{(\ell+1)}$ equals the quantity inside $\sigma(\cdot)$ in eq. 2. See the description of eq. 1 for more details about the notation.
For the loss of a minibatch: $L_{\text{batch}} = \sum_{v \in \mathcal{G}_s} L_v/\lambda_v$. Again, to make sure that, on expectation, the minibatch loss is equal to the full batch loss $L_{\text{full}} = 1/|\mathcal{V}| \sum_{v \in \mathcal{V}} L_v$, $\lambda_v \equiv p_v$. All needed to compute the normalization factors $\alpha_{v,u}$ and λ_v are the sampling probabilities of nodes and edges. This are estimated using counts in a pre-training phase in which several subgraphs are sampled according to a specified sampling algorithm. After this phase is finished, the sampling probabilities are set as $\alpha_{u,v} = C_{u,v}/N$ and $\lambda_v = C_v/N$, where $C_{u,v}$ and C_v are the count of the edge conncting nodes u and v and the count of node v respectively. N is the number of sampled subgraphs in this pre-training phase[9].

4 Research Reproduction

This work aims to reproduce the following part of the original `GraphSAINT` paper: Regarding the proposed samplers, namely, *node* sampler, *edge* sampler, *random walk (RW)* sampler and *multi dimensional RW* sampler, this work reproduces all of them but the last one, which, as claimed by the authors, is the more expensive to execute. The edge and RW samplers are found to perform the best in the original paper. The **node** sampler picks nodes randomly according to a probability distribution based on nodes degrees (as in [1]). The **edge** sampler picks edges at random, and the probability of picking and edge is $p_e \propto \tilde{\boldsymbol{A}}_{u,v} + \tilde{\boldsymbol{A}}_{v,u}$ where e is the edge connecting u and v, as motivated before. The **ranodm walk** sampler performs a random walk from a number of root nodes and for a given depth. The pseudocode for the samplers is attached in the Appendix A of this work for convenience. Regarding the datasets used for node classification in the inductive, supervised learning setting, namely, PPI (Protein-Protein Interaction), Flickr, Reddit, Yelp and Amazon, ordered in increasing size, only the first three of them are included in this study. The main reason is that the code used for the reproduction always tries to use a CUDA-compatible GPU, if available. Since the full graph is needed to be fit in memory for evaluation (both for validation and test sets), an "out-of-memory" exception is thrown when trying to do this for both the Yelp and Amazon datasets. On [8], the authors allow for CPU evaluation. This adaptation (CPU evaluation) is left for future work and is not part of this reproduction. Regarding the proposed network architectures in the original paper, here only the GraphSAGE layer propagation rule is used, with a small modification that is proposed on the original paper and detailed as follows: the "order" of a GraphSAGE layer is defined as the exponent which the adjacency matrix is raised to. For order 0 (the identity), this is just a regular MLP, order 1 is the original GraphSAGE layer and higher order correspond to aggregating information from *order*-hop neighbors.

4.1 Experiments & Results

The code used to obtained the results presented in this and following sections can be found here[10]. The statistics and splits of the datasets used in the reproduction (and extension) of `GraphSAINT` can be seen in table 1. The **(s)** and **(m)** in table 1 stand for **s**ingle-class and **m**ulti-class classification. On the PPI dataset, nodes are proteins and edges represent interactions between them. Proteins

[9] Note that this sampled subgraphs in the pre-training phase are kept and used for training later.
[10] https://github.com/arturoam00/graph_saint_repro

can belong to multiple classes, so the task is a multi-class classification trying to predict the class of a protein. For Reddit, nodes are Reddit posts and edges represent links between posts that share comments. The task is to predict post categories. In Flickr, nodes are images in a social network and edges represent similar properties or interactions between images. The task is to predict image categories (these are the labels).

Table 1: Dataset statistics and splits.

Dataset	Nodes	Edges	Degree	Feature	Classes	Train / Val / Test
PPI	14,755	225,270	15	50	121 (m)	0.66 / 0.12 / 0.22
Flickr	89,250	899,756	10	500	7 (s)	0.50 / 0.25 / 0.25
Reddit	232,965	11,606,919	50	602	41 (s)	0.66 / 0.10 / 0.24

The results of the reproduction on the mentioned datasets can be seen in table 2 with the prefix **Repro**. This are F1-micro scores on the test sets averaged over three independent runs with the same set of hyperparameters. All results correspond to a two-layer GCN (with possible extra GraphSAGE layers of order 0) using the concatenation operation for order-1 GraphSAGE layers. For the multi-class classification task, the Sigmoid + Binary-Cross-Entropy loss is used, and the Cross-Entropy loss for the single class classification tasks. The training is performed using SGD with the Adam optimizer [4]. A detailed description of the hyperparameters used to obtained this results can be found on the Appendix A. For the reader's convenience, the results obtained in the original paper are included in the table with the prefix **Original**.

Table 2: F1-micro scores on the test set obtained in the reproduction of `GraphSAINT` and original results. Results are the average over 3 independent identical runs.

Method	**PPI**	**Flickr**	**Reddit**
Original-Node	0.960 ± 0.001	0.507 ± 0.001	0.962 ± 0.001
Original-Edge	0.981 ± 0.007	0.510 ± 0.002	0.966 ± 0.001
Original-RW	0.981 ± 0.004	0.511 ± 0.001	0.966 ± 0.001
Repro-Node	0.923 ± 0.006	0.528 ± 0.015	0.962 ± 0.000
Repro-Edge	$\mathbf{0.998} \pm 0.000$	$\mathbf{0.545} \pm 0.009$	$\mathbf{0.978} \pm 0.002$
Repro-RW	0.995 ± 0.000	$\mathbf{0.535} \pm 0.027$	0.972 ± 0.001

Reproduction results are close to the original ones, and even higher in most of the cases. All hyperparameters have been kept the same as in the original paper, except for the PPI dataset, for which the normalization of the adjacency matrix has been done symmetrically, as oppesed to row-wise, as the authors claim they did.

5 Research Extension

The sampling algorithm presented in the introduction is found to be difficult to implemet efficiently, so it is slightly modified as follows: instead of embedding the cosine similarity between features with the edge sampler, it is done with the RW walk sampler (see Appendix A for the original RW sampler from `GraphSAINT`). The pseudocode for this algorithm is presented in 1. A random[11] walk with a fixed number of roots (starting nodes) and a fixed depth (how many "hops" to take from the root) is performed. When selecting a node from the root's neighborhood, instead of doing it randomly, the neighbor whose features are the most similar to the root node is selected. This cosine similarity is basically the dot product of the root with its neighbors properly normalized. Note that line 10 in 1 assumes broadcasting along the right dimensions.

Algorithm 1 Feature-aware random walk sampler

Require: Training graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; Sampling parameters: node budget n; edge budget m; number of roots r; random walk length h
Ensure: Sampled graph $G_s(\mathcal{V}_s, \mathcal{E}_s)$

1: **function** FARW($\mathcal{G}, r, h$) ▷ Feature-aware random walk sampler
2: $\mathcal{V}_{\text{root}} \leftarrow r$ root nodes sampled uniformly at random (with replacement) from $\mathcal{V}$
3: $\mathcal{V}_s \leftarrow \mathcal{V}_{\text{root}}$
4: **for** $v \in \mathcal{V}_{\text{root}}$ **do**
5: $u \leftarrow v$
6: $u_f \leftarrow$ vector of u's features
7: **for** $d = 1$ to h **do**
8: $n\_map \leftarrow$ Neighbors of u index to train node id map
9: $N_f \leftarrow u$'s neighbors feature matrix
10: $u \leftarrow n\_map[\text{argmax}(S(u, N_f))]$
11: $\mathcal{V}_s \leftarrow \mathcal{V}_s \cup \{u\}$
12: $G_s \leftarrow$ Node induced subgraph of G from $\mathcal{V}_s$

As mentioned, a purely random sampler is implemeted to be used as a baseline. This sampler just randomly selects a subset of nodes given a fixed subset size. Then, the subgraph that this subset of nodes induce is returned for the full GCN.

5.1 Experiments & Results

The datasets on which the contribution is evaluated are the same as the one presented before (table 1). On table 3 the results for the F1-micro scores on the test sets are presented. "Random" stands for the purely random baseline sampler and "FARW" stands for Feature-Aware RW sampler (both part of this contribution). The experiment setup is analogous as the one described in the previous section. Hyperparameters used for the different samplers can be found on table 4 in the Appendix. The only new data on table 3 w.r.t. table 2 are the rows "Repro-Random" and "Repro-FARW". Interestingly enough, the random node sampler is the one with the highest accuracy on two of the

[11] Now the walk is not completely random, since the neighbor selection is biased. The selection of the root nodes is still random.

Table 3: F1-micro scores on the test set obtained in the reproduction of `GraphSAINT`. Results are the average over 3 independent identical runs.

Method	**PPI**	**Flickr**	**Reddit**
Original-Node	0.960 ± 0.001	0.507 ± 0.001	0.962 ± 0.001
Original-Edge	0.981 ± 0.007	0.510 ± 0.002	0.966 ± 0.001
Original-RW	0.981 ± 0.004	0.511 ± 0.001	0.966 ± 0.001
Repro-Random	$\mathbf{0.988} \pm 0.001$	$\mathbf{0.547} \pm 0.005$	0.974 ± 0.000
Repro-Node	0.923 ± 0.006	0.528 ± 0.015	0.962 ± 0.000
Repro-Edge	$\mathbf{0.998} \pm 0.000$	$\mathbf{0.545} \pm 0.009$	$\mathbf{0.978} \pm 0.002$
Repro-RW	0.995 ± 0.000	0.535 ± 0.027	0.972 ± 0.001
Repro-FARW	0.903 ± 0.048	0.509 ± 0.002	0.962 ± 0.002

three data sets used. The feature-aware random sampler doesn't present accuracy improvements on any of the data sets. Again, the results from the original paper [8] are included in the table, prefixed with **Original**.

6 Discussion & Conclusion

A considerable part of the paper `GraphSAINT` [8] has been reproduced succesfully in this work. The results of the experiments confirm the value of this method and highlight the potential of graph-based samplers in the context of Graph Convolutional Networks. Moreover, a new idea for a graph sampler, understood as an extension of `GraphSAINT`, has been introduced. This new sampling algorithm is a modification of a random-walk based sampler that takes neighbor's features into account to "bias" the walk. The intuiton behind this, is that nodes that share similar features are influential to each other and therfore it is interseting to keep them on the same subgraph. Alternatively, one can think that is actually the oppossite what is more intersting to have (i.e. very dissimilar nodes w.r.t. their features), to have a more diverse and information-rich subgraph. This could be an interesting extension to work on. Additionally, the results of different sampling algorithms in the used datasets have been compared to a random baseline, picking nodes purely at random to construct the subgraphs that are used for the GCN. This method, although seemingly naiive, achieves excellent results and even outperforms some (apparently) better-thought sampling methods. This highlihgts the importace of random baselines and potentially suggest the need for more difficult classification tasks. Regarding the limitations of this work, experiments were not conducted in the two biggest data sets (Yelp and Amazon) because of out-of-memory issues. Because of time constraints, the adaptation of the reproduction code to overcome this problem is left for future work. With respect to running times, the results obtained in this reproduction are clearly worse than those of the original work. Although not reported in the results, it is worth mentioning that there have been big differences on the sampling pre-processing phase times, needed to estimate the normalization factors for the loss and the node features. This was expected because all the code used here has been written in Python, while the authors of [8] use cython to efficiently parallelize sampling.

References

1. Chen, J., Ma, T., Xiao, C.: FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. CoRR **abs/1801.10247** (2018). arXiv: 1801.10247. http://arxiv.org/abs/1801.10247
2. Chiang, W.-L. *et al.*: Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In. KDD '19. ACM (2019). https://doi.org/10.1145/3292500.3330925
3. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive Representation Learning on Large Graphs. CoRR **abs/1706.02216** (2017). arXiv: 1706.02216. http://arxiv.org/abs/1706.02216
4. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015). http://arxiv.org/abs/1412.6980
5. Kipf, T.N., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. CoRR **abs/1609.02907** (2016). arXiv: 1609.02907. http://arxiv.org/abs/1609.02907
6. Veličković, P. *et al.*: Graph Attention Networks, (2018). arXiv: 1710.10903 [stat.ML]. https://arxiv.org/abs/1710.10903.
7. Wikipedia contributors, Wikidata — Wikipedia, the free encyclopedia, (2024). https://en.wikipedia.org/wiki/Wikidata. [Online; accessed 3-February-2025].
8. Zeng, H. *et al.*: GraphSAINT: Graph Sampling Based Inductive Learning Method. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020). https://openreview.net/forum?id=BJe8pkHFwS
9. Zou, D. *et al.*: Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. CoRR **abs/1911.07323** (2019). arXiv: 1911.07323. http://arxiv.org/abs/1911.07323

A Appendix

Algorithm 2 Graph sampling algorithms used in the reproduction of GraphSAINT

Require: Training graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; Sampling parameters: node budget n; edge budget m; number of roots r; random walk length h

Ensure: Sampled graph $G_s(\mathcal{V}_s, \mathcal{E}_s)$

function NODE($\mathcal{G}, n$) ▷ Node sampler

  $P(v) \equiv \left\|\widetilde{\mathbf{A}}_{:,v}\right\|^2 / \sum_{v' \in \mathcal{V}} \left\|\widetilde{\mathbf{A}}_{:,v'}\right\|^2$

  $\mathcal{V}_s \leftarrow n$ nodes randomly sampled (with replacement) from $\mathcal{V}$ according to P

  $G_s \leftarrow$ Node induced subgraph of G from $\mathcal{V}_s$

function EDGE($\mathcal{G}, m$) ▷ Edge sampler (approximate version)

  $P((u,v)) \equiv \left(\frac{1}{\deg(u)} + \frac{1}{\deg(v)}\right) / \sum_{(u',v') \in \mathcal{E}} \left(\frac{1}{\deg(u')} + \frac{1}{\deg(v')}\right)$

  $\mathcal{E}_s \leftarrow m$ edges randomly sampled (with replacement) from $\mathcal{E}$ according to P

  $\mathcal{V}_s \leftarrow$ Set of nodes that are end-points of edges in $\mathcal{E}_s$

  $G_s \leftarrow$ Node induced subgraph of G from $\mathcal{V}_s$

function RW($\mathcal{G}, r, h$) ▷ Random walk sampler

  $\mathcal{V}_{\text{root}} \leftarrow r$ root nodes sampled uniformly at random (with replacement) from $\mathcal{V}$

  $\mathcal{V}_s \leftarrow \mathcal{V}_{\text{root}}$

  for $v \in \mathcal{V}_{\text{root}}$ **do**

    $u \leftarrow v$

    for $d = 1$ to h **do**

      $u \leftarrow$ Node sampled uniformly at random from u's neighbors

      $\mathcal{V}_s \leftarrow \mathcal{V}_s \cup \{u\}$

  $G_s \leftarrow$ Node induced subgraph of G from $\mathcal{V}_s$

Table 4: Hyperparameters used for the resuls of both tables 2 and 3. Please **note** that for the random node sampler, the hyperparameters used are the same as the ones presented here for the "Node" sampler.

Sampler	**Dataset**	**Training**			**Sampling**		
		Learning rate	**Dropout**	**Node budget**	**Edge budget**	**Roots**	**Walk length**
Node	PPI	0.01	0.0	6000	—	—	—
	Flickr	0.01	0.2	8000	—	—	—
	Reddit	0.01	0.1	8000	—	—	—
	Yelp	0.01	0.1	5000	—	—	—
	Amazon	0.01	0.1	4500	—	—	—
Edge	PPI	0.01	0.1	—	4000	—	—
	Flickr	0.01	0.2	—	6000	—	—
	Reddit	0.01	0.1	—	6000	—	—
	Yelp	0.01	0.1	—	2500	—	—
	Amazon	0.01	0.1	—	2000	—	—
RW	PPI	0.01	0.1	—	—	3000	2
	Flickr	0.01	0.2	—	—	6000	2
	Reddit	0.01	0.1	—	—	2000	4
	Yelp	0.01	0.1	—	—	1250	2
	Amazon	0.01	0.1	—	—	1500	2
FARW	PPI	0.01	0.1	—	—	3000	2
	Flickr	0.01	0.2	—	—	6000	2
	Reddit	0.01	0.1	—	—	2000	4
	Yelp	0.01	0.1	—	—	1250	2
	Amazon	0.01	0.1	—	—	1500	2