An Efficient Sampling Framework for Graph Convolutional Network Training

Abderaouf Gacem
Université Lyon 1
Hamida Seba (hamida.seba@univ-lyon1.fr)
Université Lyon 1 https://orcid.org/0000-0003-0670-815X
Mohammed Haddad
Université Lyon 1

Research Article

Keywords: Graph Convolutional Networks, Graph Sampling, Minibatch Training

DOI: https://doi.org/10.21203/rs.3.rs-2648725/v1

An Efficient Sampling Framework for Graph Convolutional Network Training

Abderaouf Gacem, Hamida Seba, Mohammed Haddad
Univ Lyon, UCBL, CNRS, INSA Lyon, LIRIS, UMR5205,
VilleurbanneF-69622, France
{abderaouf.gacem, hamida.seba, mohammed.haddad}@univ-lyon1.fr

Abstract

Graph Convolutional Networks (GCNs) are a subcategory of neural networks that can take a graph as input. They have been solicited lately due to their success and the ubiquity of data that can be represented as graphs. However, handling large graphs has been one of the main challenges for GCNs, along with the oversmoothing problem that can be caused by the depth or the large receptive field of GCNs. A lot of work has been done to address this problem with different approaches. A promising one is the consideration of the minibatch training paradigm and extending it to graph structured data by extracting subgraphs to be used as batches. Unlike the entries of a dataset of images that are independent from one another, the essence of a graph is its topology structure, hence the dependency between its nodes. Thus, the strategy of selecting subgraphs to form minibatches is a difficult task with a huge impact on the results of the training process. In this work, we propose a general framework for generating minibatches in an effective way that ensures a minimum loss in node interdependence information, preserves the original graph properties, and diversifies the samples for the GCN to better generalize afterwards. We test our training process on real world datasets with several well known GCN models and show the results improvement compared to existing methods. The source code and data are publicly available at https://gitlab.liris.cnrs.fr/hseba/sffs-gcn.

:

1 Introduction

Graph representation learning (GRL) has recently known a tremendous success, especially with the emergence of graph neural networks. The aim of GRL is to embed the graph or its components, mainly nodes, into a continuous vectorial representation that can be used to represent the graph more efficiently and effectively in a machine learning setting. Graphs (or networks) are a general abstraction model that can be used to represent a wide range of real world data involving interacting objects. With the availability of large graph datasets and the growing use of their applications, such as protein interface prediction [1] and network traffic forecasting [2], involving machine learning (deep learning specifically) in graph based tasks became an urgent issue. To do so, the scientific community tried to exploit and extend various existing mechanisms to the graph domain. In [3, 4], the authors exploit random walks to extend the *Skipgram* [5] model, used in natural language processing, to graphs. Prior works such as the one described in [6, 7] proposed neural networks designed to operate directly on graphs. The work of [7] can be considered as an extension of recurrent neural networks for example. These works were the start of a series of neural network models known as Graph Neural Networks (GNNs).

Driven by the success of convolutional neural networks (CNNs) mainly in image and natural language processing, the authors of [8] combine spectral graph theory and signal processing to define a graph convolutional network (GCN). Many works followed [9, 10, 11, 12, 13, 14], all of them are usually united under the gather-aggregate-update framework [15] to learn representations for nodes. In this framework, the initial representation h_v of a node v is its set of features, then on each layer of the GCN, the representations of v's neighbors are gathered, aggregated, and finally used to update h_v and pass it to the next layer. This process can be perceived as if adjacent nodes pass their current representations to a target node to use them in updating its own representation and is usually called *message passing*.

This learning procedure causes two main limitations on GCNs:

- Only small datasets could be apprehended by such models, and
- These deep learning models can't go so deep in practice. In fact, the model's performance starts decreasing beyond two-layer depth.

These limitations are explained by the fact that the receptive field per node, for its final representation to be computed, increases exponentially as the model goes deeper. Besides, iteratively aggregating node representations causes their particularities to fade out especially when the average operation is used as an aggregator. This phenomenon is known

as *oversmoothing* where most nodes are implicated in learning each other's representation and, as a result, they become so similar to one another that it is no longer possible for a classifier to accurately assign classes.

To solve this issue, many works have been proposed to alleviate the node dependency problem that prevents GCNs from being trained on a budget of nodes independently from the graph size. The main approaches rely on sampling methods [16, 17, 18] that select a subset of nodes for each training step. The idea is to implement a process similar to the minibatch training paradigm with graph data. In this paradigm, the dataset is divided into several batches. Then, the learning model is fed these batches successively. The model accumulates the learning going through them while avoiding being overwhelmed by the size of the full dataset. However, unlike the entries of a dataset such as images that are independent from one another, the essence of a graph is its topology structure, hence the dependency between its nodes. So, selecting nodes and connections per batch is not an easy task, thus the sampling strategy is of a big influence on the performance of a GCN.

The aim of this paper is to deal with the above issue through a new sampling framework for alleviating GCN training. We propose SFFS-GCN, a subgraph sampling method for GCN training and show its effectiveness through several experiments.
Our key contributions are as follows:

- We propose an effective sampling method to sample subgraphs preserving the original graph properties. This method is based on the *forest fire graph model* proposed in [19] to model evolving graphs. Our sampling method keeps a level of connectivity that allows to bypass node dependencies, and is diverse enough for the GCN to be able to generalize the learning.
- We show that our framework is a generalization for a number of existing sampling strategies.
- We also show that it can be used with several well-known GCN models and that these models scale better and improve their performance when used with our framework.

The remainder of the paper is organized as follows: Section 2 defines the problem of sampling for GCN training and discusses related work. Section 3 introduces the proposed framework, and Section 4 describes its evaluation experiments and discusses the results. Section 5 concludes the paper.

2 Problem Definition and Background

Let $G = (V, E, X_V, X_E)$ be an attributed graph where V is a set of n nodes ($n = |V|$, and $E \subseteq V \times V$ is a set of m edges ($m = |E|$). $X_V \in \mathbb{R}^{n \times F_V}$, resp. $X_E \in \mathbb{R}^{n \times F_E}$, is a matrix containing node features, respectively edge features. We suppose that each node has F_V features and each edge has F_E features. Let A be the adjacency matrix of G. A is a symmetric matrix of size n^2 such that $a_{ij} = a_{ji} = 1$ if nodes v_i and v_j are neighbors in G, i.e., there is an edge that connects the two nodes. The set of all neighbors of v_i in G is denoted $N(i)$.

We denote by D, the diagonal matrix that stores the node degrees, i.e., $(d_{ii} = degree(v_i))$.

On a GCN, a layer is characterized by a learnable weight matrix W and an activation function σ. The representation of a node v_i at a layer l is noted $h_i^{(l)}$. This representation generally depends on the representations $h_j^{(l-1)}$ of v_i's neighbors $N(i) = \{v_j | (v_j, v_i) \in E\}$.

GCNs were first proposed to be trained on the whole graph. This is called full-batch training. The representation of the nodes are generally initialized with the nodes features, i.e., $h_i^{(0)} = X_V(i)$. Afterwards, they are updated from layer to layer. The final representations are used to perform the targeted downstream task e.g. node classification. A loss is calculated based on the performance observed on the set of nodes used for training. Then, backpropagation is used to update the weights based on the stochastic gradient descent (SGD) algorithm. In the reference paper of GCN models [8], the authors apply the following update function:

$$h_i^{(l)} = \sigma[\, W^{(l)} \sum_{v_j \in N(i) \cup \{v_i\}} (d_{ii}.d_{jj})^{\frac{-1}{2}} h_j^{(l-1)}] \qquad (1)$$
$$h_i^{(0)} = X_V(i)$$

Where σ is the ReLU activation function This update procedure is the root of the memory bottleneck problem of GCNs. In fact, for larger graphs, the memory complexity of this operation becomes too high to be affordable. And even for small graphs, only shallow GCNs can be defined using this update function.

We can summarize the limitations of this full batch training process using this update function as follows:

1. The recursive nature of the update function leads to an exponentially larger memory space at each step,
2. The large size of the gathered loss over a whole epoch slows the convergence speed of SGD,
3. Node finale representations tend to be close to one another because of the spread over each recursive step, i.e., it is an oversmoothing problem.

To deal with these limitations, existing approaches rely mainly on sampling techniques that aim to decrease the number of nodes that the update function considers at each iteration.

Any sampling method requires dropping either nodes, edges, or both of them. The challenge is then to appropriately construct the batches so that a minimum amount of information is lost and that the overall learning comes as close as possible to learning on the full-batch graph. In order to minimize the information loss due to breaking the nodes dependency, a sampling method should satisfy the following properties:

- All the nodes should have the same sampling probability : the union of minibatches needs to cover the whole graph, and the overall nodes apparitions in the minibatches has to be as balanced as possible,
- Randomness : feeding the model different graph snapshots leads to better generalization capabilities afterwards. The sampling method should provide different subgraphs even when the starting seed set is the same,
- Connectivity : GCN performance is strongly related to node dependencies. This dependency is expressed by the connections and thus the graph connectivity. Needless to say that a GCN would perform poorly on a disconnected graph. The sampling method should then yield subgraphs suitable for a GCN to be run on,
- The minibatches should reflect the full batch graph properties such as degree distribution, density and community structures.

The main sampling approaches proposed in the literature to alleviate GCN training can be classified within the following categories:

1. **Sampling at a node level:** In order to moderate the memory complexity of GCNs training, the authors of [12, 20] propose to limit the neighborhood gathering budget for the message passing operation. In other words, they perform a node sampling prior to message passing. *Graphsage* [12] selects only k neighbors for each node and each layer training and uses them to update the representation of the node via message passing. Although the selection strategy varies from one work to another, all these methods share the burden of time complexity for the individual sampling operation. And thus, fail to properly scale to large datasets.
2. **Sampling at a layer level:** The authors in [11, 21, 22] propose to perform a layer conditioned selection of neighbors on one sampling step. Sampling at a layer level allows the neighborhood selection to be most relevant to the layer for which it is done. For instance, *FastGCN* [11] proposes the concept of importance sampling and in [21, 22], the authors perform a conditional sampling so that only the nodes needed on lower layers are sampled on topper ones. These methods limit the neighborhood exponential expansion, reduce the sampling time as well as the noise propagation that causes oversmoothing. Yet, they fail to resolve the nodes dependency and the sampling overload becomes unacceptable for very large graphs.
3. **Sampling at a graph level:** Authors in [23] argue that sampling on a graph level can be used to improve both the accuracy and the efficiency of GCNs. For a set of target nodes, they sample a shallow local subgraph based on a personalized PageRank function and run the GCN on it. In [16, 17, 18], the authors tried to generalize this model by extending the minibatch training paradigm to graphs by sampling subgraphs and using them as minibatches. *GraphSaint* [17] achieves this by applying several sampling strategies: random node sampling, random edge sampling and random walk sampling. Then the minibatch would be the subgraph induced by the visited nodes selected by the sampling strategy. *Cluster-GCN* [16] partitions the graph based on a clustering algorithm. And then, partitions are randomly fused to form minibatches. These methods introduce a new challenge: how to navigate node dependencies and select subgraphs while losing a minimum amount of information ?

In the next section, we introduce a new subgraph sampling method that focuses on preserving node dependencies to provide subgraphs exhibiting very similar properties to the ones observed on the full input graph.

3 SFFS-GCN: Subgraph Forest Fire Sampling for GCN Training

3.1 Principle

In this section, we introduce *SFFS-GCN*, an effective sampling method for GCN training. *SFFS-GCN* provides subgraphs, i.e., minibatches, exhibiting very similar properties to the ones observed on the full original graph. *SFFS-GCN* is a sampling method inspired from a graph generation model called *Fire Forest* graph model and introduced in [19] to model evolving real-world networks. The fire forest graph model is a randomized process that mimics real-world graph evolution over time. This model preserves the properties of real-world graphs such as density, communities, distances, and degree distribution. This was verified by simulation in [19]. In this process, a graph is generated by successively adding nodes. Each time a new node v is added to the graph, it is first linked with an edge to an existing node w chosen at random. Then the process picks randomly x neighbors of w, where x is a random number that is binomially distributed with mean $(1-p)^{-1}$, and links v to these x nodes. This process is then repeated recursively on the x picked nodes until there are no other nodes to visit. To ensure this process does not cycle, already visited nodes are not revisited. We say that the link to visited nodes are burned which gives a forest fire spread like process. This graph model needs only one parameter p which is called the burning probability.

We use a similar process to sample subgraphs from a graph by spreading a fire as detailed in Algorithm 1. Our sampling starts by randomly selecting a set of s node seeds from the original graph and put them in a queue in order to spread a fire starting from each one of them (cf. line 2,3). The unvisited set helps us not to visit the same node twice. That's why the seeds are removed from $V_{unvisited}$ and put in the queue.

One by one, we take out a target node from the queue (cf. line 8). We locate the unvisited nodes among its neighbors (cf. line 9) and spread a fire through a sampled percentage of them. To do that, we sample a value from a triangular distribution over the interval $[0, 1.0]$ and use the burning probability p to bias the percentage towards one interval's ends (cf. line 10). This percentage is then used to determine the number x of unvisited neighbors to be selected (cf. line 11,12). The target node is considered sampled and the burnt or selected neighbors are to be targeted next (cf. line 13). Since there is no need to consider the burnt nodes during the neighbors selection step, we remove them from the unvisited set (cf. line 14). We keep going through the nodes in the queue until the size of V' reaches S our defined sampling budget (cf. line 4). If the queue is empty before the condition is reached, we randomly extract an unvisited node to restart the fire from it (cf. line 6). The minibatch is then the induced subgraph G' using the sampled nodes in V' (cf. line 16). $G' = (V', E', X_{V'}, X_{E'})$ where $E' = \{(v_i, v_j) \in V' \times V' | (v_i, v_j) \in E\}$.

Algorithm 1 Subgraph Forest Fire Sampler (*SFFS-GCN*)

Input: graph $G = (V, E)$; p the burning probability; s the number of seeds; S the size of the sampled subgraph.
Output: A subgraph $G' = (V', E')$ such that $|V'| = S$.

$V_{unvisited} \leftarrow V$, $V' \leftarrow \varnothing$, $E' \leftarrow \varnothing$ $V_{seeds} \leftarrow random\_extract(V_{unvisited}, s)$ $queue \leftarrow queue \cup V_{seeds}$ $(|V'|) < S$ $|queue| = 0$ $queue \leftarrow queue \cup random\_extract(V_{unvisited}, 1)$ $v_{target} \leftarrow random\_extract(queue, 1)$ $neighborhood \leftarrow get\_unvisited\_neighbors(v_{target})$ $percentage \leftarrow triangular(0, p, 1)$ $x \leftarrow percentage * |neighborhood|$ $V_x \leftarrow random\_select(neighborhood, x)$ $V' \leftarrow V' \cup \{v_{target}\}$, $queue \leftarrow queue \cup V_x$ $V_{unvisited} \leftarrow V_{unvisited} \setminus V_x$ $G' \leftarrow induced\_subgraph(G, V')$

We can easily see that our sampling procedure becomes equivalent to:

- A random node sampling when the burning probability is too small,
- A breath first sampling (BFS) if the burning probability is too high,
- A snowball sampling, if we fix the number of selected neighbors to an integer $k > 2$ instead of sampling it from a probabilistic distribution, and
- A random walk sampling if this number is equal to 1, i.e., $k = 1$.

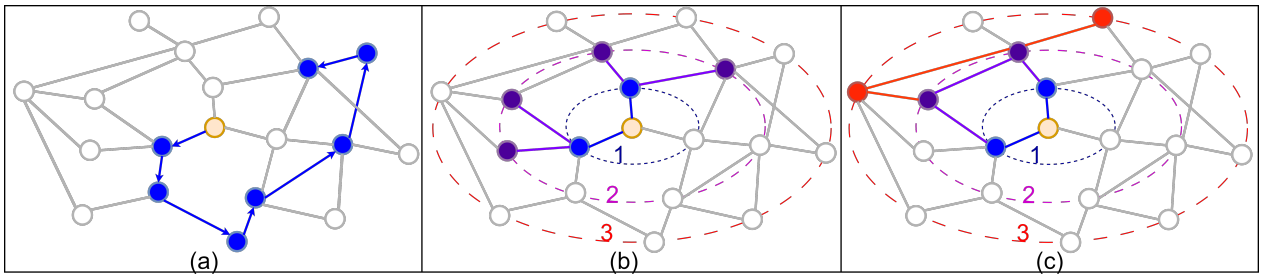

Figure 1: Illustration of *SFFS-CGN*. We consider sampling a seven nodes subgraph. All strategies begin by selecting a target or a seed node. The selection is usually randomly done. (a) is a random walk based strategy where a random walk is performed starting from the seed node. Depending on the graph structure, the walk can be trapped, visit only close neighbors, or get lost far from the starting point. In (b) and (c), the sampling expands in successive hops following neighbor connections (blue circle is 1-hop, purple circle is 2-hop, and red circle is 3-hop). In (b), a snowball strategy is adopted with the number of neighbors to be selected fixed to two neighbors. With a budget of seven nodes, the exploration is very restricted and stops at the second circle. While in (c) corresponds to *SFFS-CGN* sampling. The flexibility of the number of selected neighbors at each step allows to visit nodes both locally and in depth. Besides, the connectivity of the induced subgraph is better than the one we get by random walk sampling.

Random node sampling gives the same probability for nodes to be sampled, but it jeopardizes the subgraph connectivity. BFS gives highly connected subgraphs, but the sampling is biased towards high degree nodes and the diversity of the sampled subgraphs is quite poor since the selected nodes become similar when the starting seed set and the desired number of nodes is fixed. Snowball and random walk sampling demonstrate better results when it comes to randomness and connectivity. But the sampling is biased and if we refer to the study realized in [19], these sampling

methods do not do well preserving the graph properties. Figure 1 illustrates how *SFFS-GCN* sampling behaves when compared to these baseline sampling methods.

3.2 Normalization

A sampling based on the graph structure is the best choice to have well connected subgraphs. However, it is prone to biasing as some nodes will be selected more often than others due to degree distribution. This bias negatively impacts the learning process as much as a biased classification dataset would do. One way to reduce this bias, is to study the sampling distribution and use the formalized probabilities to normalize the message passing and consequently moderate each node's participation. Authors in [17] argue that $\alpha_{i,j} = \frac{p_{j,i}}{p_i}$ is a good normalization factor where $p_{i,j} = p_{j,i}$ is the probability of an edge $(v_j, v_i) \in E$ being sampled in a subgraph, and p_i is the probability of a node $v_i \in V$ being sampled. This normalization factor is used in the message passing step as follows:

$$h_i^{(l)} = \sigma[W^{(l)} \sum_{v_j \in N(i) \cup \{v_i\}} \frac{p_i}{p_{j,i}} (d_{ii}.d_{jj})^{\frac{-1}{2}} h_j^{(l-1)}] \tag{2}$$

The values $p_{j,i} and p_i$ can also be retrieved statistically with a nodes and edges count over a series of samples following the sampling strategy for which we want to find $p_{j,i}$ and p_i values. In other words, we run our sampling strategy k times obtaining k subgraphs on which we count nodes and edges occurrences. The obtained values are used to estimate $\alpha_{i,j}$. For more details, we refer the reader to the work done in *Graphsaint* [17]. This option may seem as a preprocessing overhead. However, in our case, the k subgraphs will be re-used in the training process.

3.3 *SFFS-GCN* Training Process

SFFS-GCN as a sampling framework can be used to enhance any GCN model. In order to integrate it in the learning process, we follow the steps described in algorithm 2. We use our sampler to generate a set of subgraphes or minibatches. This set is used to perform the needed statistics in order to compute the normalization parameters. And then, for each training epoch, we iterate through the minibatches, calculate the loss. The loss is used by the Stochastic gradient descent algorithm to backpropagate and update the learnable weights of the GCN.

Algorithm 2 GCN SFFS training

Input:
- A GCN model gcn to train;
- graph $G = (V, E, X_V, X_E)$;
- parameters p, s, and S for SFFS-GCN;
- B number of minibatches;
- e number of training epochs;

Output: trained gcn

run *FFS-GCN* on G to sample k subgraphs in order to compute the normalization factors $epoch \in [1, e]$ $batch \in [1, B]$ Run gcn on G', one of the sampled subgraphs Compute the loss l Normalize l backpropagate

4 Experiments

To show the effectiveness and efficiency of *SFFS-GCN* sampling framework, we conducted extensive experiments on real-world datasets. In this section, we first describe the datasets we use in our experiments, and then present the experimental settings. Finally, we discuss the obtained results.

4.1 Datasets

We test our subgraph forest fire sampling framework on 5 real-world datasets summarized in Table 1. The datasets vary from citation to social network, and from small to large graphs. Cora, CiteSeer, and PubMed are citation datasets. Flickr is built upon users' interaction via photo sharing on an image and video hosting website. Reddit is a collection of data that describes the relationships between users, posts, and communities on the website Reddit.

4.2 Experiment Settings

To show the effectiveness of our sampling framework in enhancing the GCN training, we use it in conjunction with the following two well known models that normally use the whole graph during training: GCN [8] described in the related work in Section 2 and GAT [9] which is an attention-based graph neural network usually used for node classification.

Table 1: Datasets

	Cora	Citeseer	Pubmed	Flickr	Reddit
# Nodes	2708	3327	19717	89250	232965
# Edges	5429	4732	44338	899756	11606919
# Features	1433	3703	500	500	602
# classes	7	6	3	7	41

Table 2: Test accuracy results on all datasets

	Cora	Citeseer	Pubmed	Flickr	Reddit
GraphSAGE	76.11%	66.92%	76.33	46.13%	93.27%
Cluster-GCN	69.17%	63.51%	80.56%	47.44%	95.07%
GCN	80.92%	71.23%	78.60%	46.55%	92.68%
SFFS-GCN$_{Low}$	76.56%	69.93%	74.27%	46.45%	85.66%
SFFS-GCN$_{High}$	79.35%	70.66%	78.23%	**47.09%**	83.74%
SFFS-GCN	**81.25%**	**73.98%**	**80.22%**	46.97%	**93.86%**
GAT	**82.94%**	72.56%	78.78%	*	*
SFFS-GAT$_{Low}$	78.31%	68.24%	75.11%	44.36%	74.45%
SFFS-GAT$_{High}$	78.66%	66.17%	73.87%	45.73%	79.89%
SFFS-GAT	80.64%	**73.27%**	**81.65%**	**47.77%**	**85.12%**

"*" memory capacity out of reach.

We base our experiments on the Pytorch Geometric [15] implementation of the two models. The reference models are a succession of two layers with the hidden size layer of 32 for Cora, Citeseer, and Pubmed; 64 for Flickr; and 128 for Reddit. We adopt the following learning hyperparameters : a dropout of 0.5 and 0.6 for GCN [8] and GAT [9] respectively; a learning rate of $0.01, 0.001$, and 0.005 for the citation datasets, Flickr and Reddit respectively; We set weight decay to 5×10^{-4} for both models except for the second layer of the GCN [8] for which we set it to 0. We then use these same models and integrate our subgraph forest fire sampling framework for a minibatches training and we denote them by *SFFS-GCN* and *SFFS-GAT*.

We compare these modified models with the original versions of *GCN* [8] and *GAT* [9] that do not use any sampling. We also compare with two other known models *GraphSAGE* [12] and *Cluster-GCN* [16] which both have their own sampling techniques described in Section 2.

For a fair comparison, we set the number of clusters in *Cluster-GCN* [16] so as to make the average subgraph size the same as ours as well as Graphsaint's [17].

We use the two new models *SFFS-GCN* and *SFFS-GAT* with different burning probabilities:

- A very small burning probability. This corresponds to a random node sampling. We denote this version with the subscript Low (SFFS-GCN$_{Low}$ and SFFS-GAT$_{Low}$).
- A very high value for the burning probability. This corresponds to a BFS node sampling. We denote this version with the subscript $High$ (SFFS-GCN$_{High}$ and SFFS-GAT$_{High}$).
- A moderate burning probability. *SFFS-GCN* and *SFFS-GAT* without subscripts correspond to this case.

We set the size of the sampled subgraphs to $S = 1000$ for Cora and Citeseer datasets, to $S = 2000$ for Pubmed and Flickr, and to $S = 6000$ for Reddit.

4.3 Results and Analysis

4.3.1 Task Performance Analysis

We trained and tested all the above models on a node classification task. We report the accuracies in Table 2.

We can first observe that using our framework with small or high burning probabilities (cf. the performance of *SFFS-GAT*$_{Low}$, *SFFS-GAT*$_{High}$, *SFFS-GCN*$_{Low}$, *SFFS-GCN*$_{High}$) is not interesting. The medium burning probability achieves the best results. In fact, with a low burning probability, the model is completely random. With a high burning probability, the model is a BFS sampler that focuses on connectivity too much. The medium burning probability captures both randomness and connectivity and is a good compromise.

We can also clearly see that *SFFS-GAT* and *SFFS-GCN*, the two frameworks that integrate our sampling process outperform plain *GCN* [8] and *GAT* [9] in most of the cases. For the GCN [8] model, *SFFS-GCN* gives better overall

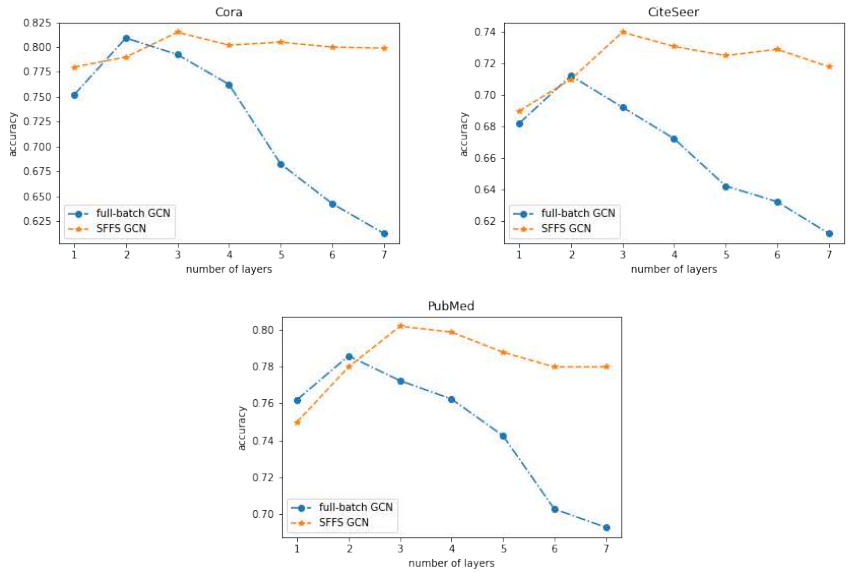

Figure 2: Effect of the number of layers on GCN training

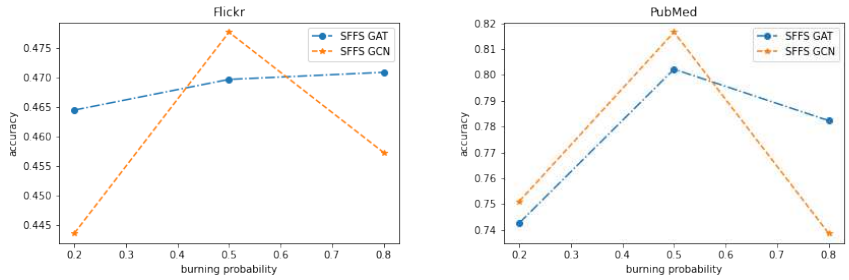

Figure 3: Effect of the burning probability on GCN training

performance than both *GraphSAGE* and *Cluster-GCN*. In the case of GAT [9] model, the subgraph sampling allows it to run and highly perform on datasets that would not fit in the GPU (e.g., on Flickr) or the CPU otherwise (e.g., on Reddit).

4.3.2 For Deeper Graph Networks

We also tested our GCN framework model with various numbers of layers. The results over Cora, CiteSeer and PubMed datasets are shown in Figure 2. The authors in [24] showed that GCN performance starts dropping when the depth goes beyond 2 layers. Figure 2 confirms it. GCN reaches its peak performance at 2 layers depth and starts dropping afterwards. This drop is caused by the *oversmoothing* problem. On the other hand, integrating *SFFS* allows *SFFS-GCN* to withhold the dropping despite the model's depth. Which proves its usefulness against *oversmoothing*.

4.3.3 Impact of the Burning Probability on GCN Training

As we discussed in the previous section, when varying the burning probability, we can make *SFFS-GCN* simulate either a BFS or a random walk behavior. Figure 3 plots the accuracy of the GCN when varying the burning probability. We can notice that a moderated burning probability is the best choice. A moderated value means that the percentage of burned neighbors is mostly around 50% without excluding the possibility of lower or higher percentage to be adopted in some cases. In fact, considering an interval of values around 50% proved itself effective since it gives better results than a sampling with a fixed 50% burning percentage according to our tests. It can be explained by the fact that a flexible percentage allows the subgraphs to be diverse. The generated subgraphs cover both local and far neighborhoods. This mix is most relevant in a social network like Flickr where interest similarities of nodes are found locally and role similarities can be found far away [4]. We also notice that the GAT model limits the accuracy drop as the burning probability goes higher. A higher burning probability means that the sampling behavior is closer to BFS. Which means that every node has a high number of neighbors with which it exchanges messages for the update

operation. This scenario increases the likelihood of noise being passed on and thus the accuracy dropping. Except that the attention mechanism in GAT helps to focus the relevant neighbors which moderates the accuracy drop.

5 Conclusion

Sampling is a technique that can be used to significantly reduce the computational cost of learning on large graphs, making it possible to apply graph neural networks to larger and more complex problems. Sampling can be performed in a number of different ways, depending on the specific problem and the desired properties of the sample. Thus, it is important to carefully design the sampling strategy to ensure that the sample is representative of the overall graph and does not introduce bias or distortion. Subgraph Forest Fire Sampling, is an effective and reliable sampling method that generates subgraphs for GCN minibatch training. The subgraphs reflect the original graph properties and allow GCNs to go deeper and help them better generalize their learning.

Acknowledgment

For the research leading to these results, the authors received funding from Agence Nationale de la Recherche (ANR) under Grant Agreement No ANR-20-CE23-0002.
All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

[1] Fout, A., Byrd, J., Shariat, B., Ben-Hur, A.: Protein interface prediction using graph convolutional networks. Advances in neural information processing systems **30** (2017)

[2] Guo, S., Lin, Y., Feng, N., Song, C., Wan, H.: Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 922–929 (2019)

[3] Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: Macskassy, S.A., Perlich, C., Leskovec, J., Wang, W., Ghani, R. (eds.) The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014, pp. 701–710 (2014). https://doi.org/10.1145/2623330.2623732

[4] Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Krishnapuram, B., Shah, M., Smola, A.J., Aggarwal, C.C., Shen, D., Rastogi, R. (eds.) Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, pp. 855–864 (2016). https://doi.org/10.1145/2939672.2939754

[5] Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: Bengio, Y., LeCun, Y. (eds.) 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings (2013). http://arxiv.org/abs/1301.3781

[6] Micheli, A.: Neural network for graphs: A contextual constructive approach. IEEE Trans. Neural Networks **20**(3), 498–511 (2009). https://doi.org/10.1109/TNN.2008.2010350

[7] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Networks **20**(1), 61–80 (2009). https://doi.org/10.1109/TNN.2008.2005605

[8] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings (2017). https://openreview.net/forum?id=SJU4ayYgl

[9] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. CoRR **abs/1710.10903** (2017) https://arxiv.org/abs/1710.109031710.10903

[10] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 5449–5458 (2018)

[11] Chen, J., Ma, T., Xiao, C.: Fastgcn: Fast learning with graph convolutional networks via importance sampling. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings (2018). https://openreview.net/forum?id=rytstxWAW

[12] Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. NIPS'17, pp. 1025–1035. Curran Associates Inc., Red Hook, NY, USA (2017)

[13] Li, G., Müller, M., Thabet, A.K., Ghanem, B.: Deepgcns: Can gcns go as deep as cnns? In: 2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019, pp. 9266–9275 (2019). https://doi.org/10.1109/ICCV.2019.00936

[14] Wang, X., Zhang, M.: How powerful are spectral graph neural networks. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., Sabato, S. (eds.) International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA. Proceedings of Machine Learning Research, vol. 162, pp. 23341–23362 (2022). https://proceedings.mlr.press/v162/wang22am.html

[15] Fey, M., Lenssen, J.E.: Fast graph representation learning with pytorch geometric. CoRR **abs/1903.02428** (2019) https://arxiv.org/abs/1903.024281903.02428

[16] Chiang, W., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, C.: Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In: Teredesai, A., Kumar, V., Li, Y., Rosales, R., Terzi, E., Karypis, G. (eds.) Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019, pp. 257–266 (2019). https://doi.org/10.1145/3292500.3330925

[17] Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.K.: Graphsaint: Graph sampling based inductive learning method. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020 (2020). https://openreview.net/forum?id=BJe8pkHFwS

[18] Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.K.: Accurate, efficient and scalable training of graph neural networks. J. Parallel Distributed Comput. **147**, 166–183 (2021). https://doi.org/10.1016/j.jpdc.2020.08.011

[19] Leskovec, J., Kleinberg, J.M., Faloutsos, C.: Graphs over time: densification laws, shrinking diameters and possible explanations. In: Grossman, R., Bayardo, R.J., Bennett, K.P. (eds.) Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005, pp. 177–187 (2005). https://doi.org/10.1145/1081870.1081893

[20] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: Guo, Y., Farooq, F. (eds.) Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018, pp. 974–983 (2018). https://doi.org/10.1145/3219819.3219890

[21] Huang, W., Zhang, T., Rong, Y., Huang, J.: Adaptive sampling towards fast graph representation learning. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pp. 4563–4572 (2018). https://proceedings.neurips.cc/paper/2018/hash/01eee509ee2f68dc6014898c309e86bf-Abstract.html

[22] Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., Gu, Q.: Layer-dependent importance sampling for training deep and large graph convolutional networks. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp. 11247–11256 (2019)

[23] Zeng, H., Zhang, M., Xia, Y., Srivastava, A., Kannan, R., Prasanna, V.K., Jin, L., Malevich, A., Chen, R.: Deep graph neural networks with shallow subgraph samplers. CoRR **abs/2012.01380** (2020) https://arxiv.org/abs/2012.013802012.01380

[24] Dwivedi, V.P., Joshi, C.K., Laurent, T., Bengio, Y., Bresson, X.: Benchmarking graph neural networks. CoRR **abs/2003.00982** (2020) https://arxiv.org/abs/2003.009822003.00982