Dynamic adjusting of neighbourhood sample size with GRAPES

Jakub Frac[1]

Vrije Universiteit Amsterdam

Abstract. Graph neural networks (GNNs) utilize the concept of aggregating information from neighboring nodes to learn node representations within a graph. However, as these networks become deeper, their receptive field — the range of neighborhood information they aggregate — exponentially increases, leading to significant memory requirements. To address this, graph sampling techniques have been developed to reduce memory consumption by selecting a subset of nodes from the graph, enabling GNNs to handle larger graphs efficiently. Traditional sampling techniques often rely on static heuristics that might not be effective across various graph structures or tasks. To overcome this limitation, GRAPES was proposed, an adaptive graph sampling strategy that dynamically selects important nodes critical for training GNN classifiers. GRAPES leverages a GFlowNet to dynamically adjust node sampling probabilities based on the specific goals of classification. To further develop this technique, we propose dynamic adjusting of sample size regulated by a hyperparameter, based on models confidence about certain nodes. It shows similar or worse performance to original GRAPES on a subset of original datasets. The code is available at: `github.com/fr30/grapes-reproduction`

1 Introduction

Graph sampling addresses the scalability issue in Graph Neural Networks (GNNs) by selecting a subset of nodes from the graph without considering their features or the specific training task.

The majority of existing graph sampling techniques prioritize mirroring the behavior of a fully-batched GNN, using an indirect strategy to achieve high accuracy ([2] [3] [4] [5]).

Younesian et al. [1] introduced an adaptive sampling strategy that allows the model to adjust to the task at hand, selectively sampling nodes that contribute to a highly accurate GNN performance. A notable aspect of this method is the use of a hyperparameter to control the size of the neighborhood sampling, which remains constant during the training phase.

We claim that the sampling model has the capability to identify the nodes that are crucial for achieving high accuracy and to filter out those that contribute only noise. As a result, we propose a method that selects a subset of sampled nodes based on the model's confidence in its selections.

2 Background

2.1 Graph Convolution Networks

Consider an undirected graph $GC = (V, E)$, comprising a set of N nodes V and edges E. The adjacency matrix A of 0s and 1s, signifies the existence of links between node pairs. The row-normalized adjacency matrix with self-loops, denoted by $\hat{A} = \tilde{D}^{-1}\tilde{A}$, is obtained by adding self-loops to A (resulting in $\tilde{A} = A+I$) and then normalizing by $\tilde{D}$, the degree matrix of $\tilde{A}$. The original paper [1] focused on the Graph Convolutional Network (GCN) model as described by Kipf ([**?**]), so we are going to use the same methodology.

The feature matrix X represents the features associated with the nodes, while Y contains the labels for a subset of nodes $V_t \subseteq V$ that are labeled.

The computation at the l-th layer of a GCN is defined as $H^l = \sigma(\tilde{A}H^{l-1}W^l)$, where W^l is weight matrix for the l-th layer, σ represents a nonlinear activation function, and the output represents the number of target classes. For an individual node $v_i \in V$, this equation describes the update mechanism:

$$h^l_{v_i} = \sigma\left(\sum_{v_j \in \mathcal{N}(v_i)} \hat{A}_{v_i,v_j} h^{l-1}_{v_j} W^l\right)$$

where $\mathcal{N}(v_i)$ is the set of v_i's neighbors, and $\hat{A}_{v_i,v_j}$ refers to the element (v_i, v_j) of $\hat{A}$.

With an increase in the number of layers, the neighborhood's size experiences exponential growth as the layer count rises. Previous works aim to mitigate this exponential increase by exploring graph sampling techniques. A particular focus is given to layer-wise sampling. Initially, target nodes are segmented into mini-batches of size b. Subsequently, at each layer, k nodes are selected from the neighbors of nodes from the preceding layer. Thus, the approximation for the l-th layer update of node v_i can be represented as:

$$\tilde{h}^l_{v_i} = \sigma\left(\sum_{v_j \in K_l} \hat{A}'^l_{v_i,v_j} \tilde{h}^{l-1}_{v_j} W^l\right)$$

where $K_l \subseteq \mathcal{N}(K_{l-1})$ denotes the set of nodes sampled in layer l, and $\mathcal{N}(K_{l-1})$ indicates the neighbors of the nodes in K_{l-1}. K_0 is the initial mini-batch of target nodes and $\hat{A}'^l(v_i, v_j)$ represents the row-normalized value of the sampled adjacency matrix A'^l for layer l, where $A'^l(v_i, v_j) = 1$ if v_j is sampled, i.e., $v_j \in K_l$, and 0 otherwise.

In the previous works ([1], [3], [4], [2], [5]) $\forall_l |K_l - K_{l-1}| = k$, meaning that k nodes are additionally sampled for each layer. In our extension this number changes across layers, but is bounded between $[1, k]$.

2.2 Generative Flow Networks

Define a GFlowNet learning scenario as a tuple $GF = (S, A, S_0, S_f, R)$, where S represents a finite collection of states forming a directed graph connected by A, a set of directed edges symbolizing actions or transitions among states. The subset $S_0 \subset S$ consists of initial states, $S_f \subset S$ dfeines the terminating states, and $R : S_f \to \mathbb{R}^+$ is the reward function applied to terminating states. At any given time t, an action $a_t \in A$ facilitates the transition from state s_t to s_{t+1}. A trajectory τ defines a sequence from an initial state s_0 to a terminating state $s_n \in S_f$, expressed as $\tau = (s_0 \to \ldots \to s_n)$.

In this framework, a GFlowNet is tasked with learning transitions from s_0 to a rewarding terminating state $R(s_n)$. The objective is to model the forward transition probabilities $P_F(s_{t+1}|s_t)$ to reflect the provided rewards, as suggested by [6]. The Trajectory Balance (TB) loss, formulated by [7], is introduced. For any given trajectory $\tau = (s_0 \to \ldots \to s_n)$ loss is computed as follows:

$$L_{TB}(\tau) = \left(\log Z(s_0) - \log \frac{\prod_{t=1}^{n} P_F(s_t|s_{t-1})}{\prod_{t=1}^{n} P_B(s_{t-1}|s_t)} R(s_n)\right)^2,$$

Here, $Z : S_0 \to \mathbb{R}^+$ estimates the network's total flow from the initial state s_0, with P_F and P_B representing the neural network-parameterized forward and backward state transition probabilities, respectively ([7]).

2.3 GRAPES

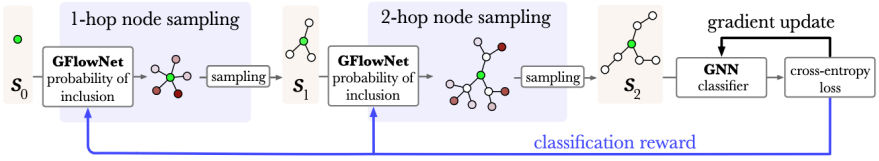

Fig. 1. A high-level schematic diagram of GRAPES from the original work [1].

Training

GRAPES [1] rephrased the problem of finding optimal neighbourhoods of size k as a reinforcement learning problem. In that case each state $s \in S$ is a sequence of sampled adjacency matrices $(A_0, \ldots, A_L)$, where $A_i \subset A$ and L is a number of sampling layers, also corresponding to the number of GCN layers.

The reward for each trajectory of length L is defined as:

$$R(s_L) = R(A_0, \ldots, A_L) = \exp\left(-\alpha \cdot L_{\text{GCNC}}(A_0, \ldots, A_L)\right)$$

Where L_{GCN} is a classification loss and α is a scaling parameter.

In the said scenario a deep neural network (in this caso also a GCN) will try to model the forward probability P_F and the network's total flow, which uniquely identifies a Markovian Flow Network [7].

More precisely, the GFlowNet is tasked with estimating the probability p_i, for each $i \in \{1, \ldots, n\}$, that a given node v_i within the neighborhood of K_l will be selected for inclusion in V_{l+1}, the set of nodes sampled for the subsequent layer $l + 1$.

This selection probability is modeled using a Bernoulli distribution. Consequently, the forward transition probability from layer l to layer $l + 1$ is given by:

$$P_F(s_{l+1}|s_l) = \prod_{v_i \in V_{l+1}} p_i \prod_{v_i \in \mathcal{N}(K_l) \setminus V_{l+1}} (1 - p_i)$$

When it comes to approximating backward probability P_B, GRAPES [1] uses a trick of concatenating indicator features to previous layer's embeddings, such that for each trajectory there is only one path to go from terminating state to the start state. This, in turn, transforms GFlowNet graph from a DAG into a directed tree.

As a result, the backward probability turns to constant $P_B = 1$ and does not require any modelling.

Sampling

In the original work [1] k nodes are aimed to be sampled without replacement. However, the forward probability distribution P_F consists of n independent Bernoulli trials, making the exact sampling of k nodes from this distribution unlikely.

Instead, the Gumbel-max trick [8] is utilized, whether a set of nodes V_l^k is selected by perturbing the log-probabilities randomly and selecting the top-k values:

$$V_l^k = \text{top}_k(\log p_1 + G_1, \ldots, \log p_n + G_n), \quad G_i \sim \text{Gumbel}(0, 1)$$

3 Varying sample size

It can be observed that sometimes smaller sample size yield better results than larger ones [3], [1]. We hypothesize, that large neighbourhood for each node introduces unnecessary noise. We can also observe a huge drop in performance when models are evaluated in the same way as they were trained - in minibatches instead of full batch.

Several methods tried to mitigate this problem by reducing variance in sampled neighbourhoods [4]. Intuitively, one can also assume that GRAPES [1] would choose optimal neighbourhoods, reducing the information-to-noise ratio. However, with a fixed sample size each of those methods is forced to choose some nodes that introduce the harmful noise.

To mitigate that, we introduce a procedure that allows model to discard the nodes that have low probabilites of being included during the sampling procedure. Namely, knowing that P_F models Bernoulli distribution for each node being included in the next layer, before the sampling occurs we will remove the nodes with small probability of inclusion.

The new set of nodes considered during sampling will be defined as follows:

$$V_i' = \{v_i : v_i \in V^i \quad \text{and} \quad p_i > \beta\}$$

Where β is a **cutoff** parameter defined by the user.

During the initial states of training the model obviously can not be sure whether to include a node or not, as the GFlowNet has not modeled the probability distribution yet. Therefore, if V_i' is an empty set, we use V_i instead.

4 Experiments

4.1 Experimental setup

We wanted to use the same experimental setup as in the original work [1]. Namely, we will use two layers of GCN with ReLU activation function for both classification and GFlowNet networks. The models are evaluated on a small subset of original GRAPES datasets for node classification task: Cora, Citeseer and Pubmed.

4.2 Evaluation protocol

To provide a fair comparison, we will use fine-tuned GRAPES from the source code provided by the authors [1]. We will introduce a change, such that testing is conducted in minibatch fashion and compare the final accuracies for both models.

As for our method, it requires a cutoff parameter β to be tuned, and through experiments we found out that the best results yields $\beta = 0.9$.

On top of that we will also compare both methods against uniform random sampling, which will serve as a sort of baseline. Basically the methods samples nodes from outermost layers in a uniform fashion - each node has the same probability of being sampled.

For each of the models we will also seperately finetune the number of training epochs for each dataset, to provide the fairest comparison.

4.3 Results

We present accuracy scores for all three models, two configurations for each one in the table 4.3. As we can see, our method does not manage to reliably outperform GRAPES or even uniform sampling.

The biggest difference can be seen when all models are evaluated on dataset Cora, with GRAPES providing the biggest increase in accuracy, while our method

performs similarly to the baseline. Also in case of Pubmed, GRAPES seems to outperform all the other methods by a small margin.

What's more interesting, is that uniform sampler seems to perform quite well or even comparably well to both GRAPES and our methods in some scenarios. In case of Citeseer none of the models seems to have a strong upper hand.

Generally our sampling method does not seem to provide any useful advantage over the original original work (GRAPES).

Method	**Cora**	**Citeseer**	**Pubmed**
Uniform_{16}	75.30 ± 0.61	75.24 ± 0.47	87.75 ± 0.13
Uniform_{128}	73.21 ± 0.74	74.09 ± 0.55	88.03 ± 0.20
GRAPES_{16}	79.27 ± 0.82	75.98 ± 0.53	**89.04 ± 0.29**
GRAPES_{128}	**82.33 ± 0.38**	76.01 ± 0.48	88.73 ± 0.40
ours_{16}	74.77 ± 0.56	**76.19 ± 0.29**	87.82 ± 0.18
ours_{128}	74.27 ± 1.03	75.68 ± 0.59	87.89 ± 0.26

Table 1. Accuracy on Cora, Citeseer, and Pubmed datasets.

5 Further work

5.1 Sampling procedure

The idea of training the model with adaptable sample size for each neighbourhood sounds like an interesting area for further research. We have to admit that this particular implementation of it had a few rough edges - there is no solid theoretical reason to cut off nodes with small Bernoulli distribution parameters.

5.2 Planetoid datasets

Given the fact how well uniform sampling performs, perhaps it is worth investigating credibility of the Planetoid datasets - they manage to provide a proof of work that one's algorithm works at all, but they do not seem to provide a reliable proof of one method's advantage over the other one.

5.3 Minibatch vs Full batch testing gap

Plenty of works ([1], [3], [2], [5]) perform training with sampling and testing with full batch. Probably in many practical scenarios it is required for a model to have the same setup during inference as in training.

It is noteworthy that there exists a huge gap in accuracy between full batch and minibatch testing and perhaps those results should also be included in further works.

References

1. Taraneh Younesian, Thiviyan Thanapalasingam, Emile van Krieken, Daniel Daza, Peter Bloem. GRAPES: Learning to Sample Graphs for Scalable Graph Neural Networks.
2. Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In 6th International Conference on Learning Representations, ICLR 2018, Conference Track Proceedings. OpenReview.net, 2018b.
3. Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, pp. 11247–11256, 2019.
4. Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. Advances in neural information processing systems, 31, 2018.
5. Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph- SAINT: Graph sampling based inductive learning method. arXiv preprint arXiv:1907.04931, 2019.
6. Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. Advances in Neural Information Processing Systems, 34:27381–27394, 2021a
7. Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. arXiv preprint arXiv:2111.09266, 2021b
8. Iris AM Huijben, Wouter Kool, Max B Paulus, and Ruud JG Van Sloun. A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(2):1353–1371, 2022