# Faster Graph Convolutional Network on Large Scale Graph

**Sushmita Bhattacharya** 1215354543
**Tianbo Song** 1215244186
**Weiying Wang** 1215208410
**Tiankai Xie** 1215350500
**Kaiqi Zhao** 1215359301
**Shunchi Zhou** 1215153732
**Yaoxin Zhuo** 1215272110

## Abstract

Omnipresent graph data demands specialized machine learning based classification approach. Convolutional neural network has been widespread for datasets with hierarchical relationships; especially for image data. Concepts of convolution can also be applied to graphs. We can consider each node of the graph to be a feature vector with its adjacency matrix representing the pairwise relationship between other vertices. Experiments [1] has shown that, a simple architecture with two layers convolutional units with Relu non-linearity after the first layer and a softmax at the end of the second layer is rich enough to classify big graphs (Cora, Pubmed, Citeseer,Reddit). However, this approach is both CPU and memory intensive for dense graphs. Hence, Fast GCN [2] proposed to sample the vertices of the graph uniformly at each layer. Inspired by Fast GCN we experimented with several different types of weighted sampling techniques (five algorithms based on degree and PageRank). Results showed that we can achieve faster convergence as compared to FastGCN at a cost of a decreased (but comparable) accuracy numbers.

## 1 Introduction

Graphs are topological representations of the relationship between vertices and edges. A variety of real-world data have been managed and represented by graphs; e.g., social networks[3][4], physical systems[5][6], biological networks[7], knowledge graphs[8] and so on. Recently much efforts have been made to explore the potential insights hidden in graph data, i.e., node classification[9], link prediction[10], graph clustering[11], among many others.

Traditional graph-based analysis tasks have some problems that computation and space costs are very high[12][3]. In order to solve such problems, initially a efficient paradigm called graph embedding (GE) is provided. Specifically, in order to preserve the structure and content of the graph to the greatest extent, graph embedding (GE) converts graph data from high dimensional space to low dimensional space; then, the generated low-dimensional data is imported into downstream machine learning tasks as a feature[13]. After combining graph embedding (GE) with convolutional neural network (CNN)[4][2][14][15], graph convolutional network (GCN) is proposed. In CNN, to improve the learning ability and the performance of the model, shared weights and multi-layered structures are widely established[16]. Since graphs are the most typical structure of local connection, shared weights can be helpful to reduce the computation cost, and the multi-layer structure is the key for extracting the characteristics of various features[13]. Therefore, GCN is a good generalization of CNN for learning feature representations for graphs. GCN not only has the flexibility of graph embedding (GE), but also has advantages in both effectiveness and robustness.

Preprint. Under review.

The essential purpose of GCN is to extract the spatial features of graphs. In order to achieve this, the work over the past few years on graph-based convolutional network models can be divided into two categories. One is in the spectral domain, and the other is in the vertex domain (spatial domain).

The work on the spectral domain of GCN is build on spectral graph theory[17][18][19]. The main idea is to learn the properties of a graph via the eigenvectors of the adjacency matrix and Laplacian matrix associated with the graph. The limitation of this approach is that it can only learn feature representations of a fixed graph, since the a small change in a graph can affect the whole Fourier transform. So in this case, the learned spectral filters is not transferable.

Another stream of work in the vertex domain is mainly focus on finding the neighbors adjacent to each vertex so as to learn spatial features of graphs. Goyal[20] provides a comprehensive and structured analysis of various graph embedding techniques. They first introduce the embedding task and its challenges, as well as present three categories of approaches based on factorization methods, random walks, and deep learning. The methods based on factorization generate the embedding via matrix factorization. For instance, Roweis and Saul[21]introduce locally linear embedding (LLE), an unsupervised learning algorithm that computes low-dimensional, neighborhood-preserving embeddings of high-dimensional inputs; A novel factorization technique is proposed, which relies on partitioning a graph to minimize the number of neighboring vertices[22]; Cao adopts a random surfing model to capture graph structural information[23]; Mingdong Ou develops a graph embedding algorithm named High-Order Proximity preserved Embedding, which is scalable to preserve high-order proximities of large scale graphs and capable of capturing the asymmetric transitivity[24]. Another category consists of random walks based methods[25][26] that learn a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes. Tang[27] proposes method named "LINE" which optimizes a carefully designed objective function that preserves both the local and global network structures so as to be suitable for arbitrary types of information networks.

These approaches mentioned above solve the problem of embedding very large information networks into low-dimensional vector spaces while preserving the property and structure of large graphs. But they require to learn all nodes of the graph during training of the embedding, and these methods are transductive learning and cannot naturally generalize to unseen nodes. In order to solve this problem, recently there appear a few semi-supervised deep neural network models. Wang[28] first proposes a semi-supervised architecture that has multiple layers of non-linear functions to capture the highly non-linear network structure, and exploits the first-order and second-order proximity jointly to preserve the network structure. Furthermore, Kipf and Welling[1] also present an approach for semi-supervised classification by using an efficient layer-wise propagation rule. After that, GraphSAGE[15] is introduced, which learns a function that generates embedding by sampling and aggregating features from a node's local neighborhoods, instead of training individual embedding for every node of the graph.

However, the recursive neighborhood expansion across layers poses time and memory challenges for training with large, dense graphs. The memory bottleneck of deep learning based GCN is also acknowledged in GraphSAGE[15]. To relax the requirement of simultaneous availability of test data, FastGCN[2] is proposed, which interprets graph convolutions as integral transforms of embedding functions under probability measures. This is the most relevant work to our approach.

We find that FastGCN does the sampling with uniform distribution while ignoring the importance of individual nodes. In reality, nodes in the same network always have different weights. For instance, some users may have more influence than other users in the social network graph. Therefore, in order to solve the memory bottleneck of GCN and reduce the computational cost further, we propose a novel model named FasterGCN by using multiple versions of weighted sampling methods. Our sampling scheme is more efficient, leading to a substantial decrease in the gradient computation, as will be introduced in details in Section 3.2. Experimental results in Section 4 demonstrate that our proposed model FasterGCN can converge faster than FastGCN and other related methods, while classification accuracy is highly comparable.

The rest of the paper is organized as follows. In Section 2, we describe the problems we address in details. We develop our methodology in Section 3 and introduce the datasets and experiment settings in Section 4. Experimental results are reported and analyzed in section5. Finally, We conclude the paper in Section 6.

## 2 Problem description

In this paper, we focus on the feature representations for large, dense graphs in a low-computation-cost way. It aims to represent features of vertexes in the numerical vector space, where the characteristics and structure of graphs is preserved.

Traditionally, the concept of a convolutional filter for image pixels or a linear array of signals is used in convolutional neuron network(CNN). Aiming to learn feature representations, CNN essentially calculates the weighted sum of the central image pixels and the adjacent image pixels by leveraging a kernel with shared parameters. Although CNN has showed great efficiency in extracting spatial features and has been applied to images, speeches, texts, and drug discovery problems (Atlas et al.,1988; LeCun et al., 1989; 1998; 2015; Wallach et al., 2015), CNN can only process the data which is arranged in a Euclidean structure, like pixels of images.

For graph collections where discrete convolutions cannot maintain invariance and where the features of nodes are in the form of non Euclidean structure, such as social networks and information networks. So CNN is not efficient in learning spatial features of graph data, while graph convolutional network (GCN) has advantages in this field. Bringing the concept of convolution filters of CNN, GCN exploits the local and global connectivity structure of graphs as the filter to perform neighborhood mixing. The architecture can be indicated by the following equation[2]:

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)}), \tag{1}$$

where $\hat{A}$ is normalization of the normalized graph adjacency (similarity) matrix. $H^l$ is the embedding of the graph vertices in the $l$th layer. $W^l$ is the parameters matrix and $\sigma$ is the activation function like Sigmoid or ReLU.

One challenge we need to address is that the learned model from a limited set of training vertices is able to generalize well to any explanation of the graph including unseen nodes, given that new vertices may be added to the graph constantly and test data is not always available for many scenarios. For instance, a social network may increasingly expand new members, so is new products to recommend system and new articles for social news site.

The other more significant challenge for GCN is that the recursive neighborhood expansion across layers generates expensive computations and require a large amount of memory during training. Especially for large and dense graphs, neighborhoods of an individual vertex can occupy a large portion of the graph, so that a lot of data is involved and expensive computations is incurred even for a common mini-batch training with a small size.

## 3 Methodology

The notation used in this paper are introduced as follows: the matrices were annotated as bold uppercase letters like $\mathbf{X}$, vectors were annotated as lowercase letters like $\mathbf{x}$, scalars were annotated as lowercase letters like x. $\mathbf{I}_r$ means the $r \times r$ identity matrix. $\mathbf{1}$ means all ones vector and $\mathbf{0}$ means all zeros vector.

### 3.1 fastGCN

The graph convolutional layers contains spatial convolution operations on graph. Inspired by the idea of convolution filter on an image or text, GCN could use the information from the connectivity structure of the graph by performing the neighborhood mixing. In paper[1], the architecture shows as follows:

$$\hat{H}^{(l+1)} = \hat{A}H^lW^l, H^{(l+1)} = \sigma(\hat{H}^{(l+1)}), l = 0, ..., M-1, L = \frac{1}{n}\sum_{i=1}^{n} g(H^{(M)}(i,:)). \tag{2}$$

In equation 2, $\hat{A}$ is normalization of the graph adjacency (similarity) matrix. $H^l$ is the embedding of the graph vertices in the $l$th layer. $W^l$ is the parameters matrix and $\sigma$ is the activation function like Sigmoid. $M$ is the layer number of network and its value is 2 usually. $N$ is the number of nodes in the graph. $g$ is the expectation function with respect to the data distribution $D$.

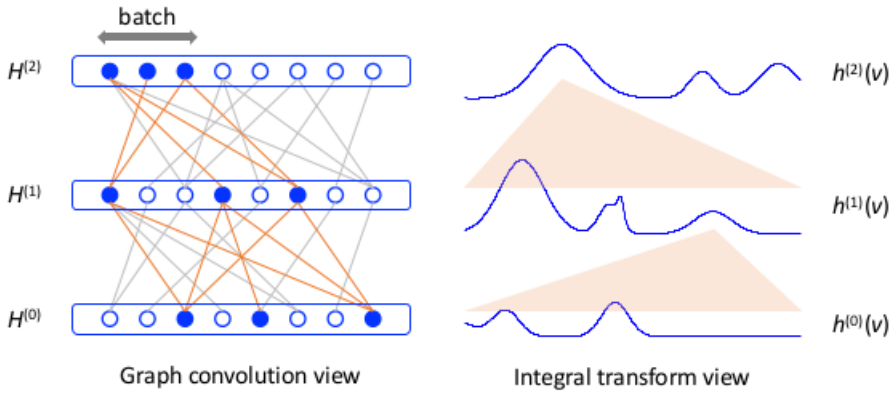


Figure 1: fastGCN figure

To resolve the problem of lack of independence caused by convolution, the authors interpret that each layer in the network defines an embedding function of the vertices that are tied to the same probability measure but are independent. The authors rewrite the equation 2 for the functional generalization.

$$\hat{h}^{(l+1)}(v) = \int \hat{A}(v,u)h^l(u)W^l dP(u), h^{(l+1)(v)} = \sigma(\hat{h}^{(l+1)(v)}), l = 0, ..., M-1 \tag{3}$$

$$L = \int g(h^{(M)}(v))dP(v). \tag{4}$$

Here, $u$ and $v$ are independent random variables, both of them have same probability measure $P$. $h^{(l)}$ is interpreted as the embedding function of $l$th layer. The authors used Figure 1 to illustrate the idea of fastGCN.

In Figure 1, it shows the two views of GCN. Each circle represents a graph vertex on the left view. Between two rows, a node $i$ is connected with node $j$ if two vertices in the graph are connected. A convolution layer uses the graph connectivity structure to mix the vertex features. The embedding function in the next layer is an integral transform (in orange are) on the right integral transform view. For the original fastGCN, all integrals are evaluated by Monte Carlo sampling and vertices were sub-sampled in each layer to approximate the convolution. Based on some theorems and formulations improvements. They proposed the fastGCN, see it details on Algorithm 1.

**Algorithm 1:** FastGCN batched training (one epoch), original version

**for** *each batch* **do**
  For each layer $l$, sample uniformly $t_l$ vertices $v_1^l, ..., v_t^l$
  **for** *each layer* $l$ **do**
    **if** $v$ *is sampled in the next layer,* **then**
      $\nabla \hat{H}^{l+1}(v,:) \leftarrow \frac{n}{t_l} \sum_{j=1}^{t_l} \hat{A}(v, u_j^{(l)}) \nabla\{H^l(u_j^{(l)},:)W^{(l)}\}$
    **end**
  **end**
  $W \leftarrow W - \eta \nabla L_{batch}$
**end**

### 3.2 FasterGCN

FastGCN, as described in Algorithm 1, samples nodes in each layer from an uniform distribution. However, nodes in the same network may have different importance. For example, some users who have more followers may have more influence than other users in a social network graph. Based on this idea, we proposed two weighted sampling methods.

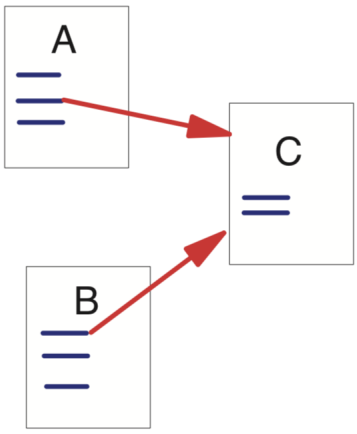


Figure 2: A and B are Backlinks of C and C is forward link of both A and B, $F_A = 1, F_B = 1, B_C = 2$

The first method uses the node's degree to represent its weight. We proposed three different sampling techniques based on the degree weight.

The second method treats the graph as a website network[29], and hence PageRank can represent the weight of each node. Let $v$ be a node in the graph and $F_v$ be the set of nodes that $v$ points to and $B_v$ be the set of nodes that point to $v$ and $N_v = |F_v|$ be the number of edges pointing outwards from $v$, with $c$ being a normalization factor. In our method, we treat the undirected graph as the bi-directional directed graph. Then simplified PageRank is defined as follows:

$$PR(v) = c \sum_{u \in B_v} \frac{PR(u)}{N_u}$$

For PageRank version, we only proposed the global probability and top-k ranking version. We could not experiment with the mix probability version because of the memory limitation of calculating the sub-graph's PageRank. The proposed algorithms are given below.

**Algorithm 2:** FasterGCN batched training (one epoch), degree global probability version

For each vertex $u$, compute sampling probability $q(u) \propto A(u,:)$ (adjacency matrix)
**for** *each batch* **do**
  For each layer $l$, sample $t_l$ vertices $v_1^l$, ... , $v_t^l$ according to distribution $q$
  **for** *each layer* $l$ **do**
    **if** $v$ *is sampled in the next layer,* **then**
      $\nabla \hat{H}^{l+1}(v,:) \leftarrow \frac{n}{t_l} \sum_{j=1}^{t_l} \hat{A}(v, u_j^{(l)}) \nabla\{H^l(u_j^{(l)},:) W^{(l)}\}$
    **end**
  **end**
  $W \leftarrow W - \eta \nabla L_{batch}$
**end**

**Algorithm 3:** FasterGCN batched training (one epoch), degree mix probability version

For each vertex $u$, compute sampling probability $q(u) \propto A(u,:)$ (adjacency matrix)
**for** *each batch* **do**
  Generated subgraph $G'$ using batch's vertices and edges
  Calculate the local batch probability $p(u) \propto A'(u,:)$
  For each layer $l$, sample $t_l$ vertices $v_1^l$, ... , $v_t^l$ according to distribution $\frac{p(u)+q(u)}{2}$
  **for** *each layer l* **do**
    **if** *$v$ is sampled in the next layer,* **then**
      $\nabla \hat{H}^{l+1}(v,:) \leftarrow \frac{n}{t_l} \sum_{j=1}^{t_l} \hat{A}(v, u_j^{(l)}) \nabla \{H^l(u_j^{(l)},:) W^{(l)}\}$
    **end**
  **end**
  $W \leftarrow W - \eta \nabla L_{batch}$
**end**

**Algorithm 4:** FasterGCN batched training (one epoch), degree top-k ranking version

For each vertex $u$, compute degree of each point $q(u) \propto A(u,:)$ (adjacency matrix)
**for** *each batch* **do**
  For each layer $l$, select highest $t_l$ vertices $v_1^l$, ... , $v_t^l$ according to degree $q$
  **for** *each layer l* **do**
    **if** *$v$ is sampled in the next layer,* **then**
      $\nabla \hat{H}^{l+1}(v,:) \leftarrow \frac{n}{t_l} \sum_{j=1}^{t_l} \hat{A}(v, u_j^{(l)}) \nabla \{H^l(u_j^{(l)},:) W^{(l)}\}$
    **end**
  **end**
  $W \leftarrow W - \eta \nabla L_{batch}$
**end**

**Algorithm 5:** FasterGCN batched training (one epoch), PageRank global probability version

For each vertex $u$, compute sampling probability $q(u) \propto PageRank(u)$
**for** *each batch* **do**
  For each layer $l$, sample $t_l$ vertices $v_1^l$, ... , $v_t^l$ according to distribution $q$
  **for** *each layer l* **do**
    **if** *$v$ is sampled in the next layer,* **then**
      $\nabla \hat{H}^{l+1}(v,:) \leftarrow \frac{n}{t_l} \sum_{j=1}^{t_l} \hat{A}(v, u_j^{(l)}) \nabla \{H^l(u_j^{(l)},:) W^{(l)}\}$
    **end**
  **end**
  $W \leftarrow W - \eta \nabla L_{batch}$
**end**

**Algorithm 6:** FasterGCN batched training (one epoch), PageRank ranking version

For each vertex $u$, compute degree of each point $q(u) \propto PageRank(u)$
**for** *each batch* **do**
  For each layer $l$, select highest $t_l$ vertices $v_1^l$, ... , $v_t^l$ according to PageRank $q$
  **for** *each layer l* **do**
    **if** *$v$ is sampled in the next layer,* **then**
      $\nabla \hat{H}^{l+1}(v,:) \leftarrow \frac{n}{t_l} \sum_{j=1}^{t_l} \hat{A}(v, u_j^{(l)}) \nabla \{H^l(u_j^{(l)},:) W^{(l)}\}$
    **end**
  **end**
  $W \leftarrow W - \eta \nabla L_{batch}$
**end**

# 4 Datasets and experiment settings

## 4.1 Dataset introduction

**Cora** The Cora dataset[30] consists of 2708 papers in the whole corpus in Machine Learning field which are classified into one of the following seven classes: Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning and Theory. These papers were selected in a way such that in the final corpus every paper cites or is cited by at least one other paper. The Cora dataset contains a vocabulary of size 1433 unique words after stemming and removing stop-words as well as all words with document frequency less than 10.

**Citeseer** The Citeseer dataset[31] consists of 3312 papers in the whole corpus which were selected in a way such that in the final corpus every paper cites or is cited by at least one other paper. All these papers are classified into one of the following six classes: Agents, AI, DB, IR, ML, HCI. The Citeseer dataset contains a vocabulary of size 3703 unique words after stemming and removing stop-words as well as all words with document frequency less than 10.

**Pubmed** The Pubmed data set[1] includes 19,717 medical-related statements selected from the PubMed article abstract and is labeled with links between relevant terms.The data accessible by PubMed can be mirrored locally using an unofficial tool such as MEDOC. There are 44,338 citation links between the nodes, for example, the relationship link between "treatment" and "cause and effect" .

**Reddit** The raddit data set[1] is a collection of 232,965 comments and posts from reddit users in the month of September, 2018. Some people built a post-post graph, 11,606,919 connections are made by same user comments both nodes,and Subreddits as the node label. W, and off-the-shelf 300-dimensional GloVe CommonCrawl word vectors[32]. For each post they concatenated the average of the embedding of post title, all post's comments, the post score, and comments number[15].

Table 1 is the summary of these four data sets.

Table 1: Dataset summary

| Dataset | Type | Nodes | Edges | Classes | Features |
|---|---|---|---|---|---|
| Citeseer | Citation network | 3327 | 4732 | 6 | 3703 |
| Cora | Citation network | 2708 | 5429 | 7 | 1433 |
| Pubmed | Citation network | 19717 | 44338 | 3 | 500 |
| Reddit | Social network | 232965 | 11606919 | 41 | 602 |

## 4.2 Experiment settings

For fair comparison, we used the same hyper-parameters settings for each data set. The training algorithm used early stopping (a regularization method to avoid over fitting by stopping once the validation loss is stable). See Table 2 for more details.

Table 2: Dataset hyper-parameters settings

| Dataset | K | learning rate | epochs | early stopping | Training/Validation/Test |
|---|---|---|---|---|---|
| Cora | 5 | 0.001 | 500 | 10 | 1208/500/1000 |
| Citeseer | 5 | 0.001 | 500 | 20 | 1827/500/1000 |
| Pubmed | 5 | 0.001 | 500 | 20 | 18217/500/1000 |
| Reddit | 5 | 0.001 | 500 | 20 | 152410/23699/55334 |

# 5 Results

For each data set, we run the baseline (fastGCN) and our five methods. To simplify the name, we named our five methods as follow: D-glob-prob (degree global probability version), D-ranking

(degree ranking version), D-mix-prob (degree mix probability version), PR-glob-prob (PageRank global probability version) and PR-ranking (PageRank ranking version).

Figure 3 to 6 show the validation loss vs training epochs of four data sets. Figure 7 shows the test accuracy of four data sets. Figure 8 and 9 contain the total training time of four data sets.

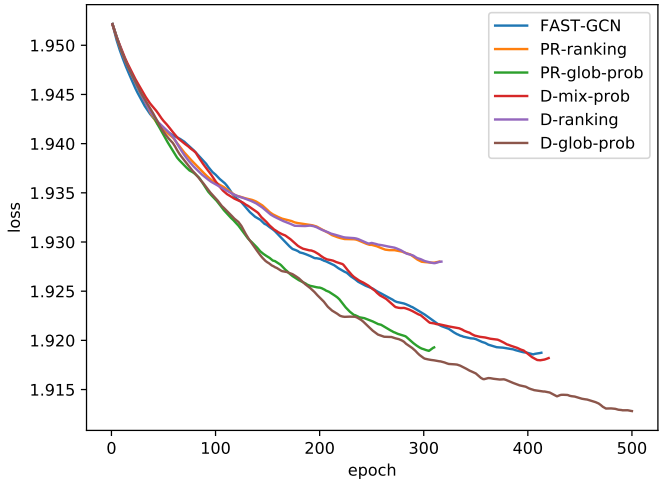


Figure 3: **Cora** validation loss vs training epochs. The PR-glob-prob and D-glob-prob converged before FastGCN. D-mix-prob took as long as fastGCN. But PR-ranking and D-ranking did not have good performance.

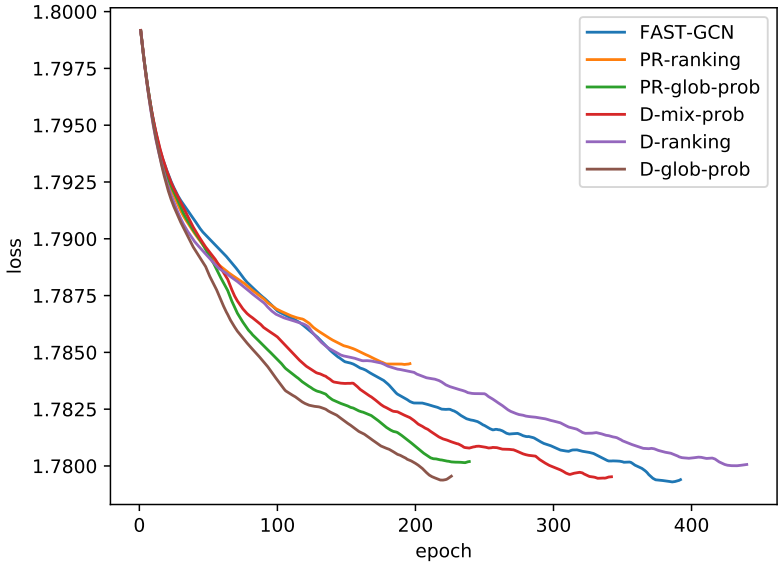


Figure 4: **Citeseer** validation loss vs training epochs. Three of the methods converged and stopped much before fastGCN. PR-ranking and D-glob-prob did especially good in terms of convergence.

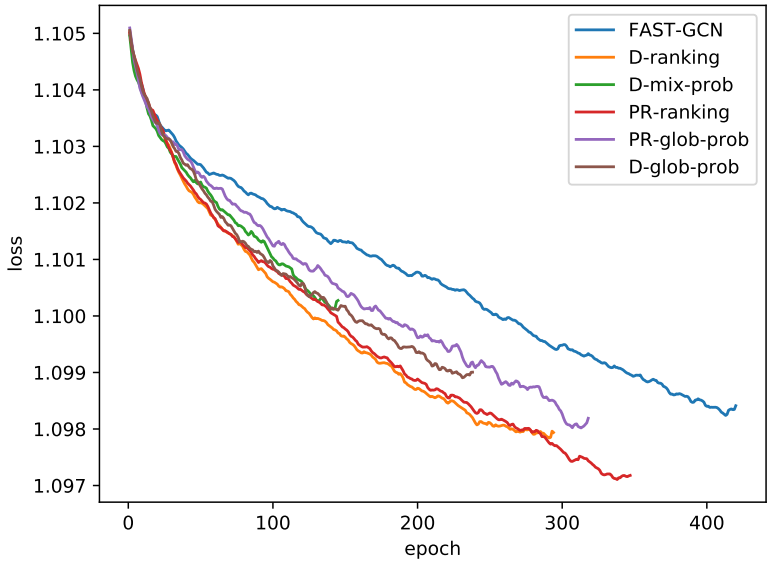


Figure 5: **Pubmed** validation loss vs training epochs. All five methods converged way before fastGCN in this medium sized dataset. D-ranking method particularly characterized the sub-graph degree very well on this data set.

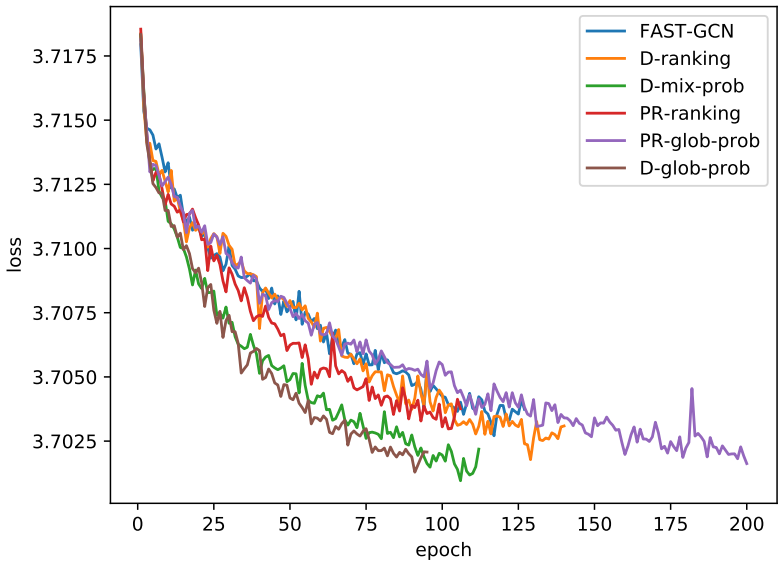


Figure 6: **Reddit** validation loss vs training epochs. PR-ranking, PR-glob-prob, D-ranking and D-mix-prob faster than fastGCN whereas D-glob-prob took especially long time as the dataset was huge and not all samples (based on global PageRank) mattered in an epoch and hence we see the oscillations in the validation loss. But once it stopped it had less loss than converged FastGCN.

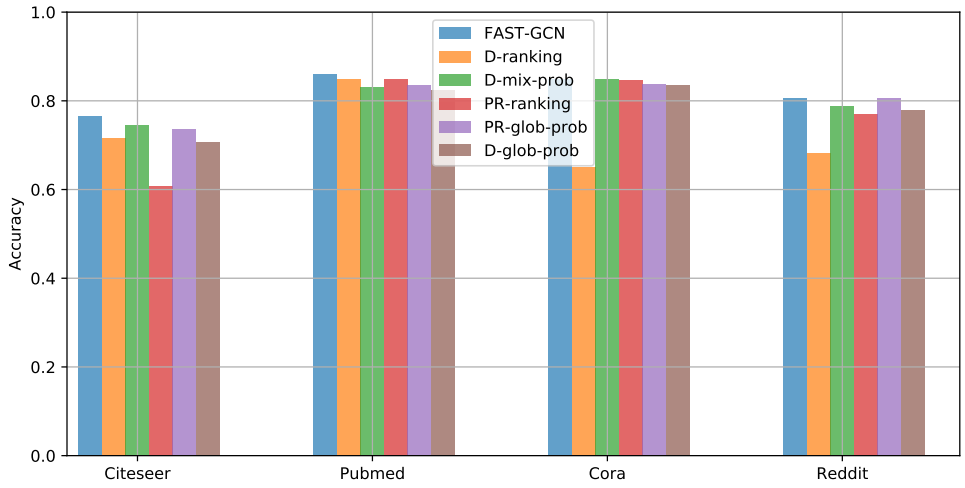


Figure 7: Test accuracy of four data sets. Due to the approximating nature of our sampling techniques we observed little degradation of test accuracy as compared to Fast GCN, but the numbers were comparable. Hence all these methods can potentially be used in larger graphs without much loss of accuracy.

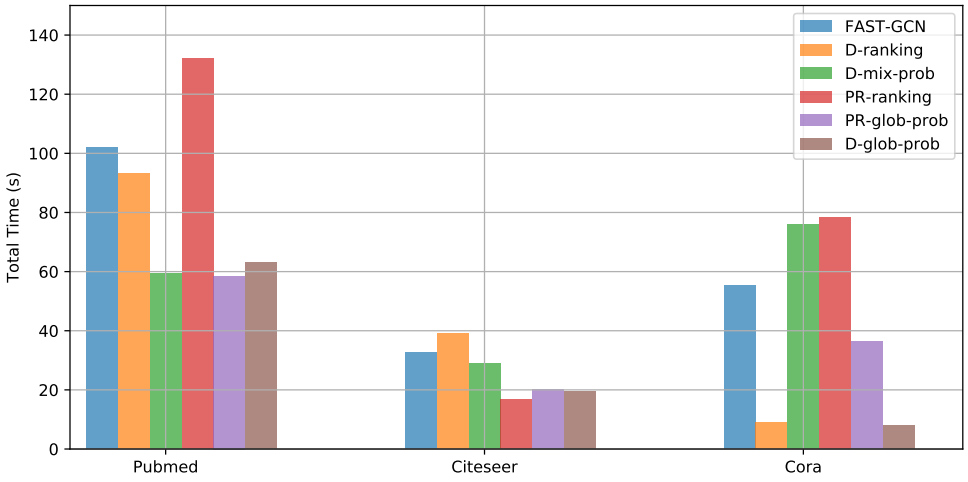


Figure 8: Total training time of cora, citeseer and pubmed data sets. We observed the training time for different techniques were similar as compared to FastGCN. This can be improved in future by optimizing our current implementation of python programs.

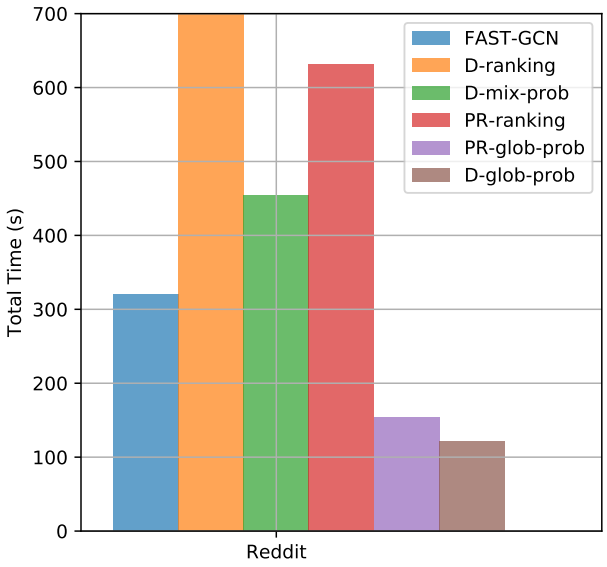


Figure 9: Total training time of reddit data set. This dataset was especially large and we had difficulty running it in our machines so we could not optimize our implementation and hence we see the increase in training time in three of the methods whereas we have significant improvement in PR-glob-prob and D-glob-prob approach even with our current implementation.

## 6 Conclusions and future work

In this work, we proposed different sampling methods to improve the convergence of validation loss over FastGCN, a fast improvement of the GCN model. Instead of the uniform sampling approach in FastGCN which didn't utilize the degree of different nodes, we encode the degree information into sampling process. To achieve that, we firstly present the sampling approaches based on degree of nodes. Then, we introduced the PageRank algorithm to extract more illustrative information for each node and save the normalized probability of each node offline. Besides, we apply top-k, global probability and global and local mix probability methods in our experiments. We tested the generalization capability of our sampling algorithm on dataset Cora, Citeseer, Pubmed and Reddit. The first three datasets are relatively small scale graph and the last one is a much larger scale graph which has 232965 nodes and more than ten millions of edges. We have compared the performance between our sampling method and FastGCN on all four datasets, under same hardware specification for each. By interpreting the result, our sampling method could reach convergence of validation loss faster than FastGCN and keep the accuracy of testing. In addition, not all proposed sampling approach could converge faster on all dataset. But for each dataset, some of our methods would reach convergence faster than baseline because of the variety of dataset's scale.

Benefit from the simple structure of original GCN, we could conveniently transfer many graph structure into it and train the convolutional neural network. In order to train the whole dataset and all nodes simultaneously, each layer of network is designed to be fully connected which is drastically computation heavy in original GCN. By introducing varies and effective sampling method, the training process would be boosted remarkably. As a consequence of not training all nodes in each layer, we have to compromise the loss of accuracy in exchange of training speed. In the future, in order to increase the accuracy and keep the same level of training time, we might need to find more substantial way to sample the points in order not to loose too much information during the process. Secondly, in our implementation, we used redundant loop to finish the sampling task which could be another burden that increase the training time. We could further optimize the code to reduce the serial operation time. Thirdly, since we only have access to dataset of FastGCN, we cannot validate our sampling method on larger scale graph dataset with million of nodes. Most processed and published

graph dataset don't have feature space for nodes but only the adjacent matrix. In future, we expect to conduct more experiments on larger scale graph datasets to validate our thoughts.

## References


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

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

[3] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

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

[5] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.

[6] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[7] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 6530–6539, 2017.

[8] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. *arXiv preprint arXiv:1706.05674*, 2017.

[9] Lisheng Qiao, Hongke Zhao, Xiaohui Huang, Kai Li, and Enhong Chen. A structure-enriched neural network for network embedding. *Expert Systems with Applications*, 117:300–311, 2019.

[10] D Liben. Nowell and j. kleinberg. *The link prediction problem for social networks. In CIKM*, 2003.

[11] Zheng Chen and Heng Ji. Graph-based clustering for computational linguistics: A survey. In *Proceedings of the 2010 workshop on Graph-based Methods for Natural Language Processing*, pages 1–9. Association for Computational Linguistics, 2010.

[12] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[13] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. Aligraph: A comprehensive graph neural network platform. *arXiv preprint arXiv:1902.08730*, 2019.

[14] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 387–396. ACM, 2017.

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

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[17] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[18] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[19] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[20] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[21] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[22] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48. ACM, 2013.

[23] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900. ACM, 2015.

[24] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114. ACM, 2016.

[25] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[26] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[27] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[28] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.

[29] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[30] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[31] Cornelia Caragea, Jian Wu, Alina Ciobanu, Kyle Williams, Juan Fernández-Ramírez, Hung-Hsuan Chen, Zhaohui Wu, and Lee Giles. Citeseer x: A scholarly big dataset. In *European Conference on Information Retrieval*, pages 311–322. Springer, 2014.

[32] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.