CZ4071 Network Science

Assignment 2 - Final Report

Survey and Implementation of Paper:

[7] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. "LayerDependent Importance Sampling for Training Deep and Large Graph Convolutional Networks". In NeurIPS 2019

Submitted by: Team 4

Eddy Lim Qing Yang	U1620115B	eddy0006@e.ntu.edu.sg
Shawn Kan Jung Tze	U1721495G	skan003@e.ntu.edu.sg
Leung Cheuk Yui	U1622880A	cleung002@e.ntu.edu.sg
Joshua Lim Ting Wei	U1622602G	jlim222@e.ntu.edu.sg
Visco Christienne Grace Regodon	U1722616L	christie001@e.ntu.edu.sg
Li Yuting	U1720729F	b170024@e.ntu.edu.sg
Nguyen Ngoc Khanh	U1720451D	C170030@e.ntu.edu.sg
Xing Wanting	U1721577A	wxing003@e.ntu.edu.sg

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
AY2019/2020 SEMESTER 2

Chapter 1

Introduction

Graph Convolutional Networks (GCNs) applies convolution filter into graphs. Each GNC layer aggregate the embeddings of a node's neighbours from the previous layer and transforms it in a non-lineary fashion to obtain the new contextualised node representation. Through stacking multiple GCN layers, each node representation is thus able to make use of a wide receptive field from both immediate and distant node neighbours, which intuitively increases model capacity [1].

An experiment will be conducted in relation to the paper in [1]. The experiment will include the implementation and adaptation of algorithms developed from the paper and the replication of experimental results from the algorithms implemented in the paper.

Chapter 2

Background

2.1 Problem Statement and Related Works

GCNs are rising in popularity due to favourable implementations in various areas. However, training GCNs is difficult in practice due to the fact that graph data can be extremely large, unlike tokens in a paragraph or pixels in an image that normally have a fixed size. A node's embedding in GCNs depend recursively on all its neighbours' embeddings, causing an exponential growth in computation dependency depending on the number of layers. Thus, GCNs are still inapplicable in large-scale graphs.

Classical solutions to the size problem includes sampling-based methods such as the "node-wise neighbour-sampling" and "layer-wise importance sampling" that train GCNs on a subset of nodes. The "node-wise neighbour-sampling", which recursively samples a fixed number of neighbour nodes and calculates each node's embedding [1]. However, when nodes share the same sampled neighbour, the neighbour's embedding has to be reevaluated and the redundancy increases exponentially with more layers. This results in an exponentially increasing computation cost with a node's neighbour size.

An the other hand, the "layer-wise importance sampling" calculates the sampling probability based on the node's degree and a fixed number of nodes is sampled for each layer. The sampled nodes are used to build a smaller, now sampled adjacency matrix, which reduces computation costs. A problem, however, is that sampled nodes from adjacent layers may not be connected due to the fact that sampling probability is independent for each layer. This may cause the matrix to be extremely sparse and so, the sampled nodes may suffer from sparse connectivity problems.

Due to the limitations of the classical methods, alternative methods such as the VR-GCN [2], FastGCN [3] and Cluster-GCN [4] have been proposed in the recent years. The FastGCN reduces the computation cost by using only sampled nodes to build a sampled adjacency matrix [3], while the VR-GCN uses variance reduction techniques to improve sample complexity to reduce computation redundancy [2] and the Cluster-GCN does preprocessing before training the GCN by restricting the sampled neighbors within some dense subgraphs through a graph clustering algorithm [4]. However, it is argued in [1] that those proposed methods still cannot resolve the issue of redundant computations completely.

2.2 Real-World Applications

Due to its flexibility, GCNs can be applied in various domains in the real-world. Examples of such domains are in the application of GCNs in Computer Vision (CV), Natural Language Processing (NLP) and other sciences.

CV has been a hot research area in the past decades. Although classic convolutional neural networks (CNN) have achieved great successes in CV, it is not feasible to encode the intrinsic graph structures in the specific learning tasks. On the other hand, GCNs have been applied and achieved a comparable or even better performance in some CV problems. For example, [5] proposes a graph CNN to leverage both the semantic graphs of words and spatial scene graph for visual relationship detection. Furthermore, in [6], it is shown that a GCN model can be used to process the input scene graph and generate the images by a cascaded refinement network for photographic image synthesis. Moreover, GCNs can also be applied in videos. [7] applied GCN to perform action recognition by proposing a spatial-temporal graph convolutional model to eliminate the need of hand-crafted part assignment which achieved a greater expressive power.

GCNs also have applications in NLP. For text classification, citation network can be constructed with the documents as nodes and the citation relationships among them as edges. And node classification can be a straightforward way to classify documents into different categories. In [8], TextGCN performs text classification by modeling a whole corpus to a heterogeneous graph and learn word embedding and document embedding simultaneously, followed by a softmax classifier for text classification. In addition, a syntactic GCN model is developed and it can be used on top of syntactic dependence trees, which is suitable for semantic role labelling [9] and neural machine translation [10].

GCNs are also applied in other domains outside of Computer Science.

In chemistry, [11] first models drug-protein target interactions and protein-protein interactions into a multimodal graph, and then graph convolutions is applied to predict polypharmacy side effects.

In material science, [12] proposes a crystal GCN to directly learn material properties from the connections of atoms in the crystal.

In social sciences, GCNs have been widely used for social recommendation to improve the recommendation performances based on user-item interactions and/or user-user interactions. [13] proposes a neural influence diffusion model for better social recommendation by considering the influence of trusted friends.

Chapter 3

The Challenges of the Problem

The goal of the LADIES sampling method is to improve on the previously implemented sampling methods. The issue with the other methods is that it requires high computation and memory cost. The weakness of node-wise neighbour-sampling method is that it recursively samples a fixed number of neighbours which leads to high computation cost. Furthermore, layer-wise importance-sampling method discards the neighbour-dependent constraints and thus the nodes sampled will lead to high memory cost.

A good sampling method should satisfy 2 criteria. The sampling method has to be layer-wise, neighbour-dependent and must use importance-sampling. The LADIES sampling method solves these issues by mini-batch processing. Every iteration, it chooses a mini-batch nodes then dependent-sampling of neighbours is applied with a fixed size of sampled nodes that limits the space complexity.

Chapter 4

Previous Work: LADIES

4.1 Independent Layer-wise Sampling Framework

Let $S_l = \{i_k^{(l)}\}_{k=1,2,...,s_l}$ where $s_l = |S_l|$ and $i_k^{(l)} \in V$ be the set of sampled nodes at the l-th layer. Hence, The value of matrix production $PH^{(l)}$ can be estimated in equation 4.1:

$$PH^{(l)} \approx PS^{(l)}H^{(l)} \tag{4.1}$$

where $S^{(l)} = \mathbb{R}^{|V|\times|V|}$ is a diagonal matrix defined by the sampling distribution adopted $p_1^{(l)}, p_2^{(l)}, ..., p_{|V|}^{(l)}$ to all nodes in V, which is defined as

$$S_{i,i}^{(l)} = \begin{cases} \frac{1}{s_l p_i^{(l)}}, & i \in S_l \\ 0, & otherwise \end{cases} \tag{4.2}$$

Define the row selection matrix $Q^{(l)} \in \mathbb{R}^{s_l \times |V|}$ as:

$$Q_{r,c}^{(l)} = \begin{cases} 1, & (r,c) = (k, i_k^{(l)}) \\ 0, & otherwise \end{cases} \tag{4.3}$$

Then the forward process of GCN with layer-wise sampling can be approximated by:

$$\tilde{Z}^{(l+1)} = \tilde{P}^{(l)}\tilde{H}^{(l)}W^{(l)}, \quad \tilde{H}^{(l)} = \sigma\tilde{Z}^{(l)} \tag{4.4}$$

where $\tilde{Z}^{(l)} \in \mathbb{R}^{s_l \times d^{(l)}}$ denotes the approximated intermediate embedding for sampled nodes at l-layer and equation 4.5 serves as a modified Laplacian matrix.

$$\tilde{P}^{(l)} = Q^{(l+1)}PS^{(l)}Q^{(l)T} \in \mathbb{R}^{s_{l+1}\times s_l} \tag{4.5}$$

4.2 Layer-dependent Importance Sampling

Sampling Distribution: The sampling distribution at l-th layer is defined in equation 4.6:

$$p_i^{(l)} = \frac{||Q^{(l+1)}P_{*,i}||_2^2}{||Q^{(l+1)}P||_F^2} \tag{4.6}$$

where $||\mathbf{M}||_2$ and $||\mathbf{M}||_F$ are respectively L2 Norm and Frobenius Norm.

Normalisation: Normalised $\tilde{P}^{(l)}$
Normalisation operation is essential since it maintains the scale of embedding in the forward process and avoid exploding and vanishing gradient. The proposed normalisation of $\tilde{P}^{(l)}$ is stated in equation 4.7:

$$\tilde{P}^{(l)} \leftarrow \mathbf{D}_{\tilde{P}^{(l)}}^{-1}\tilde{P}^{(l)} \tag{4.7}$$

Algorithm 1 Sampling Procedure of LADIES

```
1: Require: Normalised Laplacian Matrix $P$; Batch Size $b$, Sample Number $n$. Randomly sample a batch of $b$ output nodes as $Q^L$
2: for $l = L-1$ to $0$ do
3:     Get layer-dependent laplacian matrix $Q^{(l+1)}P$. Calculate sampling probability for each node using $p_i^{(l)} = \frac{||Q^{(l+1)}P_{*,i}||_2^2}{||Q^{(l+1)}P||_F^2}$, and organize them into a random diagonal matrix $S^{(l)}$
4:     Sample $n$ nodes in $l$ layer using $p^{(l)}$. The sampled nodes formulate $Q^{(l)}$
5:     Reconstruct sampled laplacian matrix between sampled nodes in layer $l$ and $l+1$ by $\tilde{P}^{(l)} = Q^{(l+1)}PS^{(l)}Q^{(l)T}$, then normalise it by $\tilde{P}^{(l)} \leftarrow \mathbf{D}^{-1}_{\tilde{P}^{(l)}}\tilde{P}^{(l)}$
6: end for
7: return Modified Laplacian Matrices $\{\tilde{P}^{(l)}\}_{l=1,\ldots,L}$ and Sample Node at Input Layer $Q^0$
```

Chapter 5

Experiment

5.1 Implementation

We managed to fix many bugs in the original code and finally produced the correct version according to the paper. Sampling Procedure is described as follow in algorithm 2, and the code can be found in listing 5.1.

Algorithm 2 LADIES Pseudo-code

```
1: for For layer from last to first do
2:     Calculate L2 Norm of Q(l)P*, i and Frobenius Norm of Q(l)P from squared normalized laplacian matrix using row-selection (Note that QM is equivalent to row-selection operator)
3:     Calculate sampling probabilities p
4:     Fix number of sampled nodes in case p is negative. (This has no effect on the code since all norms are non-negative but it is included in the original code since they wrongly used laplacian matrix instead of the square.)
5:     Random sample nodes and add the nodes of next layers.
6:     Conduct the unbiased sampling by dividing the to sampling probabilities
7:     Row-normalization to avoid explosion of feature vectors.
8: end for
```

```python
def sampler(batch_nodes: np.ndarray, samp_num_list: np.ndarray, num_nodes: int,
     lap_matrix: sparse.csr_matrix, lap2_matrix: sparse.csr_matrix, num_layers: int)
      -> SimpleNamespace:
   ##### INPUT ####################################################
F = \text{row-F = \text{row-permutation}(I_N)

permutation}(I_N)
   # samp_num_list: array of number of sampled nodes at all layers
   # num_nodes : number of graph nodes
   # lap_matrix : row-normalized laplacian matrix
   # lap2_matrix : squared lap_matrix (precomputed)
   # num_layers : len(samp_num_list)
   ##### OUTPUT ###################################################
   # adjs : P matrix
   # input_nodes: sampled nodes at input
   # output_nodes : batch_nodes
  ##################################################################
```

```python
def full_sampler(batch_nodes: np.ndarray, samp_num_list: np.ndarray, num_nodes: int, lap_matrix: sparse.csr_matrix, lap2_matrix: sparse.csr_matrix, num_layers: int) -> SimpleNamespace:
    # simply sample the full lap_matrix for every layers
    sample = SimpleNamespace(
        adjs= [lap_matrix for _ in range(num_layers)],
        input_nodes= np.arange(num_nodes),
        output_nodes= batch_nodes,
    )
    return s
def ladies_sampler(batch_nodes: np.ndarray, samp_num_list: np.ndarray, num_nodes: int, lap_matrix: sparse.csr_matrix, lap2_matrix: sparse.csr_matrix, num_layers: int) -> SimpleNamespace:
    '''
        LADIES_Sampler: Sample a fixed number of nodes per layer. The sampling probability (importance)
                         is computed adaptively according to the nodes sampled in the upper layer.
    '''
    previous_nodes = batch_nodes
    adjs  = []
    '''
        Sample nodes from top to bottom, based on the probability computed adaptively (layer-dependent).
    '''
    for d in range(num_layers):
        # row-select the lap2_matrix (U2) by previously sampled nodes
        U2 = lap2_matrix[previous_nodes , :]
        # calculate sampling probabilities
        pi = np.sum(U2, axis=0)
        p = pi / np.sum(pi)
        # get number of sampled nodes
        s_num = np.min([np.sum(p > 0), samp_num_list[d]])
        p = p.view(np.ndarray).flatten()
        # sample the next layer's nodes based on the adaptively probability (p).
        after_nodes = np.random.choice(num_nodes, s_num, p = p, replace = False)
        # Add output nodes for self-loop
        after_nodes = np.unique(np.concatenate((after_nodes, batch_nodes)))
        # row-select and col-select the lap_matrix (U), and then divided by the sampled probability for unbiased-sampling.
        adj = lap_matrix[previous_nodes, :][:, after_nodes]
        adj = adj.multiply(1/ p[after_nodes])
        # conduct row-normalization to avoid value explosion.
        adj = row_normalize(adj)
        # fill the sub-matrix into the original
        adj = sparse_fill(lap_matrix.shape, adj, previous_nodes, after_nodes)
        adjs.append(adj)
        # turn the sampled nodes as previous_nodes, recursively conduct sampling.
        previous_nodes = after_nodes
    # Reverse the sampled probability from bottom to top. Only require input how the lastly sampled nodes.
    adjs.reverse()

    sample = SimpleNamespace(
        adjs= adjs,
        input_nodes= previous_nodes,
        output_nodes= batch_nodes,
    )
    return sample
```

Listing 5.1: LADIES code

5.2 Experiment

Due to the limitation of computing power, we decided to use random block model with two same-sized clusters with connection probability as follow in equation 5.1.

$$P = \begin{bmatrix} \frac{log(N)}{N}, \frac{log(N)}{N^2} \\ \frac{log(N)}{N^2}, \frac{log(N)}{N} \end{bmatrix} \tag{5.1}$$

where $P_{i,j}$ is the connection probability between nodes in cluster i and cluster j.

The feature vector is taken as an random row-permutation of Identity matrix where feature vector is represented in equation 5.2.

$$F = \text{row-permutation}(I_N) \tag{5.2}$$

The neural network consists of 5 GCN layers of 64 hidden layer dimensions and a Linear layer at the end works as a classifier. All layers were implemented with a dropout probability of 0.5

We used cross-entropy loss with log_softmax with Adam Optimizer, the learning rate was set to 1e-3.

For each run, the model was trained in 80 epochs with two sampling modes: Full-batch and ladies. The execution time, loss and f1 score was recorded in table 5.1. The plots obtained by the experiment can be referred to in in Appendix B.

Number of Nodes	Runtime for 80 epochs	
	Full-Batch	LADIES
64	2.6	8.82
128	6.63	30.71
256	22.80	101.91
512	98.51	315.95
1024	384.68	1020.65

Table 5.1: Model runtime for 80 epochs

Number of Nodes	Convergence Epochs	
	Full-Batch	LADIES
64	35	50
128	40	36
256	40	20
512	30	18
1024	16	15

Table 5.2: Number of Epochs for convergence

Number of Nodes	Updates per epoch	
	Full-Batch	LADIES
64	1	1
128	1	2
256	1	4
512	1	8
1024	1	16

Table 5.3: Number of updates per epoch

Number of Nodes	Updates for convergence	
	Full-Batch	LADIES
64	35	50
128	40	72
256	40	80
512	30	144
1024	16	240

Table 5.4: Number of updates to convergence

Number of Nodes	Convergence time	
	Full-Batch	LADIES
64	1.3	5.8
128	3.5	15
256	12.5	25
512	38	65
1024	85	140

Table 5.5: Convergence time

As such, the observation from the results in table 5.2 is that LADIES converges much faster than Full-Batch w.r.t epochs especially for large graphs due to the nature of stochastic gradient descent. LADIES sparse matrix implementation can be improved further.

Chapter 6

Conclusion

The difference between Layer-Dependent Importance Sampling (LADIES) method and the Full-Batch sampling method is that the LADIES method does not sample the whole graph. It samples the important nodes that will be processed in the neural network. However, the Full-Batch sampling method uses the whole graph as training samples. The methods can be considered as batch gradient descent and stochastic gradient descent. Throughout the experiments, it is found that Full-Batch sampling has a better execution time as compared to LADIES sampling. However, due to space complexity, Full-Batch sampling cannot be scaled up for large-size data. On the other hand, by the nature of stochastic gradient descent, LADIES method takes the advantage of conducting many updates in a batch of data, it converges faster than Full-Batch sampling.

Chapter 7

Further Research

It has been observed that the execution time of LADIES method is at least 1.5 times slower than Full-Batch method caused by the inefficiency in sparse matrix implementation. A better implementation is possible to yield better results.

Bibliography

[1] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, “Layer-dependent importance sampling for training deep and large graph convolutional networks”, in *Advances in Neural Information Processing Systems*, 2019, pp. 11 247–11 256.

[2] J. Chen, J. Zhu, and L. Song, “Stochastic training of graph convolutional networks with variance reduction”, *arXiv preprint arXiv:1710.10568*, 2017.

[3] J. Chen, T. Ma, and C. Xiao, “Fastgcn: Fast learning with graph convolutional networks via importance sampling”, *arXiv preprint arXiv:1801.10247*, 2018.

[4] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, “Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks”, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 257–266.

[5] Z. Cui, C. Xu, W. Zheng, and J. Yang, “Context-dependent diffusion network for visual relationship detection”, in *Proceedings of the 26th ACM international conference on Multimedia*, 2018, pp. 1475–1482.

[6] Q. Chen and V. Koltun, “Photographic image synthesis with cascaded refinement networks”, in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1511–1520.

[7] S. Yan, Y. Xiong, and D. Lin, “Spatial temporal graph convolutional networks for skeleton-based action recognition”, in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[8] L. Yao, C. Mao, and Y. Luo, “Graph convolutional networks for text classification”, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 7370–7377.

[9] D. Marcheggiani and I. Titov, “Encoding sentences with graph convolutional networks for semantic role labeling”, *arXiv preprint arXiv:1703.04826*, 2017.

[10] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima’an, “Graph convolutional encoders for syntax-aware neural machine translation”, *arXiv preprint arXiv:1704.04675*, 2017.

[11] M. Zitnik, M. Agrawal, and J. Leskovec, “Modeling polypharmacy side effects with graph convolutional networks”, *Bioinformatics*, vol. 34, no. 13, pp. i457–i466, 2018.

[12] T. Xie and J. C. Grossman, “Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties”, *Physical review letters*, vol. 120, no. 14, p. 145 301, 2018.

[13] L. Wu, P. Sun, Y. Fu, R. Hong, X. Wang, and M. Wang, “A neural influence diffusion model for social recommendation”, in *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, 2019, pp. 235–244.

Appendix A

Experiment Code

Appendix B

Experiment Plots

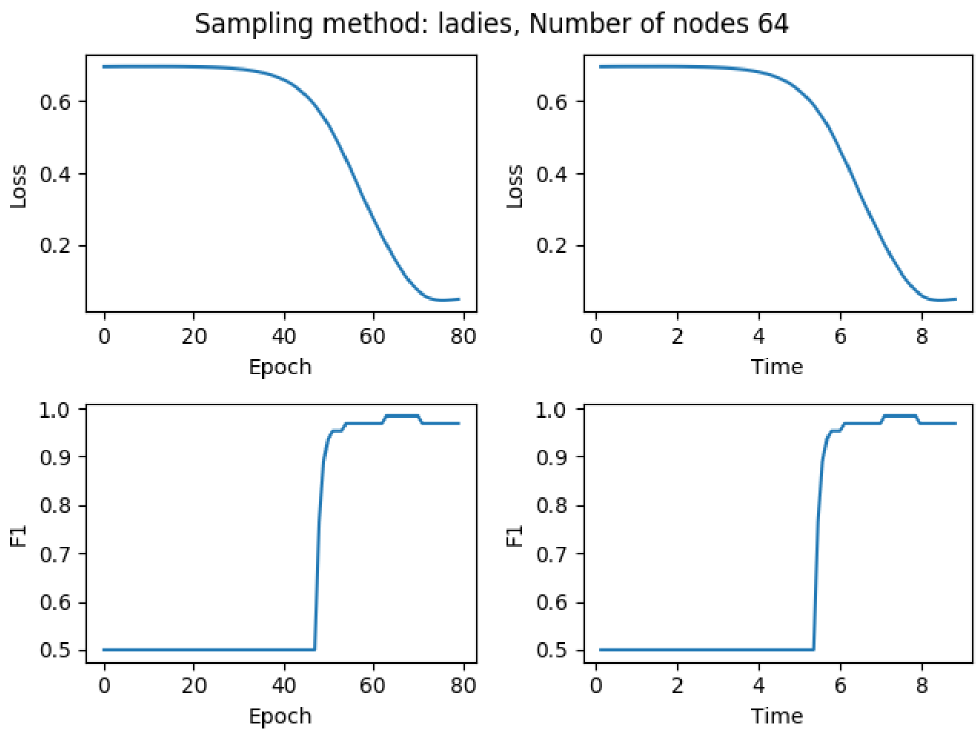

Figure B.1: Ladies with 64 nodes

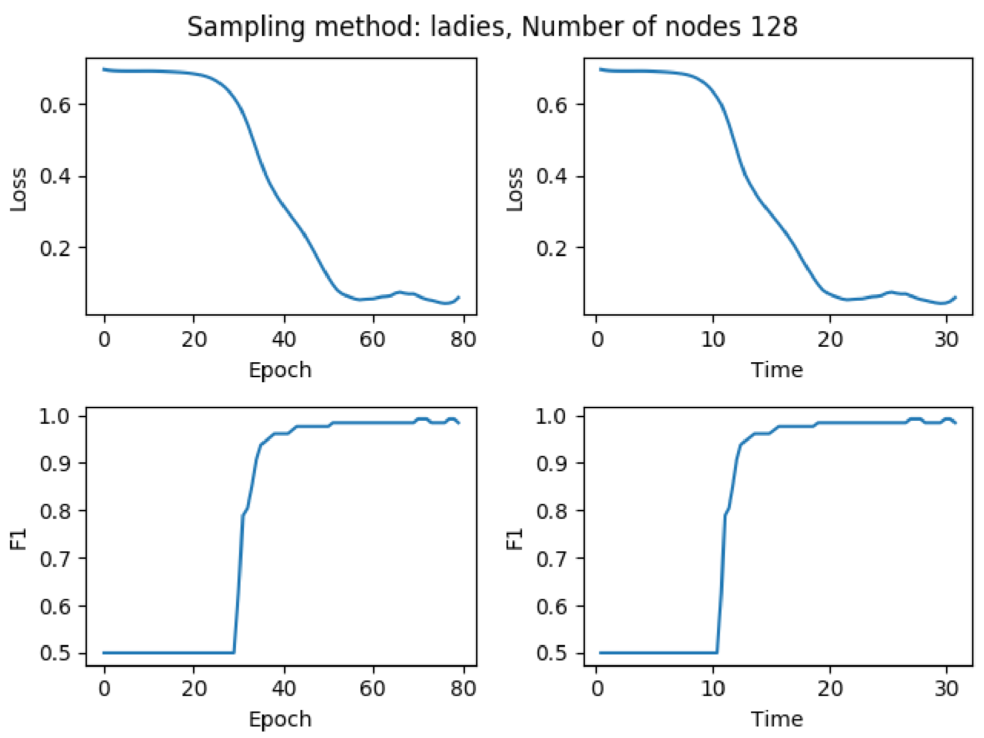

Figure B.2: Ladies with 128 nodes

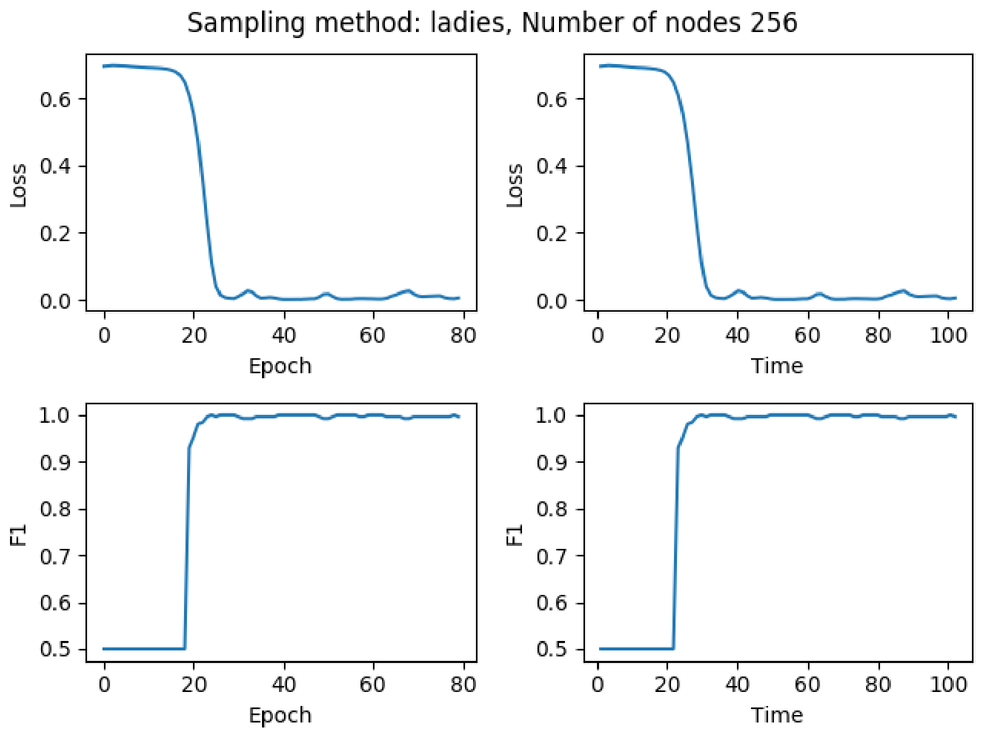

Figure B.3: Ladies with 256 nodes

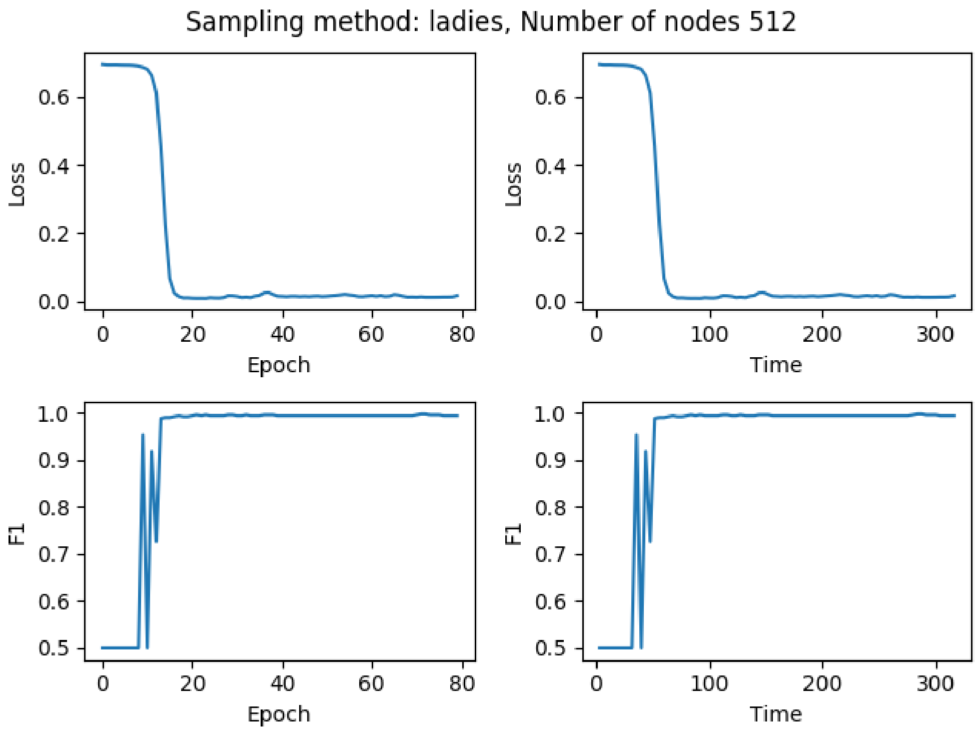

Figure B.4: Ladies with 512 nodes

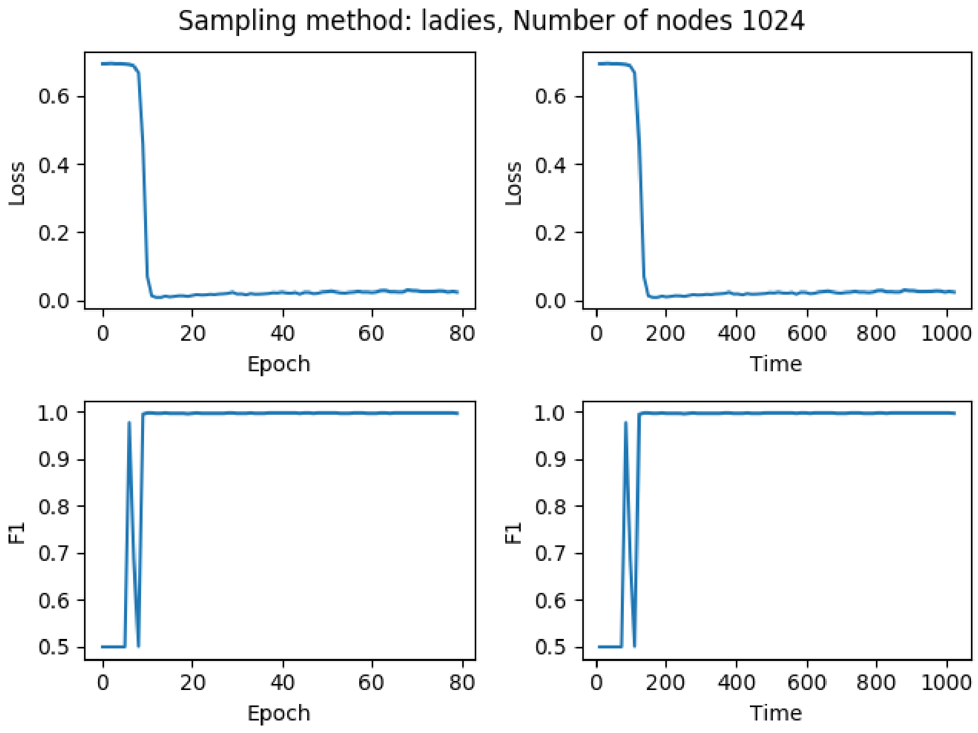

Figure B.5: Ladies with 1024 nodes

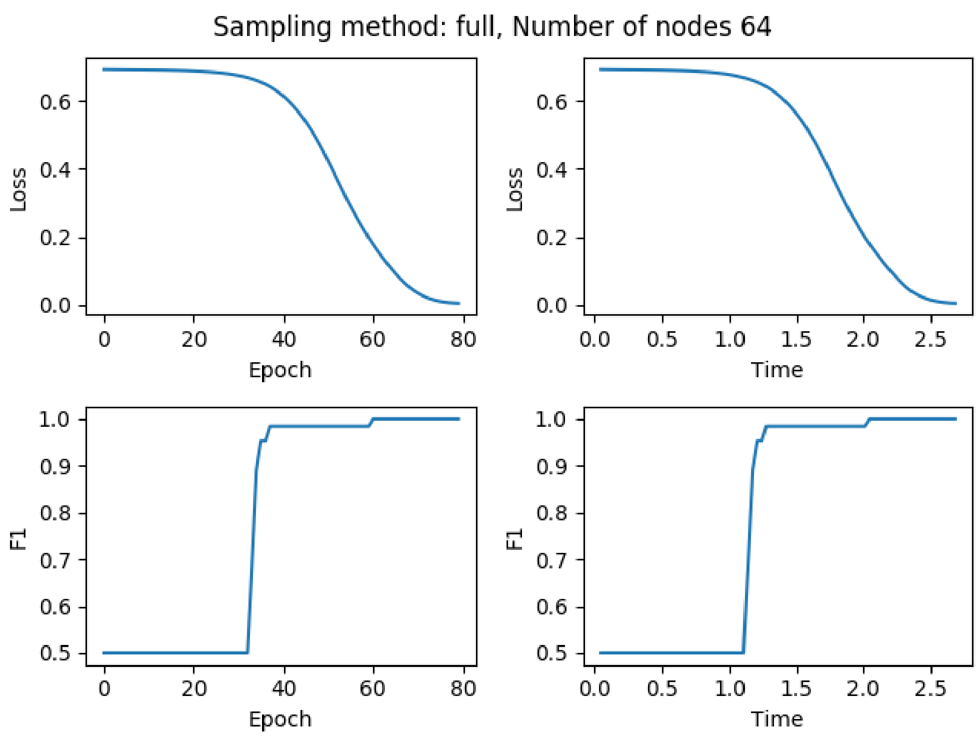

Figure B.6: Full GCN with 64 nodes

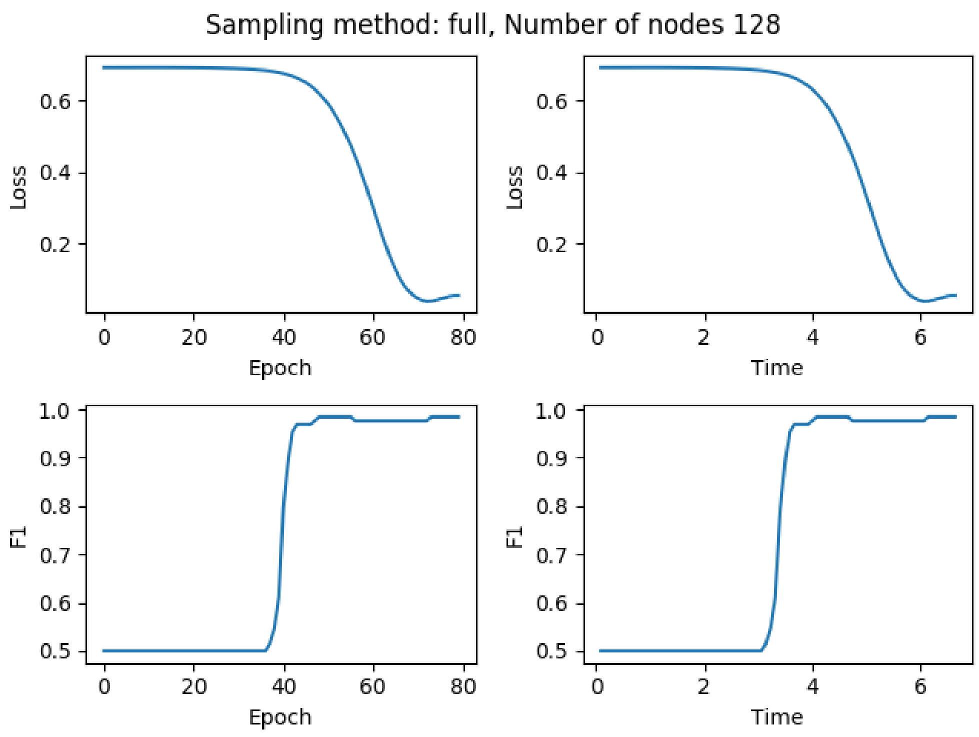

Figure B.7: Full with 128 nodes

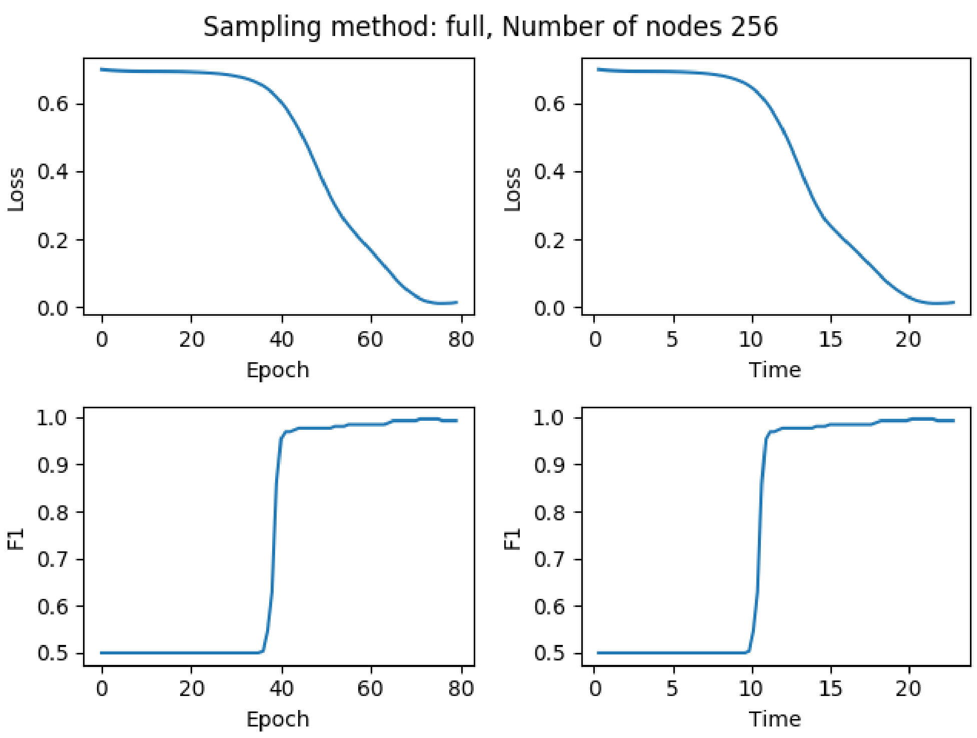

Figure B.8: Full GCN with 256 nodes

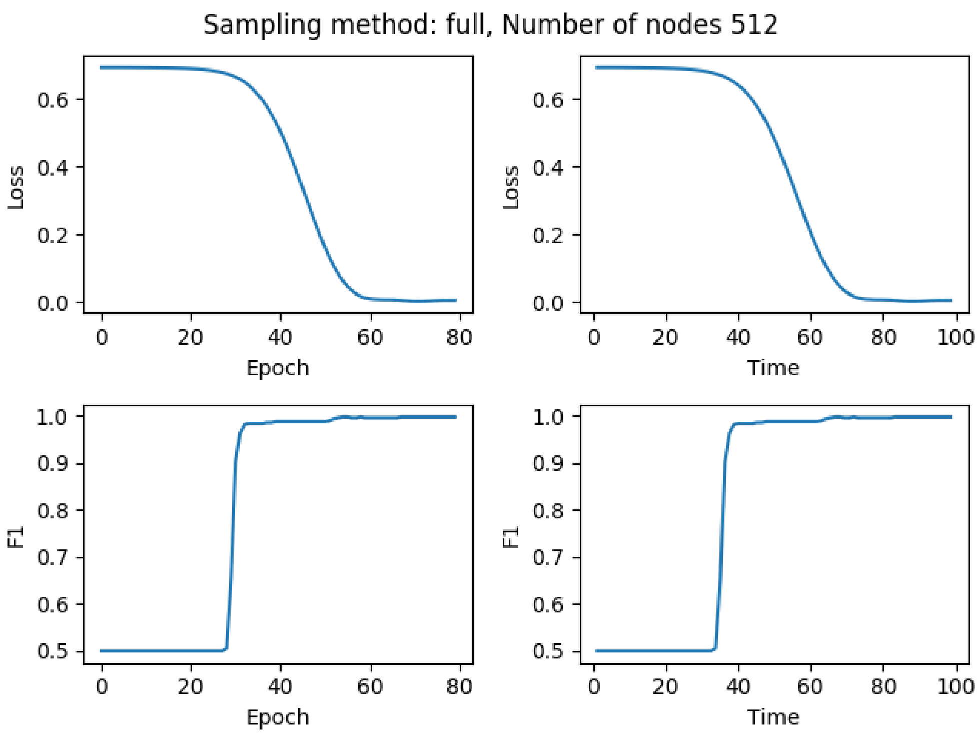

Figure B.9: Full GCN with 512 nodes

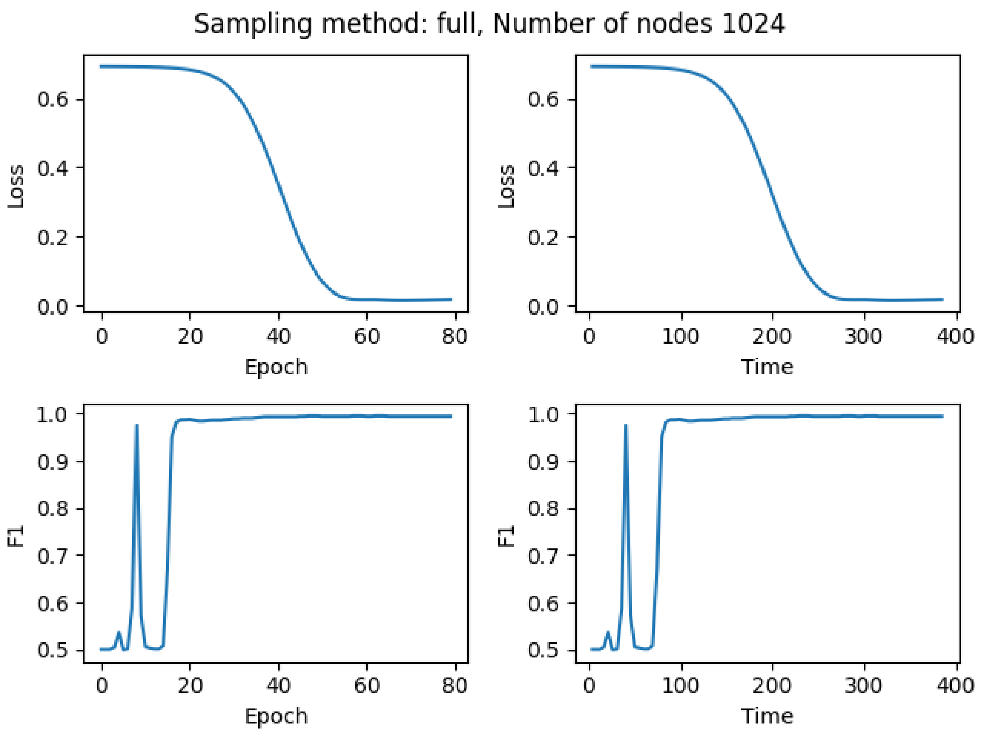

Figure B.10: Full GCN with 1024 nodes