Title
A Study on Graph Neural Network

Author
Shi, Mengyao

Publication Date
2021

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

A Study on Graph Neural Network

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

Mengyao Shi

2021

ABSTRACT OF THE THESIS

A Study on Graph Neural Network

by

Mengyao Shi
Master of Science in Computer Science
University of California, Los Angeles, 2021
Professor Cho-Jui Hsieh, Chair

This thesis summarizes the work I have done during my master's study at UCLA. We ranked 38th among all the participants of the KDD 21 challenge on large-scale graph machine learning. We built a two-stage model, taking the most out of UniMP and Correct and Smooth architectures in Pytorch. We studied a social network graph with 121 million nodes and 153 categories, achieving node classification accuracy of 65%.

The second part of thesis summarizes a mini-batch attention-based graph machine learning model that we developed. We first learned a dense self-attention based on graph node features and overlayed it with the original adjacency matrix. It achieves about the same test accuracy of $69.00 \pm 0.28\%$ on the Arxiv dataset compared to clusterGCN, but it has the potential to outperform. This is especially true when graph node features are rich and informative. Interesting results may yield for a deeper GCN.

The thesis of Mengyao Shi is approved.

Quanquan Gu

Yizhou Sun

Cho-Jui Hsieh, Committee Chair

University of California, Los Angeles

2021

To Kaiming Yang, Heng Li and my parents

TABLE OF CONTENTS

1 Introduction **1**

2 Participation of OGB-LSC KDD CUP 2021 . . . . . **7**

3 Algorithm Design for Mini-batch Attention Based Graph Machine Learning **11**

4 Experiments **14**

5 Conclusion **17**

LIST OF FIGURES

1.1 design components of graph machine learning [1] 6

2.1 The architecture of UniMP [2] 9

4.1 Scale of additional attention matrix scale β change with training epoch. 16

LIST OF TABLES

4.1 Remove a percentage of edges results 15

ACKNOWLEDGMENTS

I would like to thank my advisor, Cho-Jui Hsieh, without whose guidance and kind support this thesis would not be possible. I want to thank my research group, collaborators, and peers Ruochen Wang, Si Si, Hengda Shi, Max Wu, and Minhao Cheng. I want to thank professor Yizhou Sun, who guided me into the field of graph machine learning and data mining. I would also like to thank Professor Yong-Jae Lee and Professor Norman Matloff from UC Davis who warmly supported my transition into the Computer Science field. I am always grateful to Professor Maxwell Chertok for his guidance and support during my physics PhD.

During my master's study, I also received much help and inspiration from Chak Wong, Ahmad Emami, and Jessica Lam during the time I interned at the Machine Learning Center of Excellence, JPMorgan Chase & Co. I really enjoyed my summer and want to thank all the talented individuals I met there.

I am grateful for Kaiming Yang and Heng Li's support during my master's study and would like to dedicate this work to them. I am grateful for Doctor Diane Scheiner and Doctor Chunyan Jia's support.

2020–2021 were a challenging time due to the global world pandemic. I am grateful to be able to conduct research in the computer science field with minimal disturbance and remote working conditions. I appreciate the professors, leaders, and workers from UCLA who are trying their best to support remote learning. I am grateful for the sacrifices the world's healthcare workers have made for the general public.

CHAPTER 1

Introduction

A lot of real-world data naturally exists in a graph format. This includes social networks, financial transaction networks between different entities, supply chain networks, knowledge graphs, and so on. Graphs can be classified into different types, such as homogeneous/heterogeneous graphs, directed/undirected graphs, or static/dynamic graphs. A dynamic graph evolves with time, for example financial transactions between different entities.

For graph machine learning tasks, the graph community has been focused on the following levels,

Node level tasks focus on node-level prediction such as classification, node regression, and clustering. This is the main focus of this thesis. For example, predicting paper category in a citation network.

Edge level tasks focus on edge-level prediction such as edge classification, link prediction, and link ranking. For example, predicting competitors in a supply chain network.

Graph level tasks include graph classification and graph regression. Graph-level representation will be learned.

Figure 2.1 from review [1] gives an overview of the major design components of graph machine learning. For a propagation module, mainstream methods include a convolution operator, recurrent operator, and skip connection. Convolutional operators are mainly categorized into two: spectral and spatial. I will go over convolutional neural network [3]. For the sampling module, I will discuss one of each method, talking about node sampling method GraphSAGE [4], layer sampling LADIES [5], and subgraph sampling ClusterGCN [6] pooling

module. Then, I will talk about the attention mechanism on graphs.

Convolutional neural networks have been applied and have seen major improvements in different areas such as computer vision. For example, this is seen in image segmentation and classification and natural language processing such as machine translation. Yet, it was not until recently that people started to explore how to generalize convolutional neural networks into domains of graph-structured data [3].

Graph convolutional networks (GCN) get the name 'convolutional' from that filters are learned during the training. These filter parameters are typically shared over all locations in the graph. In a manner similar to that of computer vision, filters are learned from image input and how they are shared over all pixel locations.

Define a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each node i is associated with initial node features x_i. The initial input of the models are two fold, the graph itself and initial node features X. X is of dimension $N \times D$, $H^{(0)} = X$. The goal is to learn node representations for each node. At the final layer $H^{(L)} = Z$, Z is of dimension $N \times F$.

$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)}) \tag{1.1}$$

$\sigma(\cdot)$ is a non-linear activation like ReLU. Above is the simplest form of GCNs. More sophisticated forms of convolutional neural networks are as the following, which includes normalization of the adjacency matrix and adding identity connections.

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{1.2}$$

Where $\tilde{A}$ is $A + I$ with I being identity matrix, $\tilde{D}$ is the diagonal node degree matrix of $\tilde{A}$.

Original full-batch GCN requires calculating the representation of all nodes in the graph for each GCN layer. Since this is computationally costly and not memory efficient, full-batch GCN methods are difficult to generalize to larger graphs. Recent years have seen an

increase in work that applies convolutional neural networks on graph nodes and operates on nodes' neighborhoods. For example, GraphSAGE samples graph nodes' neighborhoods with a certain budget, and it aggregates the neighborhood's representation and updates the representation of the node itself. This method achieves impressive results across multiple large-scale graph benchmarks. Node-wise neighborhood sampling also suffers from exponential growth of neighborhood issues, and some other works explore how to solve this problem and come up with smarter sampling. LADIES, a work that performs layer-dependent importance sampling, select neighborhood nodes based on the sampled nodes in the upper layer by computing importance probability.

ClusterGCN is an efficient algorithm to train deep and large scale graphs. ClusterGCN uses graph clustering algorithm to constrain the neighborhood search within a smaller, scaled, sub graph. This strategy is really effective in improving memory usage and problem scalability. It endures tests on large-scale graphs and achieves impressive results across several benchmarks including Amazon2M with 2 million nodes and 61 million edges.

There have been many academic endeavours in recent years to understand why graph machine learning works. Correct and smooth [7] is an interesting one that separates the contributions of node features and node labels. The algorithm first ignores the graph structure and uses shallow layers to learn on node features. Then, at a later stage, they leverage the graph structure and label information to correct errors made in this first stage. They make the assumption that the nearby nodes have error correlation and prediction correlation. Because this approach abandons computational expensive convolutional neural network calculation, it has great scalability. This method also achieves impressive results on certain known datasets such as OGB-Products [8]. The correct and smooth stages can be used as a post-processing strategy for people using different initial architecture.

Recently, there has been research work in attention mechanisms on sequence-based tasks. For example, "Attention is all you need" [9] describes the approach of using attention mechanism to learn to focus on different parts of the sequence while generating the representation

of a sequence. Attention mechanisms also have generalization on music analysis and vision tasks. Researchers have also recently tried to extend attention mechanisms into graph learning domains. For example, Graph attention network (GAT)'s [10] main idea is to compute an attention map of how a node should attend to its neighborhood's representations. The computation is efficient because it can be computed in parallel manner.

The input to GAT layer is a set of node features, $\{\vec{h}_1, \vec{h}_2, ..., \vec{h}_N\}$, where N is number of nodes, $\vec{h}_i \in \mathbb{R}^F$. The output will be a set of node representations $\{\vec{h'}_1, \vec{h'}_2, ..., \vec{h'}_N\}$, where $\vec{h'}_i \in \mathbb{R}^{F'}$. To obtain sufficient expressive power, a shared linear transformation $\boldsymbol{W}$ is applied to every node. Then a self-attention coefficients $a : \mathbb{R}^F \times \mathbb{R}^{F'}$ is computed across node pairs that indicate node i's importance for node j.

$$e_{ij} = a(\boldsymbol{W}\vec{h}_i, \boldsymbol{W}\vec{h}_j) \tag{1.3}$$

In GAT, the algorithm only computes e_{ij} for first order neighbors $j \in \mathcal{N}_i$. Where $\mathcal{N}_i$ is the neighborhood of i. As per their paper, the attention mechanism is a single layer feedforward neural network with LeakyReLU nonlinearity. To stabilize the learning process, GAT uses multi-head attention, and each head's features are concatenated together. GAT is widely known to have achieved impressive results across multiple benchmarks.

Transformer architecture has been dominant in many areas in NLP [9] and Vision [11]. People have recently started to generalize and extend transformers to the graph domain. Graphomer [12] tried to generalize the transformer on a graph and asked a question, "Do transformers really perform bad for graph representation?". Our study tries to answer this open question and leverage the transformer method by incorporating the unique characteristics of a graph.

A graphomer inherits the design of transformer modules and includes design for unique characteristics of a graph. The major considerations are the following three components: centrality encoding, spatial encoding, and edge encoding in the attention.

Centrality encoding is about adding an additional feature that includes encoding of graph node centrality. The philosophy behind this is that additional information in graph nodes should not be neglected. Such as the number of followers of celebrities in a network, popularity or centrality plays an important role in making graph node predictions. Spatial encoding is about encoding the structural information of a graph. If two nodes are connected, then the distance of the shortest path would be encoded, otherwise it would encode a special value. The third consideration is edge encoding. Attention mechanism estimates a correlation for each pair of nodes, and the edges connecting them should be considered. For each pair, they find the shortest path and compute an average of the dot-products of the edge feature and a learnable embedding along the path.

Graphomer is an interesting exploration of how to extend transformers into the graph domain. And at KDD 2021 OGB-LSC [13], the graphomer achieves a top 1 result in a large-scale graph node classification task.

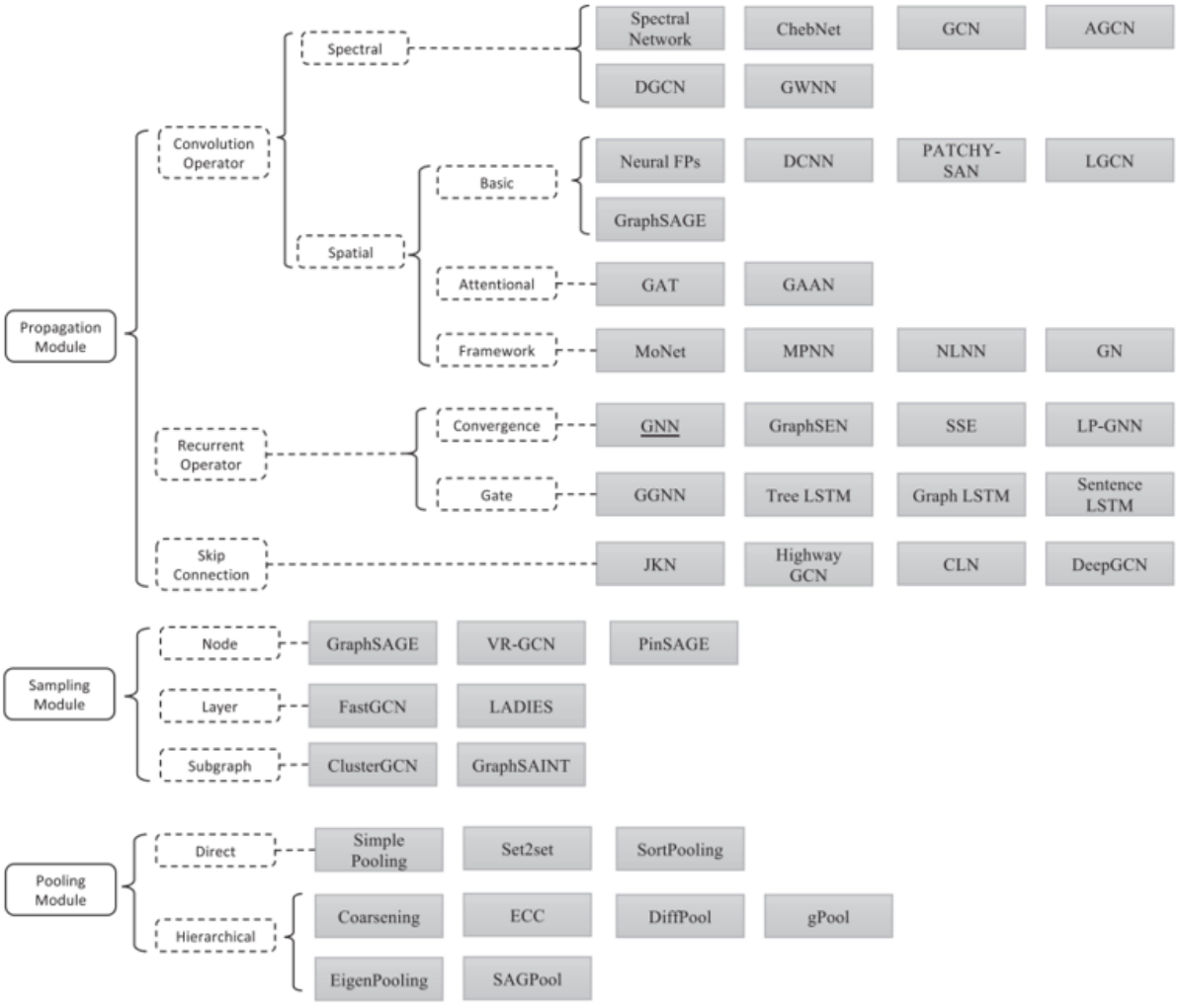

Figure 1.1: design components of graph machine learning [1]

CHAPTER 2

Participation of OGB-LSC KDD CUP 2021

Graph machine learning has attracted immerse attention in recent years because of a rising need to understand graph structured data and deliver real-world applications. And these graphs tend to come in large scales. For example, social networks connecting billions of people around the world or recommendation systems with large user bases. Being able to study graphs at a large scale has become increasingly important to make an impact on real-world data. The OGB Large Scale Challenge KDD CUP 2021 [13] was launched in March 2021 with three very challenging tasks. My team consisted of Hengda Shi, Max Wu, and I, and we participated as the team 'GoBruins' and were ranked 38^{th} on the first task. This chapter will be on the participation process, model design, and results.

The first task was called MAG240M-LSC. The goal of this task was to predict papers' subject area in a heterogeneous academic graph. MAG240M-LSC is a heterogeneous academic graph extracted from the Microsoft Academic Graph (MAG). There were 121M academic papers in English. The resultant paper set was written by 122M author entities, who are affiliated with 26K institutes. Among these papers, there were 1.3B citation links captured by MAG. Each paper was associated with its natural language title and most papers' abstracts were also available. We concatenated the titles and abstracts by period and passed it to a RoBERTa sentence encoder [14] [15], generating a 768-dimensional vector for each paper node. Among the 121M paper nodes, approximately 1.4M nodes were arXiv papers annotated with 153 arXiv subject areas, e.g., cs.LG (Machine Learning).

The major challenge of this task was the scale of the problem. We centered the design

around being able to handle a problem of this scale. Our first idea was to be able to do a local "summary" of graph nodes and make a high-level prediction of the graph super nodes. Then, we went back down to low level of graph and made individual predictions with local adjustments. Following this idea, we found an already-published paper called "GraphZoom: A multi-level spectral approach for accurate and scalable graph embedding" [16]. GraphZoom shares the same idea across famous benchmarks provided by OGB, and this type of approach usually performs well but does not rank at the top of leaderboards.

Our second idea was also centered around handling a large-scale graph. We found a recent approach called "correct and smooth". The method first predicted graph node labels based on just graph node features without considering graph structure. At the correct and smooth stage, the method leveraged graph structure information and made corrections and smoothing to improve the initial results. We tested the implementation on our MAG240M-LSC dataset and found out that on a citation network with 121M nodes and 1.3B citation links, we could get a training result within 3 hours, noting that a 250GB CPU memory is necessary for correct and smooth. The hardware we required for these results were one GeForce RTX 1080 GPU (11GB memory) and an intel(R) Xeon(R) CPU E5-2640 v4 CPU @ 2.40GHz (512GB memory).

Noticing that the first stage of "correct and smooth" is a simple MLP implementation, we asked a question: If we could use a more sophisticated model as the initial stage, but use correct and smooth as a post-processing method, will we achieve better results? We implemented a hybrid model. We chose a sophisticated attention-based model. The model is composed of two parts, the first of which is an attention-based graph learning model called "UniMP" [2]. Then we took the UniMP results to the post-processing stage and used the correct and smooth method. Here are the detailed descriptions of UniMP.

UniMP has two major interesting features in its design. 1) It uses a graph transformer as its design module. 2) It uses partially observed labels as training data. I will expand on both.

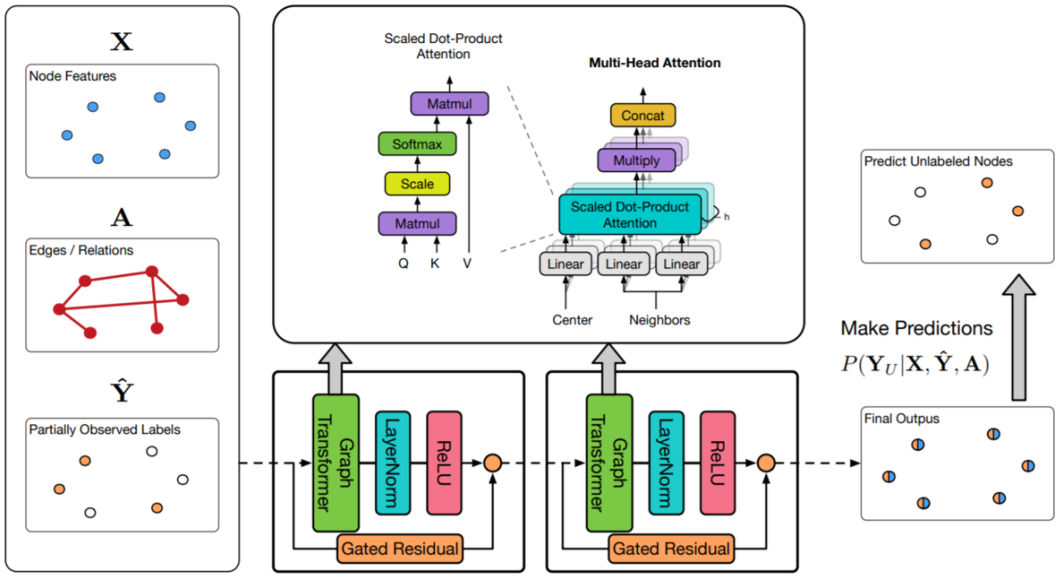

Figure 2.1: The architecture of UniMP [2]

Correct and smooth is a multistage method that corrects and smooths the first-stage results. Inputs first go through an MLP-based simple neural network that does not require an adjacency matrix at all. Then the second stage, "correct", and third stage, "smooth", will refine the learned results. In detail,

$$E_{L_t,:} = Z_{L_t,:} - Y_{L_t,:}, \quad E_{L_v,:} = 0, \quad E_{U,:} = 0 \tag{2.1}$$

Where $Z \in \mathbb{R}^{n \times c}$ is the prediction from base model, Y is ground truth. $E \in \mathbb{R}^{n \times c}$ is error matrix. Error is residual on the training data L_t. For validation dataset L_v and unlabeled U , this residual is zero. The iteration methods follows label spreading technique [17],

$$\hat{E} = argmin\ trace(W^T(I - S)W) + \mu\|W - E\|_F^2 \tag{2.2}$$

Where $W \in \mathbb{R}^{n \times c}$. The first term encourages smoothing the error of estimation, while the second term puts a constraint that makes the results, after iterations, not vary too much from the original.

On the official leaderboard, our team 'GoBruins' achieved 0.6517 test accuracy. We trained for 50 hours on a GeForce RTX 1080 GPU (11GB memory) and an intel(R) Xeon(R) CPU E5-2640 v4 CPU @ 2.40GHz (512GB memory). The parameters are summarized as follows: learning rate 0.001, num of layers 3, size $[10, 10, 10]$, hidden channels 128, heads 4, dropout 0.5 label rate 0.625, epochs 100.

The correct and smooth stage yielded an additional 6% increase of accuracy. As opposed to what was in the original paper, correct and smooth can give a major boost to the MLP baseline. I find this result really interesting. Our thoughts are that correct and smooth works best on conditions where the first-stage model does not use graph structure in training.

CHAPTER 3

Algorithm Design for Mini-batch Attention Based Graph Machine Learning

This is a joint work in collaboration with Ruochen Wang (UCLA), Minhao (UCLA), Cho-Jui Hsieh (UCLA), and Si Si (Google). Motivated by the self-attention mechanism in NLP, we designed the algorithm that I am going to discuss in this chapter. We asked one question: Does dense self-attention also work on node classification tasks? We first learned a dense self-attention based on graph node features and overlayed it with the original adjacency matrix.

So we can assume that the graph is $A + \beta W$, where A is the original graph and W is the dense graph computed by a transformer. An interesting thing about this is that it's like a sequence data with only one very long sequence (e.g., million), while in a traditional NLP or graphormer you usually have millions of short sequences.

Notice now that the new $A + \beta W$ graph structure is no longer a sparse matrix. We tackle this issue by using mini-batch methods such as using clusterGCN to partition the graph into many clusters and using attention mechanism only on subgraphs. The algorithm is summarized in Algorithm 1.

ClusterGCN samples a dense graph and restricts the neighborhood search within the subgraph. We used clusterGCN as our mini-batch algorithm, therefore I would like to sketch and design the idea, time, and space complexity of the algorithm. Vanilla mini-batch SGD have slow per epoch time. This is because, in order to calculate the gradient for one node i:

Algorithm 1 mini-batch attention based graph learning algorithm

Input: Graph A, feature X, label Y

Output: Node representation $\bar{X}$

Partition graph nodes into c clusters, $V_1, V_2, ..., V_c$ by METIS

for $i \leftarrow 1$ to max_iter **do**

  Randomly choose q clusters, $t_1, ..., t_q$ from V without replacement

  Form sub graph $\bar{G}$ with nodes $\bar{V}$ and links $A_{\bar{V},\bar{V}}$

  Use self attention mechanism to calculate matrix W with node features as input

  Compute Norm$(A + \beta W)$ and use it to replace the original adjacency matrix

  Compute g $\leftarrow \nabla \mathcal{L}_{A_{\bar{V},\bar{V}}}$

  Conduct Adam update

end for

Output weights

$\nabla loss(y_i, z_i^{(L)})$, we need the embeddings from its 1-hop neighborhood. To get the embeddings of its neighborhood, the aggregation needs to be further propagated to 2-hop neighborhoods and so on. Assume that on average each node has d degree, GCN has $L + 1$ layers, time complexity of calculation is $O(d^L F^2)$, where F is feature dimension. By using clustering algorithms such as METIS [18] first and constraining the neighborhood expansion within the subgraph, calculation can be both efficient in terms of time and memory. In summary, clusterGCN's time complexity is $O(L\|A\|_0 F + LNF^2)$ and space complexity is $O(bLF + LF^2)$. With $\|A\|_0$ being number of nonzeros in the adjacency matrix, N is number of nodes, and b is the batch size . In the original paper, authors showed how clusterGCN's time and space complexity would compare to some well-known algorithms such as GCN, GraphSage, and FastGCN [19].

Self attention is the module we used here to compute W. Let input be $H \in \mathbb{R}^{N \times d}$, where N is the number of nodes in a minibatch and d is the hidden dimension. The input H is projected by three matrices, $W_Q \in \mathbb{R}^{d \times d_K}$, $W_K \in \mathbb{R}^{d \times d_K}$. The projected results become their

corresponding representations.

$$Q = HW_Q, \quad K = HW_K \tag{3.1}$$

$$W = softmax(\frac{QK^T}{\sqrt{d_K}}) \tag{3.2}$$

We separately normalize W and adjacency matrix A, $\text{Norm}(A+\beta W) = (1-\beta)(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})+\beta W$. The normalization method is the same as that used in GCN.

To prevent/mitigate overfitting, we use random edge drop [20]. At the training time, this method automatically drops a percentage of random edges to reduce overfitting. We also use a standard random drop out. The third drop out used is attention drop out. We use ray tune to automatically find the ultimate best combination of the three drop out methods used.

CHAPTER 4

Experiments

We run an experiment with 130 epoch 10 times on Arxiv dataset. It takes 24 hours to train on a GeForce RTX 1080 GPU (11GB memory) and an intel(R) Xeon(R) CPU E5-2640 v4 CPU @ 2.40GHz (512GB memory).

Hyper-parameter fine tuning is done by using a Ray Tune automatic hyper-parameter search parallelly run on multiple GPUs. The detailed model parameters are summarized as the following: Dataset ogbn-arxiv, hidden_dim 256, lr 0.0011732, epoch 130, use_pp True, opt adam, dropout 0.2, num_layers 3, partition 1500, bsize_list 10, diag_lambda 0.0, sampling_percent 0.6, use_cluster, model trans_fix, sepnorm 1, scale_fn 'I', att_dropout 0.8. The overall experimental results are the following: Highest Train: 75.87 $\pm$ 0.07, Highest Valid: 70.36 $\pm$ 0.22, Final Train: 75.06 $\pm$ 0.54, Final Test: 69.00 $\pm$ 0.28.

The second experiment we performed was to randomly remove a percentage of edges from the beginning. The Arxiv dataset with removed edges was used as input in the training and results compared with a standard clusterGCN. The motivation of doing this experiment was that our attention matrix W is learned from graph node features. At the limit of when all the edges were removed, our methods should have been able to significantly outperform clusterGCN. This could be generalized to a graph with sparse connections but rich node features. Below are the results of this experiment.

Another experiment we performed was to visualize the β scale we learned as a learning process. In a 3 layer network architecture, we plotted the scale of each layer and showed how they evolved as the training went by. Interestingly, the first layer has an attention map scale

Remove A Percentage of Edges Results				
Model	Percentage removed	Train Accuracy	Validation Acc	Test Acc
mini-batch Attention (ours)	10%	0.73	0.68	0.67
ClusterGCN		0.69	0.70	0.68
ours	20%	0.73	0.68	0.68
ClusterGCN		0.68	0.70	0.69
ours	30%	0.72	0.69	0.67
ClusterGCN		0.69	0.70	0.68
ours	40%	0.71	0.66	0.65
ClusterGCN		0.67	0.68	0.67
ours	50%	0.70	0.66	0.65
ClusterGCN		0.65	0.68	0.66
ours	70%	0.67	0.64	0.63
ClusterGCN		0.62	0.66	0.65
ours	90%	0.63	0.60	0.59
ClusterGCN		0.57	0.61	0.59

Table 4.1: Remove a percentage of edges results

close to 0. And as the layers increased, the scales increased. See figure 4.1. This result could be generalized to cases where we have a deep graph neural network. Then the contribution from the self-attention part will be more dominant.

I analyzed nodes' degree distribution that are correctly predicted and did not see any difference in the shape of distribution. Future experiments that would be also very interesting include increasing depth and batch size.

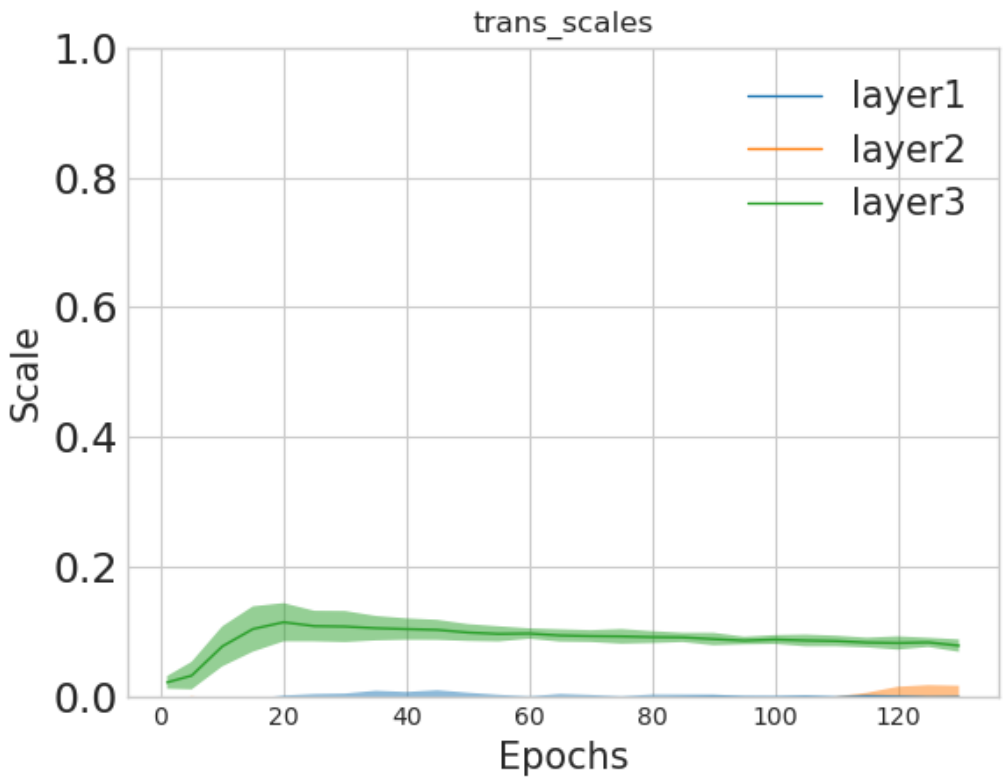

Figure 4.1: Scale of additional attention matrix scale β change with training epoch.

CHAPTER 5

Conclusion

This thesis summarizes the work I have done during my master's study at UCLA. We ranked 38th among all participants of the KDD 21 challenge on large-scale graph machine learning. We built a two-stage model. taking the most out of UniMP and correct and smooth architectures in Pytorch. We studied a social network graph with 121 million nodes and 153 categories, and we achieved node classification accuracy of 65%.

The second part of this thesis summarized a mini-batch attention-based graph machine learning model we developed. It achieves about the same test accuracy on an Arxiv dataset 70% compared to clusterGCN, but it has potential to outperform it. This is especially true at the case where the adjacency matrix of a graph is sparse or graph node features are rich and informative. When the graph structure becomes deeper, the attention map learned contributes more at an increased scale. Interesting results may be seen for a deep graph neural network.

Bibliography

[1] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012

[2] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," *arXiv preprint arXiv:2009.03509*, 2020.

[3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[4] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.

[5] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Few-shot representation learning for out-of-vocabulary words," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2019.

[6] 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 *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2019. [Online]. Available: https://arxiv.org/pdf/1905.07953.pdf

[7] Q. Huang, H. He, A. Singh, S.-N. Lim, and A. R. Benson, "Combining label propagation and simple models out-performs graph neural networks," *arXiv preprint arXiv:2010.13993*, 2020.

[8] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec,

"Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[10] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[12] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform bad for graph representation?" *Neural Information Processing Systems (NeurIPS)*, 2021.

[13] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "Ogb-lsc: A large-scale challenge for machine learning on graphs," *arXiv preprint arXiv:2103.09430*, 2021.

[14] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[15] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[16] C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng, "Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding," *arXiv preprint arXiv:1910.02370*, 2019.

[17] D. Zhou, O. Bousquet, and T. Lal, "J. weston, and b. schölkopf,"learning with local and global consistency," adv," *Neural Inform. Proc. Syst*, vol. 16, pp. 321–328, 2004.

[18] G. Karypis and V. Kumar, "Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997.

[19] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

[20] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," *arXiv preprint arXiv:1907.10903*, 2019.