DEEPERBIGGERBETTER FOR OGB-LSC AT KDD CUP 2021

TECHNICAL REPORT

Guohao Li [*]
KAUST

Jesus Zarzar [*]
KAUST

Hesham Mostafa
Intel Labs

Sohil Shah
Intel Labs

Marcel Nassar
Intel Labs

Daniel Cummings
Intel Labs

Sami Abu-El-Haija
USC

Bernard Ghanem
KAUST

Matthias Müller
Intel Labs

June 16, 2021

ABSTRACT

The Open Graph Benchmark Large-Scale Challenge aims to push the state-of-the-art on realistic large-scale graphs to new heights. In particular, the `MAG240M-LSC` dataset is one of the largest graph datasets, consisting of 121 million academic papers with 1.3 billion citation links between them. Given a 768-dimensional vector that represents the title and abstract and optional links to authors and institutions, the task is to predict the primary subject area out of 153 options. In this technical report we summarize the findings of our team *DeeperBiggerBetter* which was nominated as one of the winners of the challenge. In short, we train two R-GAT models, one with 2 layers and another with 3 layers for a total of 180M parameters. We utilize author labels as extra regularization, conduct multiple inference passes with proportional neighborhood sizes, aggregate their results by ensembling and then apply a label smoothing trick on model's predictions with author labels for post-processing. We denote our ensembled model as *GNN108M*. In the final submission, *GNN108M* achieves an accuracy of 73.53% on the held-out test set.

Keywords Open Graph Benchmark Large-Scale Challenge · KDD Cup · Graph Neural Network

1 Introduction

Graphs provide a natural data structure to model relational data through their neighborhood structure: related objects are connected by an edge with a label that represents the underlying relationship. Learning on graphs has exploded in popularity in recent years due to the impressive advances in graph neural networks (GNNs) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and their success in practical applications of many domains [11, 12, 13, 14, 15, 16, 17].

The availability of large-scale quality graph datasets [18, 19, 20] has been key to enabling advancements in graph learning research while at the same time providing reproducible benchmarks for graph models to be compared. The Open Graph Benchmark (OGB) [19] was proposed to address this need in the GNN research domain and provides a collection of datasets with graph structured data over irregular domains with different tasks such node, graph, and link prediction. This benchmark has contributed significantly to the strides achieved by recent graph learning work by providing data to train various models and evaluate them in a systematic fashion. As graph learning continues to mature, the scaling of graph neural networks to large graphs has shifted more into focus. Prior to OGB, popular node classifications datasets (e.g. CORA, CITESEER, PUBMED) [21] were of limited usefulness due to their small scale (<20,000 nodes) and they suffered from data quality issues [22]. In contrast, the largest node property prediction dataset in the initial OGB release was `papers100M` which consisted of 111 million papers indexed by Microsoft

[*]Equal contribution

Academic Graph (MAG) [20]. To push the graph scale boundary further, OGB Large-Scale Challenge (OGB-LSC) [23] was announced for the KDD Cup 2021 and provides three large datasets, one for each type of graph task [23]. The `MAG240M-LSC` [2] is one of those datasets with a node-classification task and is provided as an expansive heterogeneous graph consisting of 121 million academic papers with 1.3 billion citation links between papers where a 768-dimensional vector represents the title and abstract for each paper. Linked to the papers are 122 million author entities and 26 thousand institutions. Out of the papers, 1.4 million are arXiv papers that fall into 153 different subject areas. The objective of `MAG240M-LSC` is to predict the primary subject of the arXiv papers where the metric is classification accuracy. The dataset takes a time-based split approach where the training nodes are all arXiv papers published until 2018, validation nodes are arXiv papers published in 2019, and test nodes are arXiv papers published in 2020 and onward.

We build our models based on R-GAT [8, 24] which is implemented as one of the baseline models by the OGB-LSC team [23] with Pytorch Geometric [25]. Graph attention network (GAT) [8], a popular variant of GNNs, learns to quantify the impact of various nodes on each other using attention as a proxy for *relatedness* [26]. The attention mechanism compares the feature $\mathbf{h}_i$ of each node i in graph $(\mathcal{V}, \mathcal{E})$ to the features $\mathbf{h}_j$ of its neighbor $j \in \mathcal{N}(i)$ to produce an attention scalar α_{ij} that weights the effect of that neighbor's feature:

$$\alpha_{i,j} = \texttt{softmax}_j\Big(\texttt{LeakyReLU}(\mathbf{a}^{\mathbf{T}}[\mathbf{W}\mathbf{h}_i||\mathbf{W}\mathbf{h}_j])\Big), \quad \forall i \in \mathcal{V},\ \forall j \in \mathcal{N}(i), \tag{1}$$

where `LeakyReLU` is a leaky ReLU non-linearity function, the vector $\mathbf{a}$ and matrix $\mathbf{W}$ are learnable parameters and $||$ denotes a the concatenation operation. To empower GNNs with the ability to deal with relational data in heterogeneous graphs, relational propagation models are proposed in [27, 24]. Leveraging the relational edge labels $r_{i,j}$ to steer the attention mechanism, a multi-head R-GAT [24] is formulated as:

$$\mathbf{h}'_i = \sigma\left(\sum_r \frac{1}{K}\sum_k \sum_j \alpha^{(r,k)}_{i,j}\mathbf{W}^{(r,k)}\mathbf{h}_j\right), \quad \forall r \in \mathcal{R},\ \forall i \in \mathcal{V},\ \forall j \in \mathcal{N}(i), \tag{2}$$

where σ is a non-linearity function, K is the number of attention heads, r denotes one type of relations in the set $\mathcal{R}$, $\alpha^{(r,k)}_{i,j}$ is the attention weight between nodes i and j of the k-th head on the r-th relation and $\mathbf{W}^{(r,k)}$ is the weight matrix of the k-th head on the r-th relation.

2 Our Method

Generating Author Labels. To better utilize the author information, we generate labels for author nodes during the pre-processing step based on the subjects that each author has written as an auxiliary regularization. For any given author node i_a, if all the labeled paper nodes $j \in \mathcal{N}_p(i_a)$ linked to them share the same label y_j, this label is assigned to the author:

$$y_{i_a} = \begin{cases} y_k, & \text{if } y_j = y_k,\ \forall j,k \in \mathcal{N}_p(i_a) \\ \emptyset, & \text{otherwise} \end{cases},$$

where y_{i_a} is the subject label for author i_a, which depends on the subject labels y_j of the neighboring paper nodes $\mathcal{N}_p(i_a)$. The labeled author nodes are sampled along with the paper nodes. The models are then trained to predict these extra author classification labels along with the original paper classification labels by minimizing a cross-entropy loss. We find training to classify these author nodes along with the paper nodes helps reduce over-fitting and improves the model's performance on the validation set.

Network Architecture. We train two different models that both are based on R-GAT [8, 24]. To benefit from different receptive fields, the models are constructed with 2 or 3 R-GAT layers respectively. We use the original implementation [3] except for adding additional MLP layers and non-linearity to increase the number of learnable parameters.

Training Setting and Hyper-parameters. We first train our models on the training set and evaluate them on the validation set to find the best training setting and hyper-parameters. The finally submitted models are trained on the original training and the validation sets to leverage all the labeled data. From the evaluation on the validation set, we find that a large hidden size is essential for good performance; increasing it improves classification accuracy up to some point where it leads to overfitting. However, it incurs a high cost in memory and training time. We also experiment with several optimizers including Adam [28], AdamW and RAdam [29] and find RAdam to perform the best by a

[2] https://ogb.stanford.edu/kddcup2021/mag240m/

[3] https://github.com/snap-stanford/ogb/tree/master/examples/lsc/mag240m

Table 1: **Hyper-parameters of our models compared to the baseline R-GAT.**

	Baseline Model	Our *Model 1*	Our *Model 2*
Number of layers	2	2	3
Neighborhood size	25-15	25-15	25-20-15
Optimizer	Adam	RAdam	RAdam
Scheduler	Step	Cosine	Cosine
Hidden size (per layer)	1024	2048	1800
Training Epochs	100	100	100
Batch size (per GPU)	1024	1024	512
Number of GPUs	1	4	4
Author Features	✗	✓	✓
Extra MLP	✗	✓	✓
Number of parameters	12.26M	81.20M	99.47M
Valid Accuracy (%)	70.02	71.08	71.87

small but noticeable margin. Similarly, using the Cosine scheduler compared to the step scheduler leads to small but consistent improvements. We also try the cosine scheduler with restart, but the training behaviour is too erratic and we do not observe any performance gains. All the models are trained with `NeighborSampler` [6]. We experiment with different neighborhood sizes as well. In general, having a large number neighbors is beneficial. We also find that deeper models perform better and that it is better to decrease the number of neighbors per layer gradually. Since there is a large cost in terms of training time, we train a 2-layer (*Model 1*) and 3-layer model (*Model 2*) as a trade-off and fuse their results as described in Section *Ensembling*. With less time constraints, we believe that using a much deeper R-GAT model with reversible connections [30] would improve results further. We describe the exact hyper-parameters of our models and compare them to the baseline R-GAT model in Table 1. To realize over-parameterization [31], our models have much larger hidden size (2048 channels for *Model 1*, 1800 channels for *Model 2*) than the baseline model (1024 channels). An extra MLP layer is added after each GAT in the models. As a result, our *Model 1* and *Model 2* have large numbers of parameters with $81.20M$ and $99.47M$ respectively. The pre-processed author features, RAdam optimizer and Cosine scheduler are used for training our models. As shown in Table 1, we find that our models benefit from over-parameterization and a larger respective field. We discuss the results and more details in Section 3 (*Results*).

Inference. We find that especially for deeper network architectures, it is beneficial to evaluate on neighborhood sizes that are proportional to the training size. For example, for a training neighborhood size of $\langle 25-20-15 \rangle$, we use $\langle 250-200-150 \rangle$ or $\langle 125-100-75 \rangle$ during evaluation instead of using a fixed neighborhood size such as $\langle 160-160-160 \rangle$. To account for the randomness in neighborhood sampling, we run inference multiple times [30] with the same model for the final submission. In particular, 10 inference runs are used for the 2-layer model, and 4 inference runs with neighborhood sizes of proportions $\{1, 3, 5, 6\}$ are used for the 3-layer model. Multiple inference runs are not done with the 3-layer model due to time constraints. The results of all the inference runs are used as described in the Section 3 (*Result*).

Ensembling. We generate an ensemble of the predicted score $\mathbf{s}_{i_p}$ for paper node i_p by aggregating prediction probabilities $\mathbf{p}_{i_p}^m$ from the set of inference settings $\mathcal{M}$. For the final model, set $\mathcal{M}$ consists of 10 inference runs of the 2-layer model using a proportional neighborhood factor of 5, and inferences of the 3-layer model using proportional neighborhood factors of 1, 3, 5, and 6. Predicted probabilities are aggregated by performing a model-wise *max* aggregation to generate the predicted scores $\mathbf{s}_{i_p}$:

$$\mathbf{s}_{i_p} = \max_{m \in \mathcal{M}} \{\mathbf{p}_{i_p}^m\} \tag{3}$$

We find *max* to perform better than a *mean* aggregation when combining the 2-layer models with the 3-layer models. The choices for the multiple inferences used for aggregation were mostly determined by the time constraint. Ideally, we would have used proportional neighborhood factors in the range from 1 to 10.

Post-Processing. In order to further improve the predictions, we generate a soft label $\mathbf{p}_{i_a}$ for each author node. This label is given by averaging the one-hot encoded labels $\texttt{onehot}(y_k)$ of papers written by each author i_a. It represents the statistics that an author writes a paper on each subject given the previous subjects they have published in. For training on the training set, the soft label of author nodes are generated from the their published paper until 2018. For training on the training and validation sets, the soft label of author nodes are generated from the their published paper until 2019. For a given paper, the soft label of the paper node i_p is then generated by averaging soft labels of all its authors $\mathcal{N}_a(i_p)$. Each author-based probability vector $\mathbf{p}_{i_p}$ for paper i_p is then combined with the final model's ensemble scores

$\mathbf{s}_{i_p}$ through a convex combination with a coefficient of $\lambda = 0.4$, which we found to work best:

$$\begin{aligned} \mathbf{p}_{i_a} &= \frac{1}{|\mathcal{N}_p(i_a)|} \sum_{j \in \mathcal{N}_p(i_a)} \texttt{onehot}(y_k), \\ \mathbf{p}_{i_p} &= \frac{1}{|\mathcal{N}_a(i_p)|} \sum_{j \in \mathcal{N}_a(i_p)} \mathbf{p}_{j_a}, \\ \mathbf{pred}_{i_p} &= \lambda \mathbf{p}_{i_p} + (1-\lambda)\mathbf{s}_{i_p}. \end{aligned} \tag{4}$$

The final predicted labels l_{i_p} of paper nodes are obtained by $l_{i_p} = \texttt{argmax}(\mathbf{pred}_{i_p})$.

3 Results

We use Version 1.3.1 of the Open Graph Benchmark for all experiments. In the following we report the hardware we used and the training and inference time for the final models we submitted. In our finial submission, *Model 1* is trained for 200 epochs (about 40 hours) using 4 NVIDIA V100 (32GB) GPUs and 24 Intel Xeon CPUs with 500GB of RAM. *Model 2* is trained for 50 epochs (about 40 hours) using 4 NVIDIA RTX 6000 (48GB) GPUs and 50 Intel Xeon CPUs with 480GB of RAM. For inference, we use 8 NVIDIA RTX 3080 (24GB) GPUs and 100 Intel Xeon CPUs with 960GB of RAM for 3 hours.

Training only on the training set and evaluating on the validation set. We train *Model 1* and *Model 2* for 100 epochs on the training set and then evaluate each model for ten times on the validation set with mini-batch sampling. For *Model 1*, the evaluation is done with a fixed proportional neighborhood size of $\langle 125 - 75 \rangle$. One evaluation run results in an accuracy of 71.08% on the validation set. Aggregating 10 runs via mean aggregation slightly improves performance to 71.10%. For *Model 2*, the evaluation is done with a proportional neighborhood size of $k * \langle 25 - 20 - 15 \rangle$ where $k \in [1, 10]$. We find that aggregating the results with different proportional neighborhood sizes improves results. The average accuracy of these 10 runs is 71.79%. Combining the predictions above of *Model 1* via mean aggregation achieves a performance of 71.97%. When combined with the aggregated results of *Model 1* via max aggregation, this becomes 72.21%. The aggregation of *Model 2* with only post-processing achieves 72.55% validation accuracy. Combining the aggregated results of *Model 1*, *Model 2* and post-processing achieves 72.72% on the validation set.

Training on the training and validation sets for the final submission. We perform the equivalent steps described above with the same models trained on the training and validation set. Our final model achieves a performance of **73.53%** on the held-out test set [4]. For the submission, we trained *Model 1* for 200 epochs. We were only able to train our *Model 2* for 50 epochs due to time constraints. We perform 10 inference runs for the 2-layer model with a neighborhood size as $\langle 125 - 75 \rangle$, and 4 inference runs with proportional neighborhood sizes $k * \langle 25 - 20 - 15 \rangle$ where $k \in \{1, 3, 5, 6\}$ for the 3-layer model. Multiple inference runs are not done with the 3-layer model due to time constraints. The results of all the inference runs are used as described in the Section 3 (*Result*). We expect that training it for longer, would further improve results.

4 Conclusion and Discussion

We believe that there are several interesting directions for future work which we were unable to investigate further due to the time constraints of the competition. First, our experiments indicate that there is a correlation between the number of model parameters and task performance. We conjecture that over-parameterized models would be beneficial. Recent work enables training of deep GNN architectures with a memory footprint independent of the number of layers [30]. This seems like a very promising path towards deep over-parameterized models that can capture much more complex neighborhoods. In addition, the recent work GNNAutoScale [32] is a promising approach for sampling large neighborhoods with constant memory usage. In combination with deep residual graph neural networks [30], this seems to be a promising direction for building deep over-parameterized models that can capture large neighborhoods.

Our experiments also show the benefit of aggregating predictions from multiple models. It seems that models with different depths and neighborhoods learn complementary features. Similarly, running inference multiple times with different/randomized neighbors and then aggregating results to smooth the predictions is beneficial. We believe that further investigation of ensemble techniques and multi-view inference [30] is important. Independent of the model architecture, we find that handcrafted author labels help to boost classification accuracy. We believe that a look closer into the data can unlock further gains. Similarly, ingesting more domain knowledge into the models and investing more time in feature engineering seem to be important.

[4] https://ogb.stanford.edu/kddcup2021/leaderboard

References

[1] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

[3] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[4] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[5] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[7] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[8] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[9] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[10] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[11] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 817–826. ACM, 2009.

[12] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.

[13] Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pages 3697–3707, 2017.

[14] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019.

[15] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *The IEEE International Conference on Computer Vision*, 2019.

[16] Lowik Chanussot, Abhishek Das, Siddharth Goyal, Thibaut Lavril, Muhammed Shuaibi, Morgane Riviere, Kevin Tran, Javier Heras-Domingo, Caleb Ho, Weihua Hu, et al. The open catalyst 2020 (oc20) dataset and community challenges. *arXiv preprint arXiv:2010.09990*, 2020.

[17] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, Jan 2021.

[18] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[19] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, volume 33, pages 22118–22133, 2020.

[20] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1:396–413, 2020.

[21] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[22] Xu Zou, Qiuye Jia, Jianwei Zhang, Chang Zhou, Hongxia Yang, and Jie Tang. Dimensional reweighting graph convolutional networks, 2020.

[23] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

[24] Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y. Hammerla. Relational graph attention networks. *arxiv:1710.10903*, 2019.

[25] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *NIPS*, 2017.

[27] 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.

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[29] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020.

[30] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. In *Proceedings of Machine Learning Research*, 2021.

[31] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019.

[32] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. *arXiv preprint arXiv:2106.05609*, 2021.