Project Erdos: Towards End-to-End Compute Graph Optimization with Graph Neural Networks

Mohd Tabish Jamal
M.Sc. Digital Engineering
Otto-von-Guericke-Universität, Magdeburg
mohd.jamal@st.ovgu.de

Nazar Iqbal
M.Sc. Digital Engineering
Otto-von-Guericke-Universität, Magdeburg
nazar.iqbal@st.ovgu.de

Oishi Chatterjee
M.Sc. Data and Knowledge Engineering
Otto-von-Guericke-Universität, Magdeburg
oishi.chatterjee@st.ovgu.de

Rajasekhar Dubaguntla
M.Sc. Digital Engineering
Otto-von-Guericke-Universität, Magdeburg
raja.dubaguntla@st.ovgu.de

Rijin Shaji
M.Sc. Digital Engineering
Otto-von-Guericke-Universität, Magdeburg
rijin.shaji@st.ovgu.de

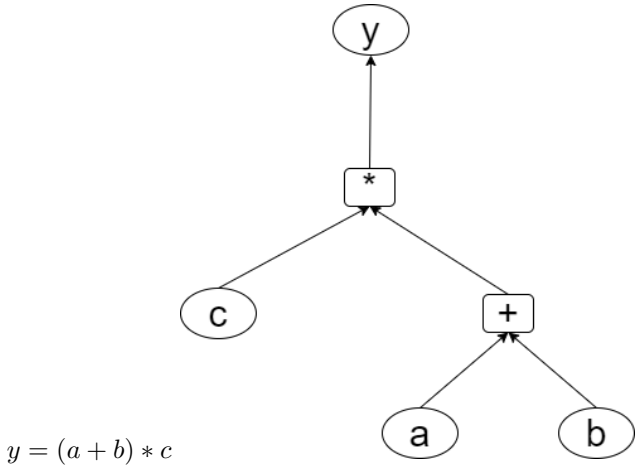
Sathya Sudha Murugan
M.Sc. Data and Knowledge Engineering
Otto-von-Guericke-Universität, Magdeburg
sathya.murugan@st.ovgu.de

***Abstract*—Machine learning models are constantly growing in terms of complexity and there is a massive growth in the computational demands required for training and inference of the neural networks in last few years. We can optimise the models only upto certain extent to run them faster. But to get faster execution, we can optimize the device placement of the computational graphs of the machine learning models.At present it's done by having a heterogeneous distributed environment consisting of a mixture of hardware devices such as CPUs and GPUs where human experts do the placement based on simple heuristics and intuitions. In this paper, we start to research on how to optimize machine learning model's execution time by training a model for node classification for device placement. First we find the best model for node classification trained on a synthetic graph dataset and then, in future work, we seek to use the model identified to train for device placement task on the dataset having machine learning models. GatedGCN is the most competitive model among the models it is compared with i.e. MoNET and GAT. Because of which, we recommend its use for the follow-up work.**

***Index Terms*—graph neural network, device placement, machine learning optimization, GatedGCN**

I. Introduction

Graphs are an abstract data structure which is defined by two components namely nodes and edges. It can be directed or undirected. Graphs are widely used in programs and applications such as navigation applications, social networking etc. We are surrounded by applications, and all the applications are transformed into a Computation graphs in order to be executed. An example of a computation graph for a mathematical equation is shown in Fig. 1. Machine learning and database operations are also transformed into computation graphs so that they can be processed on the devices such as CPUs or GPUs. In the past few years, we have seen increased demands in computation requirements because of increased complexity of Machine Learning / Deep Learning models. The three factors that can be optimized to best achieve the optimization [4]:



$y = (a + b) * c$

Figure 1. Computation Graph

- Device Placement
- Operation Scheduling
- Operation Fusion

Device placement is being widely done by applying heuristics. It speeds up the execution, but not by a significant extent as compared to other Global methods [9]. Heuristics has a limitation of not being able to foresee the next coming nodes.

Scheduling the flow of nodes from the memory to the processors is called operation scheduling. Operation scheduling is important because the efficient scheduler minimises the idle time of the device by allocating the operations based on their execution priority [4]. It is done so that the device should not wait for a task to be completed before executing the current operation at hand.

Recent research [4] shows that competitive performance can be achieved by using a graph neural network model with reinforcement learning. However this research employs a fixed graph neural network model and does not provide a study about alternative graph neural network models that could possibly provide more competitive results.

In this work we seek to identify promising graph models for the node classification task that underlies the computation graph optimization tasks described previously.

The focus of this research is the optimization of the machine learning models by device placement using node classification of computation graphs.

Our main contributions are:

- We compare graph neural network models (GatedGCN, MoNET and GAT) at a node classification task, using diverse datasets.
- We study the impact of configurable factors (such as hidden dimension size, use of edge features or positional encodings), identifying how they contribute to the end model performance.

II. Background

In this section, we introduce two topics. First is to find the best node classification model using Graph Neural Network (GNN) architectures. Second, how the optimal utilization of the devices can be done using device placement for Graph Neural Networks so as to reduce the time of execution for the machine learning models.

In graphs, we can apply machine learning algorithms such as Node Classification, Relation Prediction, Clustering and community detection, Graph classification, regression, and clustering [5]. In the research done by Dwivedi [2], they have evaluated GNN architectures. They have used message passing GCNs and have compared performance of both isotropic and anisotropic GCNs. By using a synthetic dataset generated by Stochastic Block Model (SBM), they managed to find the best model by comparing different architectures such as MoNet, GAT, GatedGCN, and GatedGCN-PE. The best test accuracy was found for GatedGCN-PE with 86.508 for PATTERN dataset and 76.082 for CLUSTER dataset [2]. More details about these models are provided in Section 3.

Traditionally placing parts of neural models is done by human experts based on simple heuristics and intuitions. In this research by Mirhoseini [8], they present a method to optimize device placement for TensorFlow computational graph. They have achieved this by using a sequence to sequence model to predict which operations in a TensorFlow graph should run on which available device which can be either CPUs or GPUs. They have used Reinforcement Learning for this task. They have set the execution time as reward signal to optimize the parameter of the model. The execution time is then minimized for achieving the best model parameter configuration.

Many applications use large machine learning or deep neural network models which has high computational requirements. According to the research done in Google Research [4], to run the model optimally, compiler places the nodes in ML accelerators such as Central Processing Unit (CPU), Graphics Processing Unit (GPUs), and Tensor Processing Unit (TPUs). Placement of nodes in these accelerators is called device placement. ML compilers usually apply heuristics which does not give an optimal solution. It cannot foresee the next parts of computational graphs. To optimize graphs, they [4] proposed an end-to-end deep reinforcement learning method and have successfully achieved 33%-60% enhancement in graph optimization as compared to Tensorflow default optimization.

A hierarchical model can also be used for the effective placement of computational graphs on hardware devices [6]. This model mainly deals with a mixture of CPUs, GPUs and other devices. The model contains two sum-networks one is assigned operations to group and second one assigns group to target devices. The main intention is to predict a placement that speeds up the training of the graph neural network. The runtime we optimize for time taken to conduct one back-propagation, one forward pass and parameter update on the target neural network. The hierarchical planner widely uses machine learning models in natural language process and computer vision. The method is effectively placing the operations in a computational graph onto the device. The approach finds highly granular parallelism in the graph. The outperform prior method by 60.6% [6].

Placeto is a reinforcement learning concept that encompasses two key concepts: Iterative placement improvements rather than a single placement output at once, and the usage of Graph Embeddings to acquire meaningful insights about the structure of the computation graph instead of depending on node labels for indexing. The Neural Network design of the Placeto's graph embedding procedure and the policy network allows the training parameters across Markov Decision Process formulation to be shared across episodes. As a result, Placeto can learn generalizable device placement strategies for any given family of graphs and may be used to forecast optimized placements for unseen graphs in the same family without any training. As a result, there is no need to retrain every time a new graph needs to be placed. [7]

Mirhoseini [8] has proposed a reinforcement learning algorithm based on policy gradient method for device placement. But it is not efficient and does not provide us with optimal training times.

Another paper proposes a new reinforcement learning algorithm called Spotlight [1] which is based on proximal policy optimization [3]. The algorithm is modelled as a multi stage Markov decision process where every stage stores the system state of the placements of the previous operations on GPU and CPU devices. To select the device for placing the next operation, a probability distribution is sampled. After placing the operation, the system goes to the next stage with a new placement state where the previous operation has been placed. This process is repeated until all the operations of the neural network are placed. The reward of the Markov decision process is the training time of the neural network with the final placement state. With this reward, the set of the probability distributions are updated and the placements are repeated. The objective of spotlight [8] is to constantly update the set of probability distributions towards desirable distributions which provide near-optimal training times.

In sum, many models are being studied for device placement or end-to-end computation graph optimization. Unfortunately the topic of studying the impact of the graph models used has not received sufficient attention.

III. Prototype Design

In this section of the paper, we will discuss about the project goals and the workflow to achieve the goal of this research.

A. Research Questions

Research Question: To what extent does the performance of models(MoNet, GAT, GatedGCN, and GatedGCN-PE) improve by changing the Parameter Configuration on traditional Node Classification Datasets such as PATTERN and CLUSTER.

We are using the same dataset used by Dwivedi [2]. The aim behind is to further tune the models which they got in benchmarking graph neural network, and use the best model to train and perform device placement for machine learning computation graphs. Towards this aim, we propose the second research question.

B. Work Flow

1) Data Selection: In this work, We have used the traditional node classification dataset, PATTERN and CLUSTER. In order to evaluate the node classification model performance, we train, test and validate them over these datasets successively.

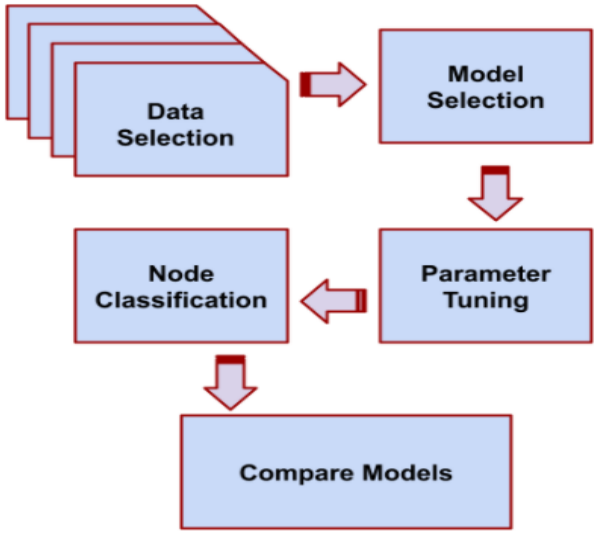


Figure 2. Flow Chart

2) Model Selection: Once the data is chosen, we select a model to perform classification over the selected dataset. The Graph Neural Network(GNN) models used in this paper are, GatedGCN(Gated Graph Convolution Networks), MoNet(Gaussian Mixture Model Networks) and GAT(Graph Attention Networks). More details on them is presented later in this section. We should note that we selected these models as the work of Diwedi [2] finds them to be the most competitive for tasks like node classification.

3) Parameter Tuning: After picking the right GNN for node classification task, the hyperparameters and network parameters are tuned. At first, the hyperparameters such as edge feature and position encoding are tested with their defaults values FALSE and TRUE respectively. later, the model is run with contradictory logical values of the same.

Meanwhile, the network parameters such as number of layers varied from Four to Eight, Sixteen and Thirty Two gradually, and the number hidden dimensions are changed to 100 , 200 from default value 70.

4) Node Classification: After successful tuning of model configuration, The selected model is trained, tested and validated over the selected dataset.

5) Compare Models: Finally, The results of models with different configurations are compared using Accuracy as an evaluation metric, In order to select an optimal model for node classification.

Table I
DATA SPLIT

	Train set	Test set	Validation set
PATTERN	10000	2000	2000
CLUSTER	10000	1000	1000

IV. EXPERIMENT SETUP

A. Datasets

We are using PATTERN and CLUSTER datasets which were generated with the Stochastic Block Model (SBM). For random graphs, the stochastic block model is a generative model. This model is widely used in statistics,machine learning and network science [10]. This model is used for producing graphs which include communities where the subsets are connected with each other with particular edge densities. Pattern dataset is for node-level tasks of graph pattern recognition and the cluster is for semi-supervised graph clustering [2]. SBM datasets consider node-level tasks of graph pattern recognition which is mainly used to model communities in social networks. The way by which SBM assigns communities to each node of the graph is, if 2 nodes belong to the same community they are connected with probability p else with probability q. Where q value acts as the noise level.

PATTERN - We generate graphs G with 5 communities with sizes between 5 to 35. The sizes are based on random selection. Each community's SBM is $p = 0.5, q = 0.35$, and a uniform random distribution is used to produce the node features on G. 100 patterns P are randomly generated which are composed of 20 nodes with intra-probability $p_P = 0.5$ and extra-probability $q_P = 0.5$. The node features for P are also generated with a uniform random distribution [2]. The graphs are of sizes 44-188 nodes. The output node labels have value 1 if the node belongs to P and value 0 if it is in G. The dataset consist of 14000 graphs and 2 classes. The average nodes and the average edges are 117.47 and 4749.15 respectively.

CLUSTER - We generate 6 SBM clusters with sizes chosen at random from 5 to 35 and probabilities $p = 0.55, q = 0.25$. The graphs are of sizes 40-190 nodes. Input for each node can be a feature of value in 0,1,2,3,4,5,6. If the value is 0, the class of the node is unknown and will be inferred by the GCN. The node belongs to class 0 if the value is 1, class 1 if the value is 2, and so on until the value is 6, which corresponds to class 5. Each community has only one labelled node, which is assigned at random. The community/cluster class labels are used to define the output node labels. The dataset consist of 12000 graphs and 6 classes. The average nodes and the average edges are 117.47 and 4301.72 respectively.
Data has been splitted in 3 sets i.e. train set, test set and validation set (shown in Table I).

B. Models

The following models were chosen for the parameter configuration task as they performed best for the node classification task [2]. The anisotrpic GNNs (GAT, MoNet, GatedGCN) performed better than the isotropic models (GCN, GraphSage).

- **Graph Attention Networks (GAT).** It obtains anisotropy in the neighbourhood aggregation function by employing the attention mechanism [14]. The network has similarities to Transformer. It comprises of a multi-headed architecture which helps to increase the learning capacity. It learns by calculating a mean from each node's neighbourhood features which are sparsely weighted by considering the importance of each neighbour.

- **Gaussian Mixture Model Networks (MoNet).** The Gaussian Mixture Model (GMM) is used to develop the Monet model architecture for applying it to graphs and also for learning from them.

- **Gated Graph Convolution Networks (Gated GCN)**. Another anisotropic variant of GCN is developed by Xavier [11] considering residual connections, batch normalization and edge gates. They are computationally and memory efficient since they rely on sparse matrix multiplication. They are improved version of isotropic GCNs. They extensively leverage the attention and gating mechanisms for the consistent performance across the graph as well as for node and edge level tasks. Gated GCN along with positional encodings (PE) is a solution for the theoretical limitation of low structural expressivity of GCNs and leads to improved performance on datasets without positional information [2].

The model is made [11] by combining Vanilla Graph Convolution Network and Edge gating mechanism [12].

$$h_i^{l+1} = f_{G-GCN}^l(h_i^l, \{h_i^l : j \to i\}) = ReLU(BN(U^l h_i^l + \sum n_{ij} \odot V^l h_j^l))$$

Where l is the layer of GNN, i, j are nodes, h_i^l weight at layer l for node i, BN is the batch normalization. This model uses both the feature vectors of the node and its neighbouring nodes. For anisotropic GCN, we use edge features which is computed by the node features and the features of neighbouring nodes. The high level diagram can be seen in Fig 3.

C. Hyperparameters

An initial learning rate is set to $5 * 10^{-5}$ which is reduced by half if the validation loss does not improve after 5 epochs. The training is stopped either when the learning rate has reached $1 * 10^{-6}$, or the computational time reaches 12 hours.

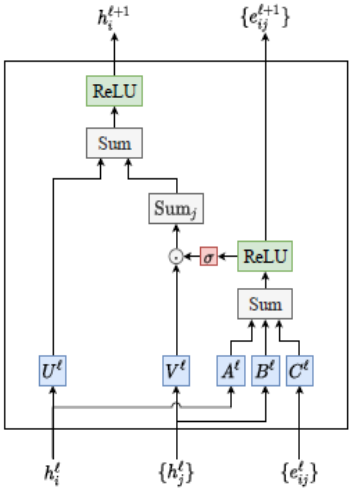


Figure 3. GatedGCN [2]

We compare the model accuracy and performance within the maximal computational time of 12 hours. The number of hidden layers are selected from 4,8,16 and 32. We use 70 as the number of hidden dimensions. Since GCNs are not able to differentiate isomorphic nodes we use positional encoding (PE) of nodes for Gated GCN model. For our experiment we are using Laplacian PE with 10 eigenvectors. In all the total number of parameters are as follows-

$$4layers \approx 100000$$

$$8layers \approx 200000$$

$$16layers \approx 400000$$

$$32layers \approx 800000$$

We have started by executing the default hyper-parameters [2](the best parameters as per Dwivedi) and started to change the parameters such as Edge features, Positional Encoding, hidden dimensions, and network layers.

D. Softwares/Libraries

We have chosen Jupyter notebook and Google collab for the execution of the benchmarking Graph Neural Network. The benchmarking framework consists of all the datasets and the models [16]. The benchmarking infrastructure is made of Pytorch and DGL. This framework consists of Data pipelines, GNN layers models, Training and evaluation functions, Network and hyperparameter configurations and scripts for reproducibility. The other libraries imported in the benchmarking framework include tensorboardX, json, tqdm etc.

E. Devices

The devices that we used are Nvidia gtx 1650 with Intel i5 CPU having 16GB ram. Another machine of execution was the Google colab. Configuration of colab is the following gpus: Tesla P100-PCIE-16GB, Tesla V100-SXM2-16GB, Tesla T4, Tesla K80.

V. Evaluation

In this section we will discuss about the evaluation metric used in this research. Dwivedi [2] in their research came up with benchmarking graph neural networks and used weighted accuracy with respect to class size. We are using same evaluation metric so as to compare the performance with theirs.

A. Accuracy

The accuracy of the classification algorithm is the measure of correctly classifying the data instances.

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Where TP is True Positive, TN is True Negative, FP is False Positive, FN is False Negative. [13].

B. Weighted accuracy

Weighted accuracy is compared by taking all the average over all the classes of the fraction of correct things in the class. More formally, the ratio of the number of correctly predicted instances of a particular class to the total number of instances of that class [15]. In this research, the weighted accuracy is taken with respect to the class size [2].

C. Accuracy Curve

Among the most the curves to understand the progress of Graph Neural Networks is the accuracy curve. A important curve is the one with both training and validation accuracy. The figure 4 is an example how the accuracy graph looks like [13].

VI. Results

In this section, we present the results and findings of the research. Models that were explored are GAT, MoNET, and GatedGCN. The results are listed in the Table II in the appendix.

The best model they [2] found was 16-layered GatedGCN with positional encoding had 86.5% accuracy for PATTERN and 76.08% accuracy for CLUSTER dataset. For the same configuration we have achieved 85.9% accuracy in PATTERN and 75.3% accuracy in CLUSTER dataset. This difference can be because of the different environments and setup of the code execution.

After tuning the models, we found that 4-layer GatedGCN

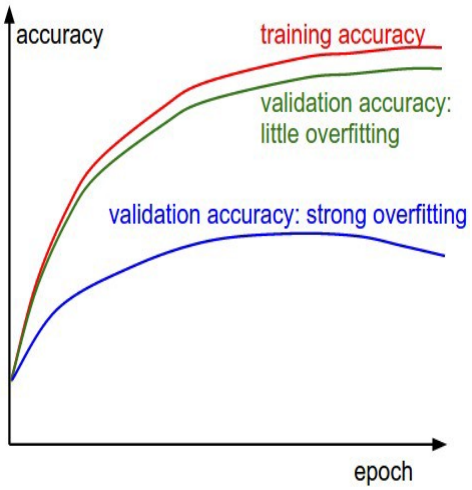


Figure 4. Accuracy Curve [13]

with positional encoding having 100 hidden dimensions gave the accuracy of 86.1% in PATTERN and 73.5% in CLUSTER. However it is not the best model for CLUSTER dataset. In CLUSTER, 16-layered GatedGCN with positional encoding gave the substantial improvement in accuracy(75.3%), which is in line with the research done previously [2]. The accuracy of 16-layered GatedGCN with positional encoding was the third best model for PATTERN dataset with slight difference of 0.2 in accuracy. But this model has high variance. Train Accuracy in CLUSTER dataset is 82.1% and Test accuracy is 75.3%.

High variance in the model is not desirable because of less generalizability. Hence for considering best models, we are taking accuracy and variance of models into account. So we consider the below models into account for the device placement of the Machine Learning models (in the order of their rank).

- GatedGCN with 8 layers and 70 hidden dimensions
- GatedGCN with 4 layers and 100 hidden dimensions
- GatedGCN with 16 layers and 70 hidden dimensions

We experimented with both combinations of edge feature (True/False). We found that edge features did not helped enhancing the accuracy. Considering 4-layered GatedGCN with 70-hidden dimensions, the accuracy declined by 1% when using edge features. On the other hand, positional encoding enhanced the performance of the model. All these GatedGCN models are with positional encodings (Can be seen in detail in Table II in appendix). We can first run the top-3 models with the Machine Learning dataset. And find the accuracy of each model.

VII. Conclusion and Future Work

A. Conclusion

The increasing complexity of machine learning and deep learning models has accelerated the need of reduce the run-time of models. Speed can be increased by optimising the models as well as by making our compiler more efficient with handling the computation graphs of models. Computation graphs can be optimised by Device Placement, Operation Scheduling, and Operation fusion. In this paper, we present a optimization solution for the computational graph execution. We have considered 3 models for our task which are GAT, Monet, GatedGCN since they performed best for the node classification task [2].

To answer the Research Question, which was to find the best model and the parameter configuration to do the node classification task. We have considered two synthetic datasets generated by SBM (PATTERN and CLUSTER) for the node classification. As a result we have found that 8-layered GatedGCN with positional encoding having 70 hidden dimensions performed best for the node classification task. We have considered weighted accuracy with respect to the class size as our evaluation metric. In other words, it is the average node-level accuracy weighted with respect to the class sizes. We have also considered variance of the model into consideration for better generalizability of the results in the unseen dataset. Second best and third best models are 4-layered GatedGCN with 100 hidden dimensions and 16-layered GatedGCN with 70 hidden dimensions respectively. The result we found was in line with the previous research done by dwivedi [2].

The other models (MoNET and GAT) were not good enough. MoNET performed good in PATTERN dataset with around 85-86% test accuracy. But the performance dropped significantly with CLUSTER dataset with test accuracy varying between 34-64% .
Similarly, GAT gave less accuracy in both PATTERN and CLUSTER dataset.

For device placement, number of classes in the node classification algorithm should be 2, i.e. one for CPU and another for GPU. From this, we can identify the nodes at the runtime to be placed at the device optimally.

B. Future Work

In this research, device placement is planned to be done by the node classification. Performance can be improved by using using deep reinforcement learning [4]. From deep reinforcement learning, each node can be placed on to a particular device (CPU/GPU) with the help of reinforcement learning where policy is set to minimize execution time. Reinforcement learning can learn and predict the unseen nodes, which may come from new machine learning models [4], hence giving better generalizability to the model. Further,

more complicated tasks, such as database optimization can be done by using Deep Reinforcement Learning. To further enhance the performance, Operation Scheduling and Operation Fusion can be implemented.

ACKNOWLEDGMENT

We thank Prof. Gunter Saake for providing us the opportunity to work on this project.We also thank Gabriel Campero Durand for his constant help and guidance.

REFERENCES

[1] Gao, Yuanxiang, Li Chen, and Baochun Li. "Spotlight: Optimizing device placement for training deep neural networks." In International Conference on Machine Learning, pp. 1676-1684. PMLR, 2018.

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

[3] Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In International conference on machine learning, pp. 1889-1897. PMLR, 2015.

[4] Zhou, Yanqi, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter Ma, Qiumin Xu, Hanxiao Liu et al. "Transferable graph optimizers for ml compilers." arXiv preprint arXiv:2010.12438 (2020).

[5] Hamilton, William L. "Graph representation learning." Synthesis Lectures on Artifical Intelligence and Machine Learning 14, no. 3 (2020): 1-159.

[6] Mirhoseini, Azalia and Goldie, Anna and Pham, Hieu and Steiner, Benoit and Le, Quoc V and Dean, Jeff. "A hierarchical model for device placement". International Conference on Learning Representations. 2018.

[7] Addanki, R., Venkatakrishnan, S. B., Gupta, S., Mao, H., & Alizadeh, M. (2019). Placeto: Learning generalizable device placement algorithms for distributed machine learning. Advances in Neural Information Processing Systems, 32.

[8] Mirhoseini, Azalia, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. "Device placement optimization with reinforcement learning." In International Conference on Machine Learning, pp. 2430-2439. PMLR, 2017.

[9] Pellegrini, François. "A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries." In European Conference on Parallel Processing, pp. 195-204. Springer, Berlin, Heidelberg, 2007.

[10] Abbe, Emmanuel. "Community detection and stochastic block models: recent developments." The Journal of Machine Learning Research 18, no. 1 (2017): 6446-6531.

[11] Bresson, Xavier, and Thomas Laurent. "Residual gated graph convnets." arXiv preprint arXiv:1711.07553 (2017).

[12] Marcheggiani, Diego, and Ivan Titov. "Encoding sentences with graph convolutional networks for semantic role labeling." arXiv preprint arXiv:1703.04826 (2017).

[13] https://towardsdatascience.com/useful-plots-to-diagnose-your-neural-network-521907fa2f45. 05/08/2021. 16:00.

[14] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).

[15] https://devblogs.microsoft.com/cse/2015/11/29/data-preparation-the-balancing-act/. 05/08/2021. 16:00.

[16] https://github.com/graphdeeplearning/benchmarking-gnns Repository for benchmarking graph neural networks

Device placement optimization with reinforcement learning. pages 2430–2439, 2017.

APPENDIX

Table II
RESULTS

Model	Epochs	Layers	Train Accuracy	Validation Accuracy	Test Accuracy	Hidden Dimensions	Edge Feature	Positional Encodings
PATTERN								
	334	4	86.5	85.9	86.1	70	false	true
	246	4	85.1	84.9	85.1	70	true	true
	343	4	86.6	85.9	86.1	100	false	true
	259	4	86.7	85.9	86.1	200	false	true
	132	4	86.6	85.9	86	300	false	true
	321	8	86.5	85.9	86	70	false	true
	120	16	86.4	85.8	85.9	70	false	true
	120	32	86	84.7	84.7	70	false	true
GatedGCN	272	4	84.5	84.8	85	70	false	false
	314	4	85.9	85.5	85.6	70	false	true
	343	4	86	85.4	85.7	70	true	true
	120	4	85.7	85.3	85.5	100	false	true
	120	4	86.1	85.3	85.5	200	false	true
	120	4	86.5	85.1	85.2	300	false	true
	324	8	86.4	85.3	85.4	70	false	true
	302	16	86.2	85.4	85.5	70	false	true
	304	32	86.6	85.3	85.3	70	false	true
MoNet	275	4	85.9	85.4	85.6	70	false	false
	592	4	76.2	73.2	73.4	70	false	true
	795	4	76.9	73.8	73.9	70	true	true
	120	4	81.7	73.8	73.7	100	false	true
	120	4	87.8	69.7	69.7	200	false	true
	120	4	92.7	53	52.9	300	false	true
	414	8	81.6	77.8	77.9	70	false	true
	260	16	83.8	78.4	78.5	70	false	true
	233	32	86.6	78.4	78.4	70	false	true
GAT	422	4	75.5	73.2	73.3	70	false	false
CLUSTER								
	293	4	75.1	73.3	73.5	70	false	true
	392	4	75.4	73.9	73.6	70	true	true
	338	4	76.3	73.6	73.5	100	false	true
	120	4	79.3	73.5	72.3	200	false	true
	120	4	82.3	71.3	70.8	300	false	true
	219	8	78	75	74.8	70	false	true
	170	16	82.1	75.5	75.3	70	false	true
	120	32	82.6	75.7	75.2	70	false	true
GatedGCN	287	4	60.5	59.1	59.3	70	false	false
	120	4	64	49.4	49.9	70	false	true
	422	4	63.7	62.9	63	70	true	true
	391	4	65.1	63.7	63.9	100	false	true
	120	4	66.5	33.8	34.5	200	false	true
	278	4	69	63.5	63.5	300	false	true
	120	8	72	57.8	58.5	70	false	true
	120	16	76.7	50.8	51.2	70	false	true
	120	32	76.7	72.3	72	70	false	true
MoNet	311	4	63.5	62.8	62.6	70	false	false
	286	4	57.7	56.8	56.8	70	false	true
	289	4	57.4	56.8	56.7	70	true	true
	323	4	63.4	57.4	57.8	100	false	true
	11	4	60.9	45.8	46.2	200	false	true
	174	4	75.9	53.9	54.1	300	false	true
	373	8	66.6	65.2	65.5	70	false	true
	402	16	71.8	68.9	69	70	false	true
	340	32	72.8	69.5	69.1	70	false	true
GAT	286	4	57.7	56.8	56.8	70	false	false