GEOMETRIC DATA ANALYSIS

SEMINAR: REPORT SUBMISSION

Review on the study "Towards Sparse Hierarchical Graph Classifiers" by Cangea et al.

Author:

SURESH KUMAR CHOUDHARY - 5504392

Freie Universität Berlin
Computer Science department
Berlin, 14195, Germany

Submitted: February 22, 2022

ABSTRACT

In this study, sparse hierarchical graph Classification architecture was implemented to predict a class label associated with an entire graph. The purpose of this study was to reduce memory usage during model training without compromising with the accuracy as previous prominent methods were not scalable to large graphs. The performance of this architecture was compared against the state-of-the-art approach and verified on several established graph classification benchmarks in graph neural network design. This method had the accuracy at most in the range of 1% as state-of-art algorithm (diffpool[1]). State of art approach(diffpool[1]) requires a quadratic $\mathcal{O}(kV^2)$ storage complexity to be executed, however, the proposed approach requires only $\mathcal{O}(V+E)$storage complexity, where k, V and E are the pooling ratio, number of nodes and edges in the graph respectively. Here author integrated numerous current advances in graphical neural network layout to illustrate that competitive hierarchical graph classification outcomes are viable without sacrificing sparsity..

1 Introduction

The study from Cangea et al.[2] looks into the problem of graph classification, which involves predicting a class label for an entire graph. Since an image can be considered a special case of a "grid-graph," this work is analogous to the image classification task. As a result, looking for a CNN generalisation to graphs seems natural.

Several proposals for an abstraction of the convolutional layer to graphs have surfaced in recent years, leading to the development of various graph convolutional layers that have proven to be quite successful on a variety of node classification benchmarks. The generalisation of the pooling layer, on the other hand, received less attention, and the results on graph classification benchmarks aren't as promising as they could be. The most typical strategy is to apply a global pooling layer on the graph's node features, which reduces the graph's representation to a single "virtual" node that summarises all of the original nodes' features. The disadvantage of this method is that it is unable to preserve the graph's topological information while summing up the node features.

The pooling layer should be comparable to the local pooling layer applied to images in order to account for the graph's structure, with the exception that graphs do not have locality. In order to construct the new graph representation, a measure of the relevance of the nodes might be employed as a replacement parameter. The graph can be coarsened using this method, which uses a hierarchical pooling layer to keep only the most important nodes in each network. However, the preceding prominent approach to pooling has quadratic memory requirements during training and is consequently now no longer scalable to huge graphs. The author's objective was to present an architecture and demonstrate performance that is comparable to the state-of-the-art approach on graph classification benchmarks but with linear storage complexity.

2 Related Work

Aiming with a better understanding of the used methods, we are interested in papers that made use of graph-based neural networks for graph classification as well as papers with a focus on representation learning and factors that influence their performance.

Krizhevsky et al.[3] presented deep convolutional neural networks(CNN) based classification method trained on ImageNet[4] dataset. Graph classification can be considered as a direct generalization of classification by Krizhevsky et al.[3]. Therefore, it's far herbal to research and generalize CNN factors to graph. Bronstein et al.[5] explained how geometric deep learning came into the picture, and why CNN fails on non euclidean domains (i.e. graph and manifolds). Hamilton et al.[6] introduced the representation learning to represent, or encode, graph structure in order that it may be effortlessly exploited via way of means of machine learning models. Battaglia et al.[7] explored how using relational inductive biases inside deep learning architectures can pave the way for learning the graph network.

Generalising the CNN to graphs has been a really active area of research, with several graph convolutional layers[8–12] broached in recent times, notably advancing the state-of-the-art on many challenging node classification benchmarks, also as link prediction. Conversely, generalising pooling layers has received significantly smaller ranges of interest via way of means of the research community.

Duvenaud et al.[13] introduced convolutional neural network whose inputs are graphs of arbitrary size and shape. Previous approaches focused on the generalization of the pooling layer to graphs through aggregating node representations. Aggregating node representations in pooling step had been done after each message passing step[13] or after the final message passing step[11,14,15] before passing the information to linear multi layer perceptron layer(MLP) to classification, unfortunately these approaches prevented the model from learning information about the topology of the graph. So, to obtain the topology information of graphs, other approaches[1,8,9,16–21] involving clustering techniques had been proposed. The assignment of nodes to clusters enabled a hierarchical coarsening of the graph. However, the majority of these approaches assumed a fixed and deterministic cluster assignment obtained by running a clustering algorithm (i.e. GraClus algorithm citedhillon2007weighted) on the graph nodes, and therefore the cluster assignment (i.e. Kmeans[20]) was not adapted to the underlying data.

The paper by Ying et al.[1] about "hierarchical graph representation learning with differentiable pooling" proposes the first CNN that can be trained with a learnable pooling operator, named DiffPool. The main idea behind DiffPool is that the assignment of nodes to clusters can be learned in a differentiable way , resulting in the cluster assignment being adjusted to fit the underlying data. DiffPool computes soft clustering assignments of the nodes, and through certain restrictions and rules applied to the assignment, the clustering it learned eventually converges to a quasi-hard clustering.

The key drawback of DiffPool is the quadratic $\mathcal{O}(kV^2)$ storage complication with a fixed pooling ratio k. This is owing to the requirement to store an entire assignment matrix which relates the nodes from the original graph to the nodes from the pooled graph in an all-pairs fashion. Consequently, the utility of DiffPool is prohibitive for large graphs.

Cangea et al.[2] leveraged latest developments in graph neural network design[10,21,22] to exhibit that without sacrificing the sparsity one can obtain satisfactory performance on end-to-end graph convolutional architectures with pooling. This study demonstrated performance that was similar to versions of DiffPool on four standard graph classification benchmarks, but with a linear $\mathcal{O}(V+E)$ storage complexity.

3 Data

Cangea et al. chose the data from the biological (Enzymes, Proteins, D&D) and scientific collaboration (Collab) datasets[23] to evaluate the graph neural network architecture. All the four datasets were graph-labeled with size 600,1113,1178,and 5000 respectively. The statistics of these can be found in Table 1. The datasets D&D,Enzymes, and Proteins constitute macromolecules. Nodes of Proteins constitute secondary structure factors and are annotated by their type, i.e., helix, sheet, or turn, in addition to numerous physical and chemical information. An edge connects nodes if they're adjacent alongside the amino acid series or one of three nearest adjacent in space. Enzymes dataset was derived from the BRENDA database (Schomburg et al.[24]. Here, the task is to assign enzymes to one of the six EC top-stage classes, which replicate the catalyzed chemical reaction. Similarly, the dataset Proteins was derived from (Dobson et al.[25]), and the venture is to are expecting whether or not a protein is an enzyme. The dataset D&D's nodes represent amino acids and edges their spatial proximity. COLLAB datasets were deduced from scientific collaboration networks where each graph represents the ego-network of a researcher

Table 1. Data Overview

Type	dataset	Size	No. of Classes	Avg. No. of Nodes	Avg. No. of Edges	Node Attr. (Dim.)
Biological	Enzymes	600	6	32.63	62.14	18
	Proteins	1113	2	39.06	72.82	29
	D & D	1178	2	284.32	715.66	-
Scientific collaboration	Collab	5000	3	74.49	2457.78	-

4 Methods

The initial setup was the same as the standard graph-based machine learning setup where authors considered only undirected and unweighted graphs. For the task of classifying a graph, a dataset contained a set of graphs, each containing a variable number of nodes, with some features linked and connected in different ways. Each graph contained N number of nodes and F number of features. Each input graph was represented by two matrices. First, the matrix of node features given by $X \in \mathbb{R}^{N\times F}$, and second an adjacency matrix, $A \in \mathbb{R}^{N\times N}$ (A is binary and symmetric). The node degree information was used as artificial node features in circumstances where the graph is featureless.

Graph classification with CNN-network required the standard convolutional layer, pooling layer, and flattening layer. The flattening layer in this case was readout layer whose output was used to do the final prediction. Each of the layers has been explained further in detail.

4.1 Convolution layer:

The used graph convolutional layer was able to extract and rearrange the features of a graph's nodes in an inductive manner. Since authors expected the prediction of unseen graph structures to not depend on the known graph structures, that's why they used mean pooling whose formula is given by equation(1):

$$MP(X,A) = \sigma(\hat{D}^{-1}\hat{A}X\theta + X\theta') \tag{1}$$

here $\hat{A} = A + I_N$ is the adjacency matrix with self connections and its degree matrix is given by $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$. For determining the output of σ , they used rectified linear activation (ReLU) function. $\theta, \theta_r \in \mathbb{R}^{F\times F'}$ were linear transformations applied to

every node. Here the transformation done by θ_l were skip connections in order to keep the central node information preserved. The output of convolulational layer was size of $N \times F'$ matrix with input matrix X ($N \times F$).

4.2 Hierarchical pooling layer

The hierarchical pooling layer coarsens the graph by dropping a few of its nodes, in this way down-sampling the graph representation in hierarchical manner. The approach utilised in this paper is different from the one utilised for DiffPool and it has been proposed some time recently in the Graph U-Net research paper[21]. In order to avoid using quadratic $\mathcal{O}(kV^2)$ storage complexity of diffPool, the hierarchical pooling layer decreases the graph size with a pooling proportion $k \in (0,1]$, so that, after the application of the layer, the graph will remain with kN nodes against the beginning N nodes. The nodes to be dropped are chosen based on the degree of projection against a learnable vector $\vec{p} \in \mathbb{R}^F$. All the features of nodes of a graph are projected to 1D through $\vec{p}$ and the resulting values measure how much information of each node can hold when projected onto the direction of $\vec{p}$. Since authors wanted to get as much information as possible from the original graph , they used the projection value for each node as a score of that node's importance in the graph. Consequently, to coarsen the graph , they performed the k-top operation to select the indices of the higher-scoring nodes. These indices are then used to intersect the adjacency matrix and the node feature matrix accordingly, keeping only the most important nodes and removing the others. To allow gradients to flow into $\vec{p}$, the projection scores were also used as gating values: the intersected node's feature matrix was multiplied element-wise by the projection scores so that nodes with lower scores will have less significant feature retention, and vice versa. The whole pooling operation, with (X, A) as input, can be visualised in Figure 1, and can be expressed mathematically by the equation(2):

$$\vec{y} = \frac{X\vec{p}}{||\vec{p}||} \quad \vec{i} = top-k(\vec{y},k) \quad \tilde{X} = X(\vec{i},:) \quad X' = (\tilde{X} \odot tanh(\vec{y}(\vec{i})) \quad A' = A_{\vec{i},\vec{i}} \tag{2}$$

where, (X',A') is the pooled graph from the original graph (X, A), $||.||$ is the L2 norm, $\odot$ is the element wise broadcasted multiplication, and $top-k$ is the operation that select the indices of the $\lceil kN \rceil$ nodes with higher score. Hierarchical pooling layer drops the nodes of graphs rather than aggregating unlike the diffpool, and it doesn't require any extra storage. The projection and slicing operations are performed directly on the original input matrices. The projection and slicing operations are performed directly on the original input matrices. Hierarchical pooling operations are performed immediately after each graph convolution layer, which together form a conv−pool block (denoted as ℓ−th block).

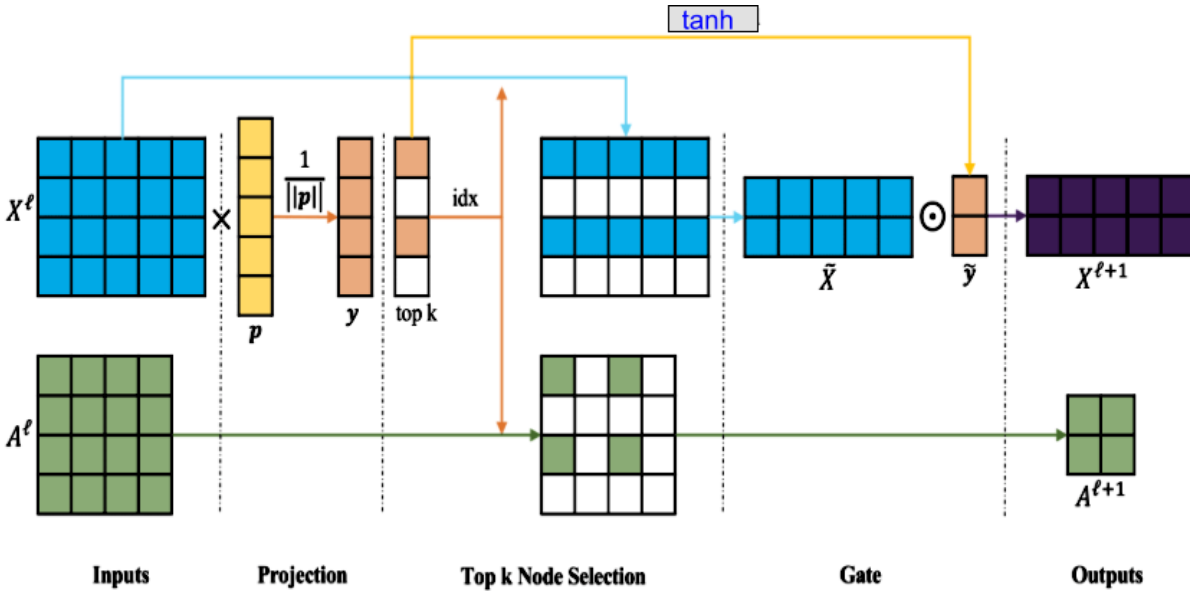

Figure 1. Pooling operation[21] at ℓ−th conv−pool block

4.3 Readout layer

The last step involves flattening the already learned representations to fixed-size vectors. To do this, authors used the three different types of global pooling operations. After each conv−pool block, the resulting information was saved by the concatenation of the global average pooling operation and the global maximum pooling operation applied to the block's output. Finally, all concatenation of global pooling results obtained by multiple conv−pool blocks was aggregated using the global sum pooling operation[26,27] and submitted to an MLP for obtaining final predictions. This whole process has been depicted in Figure 2. The mathematical representation of this was given by the equation(3):

$$\vec{s}^{(\ell)} = \frac{1}{N^{(\ell)}} \cdot \sum_{i=1}^{N^{(\ell)}} \vec{x}_i^{(\ell)} \,||\, \max_{i=1}^{N^{(\ell)}} \vec{x}_i^{(\ell)} \qquad \vec{s} = \sum_{\ell}^{L} \vec{s}^{(\ell)} \qquad predictions = MLP(\vec{s}) \tag{3}$$

where $N^{(\ell)}$ is the number of nodes in the graph, $\|$ denotes the concatenation operation, and $\vec{x}_i$ is the i−th node's features vector.

Aggregation across layers is important because it can retain information on the graph at different scales of processing. It also holds information about small graphs that can be pooled quickly if the number of nodes is too small.

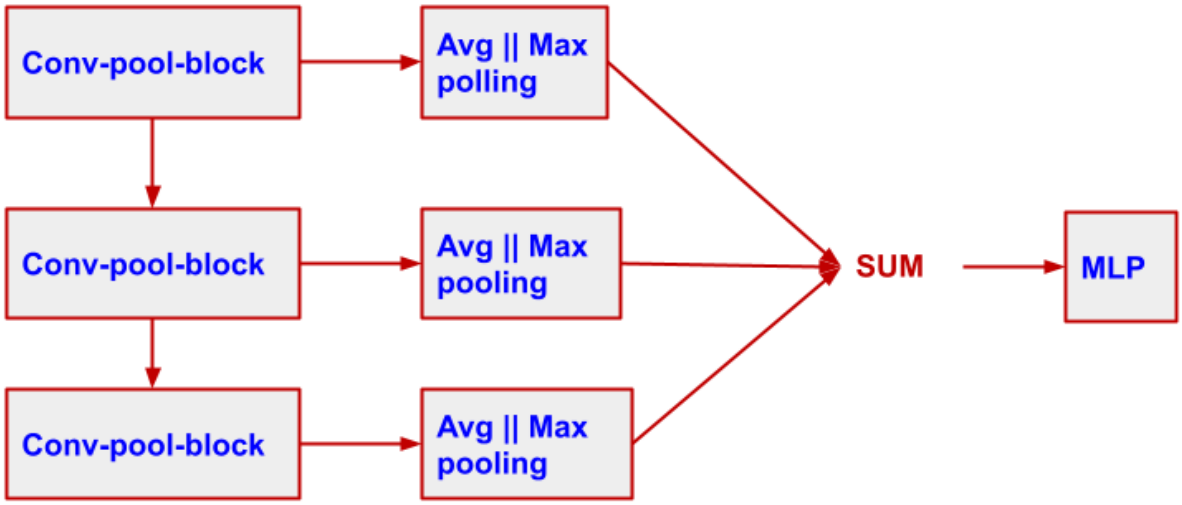

Figure 2. Readout Layer

5 Model Architecture

The entire architecture of the model consists of three transformation blocks (conv−pool blocks). Each of them is represented by a graph convolutional layer followed by a hierarchical pooling layer, and for each transformation block, a series of global average pooling and maximum global pooling operations are stored on the output. After the three transformation blocks, their representations were assembled sequentially by a global aggregate aggregation layer (global sum pooling), whose fixed-size output was submitted to the MLP to obtain the final predictions. The architecture of the proposed model (entire pipeline with only two conv−pool blocks) may be visualised in Figure 3.

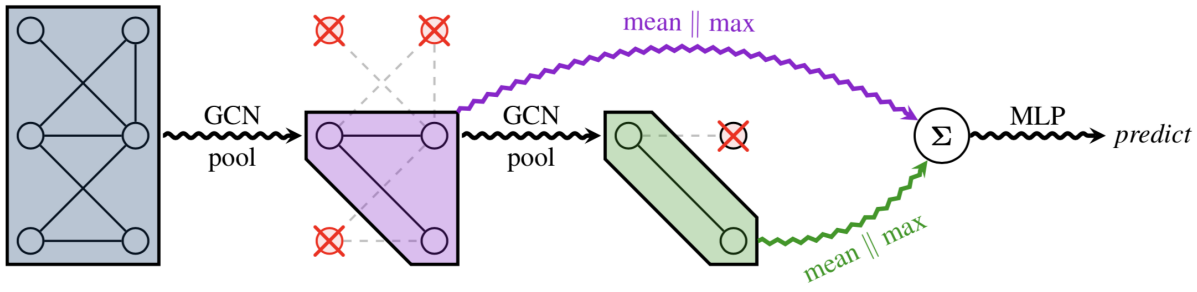

Figure 3. The full pipeline[2] of proposed model (with two conv-pool blocks, k = 0.5), taking advantage of several combinations of interlocking convolution/pooling layers (unlike DiffPool, which drop instead of aggregate nodes), plus JK-net[26,27] style summary , collects information at different scales.

6 Experiments

6.1 Evaluation metric and procedure

To check the relevance of the produced hierarchical representations, the proposed graph neural network architecture was evaluated on dataset described in Section 3. Aiming with evaluating the performance of proposed model, classification accuracy(4) was used:

$$Accuracy(\%) = \frac{Number - of - correct - predictions}{Total - number - of - predictions} * 100 \tag{4}$$

The results reported in the paper were achieved by performing a 10-fold cross-validation test for each dataset. Cross-validation is a resampling technique used to validate machine learning models against a limited set of data samples. The procedure takes a single parameter named k, this shows the number of groups in which a particular data sample is divided. Therefore, this procedure is often referred to as k-fold cross-validation. If you select a specific value for k, you can use it in your model as in form of k. For instance, k = 10 is a 10-fold cross-validation test. The general process of it is as follows:

1. shuffle the dataset randomly
2. divide the dataset into k groups
3. for each Unique group:
   (a) Use group as holdout or test dataset
   (b) Fit model with remaining group as training dataset
   (c) Build a Model, evaluate on test set
   (d) Retain evaluation scores and discard model
4. Summarize the evaluation scores of models

6.2 Model parameters

As mentioned earlier, the model consists of three convolutional pooling blocks, each containing a graph convolutional layer with 128 (Enzymes and Collab) or 64 (Proteins and D&D) features, followed by a hierarchical pooling step. The pooling ratio $k = 0.8$ ensures that after coarsening each graph, enough information is obtained by retaining 80% of the nodes in the graph. A learning rate of 0.005 was used for proteins and 0.0005 for all other datasets. Adam optimizer[28] was used to optimize the cost function for 100 epochs on Enzymes, 40 on Proteins, 20 on D&D and 30 on Collab. The summary of model parameters can be seen in Table 2.

Table 2. Model parameters used by Cangea et al.[2]

Dataset	No. of Feature Layers	Learning Rate	Epochs
Enzymes	128	0.0005	100
Proteins	64	0.005	40
D&D	64	0.0005	20
Collab	128	0.0005	30

6.3 My implementation

For this review, I replicated the same model architecture described by Cangea et al. to try to get the same results. I used Python 3.6 and pytorch to implement all layers and experimented with Google Colab GPUs. I utilised the pytorch geometric library with TopKPooling and GCNConv functionality. I used the same parameters mentioned in Table 2. The model was trained on all four datasets and their respective results has been listed below in result section. Implemented code can be found on my github [*] account.

[*] https://github.com/sureshkuc/Freie-Universitat-Berlin/tree/main/GEOMETRIC-DATA-ANALYSIS

6.4 Results

When comparing graph classification performance, authors considered baselines based on graph neural nework (combined with various pooling methods) as well as modern kernel-based approaches. Graph neural nework-based methods were PATCHYSAN[16], GRAPHSAGE[22], ECC[18], SET2SET[29], SORTPOOL[30], DiffPoolL−Det[31], DiffPool−NLP, and Diffpool[1]. However, authors focused only on state of the art approaches (DiffPool variants) to compare proposed method performance.

Table 3 shows the performance results of the proposed model compared to the kernel based and graph neural network based models. The proposed model has successfully competed with the DiffPool variant as well as other methods and has obtained results that are very close to state-of-the-art method. The proposed method got the accuracy(%) 64.17, 78.59, 74.54, and 75.46 on datasets Enzymes, D&D, Collab, and Proteins respectively.

Table 3. Classification accuracy percentages in comparison

		Datasets			
Approach	**Model Name**	Enzymes	D&D	Collab	Proteins
Kernel Based	Graphlet	41.03	74.85	64.66	72.91
	Shortest-path	42.32	78.86	59.10	76.43
	1-WL	53.43	74.02	78.61	73.76
	WL-QA	60.13	79.04	80.74	75.26
Graph Neural Network Based	PatchySAN	-	76.27	72.60	75.00
	GraphSAGE	54.25	75.42	68.25	70.48
	ECC	53.50	74.10	67.79	72.65
	Set2Set	60.15	78.12	71.75	74.29
	SortPool	57.12	79.37	73.76	75.54
	DiffPool-Det	58.33	75.47	**82.13**	75.62
	DiffPool-NoLP	62.67	79.98	75.63	77.42
	DiffPool	**64.23**	**81.15**	75.50	**78.10**
Graph Neural Network Based	**Paper's Model**	64.17	78.59	74.54	75.46
Graph Neural Network Based	**My implementation**	46.36	76.81	72.92	73

Unlike DiffPool, proposed model by cangea et al. does not require quadratic memory, as shown in the experiments in Figure 4. Thus, the method represents a scalable hierarchical graph classification algorithm suitable for large real-world datasets. Authors also have supported this assertion empirically through experiments with random inputs and demonstrated that their method compares favorably with DiffPool on larger graphs, even if the pool layer does not remove any nodes (compared to a retention factor of 0.25 for DiffPool).

7 Discussion and Conclusion

As can be seen from the percentage of classification accuracy, the proposed model was able to achieve the goal of achieving performance comparable to DiffPool in a graph classification benchmark with a linear storage complexity. Proposed approach received accuracy at most with in range of 1% as state of art algorithm (diffpool).

Using models of hybrid methods, advance aggregation methods, and hierarchical graph pooling with structure learning can significantly improve the results of future studies.

Authors used the accuracy as evaluation metric assuming the data is balanced, however the data classes are not balanced. So there is a scope of research in this direction to find out the appropriate evaluation metric for the model.

Fortunately I was able to reproduce the results presented in the paper atmost with in the range of 1% range as proposed method for D%D, Proteins and Collab datasets, however for Enzymes dataset I got only 46% accuracy. As I didn't find the random seed setting in the paper by Cangea et al. and an official implementation is not published on the Internet so it would not be possible to get the same accuracies with same model parameters as mentioned by the authors, however one can get nearby accuracies of proposed method with the same.

The model accuracy depend on the number of epochs, learning rate, number of feature layers, and optimization method as well as studied case, one need to find the optimized parameters for the model to get best performance.

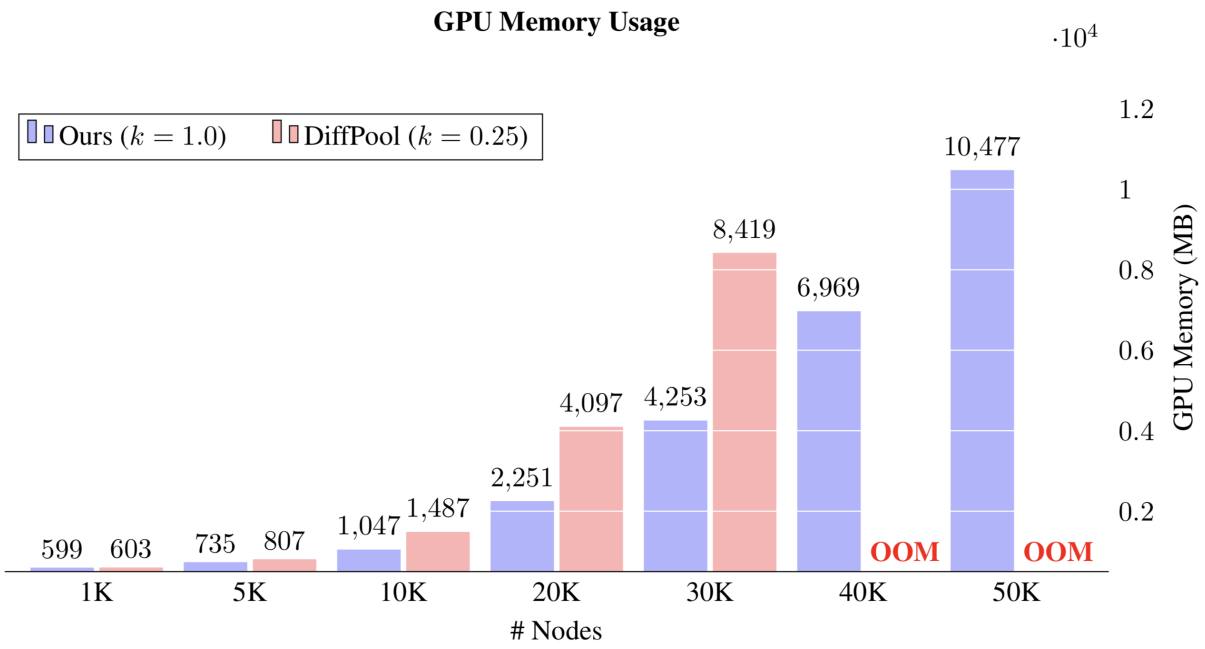

Figure 4. GPU memory usage of proposed method[2] (with no pooling; k = 1.0) and DiffPool (k = 0.25) during training ,from Cangea et al.[2], "OOM" denotes out-of-memory

References

1. Ying, Z. *et al.* Hierarchical graph representation learning with differentiable pooling. *Adv. neural information processing systems* **31** (2018).
2. Cangea, C., Veličković, P., Jovanović, N., Kipf, T. & Liò, P. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287* (2018).
3. Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Adv. neural information processing systems* **25** (2012).
4. Deng, J. *et al.* Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255 (Ieee, 2009).
5. Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A. & Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Process. Mag.* **34**, 18–42 (2017).
6. Hamilton, W. L., Ying, R. & Leskovec, J. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).
7. Battaglia, P. W. *et al.* Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).
8. Bruna, J., Zaremba, W., Szlam, A. & LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).
9. Defferrard, M., Bresson, X. & Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. *Adv. neural information processing systems* **29** (2016).
10. Kipf, T. N. & Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
11. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, 1263–1272 (PMLR, 2017).
12. Velickovic, P. *et al.* Graph attention networks. international conference on learning representations (2018). (2018).
13. Duvenaud, D. K. *et al.* Convolutional networks on graphs for learning molecular fingerprints. *Adv. neural information processing systems* **28** (2015).

14. Li, Y., Tarlow, D., Brockschmidt, M. & Zemel, R. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

15. Dai, H., Dai, B. & Song, L. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, 2702–2711 (PMLR, 2016).

16. Niepert, M., Ahmed, M. & Kutzkov, K. Learning convolutional neural networks for graphs. In *International conference on machine learning*, 2014–2023 (PMLR, 2016).

17. Monti, F. *et al.* Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 5115–5124 (2017).

18. Simonovsky, M. & Komodakis, N. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3693–3702 (2017).

19. Fey, M., Lenssen, J. E., Weichert, F. & Müller, H. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 869–877 (2018).

20. Mrowca, D. *et al.* Flexible neural representation for physics prediction. *Adv. neural information processing systems* **31** (2018).

21. Gao, H. & Ji, S. Graph u-nets. In *international conference on machine learning*, 2083–2092 (PMLR, 2019).

22. Hamilton, W., Ying, Z. & Leskovec, J. Inductive representation learning on large graphs. *Adv. neural information processing systems* **30** (2017).

23. Kersting, K., Kriege, N. M., Morris, C., Mutzel, P. & Neumann, M. Benchmark data sets for graph kernels. *URL http://graphkernels. cs. tu-dortmund. de* **29** (2016).

24. Schomburg, I. *et al.* Brenda, the enzyme database: updates and major new developments. *Nucleic acids research* **32**, D431–D433 (2004).

25. Dobson, P. D. & Doig, A. J. Distinguishing enzyme structures from non-enzymes without alignments. *J. molecular biology* **330**, 771–783 (2003).

26. Xu, K., Hu, W., Leskovec, J. & Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

27. Xu, K. *et al.* Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, 5453–5462 (PMLR, 2018).

28. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

29. Vinyals, O., Bengio, S. & Kudlur, M. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391* (2015).

30. Zhang, M., Cui, Z., Neumann, M. & Chen, Y. An end-to-end deep learning architecture for graph classification. In *Thirty-second AAAI conference on artificial intelligence* (2018).

31. Dhillon, I. S., Guan, Y. & Kulis, B. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis machine intelligence* **29**, 1944–1957 (2007).