NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme 'Data Science and Business Analytics'

UDC _____

Research Project Report

On the topic 'Topological and Geometric Deep Learning'

Fulfilled by the Student:
Chechulin Nikolay Dmitrievich, Signature: _____
Group DSBA-201

Checked by the Project Supervisor
Kachan Oleg Nikolaevich,
Research assistant at International Laboratory for Applied Topology and Applications
June 1, 2022
Grade: _____ Signature: _____

Moscow, 2022

Abstract

Nowadays, a need to analyze more complex data arises. Some objects and relations can not be represented as vectors in Euclidean space, and, therefore, we have to consider graphs — sets of nodes and connections between them — as a subject of analysis. They are able to hold more information than traditional Euclidean object — for example, nodes and edges may have their own features. It is also possible to extract topological information from a graph — by determining its 'shape' we can understand the relation between nodes and somehow make use of this information.

This poses a huge problem: we have to invent new algorithms, adapt known techniques and constantly improve them in order to work with such a complex data.

Our goal is to research the efficiency of several tweaks of existing models and means of preprocessing graphs in order to increase several key indicators.

We will consider several topological methods of data preprocessing. In particular, we will see and benchmark how merging simplexes of higher dimensions (in our case the third one) allow us to greatly reduce learning time while preserving the accuracy of a Graph Neural Network approximately at the same level. This is especially important for production use of such neural networks, since these systems often demand reducing the response time and server load.

Keywords: Graph Neural Network, Graph Convolutional Network, Graph Attention Network, Simplicial Complex, Clique

Contents

1 Definitions

Graph

Graph is a tuple (V, E), where V is the set of all nodes v, and E is the set of edges $e_i = (v_j, v_k)$.

Graphs can be undirected ($(v_j, v_k) \equiv (v_k, v_j)$) and directed, where presence of the edge (v_j, v_k) does not imply that edge (v_j, v_j) exists. Edges can also have weights which show some information about the tightness of connection of two nodes. In this case $e_i = (v_j, v_k, w_i)$.

For instance, a road map of a country is an undirected weighted graph, where cities are nodes, roads are edges, and distances are weights.

Clique

A graph clique is a subset of its nodes such that it is fully connected. So, n-clique is a set of n nodes and $\frac{n \cdot (n-1)}{2}$ edges [1].

We will work mainly with 3-cliques, and often will refer to them as triangles.

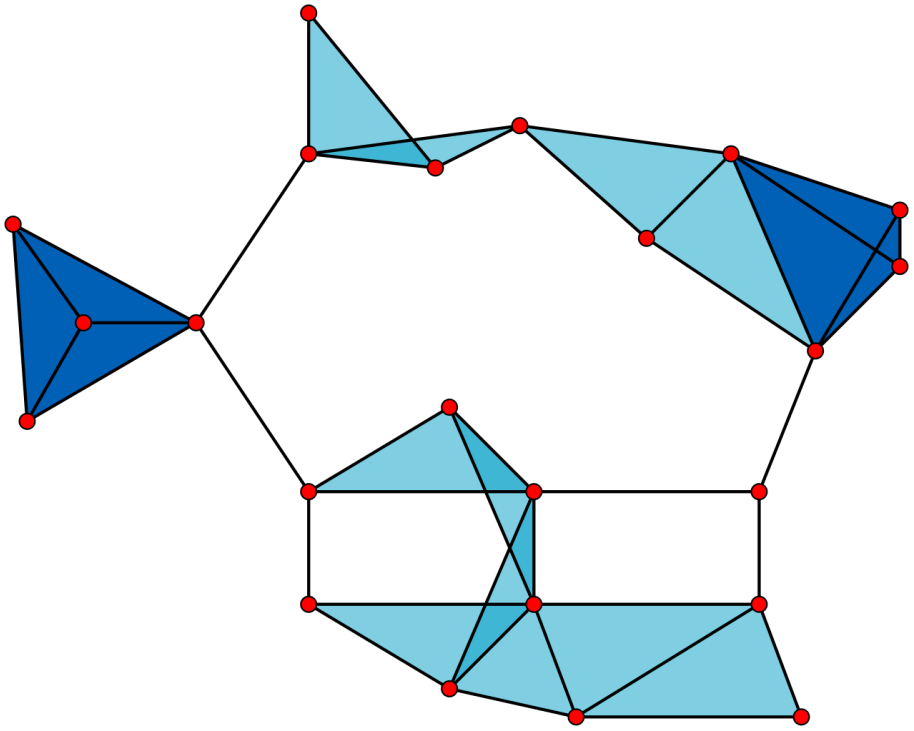


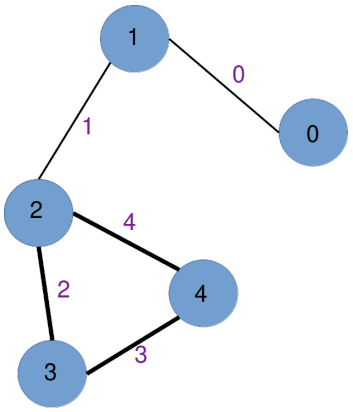
Figure 1: A graph with 23 1-vertex cliques (the vertices), 42 2-vertex cliques (the edges), 19 3-vertex cliques (light and dark blue triangles), and 2 4-vertex cliques (dark blue areas).

Adjacency Matrix

$|V| \times |V|$ adjacency matrix A is defined as follows: $A_{i,j} = 1$ if there is an edge from v_i to v_j. In our project we will consider undirected graphs with self-loops. This means that $\forall i, j\ A_{i,j} = A_{j,i}$ and $\forall i\ A_{i,i} = 1$.

Incidence Matrix

If we have an undirected graph (V, E), its incidence matrix ∇ of size $|V| \times |E|$ such that $A_{i,j} = 1$ if i-th vertex is a vertex of j-th edge. In directed case, we mark the initial vertex as -1, and the terminal as 1 [3].



The incidence matrix of the graph on the left would be as follows:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Degree Matrix

$|V| \times |V|$ diagonal degree matrix D is defined as follows: $D_{i,i} = \Sigma_j A_{i,j}$ (that is, $D_{i,i} = in\_degree(v_i) + out\_degree(v_i)$)

Laplacian matrix

Laplacian matrix — A matrix representation of a graph. Usually is calculated using the following formula [5]:

$$L_{i,j} = \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise,} \end{cases}$$

However, other definitions also take place: $L = D - A$, where D is a degree matrix and A is an adjacency matrix. Another way to calculate a Laplacian is $L = \nabla\nabla^T$, where ∇ is an incidence matrix.

Convolutional Graph Network

CGN *(Convolutional Graph Network)* — A type of GNN which generalizes the convolution operation to graphs. Often we encounter convolution while we work with grid-structured data like images, but here we use same idea (aggregate features of the neighbors) on nodes instead of pixels [11] [4].

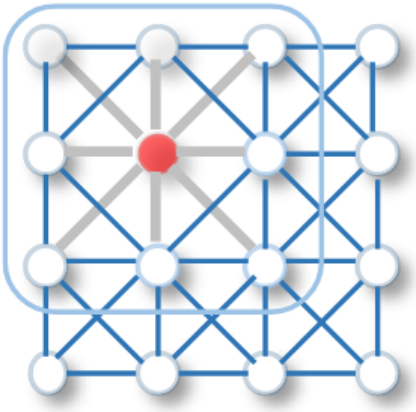


Figure 2: Convolution on image

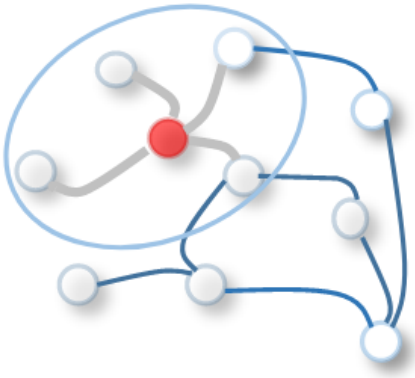


Figure 3: Convolution on graph

Assume we have a graph of N nodes, where each node has F features. We can construct an $N \times F$ matrix called feature matrix. The first layer takes the feature matrix, and performs the following operation: $Z = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}XW$, where:

- Z is resulting $N \times C$ signal
- D is $N \times N$ degree matrix
- A is $N \times N$ adjacency matrix with self-loops
- X is $N \times F$ feature matrix (input signal)
- W is $F \times C$ learnable weight matrix

Then the output Z is directed into next layer, which does practically the same. There might be many convolutional layers, but usually models only have 2.

The last (output) layer usually applies softmax function to each row resulting in a new matrix S. Then, in order to classify a node v_i we simply take the index of maximum of S_i.

The architecture of a graph convolutional network is presented on the figure below [10]

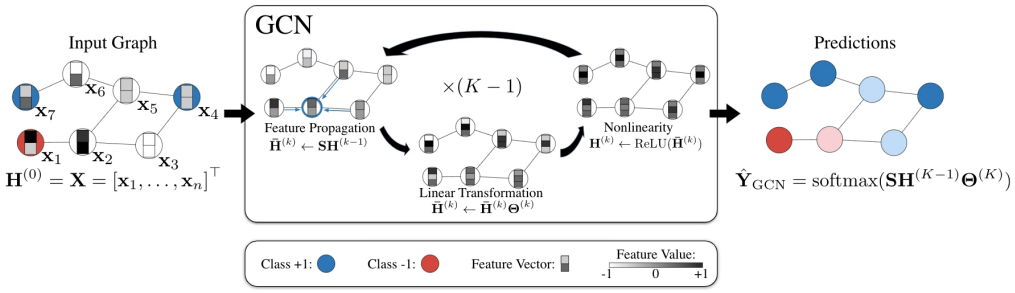


Figure 4: Architecture of a GCN

Graph Attention Network

GAT *(Graph Attention Network)* — A type of GNN which uses attention mechanism (also borrowed from 'casual' neural networks) which allows us to work with inputs of variable sizes and to focus on the most important features [9]. The attention mechanism is a function $a : \mathbb{R}^C \times \mathbb{R}^C \to \mathbb{R}$ which takes two feature vectors X_i, X_j and returns a scalar representing how tight the connection between v_i and v_j is.

We introduce an $N \times N$ matrix e storing the attention between the nodes: $e_{i,j} = a\left(W \cdot X_i,\ W \cdot X_j\right)$. Now we have to be careful about choice of i and j, since if we calculate the attention between all the nodes, we will completely drop structural information of the graph. One suggested solution is to use a neighborhood $\mathcal{N}_i$ of a vertex v_i and then compute $e_{i,j}$ for all $j \in \mathcal{N}_i$. Existing model [9] uses neighborhood of size 1 (that is, v_i itself and all of its neighbors v_j such that $\exists e = (v_i, v_j)$), and it seems to perform great.

One might also want to normalize the coefficients. In order to do that, we can apply softmax function: $c_{i,j} = \mathrm{softmax}_j\left(e_{i,j}\right) = \frac{\exp\left(e_{i,j}\right)}{\Sigma_k \exp\left(e_{i,k}\right)}$.

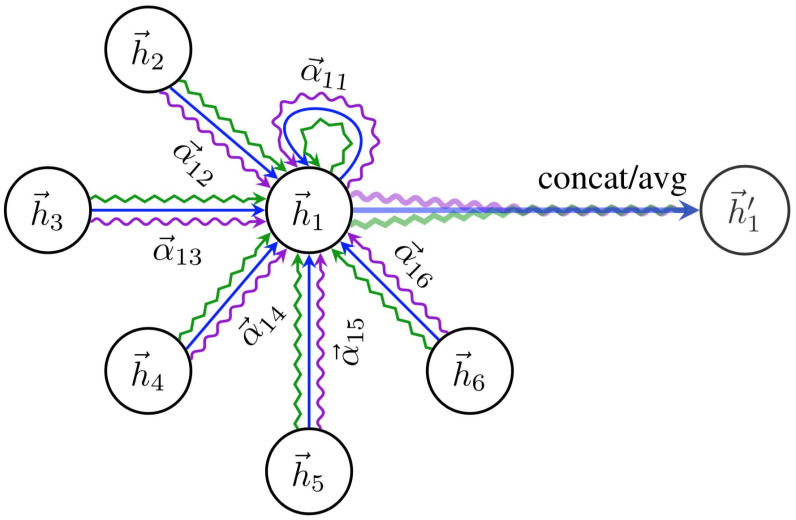


Figure 5: An example of multi-head attention in a neighborhood of size 1

2 Introduction

2.1 Relevance

The field of research (graph neural networks) might be considered relatively new, and, therefore, there is a huge number of possible improvements to be made to existing models and approaches. Our ultimate goal is to improve the accuracy of node and graph classification.

For example, one of the proposed changes is to modify a Laplacian in such a way that it does not break existing model and improves it. Our initial results have shown that our approach indeed works well on Karate club dataset, where we had to classify nodes, as shown on Figure 6.

Despite already impressive accuracies achieved by other researchers, we also want to focus on learning time, since this characteristic is crucial for production use of models. As we will show later, they are used in a wide variety of tasks, some of which require awaiting of response. One of the techniques proposed can speed up learning process and prediction speed, while not dropping accuracy drastically.

The relative novelty of the field allows us to find new experiments and techniques.

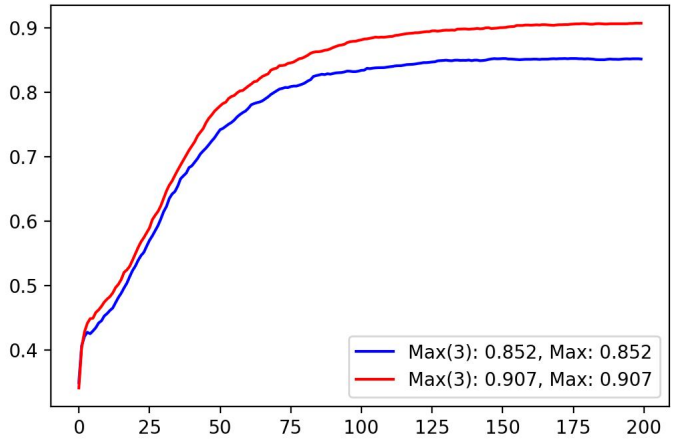


Figure 6: Default Laplacian (in blue) versus our Laplacian (in red). Y-axis is the accuracy, X-axis is the number of epochs

2.2 Subject of research

Let us explain the tasks we can solve using GNNs *Graph Neural Networks* in more detail. There are three of them:

- *Node classification* — given a graph with several labelled nodes and several classes predict a class of an unlabelled node. For example, the CORA [8] contains information about scientific papers. Each paper belongs to exactly one of seven given classes. Our task is to determine the class of a new paper given its feature vector and connections with other papers (in this case the edges of the graph are citations, the edges are undirected).

- *Graph classification* — determine type of graph. A good example of this task is presented classification of molecules [12] or image classification [2], as well as in many other fields.

- *Link prediction* — determine if two given nodes should have an edge between them. A simple example could be friends suggestions in a social network.

In our research we will only consider the first two problems.

As we established, we want to consider several changes in order to improve the accuracy. The tweaks we propose include but are not limited to:

- *Altering the way we compute Laplacian* — a characteristic matrix of a graph.
- *Edge embeddings*
- *Using connectivity over simplices of higher dimension*. This means that in some cases we might want to consider a group of nodes as a separate object, therefore, increasing the connectivity factor. We will only work with 3-simplices:

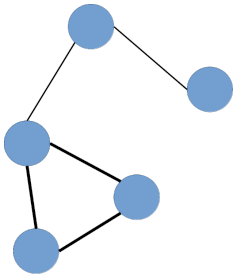


Figure 7: A part of some graph

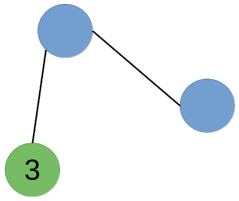


Figure 8: Three nodes from the left image united in 3-simplex having properties of the initial vertices

3 Experiments

3.1 3-Clique merge with insertion

Research method

We will compare the models performance on Planetoid/CORA dataset [8] while classifying nodes of the graph. The dataset consists of exactly one undirected graph. The graph represents a citation network — nodes are scientific papers, and edges are citations (if two nodes are connected then one paper cites the other).

There are 2708 nodes and 10556 edges, and each node has a feature vector of length 1433 which stores the number of occurrences of 1433 predefined words. Each node (paper) is a member of exactly one of seven classes.

The training node label rate is 5%, which means that the neural network knows classes of 5% of nodes while learning.

The benchmark will be performed by taking average of 100 runs of 300 epochs.

Description

In this experiment we find all the 3-cliques [1] (triangles) in the graph, then sort them based on 'distance' and replace each 3-clique with one node (called *merged node*) sharing the features of the initial three.

The following figures show exactly how a 3-clique is merged: the resulting feature vector is the average of the feature vectors of initial nodes: $\frac{1}{3} \cdot ((1,4,5) + (2,7,1) + (0,1,6)) = \frac{1}{3} \cdot (3,12,12) = (1,4,4)$

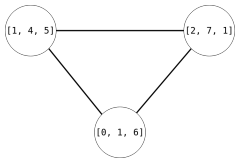


Figure 9: An example of a 3-clique

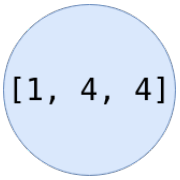


Figure 10: Same 3-clique but merged

The source code is available in the project repository and can be found in the 'experiments' directory [7].

Algorithm

Firstly, let us define a 3-clique more formally. We define it as follows: $\{(a,b,c) \mid a,b,c \in V \wedge (a,b), (a,c), (b,c) \in E\}$.

While merging nodes in the cliques, we might face a problem: some of the nodes might participate in several 3-cliques. In order to solve it, we need to introduce a mechanism for ordering cliques so that we could show preference to one instead of the other. Naturally, such a mechanism should

rely on the nodes feature vectors. There are 2 major distance calculating algorithms: Euclidean distance and Manhattan distance. Since the feature vectors are very sparse, the algorithms give quite close results and there is no need to consider them separately.

Using Euclidean distance we can map each 3-clique into a real number by calculating the sum of pairwise distances: $|a_f - b_f|^2 + |a_f - c_f|^2 + |b_f - c_f|^2$, where v_f stands for node's v feature vector. The resulting ordering now allows us to choose one 3-clique instead of another, since we know that one has its nodes 'closer' to each other.

Therefore, one can simply sort all the 3-cliques in ascending order, merge each 3-clique and filter out all the 'later' 3-cliques containing nodes from the current one. This allows us pick the closest and most valuable nodes in a greedy way while also saving us from using a 'new' node in another 3-clique ($(a, b, c) \rightarrow a'$; $(a', d, e) \rightarrow a''$ is not allowed, since the graph could collapse).

1. Find all the 3-cliques in graph.
2. Sort them according to the sum of distances between nodes
3. Merge all 'valid' 3-cliques in one node. While doing so, we have to preserve all the edges coming in and out of the triplet. For each clique all the other cliques containing nodes from the current one have to be filtered out, so the 'merged' nodes will are not merged with others.
4. Generate new dataset from the resulting graph.
5. Pass the generated dataset to model.

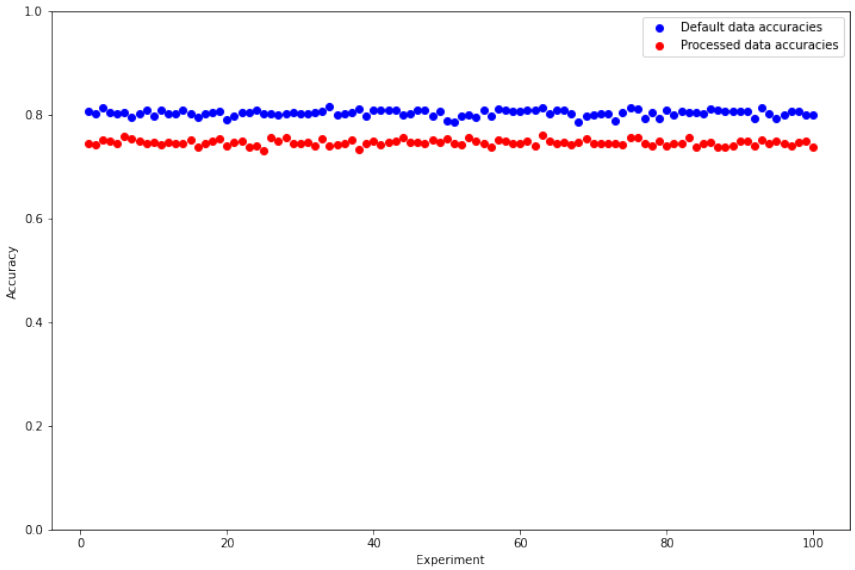
The process is shown on Figure 7.

Results and interpretation

A benchmark on the Cora dataset [8] had shown that we are able to remove a total of 21.71% of nodes (the number reduced from 2708 to 2120) and 56.73% edges (from 10556 to 4568). Since the operation of merging of all

cliques fast comparing to the model's learning time, we reduced the latter approximately by 29.5% (on average, went from 4.171s to 2.939s). This improvement can be also seen on the loss graph of our models:

Speaking of accuracy, we claim it did not change much. On average, the accuracy of a default model is 0.803, and the model with 3-cliques merged performs practically same — the average accuracy is 0.746 (both results were achieved by taking average of 100 runs with 300 epochs). So, the accuracy dropped by 7% (0.057 in absolute value), while the speed increase was almost 30%.



Therefore, we can claim that the only purpose of merging 3-cliques is reducing the learning time without significant loss of accuracy. It doesn't make much sense for theoretical purposes, however, is definitely an advantage in production.

3.2 Infinite 3-clique merge

Description

In this experiment, the methodology and the algorithm are similar to the 'finite 3-clique merge'. The only difference is that we *allow* merged nodes to

participate in merge process again. The source code is available on GitHub as well [6].

Results and interpretation

The proposed method allowed us to remove the majority of information from a graph. This merge, or convolution, managed to delete as many as 60.12% of the nodes and 93.79% of edges. The performance of learning after this operation increased drastically — the learning time went from 2.537 seconds to as low as 1.059 seconds (which is only 42% of the original time).

We can clearly see the improvement on the loss graph below. We estimate the greatest improvement to be at around 160 epochs, not 300 as we benchmarked.

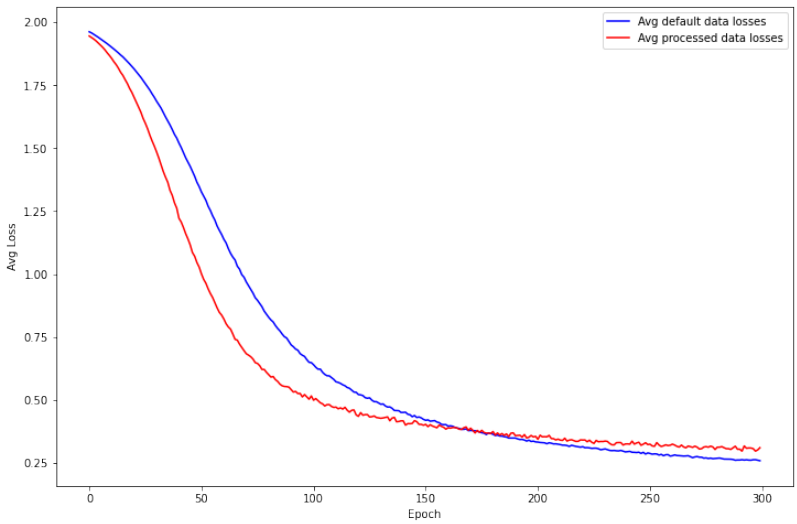


Figure 11: Average loss for ‘default’ and processed data

However, this comes at a high cost of decreased accuracy. Average accuracy went from 0.809 to 0.517 which is a 36% decrease.

This extreme graph reduction is probably applicable in some cases, but we think that it would be better to find a balance in the number of number of merges — heuristically find the number of iterations of algorithm described in Section 3.1.

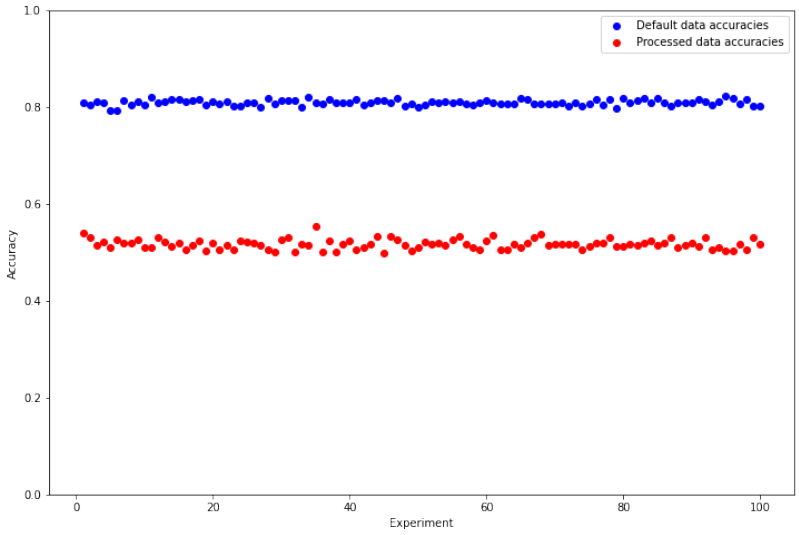


Figure 12: Accuracies of models with ‘default’ and processed data

3.3 Prospects

In future we want to focus on topological aspect of the transformation presented in the paper. In particular, we are interested in ‘infinite merge’, where ‘merged nodes’ can be merged again. We also want to see how the transformation presented can be used in other graph tasks, for example, in graph classification.

Another interesting idea is to exploit other topological feature of a graph. Sometimes nodes can be weakly connected to each other via edges, while probably having another mean of connections. For instance, if we consider a map with several coastal cities, we will see that the cities are ‘chained’ (each of them has connections with 2 neighbors) and form some kind of a loop. We think it is possible to extract information from this form of connection.

References

[1] "Clique". In: *Wikipedia* (). URL: https://en.wikipedia.org/wiki/Clique_(graph_theory).

[2] Vijay Prakash Dwivedi et al. "Benchmarking Graph Neural Networks". In: *CoRR* abs/2003.00982 (2020). arXiv: 2003.00982. URL: https://arxiv.org/abs/2003.00982.

[3] "Incidence matrix". In: *Wikipedia* (). URL: https://en.wikipedia.org/wiki/Incidence_matrix.

[4] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *arXiv:1609.02907 [cs, stat]* (Feb. 2017). arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[5] "Laplacian matrix". In: *Wikipedia* (). URL: https://en.wikipedia.org/wiki/Laplacian_matrix.

[6] Nikolay Chechulin. *3-clique infinite merge with insertion experiment*. 2022. URL: https://github.com/NChechulin/GNN-project/blob/master/experiments/3clique/3clique_infinite_merge.ipynb.

[7] Nikolay Chechulin. *3-clique merge with insertion experiment*. 2022. URL: https://github.com/NChechulin/GNN-project/blob/master/experiments/3clique/3clique_merge.ipynb.

[8] *The CORA dataset*. URL: https://graphsandnetworks.com/the-cora-dataset/.

[9] Petar Veličković et al. "Graph Attention Networks". In: *arXiv:1710.10903 [cs, stat]* (Feb. 2018). arXiv: 1710.10903. URL: http://arxiv.org/abs/1710.10903.

[10] Felix Wu et al. "Simplifying Graph Convolutional Networks". In: *arXiv:1902.07153 [cs, stat]* (June 2019). arXiv: 1902.07153. URL: http://arxiv.org/abs/1902.07153.

[11] Zonghan Wu et al. “A Comprehensive Survey on Graph Neural Networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (Jan. 2021), pp. 4–24. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2020.2978386.

[12] Keyulu Xu et al. “How Powerful are Graph Neural Networks?” In: *CoRR* abs/1810.00826 (2018). arXiv: 1810.00826. URL: http://arxiv.org/abs/1810.00826.