Topological and geometric deep learning

Student: **Nikolay Chechulin**, DSBA-201
Academic Supervisor: **Oleg Kachan**

NATIONAL RESEARCH
UNIVERSITY
Faculty of Computer Science
Bachelor's programme 'Data Science and Business Analytics'
Moscow, Russia
2022

Contents

1 Introduction

1.1 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. 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.

1.2 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:

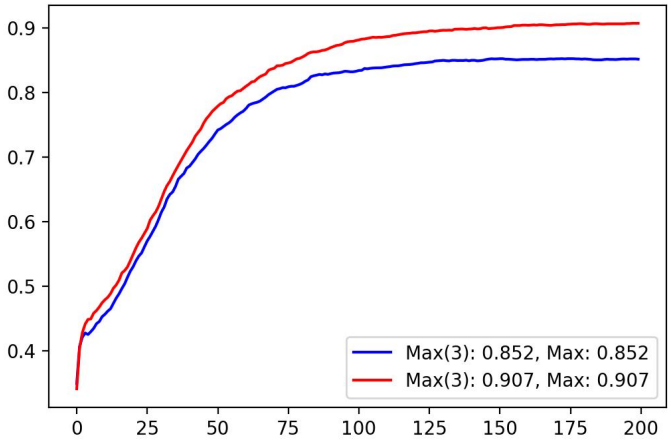


Figure 1: Default Laplacian (in blue) versus our Laplacian (in red). Y-axis is the accuracy, X-axis is the number of epochs

1.3 Subject of research

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

- *Node classification* — given a graph with several labeled nodes and several classes predict a class of an unlabeled node. For example, given a set of scientific papers and their citations, determine the type of new paper.

- *Graph classification* — determine type of graph. For instance, check if a molecule affects certain parameters of an organism.
- *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:

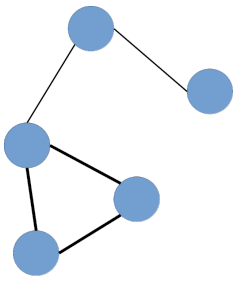


Figure 2: A part of some graph

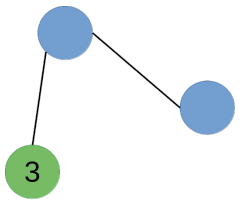


Figure 3: Three nodes from the left image united in 3-simplex having properties of the initial vertices

2 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 [2].

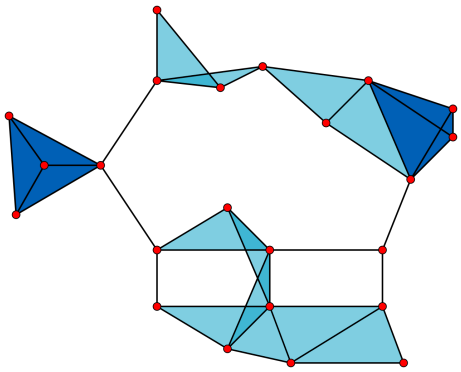


Figure 4: A graph with 23 1-vertex cliques (the vertices), 42 2-vertex cliques (the edges), 19 3-vertex cliques (light and dark blue triangles), and 2 4-vertex cliques (dark blue areas).

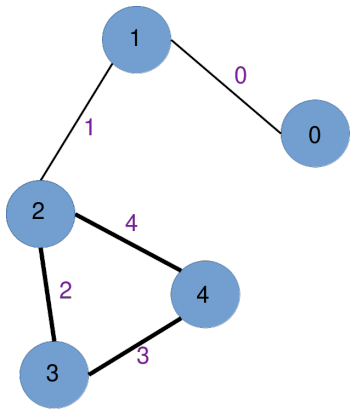
Adjacency Matrix

$|V| \times |V|$ 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 [8] [4].

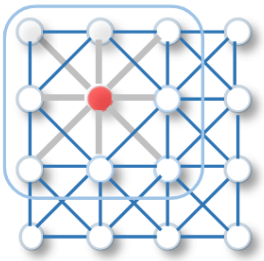


Figure 5: Convolution on image

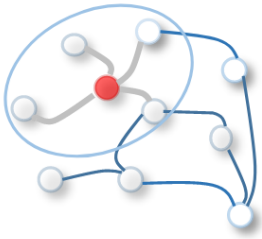


Figure 6: Convolution on graph

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: $Z = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} XW$, 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.

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 [7]. 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 [7] 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: $c_{i,j} = \text{softmax}_j\left(e_{i,j}\right) = \frac{\exp\left(e_{i,j}\right)}{\Sigma_k \exp\left(e_{i,k}\right)}$.

3 Experiments

3.1 3-Clique merge

Description

In this experiment we find all the 3-cliqes [2] (triangles) in the graph, then sort them based on 'distance' and replace each 3-clique with one node sharing the features of the initial three.

The source code is available in the project repository and can be found in the 'experiments' directory [1].

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.

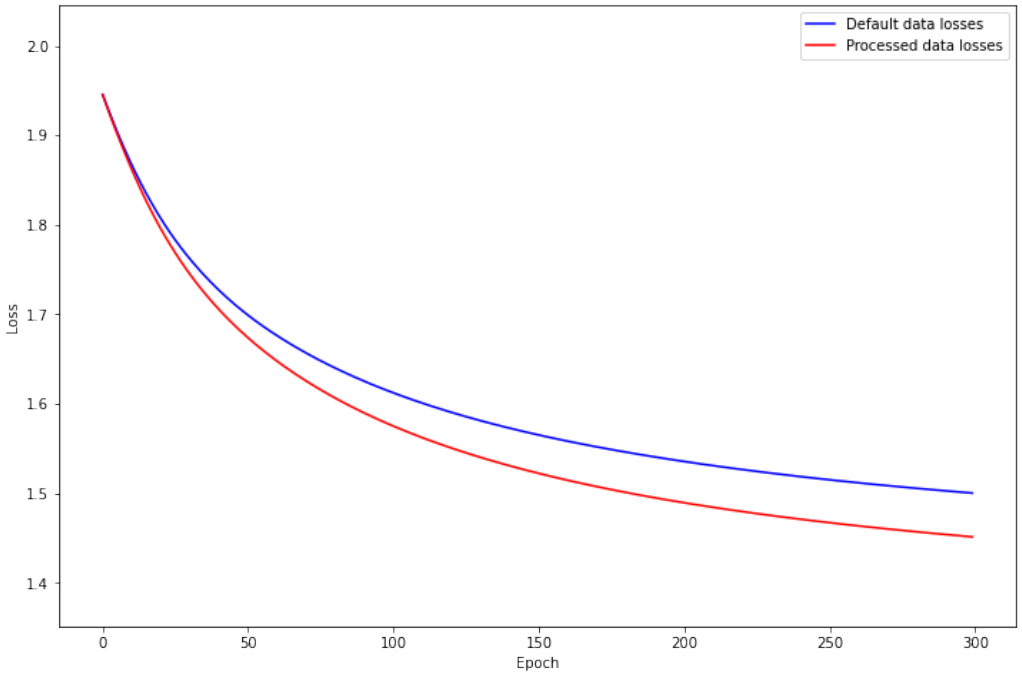
Therefore, one can simply sort all the 3-cliques in ascending order, merge each 3-clique and filter out all the 'later' 3-cliques containing nodes from the current one. This allows us pick the closest and most valuable nodes in a greedy way while also saving us from using a 'new' node in another 3-clique ($(a, b, c) \to a'$; $(a', d, e) \to a''$ 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 preserving all the edges coming in and out of the triplet filtering out all the cliques containing nodes from the merged ones
4. Generate new dataset from the resulting graph

Results and interpretation of them

A benchmark on the Cora dataset [6] had shown that we are able to remove a total of 12.2% of nodes (the number reduced from 2708 to 2378) and 42.3% edges (from 10556 to 6094). Since the operation of merging of all cliques fast comparing to the model's learning time, we reduced the latter

approximately by 23%. 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 lies somewhere between 0.7 and 0.75 and the model with 3-cliques merged performs practically same with an error of ±0.005 (for example, 0.725 vs 0.72402).

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

References

[1] *3-clique merge experiment*. URL: `https://github.com/NChechulin/GNN-project/blob/master/experiments/3clique/3clique_merge.ipynb`.

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

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

[7] 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.

[8] 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.