GRAPH-BERT: Only Attention is Needed for Learning Graph Representations (22-2-R-5)

Dr. Renata Avros
Prof. Zeev Volkovich

Yaniv Sokolov - Yaniv.sokolov@e.braude.ac.il
Shay Maryuma – Shay.Maryuma@e.braude.ac.il

January 2022

Contents

1 Introduction

Publishing an article in a specific journal requires an abundance of quotes from the collection of articles that belongs to the journal. To measure the number of quotes, the journal uses Journal Impact Factor (JIF) to measure the frequency with which the average article in a journal has been cited in a particular year. It is used to measure the importance or rank of a journal by calculating the times its articles are cited. Hence, journalists inflate the paper with unnecessary citations that do not donate to the article, this condition is called Coercive Citation. Coercive Citation is when an author submits a manuscript for publication in a scientific journal, the editor may request that the article's citations be expanded before it will be published. This is a part of the standard peer-review process and is meant to improve the paper. On the other hand, the coercive citation is a specific unethical business practice in which the editor asks the author to add citations to papers published in the same journal (self-citation) and in particular to cite papers that the author regards as duplicate or irrelevant. This project is designed for research authority to avoid a coercive citation. Today, to indicate the problem in a specific paper, an article is supervised manually which leads to struggles:

- Millions of articles exist, which cost a lot of time and money for the journal.
- Manual testing could lead to human error.

The aim of the project is to create a transformer neural network with GRAPH-BERT (Bidirectional Encoder Representations from Transformers) that can decide if a quote is relevant to a specific article with a quick fine-tuning, each user (a journal or an article writer) could easily use the project on his personal computer. The GRAPH-BERT is a part of the GNN (Graph Neural Network) Model, using traditional deep learning with a transformer-based encoder. In the following sections, we review the background of the problem and the solution. We describe the method by which we solved the problem and the method we propose.

2 Background and Related Work

2.1 Background

2.1.1 Graph

A graph provides a unified representation of many interconnected data in the real world, which can model both the diverse attribute information of the node entities and the extensive connections among these nodes. For instance, the relation between words, online social media, and biological molecules.

2.1.2 Word2Vec

Word2vec is a technique for natural language processing (NLP). The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector. The vectors are chosen carefully such that a simple mathematical function indicates the level of semantic similarity between the words represented by those vectors. The word2vec

algorithm has two different architectures, both are learning the underlying word representations for each word by using neural networks. In the Continuous Bag of Words Model (CBOW) model, the distributed representations of context (or surrounding words) are combined to predict the word in the middle. While in the Skip-gram model, the distributed representation of the input word is used to predict the context [1].

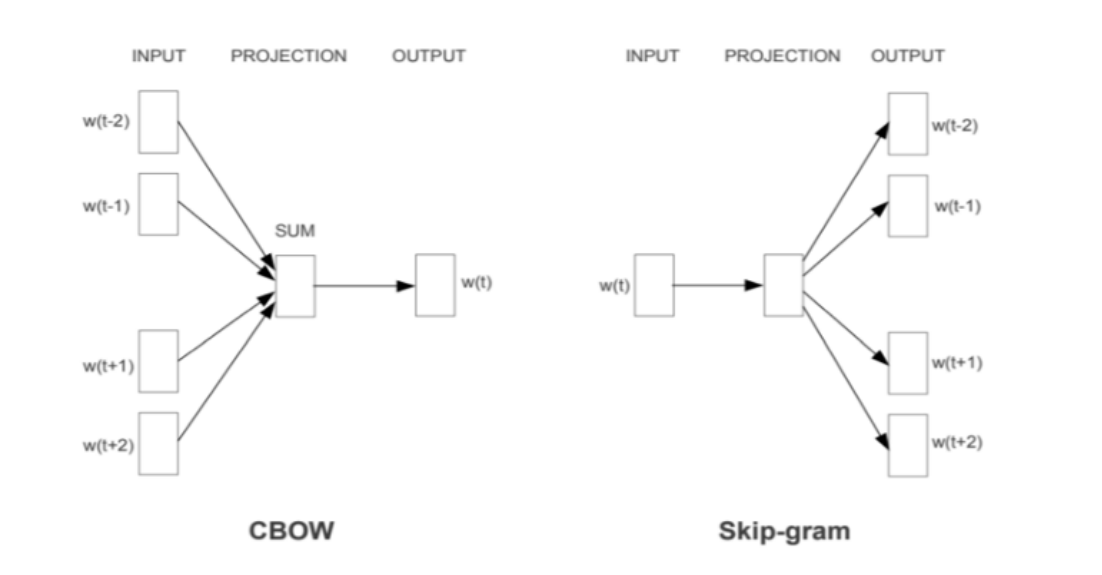

Figure 1: Word2Vec CBOW vs Skip-gram

2.1.3 Context2Vec

Context2vec is an unsupervised model for learning generic context embedding of wide sentential contexts, using a bidirectional LSTM. A large plain text corpora is trained on to learn a neural model that embeds entire sentential contexts and target words in the same low-dimensional space, which is optimized to reflect inter-dependencies between targets and their entire sentential context. In contrast to word2vec that use context modeling mostly internally and considers the target word embeddings as their main output, the focus of context2vec is the context representation. context2vec achieves its objective by assigning similar embeddings to sentential contexts and their associated target words [2].

2.1.4 Neural Network (NN)

Neural networks reflect the behavior of the human brain, allowing computer programs to recognize patterns and solve common problems in the fields of AI, machine learning, and deep learning. Artificial neural networks (ANNs) are comprised of a node layer, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of

the network. Otherwise, no data is passed along to the next layer of the network. Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to manual identification by human experts. One of the most well-known neural networks is Google's search algorithm [3].

2.1.5 Transformer

The architecture of the Transformer system follows the so-called encoder-decoder paradigm, trained in an end-to-end fashion. Without using any recurrent layer, the model takes advantage of the positional embedding as a mechanism to encode order within a sentence. The encoder typically stacks 6 identical layers, in which each of them makes use of the so-called multi-head attention and of a 2 sublayers feed-forward network, coupled with layer normalization and residual connection. The multi-head attention mechanism computes attention weights, namely a softmax distribution, for each word within a sentence, including the word itself [4].

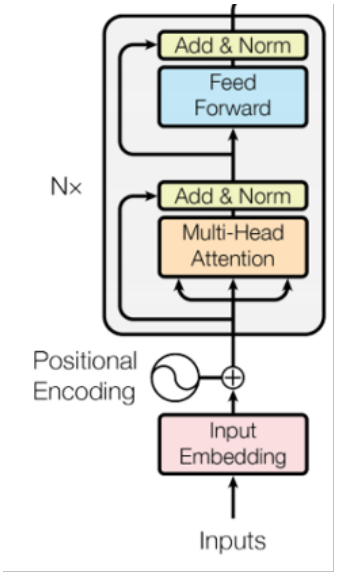

Figure 2: Transformer based encoder

2.1.6 Expectation Maximization (EM)

The Expectation-Maximization (EM) algorithm is a way to find maximum-likelihood estimates for model parameters when your data is incomplete, has missing data points, or has unobserved latent variables. It is an iterative way to approximate the maximum likelihood function. While maximum likelihood estimation can find the "best fit" model for a set of data, it does not work particularly well for incomplete data sets. The more complex EM algorithm can find model parameters even if you have missing data. It works by choosing random values for the missing

data points and using those guesses to estimate a second set of data. The new values are used to create a better guess for the first set, and the process continues until the algorithm converges on a fixed point.

2.1.7 Graph Neural Network (GNN)

Graph Neural Networks (GNNs) are an effective framework for representation learning of graphs. GNNs follow a neighborhood aggregation scheme, where the representation vector of a node is computed by recursively aggregating and transforming the representation vectors of its neighboring nodes Graph Neural Networks (GNNs) are a class of deep learning methods designed to perform inference on data described by graphs. GNNs are neural networks that can be directly applied to graphs and provide an easy way to do node-level, edge-level, and graph-level prediction tasks. GNNs can do what Convolutional Neural Networks (CNNs) failed to do. In NLP, the text is a type of sequential data that can be described by an RNN or an LSTM. However, graphs are heavily used in various NLP tasks, due to their naturalness and ease of representation. Recently, there has been a surge of interest in applying GNNs for many NLP problems like text classification, exploiting semantics in machine translation, user geolocation, relation extraction, or question answering. Every node is an entity, edges describe relations between them. In NLP research, the problem of question answering is not recent. But it was limited by the existing database [5].

2.1.8 Graph Convolutional Network (GCN)

Graph Convolutional Networks (GCNs) and their variants have experienced significant attention and have become the de facto methods for learning graph representations. GCNs derive inspiration primarily from recent deep learning approaches, and as a result, may inherit unnecessary complexity and redundant computation. The simplest GCN has only three different operators:

- Graph convolution
- Linear layer
- Non-linear activation.

The operations are usually done in this order. Together, they make up one network layer. One or more layers can be combined to form a complete GCN. After a test that has been conducted, there is no project that explores this kind of subject [6].

2.2 Related Work

2.2.1 Graph Neural Network

Representative examples of GNNs proposed by present include GCN, GraphSAGE and LOOPY-NET, based on which various extended models have been introduced as well. GCN and its variant models are all based on the approximated graph convolutional operator which may lead to the suspended animation problem and over-smoothing problem for deep model architectures. To handle such problems, generalizes the graph raw residual terms in and proposes a method based on graph residual learning. proposes to adopt dense connections and dilated convolutions into the GCN architecture [7].

2.2.2 BERT and TRANSFORMER

The leading sequence transduction models in Natural Language Processing (NLP) are complicated recurrent or convolutional neural networks. However, because to the fundamentally sequential structure of training samples, parallelization is not possible. As a result, the authors suggest the TRANSFORMER, a new network architecture based exclusively on attention processes, with no repetition or convolutions. BERT for deep language understanding is also introduced with TRANSFORMER, which achieves new state-of-the-art outcomes on eleven natural language processing tasks. TRANSFORMER and BERT-based learning techniques have been widely applied in many learning tasks in recent years [7].

3 Project Algorithm

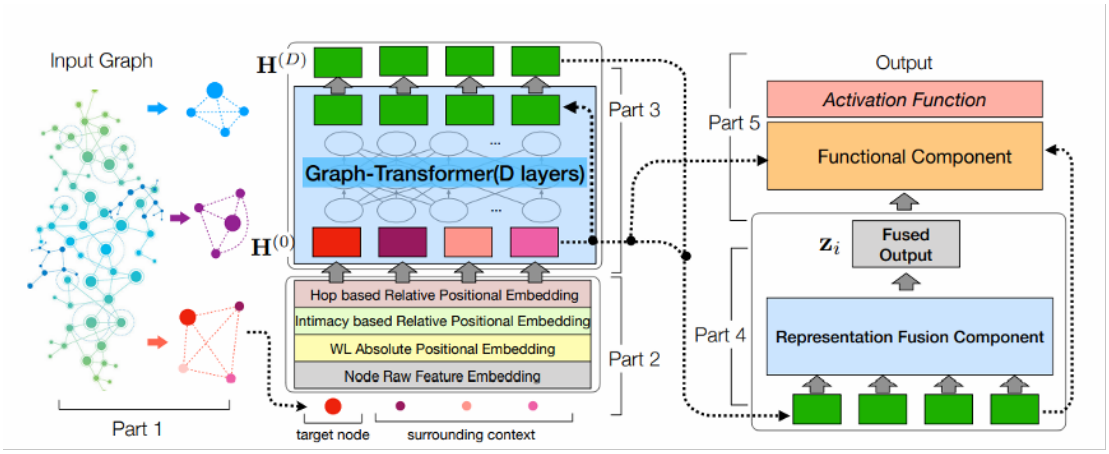

Figure 3: Graph BERT Scheme

The input to the Graph-BERT algorithm is a Linkless Subgraph.
Linkless Subgraph Batching
Intimacy matrix S denoted as:

$$S = a(I - (1-a)\bar{A})^{-1}$$

In this project, the value a=0.15.

$$\bar{A} = AD^{-1}.$$

A - Adjacency matrix of the input graph.
D - Corresponding diagonal matrix:

$$D(i,i) = \Sigma_j A(i,j)$$

.

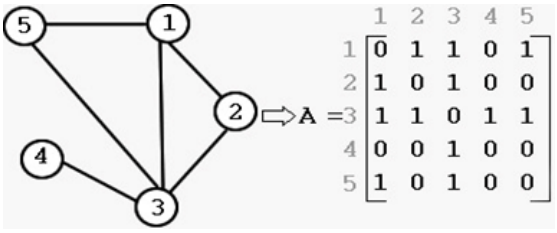

Figure 4: Adjacency matrix of the input graph.

To determine the intimacy score between two nodes v_i and v_j we need to look at the intimacy matrix in the place - S (i, j).

For every node we need to define the number of related K nodes that will be in the subgraph, it is determined by the intimacy matrix. A minimum intimacy score threshold is defined for the nodes to involve in the subgraph [7].

Node input vector embedding
The GRAPH-BERT model needs to serialize the input subgraph nodes into order list in the algorithm in order to be learned, all the nodes in the sampled Linkless subgraph can be denote as a node list $[v_i, v_{(i,1)}, \ldots, v_{(i,m)}]$, where $v_{(i,j)}$will be placed ahead of $v_{(i,m)}\, if\, S(i,j) > S(i,m)$.
Raw Feature Vector Embedding
Convert a node to a vector by using a relevant attribute. Depending on the input raw features properties, different models can be used to define the embedded function, in the project the node is already embedded vector by algorithm context2vec.

$$e_j^x = x_j \in R_h^{d*1}, x_j \in nodes$$

d_h - Dimension of the embedded vector
In the original algorithm a node must be injected to three different functions, in our version of the algorithm, instead of using the functions, we take three constant vectors, these three vectors will be marked as

$$e_j^{((r))}, e_j^{((p))}, e_j^{((d))}.$$

Graph Transformer based Encoder
Based on the computed embedding vectors defined above, aggregation is needed to define the initial input vectors for nodes in the subgraph g_i as follows:

$$h_j^{(0)} = Aggregate(e_j^x, e_j^r, e_j^p, e_j^d)$$

The aggregate function defined as vector summation. The initial input vector for all the nodes in the subgraph can be organized into a matrix.

$$H^{(0)} = [h_i^{(0)}, h_{i,1}^{(0)}, \ldots, h_{i,k}^{(0)}]$$

k is the number of nodes in the subgraph. The graph-transformer based encoder updates the nodes representations iteratively with multiple layers (D layers), and the output of each layer can be denoted as:

$$H^{(l)} = G - Transformer(H^{(l-1)})$$

$$= softmax((QK^T)/\sqrt{(d_h)})V$$

$$Q = H^{(l-1)}W_Q^{(l)}$$

$$K = H^{(l-1)}W_K^{(l)}$$

$$V = H^{(l-1)}W_V^{(l)}$$

$$W_Q^{(l)}, W_K^{(l)}, W_V^{(l)} \in R^{d_h d_h}$$

The softmax function idea is to normalize the matrix so that the sum of all the values in the matrix equals to one:

$$(QK^T)/\sqrt{d_h} = M => softmax(M) => M(i,j) = e^{M(i,j)}/(\Sigma_(p,t)e^{M(p,t)})$$

$X_i \in R^{(k+1)d_x}$ - The raw features of all the nodes in the subgraph. Based on the graph-transformer function, the output of the transformer is:
$z_i = Fusion(H^{(D)})$, D is the number of the layers in the transformer. In the algorithm we interested to get the representations of the target node only. Function Fusion averages the representations of the matrix columns which defines the final state of the target node [7].

Pre-training
The GRAPH-BERT can be pre-trained with two tasks:
(1) node attribute reconstruction, and
(2) graph structure recovery.
In our project the relevant pre-train is task (2) - graph structure recovery.

Graph Structure Recovery
To ensure such representation vectors the graph structure information can also be captured, the graph structure recovery task is also used as a pre-training task. Formally, for any two nodes v_i and v_j , based on their learned representations, the inferred connection score between the nodes can be denoted by computing their similarity $\hat{S}_{(i,j)} = (z_i^T z_j)/|z_i||z_j|$ compared against the ground truth graph intimacy matrix, the introduced loss term can be denoted as follows:

$$l = 1/|V|^2 |S - \hat{S}|^2$$

$$\hat{S} \in R^{|V||V|} \; with \; \hat{S}(i,j) = \hat{S}_{(i,j)}.$$

Model Transfer and Fine-tuning
In applying the learned GRAPH-BERT into new learning tasks, the learned graph representations can be either fed into the new tasks directly or with necessary adjustment, namely, finetuning. This project uses graph clustering, that learned representations directly [7].

Graph Clustering
For the graph clustering task, the main objective is to partition nodes in the graph into several different clusters, for example, $C = C_1, \ldots, C_l$, l is a predetermined parameter. For each objective cluster, we can denote its center as a variable vector $\mu_j = \Sigma_{(v_i \in C_j)} z_i$. For the graph clustering tasks, the main objective is to group similar nodes into the same cluster, whereas the different nodes will be partitioned into different clusters instead. Therefore, the objective function of graph clustering can be defined as follows:

$$\min_{(\mu_1,,\mu_l)} \min_{(C)} \Sigma_{j=1}^{l} \Sigma_{v_i C_j} |z_i - \mu_j|^2$$

The above objective function involves multiple variables to be learned concurrently, which can be trained with the EM (Expectation Maximization) algorithm much more effectively instead of error backpropagation. Therefore, instead of re-training the above graph clustering model together with the GRAPH-BERT, the learned node representations will be taken as the node feature input for learning the graph clustering model instead.

The mentioned algorithm eventually needs to be wrapped by using Python script. The dataset consists of a node vector that is composed of articles. When an article quotes another article, it is represented in the graph as a link between the two nodes. At the training stage, a dataset is loaded and injected into GRAPH-BERT's training. For the fine-tuning stage, the identified nodes need to be clustered. After the pre-training and fine-tuning stages, randomly k precent of the connections are omitted in the graph. The GRAPH-BERT algorithm is applied to aim to recover the omitted connections. Afterward, an evaluation of the restored connections needs to be done, the steps are repeated M times, and the recovery proportion is calculated and kept for each one of the edges. In the end, a graph of relevant data will be presented, that will tell if a particular article should insert a citation or not.

3.1 Flow Diagram

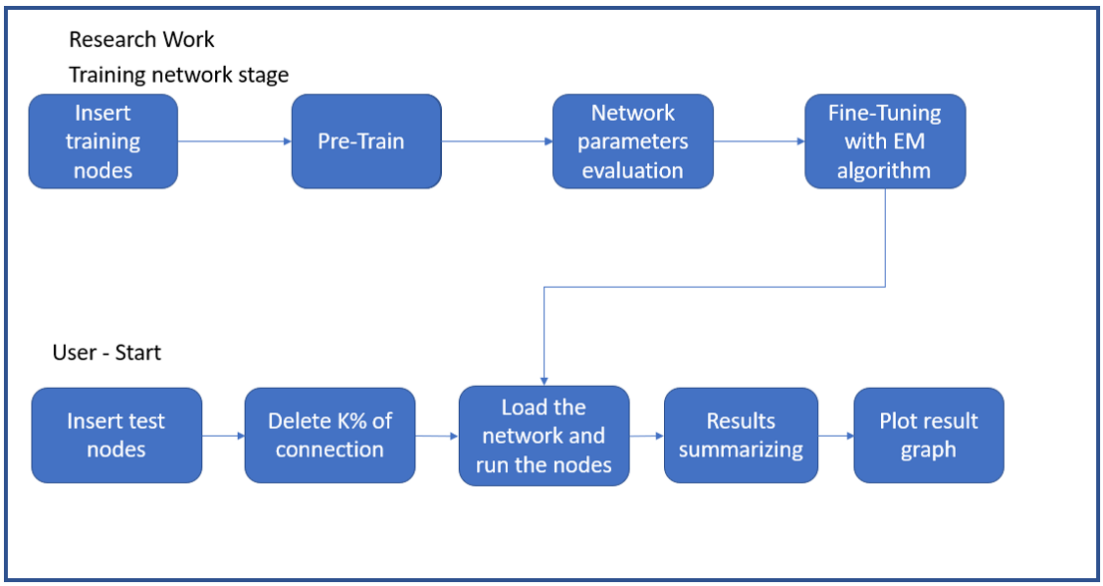

Figure 5: Flow Diagram

3.2 Use-Case

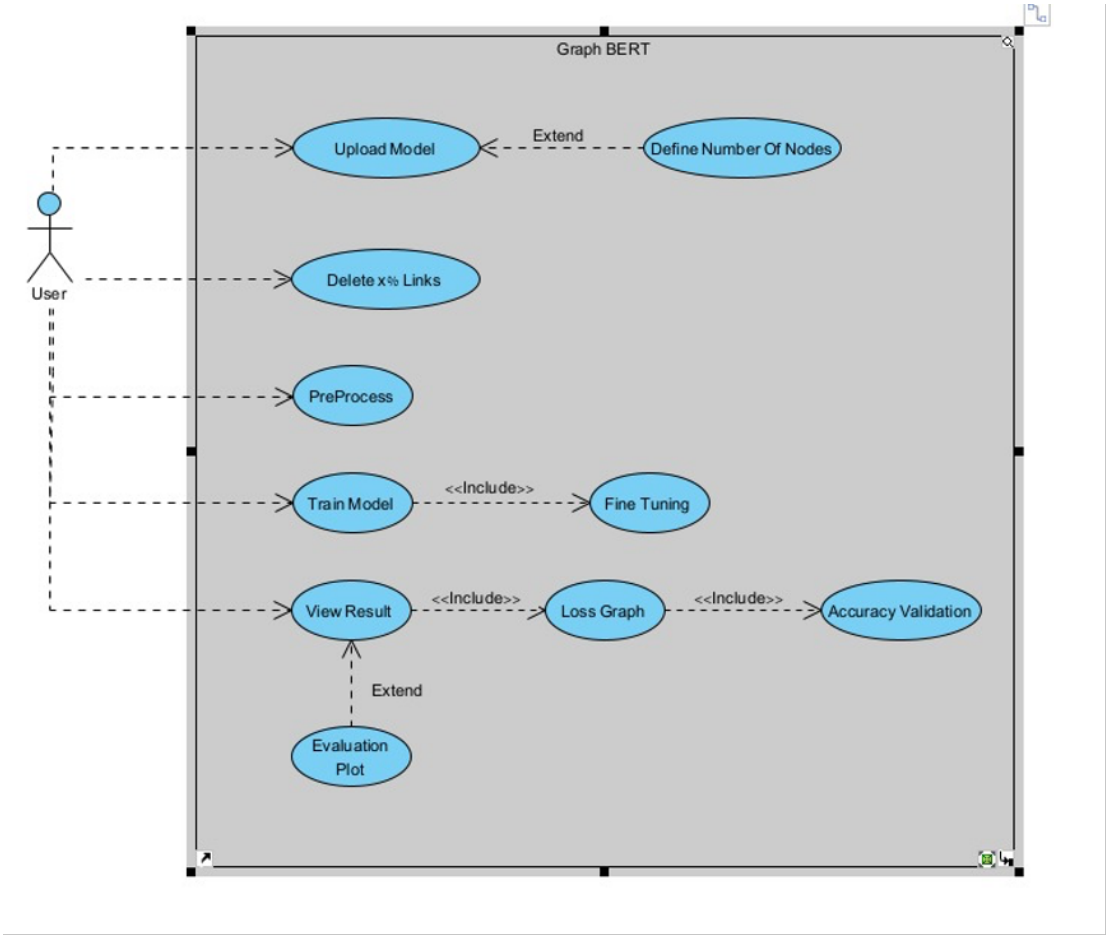

Figure 6: Use-Case

3.3 Data Set

The initial phase of this software engineering project involves acquiring the Cora dataset from the internet. This dataset consists of two files: the "link" file and the "node" file. The "node" file comprises a series of records, each representing an article and containing an ID, a feature vector, and a main subject classification. The feature vector, comprising binary values indicating the presence or absence of each word in a predefined vocabulary, has a size of 1433. The "link" file, on the other hand, consists of records representing connections between articles in the form of pairs of article IDs.

4 Results

4.1 Network Result

In this section, we will show the scores of our model. The model was trained for 150 and 200 epochs. We used the Adam optimizer. The learning rate started from $1 \times e - 02$. We train our model by 500 nodes and links between the nodes.

4.1.1 Train with 150 epochs

- Epoch: 0001 loss_train: 0.0806 time: 1.1541s
- Epoch: 0050 loss_train: 0.0323 time: 0.4207s
- Epoch: 0100 loss_train: 0.0322 time: 0.4421s
- Epoch: 0150 loss_train: 0.0322 time: 0.4184s

4.1.2 Fine Tuning with 150 epochs

- Epoch: 0001 loss_train: 1.9614 acc_train: 0.1380 loss_val: 1.8001 acc_val: 0.2300 loss_test: 1.7083 acc_test: 0.3170 time: 0.1411s
- Epoch: 0050 loss_train: 0.0061 acc_train: 1.0000 loss_val: 0.8224 acc_val: 0.8100 loss_test: 0.6760 acc_test: 0.8420 time: 0.1538s
- Epoch: 0100 loss_train: 0.0039 acc_train: 1.0000 loss_val: 0.8508 acc_val: 0.7700 loss_test: 0.6535 acc_test: 0.8430 time: 0.1482s
- Epoch: 0150 loss_train: 0.0264 acc_train: 0.9920 loss_val: 5.1562 acc_val: 0.7500 loss_test: 5.0331 acc_test: 0.2380 time: 0.1662s

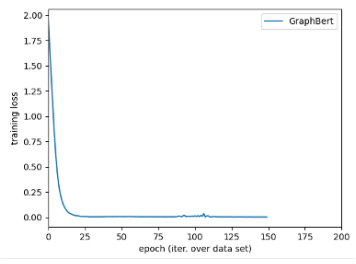

Figure 7: Training loss with 150 epochs

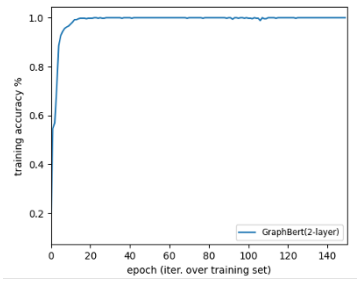

Figure 8: Training acc with 150 epochs

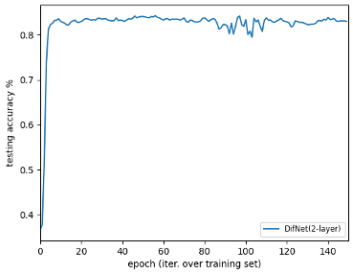

Figure 9: Testing acc with 150 epochs

4.1.3 Train with 200 epochs

- Epoch: 0001 loss_train: 0.0806 time: 0.4334s
- Epoch: 0100 loss_train: 0.0322 time: 0.4168s
- Epoch: 0150 loss_train: 0.0322 time: 0.4244s
- Epoch: 0200 loss_train: 0.0322 time: 0.4112s

4.1.4 Fine Tuning with 150 epochs

- Epoch: 0001 loss_train: 1.9480 acc_train: 0.1620 loss_val: 1.7789 acc_val: 0.2300 loss_test: 1.6961 acc_test: 0.3100 time: 0.1414s
- Epoch: 0100 loss_train: 0.0043 acc_train: 1.0000 loss_val: 0.8748 acc_val: 0.7850 loss_test: 0.7032 acc_test: 0.8340 time: 0.1264s
- Epoch: 0150 loss_train: 0.0151 acc_train: 0.9980 loss_val: 1.0517 acc_val: 0.7600 loss_test: 0.8221 acc_test: 0.8210 time: 0.1419s
- Epoch: 0200 loss_train: 0.0018 acc_train: 1.0000 loss_val: 1.1687 acc_val: 0.7700 loss_test: 0.8564 acc_test: 0.8230 time: 0.1546s

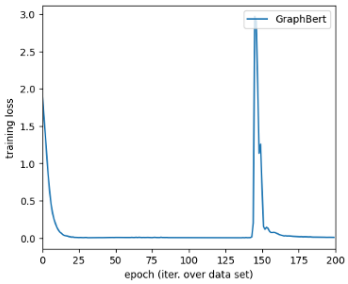

Figure 10: Training loss with 200 epochs

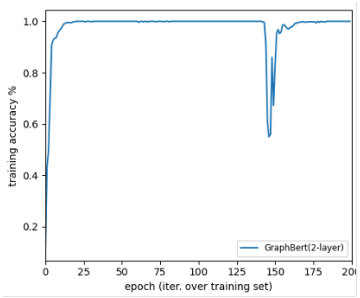

Figure 11: Training acc with 200 epochs

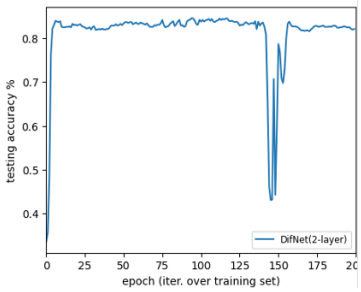

Figure 12: Testing acc with 150 epochs

With these results, the software is trained better with 150 epochs. A jump in the graph for 200 epochs can be caused by different factors:

1. Hyperparameters: The default values of the hyperparameters in the ADAM optimizer may not be appropriate for the current problem, causing the loss to jump around.
2. Overfitting: If the model is overfitting, the training loss may decrease, but the validation loss will increase, and the model will not generalize well.
3. Non-convex optimization problem: If the problem is non-convex, the optimizer may be getting stuck in a local minimum, causing the loss to jump around.

4.1.5 Result for different delete links

In this section we will show the result of the project. After we choose to train the network with 500 nodes, we take 1000 nodes and insert to the trained model. The model output for each node is the probability that the node is belong to specific cluster. For each node we have vector of features with the size 1433, for representing the nodes on graph we use PCA method that reducing dimensions.

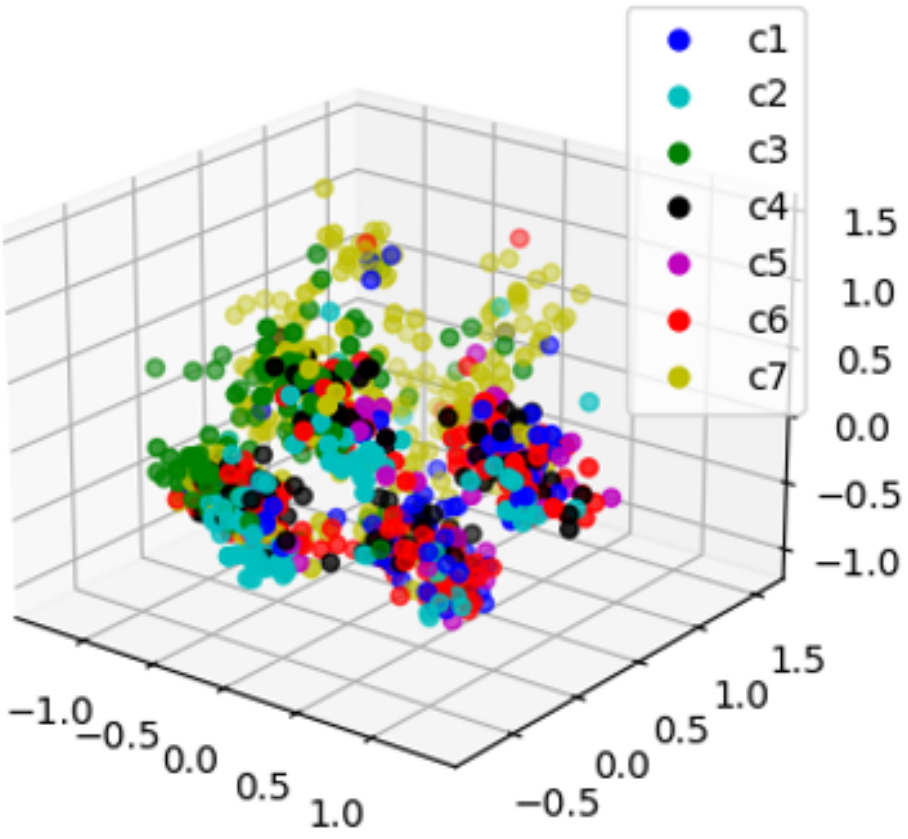

Figure 13: Full clustered data

In this figure, the result for 1000 nodes and their corresponding cluster assignments is shown. Now, we take a subset of 500 nodes that were trained and randomly remove 10 present of their connections. We then train the model on this modified set of nodes and links and apply it to the same set of 1000 nodes. To evaluate the performance, we use the Hungarian cluster matching algorithm to determine the correct cluster assignments.

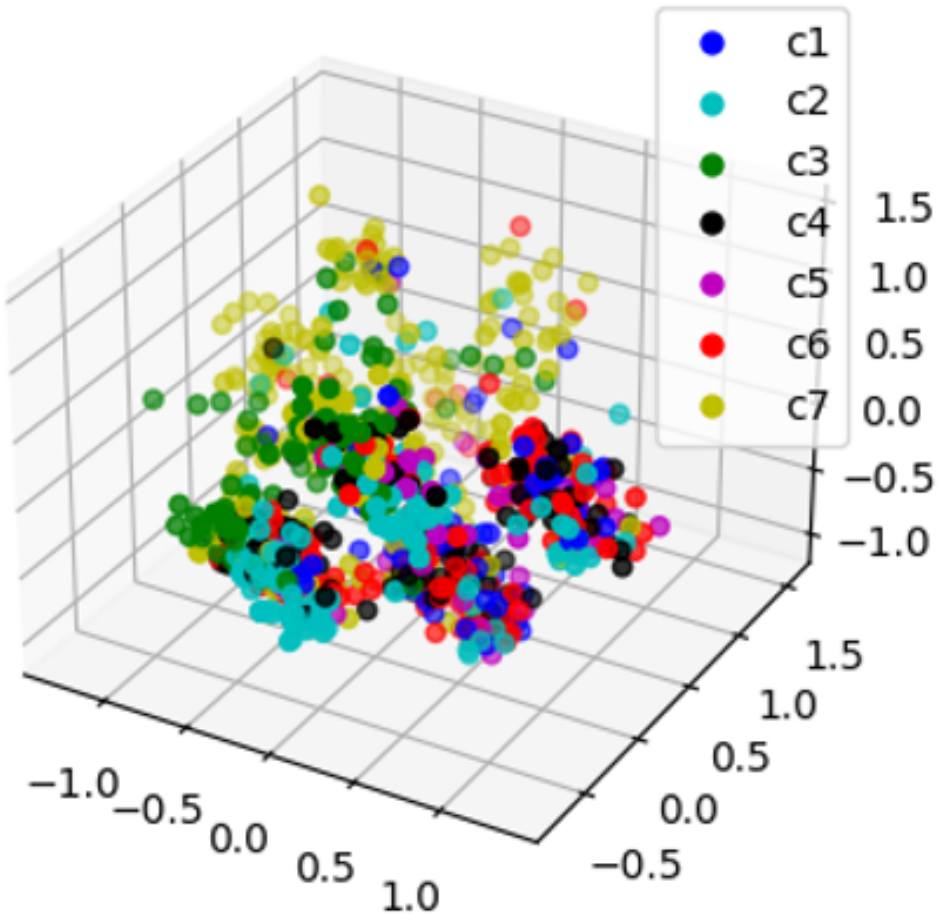

Figure 14: 10 present deleted training clustered data

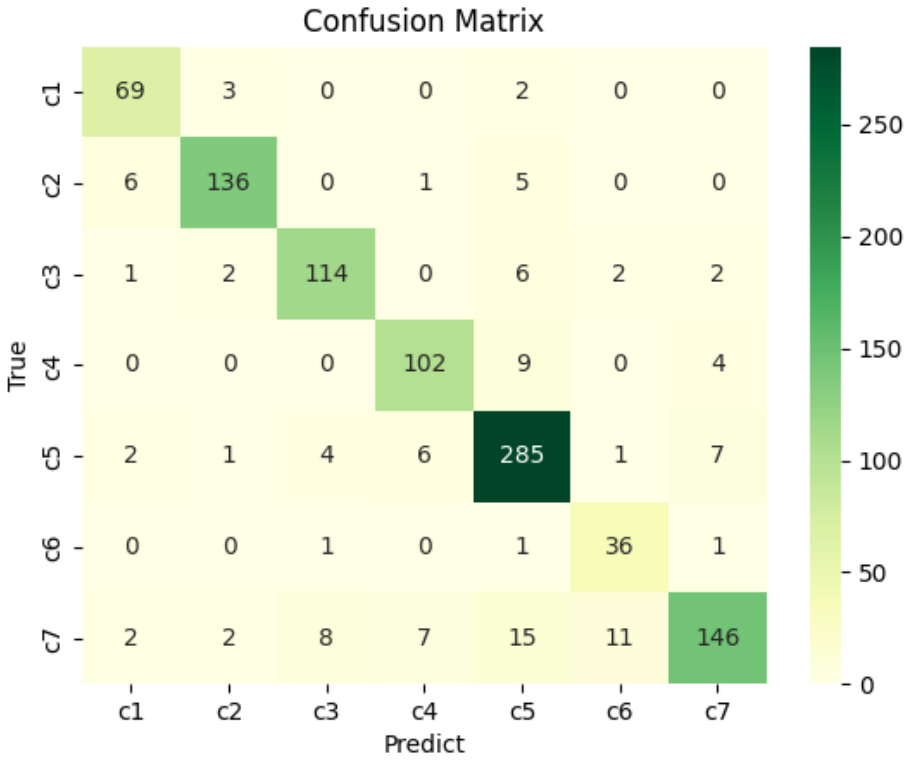

Figure 15: Confusion matrix full training against 10 present deleted

The confusion matrix illustrates the comparison between the clustering results of the full training set and the modified training set where 10 present of the links were removed. The experiment was conducted 20 times and the results were averaged. We can observe that approximately 10 present of the cluster assignments are incorrect. Next, we take the same set of 500 trained nodes and remove 20 present of the links, repeating the process 20 times.

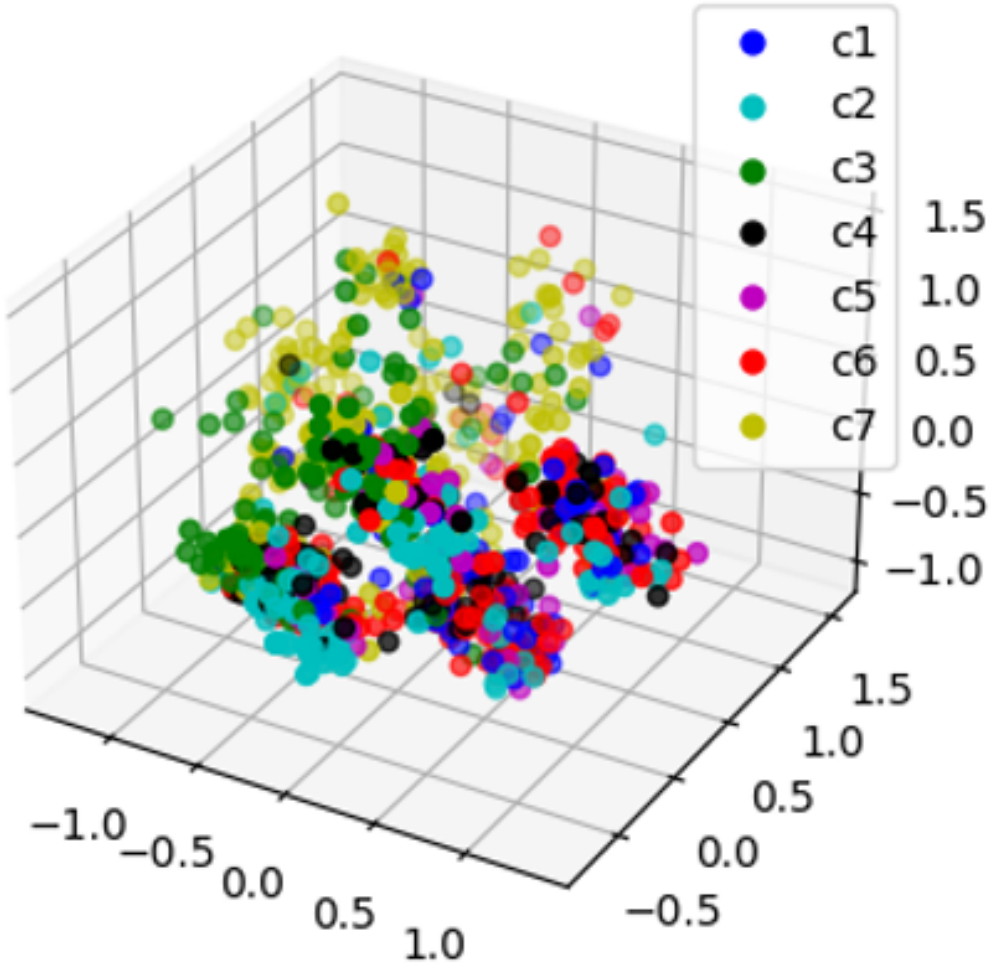

Figure 16: 20 present deleted training clustered data

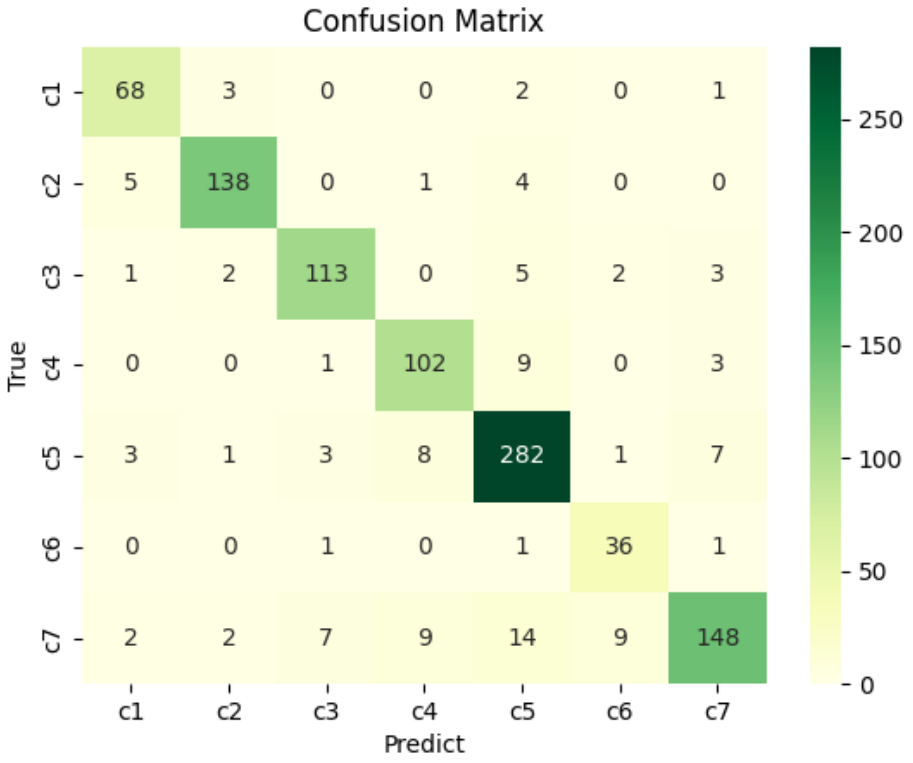

Figure 17: Confusion matrix full training against 20 present deleted

The confusion matrix illustrates the comparison between the clustering results of the full training set and the modified training set where 20 present of the links were removed. The experiment was conducted 20 times and the results were averaged. We can observe that there is little change in the results.

5 Conclusions

In this research, for unnatural connections in the graph, as can be seen in the results of the experiment, we see that the resulting classification is different, apparently indicating a connection that should not exist, so we can actually prove for certain articles whether they were forced to cite or not.

6 User Documentation

6.1 User Guide

The provided GUI code is a user interface for a program called "Graph-BERT", which likely involves some type of machine learning or natural language processing tasks. The GUI allows the user to perform various actions through a series of buttons, such as pre-processing data, fine-tuning a model, and evaluating plots. The GUI also includes a progress bar to display the progress of the current task and a text box for the user to input a percentage of nodes to delete. The general purpose of the GUI is to provide an easy-to-use interface for the user to interact with the Graph-BERT program, allowing them to perform different tasks and view the results in an organized and user-friendly manner.

6.2 Installations

To run the provided GUI (Graphical User Interface) code, you will need to have the following software installed on your machine:

- Python 3.10.
- Tkinter, tkinter, ttk.
- PIL (Python Imaging Library).
- OS.

You will also need to make sure that the necessary dependencies for these libraries and modules are also installed.
You can install these modules by running the following commands on your command prompt or terminal:

- To install Python 3.10:
  Windows: choco install python –version 3.10.0
  Ubuntu: sudo apt-get install Python3.10
- Tkinter: already included in Python3.
- PIL: pip install pillow.
- OS: already included in Python3

Once you have installed all the necessary software and modules, you should be able to run the GUI code without any issues.

6.3 Operating Instructions

The first button on the left panel is the "PreProcessing" button. This button is used to delete a specified percentage of random links from the dataset and preprocess the remaining data. The user can specify the percentage of links to be deleted by entering a number in the text box next to the "Delete number of rows (%)" label. Once the user has entered the desired percentage,

they can click the "PreProcessing" button to start the preprocessing process. This process will update the progress bar as it progresses.

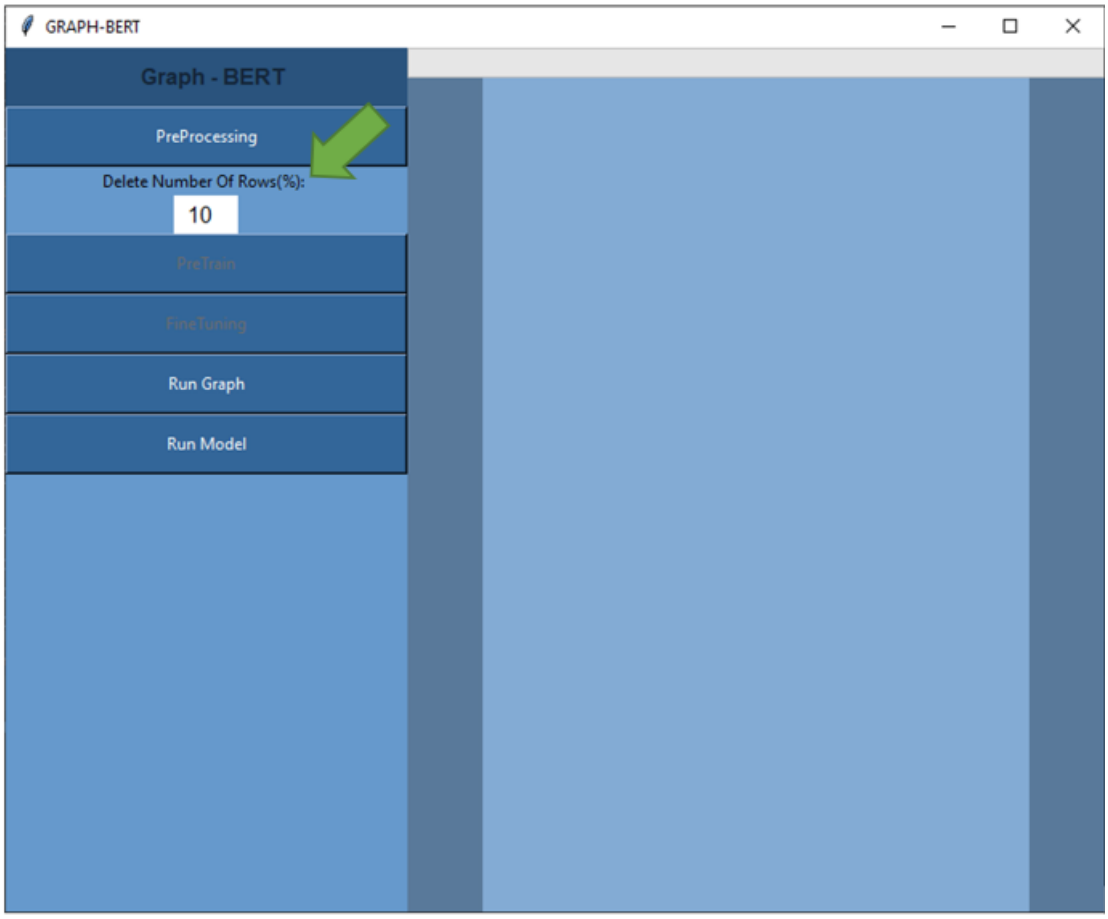

Figure 18: GUI explanation

After the user chose the percentage of nodes to delete (or chose to stay with the default of 10%), the user may click the "PreProccess" button.

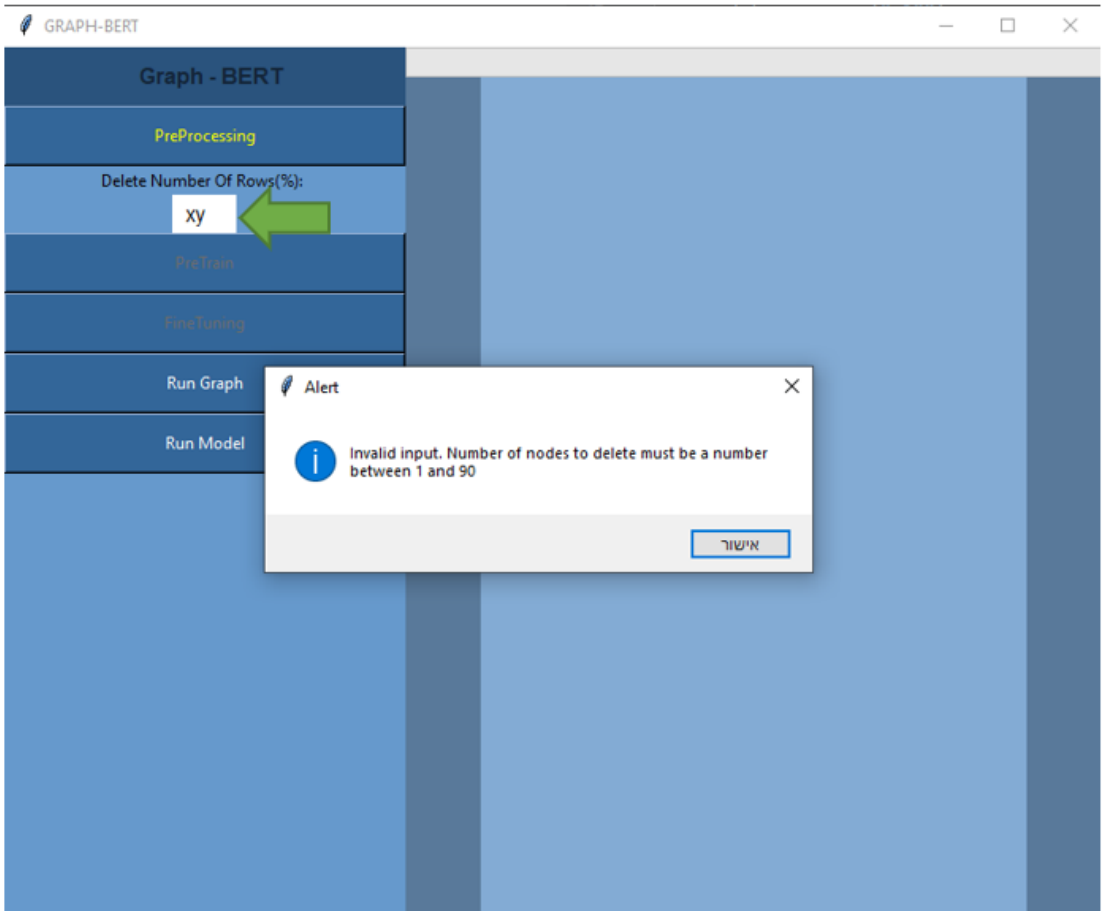

Figure 19: GUI explanation

It is imperative that the user adheres to the specified input guidelines when utilizing this feature. Specifically, it is required that the user inputs a numerical value within the range of 1 to 90. Any deviation from this input range will result in an error being generated. In the event that an error is encountered as a result of an invalid input, the text box will automatically revert to its default value of 10. This serves as a reminder for the user to input a valid value.

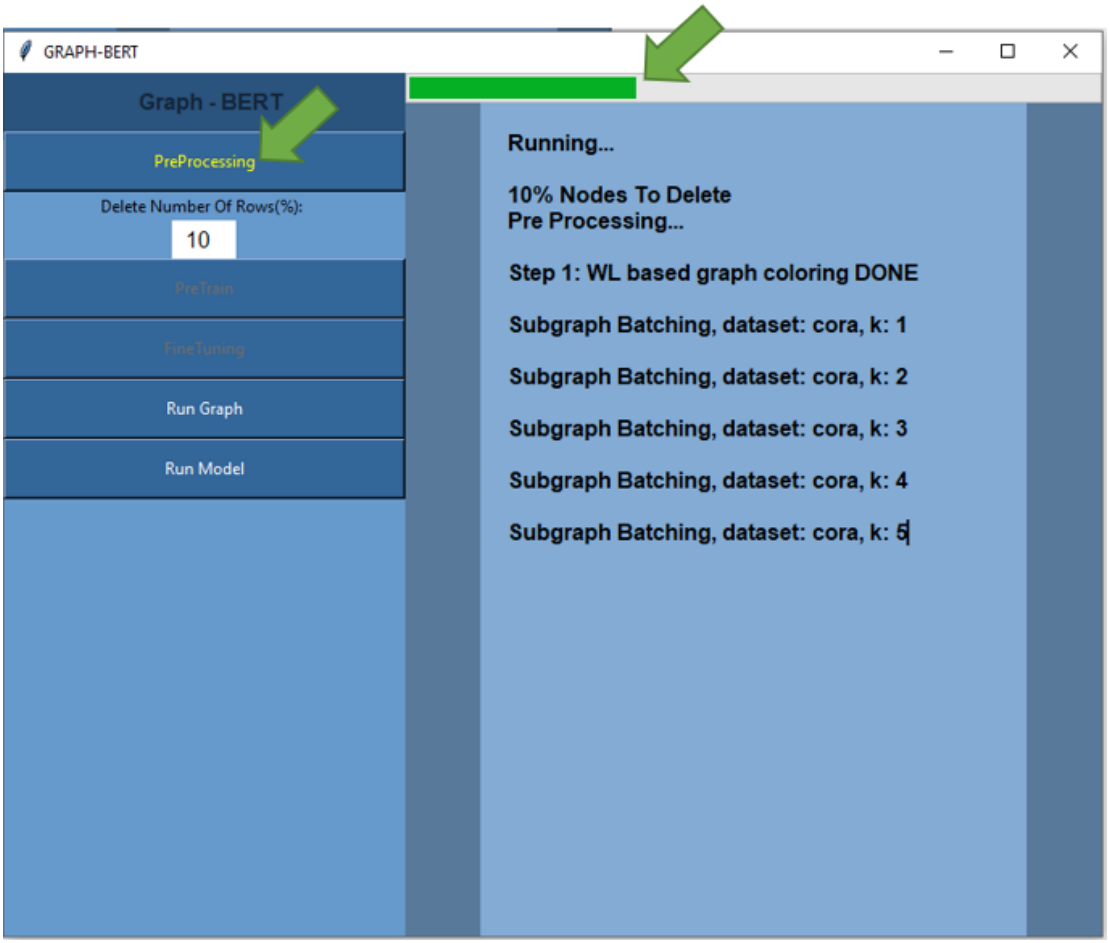

Figure 20: GUI explanation

Upon activation of the "PreProccess" function, the system automatically triggers the subsequent stages of "PreTrain" and "FineTuning". As a result, the buttons for these functions are disabled to prevent manual activation during this automated process. The next button on the left panel is the "Pre-training" button. This button is used to pre-train the model on the preprocessed data.

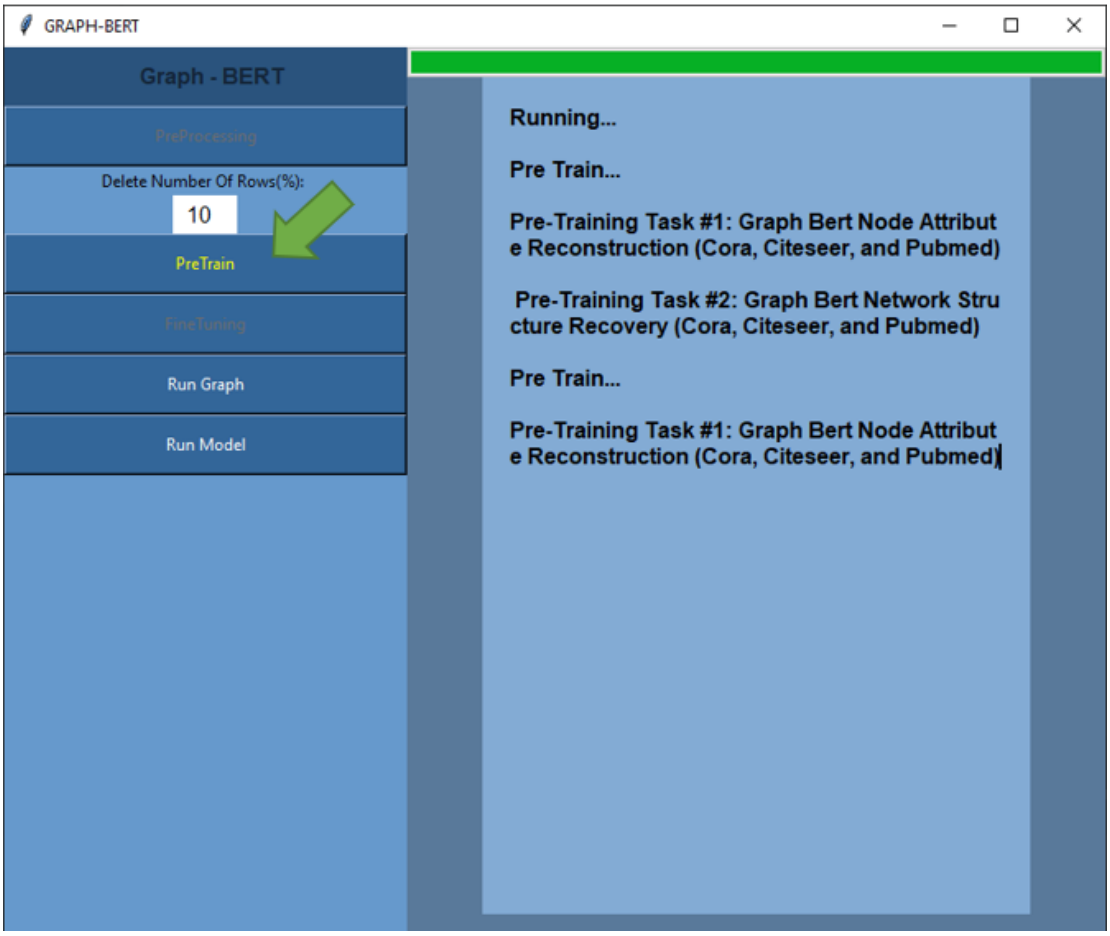

Figure 21: GUI explanation

The "Fine-tuning" button is used to fine-tune the model on a specific task.

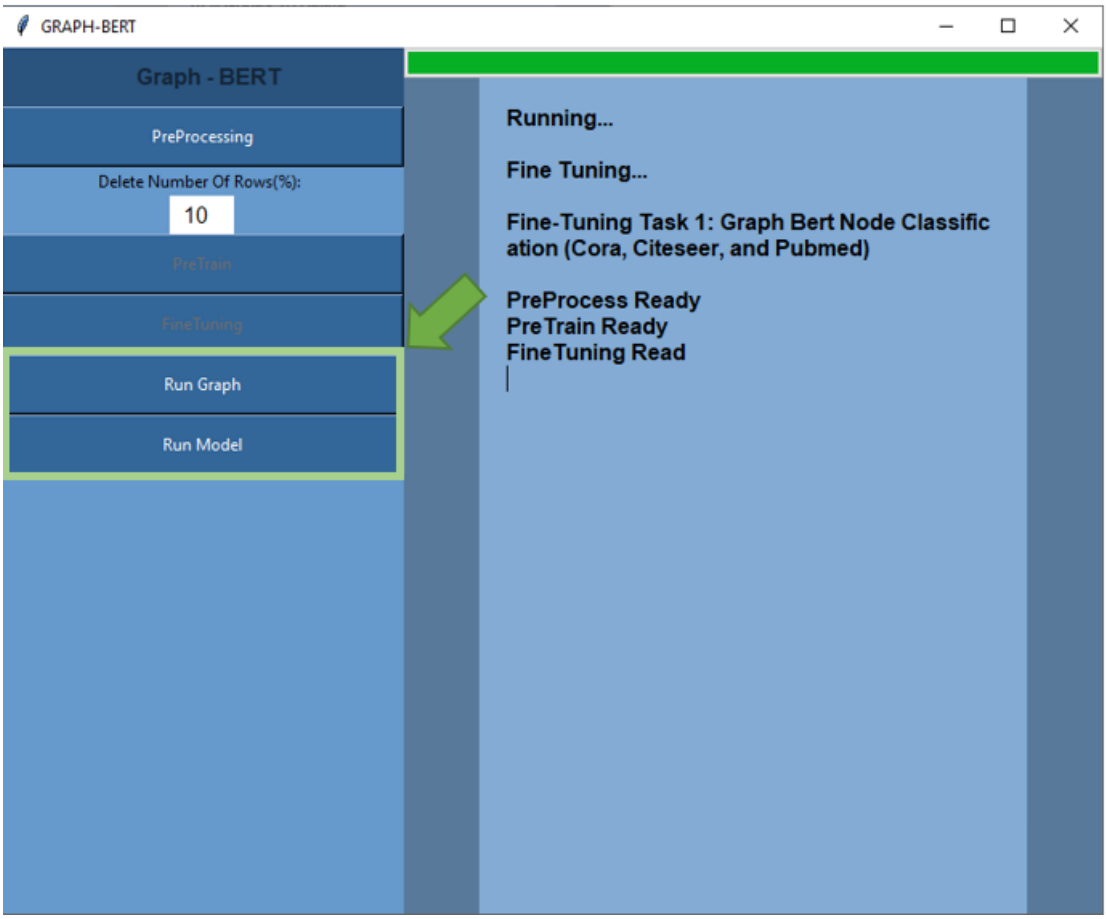

Figure 22: GUI explanation

Upon completion of the three processing stages, the user has the option to re-execute the Pre-Proccess function with a modified percentage or to analyze the performance of the model through the utilization of Evaluation Graphs. This can be accomplished by selecting the "Run Graph" button. Additionally, the user can also deploy the model for use by selecting the "Run Model" button. It is important to note that the generation of Evaluation Graphs and the ability to run the model are dependent on the successful completion of the first three stages and the subsequent creation of relevant images. Re-execution of the PreProccess function will result in the deletion of these images, and subsequently, the disabling of the "Run Graph" and "Run Model" buttons until the completion of a new set of images.

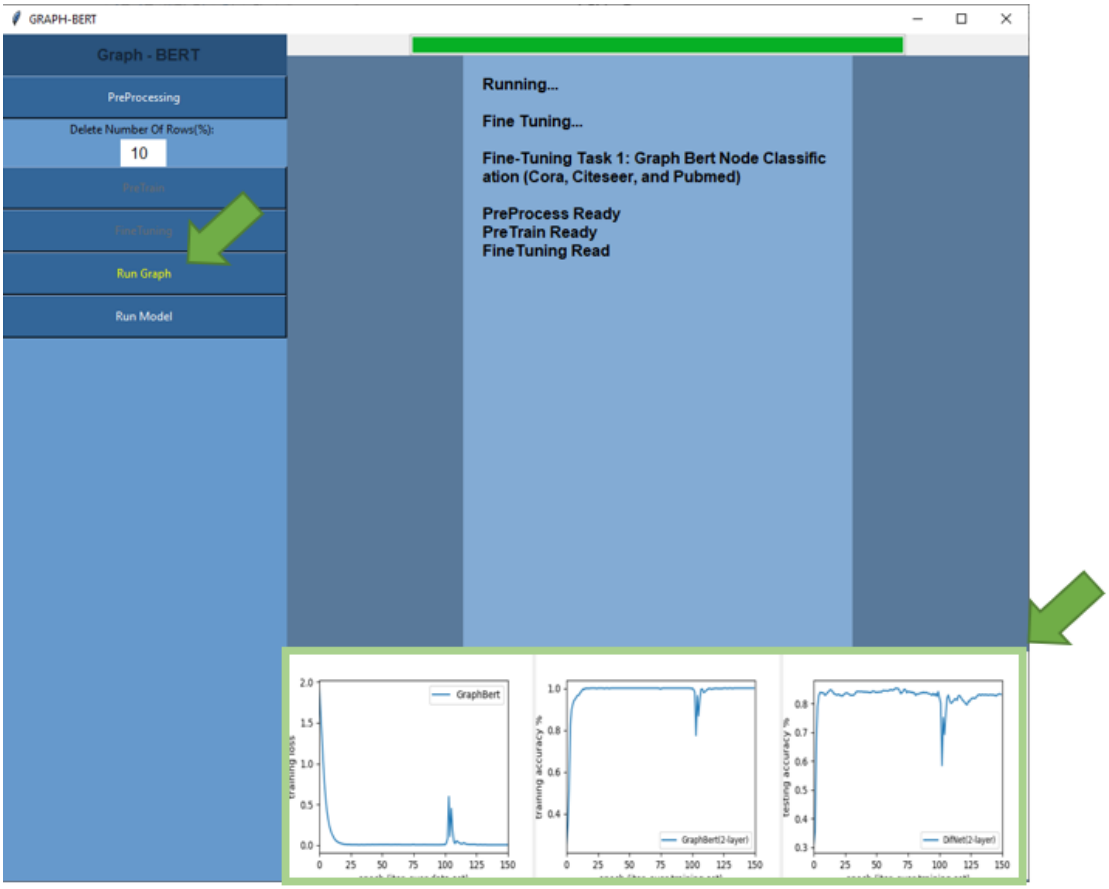

Figure 23: GUI explanation

The "Load & Run" button is used to load and run the model on a specific task. The user can select the task to run the model on by clicking on the drop-down menu next to the button.

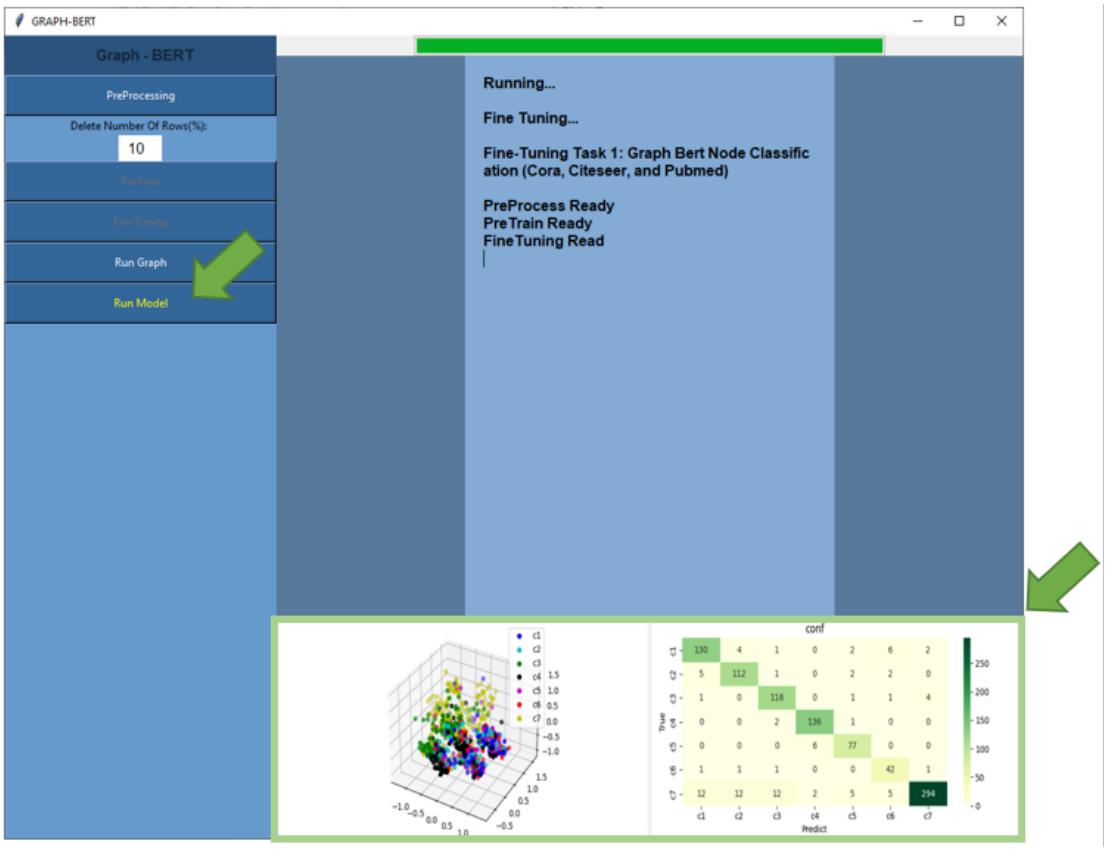

Figure 24: GUI explanation

The right panel of the GUI is used to display the results of the selected operation. The user can also find a text box which shows the current task and its progress.
To use the GUI, the user should first specify the percentage of links to be deleted and click the "PreProcessing" button. Once the preprocessing process is complete, the user can click the "Pre-training" button to start the pre-training process. After pre-training, the user can select a task and click the "Fine-tuning" button to fine-tune the model on that task. The user can then generate evaluation plots for the fine-tuned model by clicking the "Evaluation Plots" button and select a task and click the "Load Run" button to run the model on that task.

7 References

[1] CHURCH, Kenneth Ward. Word2Vec. Natural Language Engineering, 2017, 23.1: 155-162.

[2] MELAMUD, Oren; GOLDBERGER, Jacob; DAGAN, Ido. context2vec: Learning generic context embedding with bidirectional lstm. In: Proceedings of the 20th SIGNLL conference on computational natural language learning. 2016. p. 51-61.

[3] ABIODUN, Oludare Isaac, et al. State-of-the-art in artificial neural network applications: A survey. Heliyon, 2018, 4.11: e00938.

[4] RAGANATO, Alessandro, et al. An analysis of encoder representations in transformer-based machine translation. In: Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP. The Association for Computational Linguistics, 2018.

[5] SCARSELLI, Franco, et al. The graph neural network model. IEEE transactions on neural networks, 2008, 20.1: 61-80.

[6] HE, Xiangnan, et al. Lightgcn: Simplifying and powering graph convolution network for recommendation. In: Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval. 2020. p. 639-648.

[7] ZHANG, Jiawei, et al. Graph-Bert: Only attention is needed for learning graph representations. arXiv preprint arXiv:2001.05140, 2020.

List of Figures