GATTO: Can Topological Information Improve Node Classification via GAT?

Final Report for Learning from Network's project

Francesco Biscaccia Carrara *(2120934)*, Riccardo Modolo *(2123750)*,
Alessandro Viespoli *(2120824)*

Master Degree in Computer Engineering
University of Padova

***Abstract*—This study introduces GATTO (Graph Attention Network with TOpological Information), a framework designed to enhance the performance of Graph Attention Networks (GAT) for node classification tasks by incorporating topological features. The research evaluates GATTO on two citation network datasets (Cora and Citeseer) by adding four topological features. While GATTO showed slight improvements in absolute performance (0.14%-0.54%), statistical analysis revealed no significant difference compared to standard GAT implementation. The results indicate that the additional computational cost of the topological features does not justify their inclusion for small datasets.**

I. Introduction

Node classification is a crucial research topic in graph-based learning and has significant applications in various fields, including social network analysis, recommendation systems, and biological networks. One of the most recent techniques for obtaining a high-quality node classifier is the Graph Attention Network (GAT), in which the graph's embedding is learned by considering the embeddings of neighboring nodes, scaled by learnable attention weights. By performing a transductive learning task, GAT achieves good generalization performance. However, we propose a new framework, GATTO (Graph Attention Network with TOpological Information), which aims to improve classic GAT classification by incorporating topological features derived from the graph or its embedding.

II. Data

A. Original Data

The datasets used are the same as those in the GAT[**GAT**] paper. The table below summarizes the key characteristics of the datasets.

Network	**Nodes**	**Edges**	**Labels**	**Features**
Cora	2708	5429	7	1443
Citeseer	3327	4732	6	3703

In both datasets, each node has a single label. The graphs are directed, unweighted, and do not contain self-loops.

B. Enhanced Data

For our experiment, the following features are computed and assigned to each node:

- **Degree Centrality**: The fraction of nodes to which a node is connected.
- **Betweenness Centrality**: The sum of the fractions of all-pairs shortest paths that pass through a given node.
- **Closeness Centrality**: The reciprocal of the average shortest path distance from a given node to all other reachable nodes.
- **Suggested Label**: The label assigned by clustering the graph's node embeddings.

III. Implementation

A. Module Framework Idea

The practical implementation[**GATTO**] adheres to the conceptual framework. It consists of two main blocks:

- **Precomputation Module**: The class that computes the extra features from the graph (or its embedding) and returns them as a feature matrix.
- **GAT Module**: The GAT implementation for training and predicting node labels.

The implementation of this framework is available on the GitHub repository in the *code* section, written in Python 3.8.10 with the required packages listed in the Singularity.def file.

B. Precomputation Module

Topological features can be computed by importing the dataset into a NetworkX[**SciPyProceedings_11**] object and using the available methods to calculate the desired features. For the embeddings, we use Node2Vec[**node2vec**] with standard parameters. The resulting embeddings reside in $\mathbb{R}^{128}$, and based on the number of labels in the graph, we select:

- **K-Means++** if $\log_{10}(\text{labels}) < 2$.
- **AgglomerativeClustering** otherwise.

These clustering methods are implemented using the Scikit-learn package. Since the framework is designed to be scalable and modular, the extra features are saved into a pandas DataFrame after the feature-building phase. This allows GATTO to function as a standard GAT network if the Precomputation step is skipped.

C. GAT Module

For our experiment, we use the same configuration presented in the GAT paper for **Transductive Learning**.
The network consists of a two-layer GAT model.

1) The first layer has $K = 8$ attention heads, each computing $F' = 8$ features, followed by an ELU (Exponential Linear Unit) activation.
2) The second layer is used for classification: a single attention head computes C features (where C is the number of classes), followed by a softmax activation.

A dropout rate of $p = 0.5$ is applied to both the layers' inputs and the *normalized attention coefficients*.
For the loss function, we use *cross-entropy loss*.
This is implemented using the TensorFlow and StellarGraph packages. Since the input data to the GAT model is binary, we *binarize* the data before training.

IV. Performance Analysis

A. Experimental Setup

The algorithms were executed on two different machines: the *CAPRI High-Performance Computing* system for feature matrix extraction and a general-purpose computer equipped with an *Nvidia RTX 2060* GPU for training and tuning the GAT network. The dataset was divided such that 70% was allocated for training, 10% of the remaining 30% was used for validation, and the rest was reserved for testing.

B. Statistical Test

To compare the performance of the GATTO algorithm with the GAT baseline algorithm, we conducted 10 runs for each technique on each dataset. The following parameters were estimated during the evaluation:

- **Accuracy**: The ratio of correctly predicted instances to the total number of instances.
- **Precision**: The ratio of true positive predictions to the total number of positive predictions.
- **Recall**: The ratio of true positive predictions to the total number of actual positive instances.
- **F1 Score**: The harmonic mean of precision and recall, providing a balanced measure of the two.

We employ the following statistical tests to evaluate and compare the performance of the algorithms:

1) Since all scores are bounded in $[0, 1]$, we use the Shapiro-Wilk Test to assess the normality of the score distributions.
2) If the Shapiro-Wilk Test indicates normality, we perform:
   - Two-Sample t-Test under the assumption of equal variances.
   - Two-Sample t-Test under the assumption of unequal variances.
   - Wilcoxon Signed-Rank Test as a non-parametric alternative.

3) If the Shapiro-Wilk Test rejects normality, we rely solely on the Wilcoxon Signed-Rank Test, as this provides robust results in non-normal scenarios.

The described statistical tests are implemented and available in the GitHub repository as the *Results.R* file, located in the *code/results* section.

V. Results

The following tables present the mean and variance of each score for each algorithm, computed based on the previously described tests.

	Accuracy	Precision	Recall	F1 Score
GAT	0.888	0.891	0.888	0.888
GATTO	0.890	0.893	0.890	0.890

TABLE I: Cora: Averages of the scores for GAT and GATTO

	Accuracy	Precision	Recall	F1 Score
GAT	$2.566e-5$	$2.693e-5$	$2.566e-5$	$2.564e-5$
GATTO	$1.823e-5$	$1.611e-5$	$1.823e-5$	$1.837e-5$

TABLE II: **Cora**: Variances of the scores for GAT and GATTO

	Accuracy	Precision	Recall	F1 Score
GAT	0.737	0.732	0.737	0.731
GATTO	0.736	0.731	0.736	0.730

TABLE III: **CiteSeer**: Averages of the scores for GAT and GATTO

	Accuracy	Precision	Recall	F1 Score
GAT	$2.580e-5$	$2.819e-5$	$2.580e-5$	$2.911e-5$
GATTO	$1.011e-5$	$9.231e-6$	$1.011e-5$	$9.936e-6$

TABLE IV: **CiteSeer**: Variances of the scores for GAT and GATTO

From a superficial analysis, it appears that GATTO performs slightly better in terms of both average and variance values. However, after running the statistical tests described in the previous section [Table V and Table VI], we find that, under the condition of a Type-I error bound $\alpha \leq 0.05$, the p-values returned by the tests indicate there is no statistically significant evidence that the algorithms perform differently, at least on small datasets.

This result might be attributed to the minimal impact of adding a small "piece of information" to the nodes (4 features out of 1447 for Cora, and 4 features out of 3707 for CiteSeer), which does not significantly influence the overall performance of the GAT network. Based on these considerations, the Precomputation Module can be discarded, as it is computationally expensive and does not yield a meaningful performance improvement. While it provides a modest absolute boost of $0.14\% - 0.54\%$, this improvement is not substantial enough to justify the additional computational cost.

A. Confusion Matrices

For each run, the corresponding confusion matrix was saved. All plots are available in the *code/logs* section, with a selection provided as examples [Figure 1].

VI. Conclusion and Future works

As discussed in the previous section, in terms of performance and efficiency, the Precomputation Module can be omitted, relying solely on the GAT module.
However, several potential improvements could be explored in future work:

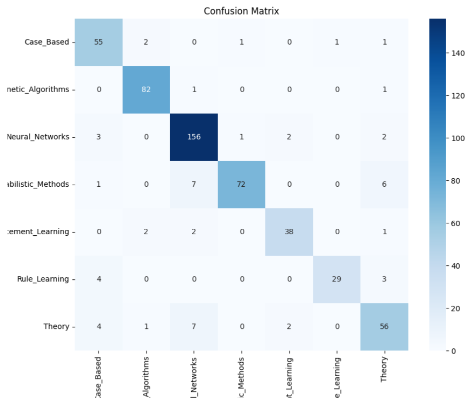
1) Experiment with different types of embeddings prior to the clustering phase.
2) Explore alternative clustering techniques, potentially leveraging MapReduce for scalability.
3) Evaluate GATTO on larger datasets to test its scalability and effectiveness.
4) Perform extensive hyperparameter tuning for the GAT module to optimize performance.
5) Extend the GATTO model to graphs with nodes lacking native features.

	2S-T-Test (same variance)	2S-T-Test (diff variance)	Wilcoxon-Test
Accuracy	0.546	0.546	0.670
Precision	0.529	0.530	0.570
Recall	0.546	0.546	0.670
F1 Score	0.524	0.525	0.677

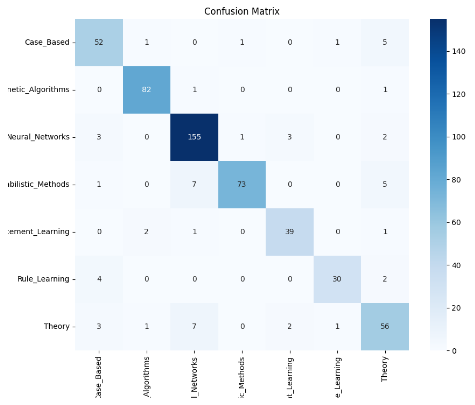
TABLE V: **Cora**: P-values for each test

	2S-T-Test (same variance)	2S-T-Test (diff variance)	Wilcoxon-Test
Accuracy	/	/	0.908
Precision	0.567	0.569	0.969
Recall	/	/	0.908
F1 Score	0.583	0.585	0.969

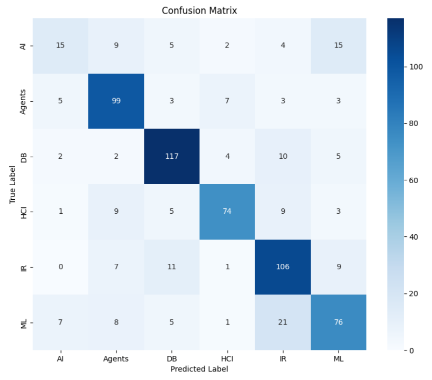
TABLE VI: **CiteSeer**: P-values for each test



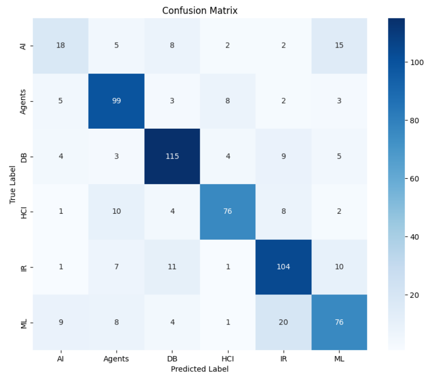
(a) **Cora**: confusion matrix without additional features



(b) **Cora**: Confusion Matrix with additional features



(c) **CiteSeer**: Confusion Matrix without additional features



(d) **CiteSeer**: Confusion Matrix with additional features

Fig. 1: Confusion Matrices for GAT and GATTO

Work Report

In this section, we describe the distribution of the work and the detailed contributions of each member.

- *Francesco Biscaccia Carrara (2120934)*: **40% of the work**. Developed the code for the Precomputation Module, ran tests on CAPRI, and performed statistical tests.

- *Riccardo Modolo (2123750)*: **40% of the work**. Developed the code to retrieve data, reviewed the code, and wrote the Proposal, Midterm, and Final Paper.
- *Alessandro Viespoli (2120824)*: **20% of the work**. Developed the code for the GAT model and for plotting the results.
- *ChatGPT*: Contributed to correcting grammar errors and producing the tables.