Project Draft: Reproduce and Experiment with Graph Attention Networks

Valle-Mena, Ricardo Ruy and Soni, Kushagra
{rrv4, soni14}@illinois.edu

Group ID: 179
Paper ID: 7
Code link: https://github.com/kushagrasoni/CS598_DLH_GAT_Implementation

1 Introduction

Graph Attention Networks (GATs) [1] are a type of neural network architecture designed for processing graph-structured data. They aim to overcome the limitations of previous architectures by being able to operate on arbitrarily structured graphs in a parallelizable manner. GATs use a masked shared self-attention mechanism to assign weights to each node's neighbors and combine their features, resulting in a new set of features for the node in question. GATs have shown promising results in various graph-based tasks, including node classification and link prediction.

2 Scope of Reproducibility

2.1 Background and Problem Statement

Before Graph Attention Networks (GATs), several neural network architectures were proposed to process graph-structured data, including Graph Convolutional Networks (GCNs), GraphSAGE, and DeepWalk. However, these architectures had one or more of the following limitations:

- Inability to operate on arbitrarily structured graphs: GCNs, for example, are limited to processing homogeneous graphs where all nodes have the same features.
- Need to sample from input graphs: some architectures, like GraphSAGE, require sampling from input graphs to create mini-batches, which can lead to information loss.
- Learning separate weight matrices for different node degrees: GCNs use different weight matrices for nodes with different degrees, making the model less scalable.
- Inability to parallelize across nodes: traditional GCNs perform summation operations over a node's neighbors, which cannot be parallelized.

Overall, prior architectures faced difficulties in handling the complexity and heterogeneity of real-world graphs, which motivated the development of Graph Attention Networks.

GATs aim to be able to operate on arbitrarily structured graphs in a manner that is parallelizable across nodes in the graph, thus having none of the limitations mentioned previously. GATs use a masked shared self-attention mechanism. The mask ensures that, for a given node, only features from first-degree neighbours are taken into consideration. The self-attention mechanism allows the model to assign arbitrary weights to each of a given node's neighbours, which then allows the neighbours' features to be combined, which results in a new set of features for the node in question. The paper only mentions using the resulting features for classification tasks, but it should also be possible to use these features for regression tasks.

2.2 Objectives

Our goal is to reproduce the results reported in the original paper, which used the Cora, Citeseer, Pubmed, and Protein-protein interaction datasets. We will do our best to reproduce the same architectural details as in the paper, aiming for

- Classification accuracies of approximately 83.0%, 72.5%, and 79% in the Cora, Citeseer, and Pubmed datasets, respectively,
- Micro-averaged F1 score of approximately 0.973 in the protein-protein interaction dataset.

We hypothesize that our results will be as good or better than those found in previous studies.

In all four cases, the results from the paper found were as good or better as the results found in previous studies. We aim to test the hypothesis that we will also get better results than previous studies and that our results will be similar to those in the paper. Furthermore, we aim to conduct a few ablation studies and architectural variations, such as:

- Replacing the self-attention mechanism with other metrics, such as constants, random weights, and Pearson and Spearman correlation coefficients.

- Replacing the shared weight matrix W with the encoding phase of an autoencoder, by first training an autoencoder on the data.

3 Methodology

All datasets from the paper are publicly available in multiple locations, including the Pytorch Geometric library. We have been running our experiments mostly locally, with some experimentation on Google Colab as well. Both appear to provide sufficient computational resources. This is because the datasets are all relatively small, with the largest of the four being 15.5MB when compressed.

To achieve our goals, we plan to modify the official GAT implementation [2] to suit our needs and make it possible to run the model variants described above.

Through this project, we hope to deepen our understanding of GATs and contribute to the ongoing research in this field.

3.1 Model Description

There are several variants of the same model used in the paper.

For the Cora and Citeseer datasets, two-layer GAT models were used. The first layer has 8 attention heads, projects the input graph's features to an 8-dimensional feature space, and uses an exponential linear unit (ELU) as its activation function. The second layer has a single attention head, projects the data to a C-dimensional feature space, where C is the number of classes in the dataset, and uses a softmax activation function. L2 regularization is applied with lambda = 0.0005, and droupout is applied with p = 0.6.

For the Pubmed data, the architecture is mostly the same. However, the second layer has 8 attention heads like the first layer, and the L2 regularization uses a coefficient of 0.001 instead of 0.0005.

For the protein-protein interaction data, a three-layer model is used. The first two layers have 4 attention heads, project their input data to a 256-dimensional feature space, and use an ELU activation function. The third layer has 6 attention heads, projects its input data to a 121-dimensional feature space, averages all 121 dimensions, and applies a softmax activation function.

3.2 Data Description

We used the same datasets as in the GAT paper: Cora, Citeseer, Pubmed, and Protein-Protein Interaction (PPI). The Cora, Citeseer and Pubmed datasets were originally introduced in [3] and the PPI dataset was introduced in [4].

Table 1 summarizes the basic statistics of the four datasets. Note that the number of features is different for each dataset, as is the number of classes and the sparsity of the adjacency matrix.

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
Pubmed	19,717	44,338	500	3
PPI	56,944	818,716	50	20

Table 1: Dataset statistics; Features represents Features/Node

We used the same dataset splits and evaluation metrics as in the original paper. We also used dropout with a rate of 0.6 to prevent overfitting.

3.2.1 Cora

The Cora dataset consists of 2,708 scientific publications classified into one of seven categories. The citations between publications form a graph, where each node represents a publication and each edge represents a citation.

3.2.2 Citeseer

The Citeseer dataset consists of 3,327 scientific publications classified into one of six categories. Similarly, the citations between publications form a graph.

3.2.3 Pubmed

The Pubmed dataset consists of 19,717 scientific publications from the PubMed database, where each publication is associated with one or more MeSH (Medical Subject Headings) terms. The graph is constructed using the citation links between publications and each node represents a publication. The task is to predict the MeSH terms associated with each publication.

3.2.4 PPI

In the PPI dataset, there are multiple graphs, where each graph represents a tissue and each node in the graph represents a protein. The goal of the task is to predict the biological function labels of the proteins in a previously unseen tissue graph. There are 121 possible labels that a protein node can have, and the task is to predict all of the labels for each protein node in the test graphs. The dataset is divided into 20 training graphs, 2 validation graphs, and 2 test graphs.

3.3 Model Implementation

There are a few different variants of the model used in the paper and therefore in our project.

For starters, we tried using the GAT model that is bundled with Pytorch Geometric. It appears this implementation is not flexible enough to exactly follow what the paper did, but we tried to stay as close as possible, so we called it as follows:

```
from torch_geometric.nn import GAT
model = GAT(
```

```
    in_channels=dataset.num_features,
    out_channels=dataset.num_classes,
    hidden_channels=8,
    num_layers=2,
    heads=8,
    dropout=0.6,
    act='elu',
    act_first=True
)
```

The 'hidden_channels' parameter tells us that the data from each node in the input graph are projected to an 8-dimensional space via a linear transformation (a matrix multiplication). The 'act' parameter tells us that the exponential linear unit is then applied to the transformed data. 'heads' indicates that this is repeated 8 different times, once per "attention head", meaning there are 8 separate linear transformations from the original data's space to 8-dimensional space. 'dropout' indicates that dropout is applied with parameter $p = 0.6$. Lastly, 'num_layers' indicates that what this paragraph describes is repeated twice, with the output of the first later being fed into the second layer.

As mentioned, this does not quite follow the models as described in the paper, but this was a useful step for us to figure out how to feed the data into the model, and more generally how to set everything up. Pytorch Geometric implementation of graph attention networks does not allow, for instance, to specify a different activation function for each layer, which would be required to exactly follow the paper's methodology.

We therefore aim to build our own graph attention network implementation using Pytorch Geometric GATConv class, which implements one layer of the GAT model. Using GATConv directly will allow us to specify different activation functions at each layer and other such details.

We are also attempting to rewrite GAT from scratch, by simply following the mathematical description of the model from the paper. Unfortunately, our implementation will likely be a bit lot slower than any of the currently available implementations on the net as we are still trying to understand, how the graph convolution operation (e.g. Pytorch Geometric GATConv) is equivalent to the mathematical formulae in the paper.

3.4 Computational Requirements

To reproduce the results reported in the GAT paper, we implemented the GAT model using PyTorch version 1.9.0 [5] on two platforms:

- two local systems, a macbook and a Windows both with an Intel Core i7 CPU (2.6 GHz and 1.8 GHz), 16GB RAM, and Intell UHD GPU;
- Google Colab's free GPU instance with 12GB of RAM.

Dataset	Epochs	Score Type	Results
Cora	200	Accuracy	0.82
Citeseer	200	Accuracy	0.73
Pubmed	200	Accuracy	0.79
PPI	71	F1 Score	0.374

Table 2: Results using torch geometric GAT library

On the local system, we installed PyTorch and all necessary dependencies using the pip package manager. At part of next stage of the project, we will try to use the CUDA Toolkit to enable GPU acceleration. Training and evaluating the GAT model on the Cora and Citeseer datasets took approximately 20-25 seconds per 200 epochs, depending on the batch size and learning rate used.

On Google Colab, we used the provided Jupyter notebook environment. We installed PyTorch and all necessary dependencies within the notebook. We also used Google's free Compute instance with around 12GB of RAM, which allowed us to train the GAT model on larger datasets such as Pubmed and PPI. Training the GAT model on the Pubmed dataset took approximately 20 seconds per 200 epochs, while training on the PPI dataset took approximately 30 minutes per epoch which is also not consistent due to large memory requirement.

Overall, the computational requirements for reproducing the GAT results on a local system are moderate, but may require a GPU for faster training times. Google Colab provides a convenient and free platform for reproducing the results, but training times may be longer due to the limited RAM and shared GPU resources.

4 Results

So far we have only been able to inaccurately replicate the paper's experiments: as mentioned earlier, the Pytorch Geometric GAT implementation is not flexible enough to do exactly what the paper does. However, our preliminary results are encouraging.

Our test accuracy on the Pubmed data is essentially indistinguishable from the one reported in the paper, and the test accuracy on the Citeseer and Cora datasets are about 5% worse than the ones reported in paper.

Training the model on the PPI data has proven to be significantly slower than on the other datasets. The results we are reporting here for the PPI dataset are therefore results on the training set. The model has run through 71 epochs and has achieved a micro-averaged F1 score of 0.374 on the training set. The results for the other datasets are accuracy scores on the test set after 200 epochs of training. Refer below Table 2.

We see signs of overfitting in our experiments and believe that reducing the overfitting will improve

our test accuracy further. Specifically, on the Citeseer, Cora and Pubmed datasets, the training accuracy reaches 100%. The paper implements an early stopping mechanism during training, presumably to avoid this exact problem. Unfortunately, there are no details about how the mechanism works; it is described in a single sentence. We will attempt to implement our own such mechanism and hope it will reduce overfitting and improve test accuracy.

5 Limitations and Challenges

We would like to mention that most of the datasets used are quite small and that computational resources should not be an issue. We would also identify any potential limitations or challenges that may arise during the project

References

[1] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[2] Guillem Cucurull Petar Veličković. Gat. `https://github.com/PetarV-/GAT`, 2018.

[3] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, and Brian Galligher. Collective classification in network data. In *AI magazine*, volume 29, pages 93–93. AAAI Press, 2008.

[4] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[5] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037, 2019.