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, which leads to information loss.
- Learning separate weight matrices for different node degrees: GCNs use different weight matrices for nodes with different degrees.
- Inability to parallelize training across nodes.

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 was to reproduce the results reported in the original paper, which used the Cora, Citeseer, Pubmed, and Protein-protein interaction datasets. Specifically, we aimed 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 hypothesized that our results would be as good or better than those found in previous studies.

In all four datasets, the results from the paper found were as good or better as the results found in previous studies. Furthermore, we conducted the following ablation studies:

- Using one, two, and three layer models for the Citeseer, Cora, and Pubmed datasets
- Not using dropout
- Not using L2 regularization

3 Methodology

All datasets from the paper are publicly available in multiple locations, including the Pytorch Geometric library. We have been running our experiments locally. Despite the datasets being relatively small, the PPI dataset turned out to be sufficiently large to take a prohibitively long time to run. We therefore do not provide the same amount of results for the PPI dataset as for the other three.

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 dropout 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 [2] and the PPI dataset was introduced in [3].

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.

We used the same dataset splits and evaluation metrics as in the original paper.

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.

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

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:

```python
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 built three different implementations of GAT, one using Pytorch Geometric’s GATConv class, one using Pytorch Geometric’s GATv2Conv class, and one using Pytorch primitives. This allowed us, to the best of our knowledge, to follow the paper’s methodology exactly. We say ”to the best of our knowledge” because parts of the methodology are not well explained in the paper, and the authors only published the code they used to run the model on Cora in addition to the model itself. There may therefore be details when running the model on the other three datasets where we deviate from the paper. We are confident, however, that any such deviations are small.

3.4 Computational Requirements

To reproduce the results reported in the GAT paper, we implemented the GAT model using PyTorch version 1.9.0 [4] on three 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 Intel UHD GPU;
- Google Colab’s free GPU instance with 12GB of RAM.

On the local system, we installed PyTorch and all necessary dependencies using the pip package manager. Training and evaluating the GAT model on the Cora and Citeseer datasets took approximately 20-25 seconds per 200 epochs.

We installed PyTorch and all necessary dependencies within the notebook. We also used Google’s free Compute instance with 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.

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

4 Results

As part of the experimentation, we utilized 3 different open-source GAT model versions:

- torch_geometric.nn.GAT
- torch_geometric.nn.GATConv
- torch_geometric.nn.GATv2Conv

We were able to closely replicate the paper’s experiments via 3 different: 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 also implemented early stopping strategy which improved the test accuracy results to some extent.

But, since the above GAT models libraries weren’t able to replicate the results 100% with those in the original paper, we tried to write our own GAT model implementation from scratch using torch.nn library. Unfortunately, this attempt couldn’t go as planned as we

5 Limitations and Challenges

As mentioned previously, the PPI dataset turned out to be sufficiently large to make experimenting with it prohibitively slow, so we were unable to experiment with it as thoroughly as with the other three datasets. We are unsure whether getting our code to run on a GPU would change this.

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] 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.

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

[4] 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.