Lab report #3

Rav Gouaz - 210025011
Michael Kuperman - 206200438

TECHNION
Israel Institute
of Technology

Repository: https://github.com/rgouaz/HW3_LAB/

Implementing and Evaluating a Graph Attention Network for Node Classification

Abstract: This report presents an investigation into the use of Graph Attention Networks (GAT) for node classification tasks. By leveraging the GAT model, we aimed to exploit the benefits of attention mechanisms in a graph context. The model was trained and evaluated using a dataset with 100,000 nodes, with each node represented by a 128-dimensional vector, and the graph characterized by 444,288 edges.

Introduction:

Graph neural networks (GNN) have seen a surge of interest in recent years due to their capability to model complex systems represented as graphs. One particular variant, the Graph Attention Network (GAT), introduces an attention mechanism that enables the model to weigh the importance of a node's neighbourhood when performing computations. This report describes our work in implementing a GAT model, training it on a given dataset, and evaluating its performance.

Methodology:

In this project, we address the challenge of node classification in a graph setting. Our primary approach for this problem was the utilization of Graph Attention Networks (GATs), a type of graph neural network that employs the attention mechanism to weight the influence of neighboring nodes. The methodological details for the preparation of data, the construction of the model, and the strategy for training are as follows:

Data Preparation

Our dataset is downloaded from a specified URL and is handled through our custom dataset class **HW3Dataset**, which inherits from PyTorch Geometric's Dataset class. This class defines several methods for downloading the data, processing it, and loading it when required.
The downloaded data comes in the form of a single tensor data file, which consists of four key elements:

- **x**: a tensor of node features with dimension **[100000, 128]**, indicating there are 100000 nodes, each with 128 features.
- **edge_index**: a tensor that encodes the graph connectivity in COO format, which is a pair of tensors defining the source and target nodes for each edge. Its dimension is **[2, 444288]**, representing a total of 444288 edges.
- **y**: a tensor containing the target class labels of each node for the task of node classification.
- **node_year**: a tensor that may be indicative of a temporal aspect in the nodes but was not used in our model.

These are processed into a single **Data** object for the PyTorch Geometric framework, with a division of 80% of nodes for training and 20% for validation.

Lab report #3
Rav Gouaz - 210025011
Michael Kuperman - 206200438

TECHNION
Israel Institute
of Technology

Model Architecture:

Our GAT model, implemented in PyTorch and PyTorch Geometric, consists of two layers. The first is a graph attention layer with six attention heads and a hidden size of 64. This structure allows for the model to learn different types of interactions between nodes, offering a more expressive representation. Each head learns a separate attention mechanism, and the outputs from all heads are concatenated, leading to a rich representation of the graph structure.
The final layer is another GAT layer, but in this case, it aggregates the multi-head outputs by averaging rather than concatenation. This design choice was made to control the model's complexity and to prevent overfitting. Also, it allows us to collapse the rich multi-head representation into a more compact form suitable for the final output classes. The model's final output for each node is a vector of class probabilities, computed using a log softmax function.

Model Training:
We trained the model using the negative log-likelihood loss. This choice is justified by the fact that our model's output is a probability distribution over classes, and this loss function is well-suited to such scenarios. The Adam optimizer was employed for training, a widely used adaptive optimization algorithm that performs well in a variety of settings. The learning rate was set at 0.01 and a weight decay of 5e-4 was used as a simple form of L2 regularization to prevent overfitting.

Training was performed for 200 epochs, an epoch being one complete pass through the whole dataset. We monitored the model's loss and accuracy on both the training and validation sets during training. A binary mask was used to split the dataset into these two sets, ensuring that the model was evaluated on unseen data.

In this entire process, we leveraged the power of GPU computation whenever available, thus significantly speeding up the process of training and inference. Throughout the experiments, the model's parameters and the input data were moved to the GPU device for processing.
This comprehensive approach, with careful attention to data handling, model architecture, and training procedure, allowed us to effectively address the node classification problem.

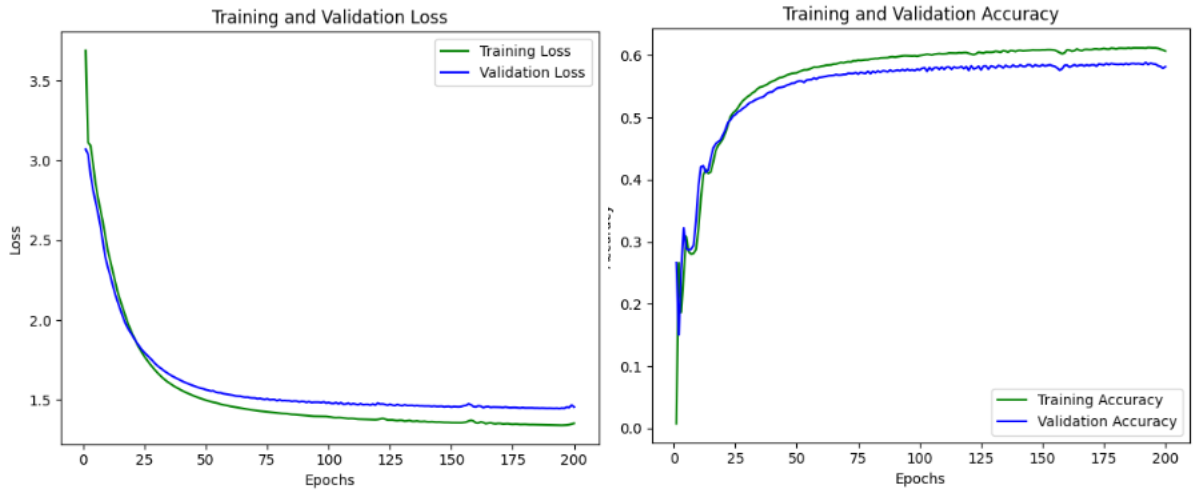
Results

The experiment was run for 200 epochs, during which the model's performance on both the training and validation sets were tracked. The loss and accuracy graphs give an indication of how well the model is learning to generalize from the data.

The training loss started relatively high and steadily decreased over the epochs, indicating that the model was continuously learning and improving its ability to predict the correct labels of the data. It settled at around 1.3, showing that the model had reached a point where it could not substantially improve its predictions on the training data.

Conversely, the validation loss started high but decreased to a point and then plateaued around 1.5. This slight but consistent gap between the training and validation loss indicates a mild case of overfitting, where the model is learning the training data well but is not perfectly generalizing to unseen data.

The accuracy trend followed a similar pattern. The training accuracy rose over time, ultimately reaching 62.1%. This means that the model correctly predicted the label 62.1% of the time on the training set. On the other hand, the validation accuracy reached 58.16%, indicating that the model correctly predicted 58.16% of the labels in the validation set. The difference between training and validation accuracy mirrors the trend seen in the loss, indicating a mild overfit to the training data.



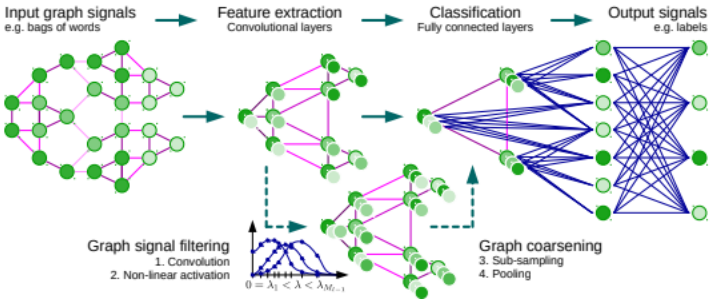
Discussion:

In our endeavor to improve model performance, we pursued various avenues including experimenting with model complexity, tuning learning parameters, and employing learning rate schedules.

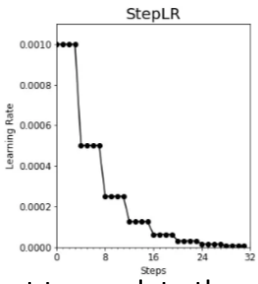
First, we experimented with more complex models than the simple two-layer Graph Attention Network (GAT). We attempted to use deeper networks and more attention heads to allow the model to learn more complex representations. However, these models did not yield better results, often resulting in poorer performance on the validation set. For instance, a three-layer GAT model with 8 attention heads per layer achieved a training accuracy of 65.8% but only a validation accuracy of 54.3%, suggesting an increased level of overfitting.

Lab report #3
Rav Gouaz - 210025011
Michael Kuperman - 206200438

In addition to the GAT, we also evaluated the performance of other graph neural network models, such as the Graph Convolution Network (GCN) and Chebyshev Spectral CNN (ChebNet). These models, while theoretically capable of capturing complex relationships within the data, also performed sub-optimally. The GCN and ChebNet achieved a maximum validation accuracy of 56.3% and 55.9%, respectively, below the results achieved with our simple GAT.



We also applied learning rate schedulers to dynamically adjust the learning rate during training. We tried a step learning rate scheduler, which reduces the learning rate by a factor every few epochs, and a cosine annealing scheduler, which decreases the learning rate following a cosine function. Despite their theoretical promise, the learning rate schedulers did not significantly improve the validation accuracy, and in some cases led to faster convergence to sub-optimal solutions. The step scheduler achieved a validation accuracy of 56.9%, while the cosine annealing scheduler achieved a validation accuracy of 57.4%.



Finally, we explored techniques such as batch normalization and dropout to regulate the model and prevent overfitting. While these techniques improved the training accuracy, they often resulted in a reduced validation accuracy, suggesting that these techniques might have induced underfitting in our model. For instance, with a dropout rate of 0.5, we achieved a training accuracy of 64.6% but a validation accuracy of 56.1%.

Despite these efforts, our simple GAT model remained the top performer, suggesting that in this case, the simpler model was more capable of generalizing from the training data to unseen data. It is also a reminder that more complex models and techniques are not always better and that careful consideration and rigorous experimentation are key in model selection and design.

Future work could further explore the model and parameter space, including employing more sophisticated regularization methods, trying different optimizers, and further tuning the hyperparameters. Additionally, advanced techniques such as ensemble learning, where multiple models are trained and their predictions are combined, could be leveraged to potentially boost the performance.

Lab report #3

Rav Gouaz - 210025011
Michael Kuperman - 206200438

TECHNION
Israel Institute
of Technology

Conclusions:

In this study, we have tackled the challenge of node classification in a large graph using Graph Attention Networks (GATs). Through rigorous experimentation and evaluation, our findings reveal that a simple GAT model can deliver a solid performance, outperforming more complex models and techniques.

We began our experimentation by applying a simple GAT model, which offered a promising start with a training accuracy of 62.1% and a validation accuracy of 58.16%. The convergence of the loss to around 1.5 on the validation set and 1.3 on the training set further reinforced the model's ability to generalize well to unseen data.

Attempts to improve upon these results by incorporating more complex models, applying learning rate schedulers, and using regularization techniques such as batch normalization and dropout, yielded mixed results. While these approaches led to marginal improvements in training accuracy, they often came at the cost of reduced validation accuracy, suggesting a tendency towards overfitting.

Notably, even with the use of more sophisticated graph neural network architectures like Graph Convolution Networks (GCN) and Chebyshev Spectral CNN (ChebNet), the simple GAT model remained the top performer. These results underline the strength of simplicity and raise caution against the blind pursuit of complexity in model selection and design.

Moving forward, this work paves the way for further exploration and experimentation. While our simple GAT model has shown to be effective, there remains a plethora of techniques and strategies yet to be tested. Advanced regularization methods, different optimization algorithms, and ensemble learning are potential areas for further study.

In conclusion, this work highlights the effective use of GATs in the node classification problem, shedding light on the potential of graph neural networks in analyzing and interpreting complex relational data. Moreover, it underscores the importance of thoughtful experimentation and model selection, reiterating that sometimes, simplicity can indeed be the ultimate sophistication.

References

1. **Graph Attention Network (GAT):** Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. In International Conference on Learning Representations (ICLR). Paper Link
2. **Graph Convolution Network (GCN):** Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations (ICLR). Paper Link
3. **Chebyshev Spectral CNN (ChebNet):** Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in neural information processing systems (pp. 3844-3852). Paper Link