Schul, May 2025

NI-GNN - Final Project Report

By Mr. Tal Schul
EuroTeQ, tal.schul@campus.technion.ac.il
Under Prof. Čepek Miroslav

Table of Contents

1. The Data

For the project I processed a subset of the dataset that was described by the paper "Image-based Recommendations on Styles and Substitutes" (McAuley et al., 2015). Specifically, the subset that was curated by the DGL Team (see citation).
Per their description: nodes represent goods, edges indicate that two goods are frequently bought together, node features are bag-of-words encoded product reviews, and class labels are given by the product category.

The dataset contains 13,752 nodes, 491,722 edges, and 10 node classes.

The GNN's objective is to learn a function that maps each node's features and its neighborhood structure to a category label.

2. Preprocessing

I performed an 80/20 train-test split on the dataset to evaluate the model's generalization ability. This was achieved by a node-wise split, meaning that the nodes were randomly divided into training and testing sets. I will clarify that edges that connect training and testing nodes are not removed during the split. This means that the GNN can still access information from a test node's neighbors.

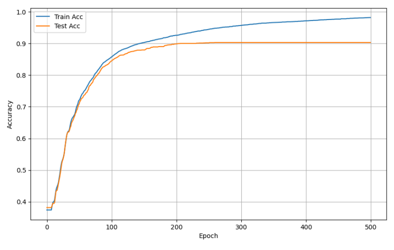
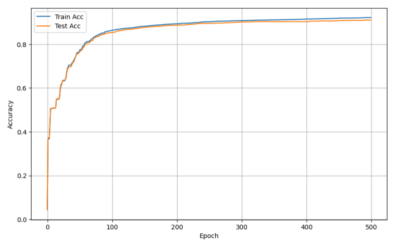
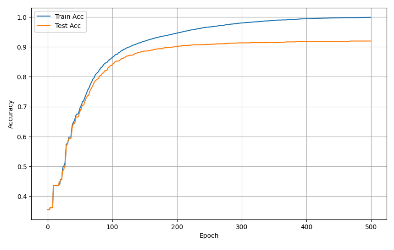
3. Training and Evaluation

All the GNN models were trained for 500 epochs using the Adam optimizer with a learning rate of 0.001 and a cross-entropy loss.
To monitor performance, accuracy is evaluated on both the training and testing nodes after each epoch. In order to see the improvement in accuracy of the different models across epochs I visualized for each epoch the maximum accuracy up to that point. This approach smooths out

fluctuations and highlights the overall trend of improvement, making it easier to see which model reaches higher accuracy faster and how early performance plateaus.
This visualization was generated separately for both the training and testing accuracies for each model.

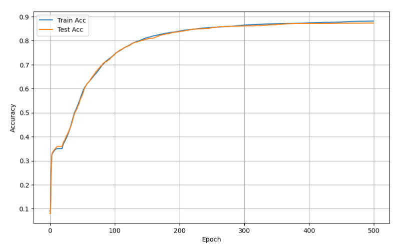
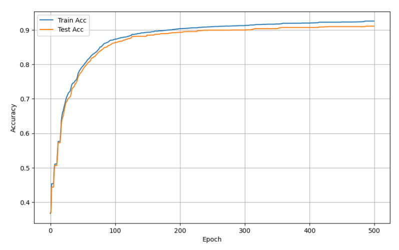
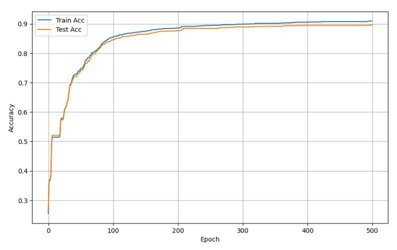
4. Experiments
The primary goal of this project was to build a GNN model and systematically experiment with its architecture.
I implemented and evaluated several configurations of GNNs, focusing on GraphSAGE (Hamilton et al., 2017) and GAT (Veličković et al., 2017).
For GraphSAGE, I tested different aggregator types and layer depths. I also experimented with various hidden dimensions.
In addition, I evaluated GAT models with different numbers of attention heads.

4.1. Varying Aggregator Types in GraphSAGE
In this experiment, the goal was to evaluate how the choice of aggregator function affects the performance of a GraphSAGE model. The aggregator defines how information from neighboring nodes is combined during message passing. I tested three common aggregator types: mean, GCN, and pool. To ensure a fair comparison, other model settings were kept constant across runs: each model used 2 layers, a hidden dimension of 64, and was trained for 500 epochs using the same training/testing split, optimizer, and learning rate as described above.

Mean		GCN		Pool	
					
Max Train Acc	0.982	Max Train Acc	0.922	Max Train Acc	0.999
Max Test Acc	0.903	Max Test Acc	0.910	Max Test Acc	**0.920**

Inherently, the results did not vary by a lot with the pool aggregator (the best one) giving results that are around 2% better than the worse aggregator (mean).

4.2. Varying the Number of GraphSAGE Layers
In this experiment, I tested configurations with 1, 2, and 3 layers to observe how stacking more layers influences the model's ability to capture information from larger neighborhoods.
To isolate the effect of depth, all other parameters were kept constant: each model used the GCN aggregator (that worked decently well in the previous experiment), a hidden dimension of 64, and was trained with the same optimizer, learning rate, and dataset split for 500 epochs.

1 Layer		2 Layers		3 Layers	
					
Max Train Acc Max Test Acc	0.881 0.873	Max Train Acc Max Test Acc	0.925 **0.911**	Max Train Acc Max Test Acc	0.909 0.896

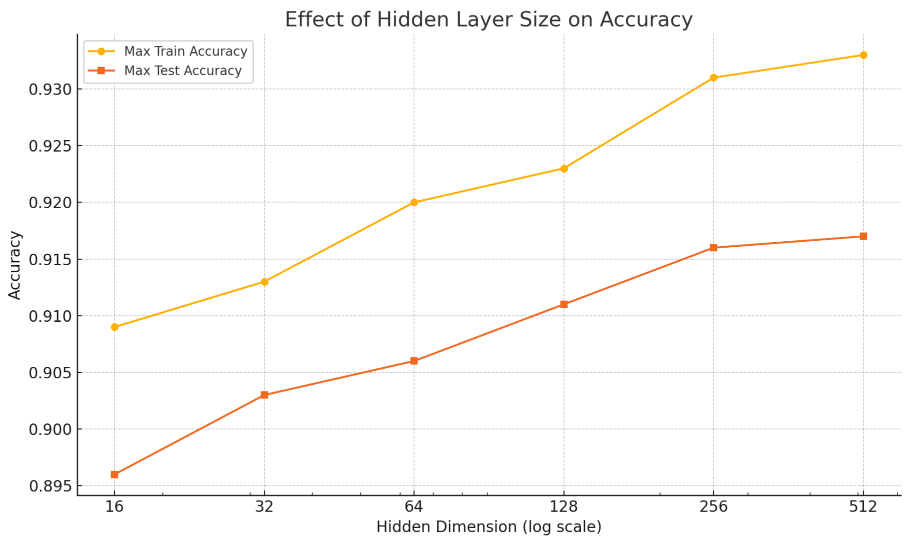
A surprising result of this experiment was that increasing the number of layers did not always lead to better accuracy. One possible explanation is that deeper models have a higher capacity to overfit due to more parameters. However, this hypothesis is not fully supported here, as the training accuracy did not show a significant increase with more layers.
Another possible explanation is the phenomenon of over-smoothing. As the number of layers increases, node representations are aggregated from increasingly large neighborhoods. This can cause the features of different nodes to become similar, hurting the model's ability to differentiate nodes.

4.3. Varying Hidden Layer Dimensions
In this experiment, I explored how the hidden layer size impacts the performance of the GraphSAGE model. I tested a range of hidden sizes: 16, 32, 64, 128, 256, and 512.
To ensure a controlled comparison, all other aspects of the model were kept constant: each model used the GCN aggregator, had 2 layers, and was trained using the same optimizer, learning rate, and dataset split for 500 epochs.

* In order to not overwhelm this project report I will not present all the accuracy graphs here as before.

Hidden Dim	16	32	64	128	256	512
Max Train Acc	0.909	0.913	0.920	0.923	0.931	0.933
Max Test Acc	0.896	0.903	0.906	0.911	0.916	**0.917**

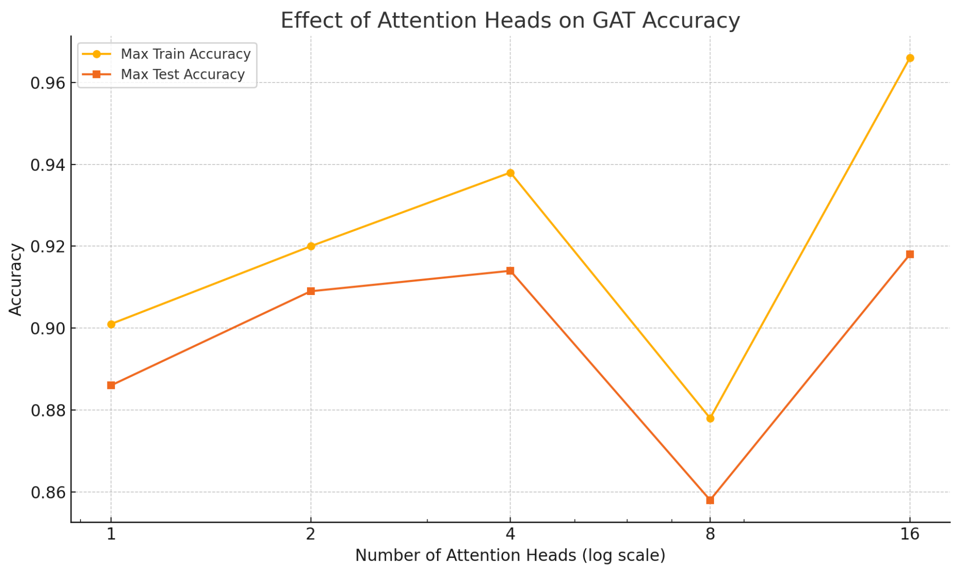
Here is the same information in a plot; showing how different hidden layer sizes affect training and testing accuracy, with the x-axis on a logarithmic scale.



4.4. Varying the Number of Attention Heads in GAT
In this experiment I wanted to show how different types of GNN blocks perform. I explored how changing the number of attention heads affects the performance of a GAT.
I tested GAT configurations with 1, 2, 4, 8, and 16 attention heads. To ensure consistency, each model used a single hidden layer with 64 dimensions. All other training settings stayed the same as before.

* In order to not overwhelm this project report I will not present all the accuracy graphs here as before.

Num Attention Heads	1	2	4	8	16
Max Train Acc	0.901	0.920	0.938	0.878	0.966
Max Test Acc	0.886	0.909	0.914	0.858	**0.918**



Overall it seems that the more attention heads used, the better the model will be, although the results do not seem conclusive.

5. Conclusions

This project evaluated key architectural choices in GraphSAGE and GAT models. Results showed that the pool aggregator outperformed others in GraphSAGE, while a 2-layer model offered the best trade-off between performance and complexity. It was apparent that hidden dimensions improved accuracy. For GAT, more attention heads generally led to better results. Overall, careful tuning of GNN architectures can yield performance improvements.

6. Bibliography

- McAuley, Julian, et al. "Image-based recommendations on styles and substitutes." Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval. 2015.
- DGL Team. (2018). Amazon Buy Computer Dataset. AmazonCoBuyComputerDataset - DGL 2.5 documentation. https://www.dgl.ai/dgl_docs/generated/dgl.data.AmazonCoBuyComputerDataset.html#dgl.data.AmazonCoBuyComputerDataset
- Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." Advances in neural information processing systems 30 (2017).
- Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.