ADLG Project 1: Training GNNs

Ziheng Chi, Federico Felizzi, Jiaqing Xie
Student ID: 22-945-471, 09-901-240, 22-943-914
Department of Computer Science, ETH Zurich
{zihchi, ffelizzi, jiaxie}@student.ethz.ch

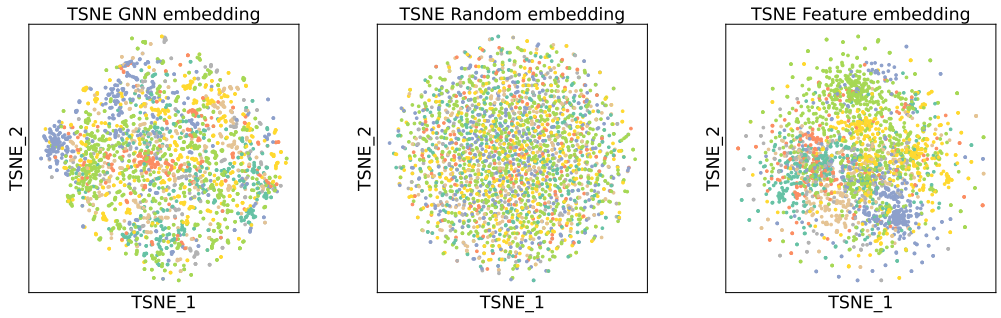
1 Transductive Learning (Click Colab Link here)

1.1 Data Exploration. Cora dataset has **2708** nodes and **5278** edges, with **1433** features for each node. The longest shortest path (diameter) is **19**. Since the graph is not fully connected, we choose the largest diameter among all connected components. Otherwise it should be infinity. Other important graph statistics are explored such as number of classes: **7**, average node degree: **3.90**, number of training nodes: **140** train masks, training node label rate: **0.05**, containing isolated nodes: **False**, containing self-loops: **False**, is undirected: **True**. The numbers of nodes for training, validation and test dataset are 140, and 500, 1000 respectively.

1.2 Label Propagation with full observations. We employed dictionaries to store information about neighbors and determined the label of the central node by considering the most common labels among its neighbors (found in the dictionary values). In case of a tie, we choose randomly. Using this nearest neighbour classifier, we achieved mean **0.86** accuracy with **0.00 (0.001)** standard deviation.

1.3 Baseline without Graph Structure. We used five different supervised learning models: Multi-layer Perceptrons (MLP), Logistic Regression, Random Forests, Gradient Boosted Trees, and Support Vector Machine to predict labels, only considering node features as input. Among all models, Logistic Regression achieved the best performance with mean accuracy of **0.58** and standard deviation of **0.00**.

1.4 Untrained GNNs. We generated three types of embeddings: embeddings from untrained GNN



(a) GNN predictions (b) Random vectors (c) Feature vectors

Figure 1: Visualization of embeddings.

model, real random vectors (same size as the randomized GNN embedding), and vanilla feature vectors. Then we applied TSNE to visualize those embeddings in a 2D space. This experiment has shown that GNNs introduce a strong inductive bias, assigning similar embeddings for nodes close to each other, which could be observed in figure (c). We can observe less clusters in figure (a) with untrained GNN embedding, with worser performance since GNN model is randomly initialized and untrained. There's obviously no clusters shown in figure (b) as the vectors are randomly generated.

We generated three types of embeddings: embeddings from untrained GNN model, embeddings from untrained MLP model, and real random vectors (same size as the randomized GNN embedding). Then we applied logistic regression with these embeddings for classification. The results obtained from three methods varied significantly. We found that the embeddings generated from GNN can

achieve mean accuracy of **0.38** and standard deviation of **0.07**, while applying the same algorithm with real random vectors only achieved **0.14** and **0.00** respectively.

1.5 Trained GNNs. We have applied a two-layer GCN model to perform node classification:

$$\hat{y} = \textbf{Softmax}(\textbf{Linear}(\textbf{GCN}(\textbf{ReLU}(\textbf{GCN}(\mathbf{x}; \Theta)))))$$

We achieved an accuracy of **0.805** and a standard deviation of **0.004** after three runs on test set. A good seed would achieve a better result around 0.82.

1.6 Visualizing Graph Attention Networks. We replaced the GCN layers in task 2.5 by GAT layers to perform the same task:

$$\hat{y} = \textbf{Softmax}(\textbf{Linear}(\textbf{GAT}(\textbf{ReLU}(\textbf{GAT}(\mathbf{x}; \Theta)))))$$

We achieved an accuracy of **0.804** and a standard deviation of **0.000** after three runs on test set. A good seed would achieve a better result around 0.82. Then we performed a forward pass and

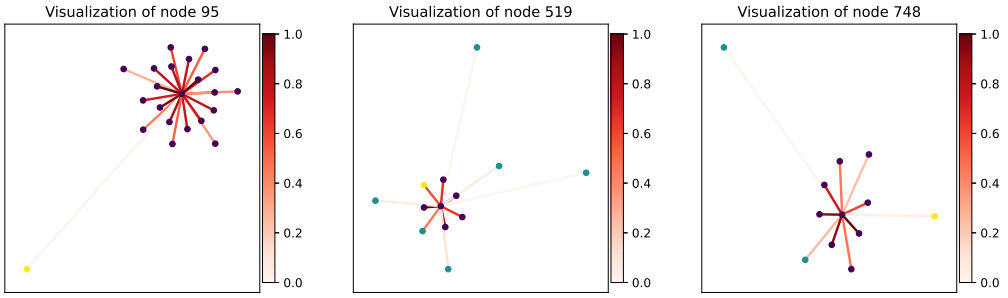


Figure 2: Visualization of attention weights from one-hop neighbors.

extracted the attention weights from the last GAT layer. We randomly chose three nodes from test set and visualized the subgraphs centered around these nodes in Fig. 2. Node colors identify labels, and edge colors illustrate attention weights. We found that neighbours with the same ground truth label were assigned higher attention than those with different labels in GAT layers.

Model	Mean	Standard Deviation
Majority Estimation	0.86	0.00
Multi Layer Perception	0.45	0.00
Logistic Regression	0.58	0.00
Random Forests	0.57	0.02
Gradient Boosted Trees	0.53	0.01
Support Vector Machine	0.56	0.00
Untrained GNN	0.38	0.07
Untrained MLP	0.23	0.03
Random Embeddings	0.14	0.00
GCN Layers	0.81	0.00
GAT Layers	0.80	0.00

Table 1: Results of each subtask in task 2.

2 Inductive Learning (Click Colab Link here)

2.1 Baseline Implementation. In this task, we choose GCN [4], GraphSAGE [3], GAT [5], and GIN [6] as our backbone GNN model. A better design of GNN architectures leads to better performance [9]. As specified in designing space for graph neural networks, a consecutive pre-processing MLP layers, message passing layers with intra-connection, and post-precessing MLP layers ensures a relatively reasonable GNN design. We fix the number of pre-processing MLP layers to be 2, and the number of post-precessing MLP layers to be 3. The message passing blocks of GNN is defined as:

$$h^t = \textbf{PReLU}\left(\textbf{Dropout}\left(\textbf{BN}\left(W_t h^{t-1} + b_t\right)\right)\right) \quad (1)$$

where we use PReLU as our main activation. We fixed some hyper-parameters for testing: model depth = **2**, dimension of graph embedding = **128**, dropout rate = **0.15**, training epochs = **800**, aggregation method = **mean**, batch size = **64**, num of tests = **100** which is required as minimum numner of test samples. We also allow skip connections between layers. GraphSAGE and GIN have shown the best performance, which achieved accuracy of **0.55** (> 0.40) individually. GAT has the worst performance since graph attention doesn't work well in graph-level tasks as it focused more on node representations. The results are presented in table 2.

2.2 Improvement on pooling. Some hierarchical pooling methods [8, 10] successfully solved the flat problem caused by GNN encountered in graph classification tasks. Some other works considered additional feature augmentations such as spectral or topological features [1, 2, 7] Especially in this task, we simply chose one hierarchical pooling methods which is called DiffPool [8]. We use GraphSAGE as our backbone model for testing DiffPool since it performed the best in task 2.1. Specifically, we created an additional assignment matrix to learn assignment with a fixed assignment ratio 0.1, which used the same settings in the original paper of DiffPool. The hidden dimension is set to 64. Number of test sets is 100 as required. Batch size is equal to 25. We also permuted the dataset for three times using three seeds to reproduce results. Observing from table 2, we conclude that using an effective pooling method such as DiffPool will lead to a better performance in graph classification tasks. We run experiments either locally with GeForce RTX 4070 or Google Colab V100 GPU. Running with V100 would have similar results with an average accuracy around 0.6 with assignment ratio 0.25 while other hyper-parameters were unchanged. Unlike in 2.1, choosing specific seeds didn't reach 0.6 in average, in 2.2 we could easily choose seeds to reach an accuracy of 0.61.

Model	Accuracy
GCN	0.46 ±0.05
GRAPHSAGE	0.55 ±0.04
GAT	0.42 ±0.03
GIN	0.55 ±0.04
DIFFPOOL + SAGE	0.61 ±0.01

Table 2: Classification accuracy on Enzymes Dataset with backbone and improved GNN models.

3 Custom Message Passing Layers (Click Colab Link here)

We supplemented the `CustomGraphSage` layer. For the `forward` function, we follow the equation (2) and (3) in project description, considering the effect of normalization. For the `message` function, we simply pass x_j as the neighbourhood information. For the `aggregate` function, we take the advantage of scatter function in `torch_scatter` library with a `mean` reduce operation. The `propagate` function is called by passing `edge_index`, input `x` and `size`. Besides, we have also implemented `CustomGraphSageGRU` where we replaced the mean aggregation in `CustomGraphSage` by `GRUAggregation` while we kept other functions unchanged. More importantly, we used the function `sort_edge_index` defined in `torch_geometric.utils` to avoid further permutation augmentation.

We have compared the performances of three GNN implementations: (1) GNN with `CustomGraphSage` layers; (2) GNN with `SAGEConv` layers; (3) GNN with `CustomGraphSageGRU` layers. We fixed the same set of hyper-parameters for all models in experiments: num_layers = **3**, pool = **global_mean_pool**, pre_processing = **False**, post_processing = **True**. Especially, we did not follow the same GNN design space as we did in task 3, since here we only aim to check the equivalence of `SAGEConv` and `CustomGraphSage`. Final results show that these two classes shown approximately the same functionality and `SAGEConv` improves somehow with GRU aggregation.

Model	Mean	Standard Deviation
CUSTOMGRAPHSAGE	0.31	0.05
SAGECONV	0.33	0.05
CUSTOMGRAPHSAGEGRU	0.42	0.05

Table 3: Comparison of different GNN layers in task 4.

References

[1] Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gauzere, Sebastien Adam, and Paul Honeine. Bridging the gap between spectral and spatial domains in graph neural networks. *arXiv preprint arXiv:2003.11702*, 2020.

[2] Ting Chen, Song Bian, and Yizhou Sun. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv preprint arXiv:1905.04579*, 2019.

[3] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[4] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016.

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

[6] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[7] Mingqi Yang, Yanming Shen, Rui Li, Heng Qi, Qiang Zhang, and Baocai Yin. A new perspective on the effects of spectrum in graph neural networks. In *International Conference on Machine Learning*, pages 25261–25279. PMLR, 2022.

[8] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

[9] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33:17009–17021, 2020.

[10] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*, 2019.