EEG Analysis with Graph Neural Network

Tsai Hao-Yu
Department of Computer Science,
National Tsing Hua University
Kaohsiung City, Taiwan
ipadwifi20000@gmail.com

Chang Chia-Hsin
College of Electrical Engineering & Computer Science,
National Tsing Hua University
Hsinchu City, Taiwan
claristathio@gmail.com

Chou Kuang-Chi
College of Electrical Engineering & Computer Science,
National Tsing Hua University
Tainan City, Taiwan
joeckc17@gapp.nthu.edu.tw

Wang Wei-Hsiang
Department of Computer Science,
National Tsing Hua University
Taoyuan City, Taiwan
wayne072392@gmail.com

Chuang Kai-Hsiang
Department of Computer Science,
National Tsing Hua University,
New Taipei City, Taiwan
kevin2002611@gmail.com

Chen Yen-Cheng
Department of Computer Science,
National Tsing Hua University,
Hsinchu City, Taiwan
kevin900808@gmail.com

***Abstract*—We combined our idea with some previous studies to create different models to analyze an Electroencephalogram (EEG) dataset. We mainly tried three kinds of models: Graph Convolutional Network (GCN) model, Graph Attention Networks (GAT) model, and GraphSAGE model. To make those models work better, we tried several ways such as data preprocessing, edge formulation, and pooling. The various combinations we tried resulted in around thirty percent accuracy at most. In order to make the results better, we also tried some additional models and merge the categories of the dataset.**

I. Introduction

EEG measures and records the electrical impulses which is produced by the brain's neurons by attaching electrodes to the scalp. Convolutional neural networks (CNN) have been frequently used to extract subject-invariant features from EEG for classification tasks. However, traditional machine learning approaches such as CNN and RNN failed to utilize the spatial relations of the EEG electrodes. Therefore, we utilized graph neural network (GNN) to analyze EEG in this research. With GNN, we can exploit the spatial information, which shows potential for improvement.

Ten participants, aged 21 to 57 years (median age 30.5 years; 3 female; 1 left-handed), were included in the dataset in our project. All participants reported normal color vision and either normal or corrected-to-normal vision. 72 photographs of real objects were used as stimuli, taken from the 92-image set used in past RSA studies. The quantity of inanimate images is reduced in the set in order to ensure that each image category would contain the same number of exemplars—specifically, 12 images from each of the following six categories: Human Body (HB), Human Face (HF), Animal Body (AB), Animal Face (AF), Fruit Vegetable (FV), and Inanimate Object (IO). Each participant completed two experimental sessions, each containing three blocks, spaced between six and eight days apart. In total, each participant completed 72 trials of each of the 72 images, for a total of 5,184 trials per participant.

In this project, we tried several different data preprocessing methods, edge formulations, pooling methods, and models which have varying results and provide us more information about the nature of the models and EEG in general. And we tried to figure out that which combination would generate the best accuracy.

II. Methods

We built multiple different GNN models, and then we tried to improve the results generated by the models through data preprocessing, pooling, and different training methods.

II.1. Models

To apply GNN on this problem, we must transform our data into graph. Therefore, we treated each electrode as nodes and their time samples as features, and we only needed to construct edges between nodes. We assumed that if there is an edge between two nodes, their features should be similar. Hence, we used k nearest neighbors (KNN) with different distance metric, such like Euclidean distance, cosine distance and Pearson correlation coefficients, to construct edges. After transforming our data into graph, we could use GNN to classify our data. We mainly tried the total of three models in our study: GCN model, GAT model, and GraphSAGE model.

II.1.1. GCN Model

In GCN, we tried to use adjacency matrix A to spread the information of each node to their neighbors. So, we have the first formula. Let $H \in R^{N\times D}$ be the matrix of activations and $X \in R^{N\times D}$ be input feature matrix of N nodes with D features:

$$H = AX$$

However, there are two problems. The information of nodes themselves would not be preserved after propagation and the degree of nodes would affect the new features. For the first problem, we can add self-loop in the graph, so we use new adjacency matrix $\tilde{A} = A + I_N$ in new formula. For the second

problem, we use degree matrix $\widetilde{D}$, where $\widetilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, to regularize nodes' features. Since $\widetilde{D}^{-1}$ is the matrix, whose elements are reciprocals of those of $\widetilde{D}$, so nodes will use the weighted average of their neighbors' feature to update. The second formula will be:

$$H = \widetilde{D}^{-1}\tilde{A}X$$

But $\widetilde{D}^{-1}\tilde{A}$ only scale by rows and ignore their corresponding columns. Hence, we need to add another $\widetilde{D}^{-1}$ to scale their columns. Nevertheless, $\widetilde{D}^{-1}\tilde{A}\widetilde{D}^{-1}$ would use the square of degree to scale features and the degree of nodes can still affect the new features. Hence, we use $\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}$ to regularize features. The third formula will be:

$$H = \widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}X$$

Finally, we add layer-specific trainable weight matrix $W^{(l)}$ for training and activation function $\sigma(\cdot)$ for non-linear operation. As a result, the final propagation rule will be:

$$H^{(l+1)} = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

, where $H^{(l)} \in R^{N\times D}$ is the matrix of activations in the l^{th} layer and $H^{(0)} = X$.

II.1.2. GAT Model

We use the module on torch_geometric.nn.conv.GATConv to implement the GAT method. An attention mechanism is used to calculate the weight of each node to its neighbors, which enables the model to dynamically adjust the strength of relationships between different nodes. The self-attention mechanism allows each node to adjust its weight according to its own characteristics and its neighbors. This enables GAT to learn the interactions of nodes with their neighbors rather than just fixed predefined weights. The attention mechanism of GAT also uses the softmax function, which allows the model to automatically learn the relationship weights between different nodes.

Additionally, the GAT model has built-in regularization terms because dropout is used in the attention mechanism. The use of dropout in the attention mechanism is a simple and effective regularization technique that helps prevent model overfitting. The formula of our GAT model is as the following:

$$Attention(h_i, h_j) = softmax(\frac{(Wh_i)^T(Wh_j)}{\sqrt{d}})$$

h_i and h_j are the features of nodes i and j respectively, W is the weight matrix of learning, and d is the dimension of the feature. The purpose of this attention mechanism is to calculate the degree of attention of nodes i and j and use it as the attention weight. After testing some dropout parameter, we found that we could have the best results by setting it to 0.5 ~ 0.6.

II.1.3. GraphSAGE Model

In the GraphSAGE model, we first randomly sampled a fixed-size subset for each node from its neighbors. This helps with large graphs since all neighbors do not need to be considered, only a subset is concerned.

In the process of forward propagation, GraphSAGE uses fully connected layers and CNN to convert each node and sampled neighbors into low-dimensional embedding, which represents the information of the node and local structure. Generate node embedding using k hop (for each hop: k = 2,3,…,K).

Then we can use CNN to obtain the representation of node v in k hop for the neighbors of v in k-1 hop, which includes the information of the target and neighbor nodes in k hop. Finally, z(v) is obtained, the representation of node v in K hop, including all information from 1 hop to k hop.

To integrate information, We use average aggregation to merge multiple features into a new feature. Average aggregation let each dimension of the neighbor embedding be averaged, and then splice it with the target node embedding for non-linear transformation. The aggregated embedding is used as the input of the fully connected layer to predict the label of the target node.

II.2. Data Preprocessing

For the data preprocessing part, we tried three methods: Fast Fourier Transform (FFT), Discrete Wavelet Transform (DWT), and Principal Component Analysis (PCA) to preprocess the dataset.

In FFT, amplitudes are obtained from the result of fast Fourier transform of raw EEG signals on each channel. The raw data is replaced by a vector containing 16 different frequency amplitudes, ranging from 0Hz ,1.953125Hz, 2*1.953125Hz, to 15*1.953125Hz.

In Discrete Wavelet Transform, the choice of an appropriate wavelet and the number of decomposition levels is crucial for signal analysis. Previous studies [1][2] often utilized Daubechies 4 (db4) at level 4 due to its smoothing feature, which is suitable for detecting changes in EEG signals. However, due to limitations in the number of data points, level 2 is selected in this study. Detail coefficients (D1, D2) and the approximation coefficients (A2) are obtained. The energy and standard deviation of coefficients are used as input data, with energy calculated as the sum of the square of each coefficient's elements and standard deviation calculated as the standard deviation of each coefficient's elements.

In PCA, we applied the PCA technique to preprocess the EEG time domain signal, hoping to remove the secondary pieces of signal, and improve the model performance compared to other data preprocessing techniques. The EEG signal we had was sampled into 32 timestamps, with 124 channels. We seen this data as 124 by 32 array and fed it into PCA to try to reduce a specific portion of dimension (about 1/3), and therefore the number of the features of a single channel becomes 24.

However, we finally figured out that the accuracy with raw data is higher than the accuracy with the preprocessed data through FFT, the preprocessed data through DWT, and the preprocessed data through PCA. We then use raw data directly as the input for the model.

II.3. Pooling

We tried the most common and simplest methods called global mean pooling as well as global max pooling, which

returns the average node features and the channel-wise maximum, respectively. After implementing this pooling method, consider the performance still didn't meet our expectations, we referred to the methods mentioned in [3], trying to implement the Set2Set method and SAGpool (self-attention-graph pooling), both have learnable parameters to optimize.

The Set2Set method make use of the Seq2Seq framework, using LSTMs with attention mechanisms for graph level pooling. The LSTM learn to generate a final context vector served as the aggregated representation of the entire set of node embeddings. The SAGpool method, using attention mechanisms to rank every node to retain the top k important node (default is half of the total nodes). It is a node level pooling so hence we combine the global mean pooling and global max pooling to generate the final graph representation.

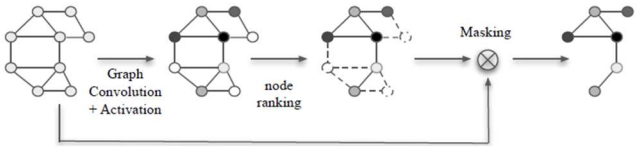

Fig 1. Down sampling with SAGpool operator

To implement these three methods, we make use of the Pytorch geometric framework, it packages these three methods so that we can simply call them.

II.4. Training Method

To make the performance look better, we turned to train 10 models for 10 participants, and took the averaged performance as result. Each table in the Results section used this method.

III. Results

III.1. Edge Formulation

After K GCN layers, each node's information would be spread to their k-hop neighbors. But as K get larger, the model may suffer over-smoothing problem. Hence, in this project, we only use three GCN layers: K = 3, K = 6, and K = 10 in our models. Table 1. shows the accuracy of each model with different distance metrics (Pearson correlation coefficients, Euclidean distance, and cosine similarity) and different edge formulations.

K = 3	GCN	GAT	GraphSAGE
Pearson	28.8247%	27.5915%	28.1503%
Euclidean	29.4412 %	26.0501%	27.8035%
Cosine	29.1137%	26.9557%	27.8035%

K = 6	GCN	GAT	GraphSAGE
Pearson	28.9403%	29.6146%	28.1888%
Euclidean	28.6705%	28.6705%	28.3237%
Cosine	28.9981%	27.2640%	27.2062%

K = 10	GCN	GAT	GraphSAGE
Pearson	28.9595%	28.7476%	29.0944%
Euclidean	28.8632%	28.0154%	28.9403%
Cosine	28.1696%	27.3218%	28.4008%

Table 1. Accuracy of each model with different distance metrics and different edge formulations.

After testing our models, we found out that GCN with Euclidean distance performed the best when K = 3; GAT with Pearson correlation coefficients performed the best when K = 6; and GraphSAGE with Pearson correlation coefficients performed the best when K = 10.

III.2. Data Preprocessing

As we mentioned before, we figured out that the accuracy with raw data is higher than the accuracy with the other three preprocessed data for the data preprocessing part. Table 2. shows the detailed accuracy of each data preprocessing method.

	raw data	FFT	DWT	PCA
GCN	31.21%	24.66%	20.81%	26.57%
GAT	30.44%	20.32%	18.50%	28.03%

Table 2. Accuracy of different data preprocessing methods

III.3. Pooling

As Table 3. shows, although we try different pooling methods, aiming to improve the performance, the accuracy of the model gets even worse. Therefore, we stop finding other pooling methods because we don't expect they will work well.

Model: GCN with Euclidean edge formulation			
	Accuracy		
Pooling Method	K = 3	K = 6	K = 10
Mean & Max pool	29.27%	29.35%	29.06%
Set2Set	26.61%	27.21%	27.01%
SAGpool	27.69%	26.92%	27.98%

Table 3. Accuracy of different pooling methods

IV. Discussion/Conclusion

We divided this section into two parts: Other Models and 2-Class Prediction.

IV.1. Other Models

We also tried other methods on this problem, such as LDA, Gaussian Naïve Bayes Classifier, Random Forest, SVM with One-vs-One strategy, Full connected network and CNN. Here are the results.

LDA	GNB	Random Forest	SVM	FNN	CNN
6.9207%	8.1505%	9.2597%	11.1406%	16.7630%	18.4971%

Table 4. Accuracy of other models

IV.2. 2-Class Prediction

To make the performance look better and to check whether our model can classify the dataset correctly, we merged the original six categories into two: animate class and inanimate class, and then we used the GCN model to test.

2-Class classification	
Model	Accuracy
GCN	68.59%

Table 5. Accuracy of merging six categories into two

According to the result, we can say this dataset can be roughly classified by GNN.

V. Data and Code Availability

You can access the following link to check out the data and code of our project.

Repository Link:
https://github.com/KevinKai02/Group3_Final/tree/main

VI. Author Contribution Statements

Tsai Hao-Yu (17%): Data preprocessing, Researching, Testing.
Chang Chia-Hsin (15%): Final presentation writing, Final presentation speech.
Chou Kuang-Chi (17%): Model building, Researching, Testing.
Wang Wei-Hsiang (17%): Model building, Researching, Testing.
Chuang Kai-Hsiang (17%): Repository, Report writing.
Chen Yen-Cheng (17%): Model building, Researching, Testing.

VII. Reference

[1] I. Omerhodzic, S. Avdakovic, A. Nuhanovic, K. Dizdarevic. "Energy Distribution of EEG Signals: EEG Signal Wavelet-Neural Network Classifier."

[2] Pari Jahankhani, Vassilis Kodogiannis, Kenneth Revett. "EEG Signal Classification Using Wavelet Feature Extraction and Neural Networks." IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing. IEEE, 2006.

[3] Demir, Andac, et al. "EEG-GNN: Graph neural networks for classification of electroencephalogram (EEG) signals." 2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC). IEEE, 2021.

[4] Mehmet Akin. "Comparison of Wavelet Transform and FFT Methods in the Analysis of EEG Signals." Journal of Medical Systems.

[5] Andac Demir, Toshiaki Koike-Akino, Ye Wang, Masaki Haruna, Deniz Erdogmus. "EEG-GNN: Graph Neural Networks for Classification of Electroencephalogram (EEG) Signals."

[6] Thomas N. Kipf, Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." ICLR, 2017.