MSc. DSAI

Predicting age from brain networks using Graph Neural Networks

Research Project report

November 2022

par

Ufuk Cem Birbiri

Academic tutor: Mathieu Desroches

Table des matières

1 Introduction

There are many ways to define the data such as images, text, vectors, or simply graphs. Graphs are everywhere ; the objects are often represented by their connections to objects. Social networks, biological molecules, mathematical equations, brain networks, images, or even texts can be represented as a graph. Researchers developed graph neural networks(GNN) to use graph datasets in neural networks over a decade ago[1]. There are many applications that one can do using GNN.

1.1 Applications of GNNs

1. Node Classification :
The data points in the graph are called nodes. Node classification is a problem in which we determine the labels of nodes. It is usually a semi-supervised method where only some part of the graph is labeled.

2. Graph visualization :
This application consists of the intersection of mathematics and computer science by using graph theory as well as information visualization. The main goal is to extract the graph visualization that shows the structures of the graph to help users to better analyze the graph.

3. Edge Prediction :
The nodes in the graph are connected by edges. Here, the goal is to predict whether there are possible linkages between nodes. As an example, it is used in social networks on Facebook to suggest potential friends to users[2]. Also, shopping websites use this approach in recommender systems or it is used in the prediction of criminal relations.

4. Graph Classification :
The goal is to predict the class of the graph itself. Graph classification is used in drug discovery as chemical molecules that can be classified as toxic or non-toxic[3] or in text classification[4].

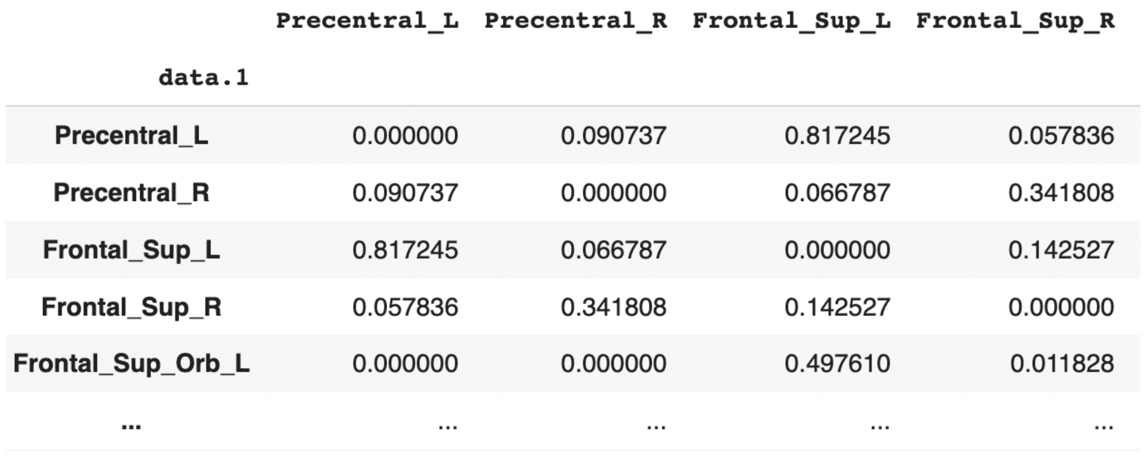
5. Community Detection :
In this application, the nodes in a graph are partitioned into communities according to their similarities[5]. Finding different subgroups in a social network can be an example.

5. Anomaly Detection :
Anomalies in graph data are called outliers nodes. In this task, GNN is used to find outlier nodes in an unsupervised method.

In this work, I applied graph classification methods to predict the age of patients by using brain networks.

2 Dataset

The data that I used in this project is the Human Connectome dataset [6]. The dataset includes 998 patients with various features such as diffuse, on tensor image(DTI) connectivity matrices of the brain, fMRI connectivity matrices of the brain, personal information (family history, personality, age, gender, etc.), cognitive behavior, motor skills, emotion, sensory or brain features(surface area, thickness, gray matter volume,..)
The pairwise connections of the 116 brain regions have been measured using DTI connectivity matrices for each patient whose shape is (116, 116). Each row and column is a brain region. An example of a DTI connectivity matrix is shown in Fig 1. The type of connectivity matrices is Pandas data frame.



	Precentral_L	Precentral_R	Frontal_Sup_L	Frontal_Sup_R
data.1				
Precentral_L	0.000000	0.090737	0.817245	0.057836
Precentral_R	0.090737	0.000000	0.066787	0.341808
Frontal_Sup_L	0.817245	0.066787	0.000000	0.142527
Frontal_Sup_R	0.057836	0.341808	0.142527	0.000000
Frontal_Sup_Orb_L	0.000000	0.000000	0.497610	0.011828
...	...	...	...	...

FIGURE 1 – *An example of a DTI connectivity matrix of one patient. The shape of the matrix is (116, 116).*

In addition, there is the age information of each patient in the dataset. There are four categories for age : [22-25, 26-30, 31-35, 36+]. These four categories are converted to categorical variables to be used as class labels in the graph neural network.

3 Methodology

Firstly, the connectivity matrices are converted to numpy arrays using to_numpy() function[7]. Then, they are converted to networkx graphs[8] using from_numpy_matrix()[9]

function. To use the networkx graphs in Pytorch graph neural networks we need to convert them into torch.geometric Data type[10]. Before diving into the explanations, let me introduce the torch.geometric Data type because one by one we need to convert our networkx features into this data type.

Torch.geometric Data type

This data object describes a homogeneous graph. The data object can have node-level, edge-level, and graph-level attributes. This data type mimics the behavior of a regular Python dictionary. Also, it provides basic PyTorch tensor functionalities as well as useful functions for analyzing graph structures. The parameters of this data object are explained in the following list.

— **x** : Torch tensor type. It includes the node attribute, and it is a shape of $[num\_nodes, num\_node\_features]$. In my network, it corresponds to eigenvector centrality.

— **edge_index** : LongTensor type. It includes the edge indexes in COO format with the shape $[2, num\_edges]$

— **edge_attr** : Tensor type. It is the edge attribute matrix with shape $[num\_edges, num\_edge\_features]$. In my model, it corresponds to connection values between brain regions in the DTI connectivity matrix

— **y** : Tensor type. Class labels of the graphs with the shape $[num\_nodes, 1]$. It corresponds to categorical age labels.

Before converting our networkx graph into torch.geometric Data type, we need to calculate the eigenvector centrality for each node, using eigenvector_centrality() function[11] in networkx library. Eigenvector centrality values will be assigned to each node as a node attribute i.e they will be the **x parameter** in the torch.geometric Data type.

1. How to compute the eigenvector centrality for the parameter x ?

Eigenvector centrality calculates the centrality value for a node based on the centrality of its neighbors. The eigenvector centrality for node i is the $i-th$ entry of the vector characterized by the equation

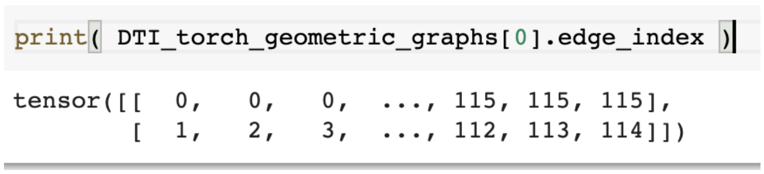
$$Ax = \lambda x$$

where A is the adjacency matrix of the graph G with eigenvalue λ . By the advantage of the $Perron\breve{}Frobenius$ theorem[12], there exists a unique solution x. If $lambda$ is the

largest eigenvalue of the adjacency matrix A, we can say that the unique solution x has positive values in all of the entries[13].
As a result, I calculated the eigenvector centralities of all the nodes in the graph G and assign them as a node attribute.

2. How to compute edge_index ?

The edge index matrix represents the edges in the graph. Its shape is [2, num_edges] and its shape varies for each graph since each graph has a different number of edges. The type of this variable is a LongTensor.There is a function in torch.geometric library that computes the edge index directly, called $from\_networkx(G)$. It involves tuples that have edge numbers. An example is shown in Fig2. If there is an edge between two nodes, then the node indexes are saved in this matrix. For example in Fig2, there is a between node 0 and node 1, so they are located in the edge_index matrix.



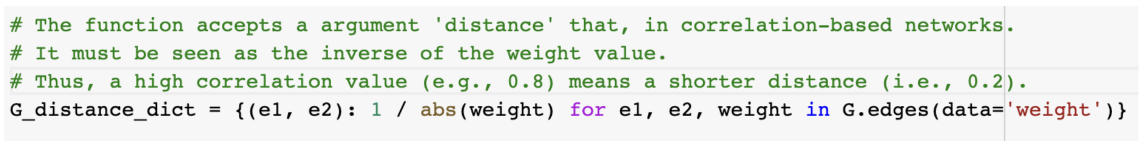
```
print( DTI_torch_geometric_graphs[0].edge_index )

tensor([[  0,   0,   0,  ..., 115, 115, 115],
        [  1,   2,   3,  ..., 112, 113, 114]])
```

FIGURE 2 – *Example of the edge_index of a torch.geometric graph*

3. How to compute edge_attr ?

The edge attribute represents the edge features in the graph. Basically, it is the value in the DTI connectivity matrix between brain regions. Its shape is $[num\_edges, num\_edge\_features]$, in my case, its shape is $[num\_edges, 1]$ since we have one edge feature. I use the G.edges() function in the networkx library to retrieve the edge attribute. While computing the edge attributes we consider the edge feature as $1/abs(edge\_attr)$, because the edge attribute should be seen as the inverse of the edge_feature. Thus a high correlation value (e.g., 0.8) means a shorter distance (i.e., 0.2). Computing the edge_attr can be seen in Fig4



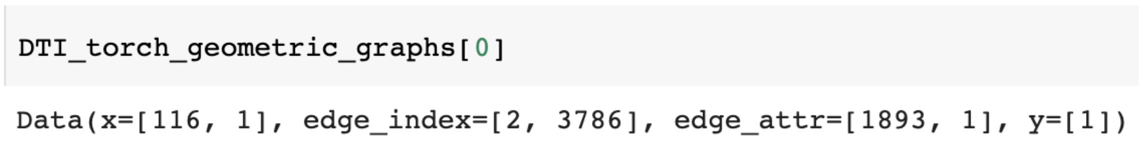
```
# The function accepts a argument 'distance' that, in correlation-based networks.
# It must be seen as the inverse of the weight value.
# Thus, a high correlation value (e.g., 0.8) means a shorter distance (i.e., 0.2).
G_distance_dict = {(e1, e2): 1 / abs(weight) for e1, e2, weight in G.edges(data='weight')}
```

FIGURE 3 – *Example of how to calculate the edge_attr of a torch.geometric graph*

4. How to compute the class label y ?

The class labels are the age of the patients in the categorical variable format. We assign each patient's age to their y value.

After computing all these parameters and converting our 998 networkx graphs, the torch.geometric Data type looks like in the Fig4



```
DTI_torch_geometric_graphs[0]

Data(x=[116, 1], edge_index=[2, 3786], edge_attr=[1893, 1], y=[1])
```

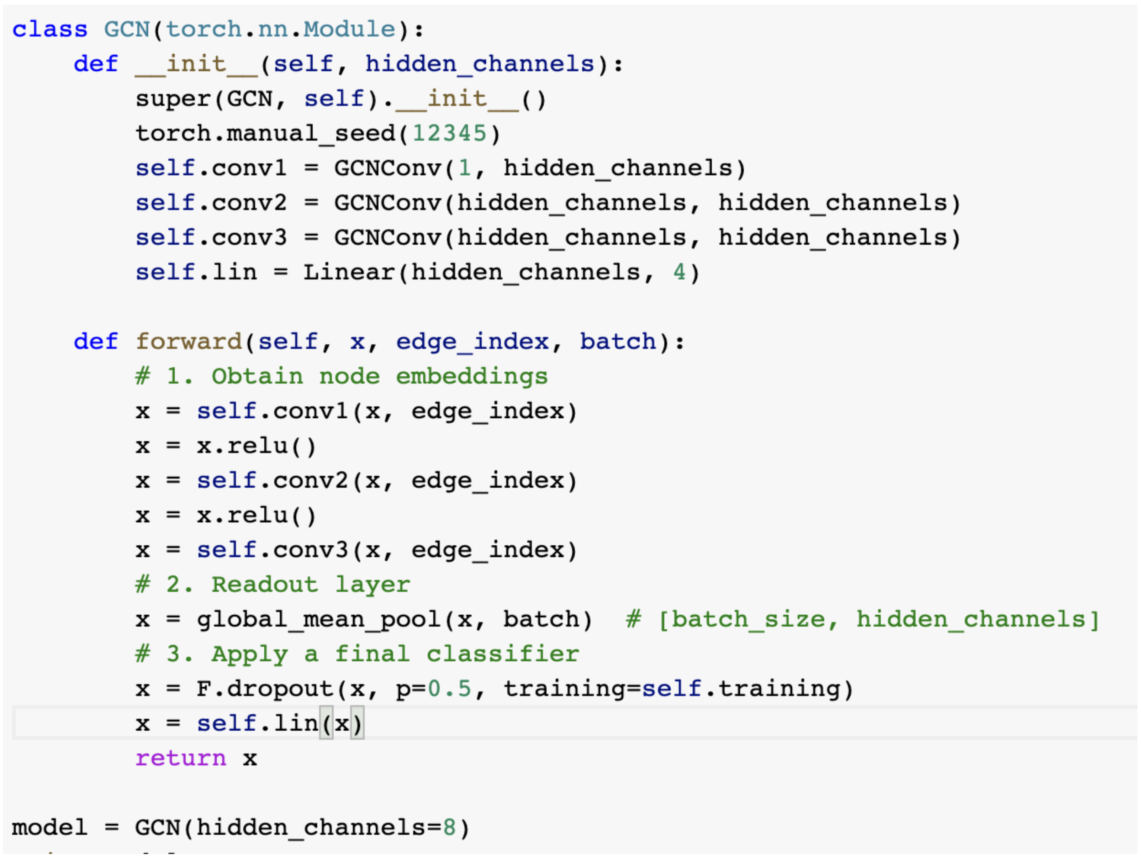
FIGURE 4 – *An example of the final version of the torch.geometric Data graph.*

4 Model selection and Results

The goal of this project is to predict the patient's age using DTI connectivity matrices by converting them to graph representations. Then the problem becomes a graph classification problem. Pytorch geometric library provides potential models for this task. I adapted a pre-implemented network that uses the graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper[**?**]. In this network, three graph convolutional layers(GCNConv[14]) were used followed by a linear layer. Relu activation function is added as an activation function. To overcome the over-fitting, dropout is used before the last linear layer. In addition, Pytorch geometric provides a pooling layer called *global_mean_pool* located before the dropout which returns batch-wise and graph-level outputs by taking the average of node features across the node dimensions. The GNN network is shown in Fig5

The dataset was split into training(70%) and validation(30%). For the training phase, the Adam optimizer is used as an optimizer with a learning rate of 0.0001 and batch size 64. Since it is a multi-class classification problem, the cross-entropy loss function was used with the total number of epochs as 20.

When I train the network I always have the same result. The training accuracy was always 0.4898 and the validation accuracy was 0.3307. I assume that the gradient calculation is stuck in local minima and cannot find the global minima. The possible reasons for this problem could be the wrong optimizer, learning rate, or wrong model selection. There could be more suitable GNN networks that work better with my datasets. Recently I am doing a literature search and trying other models for GNN graph classification.



```
class GCN(torch.nn.Module):
    def __init__(self, hidden_channels):
        super(GCN, self).__init__()
        torch.manual_seed(12345)
        self.conv1 = GCNConv(1, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.conv3 = GCNConv(hidden_channels, hidden_channels)
        self.lin = Linear(hidden_channels, 4)

    def forward(self, x, edge_index, batch):
        # 1. Obtain node embeddings
        x = self.conv1(x, edge_index)
        x = x.relu()
        x = self.conv2(x, edge_index)
        x = x.relu()
        x = self.conv3(x, edge_index)
        # 2. Readout layer
        x = global_mean_pool(x, batch)  # [batch_size, hidden_channels]
        # 3. Apply a final classifier
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.lin(x)
        return x

model = GCN(hidden_channels=8)
```

Figure 5 – *The GNN for graph classification.*

Bibliographie

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1) :61–80, 2008.

[2] Abhay Singh, Qian Huang, Sijia Linda Huang, Omkar Bhalerao, Horace He, Ser-Nam Lim, and Austin R Benson. Edge proposal sets for link prediction. *arXiv preprint arXiv :2106.15810*, 2021.

[3] Oliver Wieder, Stefan Kohlbacher, Mélaine Kuenemann, Arthur Garon, Pierre Ducrot, Thomas Seidel, and Thierry Langer. A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today : Technologies*, 37 :1–12, 2020.

[4] Lianzhe Huang, Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. Text level graph neural network for text classification. *arXiv preprint arXiv :1910.02356*, 2019.

[5] Oleksandr Shchur and Stephan Günnemann. Overlapping community detection with graph neural networks. *arXiv preprint arXiv :1909.12201*, 2019.

[6]

[7] to numpy, `https://pandas.pydata.org/pandas-docs/pandas.DataFrame.to_numpy.html`.

[8] Networkx graph, `https://networkx.org/documentation/stable/tutorial.html`.

[9] from numpy matrix, `https://networkx.org/documentation/from_numpy_matrix.html`.

[10] torch.geometric Data, `https://pytorch-geometric/torch_geometric.data.Data`.

[11] eigenvector centrality, `https://networkx.org/documentation/eigenvector_centrality`.

[12] Phillip Bonacich. Power and centrality : A family of measures. *American journal of sociology*, 92(5) :1170–1182, 1987.

[13] Markus Brede. Networks—an introduction. mark ej newman.(2010, oxford university press.) 65.38,£ 35.96 (hardcover), 772 pages. isbn-978-0-19-920665-0., 2012.

[14]