The University of New South Wales - COMP9312 - 22T2 - **Data Analytics for Graphs**

Project Homepage / Q2

Q2 (10 marks)

Graph Convolutional Network.

1: Probelm Statement:

In this question, we will introduce the PyTorch Geometric(PyG) and use the PyG to build the graph convolutional network for node classification. Also, we will explore implementing the GCN layer by ourselves and by using other message-passing layers.

2: Doing this project

Open the code template file Q2.ipynb and make a copy of the file in your own google drive. You need to implement

a. Basic attributes of Cora (3 points).
b. Implement the forward function in Graph Convolutional Network (3 points).
c. Implement the message passing in GCN by ourselves (4 points).
d. (optional) Another two message passing mechanism.

3: Basic of PyG:

PyTorch Geometric is an extension library for PyTorch. It provides useful primitives to develop Graph Deep Learning models, including various graph neural network layers and a large number of benchmark datasets.

PyG uses the Class `Data` to store the graph structure.

PyG also provides many datasets. Here, we use Cora which is often used for node classfication. It contains one graph, and the node in the graph belongs to 7 classes. Use the following command to download the dataset.

```
dataset = Planetoid(root='/tmp/Cora', name='Cora')
```

Note: dataset may contain many graphs. Cora just contains one graph. If we want to retrieve the graph, we can use:

```
Data = dataset[0]
```

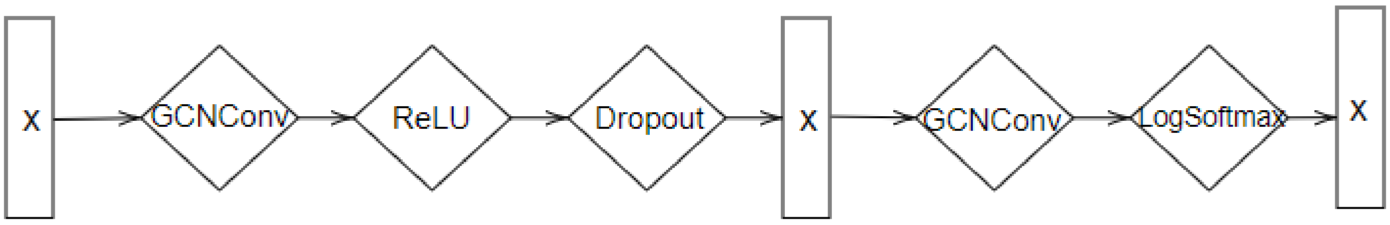
Now you need to output the following basic attributes of the graph:

a. the number of graph in the dataset.
b. the number of node of first graph in Cora.
c. the number of edges of first graph in Co.
d. the dimension of feature.
e. the number of classes.
f. check if there are isolated nodes.

4: Graph Convolutional Network By PyG

Here, we use a two-layer GCN model. The GCN layer is implemented by GCNConv which is provided by PyG.

You need to follow the figure below to implement the forward function.



The network structure.

```
class GCN(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.conv1 = GCNConv(in_channels, 16)
        self.conv2 = GCNConv(16, out_channels)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index

        ############# Your code here ############
        ## Note:
        ## 1. Construct the network as shown in the figure
        ## 2. torch.nn.functional.relu and torch.nn.functional.dropout are useful
        ## For more information please refer to the documentation:
        ## https://pytorch.org/docs/stable/nn.functional.html
        ## 3. Don't forget to set F.dropout training to self.training
        ## (~5 lines of code)
        #########################################
```

5: Graph Convolutional Networks By ourselves

In lectures, we have learnt about the matrix formulation of GCN. In last question, we have integrated the GCNConv module provided by PyG to train a model.

In this part, we aim to implement the message passing algorithm in GCN by ourselves.

There are many variants of GCN formulation. Here, we use the matrix formulation of GCN as following:
$X^{k+1} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}X^k\theta)$

Where X is the feature matrix, θ is the model parameters, $I \in R^{n*n}$ is the identity matrix, $D \in R^{n*n}$ is the degree matrix, n is the number of nodes in the graph, $A \in R^{n*n}$ is the adjancent matrix.

```
class Encoder(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels):
        super(Encoder, self).__init__()
        self.in_channels = in_channels
        self.hidden_channels = hidden_channels
        self.linear1 = torch.nn.Linear(in_channels, hidden_channels)

    def forward(self, x, edge_index):
        ############# Your code here ############
            ## Note:
            ## 1. Construct adjacnet matrix A of the graph
            ## 2. Calculte the degree matrix D
            ## 3. Calculte A_hat
            ## 4. Calculte the noramlized A_hat
            ## 5. multiply noramlized A_hat with x and theta
            ## np.diag, np.power, np.eye are useful
            ## (~11 lines of code)
        #########################################


        return x
```

For more details about the code, please refer to the part 7 in the template.

6: (Optional) Use another two methods of the message passing layer in the above GNN model.

We've defined a GNN class that uses GCNConvolutional layers, suggest and test th accuracy of two other variations of this GNN class that uses a different message passing mechanism by replacing the above GCNConv layer.

You can use the function provided by PyG or implemented by yourself.

7: Notes:

a. Test your code in the Colab if you do not installed the library on your own machine.
b. Up to 2 points for the optional question and up to 10 points for the whole Q2.
c. Include a **short description** about the message pass layer you choose and their **accuracy** in Q2.ipynb or in a separated Q2.pdf.
d. Required Files: Compress all Q2 related files (Q2.ipynb and an optional Q2.pdf if you want create a documet in a seperated file) into Q2.zip. Together with Q1.zip, submit them on Moodle.

END OF QUESTION