Social Network Computing

Section 9 Code Explanation

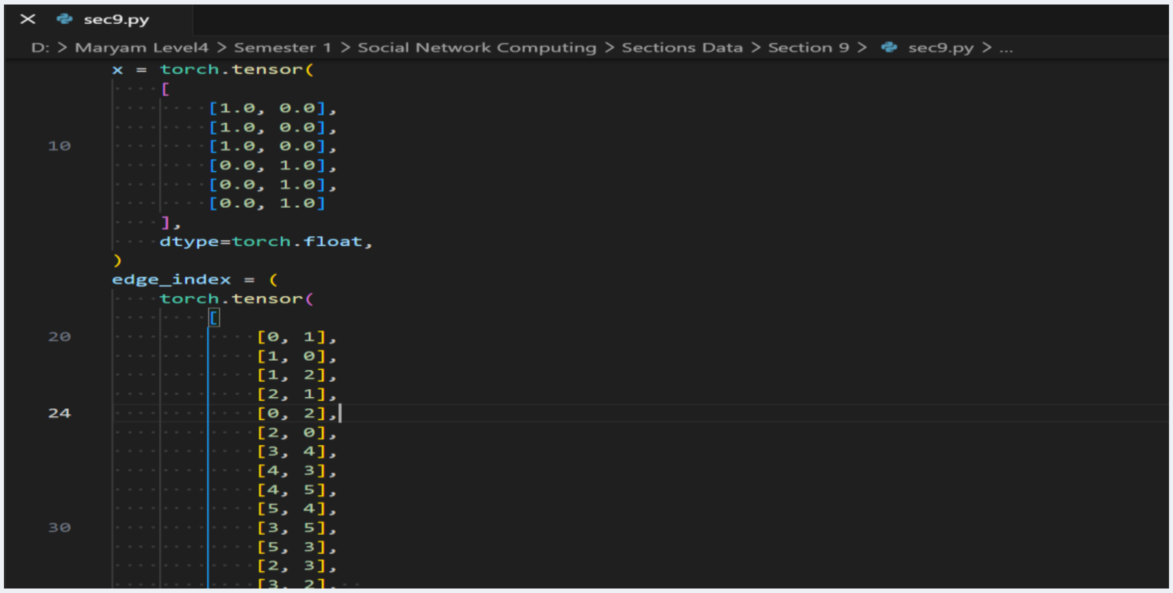
Name : Maryam Waheed Zamel

ID : 2205154

Introduction

This code trains GraphSAGE Neural Network using pyTorch geometric to classify nodes in small graph as benign (has label 0) and malicious (has label 1)

Code Explanation

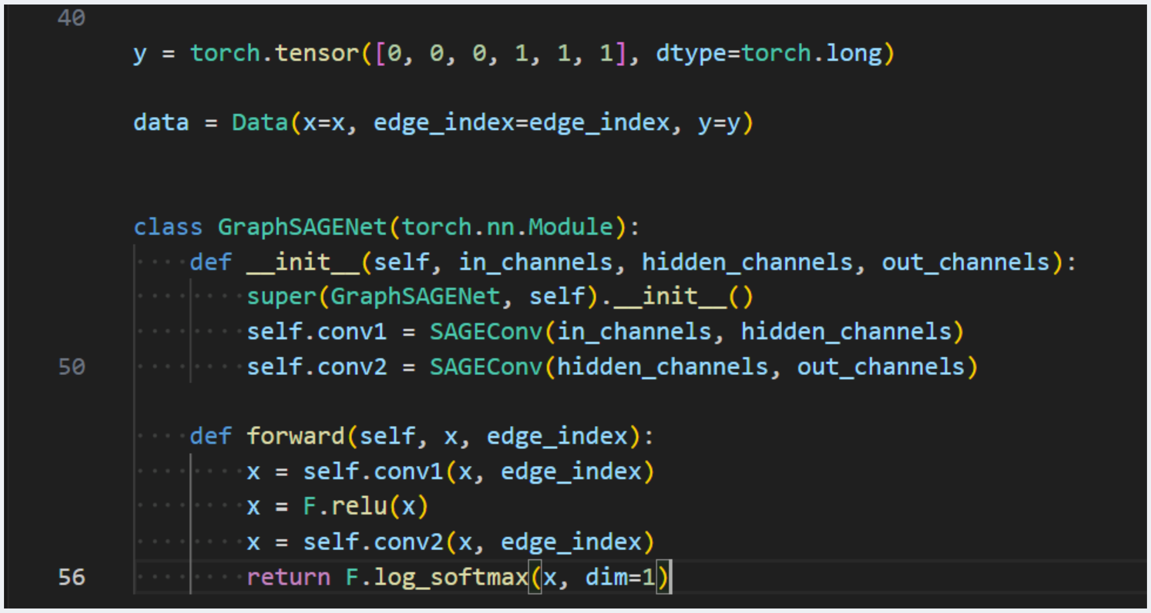

sec9.py

D: > Maryam Level4 > Semester 1 > Social Network Computing > Sections Data > Section 9 > sec9.py > ...

```python
x = torch.tensor(
    [
        [1.0, 0.0],
        [1.0, 0.0],
        [1.0, 0.0],
        [0.0, 1.0],
        [0.0, 1.0],
        [0.0, 1.0]
    ],
    dtype=torch.float,
)
edge_index = (
    torch.tensor(
        [
            [0, 1],
            [1, 0],
            [1, 2],
            [2, 1],
            [0, 2],
            [2, 0],
            [3, 4],
            [4, 3],
            [4, 5],
            [5, 4],
            [3, 5],
            [5, 3],
            [2, 3],
            [3, 2],
```

- Create small graph consists of 6 nodes each node has 2 features which benign node is [1,0] and malicious node is [0,1].

- This graph will be undirected graph as edges are bidirectional there are edge from each node to another so this graph is also connected graph



```python
y = torch.tensor([0, 0, 0, 1, 1, 1], dtype=torch.long)

data = Data(x=x, edge_index=edge_index, y=y)


class GraphSAGENet(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GraphSAGENet, self).__init__()
        self.conv1 = SAGEConv(in_channels, hidden_channels)
        self.conv2 = SAGEConv(hidden_channels, out_channels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

- Here variable “y” represents labels as I defined before benign node has label 0 and malicious node has label 1

- Data variable contain entire graph (nodes and edges)

- Define class of GraphSAGENet that inherits torch.nn.Module which is the base class for all neural network modules in PyTorch

- Define constructor and 3 layers of neural network (input , hidden and output layer) and calls parent class of torch.nn.Module using super keyword

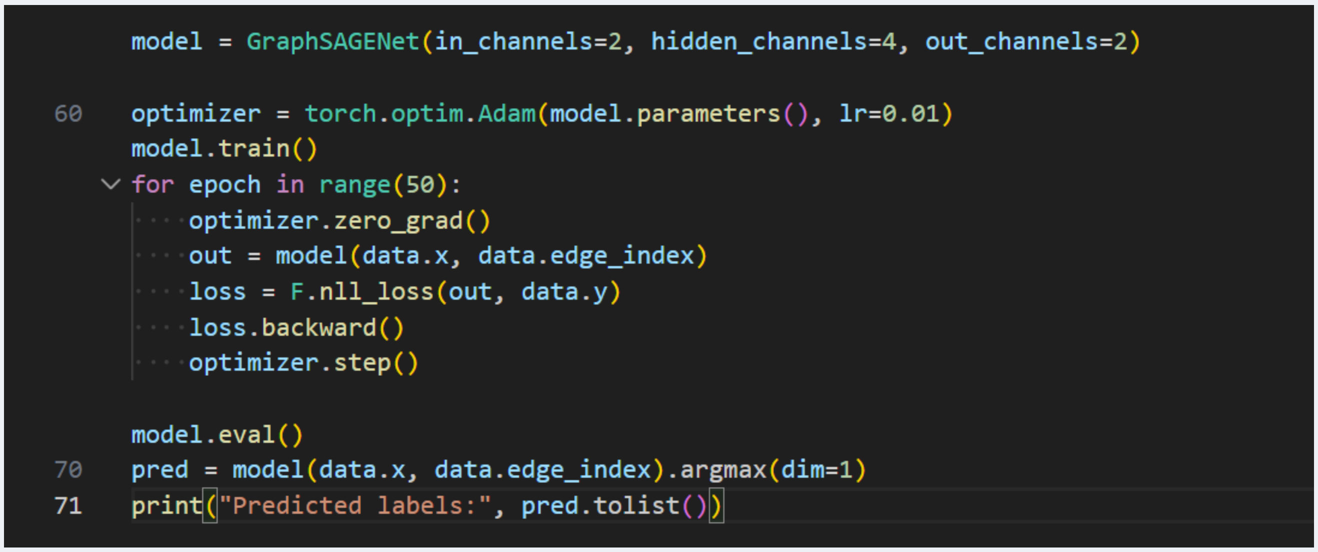
- Define 2 layers GraphSAGE model

  For the 1st layer takes in-channel as input and send output to hidden layer

  For the 2nd layer takes output of the hidden layer as input and introduce final output

- Define forward function which means that flow starting from input layer until reaching to the output layer

    1. Applies the first GraphSAGE layer to the input features x using the provided edges
    2. Applies the ReLU activation function which replaces negative values with zero. it introduces nonlinearity so the network can learn complex mappings.
    3. Applies the second GraphSAGE layer to the activated hidden features. This aggregates neighbor information again now over the hidden features and maps to out channels
    4. Applies log_softmax across the class dimension and returns the result which converts logits to log-probabilities for each class per node.



```
model = GraphSAGENet(in_channels=2, hidden_channels=4, out_channels=2)

optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
model.train()
for epoch in range(50):
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = F.nll_loss(out, data.y)
    loss.backward()
    optimizer.step()

model.eval()
pred = model(data.x, data.edge_index).argmax(dim=1)
print("Predicted labels:", pred.tolist())
```

- After defining class of GraphSAGENet, create model new object from this class and define that
    1. In_channel =2 which means that each node has 2 features.
    2. Hidden_channel = 4 which means that the first SAGE layer will output a 4 dim
    3. Out_channel = 2 which means that the final prediction has 2 classes (benign, malicious).

- Using Adam optimizer which is mainly used to update weights during training and define learning rate = 0.01 this value is suitable to handle overfitting then start training

- Define number of epochs = 50 which means that 50 iterations will be trained and each iteration updates model weights
- Compute loss by comparing predicted log probabilities introduced by (out) with true labels and based on this comparison compute how much each weight contributed in loss then update model weights
- In the final step start evaluation of the model and printing predicting labels

Thank You 😊