Mohammed Yasser El.Sharkawey.
2205149.

Graph SAGE Node Classification Report

---

1. Installing Required Libraries

The first step of the code is to install the torch_geometric library, then import PyTorch and the SAGEConv layer. These libraries are necessary to build and train graph neural networks.

```
!pip install torch_geometric

import torch
from torch_geometric.data import Data
from torch_geometric.nn import SAGEConv
import torch.nn.functional as F
```

---

2. Setting the Node Features

We create a matrix called x, where each row represents a node, and each node has 2 features.

- **The first three nodes have low feature values, representing benign users.**
- **The last three nodes have high feature values, representing malicious users.**

```
x = torch.tensor(
    [
        [1.0, 0.0],  # Node 0 (benign)
        [1.0, 0.0],  # Node 1 (benign)
        [1.0, 0.0],  # Node 2 (benign)
        [0.0, 1.0],  # Node 3 (malicious)
        [0.0, 1.0],  # Node 4 (malicious)
        [0.0, 1.0]   # Node 5 (malicious)
    ],
    dtype=torch.float,
)
```

- **Number of nodes: 6**
- **Number of features per node: 2**

3. Defining the Graph Edges

We create an edge list called edge_index, which tells the model which nodes are connected in the graph.

Group Structure:

- **Nodes 0, 1, and 2 are fully connected together → benign group**
- **Nodes 3, 4, and 5 are fully connected together → malicious group**
- **There is one cross-edge connecting node 2 (benign) to node 3 (malicious)**

```
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],  # cross-edge between
benign node 2 and malicious node 3
        ],
        dtype=torch.long,
    )
    .t()
    .contiguous()
)
```

- Total edges: 14 (undirected)

---

4. Labels for Training

We assign labels for each node to indicate its class:

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

- First 3 nodes → label 0 (benign)

- Last 3 nodes → label 1 (malicious)

---

5. Building the GraphSAGE Model

We define a neural network model with two GraphSAGE layers.

```
class GraphSAGEModel(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = SAGEConv(2, 4)
        self.conv2 = SAGEConv(4, 2)

    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)
```

Layer Descriptions:

Layer 1

```
self.conv1 = SAGEConv(2, 4)
```

- Input: 2 features

- Output: 4 new features

Layer 2

```
self.conv2 = SAGEConv(4, 2)
```

- Input: 4 features
- Output: 2 values (one for each class: benign or malicious)

---

6. Training Setup

We instantiate the model and set up an optimizer.

```
model = GraphSAGEModel()
optimizer =
torch.optim.Adam(model.parameters(),
lr=0.01)
```

- Optimizer: Adam
- Learning rate: 0.01

The optimizer updates model parameters during training.

---

7. Training Loop

We train the model for 200 epochs.

```
for epoch in range(200):
    optimizer.zero_grad()
    out = model(x, edge_index)
    loss = F.nll_loss(out, y)
    loss.backward()
    optimizer.step()
```

Each iteration performs:

1. Reset gradients
2. Run the model
3. Compute loss
4. Backpropagate gradients
5. Update weights with Adam

This process adjusts model parameters to reduce classification error.

---

8. Final Prediction

After training, we compute the final predictions:

```
pred = out.argmax(dim=1)
print("Predictions:", pred)
```

- `argmax` selects the class with the highest probability for each node
- Output is a tensor of length 6, one prediction per node

Example output:

```
Predictions: tensor([0, 0, 0, 1, 1, 1])
```

Interpretation:

- **Nodes 0, 1, 2 → benign**
- **Nodes 3, 4, 5 → malicious**

The model successfully distinguishes between benign and malicious nodes.

Conclusion

This code demonstrates a simple example of node classification using GraphSAGE.
The model learns node representations based on both node features and graph structure, enabling it to classify them into benign or malicious categories.

Even with a small dataset, the model achieves perfect classification after training.
