Ranim gamal 2205186

This code implements a simple node classification task using the GraphSAGE (Graph Sample and Aggregate) algorithm on a small synthetic graph. The goal is to classify nodes as either "benign" (label 0) or "malicious" (label 1) based on their features and graph structure. The implementation uses PyTorch Geometric, a library for deep learning on graphs. The code defines a graph with 6 nodes, trains a two-layer GraphSAGE model for 50 epochs, and outputs the predicted labels after training.

The code is self-contained and runs in an environment with PyTorch and PyTorch Geometric installed (noting the initial !pip install torch_geometric command, which is typical for Jupyter notebooks or Colab).

Data Description

The graph is a small, undirected network with the following components:

- **Nodes (6 total)**:

Nodes 0, 1, 2: Benign users, each with feature vector [1.0, 0.0].

Nodes 3, 4, 5: Malicious users, each with feature vector [0.0, 1.0].

Features are 2-dimensional, where the first dimension indicates benign (1) and the second indicates malicious (1). This is a simplistic encoding for demonstration.

Edges:

Benign nodes (0-1-2) form a fully connected clique (3 edges each way, but listed as undirected pairs).

Malicious nodes (3-4-5) also form a fully connected clique.

One cross-edge connects benign node 2 to malicious node 3.

Total edges: 14 (7 undirected pairs, each represented as two directed edges for PyTorch Geometric's format).

The edge_index is a tensor of shape [2, 14], where each column is a directed edge (e.g., [source, target]).

Labels:

y = [0, 0, 0, 1, 1, 1]: Ground truth labels matching the benign/malicious split.

The data is packaged into a torch_geometric.data.Data object, which is the standard format for PyTorch Geometric.

Model Architecture

The model is a two-layer GraphSAGE network defined in the GraphSAGENet class:

Layer 1: SAGEConv(in_channels=2, hidden_channels=4) – Takes 2D node features and aggregates information from neighbors to produce 4D hidden embeddings.

Layer 2: SAGEConv(hidden_channels=4, out_channels=2) – Produces 2D output logits for the two classes (benign vs. malicious).

Forward Pass:

Applies the first convolution, followed by ReLU activation for non-linearity.

Applies the second convolution.

Outputs log-probabilities using F.log_softmax(dim=1) for classification.

Key Mechanism: GraphSAGE samples and aggregates neighbor features, allowing the model to learn node representations based on local graph structure. This is particularly useful for tasks like anomaly detection or user classification in social networks.

The model is instantiated with in_channels=2, hidden_channels=4, and out_channels=2.

Training Process

Optimizer: Adam with a learning rate of 0.01.

Loss Function: Negative Log-Likelihood (F.nll_loss), which is standard for multi-class classification with log-probabilities.

Training Loop:

Runs for 50 epochs.

In each epoch: Zero gradients, forward pass on the entire graph, compute loss against true labels, backpropagate, and update parameters.

Notes: This is a transductive setting (training on the full graph), common for small datasets. No train/validation split is used, so the model may overfit, but it's sufficient for demonstration.

Results and Evaluation

After training, the model is set to evaluation mode (model.eval()), and predictions are generated:

pred = model(data.x, data.edge_index).argmax(dim=1) computes the class with the highest probability for each node.

Example output (based on typical runs; actual results may vary slightly due to randomness): [0, 0, 0, 1, 1, 1].

This matches the ground truth perfectly, indicating the model learned to distinguish benign from malicious nodes based on features and connectivity.

The cross-edge (2-3) likely helps the model propagate information, allowing node 2 to influence node 3 and vice versa, reinforcing class separation.

In a real scenario, accuracy would be computed as (pred == data.y).sum().item() / len(data.y), which here would be 100%.

Analysis and Insights

- **Strengths**:

GraphSAGE effectively captures graph structure: Benign nodes aggregate similar features, while malicious nodes do the same, and the cross-edge introduces some mixing.

The model converges quickly on this small dataset, demonstrating the power of GNNs for node-level tasks.

PyTorch Geometric simplifies graph handling, making it accessible for prototyping.

Limitations and Potential Improvements:

Dataset Size: With only 6 nodes, this is a toy example. Real-world graphs (e.g., social networks) have thousands or millions of nodes, requiring scalable sampling (GraphSAGE's strength).

Overfitting: No regularization or validation; add dropout, early stopping, or a larger graph.

Features: Node features are hand-crafted and binary. In practice, use embeddings from text, behavior, or other data.

Evaluation: Add metrics like F1-score, especially for imbalanced classes. Test on unseen nodes.

Extensions: For malicious user detection, incorporate temporal edges, use GAT (attention-based) layers, or add unsupervised pre-training.

Runtime: Training is fast (<1 second), but for larger graphs, GPU acceleration is needed.

It works because The clique structures create homophilic communities (benign nodes connect to benign, malicious to malicious), which GNNs exploit. The cross-edge tests generalization.