Social Network GraphSAGE Report

Omar Ahmed Hemaid

2205213

---

1. Core Idea

- GraphSAGE learns how to **aggregate a node's features with the features of its neighbors**.
- It does not store a fixed embedding per node; it learns a **procedure** for generating embeddings.
- This makes it suitable for **large**, **dynamic**, and **evolving** graphs.

2. How GraphSAGE Operates

- Each layer samples a **fixed number of neighbors** instead of using the entire neighborhood.
- Neighbor features are combined using a **trainable aggregation function**.
- The node's own features are then **merged** with the aggregated neighbor representation.
- Stacking multiple layers allows the model to capture **multi-hop context**.
- The final output of the model is a **vector embedding** for each node.

3. Aggregator Types (Technical Characteristics)

Mean Aggregator

- Averages neighbor features.
- Simple, stable, and behaves similarly to classic GCN propagation.
- Best for graphs where neighbors are relatively homogeneous.

Pooling Aggregators

- Apply a small neural network to each neighbor before pooling.
- Capture nonlinear feature interactions.
- Useful when neighbors have diverse or high-dimensional attributes.

Sequence-Based Aggregators

- Use sequence models (e.g., LSTM) applied to neighbor sets.
- Can represent more complex neighborhood patterns.
- Higher capacity but more expensive and more prone to overfitting.

4. Inductive Capability

- Once trained, GraphSAGE can produce embeddings for **new nodes** without retraining.
- Only needs the new node's features and its sampled neighbors.
- This is essential in environments where data arrives continuously (social networks, threat feeds, IoT networks, etc.).

5. Computational Advantages

- Complexity depends on the **sample size**, not the total node degree.
- Scales well to graphs with millions of nodes.

- Memory usage remains controlled because the model does **not maintain embeddings for every node**.
- Parallelizable and suitable for distributed training setups.

6. Embedding Behavior

- Embeddings encode:
  - local structure
  - similarity of attributes
  - community membership
  - behavior patterns of connected nodes
- Nodes with similar structural and feature signatures move into similar regions of the embedding space.
- Nodes whose neighborhoods differ significantly produce embeddings that stand out.

7. Security-Relevant Characteristics

Bot / Fake Account Detection

- Bots typically show abnormal connectivity patterns (excessive following, low reciprocity, connection bursts).
- GraphSAGE highlights these patterns because neighbor features influence the final embedding.
- Clusters of bots tend to form **tight, high-similarity embedding regions**.

Misinformation and Spam Propagation

- Malicious content often spreads through identifiable network substructures.
- Embeddings capture the relationships between users and content items.
- Repeated interactions across the same group of accounts become visible.

Collusive Behavior (Fake Reviews, Engagement Rings)

- Collusion rings usually present dense subgraphs with repetitive behavior.
- Their embeddings often collapse together into a distinct pocket of the representation space.
- Useful for downstream classifiers or anomaly-scoring systems.

Account Takeover and Behavioral Drift

- Sudden changes in who a user interacts with or what they post shift the embedding significantly.
- Analysts can monitor embedding changes over time to flag suspicious accounts.

Unsupervised Anomaly Detection

- Even without malicious labels, embeddings can:
  - expose outliers
  - reveal suspicious subgraphs
  - identify unusual interaction patterns
- Common techniques: clustering, nearest-neighbor distances, density-based anomaly scoring.

Minimal Code Skeleton (PyTorch Geometric)

```python
class SAGENet(torch.nn.Module):
    def __init__(self, in_dim, hid_dim, out_dim):
        super().__init__()
        self.conv1 = SAGEConv(in_dim, hid_dim)
        self.conv2 = SAGEConv(hid_dim, out_dim)

    def forward(self, x, edge_index):
        h = self.conv1(x, edge_index)
        h = torch.relu(h)
        h = self.conv2(h, edge_index)
        return h
```

Output embeddings can then be used for:

- classification
- anomaly scoring
- clustering
- temporal drift analysis

8. Comparison with Standard GCN

Property	GCN	GraphSAGE
Neighborhood	Full (spectral)	Sampled (stochastic)
Generalization	Transductive	Inductive
Memory usage	(O(	E
Update rule	Laplacian-based	Learned aggregation
Large-scale performance	Limited	High
New node embedding	Requires retraining	Immediate

9. Typical Workflow in a Security Pipeline

- Build a graph of users, posts, accounts, devices, or transactions.
- Construct feature vectors (behavior metrics, textual embeddings, metadata).
- Train GraphSAGE on known benign/malicious examples or only benign behavior.
- Generate embeddings for every node.
- Feed embeddings into:
    - classification models
    - anomaly detectors
    - clustering systems
    - risk scoring engines
- Continuously re-embed new accounts or entities as they appear.

10. Key Takeaways

- GraphSAGE is designed for **inductive**, **scalable**, and **dynamic** graph representation learning.

- The sampling mechanism gives it a practical advantage in massive graphs.
- Embeddings naturally encode behavioral and structural signatures relevant to security.
- Malicious clusters, botnets, collusion groups, and anomalous accounts tend to be **separable** in the learned embedding space.
- Well-suited for threat detection systems that need to adapt to fast-changing data.