Enhancing Fraud Detection via GNNs with Synthetic Fraud Node Generation and Integrated Structural Features

Georgia Kapetadimitri

A thesis submitted to the faculty of
University of Macedonia
in partial fulfillment of the requirements for the degree of

Masters of Science

Supervisor: Dimitrios Hristu-Varsakelis

DEPARTMENT OF APPLIED INFORMATICS

UNIVERSITY OF MACEDONIA

Abstract

Graph Neural Networks are widely employed for node classification in attributed networks. When it comes to fraud detection, however, GNNs can perform poorly, because a node's features are typically computed based on its local neighborhood, and this allows fraudsters to "blend in" among legitimate users. In this thesis, GNNs and supervised contrastive learning are proposed for fraud detection on datasets where fraudsters may use intricate strategies to camouflage themselves within the network. We train our GNNs using novel structural features in addition to those typically used in similar studies. The proposed features are based on the empirical probability distributions of various graph structural attributes which are extracted from a given dataset. We also apply supervised contrastive learning, enhanced with synthetic samples for the minority class (i.e., the fraudsters). Under our approach, the classifying capability of the GNN (measured via F1-macro, AUC, Recall) is improved by boosting the representation power of the calculated embeddings that maximize the similarity between legitimate users while minimizing that between fraudsters and legitimate users. Numerical experiments on two real-world multi-relation graph datasets (Amazon and YelpChi) demonstrate the effectiveness of the proposed method, whose improvements over the state-of the-art were especially significant in the larger YelpChi dataset.

Keywords: graph neural network, fraud detection, contrastive learning

Table of Contents

List of Figures

List of Tables

1 Introduction

This thesis is concerned with the problem of fraud detection in attributed networks via graph neural networks (GNNs). Fraud detection is a critical task in various domains such as financial services, e-commerce, and social networks, where identifying fraudulent activities can prevent significant financial losses and protect user integrity. In the recent years, GNNs have received significant attention [1], [2], [3]. This surge in interest is due to the powerful ability of GNNs to model complex relationships and interactions within graph-structured data, making them suitable for a wide range of applications, including node classification, link prediction, and graph classification.

GNNs operate by iteratively aggregating and transforming information from a node's neighbors, effectively capturing the dependencies and patterns within the graph. By learning rich node representations that encode both local and global graph structures, GNNs can leverage the connectivity and attributes of the graph to improve predictive accuracy. As a type of neural network specifically designed for graph data, GNNs extend the capabilities of traditional neural networks to handle the intricacies of graph-based systems. They use layers of neural computations in order to reach nodes that are more distant neighbors than the direct ones thus progressively refining the understanding of each node's context.

Yet, "simple" GNNs are not by themselves sufficient to perform effective fraud detection in multi-attribute graphs because of various challenges inherent in this task. GNNs are based on the homophily assumption, meaning that all nodes in the graph are of the same class and have similar features. This assumption works well in scenarios where similar nodes are connected, such as in social networks where people with similar interests form communities. However, in a fraud detection setting there are at least two dissimilar classes of nodes, namely "legitimate" users vs. "fraudsters", the latter usually being a small fraction of the total nodes. In this context, fraudsters aim to exploit the network structure to their advantage, making their detection challenging. One may expect that these two classes exhibit different features and characteristics. Legitimate users typically exhibit consistent and predictable behavior patterns, while fraudsters often employ sophisticated techniques to mask their activities and mimic legitimate behavior. Moreover, fraudsters may deliberately alter

their characteristics in order to resemble those of the legitimate users. This behavior is called “camouflage" [4] and consists of two separate devious practices illustrated in Figure 1.1: i) feature camouflage; fraudsters may attempt to modify their features in order to “look like" regular users, and ii) structural inconsistency; fraudulent users tend to connect less with other fraudsters and more with legitimate ones in order to blend in. As a result, the neighborhood aggregation process that occurs during GNN training will tend to blend together the features of legitimate users with those of the fraudsters, resulting in a "fading away" of the dissimilarity of the fraudsters. This blending effect undermines the model’s ability to accurately classify nodes, as the distinct features of fraudsters become diluted with those of legitimate users. This is why directly using GNNs for fraud detection is not a very effective approach.

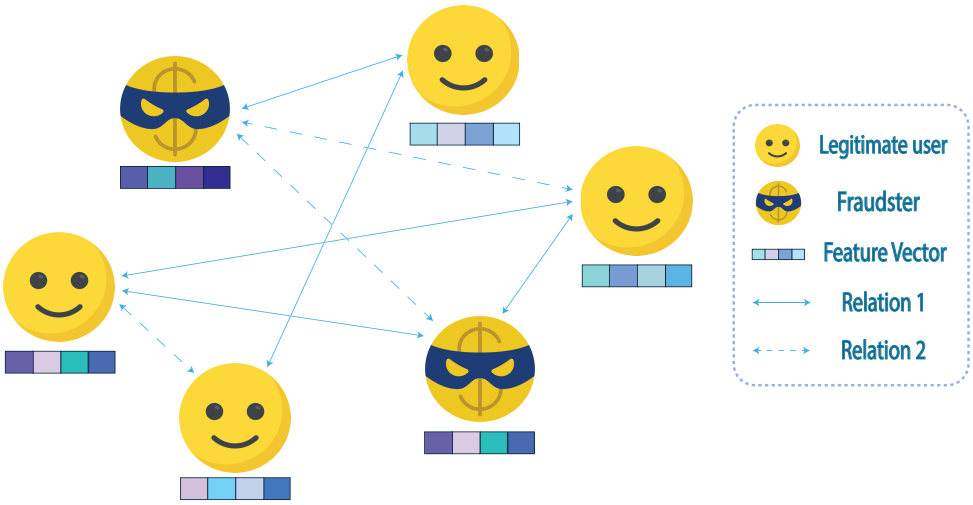

Figure 1.1: Toy example of a multi-relation graph that illustrates 2 techniques fraudsters use to disguise themselves. Relation 1 may correspond to having reviewed at least one product in common while relation 2 may correspond to having purchased the same product in a given time period. 1) Feature camouflage: fraudsters adjust their attributes to blend in with legitimate entities. An attribute could be the percentage of special characters appearing in the user’s review. The fraudster situated at the bottom of the image tries to imitate the features (seen by the resemblance of the colors in their feature vector) of the leftmost legitimate user with whom they are connected by relation 1. 2) Structural inconsistency: fraudsters connect more often with benign entities than with other fraudsters.

While GNN-based models have provided some solutions for effectively addressing the imbalanced nature of the problem, and have demonstrated very good results in fraud detection tasks, there are still challenges associated with their use. One of those challenges is to improve their ability to distinguish between fraudster and legitimate users. Because of camouflage, atomic features of a node can be unreliable when used in the GNN process. Furthermore, although previous methods have addressed the general imbalance in the dataset by employing methods such as a neighborhood

sampler [5], [6], [7] within the supervised contrastive learning process, the imbalance between the 2 classes remains, resulting in fewer contrastive pairs for the model to learn from and in contrastive embeddings that are not sufficiently representative of the nodes.

1.1 Contribution

The contribution of this thesis is to propose a novel approach that combines GNNs with i) supervised contrastive learning, ii) the introduction of additional graph-structural features, and iii) synthetic fraud nodes, to improve fraud detection in networks. To tackle the aforementioned difficulties, and provide more representative node features and more positive-negative samples in the supervised contrastive learning process, a new model, termed **B**oosted **R**epresentation with **I**ntegrated Structural Features and Synthetic Fraud G**E**neration (BRIE) is proposed in this thesis. The new structural features which we propose are used alongside the basic pre-existing features which are part of similar studies, and are selected specifically for mitigating the camouflage of fraudsters. These features do not describe a specific property of a node but instead consider the relations between the node and others within the entire graph. As we will see, this will lead to improved discrimination between fraudsters and legitimate users.

In order to address the imbalance between the two classes in the supervised contrastive learning module, the training dataset is enriched with synthetic fraud nodes; these are created based on the statistical properties of each class's node features, independent of the main dataset (i.e., without having to introduce new edges linking synthetic nodes to existing ones). This way, the quality of the embeddings produced within the module is improved, as the embeddings become more robust and representative of the nodes of the dataset, making them more effective in distinguishing between fraudsters and legitimate users. The proposed approach was tested on two real-world datasets, against some of the recent GNN-based fraud detection approaches, in terms of F1-score, AUC, and recall. The model showed significant improvements over the state-of-the-art, especially in the larger YelpChi dataset [8] where superior performance was attained, highlighting the effectiveness of the proposed solution.

The remainder of the thesis is organized as follows. Section 2 discusses related literature and introduces relevant definitions. In Section 3 the proposed model is presented in detail. Section 4 provides training details, and discusses the performance of the model.

2 Background

In this section we discuss the relevant literature and we define some important notation and the main problem.

2.1 Fraud detection

Traditional approaches to fraud detection have long been foundational in safeguarding against fraudulent activities. One of the most prominent methods is the use of rule-based systems. These systems rely on predefined rules and criteria established by domain experts to identify potential fraud. For instance, a rule might flag transactions exceeding a certain amount or those occurring outside of normal business hours. While rule-based systems are straightforward to implement and understand, their rigidity often means they struggle to adapt to new and evolving types of fraud. A paper that implements a rule-based expert system is mentioned later [9].

Another traditional approach is statistical analysis, which utilizes statistical techniques to detect anomalies or outliers in data. Methods like Z-score analysis, regression analysis, and Bayesian networks help identify unusual patterns that may indicate fraud. For example, in [10], Benford's Law is used to predict the frequency distribution of leading digits in numerical data to spot irregularities. The authors apply a multivariate approach, transforming the digit distribution into isometric log-ratio coordinates to perform a statistical test. This method is then used to detect anomalies in music streaming data. Statistical methods are effective for identifying simple anomalies but can be limited by the assumptions required for the models to function correctly.

Anomaly detection approaches focus on identifying deviations from normal behavior that could indicate fraud. Techniques such as clustering and nearest neighbor analysis are commonly used. For instance, clustering algorithms group similar transactions together and flag outliers that don't fit into any cluster. These methods are particularly effective for detecting unknown fraud types but can produce false positives if the definition of normal behavior is too broad. SODRNN (Sliding Window-based Outlier Detection using Reverse k-Nearest Neighbors) [11] performs credit card fraud detection using a data stream outlier detection algorithm called . This algorithm requires only one pass of the data stream, making it efficient for

real-time processing. Unlike traditional methods that need multiple scans of the database, SODRNN is well-suited for dynamic environments where data continuously flows. The algorithm identifies outliers by examining the relationship between data points and their reverse k-nearest neighbors within a sliding window framework.

Expert systems leverage a knowledge base of fraud indicators and rules derived from expert knowledge to evaluate transactions. These systems apply logical reasoning to compare transactions against the knowledge base. For example, an expert system might use specific rules related to known fraud schemes in credit card transactions. Although expert systems effectively utilize domain knowledge, they can be inflexible and challenging to scale as the volume of transactions increases.

Finally, behavioral analysis monitors user behavior to detect deviations that might indicate fraud. This approach analyzes patterns such as login times, transaction habits, and device usage. For example, an anomaly in login times or locations could signal unusual access patterns. Behavioral analysis can be highly effective in identifying fraud but requires continuous data collection and sophisticated analytics to differentiate between normal and fraudulent behaviors. Such a behavioral analysis system that incorporates machine learning is discussed later [12].

The approaches outlined above form the bedrock of fraud detection strategies, each with its strengths and limitations, and are often used in combination with newer, more adaptive methods to enhance overall detection capabilities. In [9] an existing rule-based expert system for fraud detection is enhanced by integrating Classification Based on Association Rule Mining (CARM). This involves identifying correlations between existing rules within the expert system and generating new rules based on these associations. The process begins by characterizing the existing rule-based system, which involves evaluating claims against a set of predefined rules, each with a specific weight. When the aggregated weight of triggered rules surpasses a certain threshold, the claim is flagged for further investigation. The integration of CARM allows the system to discover meaningful associations between rules and incorporate them as additional rules.

Data mining and machine learning techniques have also been employed to detect fraud by analyzing large datasets to uncover hidden patterns and correlations. Algorithms such as decision trees and support vector machines (SVMs) are trained on historical data to classify transactions as fraudulent or legitimate. A decision tree, for example, uses a series of decision rules derived from training data to classify transactions. These methods are more flexible and adaptive than rule-based systems but typically require substantial amounts of labeled data for training, which can be a limitation. In [12] a hybrid model is proposes that combines machine learning techniques with behavioral analytics. The model uses behavioral biometrics, such as keystroke dynamics, mouse movements,

and device usage patterns, to create unique user profiles. Machine learning algorithms such as SVMs, Random Forests and Gradient Boosting Machines (GBMs) then analyze these profiles to identify deviations from typical behavior, flagging potential fraudulent activities.

Unsupervised neural networks are also very efficient in fraud detection. They can detect new, previously unseen types of fraud and they do not require labeled training data, which is often scarce or unavailable. However, they are computationally demanding due to the complexity of training deep networks. Unsupervised neural networks, such as autoencoders, can detect fraud by learning the normal patterns of transaction data and identifying anomalies that deviate from these patterns. They compress input data into a lower-dimensional representation and then reconstruct it; significant reconstruction errors indicate potential fraudulent activities. Generative Adversarial Networks (GANs) consist of two neural networks: a generator and a discriminator. The generator creates data samples, while the discriminator evaluates their authenticity against real data. By training on non-fraudulent transactions, the generator learns to produce data that closely resembles normal transactions, and the discriminator becomes adept at distinguishing real from generated data. Some other unsupervised neural networks techniques are Restricted Boltzmann Machines (RBMs) and Self-Organizing Maps (SOMs). In [13] an adversarial deep denoising autoencoder is used for telecom fraud detection. The encoder functions as a discriminator by transforming input vectors into latent representations, which effectively handles noisy input, and employing Gaussian Mixture Models (GMMs) to estimate the probability of the transfer being genuine. The decoder acts as the generator, accepting the latent representation created by the discriminator and producing fake data to challenge the discriminator. The method leverages adversarial training, where the generator and discriminator engage in a minimax game to optimize their respective performances.

2.2 Graph-based Fraud detection

Standard graph-based fraud detection methods utilize various data mining and machine learning techniques and can be classified in 5 distinct categories: community-based, probabilistic-based, structural-based, compression-based and decomposition-based approaches [14].

Community-based techniques focus on identifying anomalous subgraphs within networks by analyzing the clustering and community structures. For example, GGSpam [15] detects review spammer groups in online product reviews. It models spammer groups as bi-connected graphs. Given a reviewer graph, GGSpam identifies all the bi-connected components whose spamicity scores exceed the given spam threshold. For large unsuspicious bi-connected graphs, the minimum cut algorithm is used to split the graph, and the smaller graphs are further processed

recursively. In [16] a two-step approach for identifying groups of opinion spammers and their targeted products is offered. First, it introduces the Network Footprint Score (NFS), a graph-based metric to evaluate the likelihood of products being targeted by spam campaigns based on statistical distortions in network characteristics. Second, it uses GroupStrainer to identify clusters of spammers within a 2-hop subgraph around the products identified by the NFS as likely targets of spam.

Probabilistic-based methods utilize probabilistic models to capture the uncertainty and relationships between nodes in a graph, enabling the detection of unusual patterns. In [17] the relations between providers (hospitals) and consumers (cities) are analyzed in order to find healthcare fraud committed by hospitals. The study assigns anomaly scores to each city based on the rate of procedures performed on its population compared to those of the closest (in terms of their features) cities. After analyzing the cities, the study transfers the anomaly scores from cities to hospitals based on the number of procedures each hospital performed in the population of each city. NetSpam [18] detects spam reviews by modeling review datasets as heterogeneous information networks. It utilizes the concept of metapaths to capture relationships between different features of reviews, such as user behavior and linguistic characteristics. The method calculates weights for each feature type to determine their importance in identifying spam, which enhances the accuracy of the spam detection process. These weights are used to compute a probability score for each review, ranking them based on their likelihood of being spam.

Structural-based approaches emphasize the overall network structure, including node connections and attributes, to identify anomalies based on deviations from expected patterns. For example, CATCHSYNC [19] detects fraudsters in social networks by identifying suspicious nodes exhibiting synchronized behavior and rare connectivity patterns in large directed graphs. To achieve this, CATCHSYNC creates feature spaces for target nodes based on characteristics like in-degree and then calculates synchronicity and normality values for source nodes, representing how closely their behavior aligns with each other and the norm in the graph. FRAUDAR [20] detects fraudsters in graphs by identifying unusually dense subgraphs, even when fraudsters use camouflage techniques. It calculates a suspiciousness metric based on the density of connections, distinguishing between normal and fraudulent behavior. FRAUDAR provides theoretical limits on the maximum density a group of fraudsters can achieve without being detected.

Compression-based methods aim to reduce the complexity of network data while preserving essential information for anomaly detection, often by compressing the graph representation. CoDetect [21] designed for financial fraud detection, leverages both graph matrix S that detects fraud activities and feature matrix F that traces fraud patterns simultaneously. It utilizes the latent company feature matrix U, learned from both S and F, which

represents the latent features of companies within the financial network and uncovers hidden relationships and similarities among companies.

Decomposition-based approaches break down complex networks into simpler components or subgraphs, allowing for the analysis of local anomalies and patterns within the network. HoloScope [22] integrates information from graph topology and spikes to detect fraudulent users and objects through its dynamic contrast suspiciousness metric. This metric emphasizes the contrast patterns between fraudsters and normal users in an unsupervised manner. Thus, HoloScope by using this metric that integrates information from graph topology, temporal bursts and drops and rating deviation, HoloScope can effectively identify suspicious behaviors that indicate fraudulent users. In DenseAlert [23], the data is represented as a multi-dimensional tensor, which can be decomposed into smaller parts (subtensors). Each subtensor represents a subset of the data, capturing interactions or relationships among specific dimensions. DenseAlert detects the sudden emergence of dense subtensors within a tensor stream, which indicates fraudulent or anomalous activities. It achieves this by maintaining and updating dense subtensors in real-time as new data arrives, instead of processing the entire dataset at once. This method is highly effective for identifying anomalies such as network attacks, bot activities, and rating manipulations by spotting dense subtensors that form rapidly within short time frames. DenseAlert leverages incremental updates to achieve faster processing times offering a practical solution for real-world anomaly detection scenarios.

2.3 Graph Neural Networks

Graph Neural Networks (GNNs) are a class of deep learning models designed for processing graph-structured data. Some of the earliest models that set the foundations for the current GNNs are proposed in [24], [25]. Unlike traditional neural networks that work well with grid-structured data, GNNs excel in handling non-Euclidean and irregularly structured information, where entities are interconnected in a graph. They are particularly useful for tasks involving modeling relationships between nodes, uncovering patterns in complex networks, and predicting node properties.

The key characteristic of a GNN is its utilization of neural message passing, where vector messages are transmitted between nodes and undergo updates via neural networks. In every iteration of message passing within a GNN, the hidden representation $h_u^{(k)}$ for each node $u \in V$ is adjusted based on the aggregated information from the graph neighborhood $N(u)$ of u. Figure 2.1 depicts graphically the process.
This message-passing update can be expressed as follows:

$$h_u^{(k+1)} = \text{UPDATE}\left(h_u^{(k)}, \text{AGGREGATE}(\{h_v^{(k)}, \forall v \in N(u)\})\right) \tag{2.1}$$

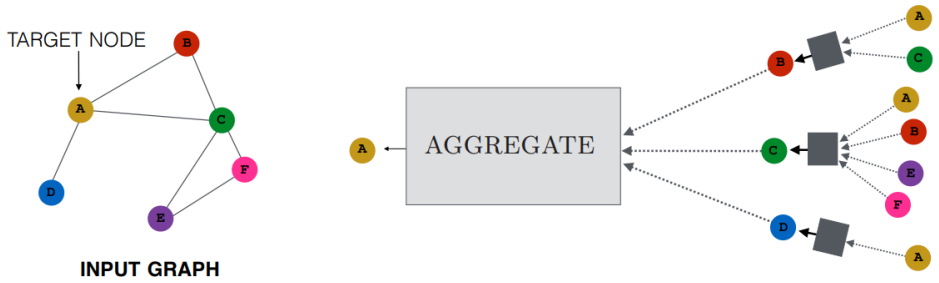

Figure 2.1: Overview of how a single node (A in this case) aggregates messages from its local neighborhood, source [26]

$$= \text{UPDATE}(h_u^{(k)}, m_{N(u)}^{(k)}) \tag{2.2}$$

where UPDATE and AGGREGATE are arbitrary differentiable functions (i.e., neural networks) and $m_{N(u)}$ is the "message" that is aggregated from u's graph neighborhood $N(u)$. At each iteration[1] k of the GNN, the AGGREGATE function takes as input the set of embeddings of the nodes in u's graph neighborhood $N(u)$ and generates a message $m_{N(u)}^{(k)}$ based on this aggregated neighborhood information. The update function, referred to as UPDATE, then combines the message $m_{N(u)}^{(k)}$ with the previous embedding $h_u^{(k-1)}$ of node u in order to produce the updated embedding $h_u^{(k)}$. The initial embeddings at $k = 0$ are set to the input features for every node, i.e., $h_u^{(0)} = x_u, \forall u \in V$. Once the GNN message passing has been run for K iterations, the output from the final layer can be used to define the embeddings for each node, i.e.,

[1] The different iterations of message passing are popularly known as the different "layers" of the GNN.

$$z_u = h_u^{(K)}, \forall u \in V \tag{2.3}$$

Since the AGGREGATE function takes a set as input, GNNs defined in this way are permutation equivariant by design. [26]

Formal definition of GNN

To make the abstract framework of GNNs implementable, specific definitions for the UPDATE and AGGREGATE functions must be provided. Starting from the basic GNN framework, the message passing is defined as,

$$h_u^{(k)} = \sigma\left(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_{\text{neigh}}^{(k)} \sum_{v \in N(u)} h_v^{(k-1)} + b^{(k)}\right) \tag{2.4}$$

where $W_{\text{self}}^{(k)}, W_{\text{neigh}}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$ are trainable parameter matrices, σ denotes a non-linear function (e.g., a tanh or ReLU) and $b^{(k)} \in \mathbb{R}^{d^{(k)}}$ is the bias.

The basic GNN can alternatively be defined through the UPDATE and AGGREGATE functions.

$$m_{N(u)} = \sum_{v \in N(u)} h_v \tag{2.5}$$

$$\text{UPDATE}(h_u, m_{N(u)}) = \sigma\left(W_{\text{self}} h_u + W_{\text{neigh}} m_{N(u)}\right) \tag{2.6}$$

where

$$m_{N(u)} = \text{AGGREGATE}\left(\{h_v, \forall v \in N(u)\}\right) \tag{2.7}$$

can be used as a generic formula to denote the message that has been aggregated from u's graph neighborhood.

Message Passing with Self-loops

A simplification to Equation 2.4 is to add self-loops thus omitting the UPDATE step, defining the message-passing as:

$$h_u^{(k)} = \text{AGGREGATE}\left(\{h_v^{(k-1)}, \forall v \in N(u) \cup \{u\}\}\right). \tag{2.8}$$

Simplifying the message passing in this manner can mitigate overfitting. However, it significantly constrains the expressive power of the GNN because it does not distinguish information originating from a node's neighbors and that from the node itself.

2.3.1 Aggregation varieties

The basic neighborhood aggregation operation (Equation 2.5) involves simply summing the embeddings of neighboring nodes. However, this approach presents a problem as it can be unstable and greatly influenced by node degrees.

One solution to this issue is to normalize the aggregation operation based on the degrees of the involved nodes. The most straightforward method is to compute an average rather than a sum,

$$m_{N(u)} = \frac{1}{|N(u)|} \sum_{v \in N(u)} h_v. \tag{2.9}$$

A very popular alternative is the symmetric normalization introduced by [27],

$$m_{N(u)} = \sum_{v \in N(u)} \frac{\sqrt{h_v}}{\sqrt{|N(u)||N(v)|}}. \tag{2.10}$$

Intuitively, symmetric normalization aggregation can be explained as, in a citation graph, which is the type of data analyzed by [27], data from nodes with very high degrees (such as papers cited numerous

times) might not be useful into inferring community membership within the graph. This limitation arises because highly cited papers can span diverse subfields, diminishing their relevance to community inference. However, symmetric normalization can also be derived from spectral graph theory. Specifically, the fusion of symmetric-normalized aggregation (Equation 2.10) and the basic GNN update function (Equation 2.6) yields a first-order approximation of a spectral graph convolution.

One of the most widely-used baseline models for graph neural networks, the Graph Convolutional Network (GCN) [27], adopts the symmetric-normalized aggregation along with the self-loop update approach. In the GCN model, the message passing function is defined as,

$$h_u^{(k)} = \sigma\left(W^{(k)} \sum_{v \in N(u) \cup \{u\}} \frac{h_v}{\sqrt{|N(u)||N(v)|}}\right) \tag{2.11}$$

Attentional GNNs

A commonly adopted technique for enhancing the aggregation layer in GNNs is to incorporate *attention* [28]. The core concept involves assigning attention weights or importance scores to each neighbor node. These weights determine the influence of each neighbor during the aggregation process. The pioneering GNN model to implement this attention mechanism was the Graph Attention Network (GAT) [29]. In GAT, attention weights are utilized to compute a weighted sum of the neighboring node features.

$$m_{N(u)} = \sum_{v \in N(u)} \alpha_{u,v} h_v \tag{2.12}$$

where $\alpha_{u,v}$ denotes the attention on neighbor $v \in N(u)$ when aggregating information at node u. In the GAT paper, the attention weights are defined as

$$\alpha_{u,v} = \frac{\exp\left(\mathbf{a}^\top [W h_u \oplus W h_v]\right)}{\sum_{v' \in N(u)} \exp\left(\mathbf{a}^\top [W h_u \oplus W h_{v'}]\right)} \tag{2.13}$$

where $\mathbf{a}$ is a trainable attention vector, W represents a trainable matrix, and $\oplus$ denotes the concatenation operation.

The GAT-style attention is recognized for its effectiveness with graph data. However, in principle, any standard attention model from the broader deep learning literature can be employed [28]. Common variants of attention include the bilinear attention model.

$$\alpha_{u,v} = \frac{\exp\left(h_u^\top \mathbf{W} h_v\right)}{\sum_{v' \in N(u)} \exp\left(h_u^\top \mathbf{W} h_{v'}\right)} \tag{2.14}$$

2.3.2 Update varieties

The AGGREGATE operator within GNN models has often been the primary focus of researchers, with a plethora of proposed novel architectures and

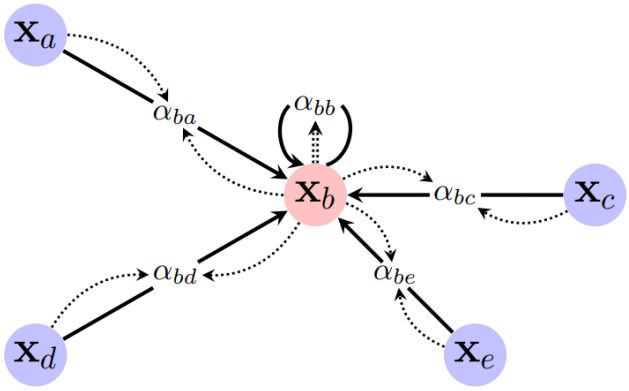

Figure 2.2: Message passing using an attentional GNN. With α are depicted the attentional weights.

variations. However, it's important to recognize that GNN message passing consists of two crucial steps: aggregation and updating. The UPDATE operator, while perhaps not as spotlighted, is equally significant in shaping the capabilities and inductive biases of the GNN model.

Over-smoothing presents a significant challenge in GNNs. This phenomenon occurs when node-specific details are diluted or lost over multiple iterations of message passing in GNNs. Essentially, over-smoothing arises when the information gathered from neighboring nodes during message passing begins to overpower the updated node representations. Consequently, the updated representations (i.e. the $h_u^{(k+1)}$ vectors) become overly reliant on the aggregated messages from neighbors (i.e., the $m_{N(u)}$ vectors), neglecting the node representations from the previous layers (i.e., the $h_u^{(k)}$ vectors). To mitigate this problem, one effective approach is to incorporate techniques like vector concatenation or *skip connections*. These methods aim to directly preserve information from earlier stages of message passing during the update phase. These concatenation and skip-connection techniques can be combined with most other GNN update methods. Therefore, considering $\text{UPDATE}_{\text{base}}$ as the fundamental update function that will be enhanced (for example, $\text{UPDATE}_{\text{base}}$ may be defined as in Equation 2.6), then various skip-connection updates build upon this base function.

One of the most straightforward skip-connection updates involves using concatenation to retain additional node-level information throughout the message passing process.

$$\text{UPDATE}_{\text{concat}}(h_u, m_N(u)) = [\text{UPDATE}_{\text{base}}(h_u, m_{N(u)}) \oplus h_u] \tag{2.15}$$

This process involves combining the output of the base update function with the previous-layer representation of the node by simply concatenating them. The underlying idea is to prompt the model to separate information

during message passing, distinguishing between the information received from neighboring nodes ($m_{N(u)}$) and the current representation of each node (h_u).

The GraphSAGE framework [30] introduced the concatenation-based skip connection, which was among the pioneering works to emphasize the potential advantages of such alterations to the update function. However, aside from concatenation, alternative forms of skip connections, like the linear interpolation method suggested by [31], can also be utilized.

$$\text{UPDATE}_{\text{interpolate}}(h_u, m_N(u)) = \alpha_1 \odot \text{UPDATE}_{\text{base}}(h_u, m_N(u)) + \alpha_2 \odot h_u \quad (2.16)$$

where $\alpha_1, \alpha_2 \in [0,1]^d$ are gating vectors with $\alpha_2 = 1 - \alpha_1$ and $\odot$ denotes element-wise multiplication. In this method, the updated representation is determined through a linear interpolation between the previous representation and the representation that was updated based on the neighborhood information. The gating parameter α_1 can be trained alongside the model using various techniques such as using an MLP on the current node representations to generate these gating parameters.

Jumping Knowledge

An alternative method to enhance the quality of final node representations involves leveraging the representations at every layer of message passing, rather than solely relying on the output of the final layer. The final node representations in this method are defined as,

$$z_u = f_{JK}(h_u^{(0)} \oplus h_u^{(1)} \oplus \ldots \oplus h_u^{(K)}) \quad (2.17)$$

This method, referred to as adding *jumping knowledge (JK) connections*, was initially introduced and analyzed by [32]. In numerous scenarios, the function f_{JK} can be set as the identity function, meaning just a concatenation of the node embeddings from each layer. Nonetheless, in [32] they also investigate alternative approaches like max-pooling methods and LSTM attention layers.

2.3.3 Graph Pooling

The message passing technique generates embeddings for individual nodes, but what if the aim is to make predictions for the entire graph instead? Essentially, the focus has been on learning representations for each node, but what if it is needed to learn an embedding that represents the entire graph $\mathcal{G}$? This is commonly known as *graph pooling*, as it involves aggregating the node embeddings to learn a representation for the entire graph.

Graph pooling can be solved like a set pooling problem. The aim to create a pooling function f_p that takes a set of node embeddings $z_1, \ldots, z_{|V|}$ and produces an embedding $z_{\mathcal{G}}$ representing the entire graph. Essentially,

any method discussed earlier for learning from sets of neighbor embeddings can be adapted for graph-level pooling.

In practice, there are two commonly used approaches for learning graph-level embeddings through set pooling. The first method is straightforward: it is a sum (or average) of the node embeddings:

$$z_{\mathcal{G}} = \sum_{v \in V} z_u \cdot f_n(|V|) \tag{2.18}$$

Another widely adopted method for set-based pooling involves leveraging a blend of LSTMs and attention mechanisms, drawing inspiration from [33]. In this pooling technique, a sequence of attention-based aggregations is iterated using the following set of equations, which are repeated for $t = 1, \ldots, T$ steps:

$$q_t = \text{LSTM}(o_{t-1}, q_{t-1}) \tag{2.19}$$

$$e_{v,t} = f_a(z_v, q_t), \forall v \in V \tag{2.20}$$

$$a_{v,t} = \frac{\exp(e_{v,t})}{\sum_{u \in V} \exp(e_{u,t})}, \forall v \in V \tag{2.21}$$

$$o_t = \sum_{v \in V} a_{v,t} z_v \tag{2.22}$$

In the given equations, the q_t vector represents a query vector for the attention at each iteration t. In Equation 2.20, q_t is employed to calculate an attention score for each node using an attention function $f_a : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ (for example, a dot product). Then, this attention score is normalized in Equation 2.21. Finally, in Equation 2.22, a weighted sum of the node embeddings is computed based on the attention weights, and this weighted sum is utilized to update the query vector using an LSTM update (Equation 2.19).

Typically, the q_0 and o_0 vectors are initialized with all-zero values. After iterating through Equations 2.19 - 2.22 for T iterations, an embedding for the entire graph is computed as:

$$z_{\mathcal{G}} = o_1 \oplus o_2 \oplus \ldots \oplus o_T. \tag{2.23}$$

This method presents an advanced architecture for attention-based pooling over a set, and it has gained popularity as a pooling technique in numerous graph-level classification tasks.

However, set pooling methods have a drawback in that they don't take advantage of the inherent structure of the graph. While it's understandable to treat graph pooling as a set learning problem, there are potential advantages in leveraging the graph's topology during the pooling process. A common approach to address this is by incorporating graph clustering

or coarsening techniques to aggregate node representations. In these types of methods, one assumes the existence of a clustering function $f_c : \mathcal{G} \times \mathbb{R}^{|V| \times d} \rightarrow \mathbb{R}^{+|V| \times c}$ which maps all nodes in the graph in some manner to c clusters. Specifically, this function is expected to output an assignment matrix $S = f_c(\mathcal{G}, Z)$, where $S[u, i] \in \mathbb{R}^+$ represents the strength of the association between node u and cluster i. An example of such a function could be the spectral clustering, where the cluster assignment is determined based on the spectral decomposition of the graph's adjacency matrix. More complex functions can be employed in order to predict cluster assignments such as another GNN as proposed in the DiffPool method [34].

Irrespective of the method employed to create the cluster assignment matrix S, the fundamental concept behind graph coarsening approaches is to utilize this matrix for graph reduction. Essentially, leveraging the assignment matrix S in order to calculate a new, condensed adjacency matrix,

$$A_{\text{new}} = S^T A S \in \mathbb{R}^{c \times c} \tag{2.24}$$

and a new set of node features

$$X_{\text{new}} = S^T X \in \mathbb{R}^{c \times d}. \tag{2.25}$$

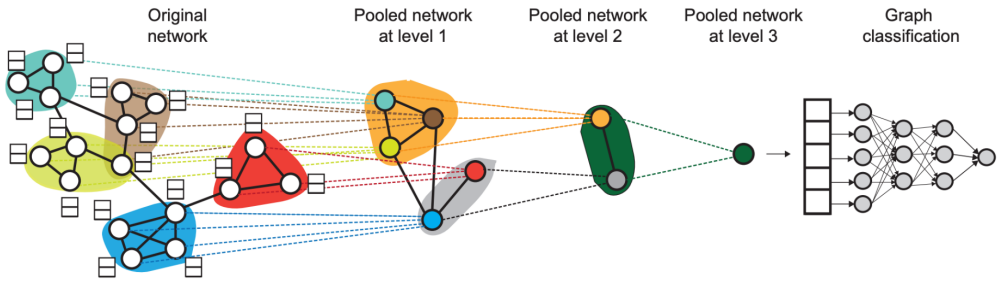

Figure 2.3: High-level illustration of DiffPool. At each hierarchical layer, a GNN model is employed in order to generate embeddings for nodes. Subsequently, these learned embeddings are utilized to group nodes into clusters, after which another GNN layer is applied to this coarsened graph. This iterative process is repeated for L layers, and the resulting final output representation is utilized for graph classification.

Consequently, the new adjacency matrix now reflects the connection strength (edges) between the clusters in the graph, while the new feature matrix encapsulates the combined embeddings for all nodes allocated to each cluster. Subsequently, one can apply a GNN to this coarsened graph and iterate the entire coarsening process for several iterations, progressively reducing the graph size at each step. Ultimately, the graph's final representation is computed by employing set pooling over the embeddings of nodes in a suitably coarsened graph.

2.4 Contrastive Learning

Contrastive learning was initially introduced in 2013 [35] for natural language processing. Their framework utilized co-occurring words as semantically similar pairs and employed negative sampling (picking of a random word to form a pair) for the development of word embeddings. Nowadays it is a machine learning technique used widely in self-supervised computer vision tasks. CL approaches teach a model to pull together the augmentations of a target image (*anchor*) and a matching (*positive*) image in embedding space, while also pushing apart the anchor from many non-matching (*negative*) images. This process brings similar class embeddings closer together while pushing embeddings from different classes farther apart. To achieve close proximity between the embeddings of positively related pairs in a metric space, the goal is to make the representations invariant to irrelevant differences within positive pairs in the input space. At the same time, the model should learn a covariant representation for negatively related pairs in order to represent the large distances between their embeddings in the metric space [36]. When quantifying the similarity between instances, distance metrics are often used, with Euclidean distance or cosine similarity being popular choices.

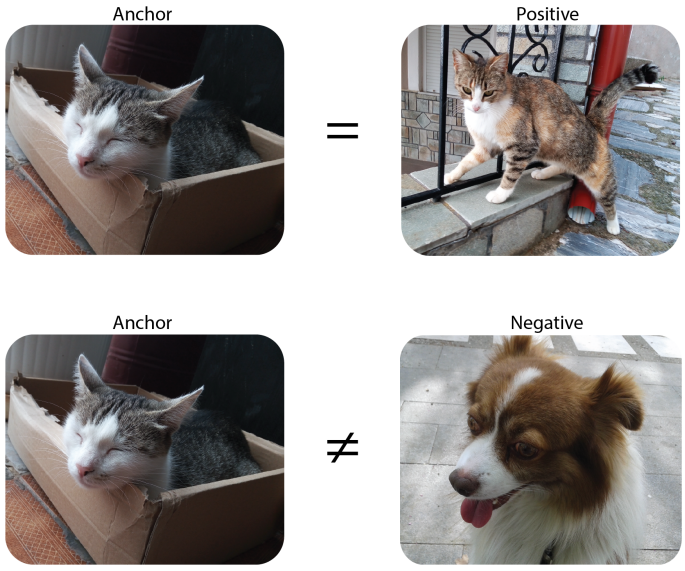

Figure 2.4: Example of negative and positive pairs in contrastive learning. Up: a positive pair. Down: a negative pair

2.4.1 Augmentation

Data augmentation is frequently the starting point in contrastive learning, where unlabeled data undergoes transformations or perturbations to generate varied instances or views. The aim is to enhance data diversity, enabling the model to encounter different angles of the same data. Popular

techniques involve altering images through cropping, flipping, rotation, random cropping, and adjusting colors. Examples of these techniques are shown in Figure 2.5. Through creating diverse instances, contrastive learning ensures that the model captures relevant information regardless of variations in the input data. Data augmentation techniques are not limited to images; they can be applied to various types of data including text, audio, and numerical data. For text data, augmentation techniques may involve synonym replacement, adding or removing words, paraphrasing, or introducing typographical errors. In audio data, augmentation might include adding background noise, changing pitch or speed, or compression (diversifying audio data by compressing audio samples at different levels). For numerical data, techniques such as adding noise, scaling, shifting, or interpolating values can be used. For time-series, some common techniques are adding noise, time warping (stretching or compressing the time axis of a time series, which changes the speed at which events occur), time shifting. The goal remains the same: to increase the variability of the data and expose the model to different perspectives, facilitating robust learning across diverse inputs.

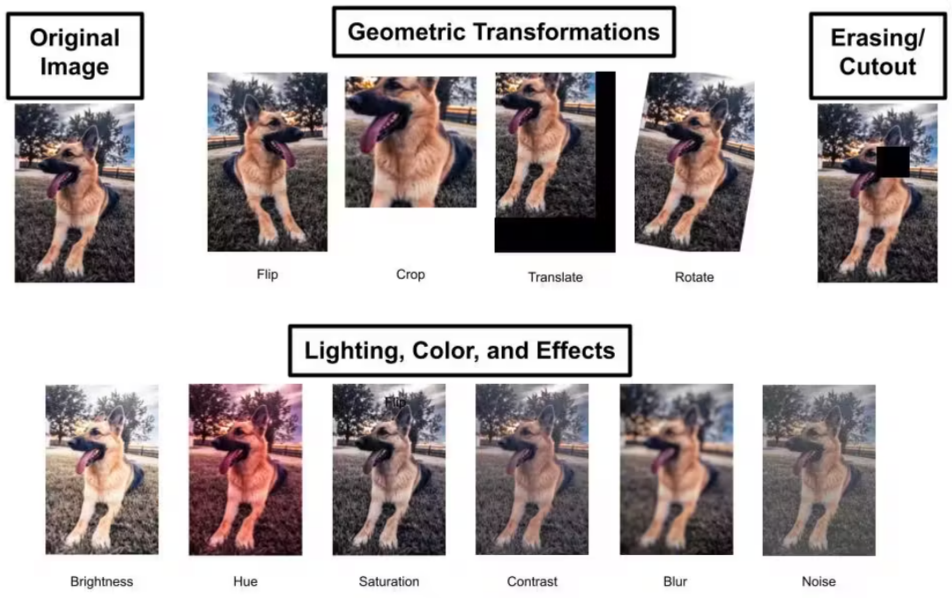

Figure 2.5: Overview of different types of augmentations for an image, source

2.4.2 Loss Functions in Contrastive Learning

Max-Margin Loss (Contrastive Loss)

This classic loss function, one of the earliest proposed in contrastive learning literature, introduced in [37] as *Contrastive Loss*, operates by maximizing the distance between samples from different distributions while minimizing the distance between samples from the same distribution. Mathematically,

it can be expressed as follows:

$$L_{\text{Max-Margin}}(x_i, x_j) = \mathbb{I}_{y_i=y_j}||z_i - z_j||_2^2 + \mathbb{I}_{y_i\neq y_j}\max(0, \epsilon - ||z_i - z_j||_2^2) \quad (2.26)$$

where $\mathbb{I}$ is the indicator function, y_i, y_j are the labels of data samples x_i, x_j and ϵ is a margin. The loss function penalizes the model if dissimilar data samples have a distance from each other smaller than ϵ, because, in that case the term $\epsilon - ||z_i - z_j||_2^2$ in the right hand of Equation 2.26 is bigger than 0. Also, by utilizing the margin ϵ, the loss function avoids pushing dissimilar points farther apart than required.

Triplet Loss

Triplet Loss introduced in [38] has been a very popular loss function for tasks involving metric learning and similarity learning. Triplet loss shares similarities with Max-Margin loss as both aim to minimize the distance between similar instances and maximize the distance between dissimilar ones. However, in triplet loss (and in many subsequent types of losses), unlike Max-Margin loss, the computation of loss involves taking a *positive* and a *negative* sample alongside the *anchor* sample simultaneously. Moreover, triplet Loss promotes the idea that dissimilar pairs should be separated from similar pairs by a minimum margin value.

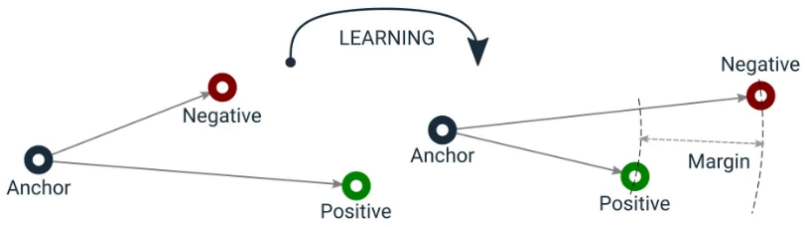

Figure 2.6: The Triplet Loss minimizes the distance between an anchor and a positive, both of which have the same identity, and maximizes the distance between the anchor and a negative of a different identity.

$$L_{\text{Triplet}}(x_i^a, x_i^p, x_i^n) = \max\left(||z_i^a - z_i^p||_2^2 - ||z_i^a - z_i^n||_2^2 + \epsilon, 0\right) \quad (2.27)$$

where x_i^a is the anchor, x_i^p is the positive i.e. a sample that has the same label as the anchor, x_i^n is a negative i.e. a sample that has a label different from that of the anchor and ϵ is a margin value that sets the distance between negative and positive samples. z_i, z_j are the embeddings of the elements x_i, x_j as produced by the model. Applying the contrastive loss on these projections z instead of x leads to an enhancement in the quality of the learned representations and accelerates the computation of the loss. Triplet loss differs from Max-margin loss in that it doesn't enforce anchor and positive samples to be encoded into the same point in the

vector space. This characteristic of Triplet loss allows for some degree of intra-class variance, unlike Max-margin loss, which essentially compels the distance between an anchor and any positive sample to approach zero. Triplet loss enables the stretching of clusters to encompass outliers while maintaining a margin between samples from different clusters e.g., negative pairs. Moreover, Triplet loss exhibits less greediness. In contrast to Max-margin loss, it considers its objective fulfilled when dissimilar samples are easily distinguishable from similar ones. It refrains from adjusting distances within a positive cluster unless negative examples intervene. This behavior stems from Triplet loss's aim to maintain a margin between distances of negative and positive pairs. Conversely, Max-margin loss only considers the margin value when assessing dissimilar pairs, disregarding the placement of similar pairs at that moment. Consequently, Max-margin loss may converge to a local minimum sooner, whereas Triplet loss may persist in organizing the vector space to a more optimal state. This behaviour is also illustrated in Figure 2.7.

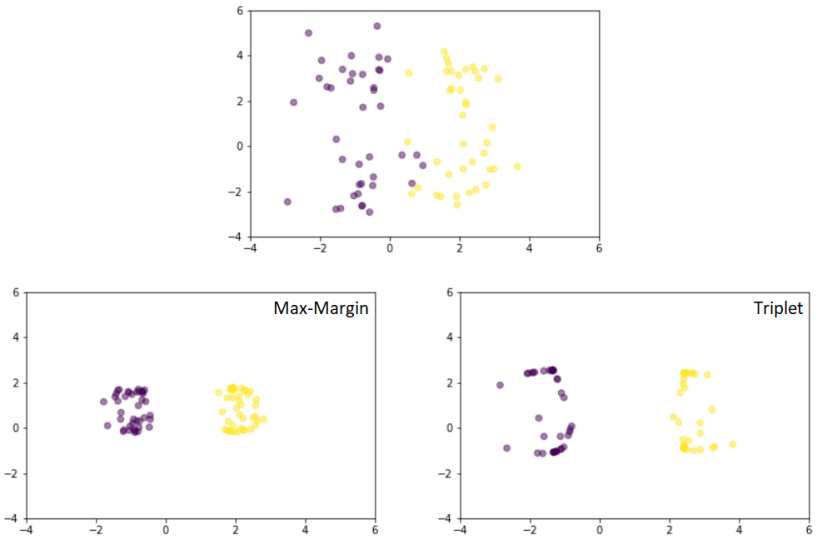

Figure 2.7: Example of organizing the vector space by Max-Margin loss and Triplet loss, source. On the upper half, the vector space before contrastive learning has been applied. Down left: Vector space after Max-margin loss contrastive learning. Max-margin has the tendency to make samples fall into the same point in the vector space. Down right: Vector space after Triplet loss contrastive learning.

NT-Xent

The NT-Xent loss (Normalized temperature-scaled cross entropy loss) was introduced in SimCLR [39]. In NT-Xent, the distance metric used is the cosine similarity, which can be expressed for a given pair (a, b) as:

$$sim(a,b) = \frac{a^T b}{\|a\| \cdot \|b\|} \tag{2.28}$$

For a given positive pair of elements (x_i, x_j) the loss can be defined as:

$$L_{\text{NT-Xent}}(x_i, x_j) = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{k \neq i} \exp(\text{sim}(z_i, z_k)/\tau)} \tag{2.29}$$

where $\mathbb{I}_{k \neq i} \in \{0, 1\}$ is an indicator function which returns 1 if $k \neq i$ and 0 if $k = i$ in order to avoid the computation of the distance between a sample and itself. N is the size of the batch, τ is a temperature hyper-parameter. In SimCLR, 2N images are inputted, stemming from N original images. Each of these N images undergoes augmentation with a random set of transformations, yielding 2 augmented versions. Consequently, a total of 2N images are included in a single training batch for the model where the 2N-2 of them are negative samples. Results of comparisons between identical values (z_i, z_i) are disregarded, as an element being perfectly similar to itself doesn't offer any useful learning insights to the model. Temperature scaler τ is applied to the input of the softmax function to either smooth out or accentuate the output. A high τ diminishes the variance in the output distribution, leading to a milder representation of the labels. Conversely, a lower τ amplifies the variance in the output distribution, emphasizing the prominence of the maximum value compared to others.

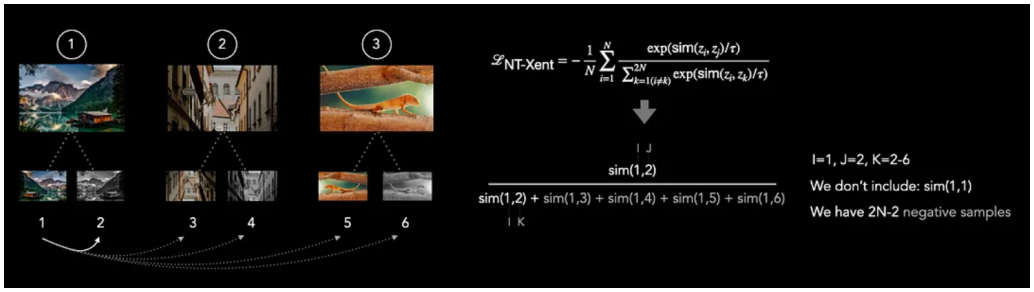

Figure 2.8: Augmentation of input and loss calculation, source

The NT-Xent loss encourages the model to pull embeddings of positive augmentation pairs closer together in the embedding space, while pushing embeddings of negative augmentation pairs farther apart. This self-supervised learning strategy aids the model in acquiring robust representations by considering diverse views of the same data instance. An example of the use of NT-Xent is depicted in Figure 2.8.

InfoNCE

The InfoNCE loss (Information Noise Contrastive Estimation) used in MoCo [40] is a close variant of NT-Xent. It utilizes the dot product instead of the cosine similarity as the distance metric and uses an additional structure, a queue instead of just exploiting all the information already contained on the current mini-batch. In the queue, the most recent encoded representations of the augmentations of the data samples are enqueued

(pushed), and the oldest ones are dequeued thus it gets repeatedly updated during the model's training. Mathematically the InfoNCE loss is formulated as,

$$L_{\text{InfoNCE}}(q) = -\log \frac{\exp(q \cdot k^+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)} \tag{2.30}$$

where K is the number of negative samples, q is the query, i.e the augmentation of the data sample that is being contrasted and k are the keys, i.e the augmentations of the negative samples. The loss diminishes when q is similar to its positive key k^+ and dissimilar to all other keys.

2.4.3 Supervised contrastive learning

Supervised contrastive learning (SCL) [41] takes the concept of self-supervised contrastive learning and extends it into a fully supervised environment by utilizing the labels of the dataset. It considers samples of the same class as positive examples and it contrasts all samples from different classes thus creating negatives examples.

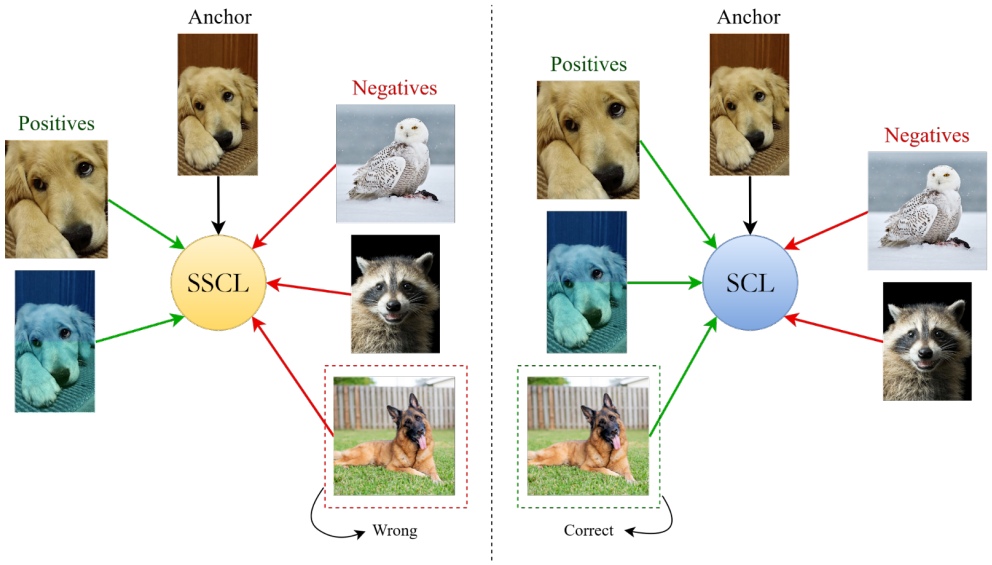

Figure 2.9: Advantage of using Supervised Contrastive Learning (SCL) instead of Semi-Supervised Contrastive Learning (SSCL), source. In the SSCL, the image of the German Shepherd (as it is was not generated through augmentation of the anchor) is wrongly considered as a negative of the anchor even though both images belong to the class of dogs. In SCL, there is no such problem as the labels are used to categorize the images.

Therefore, SCL effectively utilizes label information, ensuring that samples from the same distribution (such as multiple images of various dogs) are positioned closely together in the latent space, while samples from different classes are pushed apart in the latent space. Consequently, the loss functions employed in SCL frameworks differ from those utilized in

SSCL frameworks. The loss function used by [41] is a variation of NT-Xent loss as it incorporates multiple positive examples per anchor (which is possible due to the usage of labels), as opposed with the conventional self-supervised contrastive learning method that typically relies on just one positive instance. This loss function is termed SupCon loss and is formulated as,

$$L_{\text{SupCon}}(x_i) = \sum_{i \in I} \left(-\frac{1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp(z_i \cdot z_p / \tau)}{\sum_{a \in A(i)} \exp(z_i \cdot z_a / \tau)} \right) \tag{2.31}$$

where $i \in I \equiv \{1 \dots 2N\}$ is the index of an arbitrary augmented sample and $A(i) \equiv I \setminus \{i\}$ includes the indices of all *positive* and *negative* samples in the batch. The index i is the *anchor* and the index p is of a *positive* sample. $P(i) \equiv \{p \in A(i) : \tilde{y}_p = \tilde{y}_i\}$ is the set of indices of all positives in the multi-viewed batch distinct from i, and $|P(i)|$ is its cardinality.

2.5 Related work

GCN [27] is one of the most widely used GNNs, gaining great popularity due to its combined simplicity and effectiveness. Despite its many benefits that make it a reliable solution for many problems, it struggles in this imbalanced and with fraudster camouflage filled environment as it was not designed specifically to address such occasions. CARE-GNN [4] is a multi-relational, camouflage-resistant GNN that uses label-aware similarity and an adaptive threshold trained with reinforcement learning to filter neighbors for nodes. However it does not address the problem of imbalanced class distribution between fraudsters and benign users in the fraud graph. PC-GNN [5] is a multi-relational GNN that addresses class imbalance by oversampling the minority class and undersampling the majority one. The methods described above do not account for the impact of feature camouflage on similarity-based neighbor sampling methods and the aggregation of features. For this reason, CACO-GNN [6], a multi-relational GNN, employs supervised contrastive learning to improve the consistency of neighbor sampling by effectively compacting the feature camouflage of fraudsters in graphs. IMINF [7] is designed for fraud detection on multi-relation graphs using supervised contrastive learning like CACO-GNN, and offers specialized components, for example, an interactive learning framework designed for addressing challenges such as imbalanced label distribution and inappropriate representation of the target node during relation aggregation. Works such as [6], [7] offer significant improvements by incorporating contrastive learning but they do not compact effectively the imbalance during it as they undersample the training batch (or not address the imbalance at all) in order to mitigate the imbalance which compromises the quality of the produced embeddings.

2.6 Definitions and Problem statement

We define some necessary terminology in this section.

Definition 1 (Attributed network/graph) *An attributed graph can be denoted as* $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{X}\}$*, where* $\mathcal{V} = \{v_1, \ldots, v_N\}$ *is a set of nodes,* $\mathcal{E}$ *is the edge set,* $\mathcal{X} \in \mathbb{R}^{N \times d}$ *is the feature matrix and* d *is the number of features each node has. An extended attributed graph* $\mathcal{G}' = \{\mathcal{V}, \mathcal{E}, \mathcal{X}'\}$ *has feature matrix* $\mathcal{X}' \in \mathbb{R}^{N \times (d+k)}$ *where* k *is the number of additional types of features added to* $\mathcal{G}$*. Hence,* $\mathcal{X}_{v,:}$ *is the feature vector (row) of node* v *and* $\mathcal{X}_{:,j}$ *is the feature vector (column) for feature* j*.*

Definition 2 (Multi-relational graph) *Given an attributed multi-relational graph* $\mathcal{G} = \{\mathcal{V}, \mathcal{X}, \mathcal{E}_r|_{r=1}^{R}, \mathcal{A}_r|_{r=1}^{R}, \mathcal{Y}\}$*,* $\mathcal{E} = \{E_1, \ldots, E_R\}$ *is the edge set of* R *relations,* $\mathcal{A} = \{A_1, \ldots, A_R\}$ *is a set of corresponding adjacency matrices of* R *relations and* $\mathcal{Y} = \{\mathcal{Y}_f, \mathcal{Y}_l\}$ *are the labels of nodes* $\in \mathcal{V}$*, where* $\mathcal{Y}_f$ *corresponds to fraudsters and* $\mathcal{Y}_l$ *to legitimate nodes. For each node* $v_i \in \mathcal{V}$*,* x_i *is a* d*-dimension feature vector and* $y_i \in \mathcal{Y}$ *is the label of the node.*

Problem 1 *Given a multi-relational graph* $\mathcal{G}$*, the problem to be solved is the detection of fraud nodes in* $\mathcal{G}$*. Hence, 1) there are two types of labels,* $\mathcal{Y} = \{\mathcal{Y}_f, \mathcal{Y}_l\}$ *where* $y_i = \mathcal{Y}_f$ *is the label for a fraudster and* $y_i = \mathcal{Y}_l$ *for a legitimate user, and 2) the number of fraud nodes is much smaller than the number of legitimate ones, i.e.,* $\sum_{i=1}^{N} \mathbb{I}[y_i = \mathcal{Y}_f] \ll \sum_{i=1}^{N} \mathbb{I}[y_i = \mathcal{Y}_l]$*, where* $\mathbb{I}$ *is the indicator function.*

3 Methodology

In order to solve the Problem 1 defined in Section 2.6, we introduce a feature generation module, a synthetic fraud generation module and neighborhood sampling. This section outlines these three items, along with the training procedure, model configuration, and evaluation metrics utilized in the study.

3.1 Feature generation

For each node $v_i \in \mathcal{V}$ and for all relations $E_1, \ldots, E_R$ in a given multi-relational graph $\mathcal{G}$, we will calculate the probability distributions of six additional quantitative structural features for fraudster nodes (with label $\mathcal{Y}_f$) and legitimate nodes (with label $\mathcal{Y}_l$). These additional features are proposed here for the first time and are used alongside the basic pre-existing features of each dataset (32 for the YelpChi and 25 for the Amazon [42] dataset). The choice to introduce the new features is made because the pre-existing features describe only atomic attributes of each node and are not by themselves sufficient - for example they do not capture the local structure of the graph. The additional structural features are:

- **degree**: the degree of the node,
- **second-order degree**: the number of nodes exactly 2 hops away from the target node,
- **opposite label count**: the number of neighbors of the opposite label compared to that of the node,
- **preferential attachment**: the product of the target node's degree and the average degree of its neighbors (a variation of the original measure is used since the given problem considers static graphs),
- **clustering coefficient** [43]: the degree to which the neighbors of the target node are interconnected,
- **structural similarity** [44]: the resemblance in the local structures of nodes within a complex network, quantified using the Kullback-Leibler (K-L) divergence to measure the difference between the probability distributions of node pairs.

After computing these structural features for each node, we go on to calculate their empirical distributions for each class. Then, for each node, we replace its feature values with a sample from that feature's distribution (for the class the node belongs to). This is done in order to "prevent" any fraudster from directly manipulating its features to desired values[2]. This process is denoted as,

$$x'_{struct} \sim \text{Distribution}(\mathcal{X}_{:,struct})$$

where x'_{struct} denotes the feature column of values produced for the additional structural feature named *struct*. Then, the structural feature values are appended to the node's existing "basic" features to form the new feature vector for the node. Hence, for a target node $v \in \mathcal{V}$ (fraudster or legitimate),

$$\mathcal{X}_{v,:} = \left[x_{basic_1}, \ldots, x_{basic_d}, x'_{struct_1}, \ldots, x'_{struct_k}\right] \tag{3.1}$$

is the new feature vector of the node v. $\mathcal{X}_{v,:} \in \mathbb{R}^{(d+k)}$ where d is the number of basic features and k is the dimension of the structural feature vector. The feature generation module is illustrated in Figure 3.1. As we will later see, this replacement of the original node features by samples from their class-specific distributions will lead to better performance. We also experimented with maintaining node features at their original values, but this lead to inferior results.

2 Depending on the graph relation, some of the structural features described above may not be possible to calculate or may have a constant value across all nodes which is undesired. Thus, the number of additional structural features k may vary but will be at most 6 for any given dataset.

3.2 Contrastive similarity

To address the challenge of camouflage, an extra similarity metric is employed along with feature similarity to differentiate between node embeddings. This similarity is called contrastive similarity and is produced by utilising supervised contrastive learning. As in [6], in this study, supervised contrastive learning is used in order to measure the contrastive similarity between nodes and the learned positive prototype of the minority class. Thus the positive prototype is the representative of its class, an idea from [45]. The positive prototype is not a real node transformed in the embedding space but is, instead, an artificial embedding that is trained in order to capture the most representative features of its class. Using a positive prototype obtained by the minority class (i.e fraudsters) as the anchor instead of contrasting each target node with each fraudster removes the computational complexity. For the central node v at l-th layer under relation $\mathcal{E}_r$ the contrastive embedding is,[3]

$$C_{v,r}^{(l)} = \sigma\left(W_{v,r}^{(l)}, h_{v,r}^{(l-1)}\right) \tag{3.2}$$

where $h_{v,r}^{(l-1)}$ is the node embedding for node v at relation r learned in layer $l-1$, $W_{v,r}^{(l)}$ is the parameter that will be learned, σ is the activation

3 The notation for this and the following formulas is borrowed from [6].

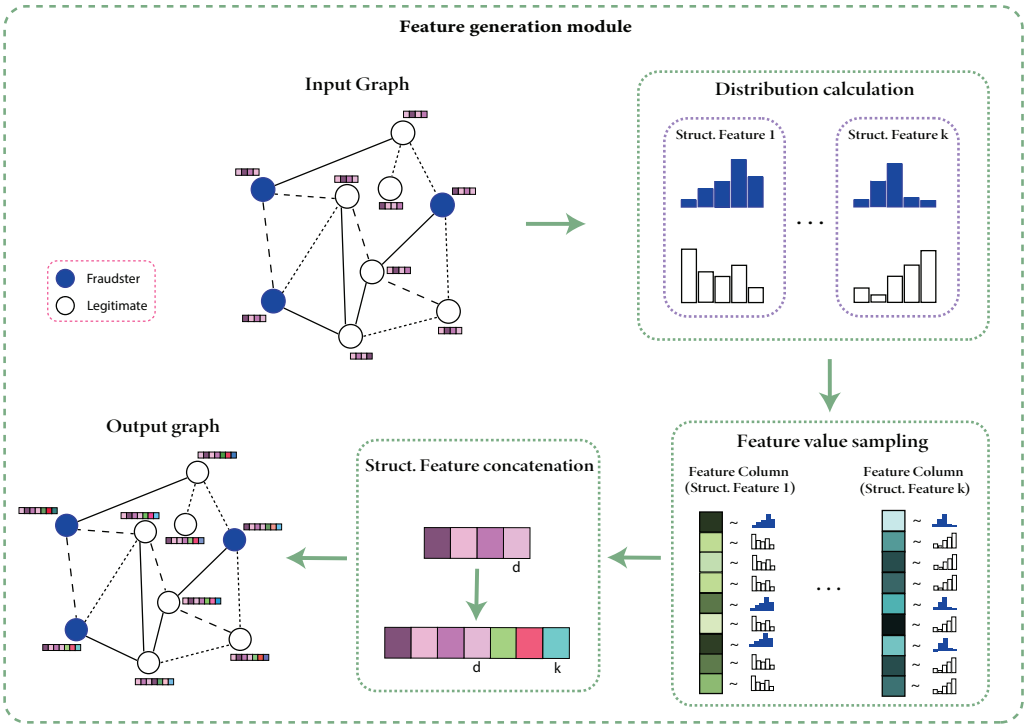

Figure 3.1: Feature generation module showing the total process of enhancing an input graph $\mathcal{G}$ with additional structural features. At the end of the process an extended graph $\mathcal{G}'$ is created where each node has an extended feature vector.

function, (set to ReLU in this study). $C_{v,r}^{(l)}$ is the contrastive embedding and is updated during the training of the model. Then, the similarity score between the positive prototype and any target node is calculated,

$$S_{v,r}^{(l)} = Similarity\left(C_{v,r}^{(l)}, t_r^{(l)}\right) \tag{3.3}$$

where $t_r^{(l)}$ is a learnable parameter. $Similarity(\cdot)$ is any distance function, in this case it is the cosine distance.

$$cos(a,b) = \frac{a \cdot b}{||a|| \cdot ||b||}. \tag{3.4}$$

Equation 3.3 will represent the probability of a node embedding being associated with a fraudster. This means that as the contrastive learning process progresses, the contrastive embedding of the fraudulent node will be close to that of the positive prototype.

Supervised contrastive learning loss

Each contrastive embedding of a node is contrasted against the positive prototype, t^l

$$\mathcal{L}^r_{\text{Contra}} = -\sum_{l=1}^{L} \sum_{\substack{y_i=1,\\ y_j=0}} \left[\log \frac{\exp(t_r^{(l)} \cdot c_{i,r}^{(l)}/\tau)}{\exp(t_r^{(l)} \cdot c_{i,r}^{(l)}/\tau) + \sum_j \exp(t_r^{(l)} \cdot c_{j,r}^{(l)}/\tau)} \right] \tag{3.5}$$

where τ is the temperature hyperparameter. The overall contrastive learning loss function can be expressed as:

$$\mathcal{L}_{\text{Contra}} = \frac{1}{R} \sum_{r=1}^{R} \mathcal{L}^r_{\text{Contra}} \tag{3.6}$$

3.3 Synthetic fraud node generation

Previous methods have introduced neighbor sampling to mitigate the imbalanced nature of the problem and the camouflage of the fraudsters. However, in the process of supervised contrastive learning, the imbalance is not addressed resulting in less negative-positive pairs for the model to learn from. The addition of synthetic frauds effectively addresses this issue, as illustrated in Figure 3.2.

Hence, for enhancing the supervised contrastive learning process, synthetic nodes with labels $y_i \in \mathcal{Y}_1$ are generated based on the distributions of the basic features and also of those of the structural features. As this process happens only in the SCL module, new edges with existing nodes do not need to be established. Only synthetic features are generated. Thus, for each basic feature $\mathcal{X}_{:,basic}$ and structural feature $\mathcal{X}_{:,struct}$, we sample from their respective distributions to create synthetic values x'_{basic} and x'_{struct}. The sampling is denoted as,

$$x'_{basic} \sim \text{Distribution}(\mathcal{X}_{:,basic})$$

$$x'_{struct} \sim \text{Distribution}(\mathcal{X}_{:,struct})$$

Then, the synthetic feature values are combined to form the feature vector for the synthetic fraud node.

$$\mathcal{X}_{synthetic,:} = \left[x'_{basic_1}, \ldots, x'_{basic_d}, x'_{struct_1}, \ldots, x'_{struct_k} \right] \tag{3.7}$$

3.4 Neighborhood sampling

Fraudulent entities adeptly disguise themselves by emulating the actions of legitimate users or establishing illicit connections with them. Consequently,

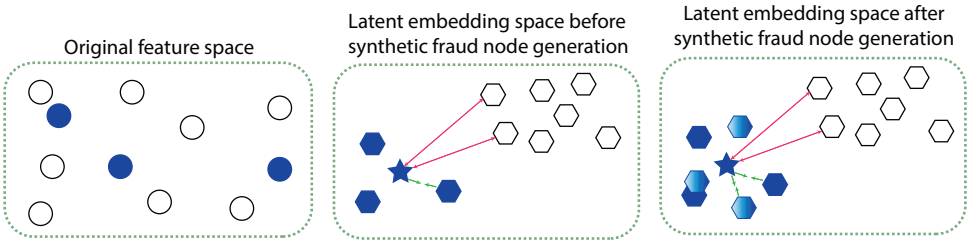

Figure 3.2: Effect of synthetic fraud generation in latent space produced through contrastive learning. Blue points indicate the fraudsters, white ones indicate the legitimate users while blue gradient points are the synthesized fraudsters. The blue star denotes the prototype of the fraudster class. The representation quality of the positive (fraud) prototype is increased as it is contrasted with more positive elements.

directly integrating all neighboring nodes in the intra-relation aggregation stage becomes unfeasible. Instead, a neighbor sampling technique that selects neighbors based on feature and contrastive similarity is used. The contrastive embedding measures the distance of the target node from the positive prototype. Contrastive similarity is the distance between two contrastive embeddings. Nodes with matching labels and the smallest contrastive similarity difference from the target node are sampled as neighbors. The set of contrastive similarity differences between a node v and each neighbors is defined as,

$$\mathbb{D}_r^{(l)}(v) = \left\{ |S_{v,r}^{(l)} - S_{u,r}^{(l)}| \right\} |_{A_r(v,u)>0} \tag{3.8}$$

The top k neighbors with the smallest difference in contrastive similarity are picked, and this set of sampled neighbors is denoted as $\dot{\mathcal{N}}_r^{(l)}(v)$.

The feature similarity score concentrates on the distance of the target node from other nodes based on their features in the latent space. A higher score signifies closer node proximity in terms of feature consistency. The set of feature similarities of a node v to its neighbors is defined as,

$$\mathbb{S}_r^{(l)}(v) = \left\{ Similarity(h_{v,r}^{(l)}, h_{u,r}^{(l)}) \right\} |_{A_r(v,u)>0} \tag{3.9}$$

The top k neighbors with the biggest feature similarity are sampled and this set is depicted as $\ddot{\mathcal{N}}_r^{(l)}(v)$. The set union of the contrastive similar neighbors and feature similar neighbors is the set of consistent neighbors of a target node v and is defined as,

$$\mathcal{N}_r^{(l)}(v) = \dot{\mathcal{N}}_r^{(l)}(v) \cup \ddot{\mathcal{N}}_r^{(l)}(v) \tag{3.10}$$

3.5 Intra-relation aggregation and Inter-relation aggregation

In the intra-relation aggregation step, for each target node v is calculated its embedding w.r.t each relation R. Therefore, the intra-relation embedding

of node v at relation r ($h_{v,r}^{(l)}$) is the aggregation of the features of v in the previous layer $l-1$, the contrastive embedding of v and the embedding of the features of the neighbors of v in the current layer. This statement can be described compactly as

$$h_{v,r}^{(l)} = \sigma\left(W_r^{(l)}\left(h_{v,r}^{(l-1)} \oplus c_{v,r}^{(l)} \oplus AGG_r\{h_{u,r}^{(l)}\}|_{r\in R}\right)\right) \tag{3.11}$$

where $\oplus$ is the concatenation operator, AGG_r is the mean aggregator and $W_r^{(l)}$ is the learnable weight matrix at layer l for relation r. Also, inter-relation aggregation is performed, where the information from all relations is aggregated in order to update the embedding of the target node v,

$$h_v^{(l)} = \sigma\left(W^{(l)}\left(h_v^{(l-1)} \oplus \{h_{u,r}^{(l)}\}|_{r\in R}\right)\right) \tag{3.12}$$

where $W^{(l)}$ is the learnable weight matrix at layer l.

3.6 Training

This section outlines the training of the model, the steps taken in order to produce the final probability of a node being fraudster or legitimate and discusses the loss function of the model. The overall training of BRIE is shown in Algorithm (3.1).

Algorithm 3.1 BRIE

input

multi-relational graph extended with structural features: $\mathcal{G}' = \{\mathcal{V}, \mathcal{X}', \mathcal{E}_r|_{r=1}^{R}, \mathcal{A}_r|_{r=1}^{R}, \mathcal{Y}\}$
batches, epochs, layers: B, E, L;

output

vector representations $h_v^{(L)}$ for each node in $\mathcal{V}$

for $e \leftarrow 1, \ldots, E$ **do**
  for $b \leftarrow 1, \ldots, B$ **do**
    Construct subgraphs using nodes in b and the edges connecting them to their immediate neighbors
    for $l \leftarrow 1, \ldots, L$ **do**
      for $r \leftarrow 1, \ldots, R$ **do**
        Sample consistent neighbors for target node v using (**??**), (3.9), (3.10)
        Synthesize fraudster nodes for contrastive learning process using (3.7)
        $h_{v,r}^{(l)} \leftarrow$ (3.11), $v \in \mathcal{V}_b$
      end for
      $h_v^{(l)} \leftarrow$ (3.12), $v \in \mathcal{V}_b$
      $\mathcal{L}_{\text{Contra}} \leftarrow$ (3.5), (3.6)
      $\mathcal{L}_{\text{GNN}} \leftarrow$ (3.14)
      $\mathcal{L} \leftarrow$ (3.15)
    end for
  end for
end for

After obtaining the final embedding $h_v^{(l)}$ for each node in $\mathcal{V}$, the embeddings are passed to an MLP in order to generate the probability vector p_v that indicates with which probability each node belongs to the two classes. It is calculated as,

$$p_v = MLP(h_v^{(L)}). \tag{3.13}$$

Then, the model is trained using a cross-entropy loss function,

$$\mathcal{L}_{\text{GNN}} = -\sum_{v \in \mathcal{V}} \left(y_v \log(p_v) + (1 - y_v) \log(1 - p_v) \right). \tag{3.14}$$

The overall loss function of the proposed model is defined as,

$$\mathcal{L} = \mathcal{L}_{\text{GNN}} + \lambda_1 \mathcal{L}_{\text{Contra}} \tag{3.15}$$

where λ_1 is hyperparameter. Figure 3.3 illustrates the training process graphically.

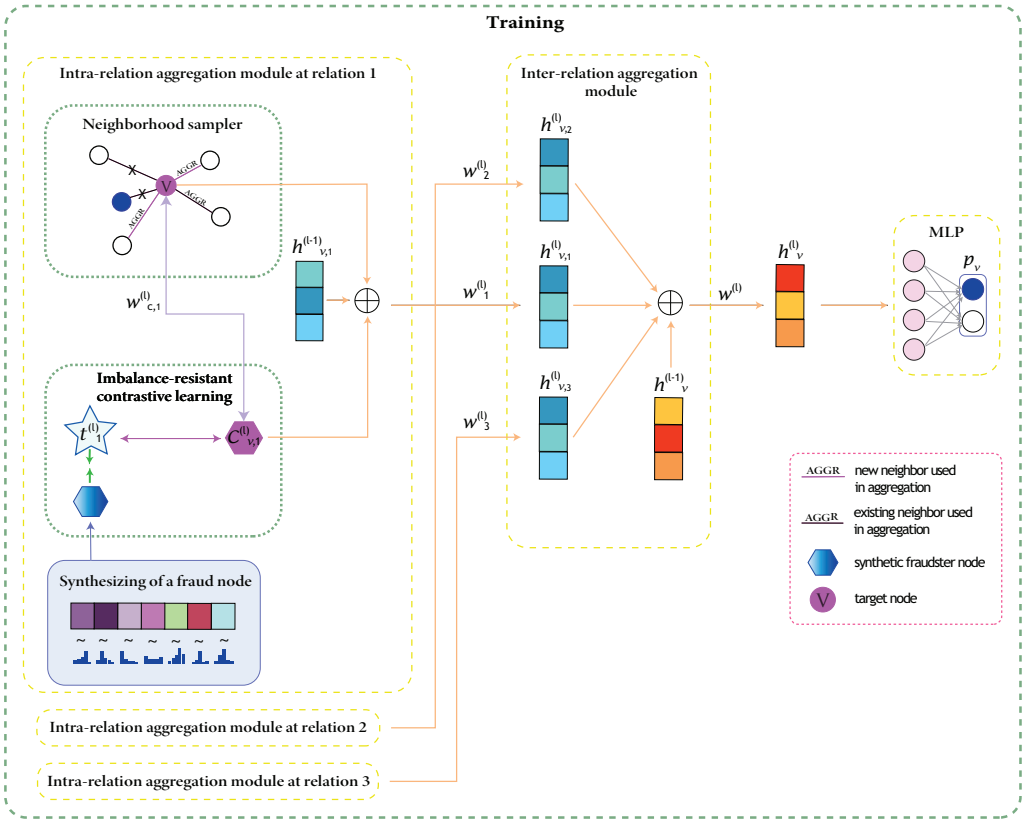

Figure 3.3: Training process for a node v. In the intra-relation aggregation the imbalance-resistant contrastive learning module, enhanced with synthesized fraud nodes collaborates with the neighborhood sampler in order to produce the node embedding $h_{v,r}$ which will feed the inter-relation aggregation. The inter-relation aggregation produces the final node embedding h_v for node v. The legitimacy of a node v is decided by a MLP at the end which transforms the node embedding h_v to a probability vector.

4 Experiments

In this section we give practical details regarding the numerical experiments and compares the results with those of recent works. We also discuss the statistical analysis of the models' performance. The code for the BRIE model is available at https://github.com/brie-user/brie-model.

4.1 Implementation details and Metrics

For training, the non-fraud samples are undersampled in order to create a non-fraud-to-fraud ratio of 3:1 for the Amazon dataset and 2:1 for YelpChi. Synthesizing fraud nodes permits BRIE to use more non-fraud samples in each training epoch as opposed to [6] and [7] where they used a 1:1 ratio. The learning rate was set to 0.01, λ to 2, temperature τ to 0.2 and number of layers L to 1. The optimizer was AdamW, and the node embedding size was 64. The model was trained using the mini-batch training approach. The batch size for Amazon was 256 while for YelpChi it was 1024. The training-test-validation split was 40%-20%-20% and the samples of each set were randomly chosen. To achieve the best parameters possible, the model was trained for a fixed number of epochs ($E = 120$). During training, at five-epoch intervals, the model was tested on the validation test and if the current parameters yielded better results than the previous ones (based on the AUC metric), then the currently best model parameters were updated. At the end of the training, the best-known parameters were restored in order to generate predictions on the test set.

For the task of imbalanced classification, 3 metrics are chosen:

- F1-macro: the unweighted mean of the F1-score of each class
- AUC: measures of a binary classification model's ability to distinguish between positive and negative examples. Is insensitive to class imbalance.
- Recall: measures the proportion of actual positive instances correctly identified by the classification model. It is calculated by dividing the number of true positives by the sum of true positives and false negatives.

4.2 Dataset

The experiments are conducted on two real-world datasets.[4] The YelpChi dataset [8] collects hotel and restaurant reviews filtered (spam) and recommended (legitimate) on Yelp. It treats reviews as nodes and are given 32 handcrafted features. The dataset consists of three relations: 1) R-U-R: connects reviews posted by the same user; 2) R-S-R: connects reviews under the same product with the same star rating (1-5 stars); 3) R-T-R: connects two reviews under the same product posted in the same month.

[4] They can be found here

The Amazon dataset [42] includes product reviews under the Musical Instruments category. It treats users as nodes and are given 25 handcrafted features from [46]. The dataset consists of three relations: 1) U-P-U: connects users reviewing at least one same product; 2) U-S-U: connects users having at least one same star rating within one week; 3) U-V-U: connects users with top 5% mutual review text similarities (measured by TF-IDF) among all users. The statistics of these two datasets are shown in Table 4.1. The two chosen datasets are widely considered as benchmarks, and have been used by several studies in order to measure performance of fraud detection models.

Table 4.1: Dataset statistics

Dataset	#Nodes (Fraud%)	Relation	#Edges
YelpChi	45,954 (14.5%)	R-U-R	49,315
		R-T-R	573,616
		R-S-R	3,402,743
		ALL	3,846,979
Amazon	11,944 (9.5%)	U-P-U	175,608
		U-S-U	3,566,479
		U-V-U	1,036,737
		ALL	4,398,392

4.3 Results and comparison with prior works

Several GNN models are compared at the task of fraud detection on 2 real-world datasets. Results from this comparison (measured in F1-macro, AUC, Recall) are shown in Table 4.2. In Fig 4.3 are shown the confusion matrices, in Fig. 4.2 the training process and in Fig. 4.3 the AUC-ROC curves from running BRIE on the 2 datasets.

The results of the statistical tests comparing BRIE, IMINF, and CACO-GNN models on the YelpChi dataset reveal mixed findings. For the comparison between BRIE and IMINF, while the t-test suggests a noticeable difference with a statistic of 3.351 and a p-value of 0.079, this difference does not reach conventional significance levels ($p < 0.05$) but for a higher

Table 4.2: Comparison of different GNN models on Amazon and YelpChi datasets. Bold entries correspond to the highest scores achieved. The list of works cited is not exhaustive; additional competitive models (which are tested on the same datasets as ours but nevertheless do not perform better than BRIE) include [47, 48, 49]

Method	Amazon			YelpChi		
	F1-macro	AUC	Recall	F1-macro	AUC	Recall
GCN	55.91	75.25	67.81	47.81	54.98	53.87
CARE-GNN	-	89.73	88.48	-	75.70	71.92
PC-GNN	89.56	95.86	-	79.87	63.00	-
CACO-GNN	89.79	97.64	92.83	74.19	87.12	77.65
IMINF	**94.86**	98.56	**97.01**	78.12	94.13	87.27
BRIE	94.41	**98.92**	95.20	**92.40**	**99.16**	**95.33**

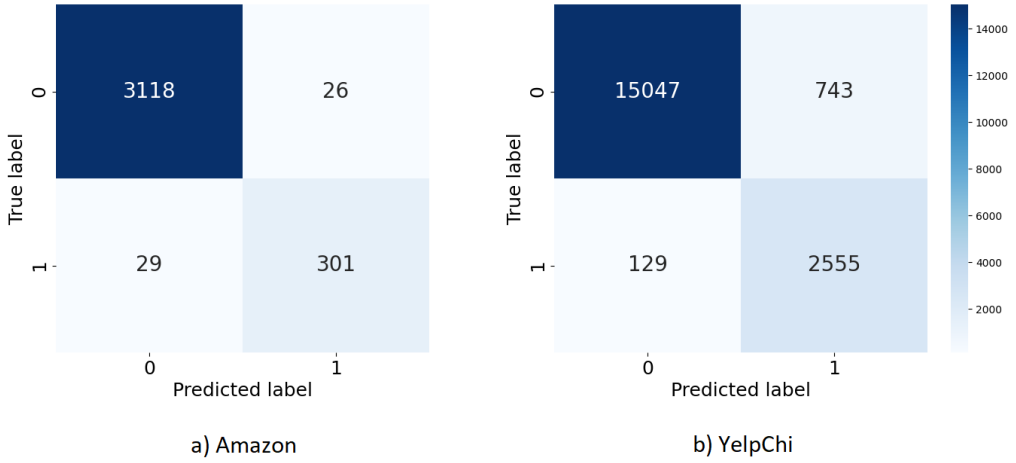

Figure 4.1: Confusion matrices.

threshold ($p < 0.10$) it does. The Wilcoxon signed-rank test corroborates this non-significant finding with a statistic of 0.0 and a p-value of 0.25. Conversely, comparing BRIE with CACO-GNN shows a statistically significant difference based on the t-test (statistic = 8.092, p-value = 0.015), indicating a significant performance disparity between these models. However, the Wilcoxon test does not support this finding (statistic = 0.0, p-value = 0.25), highlighting potential distributional differences influencing the results. Similarly, the comparison between IMINF and CACO-GNN reveals a marginally significant difference according to the t-test (statistic = 4.168, p-value = 0.053), which is not confirmed by the Wilcoxon test (statistic = 0.0, p-value = 0.25). These discrepancies underscore the importance of considering both parametric and non-parametric tests, as well as the underlying assumptions, when interpreting comparative model performance in the dataset.

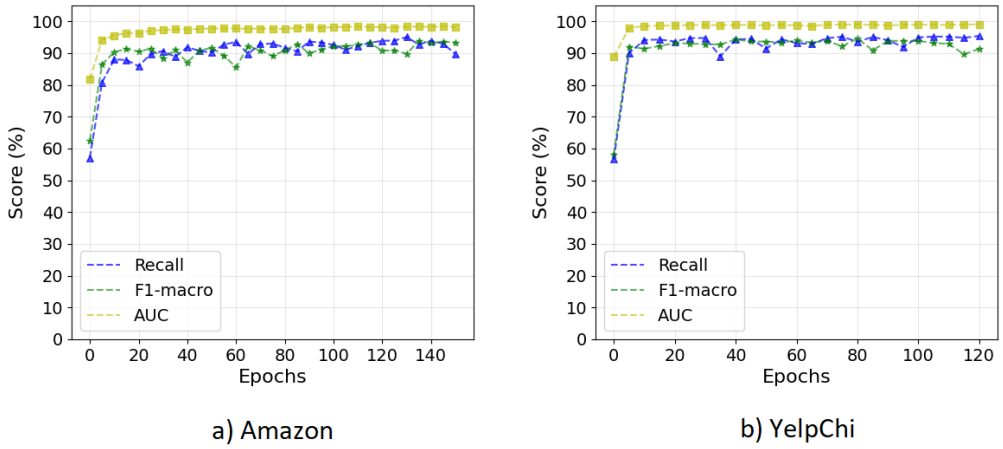

Figure 4.2: The training process.

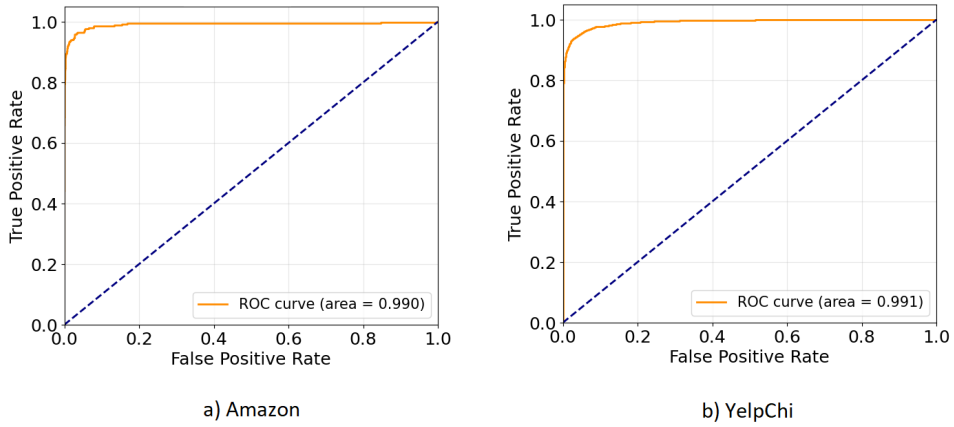

Figure 4.3: AUC-ROC curve.

Based on the statistical analysis conducted on the Amazon dataset, none of the pairs of model comparisons show statistically significant differences in performance. Specifically, the paired t-tests between BRIE and IMINF (p-value = 0.423), BRIE and CACO-GNN (p-value = 0.107), and IMINF and CACO-GNN (p-value = 0.115) all yield p-values greater than the standard significance threshold of 0.05. Similarly, the Wilcoxon signed-rank tests for the same pairs (p-values = 0.500, 0.250, and 0.250, respectively) further support the conclusion that there are no significant differences in the performance metrics among these models. Therefore, we do not have sufficient evidence to assert that any one model outperforms the others in a statistically meaningful way.

BRIE significantly outperforms state-of-the-art models such as IMINF and CACO-GNN on the YelpChi dataset, especially in terms of F1-macro and AUC scores. Its effectiveness is attributed to leveraging supervised

Table 4.3: YelpChi Dataset Statistical Tests

Wilcoxon Signed-Rank Test Results		
Comparison	Wilcoxon statistic	p-value
BRIE vs. IMINF	0.0	0.250
BRIE vs. CACO-GNN	0.0	0.250
IMINF vs. CACO-GNN	0.0	0.250

Paired t-test Results		
Comparison	t-statistic	p-value
BRIE vs. IMINF	3.351	0.787
BRIE vs. CACO-GNN	8.092	0.015
IMINF vs. CACO-GNN	4.168	0.053

Table 4.4: Amazon Dataset Statistical Tests

Wilcoxon Signed-Rank Test Results		
Comparison	Wilcoxon statistic	p-value
BRIE vs. IMINF	1.0	0.500
BRIE vs. CACO-GNN	0.0	0.250
IMINF vs. CACO-GNN	0.0	0.250

Paired t-test Results		
Comparison	t-statistic	p-value
BRIE vs. IMINF	-1.000	0.423
BRIE vs. CACO-GNN	2.803	0.107
IMINF vs. CACO-GNN	2.687	0.115

contrastive learning, integrated structural features, and synthetic fraud generation, addressing challenges like fraudster camouflage and class imbalance. On the Amazon dataset, while BRIE performs well, the differences compared to other models are not statistically significant, indicating a more competitive landscape for fraud detection on this dataset.

5 Conclusions and Future work

This thesis introduced BRIE, a Graph Neural Network (GNN)-based model aimed at improving fraud detection in multi-relation attributed networks. The motivation was to address the shortcomings of existing methods in detecting sophisticated fraud tactics in areas like financial services and e-commerce. Our approach involved using GNNs to capture complex relationships within networks, supervised contrastive learning to enhance node representation by maximizing intra-class similarity and minimizing inter-class similarity, and integrating structural features to improve the distinction between legitimate users and fraudsters. To tackle class imbalance, we enriched the training dataset with synthetic fraud nodes during the contrastive learning, enhancing the robustness and representativeness of the embeddings. Testing on two real-world datasets, Amazon and YelpChi, showcased significant improvements over state-of-the-art methods, particularly in the larger YelpChi dataset in key metrics (F1-score, AUC, Recall) compared to existing models, positioning BRIE as a promising approach for enhancing fraud detection in attributed networks.

Opportunities for further work include exploring the model's ability to dynamically adapt to evolving fraud tactics and patterns in real-time scenarios, enhancing its fraud detection capabilities in dynamic network environments. This could involve developing mechanisms for continuous learning, where BRIE updates its parameters and refines its fraud detection strategies based on new data and emerging fraud behaviors. Incorporating real-time data streams and feedback loops would enable the model to promptly identify and respond to novel fraud schemes, maintaining high accuracy and robustness. Additionally, integrating adaptive algorithms that can recognize shifts in fraud patterns and automatically adjust the model's focus could further improve its effectiveness.

Another promising direction is to enhance BRIE's interpretability by developing methods that provide clear insights into its decision-making processes. This would help users and analysts understand why certain transactions are flagged as fraudulent, facilitating more informed decision-making and trust in the model. These advancements are crucial for staying ahead of sophisticated fraudsters who continuously evolve their techniques to evade detection, ensuring that BRIE remains a cutting-edge tool in the fight against fraud in complex and ever-changing attributed networks.

Furthermore, applying BRIE to a wider range of domains beyond the YelpChi and Amazon datasets, such as financial transactions, healthcare records, and social media platforms, could demonstrate its versatility and robustness across various types of attributed networks. These advancements are crucial for staying ahead of sophisticated fraudsters who continuously evolve their techniques to evade detection, ensuring that BRIE remains a cutting-edge tool in the fight against fraud in complex and ever-changing attributed networks.

References

[1] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[2] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[3] L. Waikhom and R. Patgiri, "A survey of graph neural networks in various learning paradigms: methods, applications, and challenges," *Artificial Intelligence Review*, vol. 56, no. 7, pp. 6295–6364, 2023.

[4] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu, "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters," in *Proceedings of the 29th ACM international conference on information & knowledge management*, 2020, pp. 315–324.

[5] Y. Liu *et al.*, "Pick and Choose: A GNN-Based Imbalanced Learning Approach for Fraud Detection," in *Proceedings of the Web Conference 2021*, ser. WWW '21, Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 3168–3177.

[6] Z. Deng, G. Xin, Y. Liu, W. Wang, and B. Wang, "Contrastive graph neural network-based camouflaged fraud detector," *Information Sciences*, vol. 618, pp. 39–52, 2022.

[7] X. Wang, Z. Liu, J. Liu, and J. Liu, "Fraud detection on multi-relation graphs via imbalanced and interactive learning," *Information Sciences*, vol. 642, p. 119 153, 2023.

[8] S. Rayana and L. Akoglu, "Collective Opinion Spam Detection: Bridging Review Networks and Metadata," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15, Sydney, NSW, Australia: Association for Computing Machinery, 2015, pp. 985–994.

[9] M. Baumann, "Improving a rule-based fraud detection system with classification based on association rule mining," 2021.

[10] N. Mumic and P. Filzmoser, "A multivariate test for detecting fraud based on benford's law, with application to music streaming data," *Statistical Methods & Applications*, vol. 30, no. 3, pp. 819–840, 2021.

[11] V. R. Ganji and S. N. P. Mannem, "Credit card fraud detection using anti-k nearest neighbor algorithm," *International Journal on Computer Science and Engineering*, vol. 4, no. 6, pp. 1035–1039, 2012.

[12] H. P. Josyula, "Fraud detection in fintech leveraging machine learning and behavioral analytics," 2023.

[13] Y.-J. Zheng, X.-H. Zhou, W.-G. Sheng, Y. Xue, and S.-Y. Chen, "Generative adversarial network based telecom fraud detection at the receiving bank," *Neural Networks*, vol. 102, pp. 78–86, 2018.

[14] T. Pourhabibi, K.-L. Ong, B. H. Kam, and Y. L. Boo, "Fraud detection: A systematic literature review of graph-based anomaly detection approaches," *Decision Support Systems*, vol. 133, p. 113 303, 2020.

[15] Z. Wang, S. Gu, X. Zhao, and X. Xu, "Graph-based review spammer group detection," *Knowledge and Information Systems*, vol. 55, pp. 571–597, 2018.

[16] J. Ye and L. Akoglu, "Discovering opinion spammer groups by network footprints," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part I 15*, Springer, 2015, pp. 267–282.

[17] L. F. Carvalho, C. H. Teixeira, W. Meira, M. Ester, O. Carvalho, and M. H. Brandao, "Provider-consumer anomaly detection for healthcare systems," in *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, IEEE, 2017, pp. 229–238.

[18] S. Shehnepoor, M. Salehi, R. Farahbakhsh, and N. Crespi, "Netspam: A network-based spam detection framework for reviews in online social media," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 7, pp. 1585–1595, 2017.

[19] M. Jiang, P. Cui, A. Beutel, C. Faloutsos, and S. Yang, "Catchsync: Catching synchronized behavior in large directed graphs," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 941–950.

[20] B. Hooi, K. Shin, H. A. Song, A. Beutel, N. Shah, and C. Faloutsos, "Graph-based fraud detection in the face of camouflage," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 11, no. 4, pp. 1–26, 2017.

[21] D. Huang, D. Mu, L. Yang, and X. Cai, "Codetect: Financial fraud detection with anomaly feature detection," *IEEE Access*, vol. PP, pp. 1–1, Mar. 2018.

[22] S. Liu, B. Hooi, and C. Faloutsos, "Holoscope: Topology-and-spike aware fraud detection," ser. CIKM '17, Singapore, Singapore: Association for Computing Machinery, 2017, pp. 1539–1548.

[23] K. Shin, B. Hooi, J. Kim, and C. Faloutsos, "Densealert: Incremental dense-subtensor detection in tensor streams," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1057–1066.

[24] C. Merkwirth and T. Lengauer, "Automatic generation of complementary descriptors with molecular graph networks," *Journal of chemical information and modeling*, vol. 45, no. 5, pp. 1159–1168, 2005.

[25] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[26] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159,

[27] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[28] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[29] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[30] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[31] T. Pham, T. Tran, D. Phung, and S. Venkatesh, "Column networks for collective classification," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.

[32] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International conference on machine learning*, PMLR, 2018, pp. 5453–5462.

[33] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *arXiv preprint arXiv:1511.06391*, 2015.

[34] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *Advances in neural information processing systems*, vol. 31, 2018.

[35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[36] P. H. Le-Khac, G. Healy, and A. F. Smeaton, "Contrastive representation learning: A framework and review," *Ieee Access*, vol. 8, pp. 193 907–193 934, 2020.

[37] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 539–546 vol. 1.

[38] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[39] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, PMLR, 2020, pp. 1597–1607.

[40] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9729–9738.

[41] P. Khosla *et al.*, "Supervised contrastive learning," *Advances in neural information processing systems*, vol. 33, pp. 18 661–18 673, 2020.

[42] J. J. McAuley and J. Leskovec, "From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews," *CoRR*, vol. abs/1303.4402, 2013.

[43] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[44] J. Zhao, Y. Song, F. Liu, and Y. Deng, "The identification of influential nodes based on structure similarity," *Connection science*, vol. 33, no. 2, pp. 201–218, 2021.

[45] P. Wang, K. Han, X.-S. Wei, L. Zhang, and L. Wang, "Contrastive learning based hybrid networks for long-tailed image classification," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 943–952.

[46] S. Zhang, H. Yin, T. Chen, Q. V. N. Hung, Z. Huang, and L. Cui, *Gcn-based user representation learning for unifying robust recommendation and fraudster detection*, 2020.

[47] F. Shi and al., "H2-fdetector: A gnn-based fraud detector with homophilic and heterophilic connections," in *Proc. of the ACM Web Conference 2022*, 2022, pp. 1486–1494.

[48] P. Li, H. Yu, X. Luo, and J. Wu, "Lgm-gnn: A local and global aware memory-based graph neural network for fraud detection," *IEEE Transactions on Big Data*, 2023.

[49] M. Duan and al., "Dga-gnn: Dynamic grouping aggregation gnn for fraud detection," in *Proc. of the AAAI Conf. on Artificial Intelligence*, vol. 38, 2024, pp. 11 820–11 828.