Chapter 26
Graph Neural Networks in Anomaly Detection

Shen Wang, Philip S. Yu

Abstract Anomaly detection is an important task, which tackles the problem of discovering "different from normal" signals or patterns by analyzing a massive amount of data, thereby identifying and preventing major faults. Anomaly detection is applied to numerous high-impact applications in areas such as cyber-security, finance, e-commerce, social network, industrial monitoring, and many more mission-critical tasks. While multiple techniques have been developed in past decades in addressing unstructured collections of multi-dimensional data, graph-structure-aware techniques have recently attracted considerable attention. A number of novel techniques have been developed for anomaly detection by leveraging the graph structure. Recently, graph neural networks (GNNs), as a powerful deep-learning-based graph representation technique, has demonstrated superiority in leveraging the graph structure and been used in anomaly detection. In this chapter, we provide a general, comprehensive, and structured overview of the existing works that apply GNNs in anomaly detection.

26.1 Introduction

In the era of machine learning, sometimes, what stands out in the data is more important and interesting than the normal. This branch of task is called *anomaly detection*, which concentrates on discovering "different from normal" signals or patterns by analyzing a massive amount of data, thereby identifying and preventing major faults. This task plays a key on in several high-impact domains, such as cyber-security (network intrusion or network failure detection, malicious program

Shen Wang
Department of Computer Science, University of Illinois at Chicago, e-mail: swang224@uic.edu

Philip S. Yu
Department of Computer Science, University of Illinois at Chicago, e-mail: psyu@uic.edu

detection), finance (credit card fraud detection, malicious account detection, cashout user detection, loan fraud detection), e-commerce (reviews spam detection), social network (key player detection, anomaly user detection, real money trading detection), and industrial monitoring (fault detection).

In the past decades, many techniques have been developed for anomaly detection by leveraging the graph structure, a.k.a. graph-based anomaly detection. Unlike non-graph anomaly detection, they further take the inter-dependency among each data instance into consideration, where data instances in a wide range of disciplines, such as physics, biology, social sciences, and information systems, are inherently related to one another. Compare to the non-graph-based method, the performance of the graph-based method is greatly improved. Here, we provide an illustrative example of malicious program detection in the cyber-security domain in Figure 26.1. In a phishing email attack as shown in Figure 1, to steal sensitive data from the database of a computer/server, the attacker exploits a known venerability of Microsoft Office by sending a phishing email attached with a malicious .doc file to one of the IT staff of the enterprise. When the IT staff member opens the attached .doc file through the browser, a piece of a malicious macro is triggered. This malicious macro creates and executes a malware executable, which pretends to be an open-source Java runtime (Java.exe). This malware then opens a backdoor to the adversary, subsequently allowing the adversary to read and dump data from the target database via the affected computer. In this case, signature-based or behavior-based malware detection approaches generally do not work well in detecting the malicious program in our example. As the adversary can make the malicious program from scratch with binary obfuscation, signature-based approaches would fail due to the lack of known malicious signatures. Behavior-based approaches may not be effective unless the malware sample has previously been used to train the detection model. It might be possible to detect the malicious program using existing host-level anomaly detection techniques. These host-based anomaly detection methods can locally extract patterns from process events as the discriminators of abnormal behavior. However, such detection is based on observations of single operations, and it sacrifices the false positive rate to detect the malicious program. For example, the host-level anomaly detection can detect the fake "Java.exe" by capturing the database read. However, a Java-based SQL client may also exhibit the same operation. If we simply detect the database read, we may also classify normal Java-based SQL clients as abnormal program instances and generate false positives. In the enterprise environment, too many false positives can lead to the alert fatigue problem, causing cyber-analysts to fail to catch up with attacks. To accurately separate the database read of the malicious Java from the real Java instances, we need to consider the higher semantic-level context of the two Java instances. As shown in Figure 26.1, malicious Java is a very simple program and directly accesses the database. On the contrary, a real Java instance has to load a set of .DLL files in addition to the database read. By comparing the behavior graph of the fake Java instance with the normal ones, we can find that it is abnormal and precisely report it as a malicious program instance. Thus, leveraging the graph helps to identify the anomaly data instances.

Specifically, the benefit of graph-based method is four-folded:

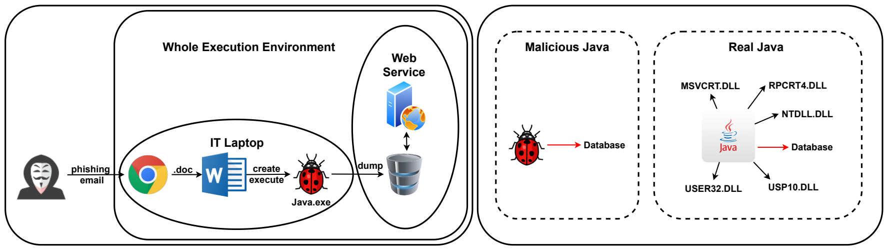

Figure 26.1: An illustrative example of malicious program detection in the cyber-security domain. The left side shows an example of a phishing email attack: the hacker creates and executes a malware executable, which pretends to be an open-source Java runtime (Java.exe); this malware then opens a backdoor to the adversary, subsequently allowing the adversary to read and dump data from the target database via the affected computer. The right side demonstrates the behavior graph of the malicious Java.exe vs. normal Java runtime.

- *Inter-dependent Property* – Data instances in a wide range of disciplines, such as physics, biology, social sciences, and information systems, are inherently related to one another and can form a graph. These graph structures can provide additional side information to identify the anomalies in addition to the attributes of each data instance.
- *Relational Property* – The anomaly data instances sometimes can exhibit themselves as relational, e.g., in the fraud domain, the context of the anomaly data instance has a high probability of being abnormal; the anomaly data instance is closely related to a group of data instances. If we identify one anomaly data instance in the graph, some other anomaly data instances based on it can be detected.
- *Fruitful Data Structure* – The graph is a data structure encoding fruitful information. The graph consists of nodes and edges, enabling the incorporation of node and edge attributes/types for anomaly data instance identification. Besides, multiple paths exist between each pair of data instances, which allows the relation extraction in different ranges.
- *Robust Data Structure* – The graph is a more adversarially robust data structure, e.g., attacker or fraudster usually can only attack or fraud the specific data instance or its context and have a limited global view of the whole graph. In this case, the anomaly data instance is harder to fit into the graph as well as possible.

Recent years have witnessed a growing interest in developing deep-learning-based algorithms on the graph, including unsupervised methods (Grover and Leskovec, 2016; Liao et al, 2018; Perozzi et al, 2014) and supervised methods (Wang et al, 2016, 2017e; Hamilton et al, 2017b; Kipf and Welling, 2017b; Veličković et al, 2018). Among these deep-learning-based algorithms on the graph, the graph neural networks (GNNs) (Hamilton et al, 2017b; Kipf and Welling, 2017b; Veličković et al, 2018), as powerful deep graph representation learning techniques, have demonstrated superiority in leveraging the graph structure. The basic idea is to aggregate information from local neighborhoods in order to combine the content feature and graph structures to learn the new graph representation. In particular, GCN (Kipf and Welling, 2017b) leverages the "graph convolution" operation to aggregate the feature of one-hop neighbors and propagate multiple-hop information via the iterative "graph convolution". GraphSage (Hamilton et al, 2017b) develops the graph neural network in an inductive setting, which performs neighborhood sampling and aggregation to generate new node representation efficiently. GAT (Veličković et al, 2018) further incorporates attention mechanism into GCN to perform the attentional aggregation of the neighborhoods. Given the importance of graph-based anomaly detection and the success of graph neural networks, both academia and industry are interested in applying GNNs to tackle the problem of anomaly detection. In recent years, some researchers have successfully applied GNNs in several important anomaly detection tasks. In this book chapter, we summarize different GNN-based anomaly detection approaches and provide taxonomies for them according to various criteria. Despite the more than 10+ papers published in the last three years, several challenges remain unsolved until now, which we summarize and introduce in this chapter as below.

- *Issues* Unlike GNNs applications in other domains, the GNNs applications in anomaly detection have several unique issues, which comes from data, task, and model. We briefly discuss and summarize them to provide a comprehensive understanding of the difficulties of the problems.
- *Pipeline* There are various GNN-based anomaly detection works. It is challenging and time-consuming to understand the big pictures of all these works. To facilitate an easy understanding of existing research on this line, we summarize the general pipeline of GNN-based anomaly detection approaches.
- *Taxonomy* There are already several works in the domain of GNN-based anomaly detection. Compared with other GNN applications, GNN-based anomaly detection is more complicated due to unique challenges and problem definitions. To provide a quick understanding of the similarity and differences between existing works, we list some representative works and summarize novel elaborated taxonomies according to various criteria. We present our taxonomy with more details in Table 1.
- *Case Studies* We provide the case studies of some representative GNN-based anomaly detection approaches.

The rest of this chapter is organized as follows. Section 2 discusses and summarizes the issues of the GNN-based anomaly detection. Section 3 provides the unified pipeline of the GNN-based anomaly detection. Section 4 provides the taxonomies of existing GNN-based anomaly detection approaches. Section 5 provides the case studies of some representative GNN-based anomaly detection approaches. In the last section, we provide the discussion and future directions.

26.2 Issues

In this section, we provide a brief discussion and summary of the issues in GNN-based anomaly detection. In particular, we group them into three: (i) data-specific issues, (ii) task-specific issues, and (iii) model-specific issues.

26.2.1 Data-specific issues

As the anomaly detection systems usually work with real-world data, they demonstrate high volume, high dimensionality, high heterogeneity, high complexity, and dynamic property.

High Volume – With the advance of information storage, it is much easier to collect large amounts of data. For example, in an e-commerce platform like Xianyu, there are over 1 billion second-hand goods published by over ten millions users; in an enterprise network monitoring system, the system event data collected from a single computer system in one day can easily reach 20 GB, and the number of events related to one specific program can easily reach thousands. It is prohibitively expensive to perform the analytic task on such massive data in terms of both time and space.

High Dimensionality – Also, benefit from the advance of the information storage, rich amount of information is collected. It results in high dimensionality of the attributes for each data instance. For example, in an e-commerce platform like Xianyu, different types of attributes are collected for each data instance, such as user demographics, interests, roles, as well as different types of relations; in an enterprise network monitoring system, each collected system event is associated with hundreds of attributes, including information of involved system entities and their relationships, which causes the curse of dimensionality.

High Heterogeneity – As rich types of information are collected, it results in high heterogeneity of the attributes for each data instance: the feature of each data instance can be multi-view or multi-sourced. For example, in an e-commerce platform like Xianyu, multiple types of data are collected from the user, such as personal profile, purchase history, explore history, and so on. Nevertheless, multi-view data like social relations and user attributes have different statistical properties. Such heterogeneity poses a great challenge to integrate multi-view data.

High Complexity – As we can collect more and more information, the collected data is complex in content: it can be categorical or numerical, which increases the difficulty of leveraging all the contents jointly.

Dynamic Property – The data collection is usually conducted every day or continuously. For example, billions of credit card transactions are performed every day; billions of click-through traces of web users are generated each day. This kind of data can be thought of as streaming data, and it demonstrates dynamic property.

The above data-specific issues are general and apply to all kinds of data. So we also need to discuss the graph-data-specific issues, including relational property, graph heterogeneity, graph dynamic, variety of definitions, lack of intrinsic distance/similarity metrics, and search space size.

Relational Property – The relational property of the data makes it challenging to quantify the anomalousness of graph objects. While in traditional outlier detection, the objects or data instances are treated as independent and identically distributed (i.i.d.) from each other, the data instances in graph data are pair-wise correlated. Thus, the "spreading activation" of anomalousness or "guilt by associations" needs to be carefully accounted for. For example, the cash-out users not only have abnormal features, but also behavior abnormally in interaction relations. They may simultaneously have many transactions and fund transfer interactions with particular merchants, which is hard to be exploited by traditional feature extraction.

Graph Heterogeneity – Similar to the general data-specific issues of high heterogeneity, the graph instance type, and relation type are usually heterogeneous. For example, in a computer system graph, there are three types of entities: process (P), file (F), and INETSocket (I) and multiple types of relations: a process forking another process (P$\rightarrow$P), a process accessing a file (P$\rightarrow$F), a process connecting to an Internet socket (P$\rightarrow$I), and so on. Due to the heterogeneity of entities (nodes) and dependencies (edges) in a heterogeneous graph, the diversities between different dependencies vary dramatically, significantly increasing the difficulty of jointly leveraging these nodes and edges.

Graph Dynamic – As the data are collected periodically or continuously, the constructed graph also demonstrates the dynamic property. It is challenging to detect the anomaly due to its dynamic nature. Some anomalous operations show some explicit patterns but try to hide them in a large graph, while others are with implicit patterns. Take an explicit anomaly pattern in a recommender system as an example. As anomalous users usually control multiple accounts to promote the target items, the edges between these accounts and items may compose a dense subgraph, which emerges in a short time period. In addition, although the accounts which involve the anomaly perform anomalous operations sometimes, these accounts perform normally most of the time, which hides their long-term anomalous behavior and increases the difficulty of detection.

Variety of Definitions – The definitions of anomalies in graphs are much more diverse than in traditional outlier detection, given the rich representation of graphs. For example, novel types of anomalies related to graph substructures are of interest for many applications, e.g., money-laundering rings in trading networks.

Lack of Intrinsic Distance/Similarity Metrics – The intrinsic distance/similarity metrics are not clear. For example, in real computer systems, given two programs with thousands of system events related to them, it is a difficult task to measure their distance/similarity.

Search Space Size – The main issue associated with more complex anomalies such as graph substructures is that the search space is huge, as in many graph-theoretical problems associated with graph search. The enumeration of possible substructures is combinatorial, making the problem of finding out the anomalies a much harder task. This search space is enlarged even more when the graphs are attributed as the possibilities span both the graph structure and the attribute space.

As a result, the graph-based anomaly detection algorithms need to be designed not only for effectiveness but also for efficiency and scalability.

26.2.2 Task-specific Issues

Due to the unique characteristics of the anomaly detection task, the issues also come from the problems, including labels quantity and quality, class imbalance and asymmetric error, and novel anomalies.

Labels Quantity and Quality – The major issue of anomaly detection is that the data often has no or very few class labels. It is unknown which data is abnormal or normal. Usually, it is costly and time-consuming to obtain ground-truth labels from the domain expert. Moreover, due to the complexity of the data, the produced label may be noisy and biased. Therefore, this issue limits the performance of the supervised machine learning algorithm. What is more, the lack of true clean labels, i.e., ground truth data, also makes the evaluation of anomaly detection techniques challenging.

Class Imbalance and Asymmetric Error – Since the anomalies are rare and only a small fraction of the data is excepted to be abnormal, the data is extremely imbalanced. Moreover, the cost of mislabeling a good data instance versus a bad instance may change depending on the application and further could be hard to estimate beforehand. For example, mis-predicting a cash-out fraudster as a normal user is essentially harmful to the whole financial system or even the national security, while mis-predicting a normal user as a fraudster could cause customer loss fidelity. Therefore, the class imbalance and asymmetric error affect the machine-learning-based method seriously.

Novel Anomalies – In some domain, such as fraud detection or malware detection, the anomalies are created by the human. They are created by analyzing the detection system and designed to be disguised as a normal instance to bypass the detection. As a result, not only should the algorithms be adaptive to changing and growing data over time, they should also be adaptive to and be able to detect novel anomalies in the face of adversaries.

26.2.3 Model-specific Issues

Apart from data-specific and task-specific issues, it is also challenging to apply the graph neural network directly to anomaly detection task sdue to its unique model properties, such as homogeneous focus and vulnerability.

Homogeneous Focus – Most graph neural network models are designed for homogeneous graph, which considers a single type of nodes and edges. In many real-world applications, data can be naturally represented as heterogeneous graphs. However, traditional GNNs treat different features equally. All the features are mapped and propagated together to get the representations of nodes. Considering that the role of each node is just a one-dimensional feature in the high dimensional feature space, there exist more features that are not related to the role, e.g., age, gender, and education. Thus the representation of applicants with neighbors of different roles has no distinction in representation space after neighbor aggregation, which causes the traditional GNNs to fail.

Vulnerability – Recently theoretical studies prove the limitations and vulnerabilities of GNNs, when graphs have noisy nodes and edges. Therefore, a small change to the node features may cause a dramatic performance drop and failing to tackle the camouflage, where fraudsters would sabotage the GNN-based fraud detectors.

26.3 Pipeline

In this section, we introduce the standard pipeline of the GNN-based anomaly detection. Typically, GNN-based anomaly detection methods consist of three important components, including graph construction and transformation, graph representation learning, and prediction.

26.3.1 Graph Construction and Transformation

As discussed in the previous section, a real-world anomaly detection system has some data-specific issues. Therefore, it requires data analysis on the raw data to address them. Then the graph can be constructed to capture the complex interactions and eliminate the data redundancies. Based on the type of the data instance and relations, the graph can be constructed as a homogeneous graph or heterogeneous graph, where a homogeneous graph only has a single-typed data instance and relation, and a heterogeneous graph has multi-typed data instances and relations. Based on the availability of the timestamp, the graph can be constructed as a static graph or a dynamic graph, where a static graph refers to the graph that has fixed nodes and edges, and a dynamic graph refers to the graph that has nodes and/or edges change over time. Based on the availability of the node and/or edge attributes, the constructed graph can be a plain graph or an attributed graph, where the plain graph only contains the structure information and the attributed graph has attributes on nodes and/or edges.

When the constructed graph is heterogeneous, simply aggregating neighbors cannot capture the semantic and structural correlations among different types of entities. To address the graph heterogeneity issue, a graph transformation is performed to transform the heterogeneous graph to a multi-channel graph guided by the meta-paths, where a meta-path (Sun et al, 2011) is a path that connects entity types via a sequence of relations over a heterogeneous network. For example, in a computer system, a meta-path can be the system events (P$\rightarrow$P, P$\rightarrow$F, and P$\rightarrow$I), with each one defining a unique relationship between two entities. The multi-channel graph is a graph with each channel constructed via a certain type of meta-path. Formally, given a heterogeneous graph $\mathscr{G}$ with a set of meta-paths $\mathscr{M} = \{M_1, ..., M_{|\mathscr{M}|}\}$, the transformed multi-channel network $\hat{\mathscr{G}}$ is defined as follows:

$$\hat{\mathscr{G}} = \{\mathscr{G}_i | \mathscr{G}_i = (\mathscr{V}_i, \mathscr{E}_i, A_i), i = 1, 2, ..., |\mathscr{M}|)\} \tag{26.1}$$

where $\mathscr{E}_i$ denotes the homogeneous links between the entities in $\mathscr{V}_i$, which are connected through the meta-path M_i. Each channel graph $\mathscr{G}_i$ is associated with an adjacency matrix A_i. $|\mathscr{M}|$ indicates the number of meta-paths. Notice that the potential meta-paths induced from the heterogeneous network can be infinite, but not everyone is relevant and useful for the specific task of interest. Fortunately, there are some algorithms (Chen and Sun, 2017) proposed recently for automatically selecting the meta-paths for particular tasks.

26.3.2 Graph Representation Learning

After the graph is constructed and transformed, graph representation learning is conducted to get the proper new representation of the graph. Generally GNNs are built by stacking seven types of basic operations, including neural aggregator function $AGG()$, linear mapping function $MAP_{linear}()$, nonlinear mapping function $MAP_{nonlinear}()$, multilayer perceptron function $MLP()$, feature concatenation $CONCAT()$, attentional feature fusion $COMB_{att}$, and readout function $Readout()$. Among these operations, linear mapping function, nonlinear mapping function, multilayer perceptron function, feature concatenation, and attentional feature fusion are typical operations used in traditional deep learning algorithms. Their formal descriptions are described as follows:

Linear Mapping Function $MAP_{linear}()$:

$$MAP_{linear}(\mathbf{x}) = \mathbf{W}\mathbf{x} \tag{26.2}$$

where $\mathbf{x}$ is the input feature vector, and $\mathbf{W}$ is the trainable weight matrix.

Nonlinear Mapping Function $MAP_{nonlinear}()$:

$$MAP_{nonlinear}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x}) \tag{26.3}$$

where $\mathbf{x}$ is the input feature vector, $\mathbf{W}$ is the trainable weight matrix, and $\sigma()$ represents the non-linear activation function.

Multilayer Perceptron Function $MLP()$:

$$MLP(\mathbf{x}) = \sigma(\mathbf{W^k} \cdots \sigma(\mathbf{W^1}\mathbf{x})) \tag{26.4}$$

where $\mathbf{x}$ is the input feature vector, $\mathbf{W^i}$ with $i = 1,...,k$ is the trainable weight matrix, k indicates the number of layers, and $\sigma()$ represents the non-linear activation function.

Feature Concatenation $CONCAT()$:

$$CONCAT(\mathbf{x}_1, \cdots \mathbf{x}_n) = [\mathbf{x}_1, \cdots \mathbf{x}_n] \tag{26.5}$$

where n indicates the number of the features.

Attentional Feature Fusion $COMB_{att}()$:

$$COMB_{att}(\mathbf{x}_1, \cdots \mathbf{x}_n) = \sum_{i=1}^{n} softmax(\mathbf{x}_i)\mathbf{x}_i \tag{26.6}$$

$$softmax(\mathbf{x}_i) = \frac{\exp(MAP(\mathbf{x}_i))}{\sum_{j=1}^{n} \exp(MAP(\mathbf{x}_j))} \tag{26.7}$$

where $MAP()$ can be linear or nonlinear.

Different from traditional deep learning algorithm, the GNNs have its unique operation–neural aggregation function $AGG()$. Based on the level of object to aggregate, it can be categorized into three specific types: node-wise neural aggregator $AGG_{node}()$, layer-wise neural aggregator $AGG_{layer}()$, and path-wise neural aggregator $AGG_{path}()$.

Node-wise Neural Aggregator $AGG_{node}()$ is the GNN module that aims to aggregate the node neighborhoods, which can be described as follows,

$$\mathbf{h}_v^{(i)(k)} = AGG_{node}(\mathbf{h}_v^{(i)(k-1)}, \{\mathbf{h}_u^{(i)(k-1)}\}_{u \in \mathcal{N}_v^i}) \tag{26.8}$$

where i is meta-path (relation) indicator, $k \in \{1, 2, ...K\}$ is the layer indicator, $\mathbf{h}_v^{(i)(k)}$ is the feature vector of node v for relation M_i at the k-th layer, $\mathcal{N}_v^i$ indicates the neighbourhoods of node v under the relation M_i. Based on the way the the node neighborhoods are aggregated, typically, the node-level neural aggregator can be GCN $AGG^{GCN}()$ (Kipf and Welling, 2017b), GAT $AGG^{GAT}()$ (Veličković et al, 2018) or Message-Passing $AGG^{MPNN}()$ (Gilmer et al, 2017). For the GCN and GAT, the formulations can be described by Equation 8. While for the Message-Passing, the edges are also used during the node-level aggregation. Formally, it can be described as follows,

$$\mathbf{h}_v^{(i)(k)} = AGG_{node}(\mathbf{h}_v^{(i)(k-1)}, \{\mathbf{h}_v^{(i)(k-1)}, \mathbf{h}_u^{(i)(k-1)}, \mathbf{h}_{vu}^{(i)(k-1)}\}_{u \in \mathcal{N}_v^i}) \tag{26.9}$$

where $\mathbf{h}_{vu}^{(i)(k-1)}$ denotes the edge embedding between the target node v and its neighbor node u, and $\{\}$ indicates a fusion function to combine the target node, its neighbor node and the corresponding edge between them.

Layer-wise Neural Aggregator $AGG_{layer}()$ is the GNN module that aims to aggregate the context information from different hops. For example, if layer number $k = 2$, the GNN gets 1-hop neighborhood information, and if layer number

$k = K + 1$, the GNN gets K-hop neighborhood information. The larger the k is, the more global information the GNN obtains. Formally, this function can be described as follows,

$$\mathbf{l}_v^{(i)(k)} = AGG_{layer}(\mathbf{l}_v^{(i)(k-1)}, \mathbf{h}_v^{(i)(k)}) \tag{26.10}$$

where $\mathbf{l}_v^{(i)(k)}$ is the aggregated representation of $(k-1)-$hop neighborhood node v for relation M_i at the k-th layer.

Path-wise Neural Aggregator $AGG_{layer}()$ is the GNN module that aims to aggregate the context information from different relations. Generally, the relation can be described by meta-path (Sun et al, 2011) based contextual search. Formally, this function can be described as follows,

$$\mathbf{p}_v^{(i)} = \mathbf{l}_v^{(i)(K)} \tag{26.11}$$

$$\mathbf{p}_v = AGG_{path}(\mathbf{p}_v^{(1)}, ...\mathbf{p}_v^{(|\mathscr{M}|)}) \tag{26.12}$$

where $\mathbf{p}_v^{(i)}$ is the aggregated final layer representation of node v for relation M_i.

Then the final node representation is described by the fusion representation from different meta-paths (relations) as follows,

$$\mathbf{h}_v^{(final)} = \mathbf{p}_v \tag{26.13}$$

Based on the task, we can also compute the graph representation by performing readout function $Readout()$ to aggregate all the nodes' final representations, which can be described as follows,

$$\mathbf{g} = Readout(\mathbf{h}_{v_1}^{(final)}, ...\mathbf{h}_{v_V}^{(final)}) \tag{26.14}$$

Typically, we can obtain different levels of graph representations, including node-level, edge-level, and graph-level. The node-level and edge-level representation are the most preliminary representations, which can be learned via graph neural network. The graph-level representation is a higher-level representation, which can be obtained by performing the readout function to the node-level and edge-level representations. Based on the target of the task, the specific level of graph representations is fed to the next stage.

26.3.3 Prediction

After the graph representation is learned, they are fed to the prediction stage. Depends on the task and the target label, there are two types of prediction: classification and matching. In the classification-based prediction, it assumes that enough labeled anomaly data instances are provided. A good classifier can be trained to identify if the given graph target is abnormal or not. As mentioned in the issues section, there might be no or few anomaly data instances. In this case, the matching-based prediction is usually used. If there are very few anomaly samples, we learn the representation of them, and when the candidate sample is similar to one of the anomaly samples, an alarm is triggered. If there is no anomaly sample, we learn the representation of the normal data instance. When the candidate sample is not similar to any of the normal samples, an alarm is triggered.

26.4 Taxonomy

In this section, we provide the taxonomies of existing GNN-based anomaly detection approaches. Due to the variety of graph data and anomalies, the GNN-based anomaly detection can have multiple taxonomies. Here we provided four types of taxonomy in order to give a quickly understand of the similarity and difference between existing works, including static/dynamic graph taxonomy, homogeneous/heterogeneous graph taxonomy, plain/attributed graph taxonomy, object taxonomy, and task taxonomy.

In **task taxonomy**, the exiting works can be categorized into GNN-based anomaly detection in financial networks, GNN-based anomaly detection in computer networks, GNN-based anomaly detection in telecom networks, GNN-based anomaly detection in social networks, GNN-based anomaly detection in opinion networks, and GNN-based anomaly detection in sensor networks.

In **anomaly taxonomy**, the existing works can be categorized into node-level anomaly detection, edge-level anomaly detection, and graph-level anomaly detection.

In **static/dynamic graph taxonomy**, the existing works can be categorized into static GNN-based anomaly detection and dynamic GNN-based anomaly detection.

In **homogeneous/heterogeneous graph taxonomy**, the exiting works can be categorized into homogeneous GNN-based anomaly detection and heterogeneous GNN-based anomaly detection.

In **plain/attributed graph taxonomy**, the exiting works can be categorized into plain GNN-based anomaly detection and attributed GNN-based anomaly detection.

In **object taxonomy**, the exiting works can be categorized into: classification-based approach and matching-based approach.

We present our taxonomy with more details in Table 1.

26.5 Case Studies

In this section, we provide the case studies to give the details of some representative GNN-based anomaly detection approaches.

Table 26.1: Summary of GNN-based anomaly detection approaches.

Approach	Year	Venue	Task	Anomaly	Static Dynamic	Homogeneous Heterogeneous	Plain Attributed	Model	Object
GEM (Liu et al, 2018f)	2018	CIKM	Malicious Account Detection	Node	Static	Heterogeneous	Attributed	GCN, Attention$_{(path)}$	Classification
HACUD (Hu et al, 2019b)	2019	AAAI	Cashout User Detection	Node	Static	Heterogeneous	Attributed	GCN, Attention$_{(feature,path)}$	Classification
DeepHGNN (Wang et al, 2019h)	2019	SDM	Malicious Program Detection	Node	Static	Heterogeneous	Attributed	GCN, Attention$_{(path)}$	Classification
MatchGNet (Wang et al, 2019i).	2019	IJCAI	Malicious Program Detection	Graph	Static	Heterogeneous	Attributed	GCN, Attention$_{(node,layer,path)}$	Matching
AddGraph (Zheng et al, 2019)	2019	IJCAI	Malicious Connection Detection	Edge	Dynamic	Homogeneous	Plain	GCN, GRU$_{att}$	Matching
SemiGNN (Wang et al, 2019b)	2019	ICDM	Malicious Account Detection	Node	Static	Heterogeneous	Attributed	GCN, Attention$_{(node,path)}$	Classification, Matching
MVAN (Tao et al, 2019)	2019	KDD	Real Money Trading Detection	Node	Static	Heterogeneous	Attributed	GAT, Attention$_{(path,view)}$	Classification
GAS (Li et al, 2019a)	2019	CIKM	Spam Detection	Edge	Static	Heterogeneous	Attributed	MPNN, Attention$_{(message)}$	Classification
iDetective (Zhang et al, 2019a)	2019	CIKM	Key Player Detection	Node	Static	Heterogeneous	Attributed	GCN, Attention$_{(path)}$	Classification
GAL (Zhao et al, 2020f)	2020	CIKM	Anomaly User Detection	Node	Static	Homogeneous	Attributed	GCN/GAT	Matching
CARE-GNN (Dou et al, 2020)	2020	CIKM	Fraud Detection	Node	Static	Heterogeneous	Attributed	GCN, Attention$_{(node)}$	Classification

26.5.1 Case Study 1: Graph Embeddings for Malicious Accounts Detection

Graph embeddings for malicious accounts detection (GEM) (Liu et al, 2018f) is the first attempt to apply the GNN to anomaly detection. The aim of GEM is to detect the malicious account at Alipay pay, a mobile cashless payment platform.

The graph constructed from the raw data is static and heterogeneous. The construed graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ consists of 7 types of nodes, including account typed nodes (U) and 6 types of device typed nodes (phone number (PN), User Machine ID (UMID), MAC address (MACA), International Mobile Subscriber Identity (IMSI), Alipay Device ID (APDID) and a random number generated via IMSI and IMEI (TID), such that $\mathscr{V} = U \cup PN \cup UMID \cup MACA \cup IMSI \cup APDID \cup TID$. To overcome the heterogeneous graph challenge and make GNN applicable to the graph, through graph transformation, GEM constructs a 6-channel graph $\hat{\mathscr{G}} = \{\mathscr{G}_i | \mathscr{G}_i = (\mathscr{V}_i, \mathscr{E}_i, A_i), i = 1, 2, ..., |\mathscr{M}|\}$ with $|\mathscr{M}| = 6$. In particular, 6 types of edges are specifically modeled to capture the edge heterogeneity, e.g., account connects phone number ($U \to PN$), account connects UMID ($U \to UMID$), account connects MAC address ($U \to MACA$), account connects IMSI ($U \to IMSI$), account connects Alipay Device ID ($U \to APDID$) and account connects TID ($U \to TID$). As the activity attributes are constructed, the constructed graph is an attributed graph. After the graphs are constructed and transformed, GEM performs a graph convolutional network to aggregate the neighborhood on each channel graph. As each channel graph is treated as a homogeneous graph corresponding to a specific relation, GNN can be directly applied to each channel graph.

During the graph representation learning stage, the node aggregated representation $\mathbf{h}_v^{(i)(k)}$ is computed by performing a GCN aggregator $AGG^{GCN}()$. To get the path aggregated representation, it adopts the attentionally feature fusion to fuse the node aggregated representation obtained in each channel graph $\mathscr{G}^i$. Besides, an activity feature for each node is constructed, and it adds the linear mapping of this activity feature to the attentional feature fusion of the path aggregated representations. Formally, the GNN operations can be described as follow.
Node-wise aggregation:

$$\begin{aligned}\mathbf{h}_v^{(i)(k)} &= AGG_{node}(\mathbf{h}_v^{(i)(k-1)}, \{\mathbf{h}_u^{(i)(k-1)}\}_{u \in \mathscr{N}_v^i}) \\ &= AGG^{GCN}(\mathbf{h}_v^{(i)(k-1)}, \{\mathbf{h}_u^{(i)(k-1)}\}_{u \in \mathscr{N}_v^i})\end{aligned} \tag{26.15}$$

Path-wise aggregation:

$$\mathbf{p}_v^{(k)} = MAP_{linear}(\mathbf{x}_v) + COMB_{att}(\mathbf{h}_v^{(1)(k))}, ..., \mathbf{h}_v^{(|\mathscr{M}|)(k)}) \tag{26.16}$$

Layer-wise aggregation:

$$\mathbf{l}_v^{(K)} = \mathbf{p}_v^{(K)} \tag{26.17}$$

Final node representation:

$$\mathbf{h}_v^{(final)} = \mathbf{l}_v^{(K)} \tag{26.18}$$

where K indicates the number of the layers.

The object of GEM is classification. It feeds the learned account node embedding to a standard logistic loss function.

26.5.2 Case Study 2: Hierarchical Attention Mechanism based Cash-out User Detection

Hierarchical attention mechanism based cash-out user detection (HACUD) (Hu et al, 2019b) applied the GNN to the fraud user detection at Credit Payment Services platform, where the fraud user performs the cash-out fraud, that pursues cash gains with illegal or insincere intent.

HACUD also constructs a static heterogeneous graph from the raw data. Specifically, it consists of multiple types of nodes (i.e., User (U), Merchant (M), Device (D)) with rich attributes and relations (i.e., fund transfer relation between users and transaction relation between users and merchants). Different from the way GEM deal with the graph heterogeneity issues, during the graph transformation stage, HACUD only models the user nodes and considers two specific types of meta-paths (relations) between pairwise of users, including User-(fund transfer)-User (UU) and User-(transaction)-Merchant-(transaction)-User (UMU) and constructs a 2-channel graph, such that $\hat{\mathscr{G}} = \{\mathscr{G}_i | \mathscr{G}_i = (\mathscr{V}_i, \mathscr{E}_i, A_i), i = 1, ..., |\mathscr{M}|\}$ with $|\mathscr{M}| = 2$ and $\mathscr{V}_i \in U$.

The two selected meta-paths capture different semantics. For example, the UU path connects users having fund transfers from one to another, while the UMU connects users having transactions with the same merchants. Then each channel graph is homogeneous and can work with GNN directly. As the user attributes are available, the constructed graph is attributed.

In the graph representation stage, the node-wise aggregation is performed to each channel graph via a convolutional graph network. Different from GEM (Liu et al, 2018f), it adds and joins the user feature $\mathbf{x}_v$ to the aggregated node representation in an attentional way. Then the node-wise aggregation extends to a 3-step procedure, including (a) initial node-wise aggregation, (b) feature fusion, and (c) feature attention. After the initial aggregated node representation $\tilde{\mathbf{h}}_v^{(i)}$ is computed vis GCN $AGG^{GCN}()$, it is fused with user feature $\mathbf{x}_v$ through a feature fusion. Next, it performs the feature attention. Since only 1-hop neighborhoods are considered, there is no layer-wise aggregation, and the final node-wise aggregated representations $\mathbf{h}_v^{(i)}$ are fed to the path-wise aggregation directly. Formally, it can be described as follows,

Node-wise aggregation:

(a)*Initial node-wise aggregation*:

$$\begin{aligned}\tilde{\mathbf{h}}_v^{(i)} &= AGG_{node}(\mathbf{h}_v^{(i)}, \{\mathbf{h}_u^{(i)}\}_{u \in \mathcal{N}_v^i}) \\ &= AGG^{GNN}(\mathbf{h}_v^{(i)}, \{\mathbf{h}_u^{(i)}\}_{u \in \mathcal{N}_v^i})\end{aligned} \tag{26.19}$$

(b)*Feature fusion*:

$$\mathbf{f}_v^{(i)} = MAP_{nonlinear}(CONCAT(MAP_{linear}(\tilde{\mathbf{h}}_v^{(i)}), MAP_{linear}(\mathbf{x}_v))) \tag{26.20}$$

(c)*Feature attention*:

$$\alpha_v^{(i)} = MAP_{nonlinear}(MAP_{nonlinear}(CONCAT(MAP_{linear}(\mathbf{x}_v), \mathbf{f}_v^{(i)})) \tag{26.21}$$

$$\mathbf{h}_v^{(i)} = softmax(\alpha_v^{(i)}) \bigodot \mathbf{f}_v^{(i)} \tag{26.22}$$

Path-wise aggregation:

$$\begin{aligned}\mathbf{p}_v &= AGG_{path}(\mathbf{h}_v^{(0)}, \mathbf{h}_v^{(1)}) \\ &= COMB_{att}(\mathbf{h}_v^{(0)}, \mathbf{h}_v^{(1)})\end{aligned} \tag{26.23}$$

Final node representation:

$$\mathbf{h}_v^{(final)} = MLP(\mathbf{p}_v) \tag{26.24}$$

where $\bigodot$ denotes the element-wise product. As only one-hop information is used, there is no layer indicator k.

As same as GEM, the object of HACUD is classification. It feeds the learned user node embedding to a standard logistic loss function.

26.5.3 Case Study 3: Attentional Heterogeneous Graph Neural Networks for Malicious Program Detection

Attentional heterogeneous graph neural network for malicious program detection (DeepHGNN) (Wang et al, 2019h) applied the GNN to the malicious program detection in a computer system of an enterprise network.

The raw data is a large volume of system behavioral data with rich information on program/process level events. A static heterogeneous graph is constructed to model the program behaviors. Formally, given the program event data across many machines within a time window (*e.g.*, 1 day), a heterogeneous graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ is constructed for the target program. $\mathscr{V}$ denotes a set of nodes, with each one representing an entity of three types: process (P), file (F), and INETSocket (I). Namely, $\mathscr{V} = P \cup F \cup I$. $\mathscr{E}$ denotes a set of edges (v_s, v_d, r) between the source entity v_s and destination entity v_d with relation r. To address the heterogeneous graph challenges, it takes three types of relations, including: (1) a process forking another process (P→P), (2) a process accessing a file (P→F), and (3) a process connecting to an Internet socket (P→I). Similar to GEM, DeepHGNN designs a graph transformation module to transform the heterogeneous graph to a 3-channel graph guided by above three meta-paths (relations), such that $\hat{\mathscr{G}} = \{\mathscr{G}_i | \mathscr{G}_i = (\mathscr{V}_i, \mathscr{E}_i, \mathrm{A}_i), i = 1, 2, ..., |\mathscr{M}|\}$ with $|\mathscr{M}| = 3$ and $\mathscr{V}_i = \mathscr{V}$. The attributes are constructed for each node. Since the process node, file node, and INETSocket node has quite different attributes, the graph statistic features $\mathbf{x}_v^{(i)(gstat)}$ are constructed and act as the node attributes.

Similar to the GEM and HACUD, DeepHGNN also adopts the graph convolutional network $AGG^{GCN}()$ for node-wise aggregation. Three layers are used in order to capture program behavior within 3-hop contexts. Different from GEM and HACUD, DeepHGNN uses the graph statistic node attributes as the initialization of the node representation for each channel graph. After the three node-wise aggregation and layer-wise aggregation, the node representations from different channel graphs are fused via the attentional feature fusion as GEM and HACUD. Formally, it can be described as follows,
Node-wise aggregation:

$$\mathbf{h}_v^{(i)(0)} = \mathbf{x}_v^{(i)(gstat)} \tag{26.25}$$

$$\begin{aligned} \mathbf{h}_v^{(i)(k)} &= AGG_{node}(\mathbf{h}_v^{(i)(k-1)}, \{\mathbf{h}_u^{(i)(k-1)}\}_{u \in \mathscr{N}_v^i}) \\ &= AGG^{GNN}(\mathbf{h}_v^{(i)(k-1)}, \{\mathbf{h}_u^{(i)(k-1)}\}_{u \in \mathscr{N}_v^i}) \end{aligned} \tag{26.26}$$

Layer-wise aggregation:

$$\mathbf{l}_v^{(i)(k)} = \mathbf{h}_v^{(i)(k)} \tag{26.27}$$

Path-wise aggregation:

$$\mathbf{p}_v = COMB_{att}(\mathbf{l}_v^{(1)(K))}, ..., \mathbf{l}_v^{(|\mathscr{M}|)(K)}) \tag{26.28}$$

Final node representation:

$$\mathbf{h}_v^{(final)} = \mathbf{p}_v \tag{26.29}$$

The object of DeepHGNN is classification. However, it is different from GEM and HACUD, which simply build single classifiers for all the samples. DeepHGNN formulates the problem of program reidentification in malicious program detection. The graph representation learning aims to learn the representation of the normal target program, and each target program learns a unique classifier. Given a target program with corresponding event data during a time window $U = \{e_1, e_2, ...\}$ and a claimed name/ID, the system checks whether it belongs to the claimed name/ID. If it matches the behavior pattern of the claimed name/ID, the predicted label should be $+1$; otherwise, it should be -1.

26.5.4 Case Study 4: Graph Matching Framework to Learn the Program Representation and Similarity Metric via Graph Neural Networks for Unknown Malicious Program Detection

Graph matching framework to learn the program representation and similarity metric via graph neural network (MatchGNet) (Wang et al, 2019i) is another GNN-based anomaly detection approach for malicious program detection in a computer system of an enterprise network. MatchGNet is different from DeepHGNN in five aspects: (1) after the graph transformation, the resulted channel graph only keep the target type node – process node, which is similar to HACUD, (2) the raw program attributes are used as the program node representation initialization, (3) the GNN aggregation is conducted hierarchically in node-wise, layer-wise, and path-wise, (4) the anomaly target is the subgraph of the target program (5) the final graph representation is fed to a similarity learning framework with contrastive loss to deal with the unknown anomaly.

It follows a similar style to construct the static heterogeneous graph from system behavioral data. In the graph transformation, it adopts three meta-paths (relations): a process forking another process ($P \rightarrow P$), two processes accessing the same file ($P \leftarrow F \rightarrow P$), and two processes opening the same internet socket ($P \leftarrow I \rightarrow P$) with each one defining a unique relationship between two processes. Based on them, a 3-channel graph is constructed from the the heterogeneous graph, such that $\hat{\mathscr{G}} = \{\mathscr{G}_i | \mathscr{G}_i = (\mathscr{V}_i, \mathscr{E}_i, A_i), i = 1, ..., |\mathscr{M}|\}$ with $|\mathscr{M}| = 3$ and $\mathscr{V}_i \in P$. Then the GNN can be directly applied to each channel graph. As only process typed nodes are available, we use the raw attributes of these process $\mathbf{x}_v$ as the node representation initialization.

During the graph representation stage, a hierarchical attentional graph neural network is designed, including node-wise attentional neural aggregator, layer-wise dense-connected neural aggregator, and path-wise attentional neural aggregator. In particular, the node-wise attentional neural aggregator aims to generate node embeddings by selectively aggregating the entities in each channel graph based on random walk scores $\alpha^i_{(u)}$. Layer-wise dense-connected neural aggregator aggregates the node embeddings generated from different layers towards a dense-connected node embedding. Path-wise attentional neural aggregator performs attentional feature fusion of the layer-wise dense-connected representations. In the end, the final node representation is used as the graph representation. Formally, it can be described as follows,
Node-wise aggregation:

$$\mathbf{h}_v^{(i)(0)} = \mathbf{x}_v \tag{26.30}$$

$$\begin{aligned} \mathbf{h}_v^{(i)(k)} &= AGG_{node}(\mathbf{h}_v^{(i)(k-1)}, \{\mathbf{h}_u^{(i)(k-1)}\}_{u \in \mathcal{N}_v^i}) \\ &= MLP((1+\varepsilon^{(k)})\mathbf{h}_v^{(i)(k-1)} + \sum_{u \in \mathcal{N}_v^i} \alpha^i_{(u)(:)} \mathbf{h}_u^{(i)(k-1)}) \end{aligned} \tag{26.31}$$

Layer-wise aggregation:

$$\begin{aligned} \mathbf{l}_v^{(i)(k)} &= AGG_{layer}(\mathbf{h}_v^{(i)(0)}, \mathbf{l}_v^{(i)(1)}, \ldots \mathbf{l}_v^{(i)(k)}) \\ &= MLP(CONCAT(\mathbf{h}_v^{(i)(0)}; \mathbf{l}_v^{(i)(1)}; \ldots \mathbf{l}_v^{(i)(k)})) \end{aligned} \tag{26.32}$$

Path-wise aggregation:

$$\mathbf{p}_v = COMB_{att}(\mathbf{l}_v^{(i)(K))}, \ldots, \mathbf{l}_v^{(|\mathcal{M}|)(K)}) \tag{26.33}$$

Final node representation:

$$\mathbf{h}_v^{(final)} = \mathbf{p}_v \tag{26.34}$$

Final graph representation:

$$\mathbf{h}_{G_v} = \mathbf{h}_v^{(final)} \tag{26.35}$$

where k indicates the number of layers, and ε is a small number. Different from GEM, HACUD, and DeepHGNN, the object of MatchGNet is matching. The final graph representation is fed to a similarity learning framework with contrastive loss to deal with the unknown anomaly. During the training, P pairs of program graph snapshots $(\mathcal{G}_{i(1)}, \mathcal{G}_{i(2)}), i \in \{1,2,\ldots P\}$ are collected with corresponding ground truth pairing information $y_i \in \{+1,-1\}$. If the pair of graph snapshots belong to the same program, the ground truth label is $y_i = +1$; otherwise, its ground truth label is $y_i = -1$. For each pair of program snapshots, a cosine score function is used to measure the similarity of the two program embeddings, and the output is defined as follows:

$$\begin{aligned} Sim(\mathcal{G}_{i(1)}, \mathcal{G}_{i(2)}) &= cos((\mathbf{h}_{\mathcal{G}_{i(1)}}, \mathbf{h}_{\mathcal{G}_{i(2)}})) \\ &= \frac{\mathbf{h}_{\mathcal{G}_{i(1)}} \cdot \mathbf{h}_{\mathcal{G}_{i(2)}}}{||\mathbf{h}_{\mathcal{G}_{i(1)}}|| \cdot ||\mathbf{h}_{\mathcal{G}_{i(2)}}||} \end{aligned} \tag{26.36}$$

Correspondingly, our objective function can be formulated as:

$$\ell = \sum_{i=1}^{P} (Sim(\mathcal{G}_{i(1)}, \mathcal{G}_{i(2)}) - y_i)^2 \tag{26.37}$$

26.5.5 Case Study 5: Anomaly Detection in Dynamic Graph Using Attention-based Temporal GCN

Anomaly detection in dynamic graph using attention-based temporal GCN (AddGraph) (Zheng et al, 2019) is the first work that applies the GNN to solve the problem of anomaly edge detection in the dynamic graph. It focuses on the modeling of the dynamic graph via GNN and performs anomaly connection detection in telecom networks and social networks. The graphs are constructed from the edge stream data, and the constructed graphs are dynamic, homogeneous, and plain.

The basic idea is to build a framework to describe the normal edges by using all possible features in the graph snapshots in the training phase, including structural, content, and temporal features. Then at the prediction stage, the matching objective is used similar to MatchGNet. In particular, AddGraph applies GCN $AGG^{GCN}()$ to compute the new current state of a node $\mathbf{c}_v^t$ by aggregating its neighborhoods in the current snapshot graph, which can be described as follows,

$$\mathbf{c}_v^t = AGG^{GCN}(\mathbf{h}_v^{t-1}) \tag{26.38}$$

As the state of a node $\mathbf{c}_v^t$ can be computed by aggregating the neighboring hidden states in the previous timestamp $t-1$, the node hidden states in a short window w can be obtained and combined to get the short-term embedding $\mathbf{s}_v^t$. In particular, an attentional feature fusion is used to combine these node hidden states in a short window, as follows,

$$\mathbf{s}_v^t = COMB_{att}(\mathbf{h}_v^{t-w}, ..., \mathbf{h}_v^{t-1}) \tag{26.39}$$

Then short-term embedding $\mathbf{s}_v^t$ and current state $\mathbf{c}_v^t$ are fed to GRU, a classic recurrent neural network, to compute the current hidden state that encoding the dynamics within the graph. This stage can be described as follows:

$$\mathbf{h}_v^t = GRU(\mathbf{c}_v^t, \mathbf{s}_v^t) \tag{26.40}$$

The object of AddGraph is matching. The hidden state of the nodes at each timestamp are used to calculate the anomalous probabilities of an existing edge and a negative sampled edge, and then feed them to a margin loss.

26.5.6 Case Study 6: GCN-based Anti-Spam for Spam Review Detection

GCN-based anti-spam (GAS) (Li et al, 2019a) applies the GNN in the spam review detection at the e-commerce platform Xianyu. Similar to previous works, the constructed graph is static, heterogeneous and attributed, such that $\mathscr{G} = (\mathscr{U}, \mathscr{I}, \mathscr{E})$. There are two types of nodes: user nodes $\mathscr{U}$ and item nodes $\mathscr{I}$. The edges $\mathscr{E}$ are a set of comments. Different from previous works, the edges $\mathscr{E}$ are the anomalies targets. Moreover, as each edge represents a sentence, edge modeling is complicated, and the number of edge types increases dramatically. To better capture the edge representation, the message-passing-like GNN is used. The edge-wise aggregation is proposed by concatenation of previous representation of the edge itself $\mathbf{h}_{iu}^{k-1}$ and corresponding user node representation $\mathbf{h}_u^{k-1}$, item node representation $\mathbf{h}_i^{k-1}$ To get the initial attributes of edge, the word2vec word embedding for each word in the comments of the edges is extracted via the embedding function pre-training on a million-scale comment dataset. Then the word embedding of each words in an edge of comments $\mathbf{w}_0, \mathbf{w}_1, ...\mathbf{w}_n$ is fed to $TextCNN()$ function to get the comments embedding $\mathbf{h}_{iu}^0$, which is used as the initial attributes of edge. Then the edge-wise aggregation is defined as:
Edge-wise aggregation:

$$\mathbf{h}_{iu}^0 = TextCNN(w_0, w_1, ...w_n) \tag{26.41}$$

$$\mathbf{h}_{iu}^k = MAP_{nonlinear}(CONCAT(\mathbf{h}_{iu}^{k-1}, \mathbf{h}_i^{k-1}, \mathbf{h}_u^{k-1})) \tag{26.42}$$

On the other hand, the node-wise aggregation also needs to take the edges into consideration. The node-wise aggregation is performed by attention feature fusion of the target node and its connected edge followed by a non-linear mapping, which can be described with (a) user node-wise aggregation, and (b) item node-wise aggregation as follows:
Node-wise aggregation:

(a)*User node-wise aggregation*:

$$\mathbf{h}_u^k = CONCAT(MAP_{linear}(\mathbf{h}_u^{k-1}), MAP_{nonlinear}(COMB_{att}(\mathbf{h}_u^{k-1}, CONCAT(\mathbf{h}_{iu}^{k-1}, \mathbf{h}_i^{k-1})))) \tag{26.43}$$

(b)*Item node-wise aggregation*:

$$\mathbf{h}_i^k = CONCAT(MAP_{linear}(\mathbf{h}_i^{k-1}), MAP_{nonlinear}(COMB_{att}(\mathbf{h}_i^{k-1}, CONCAT(\mathbf{h}_{iu}^{k-1}, \mathbf{h}_u^{k-1})))) \tag{26.44}$$

where k is the layer indicator. The final edge representation is computed by concatenation of the raw edge embedding $\mathbf{h}_{iu}^0$, new edge embedding $\mathbf{h}_{iu}^K$, corresponding new user node embedding $\mathbf{h}_u^K$, and corresponding new item node embedding $\mathbf{h}_i^K$ as follows:
Final edge representation:

$$\mathbf{h}_{iu}^{final} = CONCAT(\mathbf{h}_{vu}^0, \mathbf{h}_{vu}^K, \mathbf{h}_u^K, \mathbf{h}_i^K) \tag{26.45}$$

The object of GAS is classification, and the final edge representation is fed to a standard logistic loss function.

26.6 Future Directions

GNNs on anomaly detection is an important research direction, which leverages multi-source, multi-view features extracted from both content and structure for anomaly sample analysis and detection. It plays a key role in numerous high-impact applications in areas such as cyber-security, finance, e-commerce, social network, industrial monitoring, and many more mission-critical tasks. Due to the multiple issues from data, model and task, it still needs a lot of effort in the field. The future works are mainly lying in two perspectives: anomaly analysis and machine learning.

From an anomaly analysis perspective, there are still a lot of research questions. How to define and identify the anomalies in the graph in the different tasks? How to effectively convert the large-scale raw data to the graph? How to effectively leverage the attributes? How to model the dynamic during the graph construction? How to keep the heterogeneity during the graph construction? Recently, due to the data-specific and task-specific issues, the applications of GNN-based anomaly detection are still limited. There is still a lot of potential scenarios that can be applied.

From a machine learning perspective, lots of issues need to be addressed. How to model the graph? How to represent the graph? How to leverage the context? How to fuse the content and structure features? Which part of the structure to capture, local or global? How to provide the model explainability? How to protect the model from adversarial attacks? How to overcome the time-space scalability bottleneck. Recently, lots of contributions have been made from the machine learning perspective. However, due to the unique characteristics of the anomaly detection problem, which GNNs to use and how to apply GNNs are still critical questions. Further work will also benefit from the new findings and new models in the graph machine learning community.

Editor's Notes: Graph neural networks for anomaly detection can be considered as a downstream task of graph representation learning, where the long-term challenges in anomaly detection are coupled with the vulnerability of graph neural networks such as scalability discussed in Chapter 6 and robustness discussed in Chapter 8. Graph neural networks for anomaly detection also further benefits a wide range of downstream tasks in various interesting, important, yet usually challenging areas such as anomaly detection in dynamic networks, spam review detection for recommender system, and malware program detection, which are highly relevant to the topics introduced in Chapters 15, 19, and 22.