

## MASTER'S THESIS

Presented for the purpose of obtaining the  
NATIONAL MASTER'S DEGREE

in Robotics, Computer Science and Communication Systems  
Data Science and Smart Services

---

# Anomaly Detection with Graph Neural Networks

---

*Presented by*  
CHAYMA ELBAHRI

Defended on 21/06/2023 in front of the jury composed of:

President:	Pr. Afef Abdelkrim, Professor at ENICarthage.
Examiner:	Dr. Ikbal Msadaa, Assistant professor at ENSTAB.
Academic supervisor:	Dr. Khaled Belghith, Assistant professor at ISTIC.

Academic year: 2022-2023

# Dedication

To my parents,  
To my sister,  
To my friends.

# Acknowledgments

I am profoundly grateful to Mr. Belghith Khaled for supervising this work and offering valuable guidance and support throughout the entire project.

My sincere appreciation extends to all the professors at the Institute of Advanced Technologies in Information and Communication for imparting their knowledge and my academic growth during my research Master's studies in Data Science and Smart Service.

I would like to express my profound gratitude to the President of the jury, Prof. Afef Abdelkrim, and the Examiner, Dr. Ikbâl Msadaa, for their essential role in evaluating my research work. Their feedback and suggestions have been extremely valuable in improving the quality of this work.

I would like to express my profound gratitude to Professor Phillipe Fournier-Viger for presenting the subject upon which I dedicated my efforts.

Lastly, I am humbled and honored that the members of the jury have graciously accepted the responsibility of evaluating my work.

# Contents

<b>Dedication</b>	<b>i</b>
<b>Acknowledgments</b>	<b>ii</b>
<b>General Introduction</b>	<b>1</b>
<b>1 State of the art of GNNs</b>	<b>4</b>
1.1 Introduction . . . . .	4
1.2 Machine learning introduction . . . . .	4
1.2.1 Traditional machine learning . . . . .	4
1.2.2 Deep learning with Neural Networks . . . . .	8
1.2.3 Learning on graphs . . . . .	12
1.3 Machine learning for graphs . . . . .	12
1.3.1 What is a graph? . . . . .	13
1.3.2 Graph node-level features . . . . .	14
1.3.3 Node embedding . . . . .	15
1.3.4 Graph Neural Networks . . . . .	17
1.3.5 Graph data analysis methods . . . . .	22
1.4 Convolution on graphs . . . . .	22
1.4.1 Graph signals . . . . .	22
1.4.2 Graph Convolutional Networks . . . . .	23
1.5 Conclusion . . . . .	24
<b>2 State of the art of GNN based Anomaly Detection methods</b>	<b>25</b>
2.1 Introduction . . . . .	25
2.2 Overview on the existing Anomaly Detection approaches . . . . .	25
2.3 Spatial Anomaly Detection models . . . . .	26
2.3.1 GraphConsis framework . . . . .	26
2.3.2 Camouflage resistant GNN . . . . .	28
2.4 Spectral Anomaly Detection models . . . . .	31
2.4.1 Dominant framework . . . . .	31
2.4.2 Graph Neural Networks with adaptive receptive paths . . . . .	33
2.4.3 Adaptive multi-frequency GNN . . . . .	35
2.5 State of the art models results . . . . .	37
2.6 Conclusion . . . . .	38
<b>3 Adaptive frequency GNN for Anomaly Detection</b>	<b>39</b>
3.1 Introduction . . . . .	39
3.2 Adaptive frequency response filter GNN . . . . .	39

3.2.1	AdaGNN architecture . . . . .	40
3.2.2	Over-smoothing problem . . . . .	41
3.2.3	Explanatory example . . . . .	42
3.3	Motivation and contribution . . . . .	43
3.4	AdaGNN for Anomaly Detection . . . . .	46
3.4.1	Data preprocessing . . . . .	47
3.4.2	Feature representation . . . . .	48
3.4.3	Adaptive frequency response . . . . .	48
3.4.4	Model prediction . . . . .	51
3.4.5	Model optimisation . . . . .	51
3.5	Conclusion . . . . .	52
<b>4</b>	<b>Implementation and Experimental results</b>	<b>53</b>
4.1	Introduction . . . . .	53
4.2	Experimental setup . . . . .	53
4.2.1	Implementation details . . . . .	53
4.2.2	Databases . . . . .	53
4.2.3	Data preprocessing . . . . .	55
4.3	Experiments . . . . .	58
4.3.1	Comparison methods . . . . .	58
4.3.2	Hyperparameter tuning . . . . .	58
4.3.3	Evaluation metrics . . . . .	58
4.4	Results discussion . . . . .	59
4.4.1	Experimental results . . . . .	59
4.4.2	Results analysis . . . . .	61
4.5	Conclusion . . . . .	65
	<b>General Conclusion</b>	<b>66</b>
	<b>Bibliography</b>	<b>70</b>

# List of Figures

1.1	Machine learning (Shalev-Shwartz und Ben-David, 2014)	5
1.2	Convolutional Neural Network (Ciaburro, 2017)	10
1.3	Gradient descent convergence (Panagiotis, 2023)	11
1.4	Different data structures (Leskovec, 2023)	12
1.5	Zachary karate club network (PyG, 2023)	13
1.6	Illustration of the node embedding problem (Hamilton u. a., 2017)	16
1.7	Overview of the encoder-decoder approach (Hamilton u. a., 2017)	16
1.8	GNN layers (Leskovec, 2023)	18
1.9	GNN task levels (Leskovec, 2023)	20
1.10	GNN training for node classification (Leskovec, 2023)	21
1.11	Cyclic time series represented as a chain graph (Hamilton u. a., 2017)	22
1.12	Graph Convolutional Networks (Kipf und Welling, 2016)	24
2.1	GraphConsis framework (Liu u. a., 2020)	27
2.2	CARE-GNN aggregation process (Dou u. a., 2020a)	29
2.3	Dominant framework (Ding u. a., 2019)	31
2.4	GeniePath architecture (Liu u. a., 2019)	33
2.5	AMNet framework (Chai u. a., 2022)	35
3.1	Illustrative example of AdaGNN (Dong u. a., 2021)	42
3.2	Comparison between low and high-frequency components (Dong u. a., 2021)	44
3.3	High-frequency components after low-pass filtering (Dong u. a., 2021)	44
3.4	Anomaly-AdaGNN architecture	46
3.5	Anomaly-AdaGNN node embedding (Dong u. a., 2021)	51
4.1	Illustration of the Elliptic dataset class imbalance	56
4.2	Illustration of the Elliptic dataset after balancing classes	57
4.3	Elliptic dataset results of Anomaly-AdaGNN	60
4.4	Amazon dataset results of Anomaly-AdaGNN	60
4.5	Yelp dataset results of Anomaly-AdaGNN	61
4.6	Frequency response function learned from Elliptic by Anomaly-AdaGNN	62
4.7	Low-frequency filters learned from Elliptic by Anomaly-AdaGNN	62
4.8	High-frequency filters learned from Elliptic by Anomaly-AdaGNN	63
4.9	AUC-ROC and AUC-PR scores comparison	64

# List of Tables

2.1	AUC-ROC scores (Chai u. a., 2022) . . . . .	38
2.2	AUC-PR scores (Chai u. a., 2022) . . . . .	38
4.1	Datasets statistics . . . . .	55
4.2	Anomaly-AdaGNN Results on Different Datasets . . . . .	61
4.3	AUC-ROC and AUC-PR scores for anomaly detection models . . . . .	63

# General Introduction

In the era of machine learning, sometimes, what stands out in the data is more significant than what is normal. Anomaly detection or outlier detection means identifying rare items, events, or observations that differ significantly from the majority of the data. Anomalies, which counter the norm, have gained increasing interest in different domains because their detection holds valuable insights and enables the prevention of significant drawbacks.

Traditionally, anomalies were removed to improve statistical analysis or enhance the performance of machine learning models. However, in many applications, anomalies themselves are of significant interest and need to be accurately identified and isolated from noise or irrelevant data. In fields such as network intrusion detection, cyber-security, fraud detection in finance and e-commerce, and fault detection in industrial monitoring outlier detection has become indispensable. Various techniques, including machine learning methods, have been developed to address anomaly detection challenges. These methods leverage statistical features extracted from different aspects of user profiles, behaviors, or transaction summaries. Classical classifiers like logistic regression and neural networks are often employed for classification based on these statistical attributes (Hoogs u. a., 2007)(Yue u. a., 2007).

In the context of the given text, relationships refer to the connections and interactions between different data points within a system or dataset. These relationships can exist in various forms, such as social connections between individuals, financial transactions between users, or connections between different entities in a network. Understanding and considering these relationships is crucial because they contain valuable information and insights that can enhance the analysis and interpretation of the data. For example, in fraud detection, knowing the relationships between users, vendors, and their transaction history can help identify suspicious patterns or abnormal behavior that may indicate fraudulent activity. Traditional methods often overlook these relationships or struggle to capture their significance, relying more on predetermined rules or assumptions. On the other hand, approaches like graph neural networks (GNNs) are designed to explicitly model and leverage these relationships. GNNs can propagate information across the graph structure, allowing them to capture complex patterns and dependencies, leading to improved analysis and detection capabilities, especially in scenarios where relationships play a crucial role.

Several novel approaches leveraging the graph structure have been proposed, with GNNs demonstrating remarkable capabilities in anomaly detection. GNN-based methods leverage the expressive power of GNNs to learn node representations, enabling the discrimination between normal and anomalous nodes in the embedding space. By aggregating neighborhood data and training end-to-end, GNN-based methods alleviate the



need for extensive feature engineering and data annotation, leading to improved performance compared to non-graph-based techniques. The utilization of graph-based methods allows for the incorporation of entity relations, enhancing the detection of suspicious entities connected through similar purposes (Akoglu u. a., 2015).

Despite the successful application of existing anomaly detectors based on graph neural networks, their expressive power is constrained by their reliance on low-pass filters (Wu u. a., 2019). These filters prioritize amplifying low-frequency signals, which are characterized by smooth variations in the data while suppressing high-frequency signals, which exhibit rapid oscillations. This assumption is based on the homophily hypothesis (McPherson u. a., 2001), which suggests that nodes with similar features tend to connect in networks. However, this assumption may not hold in networks that contain anomalies, as anomalous nodes often possess distinct features that deviate from those of their neighbors. Consequently, relying solely on fixed low-pass filters proves inadequate for capturing the varying significance of different frequency elements in the data. This limitation can potentially undermine the model’s performance and restrict the richness of the learned representations.

In simpler terms, existing GNN-based anomaly detectors primarily focus on capturing smooth patterns in the data while disregarding rapid changes or anomalies. They assume that nodes with similar features are more likely to connect in a network. However, in networks with anomalies, this assumption may not hold, as anomalous nodes may exhibit unique characteristics that differ from their neighboring nodes. This means that using fixed filters that prioritize smooth signals may overlook important information and limit the model’s ability to detect anomalies effectively.

## **Problem**

The adoption of most GNNs in anomaly detection faces suboptimality due to their inherent low-pass property, which is incongruent with the nature of networks containing anomalies. GNNs tend to blur or smooth out the differences between normal and anomalous nodes, making it difficult to accurately distinguish anomalous nodes. Furthermore, these methods typically employ fixed low-pass filters that cannot adapt to the varying information contained in different frequency elements of normal and anomalous nodes. This lack of adaptivity hampers their ability to achieve optimal performance in anomaly detection tasks.

## **Contribution**

This work focuses on overcoming the inherent limitations of GNN-based graph anomaly detection from a spectral perspective. The spectral analysis provides insights into the frequency components of the data, which is crucial for understanding the low-pass property and the challenges it poses in detecting anomalies. The proposed solution tackles this issue by adaptively combining low and high-frequency information to learn node embeddings specifically designed for detecting anomalous nodes. By utilizing low-frequency information for normal nodes, the approach retains common features shared among them. On the other hand, it emphasizes high-frequency data for anomalous nodes, effectively highlighting their distinct characteristics. This adaptability in leveraging different frequency components enables the model to have the necessary inductive bias to detect anomalies

accurately. The approach can selectively use low-frequency, high-frequency, or both types of information, depending on the node being analyzed. This flexibility allows the model to effectively identify anomalous nodes by capturing the unique features that deviate from the norm. Therefore, our main contributions are summarised below:

1. Conduct an in-depth literature review on anomaly detection with graph neural networks.
2. Deploy an adaptive trainable multi-frequency filter group to capture graph signals of both low and high-frequency.
3. Evaluate the performance of the proposed model through extensive experiments and compare it with existing baseline methods.

The rest of the thesis is organized as follows :

- Chapter 1 serves as a foundation for understanding the role of graph neural networks in anomaly detection. It delivers a comprehensive review of GNNs, emphasizing their applications and importance in different domains. Also, the chapter explores the principles and mechanisms of GNNs and their expressive power making them well-suited for anomaly detection tasks. This chapter establishes the context and importance of integrating GNNs into anomaly detection methods.
- Chapter 2 focuses on existing contributions in the anomaly detection field using graph neural networks. It categorizes specific approaches that researchers employed to tackle this problem. Furthermore, the chapter compares these approaches and identifies the most efficient techniques, paving the way for the proposed solution.
- Chapter 3 presents the proposed solution introducing the Anomaly-AdaGNN model. The chapter outlines the architecture and algorithm of the model, which incorporates adaptive frequency response filtering characteristics of the AdaGNN deep-encoder. We underline the models' ability to learn node embeddings and effectively capture both low-frequency and high-frequency information.
- Chapter 4 is dedicated to the comprehensive discussion of the experimental results obtained. It describes the experimental setup and preprocessing procedures. The chapter also examines the experimental outcomes in detail, highlighting the strengths and limitations of the Anomaly-AdaGNN model.
- We finally draw a conclusion of the thesis, to summarise the main findings and contributions, and outline potential directions for future research.

Overall, the thesis provides an in-depth review of graph neural networks, discusses existing contributions and limitations, presents the proposed model and model architecture, and offers a comprehensive evaluation of the model's performance.

# Chapter 1

## State of the art of GNNs

### 1.1 Introduction

Machine learning by nature is a problem-driven field. It attempts to generate models that could learn from data to fix specific tasks. In many cases, machine learning models are categorized according to the type of problem they target. Similarly, in the context of graph data, machine learning with graphs focuses on generating models that utilize the graph structure and available attribute information. GNNs aim to develop representations of nodes that rely on the layout of the graph, as well as any attribute information available.

In this chapter, we begin by providing an introduction to machine learning and describing the fundamentals of graphs. We also discuss the utilization of node-level statistics and features in graphs. Subsequently, we present a comprehensive overview of GNNs, which can be trained to compute embeddings, represent graph data, and make predictions for various classifications. Finally, we conclude the chapter by exploring graphs from a spectral perspective and introducing the concept of frequency in graphs and convolution on graphs.

### 1.2 Machine learning introduction

Machine learning is a subset of the field of artificial intelligence, which shows that a machine is capable of learning through experience. This technology uses data and algorithms to mimic human learning while increasing accuracy. It is a method of data analysis that automates analytical model training to learn from data, identify patterns and make decisions with minimal human intervention.

#### 1.2.1 Traditional machine learning

Machine learning (Theobald, 2017) refers to the ability of a system to acquire and integrate knowledge through large-scale observations and expand by learning new knowledge rather than being programmed with it. This knowledge will then guide decision-making in applications and businesses, impacting key evolutionary metrics. Machine learning is a fundamental part of the field of data science that can be used to do tasks that humans complete with minimum effort but cannot explain how they do them.

For example, humans can recognize friends voices without much difficulty. Due to the lack of understanding of such a phenomenon (speech recognition in this case), it's not possible to develop an algorithm for such situations. Machine learning algorithms help bridge this understanding gap using models defined as an approximation of the process machines mimic. As the field of big data grows, so does the need for machine learning to help answer important business questions using data. The idea is that an algorithm can use a sample of data observations to estimate a prediction function called a model and then use the prediction function defined to predict the output of new unknown data (Russell und Norvig, 2003). These ML algorithms, trained on the data, can then be saved somewhere and called a model, which is a generalized representation of data's hidden patterns. Figure 1.1 explains the process.

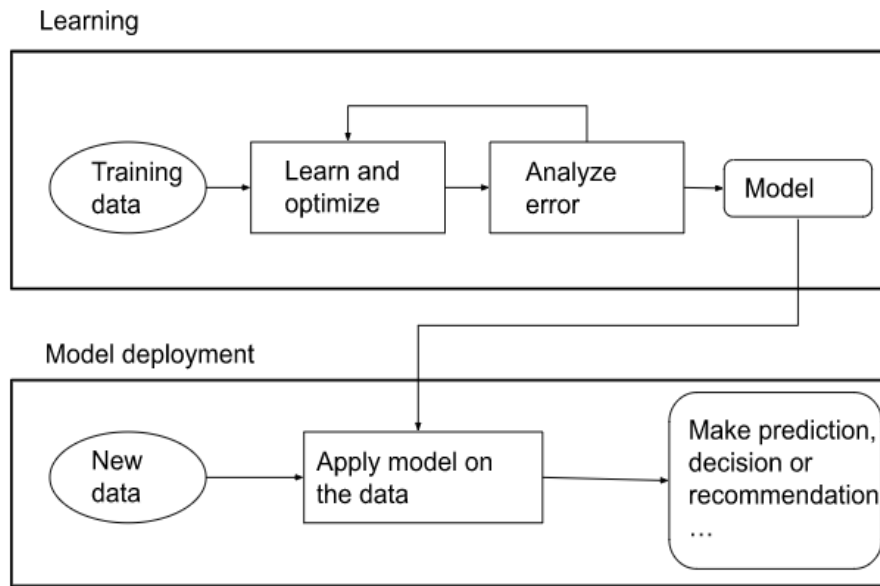


Figure 1.1: Machine learning (Shalev-Shwartz und Ben-David, 2014)

The idea of ML is that for a set of data observations  $x_i \in X$ , each observation has  $d$  attributes that describe it and there is a prediction function  $\hat{f}$  that can be calculated to predict  $\hat{y}_i$  the output of  $x_i$ . The assumption is that for each learning task, there is a real prediction function  $f$  that maps each observation  $x_i$  to its output  $y_i$

$$y_i = f(x_i),$$

and the goal of the training is to learn  $\hat{f}$  the best approximation of  $f$  and then apply  $\hat{f}$  to predict the output of unknown observations:

$$\hat{y}_i = \hat{f}(x_i)$$

### Machine learning pipeline

The machine learning pipeline (Zhang u. a., 2021) is a series of stages used to process and analyze data, build models, and make predictions. It is an important part of constructing and implementing accurate machine learning models as it provides a structured strategy

to alleviate the learning task. The process usually consists of data preparation, feature selection, model training, and evaluation:

1. Data preparation is the first step in the machine learning process and is crucial to the model's success. This step involves cleaning, transforming, and normalizing the data to make it fit the chosen machine learning algorithm. This step can include filling in non-available values, deleting noise data, and transforming categorical variables into numeric variables. The first step in the machine learning process is to prepare the data. This step is critical to the success of the model as it involves cleaning, transforming, and normalizing the data to fit the machine learning algorithm. It is important to mention that data quality directly impacts the performance of the model, therefore investing time and resources in this step is essential.
2. The feature selection step, where relevant features or attributes are picked from the dataset. This stage improves model performance and reduces overfitting by clearing unnecessary or redundant features. Feature selection can be done manually or automatically. Also, the feature selection process needs a combination of domain knowledge and a data-driven approach.
3. The model training phase is where the ML algorithm is applied to the data set. This step involves splitting the data into training and test sets. The training set is used to teach the model the prediction function and capture hidden patterns, enabling the model to fit the data. The training process of a machine learning system can be seen as three parts that repeat until the model gets to a certain precision or until a certain number of iterations is reached (Russell und Norvig, 2003):
  - The decision process: machine learning algorithms are used to predict or classify a data instance based on a set of input data. The algorithm will produce an estimate of a pattern in the data which is represented by the prediction function  $\hat{f}$ .
  - Error calculation process: an error function evaluates the predictions of the function  $f$  to specify its precision.
  - Model optimisation or backpropagation process: if  $\hat{f}$  still has a marge of error the  $\hat{f}$ 's parameters will be adjusted to minimise the error.

Different algorithms have different parameters that need to be calculated, and these parameters can be optimized using numerous techniques for example grid search and k-fold cross-validation. It should also be noted that for some problems, using a set of models can be more useful than a single model, as this can cause improved performance.

4. After training the model, the next stage is to evaluate its performance using the test dataset. This step is done by using the trained model to calculate predictions  $\hat{y}_i$  on the test set and comparing them with actual output values  $y_i$ . During the evaluation phase, the model's performance is assessed using metrics such as accuracy, precision, and recall. Accuracy ranges from 0 to 1 and measures the overall correctness of a classification model:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1.1)$$

where  $TP$  means true positives or correctly predicted positive instances,  $TN$  refers to true negatives or correctly predicted negative data instances,  $FP$  is false positives or incorrectly predicted positive instances and  $FN$  refers to false negatives or incorrectly predicted negative instances. Precision quantifies the model ability to correctly identify positive instances:

$$Precision = \frac{TP}{TP + FP} \quad (1.2)$$

and Recall measures the proportion of true positive predictions out of the total actual positive instances:

$$Recall = \frac{TP}{TP + FN} \quad (1.3)$$

In addition, it is important to evaluate the performance of the model on a variety of measures and also on different subsets of data, such as underrepresented groups, to guarantee that the model is fitting and unbiased. The model is then refined, if necessary, and deployed to construct predictions about the new data. Note that a model that performs well on the training set may not necessarily perform well on new unknown data. This is called overfitting and can be solved by using approaches such as cross-validation and early stopping. Intuitively, overfitting happens when a model or a prediction function fits the training data accurately yet it performs poorly on the test data set.

5. Finally, after the model is finalized and evaluated, it can be deployed and utilized to produce predictions about new data. However, as model performance can degrade over time due to differences in the underlying data distribution, it is important to have a monitoring and upkeep plan in place.

In summary, machine learning workflows are an essential part of constructing and deploying successful machine learning models.

### Types of machine learning tasks

ML models rely heavily on the input data and they're divided into three main categories (Shalev-Shwartz und Ben-David, 2014) based on how the training data is used:

- Supervised learning: the goal here is to use labeled data examples to train algorithms to classify or predict data and predict results accurately. The attribute vector and the output labels  $y$  are fed into the model, and the model alters its weights until the models predictions match exactly the real labels of the input.
- Unsupervised learning: also referred to as self-supervised learning, employs machine learning algorithms to analyze and cluster unlabeled datasets. This approach is effective in uncovering patterns and similarities within the data, making it more valuable for exploratory data analysis and customer segmentation.
- Semi-supervised learning: occupies a unique space between supervised and unsupervised learning. In this method, a smaller labeled dataset is used during training to guide the classification and feature extraction procedures on a larger unlabeled dataset. By leveraging the advantages of both labeled and unlabeled data, semi-supervised learning offers a compelling solution for tackling data challenges where limited labeled data is available.

- Reinforcement learning: the purpose is to train a decision-making agent to maximize the reward. The agent interacts with an environment and gains a reward or punishment depending on its actions. Reinforcement learning is the go-to method in many applications, such as games, robotics, and control systems

Overall, machine learning is an effective tool that can be applied to multiple types of problems, and understanding the different types of problems can help select the right approach. Whether the task is classifying images, predicting stock prices, or controlling a robot, machine learning can deliver effective solutions.

## 1.2.2 Deep learning with Neural Networks

Deep learning is a subfield of machine learning where we focus on training and employing deep neural networks to extract complex patterns and representations from data. It has gained attention and achieved outstanding success in different domains, from computer vision and natural language processing to speech recognition. In this section, we will introduce the basics of deep learning and demonstrate the architecture of deep learning models.

### Artificial Neural Networks

Artificial Neural Network (Luger, 2005) or **ANN** is a subset of supervised machine learning inspired by biological neural networks in the human brain and is the core of deep learning algorithms. The human brain consists of many computing devices (neurons) connected in a complex communication network, giving the brain the ability to perform intricate computations. Inspired by the structure of neural networks in the brain, an ANN consists of layers of interconnected "neurons" that take input, process it, and pass it on to the next layer. The idea behind neural networks is that numerous neurons can be linked by communication links to perform elaborate computations.

One of the most basic forms of a neural network is Rosenblatt's Perceptron (Rosenblatt, 1958). It was the first operational ANN and is one of the oldest supervised learning binary classification algorithms. Perceptron made it possible for the artificial neuron to compute the correct weights from training data by itself. The idea is that it uses a linear activation function to model the decision:

$$\hat{y}_i(x_i) = \sigma(w_0 + \sum_i^d w_i x_i) \quad (1.4)$$

the activation function  $\sigma$  is used to add non-linearities into neural networks, it checks that an incoming value is greater than a critical number meaning:

$$\hat{y}_i(x_i) = \begin{cases} 1 & \text{if } w_0 + \sum_{i=1}^d w_i x_i \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (1.5)$$

the weight vector  $W$  corresponds to the models parameters  $w_1$  that will be trained later to determine each data point importance,  $d$  is the number of input data points of  $X$ . The bias  $w_0$  is an external parameter of the neuron set to 1, it acts like an extra weight that needs to be adjusted during learning to add more flexibility to the model and determines whether a neuron is activated or not. The learning algorithm of the perceptron adapts

the value of the weights and bias so that  $\hat{y}_i(x_i)$  corresponds to the correct answer on the training data, the training input is the learning rate  $\alpha$ , the maximum number of iterations, and the training dataset that contains  $(x_i, y_i)$  and the algorithm is as follows :

1. for each pair  $(x_i, y_i) \in d$  :  
     calculate  $\hat{y}_i(x_i)$  ,  
     if  $\hat{y}_i(x_i) \neq y_i$  :  
      $w_i \leftarrow w_i + \alpha(y_i - \sigma(x_i))x_i$
2. Backpropagate to 1 until the maximum number of iterations is reached.

## Limitations

One of the main limitations of perceptrons (Rosenblatt, 1958) is that they can only solve linearly separable problems. Meaning that it is only possible to classify data bounded by a single straight line and they can only learn linear decision borders between different data classes. This is an important limitation, as many real-world problems involve more complex and non-linear relationships between input features and output labels. For instance, a dataset containing images of handwritten digits can have a nonlinear relationship between pixel values and digit designations, making it challenging for perceptrons to classify the images accurately.

To overcome this limitation, the field of deep neural networks has emerged, offering a powerful solution. Deep neural networks, also known as deep learning, are a type of artificial neural network that consists of multiple hidden layers composed of artificial neurons (Zhang u. a., 2021). The term "deep" represents the use of a significant number of layers in the network. The layers are interconnected, with the output of one layer acting as the input for the next layer. The initial layer known as the input layer, accepts the raw input data, while the final layer, called the output layer, generates predictions or decisions. The intermediate layers are referred to as hidden layers.

Deep neural networks are valuable across various domains, including image classification, speech recognition, and natural language processing. It is essential to note that despite the existence of different types of deep learning architectures, the training and learning process shares common principles. In the next section, we will introduce an example of deep machine learning models to showcase their learning process. This type of neural network operates with an important technique to our thesis called convolution.

## Convolutional Neural Networks

**CNN** stands for Convolutional Neural Network, a type of deep learning neural network typically used in image and video analysis (Zhang u. a., 2021). CNNs consist of numerous layers and can learn the spatial hierarchy of features from input data automatically and adaptively. Different filters are applied to each training data input and the output of every convolution data is the input of the next level. CNN layers perform operations to manipulate the data and learn features specific to the data. The three most common processes are convolution, activation, and pooling. Figure 1.2 is an example of a CNN network with multiple convolutional layers.



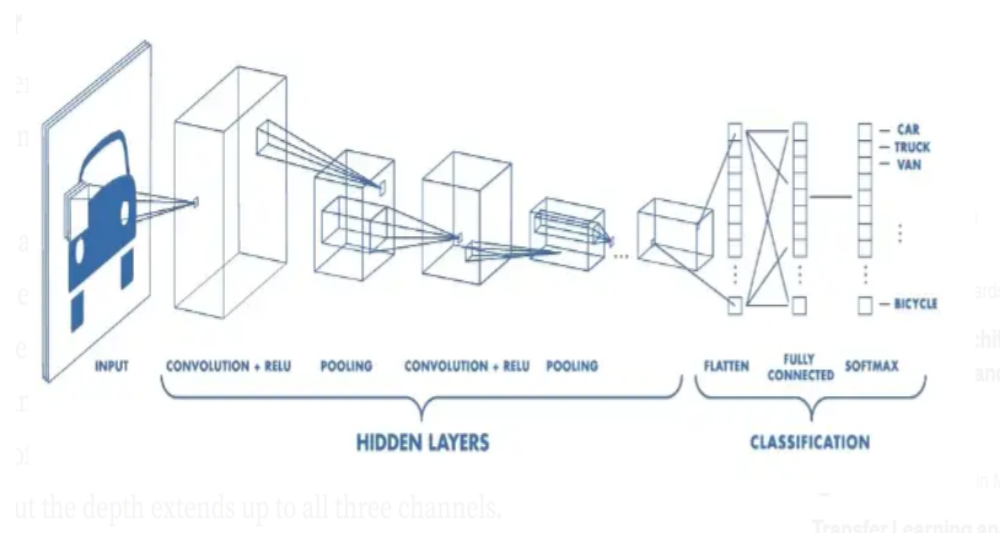


Figure 1.2: Convolutional Neural Network (Ciaburro, 2017)

These processes are duplicated over tens or hundreds of layers to learn to identify different characteristics at each layer:

- Convolutional layer: convolution means passing the input data through a series of convolution filters. Separately filters activate a specific attribute in the data. This layer uses a set of learnable filters to the input data to produce feature maps. The filters are usually short and convolved with the input data using a sliding window. In this step, the CNN model learns how to extract features from the input data.
- The activation function: an activation function, ReLu for example, connects hidden layers and allows faster and more effective training by mapping negative output to zero and preserving positive values, this function is mostly used in hidden layers but can be replaced by other similar activation functions.
- Pooling layer: this layer simplifies the output using nonlinear downsampling. It takes the maximum, average, or other summary statistics of the output values in small sub-sections of the feature map. This helps reduce the number of parameters the network has to learn and the computation required in the next layers.
- Normalisation: also called flattening applies a normalization technique (for example batch normalization) to improve network stability and accelerate convergence during the training process.
- Fully connected layer: it connects all the neurons of the previous layer to the neurons of the next layer. Neurons in this layer have full connectivity with all other activated neurons to generate the final output of the network.
- Activation function: after learning the features the CNN moves to classification. The next-to-last layer is a fully connected layer that outputs a K-dimensional (where K is the number of predictable classes) vector containing the probabilities per class of the data to be classified. This function is applied elementwise to the output of

the earlier layer to introduce nonlinearity into the network and provide the final classification output. Typically used activation functions are Sigmoid and Tanh.

### Optimisation and loss function

In deep learning problems, usually, a loss function (Luger, 2005) is defined to calculate the training error which means the difference between the model-predicted output and the true label. Then an optimization algorithm tries to minimize the loss. Gradient descent is a commonly used optimization algorithm to locate the minima of a function. Many types of loss functions exist and the choice of loss function relies on the specific learning task. One of the most common loss functions used in deep learning is the **Categorical Cross-Entropy** (Zhang u. a., 2021) loss. This function is typically used for multi-class classification tasks, it measures the difference between the predicted likelihood of the correct class and the true output. For a dataset with the size  $k$ , the equation of the Categorical Cross-Entropy loss function is in equation 1.6.

$$CE = - \sum_i^k \hat{y}_i \log(y_i) \quad (1.6)$$

After defining the loss, the gradient has to be calculated in order to backpropagate the loss through the network and tune the network parameters to optimize the defined loss function.

In order to backpropagate through the net and optimize the defined loss function by tuning the network parameters. Gradient descent updates the weights using a derivative function and the learning rate parameter to control the rate of change, the algorithm is in Figure 1.3.

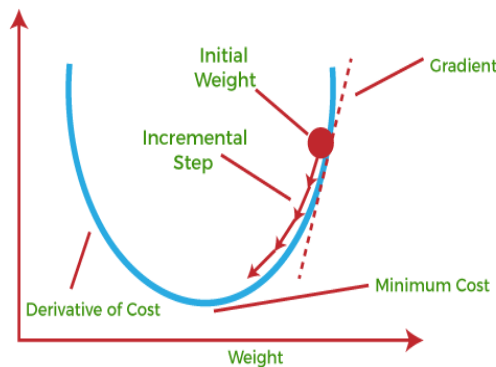


Figure 1.3: Gradient descent convergence (Panagiotis, 2023)

1. Originally, the weights of the neural network are initialized randomly.
2. The network predicts  $\hat{y}$  the labels of individuals in the learning set and then the corresponding loss for example using the Categorical Cross-Entropy loss.
3. Each weight  $w_j^t$  are changed gradually to  $w_j^{t+1}$  so that the loss is minimised :

$$w_j^{t+1} = w_j^t - \alpha \frac{dl}{dw_j} \quad (1.7)$$

the partial derivative  $\frac{dl}{dw_j}$  is a calculus tool that gives the derivative of the cost  $l$  function and finds its minima.

The learning rate  $\alpha$  is a hyperparameter that handles the incremental step size at which the optimizer updates model weights during training. A smaller learning rate may slow down convergence but reduces the possibility of exceeding the optimal solution. A larger learning rate means that the optimizer makes larger updates to the weights and may result in faster convergence but increases the likelihood of exceeding the optimal solution. The optimal learning rate may differ depending on the dataset and CNN architecture.

4. Repeat steps 3 for a maximum number of iterations or until the loss function reaches a satisfactory minimum loss (cost) value.

The final set of parameters acquired after the training process is the optimal set of parameters for the training dataset.

### 1.2.3 Learning on graphs

Given an input graph, traditional machine learning models, such as artificial neural networks, extract features at nodes, links, and graphs, and learn a model that maps the feature input to an output label without putting under consideration the link between input data nodes. As observed in the figure below, the current deep-learning toolkit is designed for simple strings and meshes. For example, as illustrated in Figure 1.4, a convolutional neural network is explicitly defined on grid-structured data (images), while a recurrent neural network is defined for strings (text). For machine learning on a generic graph, a new type of deep learning architecture is needed since the network is much more complex with arbitrary size and complex topology.

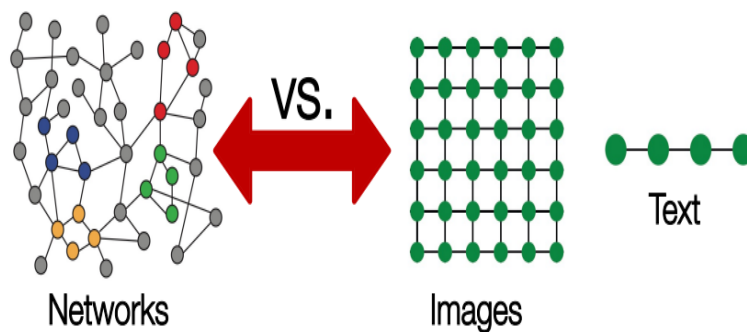


Figure 1.4: Different data structures (Leskovec, 2023)

## 1.3 Machine learning for graphs

Graph formalism is powerful because it emphasizes the relationships between points rather than just their attributes. The same graph formalism can be applied to represent social networks, interactions between drugs and proteins, or the connections between atoms. Graphs go beyond being a refined framework, they also provide a mathematical foundation to enable the investigation, understanding, and learning from complex real-world systems.

### 1.3.1 What is a graph?

To provide a clear definition of graph data, we can say that a graph is a structure composed of nodes and edges. Typically, we represent a graph as  $G = (V, E)$ , where  $V$  is a set of nodes and  $E$  denotes the edges that connect them. An edge  $(u, v) \in E$  means a connection from node  $u \in V$  to node  $v \in V$ . For example, we consider the Zachary Karate Club Network shown in Figure 1.5. This network represents the friendships among members of a karate club studied by Wayne W. Zachary from 1970 to 1972.

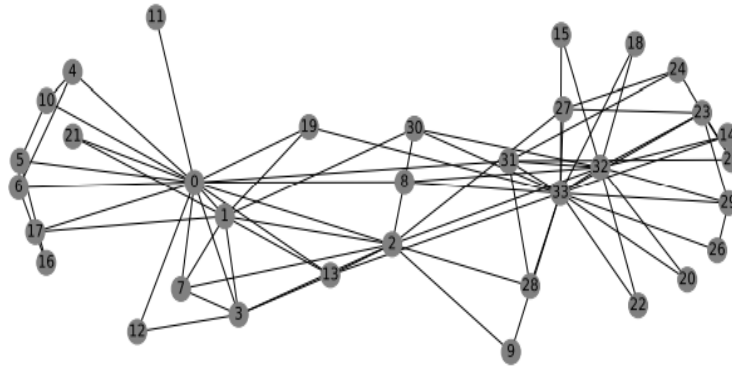


Figure 1.5: Zachary karate club network (PyG, 2023)

#### Adjacency matrix

The graph structure and nodes are represented using an adjacency matrix, denoted as  $A$ , which is a rectangular matrix with dimensions  $|V| \times |V|$ . In this matrix, the nodes are ordered so that each node corresponds to a separate row and column. The presence or absence of edges between nodes is indicated by the entries in this matrix. Specifically, for an edge  $(u, v)$  in the graph, the corresponding entry  $A[u, v]$  in the adjacency matrix is set to 1, indicating the presence of the edge. If there is no edge between nodes  $u$  and  $v$ , then  $A[u, v]$  is set to 0.

#### Graph laplacians

The graph Laplacian is primarily employed in graph spectral processing due to its more beneficial mathematical properties compared to the adjacency matrix. This matrix representation of graphs is created through several modifications of the adjacency matrix. One basic Laplacian is the unnormalized Laplacian, defined as:

$$L = D - A \quad (1.8)$$

Where  $A$  denotes the adjacency matrix and  $D$  denotes the degree matrix. The Laplacian matrix of a simple graph has several important properties, such as that it has non-negative eigenvalues  $|V|$ :

$$0 = \lambda_{|V|} \leq \lambda_{|V|-1} \leq \dots \leq \lambda_1 \quad (1.9)$$

also, the geometric multiple of Laplacian eigenvalue 0  $L$  represents how many connected nodes are in the graph. There are two common normalized variants of the Laplacian in addition to the unnormalized Laplacian. Symmetric normalized Laplacian:

$$L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \quad (1.10)$$

and random walk Laplacian is:

$$L_{RW} = D^{-1}L \quad (1.11)$$

These two formulas have similar properties to the Laplacian matrix, although their algebraic properties differ by small constants due to normalization.

### Directed and undirected graph

Undirected graphs represent a network where the connection between two nodes is always mutual. Meaning, if two nodes  $v$  and  $u$  are connected by an edge in an undirected graph, then  $u$  is connected to  $v$ , and  $v$  is connected to  $u$ . On the other hand, a directed graph is a network where connections are not mutual and an edge connects  $v$  to  $u$  does not necessarily connect  $u$  to  $v$ . If a network includes solely undirected edges the adjacency matrix  $A$  will be symmetric, while if the graph was directed then  $A$  will not necessarily be symmetric.

### Multi-relational graph

Another differentiation between graph types is heterogeneous or homogeneous graphs. Heterogeneous graphs or multi-relational graphs have different types of edges between nodes while homogenous graphs have only one type of relations. In the case of financial fraud, the link between the user and the provider must be different from the link between the user and friends. In this case, the edge notation extends to include a set of edge or relationship type  $r$ ,  $(u, r, v) \in E$ , and the adjacency matrix will also be defined differently as  $A_r$  for each edge type.

### Node features

In certain scenarios, graphics or visual representations may be linked to attribute or feature information, such as profile photos associated with users in a social network. Generally, each node has a set of features and this set of features is represented by  $X \in R^{|V| \times m}$ , where  $|V|$  denotes the number of nodes and  $m$  denotes the feature dimension. It is important to note that the ordering of nodes in this matrix aligns with the order in the adjacency matrix.

## 1.3.2 Graph node-level features

Since our goal is to classify data points, it is important to characterize the structure and status of a node in the network.

### Node degree

The most straightforward characteristic to examine for a node is its node degree, denoted as  $d_v$ . The node degree of node  $v$  represents the number of edges it has connecting to other neighboring nodes. Mathematically, it can be calculated by summing up the entries of the adjacency matrix  $A$  for that particular node:

$$d_v = \sum_{u \in V} A[v, u] \quad (1.12)$$

Node degree is generally considered an important and valuable statistic, often being one of the most significant characteristics of a node. It can also be represented using the

degree matrix  $D$ , which is a diagonal matrix. In the degree matrix, each diagonal entry  $D_{ii}$  corresponds to the degree of the respective node  $v_i$ :

$$D_{ii} = d_{v_i} \quad (1.13)$$

However, it's worth noting that while node degree measures the number of neighboring nodes, it doesn't capture their relative importance or significance.

### Node eigenvector

To gain more understanding of node significance, we can employ more robust metrics such as node centrality. Node centrality provides a nuanced assessment of a node's importance. One approach to measuring node centrality is by calculating the center of eigenvectors, the idea is that a node is considered significant if it is surrounded by influential neighbors  $u \in N(v)$ . In particular, the center of the eigenvectors for a node  $v$ , denoted as  $e_v$ :

$$e_v = \frac{1}{\lambda} \sum_{u \in N(v)} A[v, u] e_u \Leftrightarrow \lambda E = AE \quad (1.14)$$

where  $\lambda$  is the maximum eigenvalue of the adjacency matrix  $A$ , and  $E$  represents the center vector of the nodes in the graph.

### Clustering coefficient

Measure the connectivity of neighboring nodes of  $v$  and the ratio of closed triangles in the local neighborhood of a node. The common local variation of the clustering coefficient is calculated as follows (Hamilton u. a., 2017):

$$c_v = \frac{|(v_1, v_2) \in E : v_1, v_2 \in N(v)|}{\binom{d_v}{2}} \in [0, 1] \quad (1.15)$$

the numerator of this equation calculates the number of edges between the neighbors of a node  $v$  and the denominator calculates how many pairs of nodes there are. The clustering coefficient gets its name from calculating how tight a node's neighborhood is. A clustering factor of one would indicate that all the neighbors of  $v$  are also neighbors of each other in the neighborhood of  $u$ .

## 1.3.3 Node embedding

This section mentions techniques that help machine learning models learn how to represent graph nodes based on their structure and features to enhance the learning process, this technique is called node embedding illustrated in Figure 1.6.

The objective of node embedding techniques is to represent nodes as compact, low-dimensional vectors that capture both their position in the graph and the characteristics of their local graph neighborhood. These techniques operate within the framework of an encoder-decoder model, where the encoder component maps each node in the graph to a vector or embedding with reduced dimensions. Subsequently, the decoder leverages these low-dimensional node embeddings to reconstruct information about the original neighborhood structure of each node. The underlying concept is illustrated in Figure 1.7, providing a visual summary of this approach.

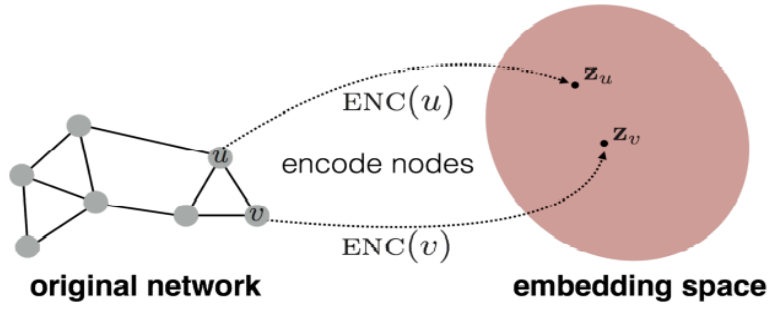


Figure 1.6: Illustration of the node embedding problem (Hamilton u. a., 2017)

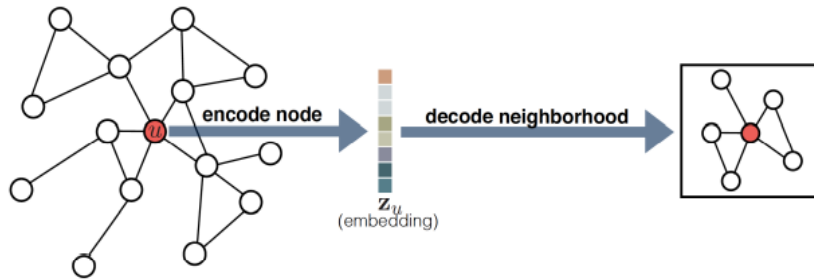


Figure 1.7: Overview of the encoder-decoder approach (Hamilton u. a., 2017)

### Encoder

The encoder maps nodes  $v \in V$  to vector embeddings  $z_v \in R^d$ . In the simplest case, the encoder has the following signature.

$$enc : V \rightarrow R^d \quad (1.16)$$

Indicating that the encoder takes the node identification as input to extend the node integration. The encoder depends on what is known as a shallow integration approach, where its role is to look for nodes embedding based on their IDs. In other words:

$$enc(v) = z_v \quad (1.17)$$

In this approach, a matrix  $Z \in R^{|V| \times d}$  is utilized to store the embedding vectors for all nodes, with  $z_v$  representing the corresponding row in  $Z$  for node  $v$ . While this method is effective, the lack of feature sharing among nodes in the encoder is a significant limitation.

### Decoder

The role of the decoder is to reconstruct particular network statistics using the node embeddings produced by the encoder. For example, when given a node represented by  $z_v$ , the decoder can try to predict the set of neighbors  $N(v)$  or the rows  $A[v]$  in the adjacency matrix graph. Although there can be multiple decoders, it is common to refer to pairwise decoders in the following manner (Hamilton u. a., 2017):

$$dec : R^d \times R^d \rightarrow R^+ \quad (1.18)$$

Pairwise decoders can be interpreted as functions that estimate the relationship or similarity between pairs of nodes. The goal is to optimize the encoder and decoder to minimize the reconstruction loss, which can be defined as follows:

$$dec(enc(u), enc(v)) = dec(z_u, z_v) \approx S[u, v] \quad (1.19)$$

Where,  $S[u, v]$  represents a graph-based similarity measure between nodes. For instance, predicting whether two nodes are neighbors would correspond to  $S[u, v] = A[u, v]$ .

### Optimisation of the Encoder-Decoder Model

To achieve the reconstruction objective of the decoder, the standard approach is to minimize the observed reconstruction loss  $L$  on a set of training node pairs  $D$ :

$$L = \sum_{(u,v) \in D} l(DEC(z_u, z_v), S[u, v]) \quad (1.20)$$

here,  $l \in R$  is a loss function that quantifies the difference between the decoded values  $dec(z_u, z_v)$  and the actual value  $S[u, v]$ .

### Limitations

In classical embedding approaches, the encoder pattern that assigns embeddings to nodes typically involves an embedding search, where each node in the graph is assigned a unique embedding. While this approach has achieved notable success, it is important to acknowledge that shallow embedding methods suffer from several significant drawbacks:

1. Firstly, shallow embedding approaches lack parameter sharing between nodes. This means that the encoder optimizes a distinct  $z$  vector for each node, resulting in inefficient parameter utilization (Hamilton u. a., 2017).
2. Secondly, shallow embedding methods do not effectively leverage node features during the encoding process. Many graph datasets contain valuable information about node features that could enhance the encoder's performance.
3. Finally, and perhaps most importantly, shallow integration methods are inherently transductive, meaning they can only generate embeddings for nodes present during the training phase.

Considering these limitations, shallow encoders are replaced by more sophisticated encoders, which we will explore in the next section.

### 1.3.4 Graph Neural Networks

Graph Neural Networks (GNNs) are a general framework for describing neural networks on graphical data, despite their fundamental differences in the data structure, Graph neural networks share some similarities with artificial neural networks. Both of them are composed of multiple layers of interconnected processing units, where each layer conducts a different type of computation to convert the input data into a useful output. Additionally, both ANNs and GNNs models learn complex nonlinear relationships from input data using backpropagation.



Graph neural networks are complex encoders capable of learning to produce the most representative node representation or embedding. Given an input graph  $G = (V, E)$ , along with a set of node features  $X \in R^{d \times |V|}$ , GNNs can use this information to generate node embeddings  $z_v, \forall v \in V$ . The basic idea is to generate node representations using the structure of the graph, as well as any existing attribute information. There are two main types of GNNs: spectral and spatial. In this section, we will start by presenting spatial GNNs that focus on the direct exchange of data between neighboring nodes and update the node embedding based on the features of their neighbors. Spectral GNNs on the other hand leverage the spectral properties of the graph Laplacian matrix to process node features and it will be discussed in the following sections. A GNN can be represented in Figure 1.8.

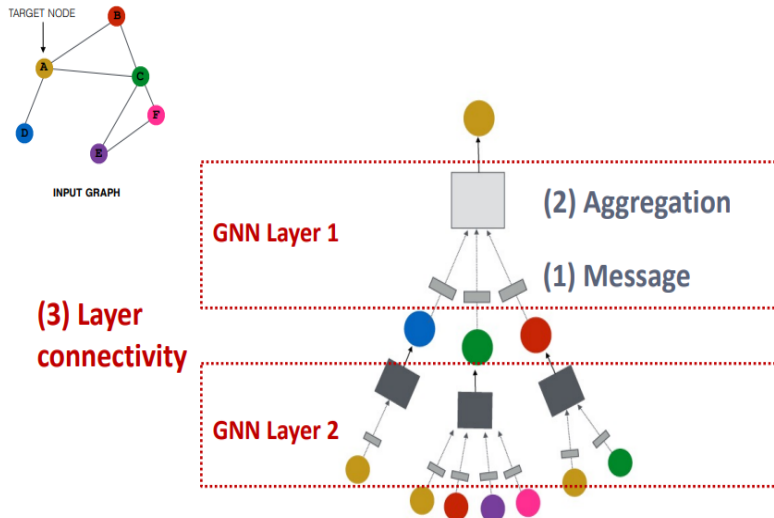


Figure 1.8: GNN layers (Leskovec, 2023)

Where the GNN layer creates embeddings based on local network neighborhoods and compresses a set of nodes into a single node in two steps messaging and then aggregation process. Each layer propagates graph data and creates node embedding based on local network neighborhoods. Nodes have new embeddings at each layer, built-in  $layer_0$  of node  $h_v^0$  is its input feature  $X_v$ , and built-in  $layer_k$  takes input from nodes with a hop  $k$ .

### Neural Message Passing

In each iteration of message-passing, a hidden embedding  $h_v^{(k)}$  is updated based on information aggregated from  $v$ 's graph neighborhood  $N(v)$ . To avoid the information of the node  $v$  itself being lost, the calculation of  $h_v^{(k)}$  must rely on  $h_v^{(k-1)}$  because each message collects data from neighboring nodes. Therefore, the two messages will be calculated as follows:

$$m_v^{(k)} = B^{(k)} h_v^{(k-1)} \quad (1.21)$$

$$m_u^{(k)} = W^{(k)} h_u^{(k-1)} \quad (1.22)$$

$m_v^{(k)}$  represents the  $v$  message,  $h_v^{(k-1)}$  is the node  $v$  representation in the previous layer, and  $B^{(k)}$  is a trainable parameter that calculates the weight of the node  $v$ .  $m_u^{(k)}$  represents

the node  $v$ 's  $k$  hop neighbor  $u \in N(v)$ ,  $h_u^{(k-1)}$  is the previous layer representation of  $u$ , and  $W^{(k)}$  is a trainable parameter that calculates the weight of the node  $v$  neighbors.

### Neighborhood Aggregation

Each node will aggregate the messages from node  $v$ 's neighbors with a Sum(), Mean(), or Max() aggregator. For example:

$$\sum_{u \in N(v)} W^{(k)} h_u^{(k-1)} + B^{(k)} h_v^{(k-1)} \quad (1.23)$$

The most basic operation for neighborhood aggregation involves summing the embeddings of the neighboring nodes. However, this approach can be unstable and highly sensitive to node degrees. For example, consider a scenario where node  $v$  has 100 times as many neighbors as node  $u$ . In this case, it is reasonable to expect that  $\|\sum_{v \in N(v)} h_v\| \gg \|\sum_{u \in N(u)} h_u\|$ . The substantial difference in volume between nodes can result in numerical instabilities and optimization difficulties. To overcome this challenge, one solution is to normalize the aggregation operation considering the degrees of the involved nodes. The simplest approach is to compute an average instead of a sum:

$$m_{N(v)} = \frac{\sum_{u \in N(v)} W^{(k)} h_u^{(k-1)}}{|N(v)|} \quad (1.24)$$

### Update Methods

In each iteration of message-passing within a Graph Neural Network, the hidden embedding  $h_v^{(k)}$  of a node  $v \in V$  is modified using aggregated information from its graph neighborhood  $N(v)$ . To provide a more abstract description of the GNN framework as a series of customizable message-passing iterations, we can define the update process as follows:

$$h_v^{(k+1)} = \text{UPDATE}^{(k)} \left( h_v^{(k)}, \text{AGGREGATE}^{(k)} \left( h_u^{(k)}, \forall u \in N(v) \right) \right) \quad (1.25)$$

$$h_v^{(k+1)} = \text{UPDATE}^{(k)} \left( h_v^{(k)}, m_{N(v)}^{(k)} \right) \quad (1.26)$$

where the terms UPDATE and AGGREGATE refer to differentiable functions that can be customised, and  $m_{N(v)}$  denotes the aggregated message obtained from the graph neighborhood  $N(v)$  of node  $v$ . Once the  $K$  layers of message-passing are executed in the GNN, the resulting output of the final layer, denoted as layer  $K$ , can be employed to represent the node embedding  $z$  for each node.

$$z_v = h_v^{(K)}, \forall v \in V \quad (1.27)$$

To make the abstract GNN framework described above more concrete and implementable, specific initializations for the UPDATE and AGGREGATE functions are required. One basic GNN framework, which is a simplified version of the original GNN models proposed by Merkwirth and Lengauer (Merkwirth und Lengauer, 2005), can be defined in equation 1.28.

$$h_v^{(k+1)} = \sigma \left( \sum_{u \in N(v)} W^{(k)} h_u^{(k-1)} + B^{(k)} h_v^{(k-1)} \right). \quad (1.28)$$

The message propagation in this GNN framework resembles that of a standard perceptron, as it involves linear operations followed by element-wise non-linearity. Initially, it aggregates incoming messages from neighboring nodes and then combines the neighbor information with the pre-embeddings of the node using linear transformations. Finally, it applies the non-linear activation function element-wise.

### Node classification

After the graph representation is learned, the model goes through the prediction step where it is possible to have three different prediction tasks or heads as depicted in Figure 1.9:

- Node-level tasks: node classification among  $C$  categories or regression on  $C$  targets based on its embeddings.
- Edge-level tasks: relation prediction using pairs of node embeddings.
- Graph-level tasks: graph label prediction using all the node embeddings in it.

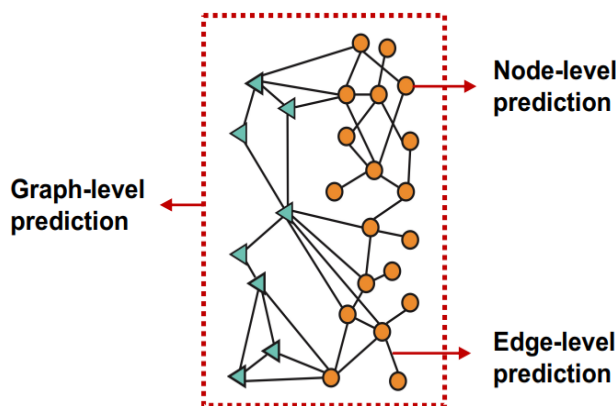


Figure 1.9: GNN task levels (Leskovec, 2023)

Consider an example of a large social network dataset comprising millions of users, some of whom are bots. It is crucial to identify these bots for various reasons, such as enforcing social network terms of service. However, manually inspecting each user to determine if they are bots would be a costly and time-consuming task. Hence, we aim to develop an automated model capable of classifying users as bots or non-bots. This situation represents a node-level classification task, where the goal is to predict the label  $y_v$  for each node  $v$ . Node classification, Figure 1.10, may initially seem similar to standard supervised classification, but there are notable differences. The most significant distinction arises from the fact that the nodes in a graph are not independent and identically distributed.

In traditional supervised machine learning models, each data point is typically assumed to be statistically independent of all other data points. Violation of this assumption necessitates modeling the relationships between data points. Moreover, machine

learning algorithms often assume that the data points are identically distributed. When this assumption is invalid, the generalization of the model to new data points cannot be guaranteed. Node classification fundamentally challenges these assumptions. Rather than modeling a set of isolated data points, we deal with interconnected nodes. Many successful node classification methods capitalize on these node connections. One particularly popular approach is exploiting the concept of homophily, which suggests that nodes tend to share attributes with their neighboring nodes in the graph.

For instance, individuals tend to be friends with others who share similar interests or demographics. Building on this concept, machine learning models can be constructed to assign similar labels to neighboring nodes in the graph. Thus, when constructing a node classification model, it is crucial to embrace this concept and model the interconnections between nodes, rather than treating nodes as independent entities.

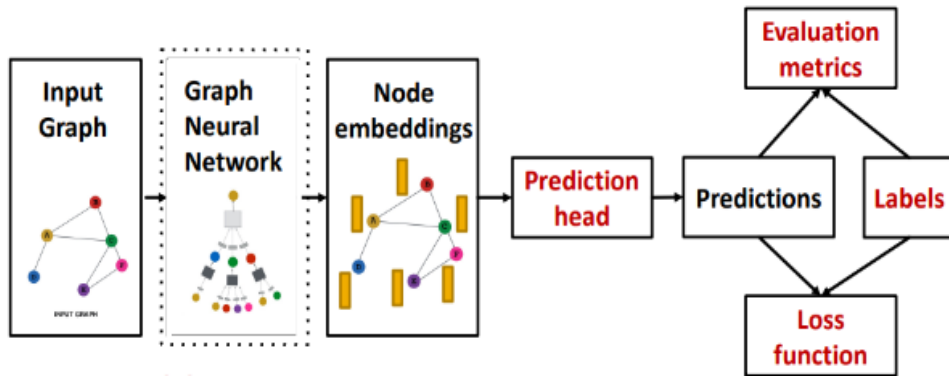


Figure 1.10: GNN training for node classification (Leskovec, 2023)

Where prediction head means prediction task, in the case of node level task it is possible to make prediction directly using node integration. After GNN computation, we have  $d$  dim node embeddings  $h_v^{(k)} \in R^d, \forall v \in V$ , assume we want to make  $l$ -way prediction which means we classify among  $l$  categories:

$$\hat{y}_v = W^{(H)} h_v^{(k)} \quad (1.29)$$

where  $W^{(H)} \in R^{l \times d}$  maps node embeddings from  $h_v^{(k)} \in R^d$  to  $\hat{y}_v \in R^l$  so that we can calculate the loss between the prediction  $\hat{y}_v$  and the label  $y_v$ . The loss is defined using a softmax classification function and negative log-likelihood loss:

$$L = \sum_{v \in V_{train}} -\log(\text{softmax}(\hat{y}_v, y_v)) \quad (1.30)$$

Here, it is assumed that  $y_v$  is a one-hot vector showing the class of training node  $v \in V_{train}$ .  $\text{Softmax}(\hat{y}_v, y_v)$  is used to represent the predicted probability that the node belongs to class  $y_v$ , computed via the softmax function:

$$\text{Softmax}(\hat{y}_v, y_v) = \sum_{i=1}^c y_v[i] \frac{\exp \hat{y}_v^T w_i}{\sum_{j=1}^c \exp \hat{y}_v^T w_j} \quad (1.31)$$

where  $w_i \in R^d$ ,  $i = 1, \dots, c$  are trainable parameters. Finally, the metrics for evaluating the performance of the graph are the same metrics of typical machine learning models. For binary classifiers, some metrics are sensitive to classification thresholds, such as accuracy, precision, and recall.

### 1.3.5 Graph data analysis methods

As we mentioned earlier, in section 1.3.4, in the graph data analysis domain, two distinct methods (McPherson u. a., 2001; Wu u. a., 2020) have emerged for analyzing the structure and characteristics of a given graph, spectral and spatial analysis:

- The spatial approach to graph analysis pertains to the study of the geometric and spatial properties of a graph, which contains an examination of the network's nodal structure and the spatial relationships between nodes and edges. The spatial analysis involves the aggregation of a node's local neighborhood information to update its representation. This approach updates node embeddings and features using the features of its neighboring nodes.
- The spectral analysis approach, on the other hand, involves using the graph Laplacian eigenvalues to process graph data. Graph signals in this case are transformed into the frequency domain and processed using techniques such as Fourier transforms and convolution. The key distinction between spatial and spectral processing in GNNs is the method of aggregating information from neighboring nodes.

In section 1.3, we investigate the techniques and methodologies used for spatial analysis in graphs. Subsequently, in the following section 1.4, we introduce the spectral analysis methods.

## 1.4 Convolution on graphs

In this section, we will dive deep into spectral-based methods theoretical foundations, we will also explore how to perform the convolution operations we studied in section 1.2.2 and introduce the concept of convolution on graphs.

### 1.4.1 Graph signals

For a discrete time-varying signal,  $f(t_0), f(t_2), \dots, f(t_{N-1})$  can be viewed as resembling a chain (or cycle) graph. In Figure 1.11, at each time point, denoted as  $t$ , we can represent a node in a chain graph as a signal value determined by the function  $f(t)$ . This allows us to express the signal as a vector  $f \in R^N$ , where each dimension corresponds to a unique node in the chain graph. The edges in the graph represent the propagation of the signal over time, which enables us to describe operations using the adjacency and Laplacian matrices of the graph.

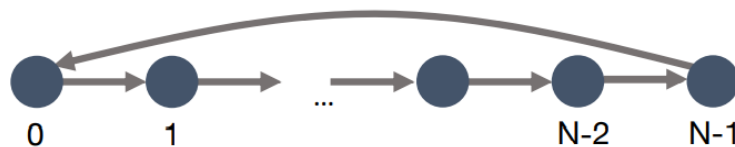


Figure 1.11: Cyclic time series represented as a chain graph (Hamilton u. a., 2017)

In the case of a chain graph, where nodes are arranged linearly, we can represent the

connections between nodes using an adjacency matrix called a circulant matrix, denoted as  $A_c$ . This matrix has a specific structure. This matrix has the following properties:

$$A_c[i, j] = \begin{cases} 1 & \text{if } j = (i + 1)_{\text{mod } N} \\ 0 & \text{otherwise} \end{cases}. \quad (1.32)$$

In simpler terms, the circulant matrix  $A_c$  indicates the connections between adjacent nodes in the chain graph. The unnormalized Laplacian matrix  $L_c$  for this graph can be defined as the difference between the identity matrix  $I$  and the adjacency matrix  $A_c$ :

$$L_c = I - A_c. \quad (1.33)$$

To understand the operations of time shifts and differences in the graph, we can use the adjacency and Laplacian matrices. Applying the adjacency matrix  $A_c$  to a signal  $f$  at time  $t$  means propagating the signal from the current node to its immediate neighbor at time  $t + 1 \pmod{N}$ , where  $N$  is the total number of nodes in the chain. We can interpret time shifts using multiplication by the adjacency matrix:

$$(A_c f)[t] = f[(t + 1) \pmod{N}]. \quad (1.34)$$

On the other hand, multiplying the signal  $f$  by the Laplacian matrix  $L_c$  calculates the difference between the signal at a particular node  $t$  and its neighboring node at time  $t + 1 \pmod{N}$ . The difference operation can be represented by multiplication with the Laplacian matrix:

$$(L_c f)[t] = f[t] - f[(t + 1) \pmod{N}]. \quad (1.35)$$

By understanding these operations, we gain insights into the relationship between the adjacency and Laplacian matrices of the graph and how they relate to signal shifts and differences. Graph filters, which are commonly used in GNNs, take advantage of these concepts. They operate in the spectral domain and modify the frequency content of graph signals. By designing graph filters with specific frequency response characteristics, GNNs can emphasize or suppress certain frequency components in the graph signals. This allows GNNs to focus on relevant information for different tasks and applications, improving their performance in analyzing and processing graph-structured data.

### 1.4.2 Graph Convolutional Networks

Graph Convolutional Network or GCN, is a type of GNN that employs a spectral-based approach. This popular model employs the symmetric normalized aggregation as well as the self-loop update method as a node representation update approach. The GCN model is defined in Figure 1.12.

The GCN model applies multiple convolutional layers to the input graph, using the polynomial  $I + A$  and a non-linearity function to transform the node features. This process generates node embeddings that are further used for prediction in a typical GNN. The convolution layer in GCN is defined as:

$$h^{(k)} = \sigma(\tilde{A} h^{(k-1)} W^{(k)}) \quad (1.36)$$

Here,  $h^{(k)}$  represents the node representations at the  $k$ -th layer. The matrix  $\tilde{A}$  is a normalized version of the adjacency matrix, and  $W^{(k)}$  is a parameter matrix that can be learned. The formula for  $\tilde{A}$  is given by:

$$\tilde{A} = (D + I)^{-\frac{1}{2}} (I + A) (D + I)^{-\frac{1}{2}} \quad (1.37)$$

where  $D$  is the degree matrix and  $A$  is the adjacency matrix. The GCN model can be seen as a modification of the fundamental GNN message-passing approach. By combining graph convolutions with non-linear functions and trainable weight matrices, the basic GNN formulation is obtained:

$$h^{(k)} = \sigma \left( \sum_{u \in N(v)} \frac{h_u^{(k-1)} W_u^{(k)}}{N(v)} + h_v^{(k-1)} W_v^{(k)} \right) \quad (1.38)$$

In this formula,  $N(v)$  represents the set of neighboring nodes of node  $v$ .  $W_u^{(k)}$  denotes the weight matrix of the neighboring nodes,  $h_u^{(k-1)}$  represents the neighboring node embedding in the previous layer,  $h_v^{(k-1)}$  is the node embedding in the previous layer, and  $W_v^{(k)}$  is the node weight.

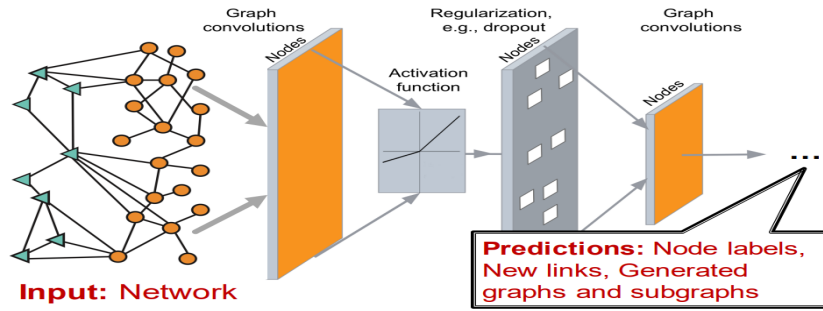


Figure 1.12: Graph Convolutional Networks (Kipf und Welling, 2016)

During the message-passing process in GNNs, there can be a problem called "over-smoothing," where important information is lost or distorted. This occurs due to the repeated application of graph convolutional operations across multiple layers, which causes node features to converge towards a common representation. As a result, finer details and local distinctions that are crucial for accurate predictions or downstream tasks may be erased. To address the over-smoothing problem, various techniques have been proposed, such as introducing skip connections to preserve local information, using adaptive or learnable aggregation functions, or exploring alternative architectures that maintain fine-grained features.

Although GCN offers advantages like handling graphs of different sizes and structures, capturing both local and global graph structure, and interpretability, it also has limitations. GCN can be seen as low-pass filters, retaining only low-frequency information from the input graph. It lacks the ability to adaptively capture useful information beyond the low-frequency component. This non-learnable graph filtering may restrict its ability to effectively capture and utilize high-frequency information for certain tasks.

## 1.5 Conclusion

In conclusion, this chapter has provided a comprehensive introduction to Graph neural networks and their key components and metrics. We have also dived into the process of learning node representations from both spatial and spectral perspectives. The next chapter will further explore the theoretical background of this work, which will help establish a deeper understanding of GNNs.

# Chapter 2

## State of the art of GNN based Anomaly Detection methods

### 2.1 Introduction

In this chapter, our goal is to dive deep into the intriguing task of anomaly detection on graphs and shed light on its most effective approaches. To achieve a comprehensive understanding, we focus on exploring representative anomaly detection techniques that leverage Graph Neural Networks. We kick off by providing an overview of the latest advancements in anomaly detection utilizing GNNs. Afterward, we dive into the architectures of these models and carefully analyze the strategies of the most recent methods. To complete our analysis, we thoroughly examine the evaluation results of these models, offering valuable insights into the practical applicability of each approach in real-world databases.

### 2.2 Overview on the existing Anomaly Detection approaches

Graph-structured data has grown in popularity due to its capacity to represent complicated real-world systems. As a result, there is increased interest in spotting anomalies in graphs. Graph neural networks have emerged as effective models for graph analysis, and numerous methods have been presented to handle the problem of attributed network anomaly detection using GNNs. The underlying idea in GNN-based anomaly detection is to leverage the expressive capabilities of GNNs to compute and learn node representations, aiming to distinguish anomalous nodes from normal nodes in the embedding space.

DOMINANT (Ding u. a., 2019), is an unsupervised GCN autoencoder that computes anomaly ranking scores and facilitates anomaly detection in attributed networks. By utilizing unsupervised learning and considering the graph structure and attributes, DOMINANT addresses the challenge of obtaining data labels. Additionally, it can capture data nonlinearities and complex interactions between features. However, it remains susceptible to the camouflage behavior of anomalous nodes. Anomalies often connect with normal nodes and mimic their features to evade detection, which is challenging in graphs that adhere to the Homophily assumptions (McPherson u. a., 2001), encouraging similarity and smoothing differences during the aggregation process.



On the other hand, the Geniepath (Liu u. a., 2019) model introduces novel aggregation methods that filter graph nodes from neighbors at varying distances to unveil anomalies. Geniepath tackles the issues faced by GCNs in recognizing meaningful receptive paths that ensure the detection of camouflaged anomalies. It incorporates an adaptive path layer with two complementary features designed for breadth and depth exploration. The former determines the significance of neighbors of different sizes, while the latter extracts and filters aggregated nodes from the neighbors. This model can be optimized through supervised or unsupervised learning, but it does not address inconsistent node behavior in graphs.

To address inconsistency problems in anomaly detection, GraphConsis (Liu u. a., 2020) combines techniques and proposes a supervised model built upon a heterogeneous graph with multiple relations. CARE-GNN (Dou u. a., 2020a) is another supervised model that recognizes two types of camouflage: feature camouflage and relational camouflage. It employs reinforcement learning to determine the optimal number of neighbors to select and aggregate across different relationships. While CARE-GNN is considered a state-of-the-art anomaly detection algorithm, it is limited to low-frequency filtering (Balcilar u. a., 2021). In the case of the semi-supervised model AMNet (Chai u. a., 2022), it learns to adaptively combine low- and high-frequency information to embed nodes for discriminating abnormalities. Instead of applying a global low-pass filter, AMNet introduces a family of multi-frequency training filters to capture both low and high-frequency graph signals simultaneously. It also incorporates a node-level attention mechanism to logically merge information from different feature frequencies at each node.

It is noteworthy that the aforementioned models can be categorized into spectral-based and spatial-based models. Both methodologies are suitable for anomaly detection, but the choice is determined by the properties of the graph data and the sort of anomaly detection task at hand. In the following section, we will look at the architecture of these models to see how they handle anomaly detection.

## 2.3 Spatial Anomaly Detection models

In this section, we focus on spatial anomaly detection, these methods play a crucial role in identifying anomalies and they are the most common approaches.

### 2.3.1 GraphConsis framework

GraphConsis (Liu u. a., 2020) is a framework specifically designed to address the issue of inconsistencies in anomaly detection on multiple relation graphs. The underlying hypothesis of this supervised model is that the inconsistencies arise from the aggregation process utilized in GNN models, which assumes that neighboring nodes possess identical properties and labels (Akoglu u. a., 2015). However, as this assumption becomes less valid, the efficacy of aggregating neighboring data for learning node representations diminishes. To tackle these challenges, GraphConsis introduces three distinct techniques targeting different problems. The first problem addressed is relation inconsistency, which arises due to the presence of multiple types of relationships linking entities. Treating all relationships equally leads to inconsistencies. In response, GraphConsis learns relation attention weights for each node’s neighbors, allowing for a more nuanced handling of

different relationships. The second problem is context inconsistency, where anomalous nodes may connect with ordinary entities, disguising their anomalous nature. To handle this issue, GraphConsis assigns a trainable context embedding to each node. This embedding serves to capture and mitigate context inconsistencies, facilitating the detection of anomalies. The third problem is feature inconsistency, which arises from the aggregation process's impact on the GNN's ability to capture and characterize the unique properties of anomalies. This, in turn, affects the model's detection capabilities. To address feature inconsistency, GraphConsis introduces a consistency score mechanism. This score helps filter out irregular neighborhoods, generating more reliable sampling probabilities and alleviating feature inconsistencies. The framework of GraphConsis is depicted in Figure 2.1.

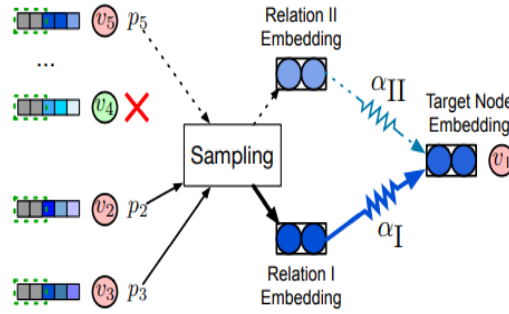


Figure 2.1: GraphConsis framework (Liu u. a., 2020)

### Context Embedding

To address the context inconsistency problem, GraphConsis introduces a trainable context embedding, denoted as  $c_v$ , for each node  $v$ , in addition to utilizing the feature vector  $x_v$ . In the evolution of the first layer of the aggregator, the formulation is modified as follows:

$$h_v^{(1)} = \{x_v \| c_v\} \oplus AGG^{(1)}(\{x_{v'} \| c_{v'} : v' \in N_v\}) \quad (2.1)$$

Here, the symbol  $\|$  represents the concatenation operation. The introduction of context embedding enables the learning of a representation that captures the local structure of the node, aiding in the discrimination of fraudulent behavior or anomalies. By incorporating the context embedding alongside the feature vector, GraphConsis enhances the model's ability to identify and distinguish anomalies within the graph.

### Neighbor Sampling

To address the feature inconsistency problem, GraphConsis introduces a mechanism for sampling related neighbors instead of treating all neighbors equally. The model computes a consistency score between node embeddings using the following formulation:

$$s^{(l)}(u, v) = \exp\left(-\|h_u^{(l)} - h_v^{(l)}\|_2^2\right) \quad (2.2)$$

Here,  $s^{(l)}(u, v)$  represents the consistency score between two nodes at the  $l$ -th layer and  $\|\cdot\|_2^2$  denotes the squared L2 norm of the vector. The model applies a threshold, denoted as  $\epsilon$ , to filter out neighbors that are not consistent. Afterward, each node  $u$  is assigned to

the filtered neighbors  $\check{N}_v$  of node  $v$  with a sampling probability that is normalized based on its consistency score:

$$p^{(l)}(u, v) = s^{(l)}(v, v) \sum_{u \in \check{N}_v} s^{(l)}(u, v) \quad (2.3)$$

This probability assignment ensures that nodes with higher consistency scores have a higher likelihood of being selected as neighbors. By sampling related neighbors and assigning probabilities based on consistency scores, GraphConsis aims to mitigate the feature inconsistency problem and improve the ability of the model to capture the unique characteristics of anomalies within the graph.

### Relation Attention

To address the relation inconsistency problem, GraphConsis incorporates relation information into the aggregation process. For each relation  $r$ , a relation vector  $t_r$  is trained, where  $r = 1, 2, \dots, R$ , to capture the specific relation information that needs to be incorporated. The self-attention mechanism is utilized to assign weights to  $Q$  sampled neighbor nodes. The weight assignment is determined by the following equation:

$$\alpha_q^{(l)} = \frac{\exp\left(\sigma\left(\left\{h_q^{(l)}|t_{r_q}\right\}a^T\right)\right)}{\sum_{q=1}^Q \exp\left(\sigma\left(\left\{h_q^{(l)}|t_{r_q}\right\}a^T\right)\right)} \quad (2.4)$$

Here,  $r_q$  represents the relation of the  $q$ -th sampled neighbor node with respect to the central node  $v$ . The activation function  $\sigma$  is applied element-wise, and  $a$  represents the attention weights shared across all attention layers. The final aggregation function is defined as follows:

$$AGG^{(l)}(\{h_q^{(l-1)}\}_{q=1}^Q) = \sum_{q=1}^Q \alpha_q^{(l)} h_q^{(l)} \quad (2.5)$$

This aggregation function combines the weighted representations of the sampled neighbor nodes to obtain the aggregated representation at layer  $l$ . By incorporating relation vectors and utilizing self-attention, GraphConsis aims to address the relation inconsistency problem and enhance the aggregation process in capturing relevant relation information.

### 2.3.2 Camouflage resistant GNN

Camouflage-resistant graph neural network or CARE-GNN (Dou u. a., 2020a) model alleviates anomalous nodes camouflage behavior in multi-relation graphs in a supervised manner, more precisely CARE-GNN is trained to detect two varieties of camouflage, feature camouflage, and relation camouflage. Feature camouflage (Ge u. a., 2018) happens when an anomalous node changes its behavior, adds special features, or utilizes deep language generation models (Kaghazgaran u. a., 2019) to smooth explicit questionable results. Relation camouflage (Kaghazgaran u. a., 2018) means that anomalous nodes can camouflage themselves by connecting to many normal nodes. CARE-GNN can detect anomalous camouflaged nodes using a label-aware similarity measure and reinforcement learning in the aggregation process. This model improves the GNN aggregation process in three steps depicted in Figure 2.2.

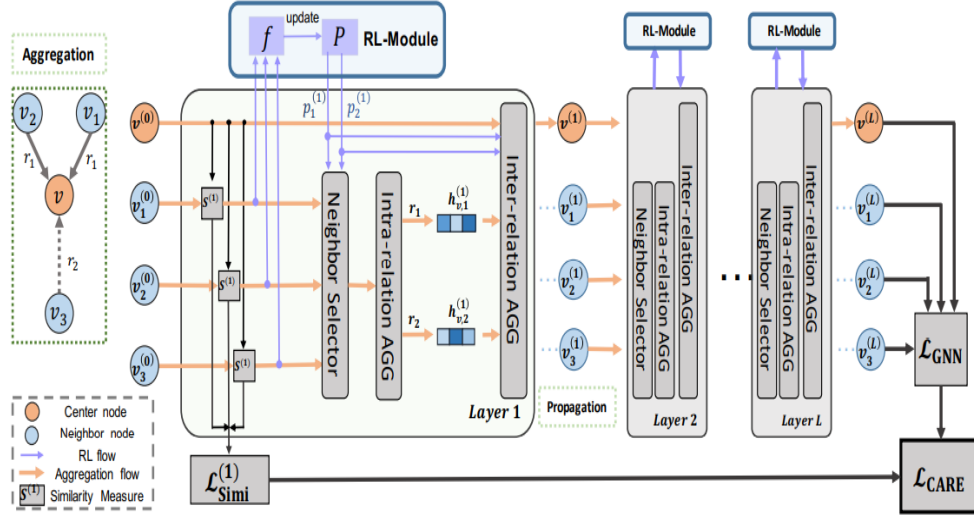


Figure 2.2: CARE-GNN aggregation process (Dou u. a., 2020a)

### Similarity measure

The CARE-GNN model integrates a similarity measurement that takes labels into account to identify informative neighbors. In each layer, a single-layer Multi-layer Perceptron (MLP) is utilized as the node label predictor, and the l1 distance between the predicted labels of two nodes serves as the similarity metric. The MLP calculates similarity measurements and applies a non-linear activation function represented by  $\sigma(\tanh)$ . For a central node  $v$  in relation  $r$  within the  $l$ -th layer, and an edge  $(v, v') \in \varepsilon_r^{(l-1)}$ , the l1 distance between  $v$  and  $v'$  is computed based on their node embeddings:

$$D^{(l)}(v, v') = \|\sigma(MLP^{(l)}(h_v^{(l-1)}) - \sigma(MLP^{(l)}(h_{v'}^{(l-1)}))\|_1 \quad (2.6)$$

the similarity between the two nodes is then determined as:

$$S^{(l)}(v, v') = 1 - D^{(l)}(v, v') \quad (2.7)$$

The output of the MLP is a scalar value, which undergoes a non-linear activation function  $\sigma(\tanh)$ . The model employs cross-entropy loss of the MLP at layer  $l$  to train the similarity measurement using supervised labels:

$$L^{(l)}Simi = \sum_{v \in V} -\log(y_v \cdot \sigma(MLP^{(l)}(h_v^{(l)}))) \quad (2.8)$$

Throughout the learning process, the parameters of the similarity measurement are directly updated using the aforementioned loss function. This approach ensures that similar neighbors can be efficiently selected in the initial batches, leading to smoother GNN training.

### Neighbor selector

During the process, the CARE-GNN model selects matching neighbors for each relationship using top-p sampling, where the adaptive filtering threshold  $p_r^{(l)}$  is learned through reinforcement learning during GNN training. For a node  $v$  in the current batch, with

respect to relation  $r$ , the neighbors are sorted in descending order based on  $S^{(l)}(v, v')$ , and the top  $p_r^{(l)}$  neighbors are chosen for the aggregation process. As mentioned earlier, reinforcement learning algorithms consist of an action space and a reward function. Initially, with a starting value of  $p_r^{(l)}$ , the neighborhood selector has two actions: increment or decrement  $p_r^{(l)}$ . The RL agent's reward depends on the mean difference in distances between two consecutive epochs. The average neighborhood distance for relation  $r$  at the  $l$ -th layer and epoch  $e$  is computed as:

$$G(D_r^{(l)})^{(e)} = \frac{\sum_{v \in V_{train}} D^{(l)}(v, v')^{(e)}}{|V_{train}|} \quad (2.9)$$

the reward function is defined as follows:

$$f(p_r^{(l)}, a_r^{(l)})^{(e)} = \begin{cases} +1, & G(D_r^{(l)})^{(e-1)} - G(D_r^{(l)})^{(e)} \geq 0, \\ -1, & G(D_r^{(l)})^{(e-1)} - G(D_r^{(l)})^{(e)} < 0, \end{cases} \quad (2.10)$$

the reward is positive when the average distance of the newly chosen neighbors in the current epoch is less than that of the previous epoch; otherwise, it is negative. To update the action greedily, CARE-GNN employs instant reward since estimating the cumulative reward is challenging. Positive reward leads to an increase in  $p_r^{(l)}$ , while negative reward results in a decrease. Once the RL module completes, the filtering threshold is fixed as the optimal value until the GNN converges. The convergence is determined based on a terminal condition where the RL has converged in the last ten epochs and indicates the discovery of an optimal threshold  $p_r^{(l)}$ :

$$\left| \sum_{e=10}^e f(p_r^{(l)}, a_r^{(l)})^{(e)} \right| \leq 2, \quad e \geq 10 \quad (2.11)$$

### Neighbor aggregator

After filtering the neighbors for each connection  $r$ , the subsequent step involves aggregating neighbor information across different relations. The model employs two aggregation functions: Intra-relation neighbor aggregation:

$$h_{v,r}^{(l)} = \text{ReLU} \left( \text{AGGr}^{(l)} \left( hv^{(l-1)} : (v, v') \in \varepsilon_r^{(l)} \right) \right) \quad (2.12)$$

where  $\text{AGGr}^{(l)}$  denotes a mean aggregator. Inter-relation aggregation:

$$h_v^{(l)} = \text{ReLU} \left( \text{AGGall}^{(l)} \left( h_v^{(l-1)} \oplus p_r^{(l)} \cdot hv, r^{(l)} \right) \right) \quad (2.13)$$

here,  $\oplus$  represents the embedding summation operation, and  $\text{AGGall}^{(l)}$  can be any type of aggregator. The final embedding is the output of the GNN at the last layer, denoted as  $z_v = h_v^{(L)}$ . The GNN parameters and the CARE-GNN model parameters are optimized using this output. The GNN loss function is defined as:

$$L_{\text{GNN}} = \sum_{v \in V} -\log(y_v \cdot \sigma(\text{MLP}(z_v))) \quad (2.14)$$

where  $y_v$  represents the ground-truth label of node  $v$ , and  $\sigma$  is the sigmoid activation function. The CARE-GNN loss function combines the GNN loss ( $L_{\text{GNN}}$ ), the similarity loss ( $L_{\text{Simi}}$ ), and a regularization term:

$$L_{\text{CARE}} = L_{\text{GNN}} + \lambda_1 L_{\text{Simi}} + \lambda_2 \|\Theta\|_2 \quad (2.15)$$

here,  $\lambda_1$  and  $\lambda_2$  are weight parameters, and  $\|\Theta\|_2$  denotes the L2-norm of the model parameters. The objective is to minimize this combined loss during training.

## 2.4 Spectral Anomaly Detection models

In this section, we concentrate on spectral anomaly detection, most of these methods are generally based on GCNs and detect anomalies using graph filtering.

### 2.4.1 Dominant framework

The Dominant (Ding u. a., 2019), is a framework that utilizes a convolutional graph convolutional network GCN-based autoencoder for anomaly detection in distributed networks. It addresses challenges such as data nonlinearity and graph sparsity and computes outlier rating scores using GCN-based autoencoders. By evaluating the reconstruction loss of node decoder functions from both a structural and attribute perspective, this model identifies anomalies. Node rankings are determined based on their deviation from the majority of reference nodes, assigning higher ranks to nodes that differ significantly. The core element of Dominant is a deep autoencoder, an unsupervised deep neural network that employs stacked encoding and decoding functions to learn a latent representation of the data. The encoder (denoted as  $Enc()$ ) maps the input data to a potentially lower-dimensional feature space, while the decoder (denoted as  $Dec()$ ) attempts to reconstruct the original data using the learned representations. During the learning process, the model minimizes a cost function. The specific details of the cost function are not provided in the given text.

$$\min E[\text{dist}(X, \text{Dec}(\text{Enc}(X)))] \quad (2.16)$$

where  $\text{dist}(\cdot, \cdot)$  represents a predefined distance metric. The deep autoencoder in this model comprises three components illustrated in Figure 2.3.

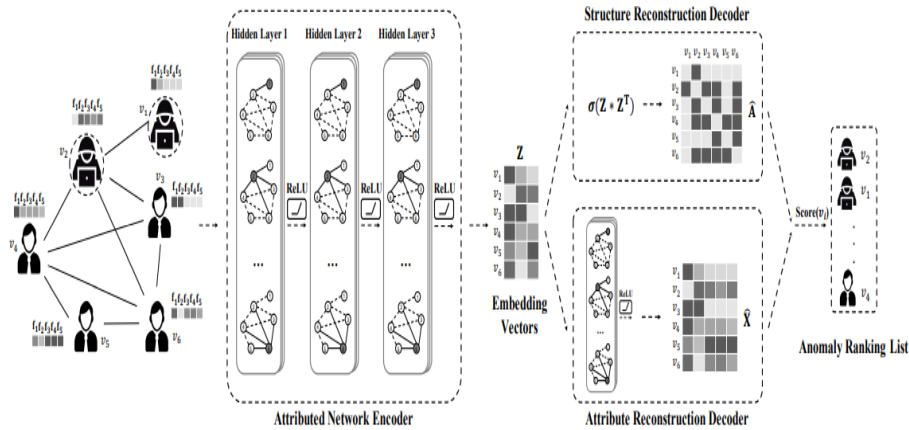


Figure 2.3: Dominant framework (Ding u. a., 2019)

#### Attributed encoder

Dominant introduces a novel type of attributed network encoder inspired by graph convolutional networks (GCN) to tackle three key challenges: network sparsity, data nonlinearity, and complex modality interactions. GCN addresses graph sparsity by considering higher-order node proximity in learning embedding representations. Furthermore, it captures the nonlinearity of the data and the intricate interplay between the two information

modalities within the attribute graph through multiple layers of nonlinear transformations. Consequently, this model employs three GCN convolutional layers to construct the attributed network encoder. Given the attribute matrix  $X$  as input, the model effectively captures the third-hop neighborhood of each node. Each layer of the graph convolutional network is defined by the function  $f(H(l), A|W(l))$  as follows:

$$H^{(1)} = f_{\text{Relu}}(X, A|W^{(0)}) \quad (2.17)$$

$$H^{(2)} = f_{\text{Relu}}(H^{(1)}, A|W^{(1)}) \quad (2.18)$$

$$Z = H^{(3)} = f_{\text{Relu}}(H^{(2)}, A|W^{(2)}) \quad (2.19)$$

the input-to-hidden layer is denoted by  $W^{(0)} \in R^{n \times h_1}$ , where  $h_1$  represents the number of feature maps. Similarly,  $W^{(1)} \in R^{h_1 \times h_2}$  and  $W^{(2)} \in R^{h_2 \times h_3}$  are the two hidden weight matrices. The activation function  $f_{\text{Relu}}$  introduces non-linearity to the model.

By applying three convolutional layers, the input-attributed network is transformed into  $h_3$ -dimensional latent representations  $Z$ . These latent representations capture the complex non-linear relationships present in both the topological network structure and the nodal attributes.

### Reconstruction decoder

In this stage, Dominant leverages the acquired latent representation  $Z$  to reconstruct the original network structure. Let  $\hat{A}$  represent the computed adjacency matrix and the structural reconstruction error  $R_S = A - \hat{A}$  can assess structural anomalies in the network. Thus, if the connectivity patterns cannot be accurately reconstructed, it indicates a mismatch between the structural data and the patterns exhibited by the most typical nodes. A higher norm of  $R_S$  suggests that the  $i$  nodes in the associated network are potentially anomalous regarding network structure. To predict the presence of links between node pairs, the decoder employs the following formula:

$$p(\hat{A}_{i,j} = 1|z_i, z_j) = \text{sigmoid}(z_i, z_j^T) \quad (2.20)$$

the link prediction layer is trained using the output of the attributed network encoder  $Z$ :

$$\hat{A} = \text{sigmoid}(ZZ^T) \quad (2.21)$$

### Attribute decoder

The decoder module is tasked with approximating node attribute data using the encoded latent representation  $Z$ . More specifically, the attribute reconstruction decoder employs separate graph convolutional layers to predict the original node attributes:

$$\hat{X} = f_{\text{Relu}}(Z, A|W^{(3)}) \quad (2.22)$$

By computing the reconstruction error as  $R_A = X - \hat{X}$ , anomalies in attribute networks can be identified. To collectively learn the reconstruction errors, the objective function of the deep graph convolutional autoencoder is defined as:

$$L = (1 - \alpha)R_S + \alpha R_A \quad (2.23)$$

The node anomalies are evaluated using the reconstruction error obtained from the deep graph convolutional autoencoder. It is important to mention that the weight matrix of the autoencoder is trained using gradient descent on the loss function. Once a certain number of iterations is completed, the anomaly score  $v_i$  is computed for each node according to the following equation:

$$\text{score}(v_i) = \|a - \hat{a}_i\|_2 + \alpha \|x_i - \hat{x}_i\|_2 \quad (2.24)$$

Based on this score, the nodes are ranked, with the highest score indicating anomalous behavior.

### 2.4.2 Graph Neural Networks with adaptive receptive paths

The GeniePath model (Liu u. a., 2019) introduced a novel approach by incorporating an adaptive path layer. Instead of simply summing the embeddings of all 2-hop neighbors, GeniePath focuses on learning the relevant receptive paths that contribute the most to the node representation. By leveraging permutation invariance and the associative properties of aggregation, GeniePath filters out noise nodes and selectively aggregates chosen nodes without relying on their specific order. This approach utilizes breadth exploration to identify influential neighborhoods and depth exploration to determine the number of useful neighbor hops. By filtering nodes in subgraphs, GeniePath learns meaningful receptive paths, which is particularly valuable in noisy graphs where different nodes have distinct roles.

The primary goal of GeniePath is to discover receptive paths for transmitting signals along learned paths, rather than relying on predefined paths. This involves expanding both breadth and depth for each node to select an appropriate subgraph. To capture the breadth, an adaptive breadth function  $\phi(A, H(t); \Theta)$  is computed. Here,  $\Theta$  represents the parameters, and the function aggregates the nodes. Additionally, an adaptive depth function  $\varphi$ , parameterized by  $\Phi$ , is employed to explore the depth of receptive paths. The overall architecture of the adaptive path layer is depicted in Figure 2.4.

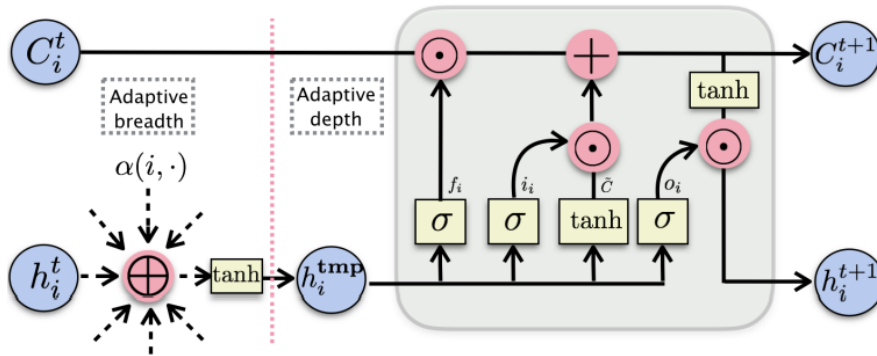


Figure 2.4: GeniePath architecture (Liu u. a., 2019)



### Adaptive breadth function

In order to explore the breadth of receptive paths, GiniePath employs an adaptive breadth function  $\phi()$  to aggregate signals, dynamically assigning varying importance to different neighbors at a single-step leap. The equation corresponding to the adaptive breadth function  $\phi()$  is as follows:

$$h_i^{(\text{tmp})} = \tanh \left( W^{(t)T} \sum_{u \in N(i) \cup v} \alpha(h_i^{(t)}, h_u^{(t)}) \cdot h_u^{(t)} \right) \quad (2.25)$$

where  $h_i^{(0)} = W_x^T X_i$  and  $X_i \in R^P$  denotes the feature vector of node  $i$ . The function assigns importance to the embedding  $h_u^{(t)}$  of any one-hop neighbor through the parameterized generalized linear attention operator  $\alpha(\cdot, \cdot)$ :

$$\alpha(x, y) = \text{softmax}_y(v^T \tanh(W_s^T x + W_d^T y)) \quad (2.26)$$

the symbol  $\oplus$  represents the summation of the attention operator  $\alpha(x, y)$ :

$$\sum_{u \in N(i) \cup v} \alpha(h_i^{(t)}, h_u^{(t)}) \quad (2.27)$$

### Adaptive depth function

To explore the depth of the receptive path, the model utilizes an adaptive depth function, which can pull and filter the aggregated nodes at the  $t$ -th distance from the target node  $i$  by capturing the dependence between aggregated signals at different depths. The following equations correspond to the adaptive depth function  $\varphi$ :

$$i_i = \sigma \left( W_i^{(t)T} h_i^{(\text{tmp})} \right), \quad f_i = \sigma \left( W_f^{(t)T} h_i^{(\text{tmp})} \right) \quad (2.28)$$

$$o_i = \sigma \left( W_o^{(t)T} h_i^{(\text{tmp})} \right), \quad \tilde{C} = \tanh \left( W_c^{(t)T} h_i^{(\text{tmp})} \right) \quad (2.29)$$

$$C_i^{(t+1)} = f_i \odot C_i^{(t)} + i_i \odot \tilde{C} \quad (2.30)$$

and finally

$$h_i^{(t+1)} = o_i \odot \tanh \left( C_i^{(t+1)} \right) \quad (2.31)$$

the memory  $C_i$  for each node  $i$  is initialized as  $C_i^{(0)} = \vec{0} \in R^K$  (where  $R^K$  represents the feature space vector), and it is updated during the exploration of receptive paths, considering the depth and breadth of the neighborhood. The unit  $i_i$  with a sigmoid output is responsible for selecting relevant nodes from  $\tilde{C}$  and incorporating them into the memory as  $i_i \odot \tilde{C}$ . The unit  $f_i$  filters out ineffective nodes from the previous memory based on the newly observed neighborhood. This allows the model to refine the memory for each node  $i$  as  $C_i^{(t+1)}$ , thereby expanding the depth of the neighborhood. Finally, the updated memory  $C_i(t+1)$  and the output unit  $o_i$  are utilised to compute the embedding of node  $i$  at the  $(t+1)$ -th layer, denoted as  $h_i(t+1)$ . These parameterized functions can be optimized using a custom-defined loss function that can include supervised or unsupervised learning. The trainable parameters include the weight matrix  $W_x \in R^{P \times K}$ ,  $\Theta = W, W_s, W_d \in R^{K \times K}$ ,  $v \in R^K$ , and  $\Phi = W_i, W_f, W_o, W_c \in R^{K \times K}$ . These parameters are compact and have the same scale as other graph neural networks.

### Permutation invariant

The notion of permutation invariance in graph learning tasks suggests that the ordering of neighbors during the aggregation phase does not impact the learning process. A function applied to a graph is considered permutation invariant with respect to the neighbors of a node  $i$  if and only if it can be expressed as  $\rho\left(\sum_{u \in N(i)} \phi(h_j)\right)$ , where  $\phi$  and  $\rho$  represent appropriate matching functions. If a function  $f$  is permutation invariant for the neighborhood  $N()$ , then  $g \circ f$  is also permutation invariant for  $N()$  as long as  $g$  does not rely on the ordering of  $N()$ . The ability to stack cascading functions is facilitated by this characteristic. Specifically, in the case of the adaptive breadth function  $\varphi()$ , it acts on a linear transformation of the neighbors, satisfying the condition of permutation invariance. The function  $\phi()$ , applied to the layers at each depth, is unaffected by the order of any 1-step neighborhood. As a result, the composition of  $\phi \circ \varphi$  maintains permutation invariance regardless of the ordering of variables.

### 2.4.3 Adaptive multi-frequency GNN

The Adaptive multi-frequency graph neural network (Chai u. a., 2022) semi-supervised model **AMNet** suggests that the low-pass property of GNNs is fundamentally inconsistent with the nature of graphs that have anomalies. GNNs are capable of smoothing the difference between the embeddings of normal nodes and abnormal ones by aggregating them. Thus, the representation of anomalies learned by GNN may be indistinguishable from normal nodes and lead to suboptimal performance for graph anomaly detection problems.

To solve this issue AMNet designed two low and high-frequency kernels for filtering anomalous signals and learn the importance of two sets of kernels adaptively to aggregate nodes and distinguish abnormal ones. Specifically, adaptive filters capture low and high-frequency graph signals simultaneously using a learned high-pass filter  $g_H$  and a learned low-pass filter  $g_L$  to filter out nodes features. Afterward, each node is aggregated with its neighbors information to capture a high-frequency signal  $Z_H^i$  and low-frequency signals  $Z_L^i$ . Figure 2.5 represents this process.

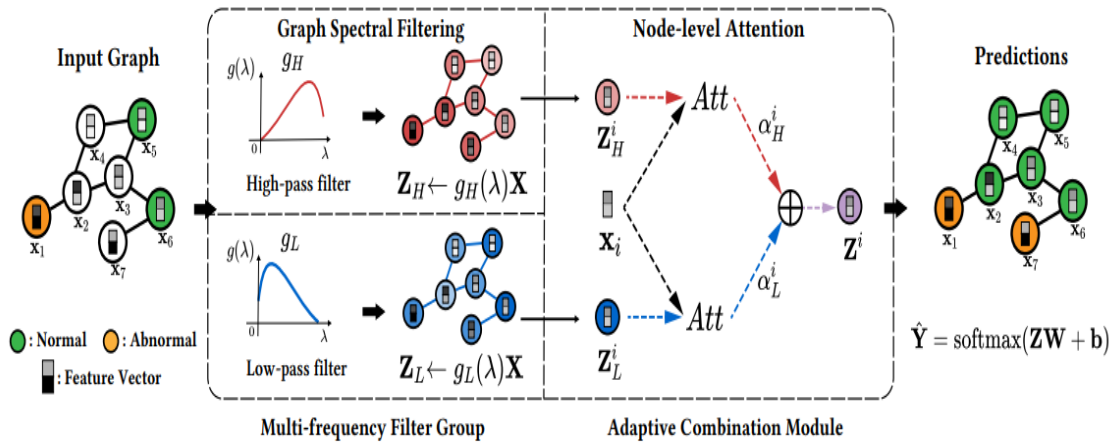


Figure 2.5: AMNet framework (Chai u. a., 2022)

After filtering, the model applies a node-level attention mechanism to adaptively combine

$Z_H^i$  and  $Z_L^i$  to obtain the final representation  $Z^i$ , which is passed to a network interface to generate a prediction for the anomaly detection task. In the following explanation, we will base our calculations on the existence of these two filters.

### Multi frequency filter group

The multi-frequency filters are a group of  $K$  learnable filters denoted as  $\{g_i\}_{i=1, \dots, K}$  ( $g_H$  and  $g_L$  in the previous figure), each of them captures graph signals with distinct frequencies. Then each node in the graph will receive  $K$  signals. The signal frequency is controlled by the learnable parameters of the graph filters. The graph signal  $Z_k$  filtered by the  $k$ -th filter is defined as:

$$Z_k = U_{g_k}(A)U^T X = U \text{diag}[g_k(\lambda_1), \dots, g_k(\lambda_n)]U^T X \quad (2.32)$$

where  $\text{diag}[\lambda_1, \dots, \lambda_n]$  is the diagonal matrix of eigenvalues,  $X$  is the signal filtered, the matrix  $U$  represents the eigenvector matrix of the graph Laplacian matrix  $L$  and the graph filters  $g_k$  are based on the Bernstein polynomial parametrisation (Chai u. a., 2022). Now each graph node  $v \in G(V, E)$  obtains  $K$  signals  $\{Z_1, \dots, Z_k\}$ , and the learnable parameters of graph filters control the frequencies of these signals.

### Adaptive combination module

To model the difference between signal preferences, a node-level attention mechanism adaptively merges the signals, and the merged embeds are obtained for the classification. Each node focuses on distinct frequency bands via an attention mechanism  $\text{att}(Z_1, \dots, Z_k)$  that learns the corresponding signal importance  $\alpha$  as follows:

$$(\alpha_1, \dots, \alpha_k) = \text{att}(Z_1, \dots, Z_k) \quad (2.33)$$

$(\alpha_1, \dots, \alpha_k)$  are the attention values of  $n$  nodes with  $(Z_1, \dots, Z_k)$  respectively. Larger  $\alpha_k^i$  indicates that the node  $v_i$  favors the  $k$ -th filter's frequency band. The final node embedding  $Z$  is obtained by combining the filtered signals:

$$Z = \sum_k \alpha_k Z_k \quad (2.34)$$

meaning that for each node  $v_i$  the node embedding  $z^i$  will equal:

$$z^i = \left( \sum_k \alpha_k^i g_k \right) \star x_i \quad (2.35)$$

This means that AMNet applies a personalized graph filter for **each node  $v_i$  specifically** and the AMNet gives adaptivity for each node and learns its graph filter separately which can be computationally expensive in the case of large datasets.

### Loss function

To enhance the distinction between anomalous and normal nodes, anomalous nodes must leverage higher-frequency information. A constraint is imposed on attention training to capture the notion that abnormal and normal nodes should focus on different filters. The margin-based constraint on attention, denoted as  $L_a$ , can be defined as follows:

$$L_a = \sum_i \max \left( 0, r_i(\alpha_L^i - \alpha_H^i) + \zeta \right) \quad (2.36)$$

here,  $\zeta$  is a slack variable that controls the margin between attention values. The value of  $r_i$  is 1 when  $Y_i = 1$  (indicating an anomalous node) and -1 otherwise. The overall loss function combines  $L_c$ , which represents the cross-entropy loss of the classification task, and  $L_a$ , the constraint on attention:

$$L = L_c + \beta L_a \quad (2.37)$$

The parameter  $\beta$  controls the weight assigned to  $L_a$  in the overall loss function. The output embedding  $Z$  is then used for semi-supervised classification. The probability  $\hat{Y} \in R^{N \times 2}$ , indicating the likelihood of nodes belonging to either the anomalous or normal class, is estimated using a linear transformation followed by a softmax function.

## 2.5 State of the art models results

To acquire an idea of the performance of the anomaly detection methods mentioned in previous sections, we have illustrated two comparison tables based on the scores of the model validation. The area under the receiver operating characteristic curve (AUC-ROC) evaluation metric measures the models capability to accurately classify nodes, while the area under the precision-recall curve (AUC-PR) evaluates their ability to identify anomalies while minimizing false positives.

To provide a comprehensive evaluation of the anomaly detection methods, we conducted experiments on two diverse datasets: Yelp and Elliptic. The Yelp dataset (Rayana und Akoglu, 2015) comprises user reviews and ratings for various businesses. Anomalies in this dataset can represent unusual user behavior or fraudulent activities. The Elliptic dataset (Weber u. a., 2019) focuses on financial transactions involving cryptocurrencies. It contains a large number of transactions, including both legitimate and anomalous ones, such as money laundering or fraudulent activities. This dataset presents unique challenges due to the complex and dynamic nature of financial transactions. Evaluating the models on the Elliptic dataset allows us to gauge their effectiveness in detecting anomalous financial behavior.

Table 2.1 presents the AUC-ROC performance of different models evaluated on the Yelp (Rayana und Akoglu, 2015) and Elliptic (Weber u. a., 2019) datasets. This table provides insights into the models effectiveness in discriminating between anomalies and normal instances in the two datasets. Table 2.2 focuses on the AUC-PR performance on the same Yelp and Elliptic datasets. It measures how well the models perform in terms of precision and recall, providing an understanding of their ability to accurately identify anomalies. These comparison tables serve as valuable references for assessing the relative performance of the anomaly detection methods and selecting the most suitable approach for specific applications and datasets.

<b>Dataset</b>	<b>Yelp</b>	<b>Elliptic</b>
DOMINANT	49.32	16.21
GeniePath	75.89	83.14
GraphConsis	70.40	86.14
CARE-GNN	78.41	85.84
AMNet	85.85	88.52

Table 2.1: AUC-ROC scores (Chai u. a., 2022)

<b>Dataset</b>	<b>Yelp</b>	<b>Elliptic</b>
DOMINANT	15.58	5.48
GeniePath	35.86	44.37
GraphConsis	27.02	62.04
CARE-GNN	38.90	49.81
AMNet	57.77	74.62

Table 2.2: AUC-PR scores (Chai u. a., 2022)

It is noticeable that AMNet reached the highest performance on both datasets Yelp and Elliptic. CARE-GNN also exhibited competitive results. GeniePath and GraphConsis were closely behind, showing moderate performance. DOMINANT produced the lowest scores, meaning it's relatively weaker. Overall, these tables indicate that AMNet and CARE-GNN achieved the highest performance on both evaluation metrics across the Yelp and Elliptic datasets.

## 2.6 Conclusion

Throughout this chapter, we have extensively explored numerous models developed for anomaly detection employing GNNs. Our comprehensive analysis covered both spectral and spatial anomaly detection methods, shedding light on their underlying principles and efficacy in identifying anomalies within graph data. Furthermore, our research has led us to discover that among these models, one notable standout is the AMNet model, which has demonstrated superior performance. This further underlined the immense potential of adaptive filters within the context of anomaly detection. Thus, we focussed on a deeper exploration of the adaptive integration of diverse signals into the anomaly detection problem. Looking ahead to the next chapter, we discuss our solution, providing detailed insights into its algorithmic framework and architectural design.

## Chapter 3

# Adaptive frequency GNN for Anomaly Detection

### 3.1 Introduction

To address the anomaly detection problem, based on the previous chapters findings, we recognize the power of adaptive frequency filtering in detecting abnormal nodes and include it in our work. Our anomaly detection model employs AdaGNN (Dong u. a., 2021), a spectral GNN embedding approach that can learn discriminative node embeddings on graphs via adaptive frequency filtering.

While AdaGNN has demonstrated exceptional performance in node classification tasks, its relevance to anomaly detection is unknown. To bridge this gap, we present Anomaly-AdaGNN, a graph convolutional encoder that uses tailored AdaGNN for anomaly detection. Anomaly-AdaGNN unlocks the AdaGNN model benefits and unveils its potential to yield outstanding results in the outlier detection field. Our goal is to develop an efficient Anomaly-AdaGNN model and compare it to AMNet (Chai u. a., 2022), one of the most recent and accurate anomaly detection methods. In the following sections, we will provide a detailed description of the AdaGNN model architectures and delve into the personalized model for anomaly detection, Anomaly-AdaGNN.

### 3.2 Adaptive frequency response filter GNN

One of the most often used spectral GNNs is the GCN model mentioned in section 1.4.2. It updates node representations depending on their neighbors via a graph convolution operation, similar to how a standard convolutional neural network updates pixel representations in a picture using surrounding pixels. GCN has various advantages however it can be seen as a low-pass filter that retains only low-frequency information from the input graph (Dong u. a., 2021) (Li u. a., 2018a) (Chai u. a., 2022). It can only do non-learnable graph filtering, which means it cannot collect relevant information that is not in the low-frequency component adaptively. High-frequency information, on the other hand, is ignored. Novel adaptive frequency filtering models, such as AdaGNN (Dong u. a., 2021), have been developed to alleviate this problem.

In this section, we will explore the architecture of the AdaGNN model, which effectively

addresses the common issue of over-smoothing encountered in many GNN models. We will delve into how the adaptive frequency components of AdaGNN capture both low and high-frequency data, resulting in improved performance and more accurate representations.

### 3.2.1 AdaGNN architecture

AdaGNN, also known as Adaptive Frequency Response Filter GNN (Dong u. a., 2021), is a powerful semi-supervised graph representation learning model. It leverages adaptive frequency response filters to conduct graph convolution operations, enabling the model to effectively learn from graph-structured data. The architecture of AdaGNN is specifically designed to dynamically adjust the spectral response of the filters based on the characteristics of the input graph, accommodating the varying graph structures and node features.

During the training process, the adaptive frequency response filters, parameterized by learnable parameters, learn to adapt to the frequency distribution of the input graph. This allows them to capture both low and high-frequency data, leading to a more comprehensive understanding of the graph and its underlying patterns. To execute the graph convolution operation, AdaGNN follows this process:

1. Input embedding: the first embedding layer computes a new feature representation for each node in the graph, as follows:

$$H^{(1)} = ReLU((H^{(0)} - \tilde{L}H^{(0)}\Phi_1)\Theta) \quad (3.1)$$

where  $\Phi_1$  is  $\Phi = diag(\phi_1, \dots, \phi_F)$ ,  $\phi_j$  represents the learnable parameter for the  $j$ -th feature channel and  $\Theta \in R^{F \times L}$  the weight matrix at the first layer and  $\tilde{L}$  is the normalised Laplacian.

2. Adaptive aggregation: for the layers  $2 \leq k \leq K$ , the model learns an adaptive aggregation function to capture more nuanced relationships between nodes, and to adapt to different graph structures. Here each feature channel is assigned to a separate filter that adaptively learns its frequency response function. The aggregation function achieves the appropriate levels of smoothness because each feature frequency is learned with an independent filter:

$$H^{(k)} = H^{(k-1)} - \tilde{L}H^{(k-1)}\Phi_k \quad (3.2)$$

The learnable parameters for the  $k$ -th layer are denoted as  $\Phi_k$ . After  $K$  layers, the output representation is denoted as  $H^{(K)}$ , and  $\tilde{L}$  graph laplacian.

3. Output projection: after obtaining the hidden states of the nodes in the graph, the prediction layer maps these hidden states to a set of target labels using a softmax function:

$$\hat{Y} = softmax(H^{(K)}W) \quad (3.3)$$

here,  $W \in R^{L \times C}$  is a weight matrix that transforms the node representations to the label space.

4. Backpropagation: for backpropagation, a loss function is used to train the model. The loss function is formulated as follows:

$$L_{AdaGNN} = - \sum_{i \in Y_L} \sum_{j=1}^C Y_{i,j} \ln \hat{Y}_{i,j} + \alpha \sum_{k=1}^K \|\Phi_k\|_1 + \beta \left( \sum_{k=1}^K \|\Phi_k\|_F^2 + \|\Theta\|_F^2 + \|W\|_F^2 \right) \quad (3.4)$$

in this equation,  $Y_L$  denotes the set of indices for labeled nodes, and  $Y \in R^{|Y_L| \times C}$  represents the ground truth labels. To maintain as much information as possible after each layer, the model incorporates  $\ell_1$ -norm regularization on  $\Phi_k$ , where  $k$  represents the layer index. Additionally, to prevent overfitting,  $\ell_2$ -norm regularization is applied to all trainable parameters, including  $\Phi_k$ ,  $\Theta$ , and  $W$ . The intensity of the regularization terms is controlled by the hyperparameters  $\alpha$  and  $\beta$ .

### 3.2.2 Over-smoothing problem

The over-smoothing problem arises when aggregating information from neighboring nodes in a graph, as explained in section 1.4.2. During the message-passing phase, each node collects information from its neighbors and updates its representation iteratively over multiple layers. However, as the number of iterations increases, information from distant nodes can dominate and spread uniformly across the graph. Oversmoothing in GNNs can negatively impact the model’s performance by making node representations too similar. This similarity makes it difficult to distinguish between nodes or capture subtle variations in the graph structure.

To address this problem, AdaGNN introduces adaptive filtering. Specifically, consider the convolution operation between a graph signal  $x$  and the filter  $g_k(\cdot)$  at layer  $k$ . Over-smoothing becomes a concern when the sum of the frequency response of the filter satisfies the condition (Dong u. a., 2021) (Li u. a., 2018b) (Nt und Maehara, 2019):

$$\lim_{K \rightarrow \infty} \sum_{k=1}^K g_k(\tilde{\lambda}_i) = 0 (\forall \tilde{\lambda}_i \neq \tilde{\lambda}_1) \quad (3.5)$$

When this condition is met, the repeated convolution operations result in over-smoothing, causing the feature values of different nodes to converge and become indistinguishable. To mitigate over-smoothing, AdaGNN introduces a set of learnable parameters denoted as  $\Phi$ . These parameters modify the smoothness of each feature channel during the information aggregation process in the spatial domain. By adjusting the frequency response values, AdaGNN prevents the overall response from reaching zero, thus avoiding excessive similarity among node representations.

The adaptive filtering capability of AdaGNN allows it to dynamically control the frequency response of the graph filters. This ensures that the model can capture and retain important information from the graph structure while maintaining distinctiveness among node representations. By adjusting the parameters  $\Phi$ , AdaGNN can fine-tune the frequency response values to strike a balance between smoothing and preserving local variations.

In summary, AdaGNN’s adaptive filtering technique enables it to overcome over-smoothing by selectively modifying the frequency response of the graph filters. This prevents the loss of discriminative information and allows the model to capture both global and local characteristics of the graph without sacrificing the number of convolutional layers.



### 3.2.3 Explanatory example

AdaGNN offers several notable advantages in graph representation learning. Firstly, it introduces the parameter  $\Phi_{j,k}$ , which provides fine-grained control over the relative importance of high-frequency and low-frequency components at each layer  $k$ . This allows for precise adjustments in the significance of different frequency components, enhancing the model's ability to capture diverse patterns and variations. Secondly, by incorporating trainable parameters  $\{\phi_{j,1}, \dots, \phi_{j,K}\}$ , AdaGNN effectively captures the importance of distinct frequency components. As a result, when multiple layers are stacked, AdaGNN can create complex frequency response functions, enabling more sophisticated learning of the graph structure.

One of the most significant advantages of AdaGNN lies in its ability to address the issue of over-smoothing. The frequency response function associated with each feature channel grants the model increased flexibility in achieving desired levels of feature smoothness. By selectively controlling the smoothness of different feature signals, AdaGNN avoids the common problem of over-smoothing while maintaining the essential details and preserving the richness of the underlying graph structure. To better illustrate this capability, Figure 1.3 demonstrates how AdaGNN adeptly adjusts the smoothness of various feature signals, ensuring an optimal balance between preserving important details and avoiding excessive smoothing:

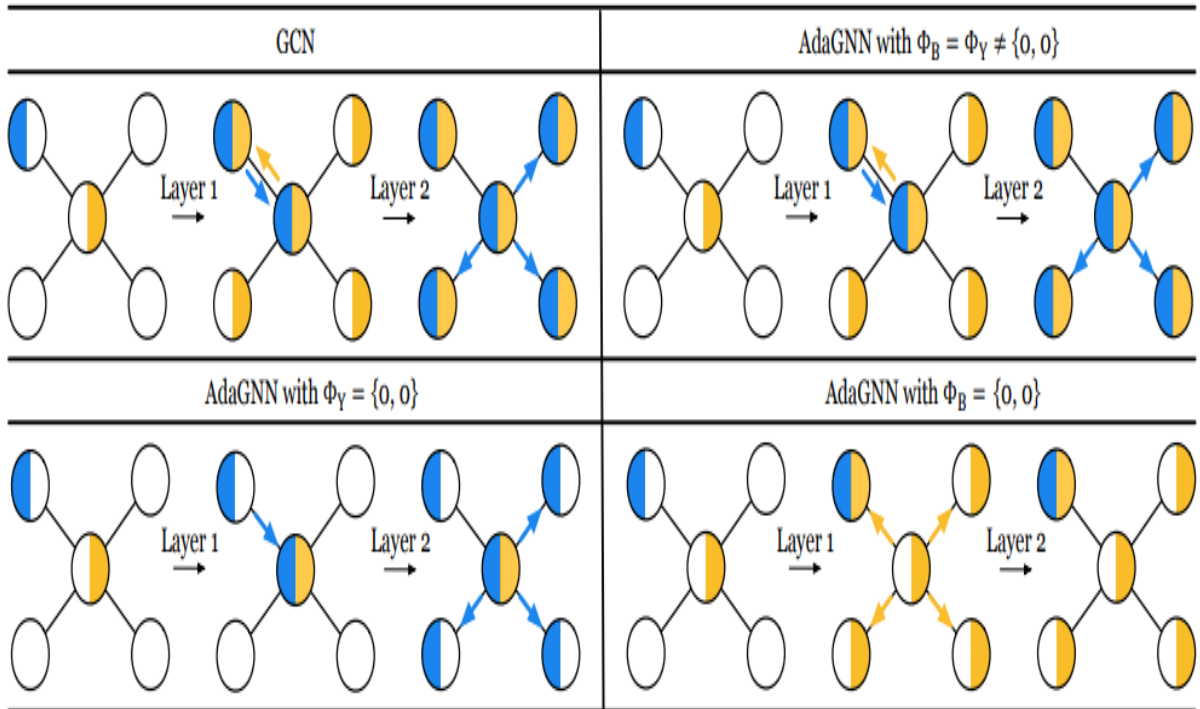


Figure 3.1: Illustrative example of AdaGNN (Dong u. a., 2021)

There are five nodes indicated by circles in the diagram, and each node has two feature values: blue and yellow. The upper left node corresponds to the blue channel, whereas the center node corresponds to the yellow channel. When using the GCN model, the feature propagation mechanism is the same for all nodes. As a result, after two layers of

propagation, the node representations become identical. This means that the GCN model does not differentiate between nodes based on their channel association. On the other hand, the AdaGNN model introduces learnable parameters to control the smoothness of representations for different channels. Specifically, for the blue channel, the learnable parameters across two layers are denoted as  $\Phi_B = \phi_{B,1}, \phi_{B,2}$ , and for the yellow channel, the parameters are denoted as  $\Phi_Y = \phi_{Y,1}, \phi_{Y,2}$ .

By incorporating these learnable parameters, AdaGNN allows for channel-specific control over the smoothness of node representations. This means that the representations of nodes associated with different channels can be adjusted independently, potentially leading to more fine-grained and expressive representations compared to the GCN model. The model suffers from the over-smoothing problem when the parameters are equal. However, AdaGNN adaptively controls the smoothness of each feature by learning the optimal parameter  $\phi$ . This example explains that decoupling the smoothness of each feature channel from each other makes nodes better discriminatory alleviating the over-smoothing problem.

While AdaGNN has shown improved performance and outperforms one of the state-of-the-art GNN models such as GCN in various graph-related tasks like node classification, link prediction, and graph classification, it was not implemented in anomaly detection tasks (Dong u. a., 2021).

### 3.3 Motivation and contribution

Graphs are commonly used to model complex real-world systems, and detecting anomalies in graphs has become a popular research topic. With the advancement of GNNs, many methods have been proposed to solve anomaly detection problems. However, existing models can only achieve non-learnable graph filters, which means that they cannot adaptively capture useful information that is not contained in the low-frequency component (Li u. a., 2018a) (Chai u. a., 2022). This makes neighboring nodes more similar which can make it difficult to detect anomalies as they may be camouflaged by the filtered signal.

The low pass property of GNNs misaligns with the nature of networks containing anomalies (Chai u. a., 2022), potentially smoothing the difference between the representations of normal and anomalous nodes by filtering out high-frequency signals. This can lead to suboptimal performance for graph anomaly detection problems. To be more specific, let's delve into the concept of frequency in the context of graph data, as discussed in section 1.4. In traditional terms, frequency refers to the number of occurrences of a repeating event per unit of time. However, when dealing with graph data, the notion of frequency is generalized to capture how rapidly a signal changes concerning the underlying structure of the graph.

In graph data, the eigenvectors of the graph Laplacian are leveraged as the basis for representing different frequencies. These eigenvectors serve as the fundamental building blocks for capturing the variations and changes within the graph data. By utilizing the eigenvectors of the graph Laplacian, we can decompose signals into different frequency components to enable us to analyze and understand how signals change and propagate throughout the graph structure, providing insights into the dynamic nature of the data.

For example, in Figure 3.2, we can observe how frequency is captured in graph data.

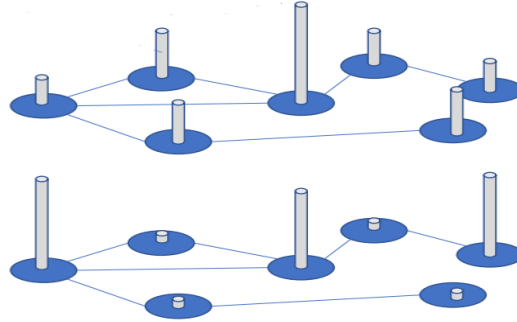


Figure 3.2: Comparison between low and high-frequency components (Dong u. a., 2021)

Assume that the signal is a specific column in the feature vector  $X$ . In the first graph, we notice that the signal changes slowly across graph edges and maintains the same value in most of the nodes meaning that the signal is a low-frequency signal, while in the second graph, the values change significantly and more rapidly across the graph which makes it a high-frequency signal. Anomalous nodes in graphs often exhibit more variance in their features, which indicates that their features are likely to be in the high-frequency components. Graph low-pass filtering refers to decreasing the weight assigned to the eigenvector basis associated with large eigenvalues of the graph Laplacian matrix. By reducing the weight on eigenvectors associated with large eigenvalues, graph low-pass filtering effectively diminishes the contribution of high-frequency components in the graph signal projected on these eigenvalues. Figure 3.3 showcases the result graph signal.

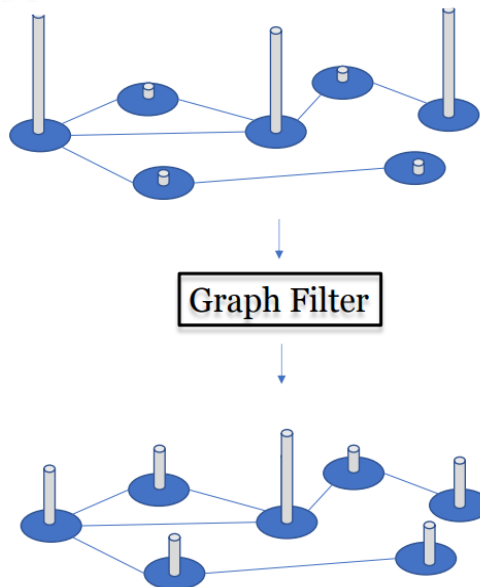


Figure 3.3: High-frequency components after low-pass filtering (Dong u. a., 2021)

Note that the outcome of graph low-pass filtering is a graph representation with a significantly smoother representation of the graph signals. This smoothing effect plays a

vital role in concealing or camouflaging anomalous nodes within the graph. As a result, the overall representation becomes more homogenous, allowing anomalous nodes to blend in with their surroundings and become less distinguishable. This smoothing effect acts as a form of camouflage, making it more challenging to detect. Consequently, the smooth representation obtained through graph low-pass filtering can aid in concealing the presence of these anomalous nodes, enhancing their ability to remain undetected within the graph structure.

Although AMNet is one of the few models that consider these issues it continues to have other limitations such as its computational complexity, which can be a limitation when dealing with large-scale graphs since it applies a personalized graph filter for each node specifically. As the size of the graph increases, the model can become computationally expensive and require significant training, restricting the application of the model to real-world graph datasets with tens of thousands of nodes and edges. Allocating a learnable parameter for each frequency component like the AMNet method requires expensive eigendecomposition (Wu u. a., 2020) making the model prone to overfitting specifically when the training data is limited. Also, the model does not take into consideration the class imbalance issue, in anomaly detection, the number of normal samples in a dataset is significantly larger than the number of anomalies. This class imbalance can affect the models ability to learn and generalize accurately on the existing data, as it may generalize on normal nodes during training.

To address these challenges, we propose the adoption of the AdaGNN model, to leverage its adaptive frequency components to detect anomalies in graphs. **Anomaly-AdaGNN**, short for Anomaly Adaptive frequency GNN, focuses on overcoming the limitations associated with anomaly detection in graph data:

1. Incorporates high pass filtering into the network architecture to preserve the graph signals high-frequency components.
2. Deploys AdaGNN power and accuracy to the anomaly detection problem.
3. Takes under consideration the imbalanced data labels issue in anomaly detection cases.

## Notations

In this model, we are working with an undirected graph represented as  $G = (V, E)$ , where  $V$  denotes the set of nodes  $v_i \in \{v_1, \dots, v_N\}$ , and  $E$  represents the set of edges connecting node pairs. The connectivity of the graph is captured by the adjacency matrix  $A \in R^{N \times N}$ . Each element  $A_{i,j}$  in the matrix is defined as follows: if node  $v_j$  is a one-hop neighbor of node  $v_i$ , then  $A_{i,j} = 1$ , otherwise  $A_{i,j} = 0$ . The adjacency matrix essentially encodes the information about direct node connections. Moreover, we have the feature matrix  $X \in R^{N \times F}$  that describes the node attributes. Each row in the matrix corresponds to a node, and each column represents a feature channel. Therefore,  $x_j \in \{x_1, \dots, x_F\}$  denotes the values of the  $j$ -th feature channel in  $X$ . The number of feature channels is denoted as  $F$ .

### 3.4 AdaGNN for Anomaly Detection

To extend the capabilities of the AdaGNN model, our primary objective is to incorporate it into the anomaly detection field. To achieve this goal, we introduce Anomaly-AdaGNN, a framework designed specifically for anomaly detection in graphs. Anomaly-AdaGNN harnesses the adaptive component of AdaGNN to effectively utilize both low and high-frequency information, eliminating the limitation typically associated with traditional low-frequency filtering approaches. By dynamically adjusting the frequency response filters, Anomaly-AdaGNN can capture and incorporate crucial information across a broad range of frequencies, resulting in a more comprehensive anomaly detection capability.

Unlike the AMNet method, our approach assigns data filters to feature channels rather than individual nodes. This strategy ensures computational efficiency while maintaining the ability to capture important features and patterns in the graph data. Furthermore, Anomaly-AdaGNN addresses the common challenge of imbalanced data labels often encountered in anomaly detection datasets. To enhance understanding, the illustrative Figure 3.4 provides a comprehensive visual representation of Anomaly-AdaGNN’s architecture.

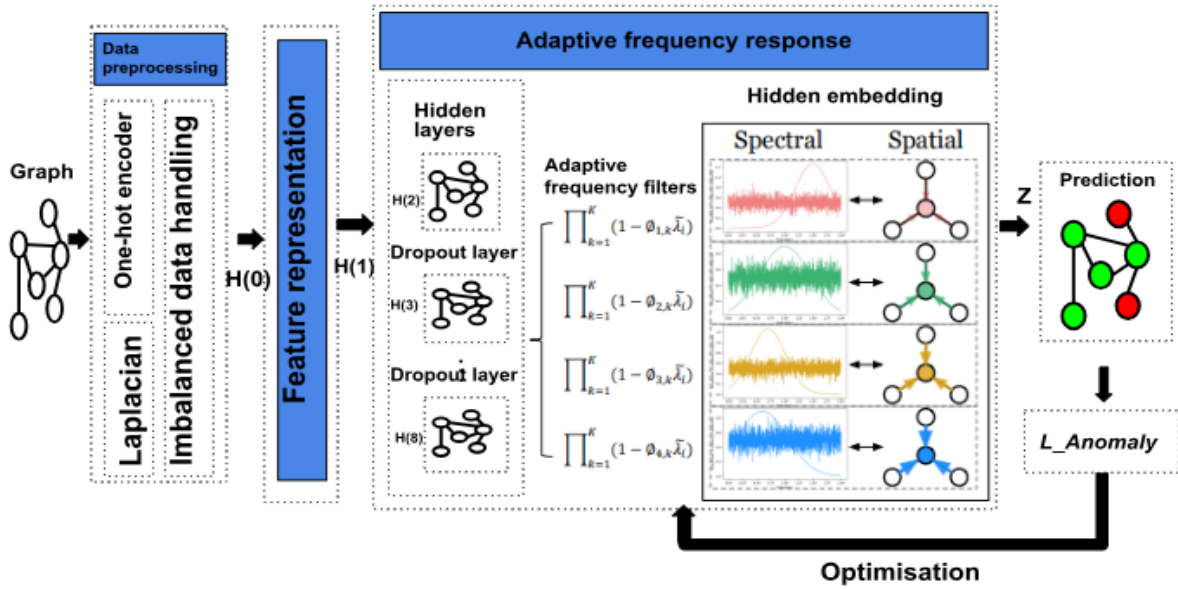


Figure 3.4: Anomaly-AdaGNN architecture

The Anomaly-AdaGNN model follows a well-defined process. It begins with a preprocessing step applied to increase the percentage of anomaly labels and to address the challenge of imbalanced data labels. Afterward, the model computes a refined feature representation for each node by considering both the graph structure and the spectral features. A learnable diagonal matrix is assigned at each layer to adaptively adjust the feature representation for each node in the graph. This matrix is trained during the adaptive filters training process and plays a crucial role in modifying the feature representation based on the node’s neighborhood.

Multiple layers of AdaGNN are stacked to create a sophisticated frequency response function, enhancing the model's ability to capture intricate patterns in the data. Additionally, dropout layers are introduced to enhance the model's generalization and robustness. The output node embeddings are used to perform the prediction task, and an anomaly loss is employed during the model's optimization step to guide the model filters toward highlighting high-frequency signals, aiding in accurate anomaly detection. This comprehensive approach enhances the model's ability to identify and differentiate anomalous nodes from normal ones, contributing to reliable and effective anomaly detection in graph datasets.

### 3.4.1 Data preprocessing

Preprocessing plays a crucial role in preparing the data for the following computations. In this section, we outline the key preprocessing steps that are necessary for the Anomaly-AdaGNN model. While there are other preprocessing techniques that we will discuss in the following chapter, let us now delve into the essential steps.

#### One-hot encoder

To facilitate the calculation of the loss function, we apply one-hot encoding to the categorical labels. This conversion transforms the labels into a binary representation, where each class is represented by a unique binary vector. The one-hot encoded ground truth labels are represented by  $Y \in \{0, 1\}^{|Y_L| \times C}$ , where  $|Y_L|$  is the available set of labeled instances in the dataset and  $C$  is the number of classes. Each row of  $Y$  corresponds to an instance, and each column represents a class. We utilise a binary matrix  $Y \in \{0, 1\}^{|Y_L| \times C}$ , where  $Y_{i,j} = 1$  if instance  $i$  belongs to class  $j$ , and  $Y_{i,j} = 0$  otherwise. Thus, the one-hot encoded labels matrix  $Y$  can be expressed as:

$$Y_{i,j} = \begin{cases} 1 & \text{if instance } i \text{ belongs to class } j \\ 0 & \text{otherwise} \end{cases} \quad (3.6)$$

In this representation, each row of  $Y$  contains a unique binary vector that indicates the class membership for that instance, following the one-hot encoding scheme.

#### Laplacian computation

The Laplacian matrix, denoted as  $L_{sym}$ , is used in the graph filtering phase. It provides insight into the structure and connectivity of the graph and allows the identification of anomalous nodes or patterns based on their deviation from the expected behavior within the graph. We utilize the eigenvectors of the graph Laplacian as the basis of the different frequencies. Specifically, the symmetric normalized Laplacian can be factored as:

$$L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = U \Lambda U^T \quad (3.7)$$

where  $U \in R^{N \times N} = [u_1, \dots, u_N]$ , where  $u_i \in R^N$  denotes the  $i_{th}$  eigenvector of  $L_{sym}$ ,  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_N)$  is the corresponding eigenvalue matrix, and  $N$  is the number of graph nodes. This vector set represents several oscillation patterns inside the graph. Each eigenvector is associated with a specific eigenvalue, which specifies the frequency associated with that eigenvector.

### Imbalanced data handling

Addressing class imbalance is crucial in anomaly detection, as anomalies are rare events compared to normal patterns. Real-world anomaly detection datasets often exhibit significant class imbalance, where the normal class dominates and anomalies are represented by a small number of instances. Failing to handle this class imbalance can result in biased models that prioritize the normal class and perform poorly in detecting anomalies. To mitigate the impact of class imbalance, we utilize the DataSplit technique. By appropriately splitting the data, we ensure a balanced distribution of normal and anomaly instances in each set. This balanced data split enhances the model's ability to detect subtle anomalies, improves overall detection performance, and reduces false positive rates. In the following chapter, we will further explore these techniques and additional preprocessing methods that further enhance the performance of the model.

### 3.4.2 Feature representation

The first layer of the model integrates the adjacency matrix  $A$  to capture the graph structure and  $L_{sym}$  to incorporate the spectral features of the input data. Following a similar approach to the AdaGNN model, the first embedding layer in Anomaly-AdaGNN employs an activation function to compute a new feature representation in  $H^{(1)}$  using the following mathematical formulation:

$$H^{(1)} = ReLU((H^{(0)} - \tilde{L}_{sym}H^{(0)}\Phi_1)\Theta) \quad (3.8)$$

where  $\Phi_1$  is  $\Phi = diag(\phi_1, \dots, \phi_F)$  of the first layer,  $\phi_j$  denotes the learnable parameter for the  $j_{th}$  feature channel,  $\Theta$  the weight matrix of the first layer,  $H^{(0)}$  is the feature matrix  $X \in R^{N \times F}$  of the graph, and  $\tilde{L}_{sym}$  is the Laplacian computed in the previous section.

The computation begins by multiplying the product of  $\tilde{L}_{sym}$ ,  $H^{(0)}$ , and  $\Phi_1$ , which captures the influence of the graph structure and spectral features on the initial feature matrix. This product represents the impact of neighboring nodes and their features. Subsequently, the result is subtracted from  $H^{(0)}$  to incorporate this influence. To introduce non-linearity, the ReLU activation function is then applied. Finally, the resulting matrix is multiplied by the weight matrix  $\Theta$  to obtain the new feature matrix  $H^{(1)}$ . This transformation integrates both local and global information from the graph, resulting in a new feature representation.

### 3.4.3 Adaptive frequency response

In our case, we will be working on eight layers of the AdaGNN model to obtain a better representation learning of the features and extract more complex and hierarchical patterns. Using deep layers in other models can cause an over-smoothing problem, however in this case the independent feature filters prevent that problem. The deep layers provide more capacity to learn intricate relationships and variations in the data, making the model more capable of capturing fine-grained details. Also, when multiple layers are stacked, the importance of different frequency components can be better captured as a set of trainable parameters, yielding a more complex frequency response function.

### Hidden layers

In the Anomaly-AdaGNN model, the process begins with each hidden layer taking the hidden embeddings generated by the previous layer  $H^{(k-1)}$  and producing new hidden embeddings  $H^{(k)}$  as output. Additionally, we have incorporated dropout layers into the model, which randomly set a fraction of the input values to zero during training to minimize the risk of overfitting. For the intermediate layers with  $2 \leq k \leq 8$ , our model employs an adaptive frequency response function to ensure computational efficiency, each feature channel has its dedicated filter that dynamically learns the frequency response function. This approach eliminates the need for creating a separate filter for each node, which could otherwise be computationally expensive. To introduce non-linearity into the embeddings, we have chosen to utilize the ReLU activation function in the model. Each hidden layer is calculated as follows:

$$H^{(k)} = \text{ReLU}(H^{(k-1)} - \tilde{L}_{sym} H^{(k-1)} \Phi_k) \quad (3.9)$$

where  $\Phi_k$  is the learnable parameters for the  $k$ -th layer. The steps involved in each hidden layer of the graph neural network are as follows:

- **Message passing:** The model performs message passing to aggregate information from the graph neighborhood of each node. For a given node  $v_i$ , the model gathers information from its graph neighbors  $N(v_i)$ . This aggregation is accomplished by utilizing the graph Laplacian matrix  $\tilde{L}_{sym}$  and the hidden embeddings  $H^{(k-1)}$  from the previous layer. The multiplication  $\tilde{L}_{sym} H^{(k-1)}$  represents the propagation of information from the neighbors to the central node  $v_i$ .
- **Aggregation:** After the message passing step, the model aggregates the gathered information to create a new hidden embedding for each node. This aggregation process combines the originally hidden embeddings  $H^{(k-1)}$  with the information propagated from the neighbors. In Anomaly-AdaGNN, the aggregation step is performed by subtracting the product  $\tilde{L}_{sym} H^{(k-1)} \Phi_k$  from  $H^{(k-1)}$ , where  $\Phi_k$  represents the learnable parameters specific to the  $k$ -th layer.
- **Update:** Once the aggregation step is complete, the hidden embeddings  $H^{(k)}$  for the  $k$ -th layer are updated. The update step introduces non-linearity to the embeddings using the ReLU activation function. ReLU applies an element-wise operation, setting negative values to zero and keeping positive values unchanged.

By repeating these steps for each hidden layer from 2 to 8, Anomaly-AdaGNN progressively refines the hidden embeddings, incorporating information from the local neighborhood up to the  $8_{\text{hop}}$  neighbors. This integration allows each node to have a representation of its immediate and more distant connections, leading to a deeper understanding of the underlying network structure.

### Adaptive frequency filters

For the first layer and the hidden layers  $2 \leq k \leq 8$ , the graph signal of each feature channel  $x_j$  ( $j \in \{1, \dots, F\}$ ) filtered by the  $j_{th}$  filter is formulated as:

$$z_j = x_j - \tilde{L}_{sym} x_j \phi_j \quad (3.10)$$



for the  $j_{th}$  feature channel and  $\phi_j$  represents the adaptive component of the graph filter, this component will learn the frequency of node features and increase the fitting capability of the model. Based on section 3.4.1 we can replace  $\tilde{L}_{sym}$  with  $\tilde{U}\tilde{\Lambda}\tilde{U}^T$  and now we have:

$$\begin{aligned}
z_j &= x_j - \tilde{L}_{sym}x_j\phi_j \\
&= x_j - \tilde{U}\tilde{\Lambda}\tilde{U}^Tx_j\phi_j \\
&= (\tilde{U}\tilde{U}^T - \tilde{U}\tilde{\Lambda}\tilde{U}^T\phi_j)x_j \\
&= \tilde{U}(I - \tilde{\Lambda}\phi_j)\tilde{U}^Tx_j \\
&= g_k(\tilde{\Lambda}i, \phi_j) \star x_j
\end{aligned} \tag{3.11}$$

consequently, the adaptive frequency response filter of the model is:

$$g_k(\tilde{\Lambda}i, \phi_j) = 1 - \phi_{j,k}\tilde{\Lambda}i \tag{3.12}$$

For our 8-layer AdaGNN framework, the adaptive frequency response function of the  $j_{th}$  input feature channel will be formulated as:

$$f_8(\tilde{\Lambda}i, \phi_j) = \prod_{k=1}^8 g_k(\tilde{\Lambda}i, \phi_{j,k}) = \prod_{k=1}^8 (1 - \phi_{j,k}\tilde{\Lambda}i) \tag{3.13}$$

where  $\phi_{j,k}$  is the learnable parameter of the  $j_{th}$  feature channel at layer  $k$ . Compared to the AmNet model, AdaGNN created an input feature channel instead of a node channel which makes it less computationally expensive. In addition, the AdaGNN parameter  $\phi_{j,k}$  can adjust the relative importance of high-frequency and low-frequency components at each layer  $k$  instead of only using low-frequency components.

### Hidden embedding

Eventually, the embedding  $Z_i$  of a node  $v_i$  with a set of  $F$  feature channels filtered in one layer will equal:

$$Z_i = X_i - \tilde{L}_{sym}X_i\Phi \tag{3.14}$$

where  $X_i$  is the node feature vector and  $\Phi$  is the set of feature filters. By subtracting the product of the graph Laplacian matrix  $\tilde{L}_{sym}$  and the node's feature vector  $X_i$  multiplied by the feature filters  $\Phi$ , we create an adaptive frequency function for each feature channel instead of generating separate adaptive filters for each node. This approach significantly reduces the computational complexity. Moreover, this type of embedding computation offers the benefit of preventing over-smoothing. By processing the model features separately for each channel, the model can retain more distinctive information and avoid excessively smoothing out the node representations.

Figure 3.5 is a visual representation of the embedding process for  $Z_i$ , where each feature channel (represented by the colors blue, yellow, green, and pink) is multiplied by its corresponding personalized set of filters  $\Phi = \text{diag}(\phi_1, \dots, \phi_F)$ . It is worth noting that the filters vary for each feature channel, allowing the model to capture both low and high-frequency features. This approach ensures that each node embedding incorporates information from neighboring nodes without excessively smoothing out the graph.

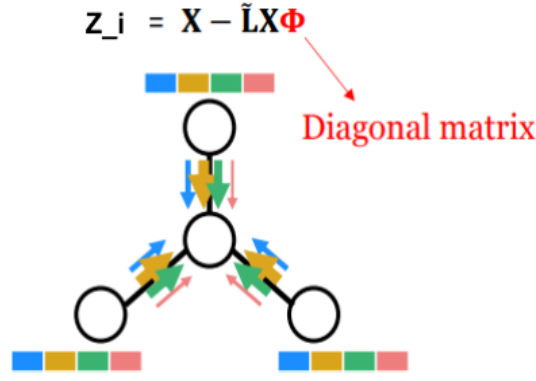


Figure 3.5: Anomaly-AdaGNN node embedding (Dong u. a., 2021)

### 3.4.4 Model prediction

Our primary objective is to enhance the accuracy of node classifications within a semi-supervised node classification framework. To achieve this, we focus on the generation and utilization of node embeddings. The embedding matrix in the last layer, denoted as  $H^{(8)}$ , serves as a key component in representing the set of nodes  $V$ . By applying a softmax activation function to  $H^{(8)}$ , we can extract representations that help determine the label class to which each node belongs. To obtain precise node classifications, we begin by using the expression:

$$H^{(8)} = ReLU(H^{(7)} - \tilde{L}_{sym}H^{(7)}\Phi_8) \quad (3.15)$$

the node representations then undergo a transition into the label space through the weight matrix  $W$ . Finally, we employ a softmax activation function to deliver the predictions represented by  $\hat{Y}$ , which allows us to determine the probabilities of nodes belonging to specific classes. The softmax classifier specifies the probability distribution across various classes. By utilizing the equation:

$$\hat{Y} = \sigma(H^{(8)}W) \quad (3.16)$$

where  $W \in R^{(L \times C)}$  denotes the weight matrix responsible for the transformation of node representations into the label space, we can derive predicted probabilities for each node's class membership. In summary, our approach leverages the power of node embeddings within a semi-supervised node classification framework using a softmax activation function and a carefully designed weight matrix.

### 3.4.5 Model optimisation

Our objective is to enhance the Anomaly-AdaGNN model for classifying anomalous node embeddings among other classes and training its adaptive frequency filter to effectively detect high and low-frequency features in nodes. To achieve this, we combine the loss function from the AdaGNN model, denoted as  $L_{AdaGNN}$ , with the Binary Cross-Entropy (BCE) Loss (Pytorch, 2023) to compute the loss for anomaly label prediction. The overall loss function is formulated as:

$$L_{Anomaly} = L_{BCE} + L_{AdaGNN} \quad (3.17)$$

here,  $L_{BCE}$  represents the binary cross-entropy loss, which is commonly employed in anomaly detection methods for binary classification tasks. The aim of using this loss

function is to leverage the prediction loss of node  $v_i$  in the anomaly class, encouraging the model to emphasize high-frequency data during the optimization phase and develop a better representation for anomalous nodes. The binary cross-entropy loss can be expressed as:

$$L_{BCE} = -\frac{1}{N} \sum_{i=1}^N [y_{i,a} \log(\hat{y}_{i,a}) + (1 - y_{i,a}) \log(1 - \hat{y}_{i,a})] \quad (3.18)$$

in this equation,  $\hat{y}_{i,a}$  represents the predicted label of node  $v_i$  in the anomaly class  $a$ ,  $y_{i,a}$  is the ground truth label of node  $v_i$  for the anomaly class, and  $N$  denotes the total number of graph nodes.

By combining  $L_{BCE}$  with  $L_{AdaGNN}$  in our Anomaly-AdaGNN model, we facilitate the learning and optimization process. The BCE loss, commonly used in anomaly detection methods, focuses on binary classification tasks. By incorporating this loss function, we encourage the model to pay attention to anomalous instances and emphasize high-frequency data during optimization. This helps the model develop a better understanding of anomalous nodes and enhances its ability to accurately classify them.

Moreover, the AdaGNN loss contributes to the overall learning and optimization process. It ensures that the model learns from the graph structure and captures relevant patterns and dependencies among nodes. This loss function assists in leveraging the adaptive frequency filter, enabling the model to identify both high and low-frequency features within nodes. By combining these two loss functions, our model benefits from their complementary nature. The BCE loss prioritizes anomaly classification, guiding the model to focus on accurately identifying and representing anomalous nodes. Simultaneously, the AdaGNN loss enhances the model's understanding of the graph structure, facilitating the extraction of meaningful features and improving overall performance.

## 3.5 Conclusion

In this chapter, we delved into the depths of the AdaGNN model and uncovered its potential as a valuable tool in the anomaly detection field. Furthermore, we introduced the graph neural network model Anomaly-AdaGNN, which integrates the power of AdaGNN's adaptive filters into the domain of anomaly detection. Overall, the Anomaly-AdaGNN model demonstrates a comprehensive approach to anomaly detection by leveraging graph neural networks, adaptive filters, and preprocessing techniques to improve detection performance and handle class imbalance. In the following chapter, we will establish the comparability of the Anomaly-AdaGNN model with state-of-the-art anomaly detection models. Through rigorous experimentation and evaluation, we seek to demonstrate the model's potential to yield excellent results on real-world datasets.

# Chapter 4

## Implementation and Experimental results

### 4.1 Introduction

In this chapter, we focus on testing the accuracy of Anomaly-AdaGNN. We detail experiments on three real-world anomaly datasets to evaluate the performance of Anomaly-AdaGNN compared to other state-of-the-art approaches such as AMNet (Chai u. a., 2022) the adaptive filtering anomaly detection model and CARE-GNN (Dou u. a., 2020a), which represent a part of the most precise anomaly detection models studied in our thesis. First, we will begin by describing the tools used while implementing our method and the preprocessing methods utilized. Then, we evaluate the effectiveness of the Anomaly-AdaGNN approach by applying it to multiple datasets. Finally, we discuss and analyze the models behavior and effectiveness.

### 4.2 Experimental setup

In this section, we provide details of the implementation process for our proposed solution, leveraging the selected tools. We outline the steps taken to develop and deploy our anomaly detection model on real-life datasets.

#### 4.2.1 Implementation details

All of the models were implemented on a virtual machine running on Windows 10 with the following specifications:

- RAM: 16.6 GB.
- Processor: common KVM processor 3.70 GHz (2 processors).

The choice was affected by the criteria of fast performance and availability of several data processing libraries, also because GNNs require optimal hardware and software resources.

#### 4.2.2 Databases

In order to assess the effectiveness of our anomaly detection model and benchmark it against state-of-the-art methods, we performed extensive evaluations employing diverse

datasets. Our objective was to compare Anomaly-AdaGNN performance against not only AMNet but also CARE-GNN a highly efficient anomaly detection model, and we aim to substantiate the effectiveness and efficiency of our proposed approach. To achieve this, we employed the Anomaly-AdaGNN framework and trained the following anomaly detection datasets.

## Yelp

Yelp is a dataset created for the study of GNN-based anomaly detection, it is also called the Yelp review dataset (Rayana und Akoglu, 2015). This dataset comprises reviews from hotels and restaurants, filtered by Yelp to include both legitimate and spam reviews. This dataset consists of interconnected nodes that represent Yelp reviews, where each node corresponds to a specific review, and nodes are linked based on the relationships between the reviews. To be more specific, nodes can be connected based on factors such as:

- Review similarity: nodes representing reviews that have similar content, sentiment, or other features can have an edge between them. The resemblance between reviews is calculated using techniques such as text similarity measures, and sentiment analysis.
- User similarity: nodes representing reviews written by the same user are connected. This helps to capture the relations between reviews by the same user and to provide insights into user behavior or preferences.
- Business similarity: nodes representing reviews of the same business or related businesses can have an edge connecting them. This allows capturing the connections between reviews of similar or related establishments, allowing analysis of business interactions and comparisons.

In this dataset, a node is labeled anomalous if it exhibits unique characteristics compared to the majority of reviews or if it represents a fake or spam review.

## Amazon

In addition to the Yelp dataset, we also examine the Amazon review dataset (McAuley und Leskovec, 2013). Specifically, we will work with the product reviews falling under the Musical Instruments category. In this dataset, each review is a node in the graph and has review text, rating, and reviewer information as node attributes. A pair of nodes have an edge based on the following relationships:

- Review similarity: edges are created between reviews that have similar characteristics, such as similar review text, ratings, or reviewer profiles. This captures the similarity or relatedness between reviews.
- Temporal order similarity: if the dataset contains temporal information (review timestamps) a connection is created between reviews that are temporally close to each other to capture the temporal ordering of reviews.
- Co-occurrence: if multiple reviews mention the same musical instrument or refer to similar topics, a connection can be created between them.

Similar to the approach with the Yelp dataset, specific reviews in the dataset may be labeled as anomalous based on different criteria. For example, unusually high or low ratings, or reviews containing suspicious or fraudulent content are labeled anomalous.

### Elliptic

To expand our study of GNN-based anomaly detection, we incorporate the Elliptic dataset (Weber u. a., 2019), this dataset offers a unique perspective on the anomaly detection problem by concentrating on the Bitcoin transaction network. Here each node in the dataset represents a transaction and has attributes such as transaction amount, and timestamp. Edges represent the flows of Bitcoin currency between these transactions. Similar to the previous datasets, the Elliptic dataset provides labels for each node indicating whether it is anomalous or not. The labeling is done based on real-world suspicious activity reports and law enforcement investigations. Anomalous nodes are associated with illegal and fraudulent activities, while normal nodes represent legitimate financial transactions. Table 4.1 shows the statistics of the dataset:

Dataset	Yelp	Amazon	Elliptic
Nodes	45,954	11,944	46,564
Features	32	25	93
Abnormal(%)	14.53	9.5	9.76

Table 4.1: Datasets statistics

### 4.2.3 Data preprocessing

During the initial phase of data preprocessing in this model, several important steps are carried out to ensure the data is properly prepared for subsequent computations. Firstly, we employ a one-hot encoding function to transform the categorical labels in the given datasets. This encoding procedure converts the categorical labels into a binary representation, enabling the calculation of the loss function at a later stage. Next, various preprocessing techniques are applied to the data. This includes constructing adjacency relation matrices, normalizing Laplacian matrices, and normalizing the feature matrix. By performing these operations, we aim to enhance the data’s suitability for subsequent computations and analyses.

Additionally, sparse matrices are converted to sparse tensors format compatible with the PyTorch framework. This conversion involves extracting the essential indices, values, and shape information from the sparse matrix and constructing an equivalent sparse tensor representation. To address the issue of imbalanced training, our anomaly-personalized DataSplit function divides the datasets. This division ensures a balanced distribution of normal and anomaly instances across the different sets, promoting effective model training and evaluation.

### Imbalanced data handling

The DataSplit is designed to address the challenge of imbalanced data. Given the node features  $X$ , corresponding labels  $Y$ , and anomaly label percentage the desired percentage

of instances used in the anomaly label. DataSplit performs stratified sampling to select a balanced number of instances from each label. The algorithm follows these steps:

1. Compute the total count of labels and select the anomaly and normal labels.
2. Determine the indices corresponding to the anomaly and normal labels within  $Y$ .
3. Calculate the number of instances to select based on anomaly label percentage.
4. Merge the chosen anomaly and normal indices into a single set of indices.
5. Subset the  $X$  and  $Y$  based on the selected indices.
6. Split the selected data into training, validation, and test sets.

The algorithm ensures the balance between the number of instances included for both the anomaly and normal labels while adhering to the desired split proportions. After splitting the data into training and validation subsets, we convert the preprocessed features and labels to PyTorch tensors to facilitate seamless integration with the neural network model employed. This assures that 50% of our training and validation data are anomalous data points to prevent generalizing the model on normal nodes behavior only. To demonstrate that Anomaly-AdaGNN balances the data labels the following graphs show the distribution of the Elliptic dataset labels before and after the splitting process:

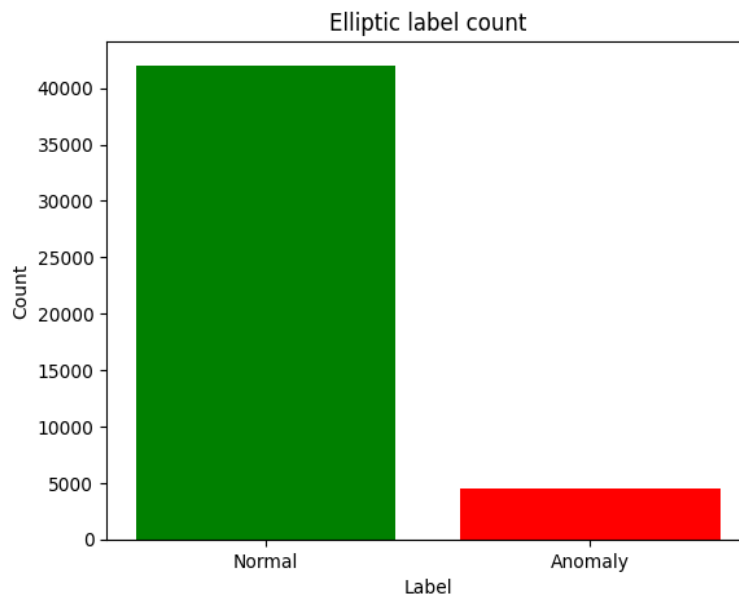


Figure 4.1: Illustration of the Elliptic dataset class imbalance

Figure 4.1 depicts a significant disparity between the anomaly labels and the normal labels. The red nodes represent the anomaly labels and their count amounts to only 4545 nodes. On the other hand, the green nodes correspond to the normal labels, and their count reaches 42019 nodes. This observation emphasizes a typical pattern found in anomaly detection datasets, where anomalies are typically a minority compared to normal instances. To tackle this inherent challenge, it is customary to train anomaly detection models, such as AMNet and CARE-GNN, on a subset of the dataset ranging from 10%

to 40%. This approach is adopted to avoid models from generalizing too much on the normal nodes, as they are overwhelmingly abundant compared to anomalies. By training on a reduced portion of the dataset, the models can focus more on accurately detecting and classifying anomalies, which is the primary objective of anomaly detection algorithms.

While training on a smaller subset can help prevent over-generalization on normal nodes, it also presents certain disadvantages. One disadvantage is that by reducing the training set, the models might have limited exposure to a variety of anomalies present in the dataset. This could potentially lead to reduced sensitivity and accuracy in detecting less common or novel anomalies that were not included in the training subset. Additionally, by focusing primarily on anomalies and training on a limited portion of the dataset, the models might not capture the full complexity and diversity of the normal instances. This could lead to increased false positives, where normal instances are mistakenly classified as anomalies. Therefore, striking a balance between training on an appropriate subset and capturing the diversity of both anomalies and normal instances is crucial to achieving optimal performance in anomaly detection models.

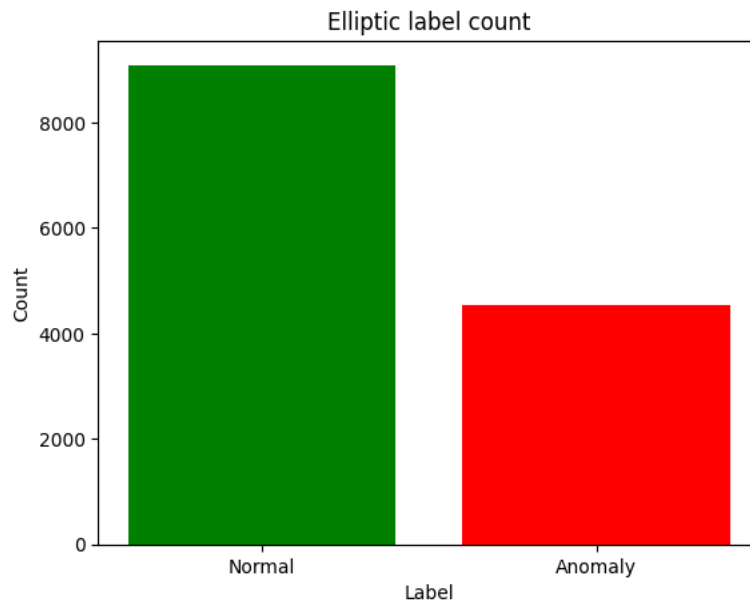


Figure 4.2: Illustration of the Elliptic dataset after balancing classes

In the provided Figure 4.2, we can observe the distribution of anomaly and normal labels, which helps us understand the balance within the dataset. Specifically, there are 4545 nodes labeled as anomalies, while 9090 nodes are labeled as normal instances. This indicates that the Anomaly-AdaGNN approach successfully addresses the common issue of imbalanced datasets in anomaly detection. It is worth noting that despite using a reduced amount of data, the model's accuracy remains intact. In this case, a smaller subset of the dataset was created, but it still represents a significant portion, approximately 29.29% of the original dataset. This highlights the effectiveness of the model in achieving a balance between accurate anomaly detection and efficient utilization of available data resources.



## 4.3 Experiments

To verify the capability of Anomaly-AdaGNN in the anomaly detection task, we compare it with various GNN baseline anomaly detection methods. This evaluation process enabled us to substantiate the efficiency and effectiveness of our proposed approach and provided valuable insights into its adaptability across diverse graph structures and real-world scenarios.

### 4.3.1 Comparison methods

First, we will do a thorough comparison between Anomaly-AdaGNN and AMNet and CARE-GNN the top-performing models on three different datasets. AMNet tackles the anomaly detection task from the spectral perspective using adaptive filtering(Chai u. a., 2022). CARE-GNN, on the other hand, is a spatial anomaly detection that utilizes reinforcement learning to select and aggregate neighbors based on different features (Dou u. a., 2020b). Our goal is to evaluate the efficiency and effectiveness of the proposed approach by comparing Anomaly-AdaGNN to AMNet specifically, which has demonstrated excellent performance. Furthermore, we extend the comparison to include other notable models such as DOMINANT, GeniePath, and GraphConsis. By considering the performance of these diverse techniques alongside Anomaly-AdaGNN, we acquire a comprehensive understanding of how Anomaly-AdaGNN compares to other state-of-the-art methods.

### 4.3.2 Hyperparameter tuning

The suggested approach utilizes the PyTorch framework along with the Adam optimizer to implement the model. The embedding layers in the model have a dimensionality of 8 for all layers. Two different activation functions are employed: ReLU for the first and hidden layers, and Softmax for the final layer. During the training process, the Adam optimizer is employed with a learning rate of 1e-3. The model undergoes training for a total of 300 epochs. To introduce regularisation, two hyperparameters named  $\alpha$ , and  $\beta$  are adjusted. The value of  $\alpha$  is set to 1e-6, while  $\beta$  is fine-tuned to 9e-12. These hyperparameters control the impact of regularisation on the models training. To prevent overfitting, a dropout rate of 0.2 is applied to the dropout layers. Dropout randomly deactivates a portion of the input units to 0 during training, which helps prevent the model from excessively relying on specific features. For the baseline methods, their original configurations are respected by utilizing their officially released implementations. This approach ensures a fair and unbiased comparison between the proposed model and the baseline methods.

### 4.3.3 Evaluation metrics

In order to evaluate and compare the performance of the Anomaly-AdaGNN model, we will utilize well-established evaluation metrics commonly used in anomaly detection. These metrics include the Area Under the Receiver Operating Characteristic curve (AUC-ROC)(Dou u. a., 2020a) and the Area Under the Precision-Recall curve (AUC-PR)(Ding u. a., 2019). The AUC-ROC metric provides a comprehensive assessment of the models ability to discriminate between anomalies and normal nodes. It considers the trade-off between the true positive rate (TPR) and the false positive rate (FPR) at various classification thresholds. By analyzing this trade-off, the AUC-ROC captures the models

effectiveness in identifying anomalies accurately. On the other hand, the AUC-PR focuses on the precision and recall trade-off. Precision measures the ratio of correctly identified anomalies among the nodes labeled as anomalous, while recall assesses the model’s ability to correctly identify all anomalies. The  $AUC - ROC$  formula is:

$$AUC - ROC = \int_0^1 TPR(FPR^{-1}(t)) dt \quad (4.1)$$

where  $TPR$  denotes the true positive rate and  $FPR$  denotes the false positive rate. The  $AUC - PR$  is equal to:

$$AUC - PR = \int_0^1 PR(R^{-1}(t)) dt \quad (4.2)$$

where  $PR$  represents the precision-recall curve, and  $R(t)$  denotes the recall at a given threshold  $t$ . These metrics are considered more suitable for anomaly detection due to the challenges they presented by anomaly detection tasks :

- Imbalanced datasets: anomaly detection datasets typically exhibit a significant class imbalance, a model that labels every node as normal will always achieve high accuracy, even if it fails to detect anomalies.
- Focus on anomalies: in the case of anomaly detection, the primary objective is to identify anomalies instead of reaching a high overall accuracy.

Overall,  $AUC - ROC$  and  $AUC - PR$  are widely used metrics in anomaly detection as they handle the challenges posed by imbalanced datasets, concentrates on detecting anomalies, and provide an exhaustive evaluation of the model’s performance. We aim to quantitatively assess and compare the effectiveness of Anomaly-AdaGNN, AmNet, and CARE-GNN in multiple anomaly detection tasks using these metrics.

## 4.4 Results discussion

In the following section, we conduct evaluations to assess the performance of Anomaly-AdaGNN on three real-world datasets. Our main objective is to address the following research questions. How does Anomaly-AdaGNN compare baseline methods in terms of performance on real-world graph anomaly detection tasks? Does Anomaly-AdaGNN effectively incorporate adaptive filtering into the network?

### 4.4.1 Experimental results

To assess the effectiveness of our proposed model on various real-world datasets, we conducted extensive training and testing on three different examples: Yelp, Amazon, and Elliptics. The objective was to detect anomalies within these datasets accurately. The following figures display the results of model training and evaluation, providing compelling evidence of the models performance in anomaly detection for each dataset. In our evaluation, we considered several performance metrics to gauge the effectiveness of Anomaly-AdaGNN. These metrics include validation accuracy ( $acc - val$ ), validation area under the receiver operating characteristic curve ( $auc - roc - val$ ), the area under the precision-recall curve ( $auc - pr - val$ ), and the time taken for the model to process the data. These metrics allow us to evaluate the models accuracy, its ability to distinguish

between normal and anomalous instances, and its efficiency in processing the data. By considering these factors, we gain a comprehensive understanding of the model performance on different datasets.

```
acc_val: 0.9213
auc_roc_val: 0.9101
auc_pr_val: 0.8180
time: 2.0099s
```

Figure 4.3: Elliptic dataset results of Anomaly-AdaGNN

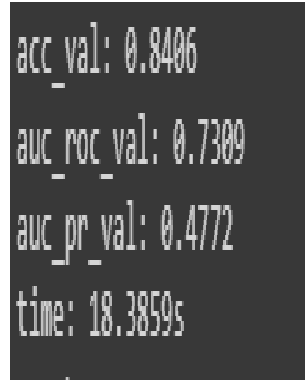
For the Elliptic dataset in Figure 4.3, the validation accuracy of 92.13% indicates the models ability to accurately classify instances. The validation AUC-ROC is 91.01% reflecting the models capability to distinguish normal and anomalous patterns. The precision-Recall AUC score is 81.80% signifies a balanced performance between precision and recall. This means that Anomaly-AdaGNN identifies anomalous nodes while minimizing false positives successfully. Finally, the model took 2.80 seconds to process one epoch.

```
acc_val: 0.9737
auc_roc_val: 0.9055
auc_pr_val: 0.7215
time: 10.7991s
Optimization Finished!
Total time elapsed: 3317.0591s
```

Figure 4.4: Amazon dataset results of Anomaly-AdaGNN

Similarly, for the Amazon dataset in Figure 4.4, the accuracy of the validation data is 97.37% demonstrating the models capacity to correctly classify instances. The validation AUC-ROC is 90.55% and the precision-Recall AUC score is 72.15% further assuring Anomaly-AdaGNN capability to identify anomalous nodes and minimise false positives successfully. Finally, the model took 10 seconds to process one epoch and 3317 seconds to finish the training and evaluation. Also When we trained the model on the Yelp dataset, Figure 4.5, we obtained an accuracy of 84.06% demonstrating the models capacity to correctly classify instances. The validation AUC-ROC is 73.09% and the AUC-PR score

is 47.72% meaning that the model performs reasonably well. Although slightly lower than the previous dataset, these values still showcase the models efficacy. Finally, the model took 18,38 seconds to process one epoch. These results suggest that the model still has some room for improvement in terms of precision and minimizing false positives.



```
acc_val: 0.8406
auc_roc_val: 0.7309
auc_pr_val: 0.4772
time: 18.3859s
```

Figure 4.5: Yelp dataset results of Anomaly-AdaGNN

The training and validation time indicates how efficient the model is in processing given datasets. For example, for the Amazon dataset, Anomaly-AdaGNN finished 300 epochs in approximately 55.28 minutes (3317 seconds). Table 4.2 summarises the results for Anomaly-AdaGNN validation on the Elliptic, Amazon, and Yelp datasets.

Dataset	Accuracy (%)	AUC-ROC (%)	AUC-PR (%)	Epoch time (seconds)
Elliptic	92.13	91.01	81.80	2.80
Amazon	97.37	90.55	72.15	10.79
Yelp	84.06	73.09	47.72	18.38

Table 4.2: Anomaly-AdaGNN Results on Different Datasets

These results exhibit the models ability to analyze and detect anomalies in complex datasets while maintaining a noncomplex architecture. Overall, these findings support that Anomaly-AdaGNN is efficient for anomaly detection tasks on the Elliptics, Yelp, and Amazon datasets.

## 4.4.2 Results analysis

In this section, we investigate the incorporation of AdaGNN adaptive filtering into the Anomaly-AdaGNN model. Additionally, we compare the performance of Anomaly-AdaGNN with state-of-the-art baselines in real-world graph anomaly detection tasks.

### Adaptive filters analysis

To offer a comprehensive insight into the model, we have visualized the frequency response function that was trained on the Elliptic dataset. This representation allows us to analyze the models behavior and understand how it responds to different frequencies within the Elliptic dataset. By examining the frequency response function, we can gain

insights into the model ability to detect anomalies and make predictions based on the frequency patterns present in the dataset. The figure showcases 93 adaptive frequency filters, represented by distinct colors, which were learned from the experimental dataset. The colored curves denote the adaptive frequency filters of the eighth layer  $f_8(\tilde{\lambda}_i, \phi_j)$  of Anomaly-AdaGNN across  $F$  feature channels. By visualizing these filter groups, we acquire valuable insights into the behavior of the Anomaly-AdaGNN model, comprehending its ability to extract and utilize diverse frequency components in the context of anomaly detection. The visualization of this frequency response function is presented in Figure 4.6.

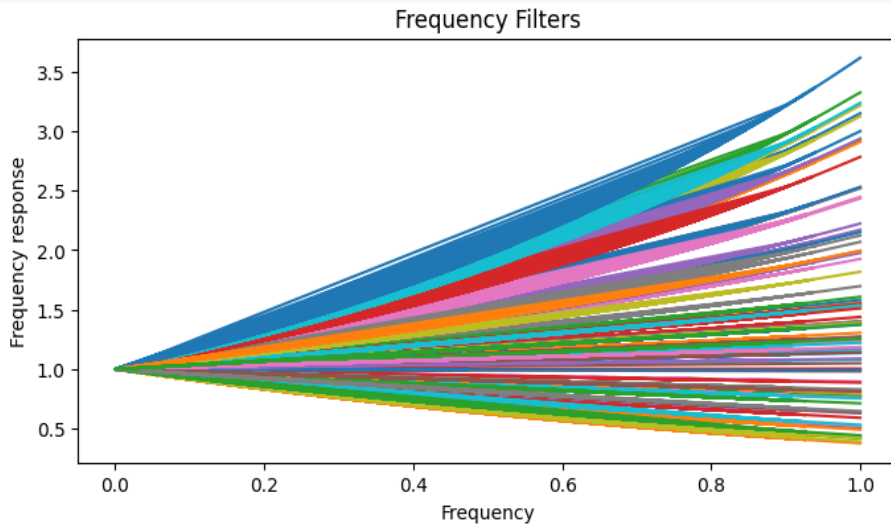


Figure 4.6: Frequency response function learned from Elliptic by Anomaly-AdaGNN

We observe a subset of filters in the Anomaly-AdaGNN model that exhibit a distinct low-pass characteristic. These filters effectively preserve the low-frequency components present in the data. This preservation of low-frequency components allows the model to capture the underlying trends and smooth variations in the graph and to detect subtle deviations from the expected behavior. By preserving the low-frequency components, Figure 4.7, the model can effectively distinguish the overall structure and patterns of the graph.

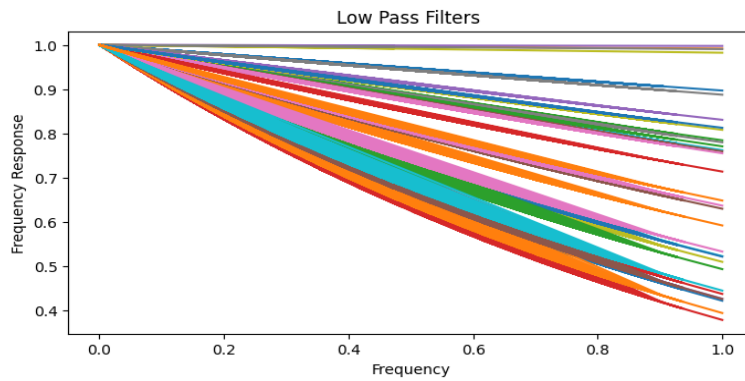


Figure 4.7: Low-frequency filters learned from Elliptic by Anomaly-AdaGNN

Another intriguing aspect of the Anomaly-AdaGNN model’s performance is the emergence of high-pass filters. Hence, the models ability to identify and accentuate anomalies characterized by distinct high-frequency patterns. By leveraging these high-pass filters, illustrated in Figure 4.8 the Anomaly-AdaGNN model becomes proficient in detecting anomalies that are often associated with rapid fluctuations, sharp transitions, or sudden bursts of activity.

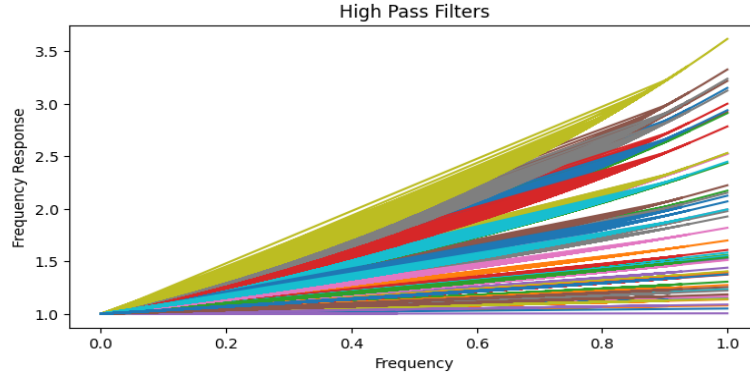


Figure 4.8: High-frequency filters learned from Elliptic by Anomaly-AdaGNN

This duality aligns perfectly with the fundamental concept that both low and high-frequency information play crucial roles in anomaly detection. Notably, it showcases the capacity of the model Anomaly-AdaGNN to learn filters that proficiently capture multiple frequency signals. By plotting the filter group, we observe that the filters exhibit both low-pass and high-pass properties. This observation aligns with the key idea that anomaly detection benefits from both low and high-frequency information. Moreover, it implies that Anomaly-AdaGNN is capable of capturing multiple frequency signals adaptively.

## Results comparison

After discussing the models training results, we now compare them to cutting-edge anomaly detection. Table 4.3 represents a comparison of the  $AUC - ROC$  and  $AUC - PR$  scores obtained by multiple models on the corresponding datasets:

Method	Yelp		Amazon		Elliptic	
	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR
CARE-GNN	78.41	38.90	89.54	52.65	85.84	49.81
AMNet	85.85	57.77	89.85	60.24	88.52	74.62
Anomaly-AdaGNN	73.09	47.72	90.55	72.15	91.01	81.80

Table 4.3: AUC-ROC and AUC-PR scores for anomaly detection models

Notably, the proposed model, Anomaly-AdaGNN, exhibited superior performance compared to CARE-GNN and AmNet. For the  $AUC - ROC$ , it achieved impressive scores of 90.55% on the Amazon dataset and 91.01% on the Elliptic dataset. However, its performance was insufficient compared to other models on the Yelp dataset and remained consistent at 73.09%.

Looking at the  $AUC - PR$  scores, for the Yelp dataset, although Anomaly-AdaGNN

scored lower than the AMNet model it still showed the models effectiveness in capturing anomalies compared to the CARE-GNN model. Moving to the Amazon dataset, Anomaly-AdaGNN achieves the highest AUC-PR score of 72.15%, highlighting its capability to capture anomalies effectively on this dataset.

Finally, for the Elliptic dataset, Anomaly-AdaGNN also achieved the highest AUC-PR score of 81.80%. Overall, this table highlights not only that Anomaly-AdaGNN showcases potential for practical application in anomaly detection tasks but also has distinguishable performance compared to the existing models, CARE-GNN and AmNet. In addition to the evaluated models mentioned in the table above, we conducted further tests on other anomaly detection models to provide a complete analysis in Figure 4.9.

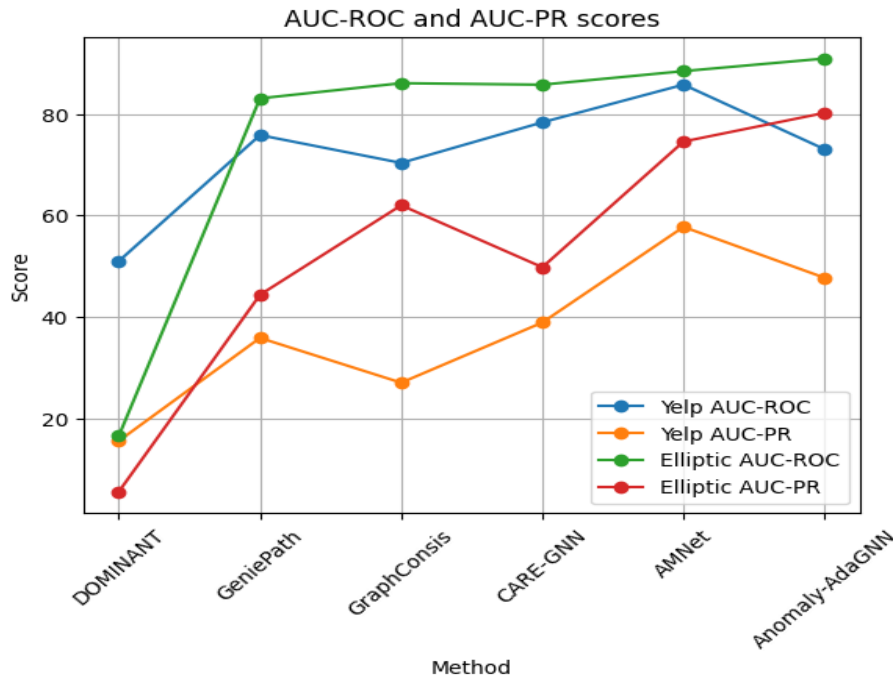


Figure 4.9: AUC-ROC and AUC-PR scores comparison

Based on the generated graph, we can analyze the AUC-ROC and AUC-PR scores for different anomaly detection methods on the Yelp and Elliptic datasets. For the Yelp dataset, DOMINANT has the lowest scores, GeniePath outperforms DOMINANT and earns superior results in both AUC-ROC and AUC-PR, and GraphConsis outperforms GeniePath. When compared to GraphConsis, CARE-GNN scores higher in both AUC-ROC and AUC-PR. AMNet beats all previous techniques, earning the highest ratings. Anomaly-AdaGNN doesn't outperform AMNet but still delivers competitive results.

Like the Yelp dataset, DOMINANT has the lowest AUC-ROC and AUC-PR values for the Elliptic dataset. GeniePath significantly outperforms DOMINANT, and GraphConsis outperforms GeniePath. In terms of AUC-ROC, CARE-GNN performs similarly to GraphConsis, but with a lower AUC-PR score. AMNet beats all previous techniques, achieving the highest AUC-ROC and AUC-PR values. Anomaly-AdaGNN works remarkably well, getting the greatest AUC-ROC and AUC-PR scores among all algorithms.

Overall, AMNet and Anomaly-AdaGNN outperform other approaches on both datasets, demonstrating their effectiveness in anomaly detection tasks. These findings emphasize the use of adaptive frequency component models, such as Anomaly-AdaGNN, in capturing high-frequency data while avoiding over-smoothing difficulties that are frequent in GNN models.

## 4.5 Conclusion

In conclusion, this chapter focused on the experimentations of our anomaly detection model, Anomaly-AdaGNN. We talked about the tools used in the implementation process and provided details about the datasets used for evaluation. We also compared Anomaly-AdaGNN against two state-of-the-art anomaly detection approaches and showcased its high accuracy compared to other models. We demonstrated the effectiveness of Anomaly-AdaGNN in detecting anomalies in real-world datasets using adaptive frequency filters. Next, we complete this master's thesis with an in detail conclusion of our work.



# General Conclusion

In this thesis, the goal was to provide a comprehensive understanding of the outlier detection problem and to introduce and analyze the Anomaly-AdaGNN model as an approach for detecting anomalies in graph data.

The initial chapter established the necessary background knowledge. We provided an introduction to machine learning, traditional machine learning and deep learning techniques, and graph representation learning. We discussed graph representation methods for graph structure and graph neural networks. Additionally, we talked about spectral filtering on graphs, including graph laplacians, and convolutions, leading to the understanding of graph convolutional networks. The focus was on building a foundation to understand graph neural networks and their applications.

In Chapter 2, we conducted a thorough review of existing GNN-based anomaly detection methods. We categorized these methods into two sections, spatial anomaly detection models and spectral anomaly detection models. Each category contained several representative models that we described and summarised in detail. We also discussed the feedback and limitations of these approaches. After providing a comprehensive overview of the existing approaches and their strengths and weaknesses we conducted that the AMNet model was the most efficient anomaly detection method due to its capability to detect both low and high-frequency data.

Chapter 3 illustrated the Anomaly-AdaGNN model, which was the main contribution of this thesis. The goal was to address the limitations of existing approaches by proposing adopting the AdaGNN model for the anomaly detection task. AdaGNN incorporates adaptive frequency response filtering in the node embedding process. We proposed to use this advantage in outlier detection, which proved its efficiency in existing research. We presented the problem statement in existing approaches that we aim to solve and described the architecture of the proposed model, highlighting its ability to adaptively filter the frequency response of the graph data.

In the implementation and experiments chapter, we provided details about our experimental setup, including implementation specifics, the databases we used, and data preprocessing procedures. We also, conducted comprehensive experiments to compare the performance of our proposed model with baseline methods. The results were analyzed, highlighting the strengths and limitations of the Anomaly-AdaGNN model. The results found demonstrated the effectiveness of the Anomaly-AdaGNN model in detecting anomalies in different datasets. Although the Anomaly-AdaGNN model did not achieve superior results in every dataset, we consistently observed comparable performance and highlighted the models' capability compared to state-of-the-art models.

In our ongoing research efforts, we are dedicated to further advancing and expanding the capabilities of the Anomaly-AdaGNN model. We aim to adapt and extend its functionality to encompass multi-relational graphs, thereby broadening the scope and unlocking new possibilities for anomaly detection in complex network structures. By incorporating multiple types of relationships and interactions within the graph data, we can enhance the model's ability to capture diverse patterns and anomalies that may exist across different relational dimensions.

In conclusion, in this thesis, we harness the power of adaptive frequency response filtering specifically designed for graph data for anomaly detection. The experimental results showcased in this research demonstrate the model's efficacy and effectiveness in accurately detecting anomalies across a variety of datasets. By leveraging the inherent structure and characteristics of the graphs, the Anomaly-AdaGNN model exhibits excellent performance and robustness, making it a valuable tool for anomaly detection in real-world scenarios.

# Bibliography

- [Akoglu u. a. 2015] AKOGLU, Leman ; TONG, Hanghang ; KOUTRA, Danai: Graph based anomaly detection and description: a survey. In: *Data mining and knowledge discovery* 29 (2015), S. 626–688
- [Balcilar u. a. 2021] BALCILAR, Muhammet ; GUILLAUME, Renton ; HÉROUX, Pierre ; GAÜZÈRE, Benoit ; ADAM, Sébastien ; HONEINE, Paul: Analyzing the expressive power of graph neural networks in a spectral perspective. In: *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021
- [Chai u. a. 2022] CHAI, Ziwei ; YOU, Siqu ; YANG, Yang ; PU, Shiliang ; XU, Jiarong ; CAI, Haoyang ; JIANG, Weihao: Can Abnormality be Detected by Graph Neural Networks? In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, Vienna, Austria, 2022, S. 23–29
- [Ciaburro 2017] CIABURRO, Giuseppe: Introducing Deep Learning. In: *MATLAB for Machine Learning*, 2017, S. 981–990
- [Ding u. a. 2019] DING, Kaize ; LI, Jundong ; BHANUSHALI, Rohit ; LIU, Huan: Deep anomaly detection on attributed networks. In: *Proceedings of the 2019 SIAM International Conference on Data Mining* SIAM (Veranst.), 2019, S. 594–602
- [Dong u. a. 2021] DONG, Yushun ; DING, Kaize ; JALAIAN, Brian ; JI, Shuiwang ; LI, Jundong: Adagnn: Graph neural networks with adaptive frequency response filter. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, S. 392–401
- [Dou u. a. 2020a] DOU, Yingdong ; LIU, Zhiwei ; SUN, Li ; DENG, Yutong ; PENG, Hao ; YU, Philip S.: Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, S. 315–324
- [Dou u. a. 2020b] DOU, Yingdong ; LIU, Zhiwei ; SUN, Li ; DENG, Yutong ; PENG, Hao ; YU, Philip S.: Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, S. 315–324
- [Ge u. a. 2018] GE, Shuaijun ; MA, Guixiang ; XIE, Sihong ; PHILIP, S Y.: Securing behavior-based opinion spam detection. In: *2018 IEEE International Conference on Big Data (Big Data)* IEEE (Veranst.), 2018, S. 112–117
- [Hamilton u. a. 2017] HAMILTON, Will ; YING, Zhitao ; LESKOVEC, Jure: Inductive representation learning on large graphs. In: *Advances in neural information processing systems* 30 (2017)

- [Hoogs u. a. 2007] HOOGS, Bethany ; KIEHL, Thomas ; LACOMB, Christina ; SENTURK, Deniz: A genetic algorithm approach to detecting temporal patterns indicative of financial statement fraud. In: *Intelligent Systems in Accounting, Finance & Management: International Journal* 15 (2007), Nr. 1-2, S. 41–56
- [Kaghazgaran u. a. 2019] KAGHAZGARAN, Parisa ; ALFIFI, Majid ; CAVERLEE, James: Wide-ranging review manipulation attacks: Model, empirical study, and countermeasures. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, S. 981–990
- [Kaghazgaran u. a. 2018] KAGHAZGARAN, Parisa ; CAVERLEE, James ; SQUICCIA-RINI, Anna: Combating crowdsourced review manipulators: A neighborhood-based approach. In: *Proceedings of the eleventh ACM international conference on web search and data mining*, 2018, S. 306–314
- [Kipf und Welling 2016] KIPF, Thomas N. ; WELLING, Max: Semi-supervised classification with graph convolutional networks. In: *arXiv preprint arXiv:1609.02907* (2016)
- [Leskovec 2023] LESKOVEC, Jure: *CS224W: Machine Learning with Graphs*. <http://cs224w.stanford.edu>. 2023. – Accessed: 2023-05-20
- [Li u. a. 2018a] LI, Qimai ; HAN, Zhichao ; WU, Xiao-Ming: Deeper insights into graph convolutional networks for semi-supervised learning. In: *Proceedings of the AAAI conference on artificial intelligence* Bd. 32, 2018
- [Li u. a. 2018b] LI, Qimai ; HAN, Zhichao ; WU, Xiao-Ming: Deeper insights into graph convolutional networks for semi-supervised learning. In: *Proceedings of the AAAI conference on artificial intelligence* Bd. 32, 2018
- [Liu u. a. 2020] LIU, Zhiwei ; DOU, Yingtong ; YU, Philip S. ; DENG, Yutong ; PENG, Hao: Alleviating the inconsistency problem of applying graph neural network to fraud detection. In: *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, 2020, S. 1569–1572
- [Liu u. a. 2019] LIU, Ziqi ; CHEN, Chaochao ; LI, Longfei ; ZHOU, Jun ; LI, Xiaolong ; SONG, Le ; QI, Yuan: Geniepath: Graph neural networks with adaptive receptive paths. In: *Proceedings of the AAAI Conference on Artificial Intelligence* Bd. 33, 2019, S. 4424–4431
- [Luger 2005] LUGER, George F.: *Artificial intelligence: structures and strategies for complex problem solving*. Pearson education, 2005
- [McAuley und Leskovec 2013] MCAULEY, Julian J. ; LESKOVEC, Jure: From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In: *Proceedings of the 22nd international conference on World Wide Web*, 2013, S. 897–908
- [McPherson u. a. 2001] MCPHERSON, Miller ; SMITH-LOVIN, Lynn ; COOK, James M.: Birds of a feather: Homophily in social networks. In: *Annual review of sociology* 27 (2001), Nr. 1, S. 415–444
- [Merkwirth und Lengauer 2005] MERKWIRTH, Christian ; LENGAUER, Thomas: Automatic generation of complementary descriptors with molecular graph networks. In: *Journal of chemical information and modeling* 45 (2005), Nr. 5, S. 1159–1168

- [Nt und Maehara 2019] NT, Hoang ; MAEHARA, Takanori: Revisiting graph neural networks: All we have is low-pass filters. In: *arXiv preprint arXiv:1905.09550* (2019)
- [Panagiotis 2023] PANAGIOTIS, Antoniadis: *Epoch vs Batch vs Mini-Batch*. <https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch>. Mar 2023
- [PyG 2023] PYG: *PyTorch Geometric Documentation: Data Handling Cheat Sheet*. [https://pytorch-geometric.readthedocs.io/en/latest/notes/data\\_cheatsheet.html](https://pytorch-geometric.readthedocs.io/en/latest/notes/data_cheatsheet.html). 2023
- [Pytorch 2023] PYTORCH: *Introduction to Deep Learning: What Are Convolutional Neural Networks? Video*. <https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html#torch.nn.BCELoss>. Mar 2023. – Accessed: 2023-05-20
- [Rayana und Akoglu 2015] RAYANA, Shebuti ; AKOGLU, Leman: Collective opinion spam detection: Bridging review networks and metadata. In: *Proceedings of the 21th acm sigkdd international conference on knowledge discovery and data mining*, 2015, S. 985–994
- [Rosenblatt 1958] ROSENBLATT, F.: The perceptron: a probabilistic model for information storage and organization in the brain. 15 (1958)
- [Russell und Norvig 2003] RUSSELL, Stuart ; NORVIG, Peter: Artificial intelligence: A modern approach, 2/E. In: *Pretence artificial Hall series in intelligence, Chapter Intelligent Agent* (2003), S. 31–52
- [Shalev-Shwartz und Ben-David 2014] SHALEV-SHWARTZ, Shai ; BEN-DAVID, Shai: *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014
- [Theobald 2017] THEOBALD, Oliver: *Machine learning for absolute beginners: a plain English introduction*. Bd. 157. Scatterplot press London, UK, 2017
- [Weber u. a. 2019] WEBER, Mark ; DOMENICONI, Giacomo ; CHEN, Jie ; WEIDELE, Daniel Karl I. ; BELLEI, Claudio ; ROBINSON, Tom ; LEISERSON, Charles E.: Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. In: *arXiv preprint arXiv:1908.02591* (2019)
- [Wu u. a. 2019] WU, Felix ; SOUZA, Amauri ; ZHANG, Tianyi ; FIFTY, Christopher ; YU, Tao ; WEINBERGER, Kilian: Simplifying graph convolutional networks. In: *International conference on machine learning* PMLR (Veranst.), 2019, S. 6861–6871
- [Wu u. a. 2020] WU, Zonghan ; PAN, Shirui ; CHEN, Fengwen ; LONG, Guodong ; ZHANG, Chengqi ; PHILIP, S Y.: A comprehensive survey on graph neural networks. In: *IEEE transactions on neural networks and learning systems* 32 (2020), Nr. 1, S. 4–24
- [Yue u. a. 2007] YUE, Dianmin ; WU, Xiaodan ; WANG, Yunfeng ; LI, Yue ; CHU, Chao-Hsien: A review of data mining-based financial fraud detection research. In: *2007 International Conference on Wireless Communications, Networking and Mobile Computing* Ieee (Veranst.), 2007, S. 5519–5522
- [Zhang u. a. 2021] ZHANG, Aston ; LIPTON, Zachary C. ; LI, Mu ; SMOLA, Alexander J.: Dive into deep learning. In: *arXiv preprint arXiv:2106.11342* (2021)