DEGREE PROJECT IN TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

Real-time Anomaly Detection on Financial Data

Master Thesis Project

Anna Martignano

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Author

Anna Martignano
martigna@kth.se | anna.martignano@mail.polimi.it
Department of Software and Computer Systems
KTH Royal Institute of Technology
Department of Electronics, Informatics and Bionengineering
Polytechnic University of Milan

Place for Project

Swedbank AB
Stockholm - Sweden
RISE Research Institute of Sweden
Kista, Stockholm - Sweden

Examiners

Prof. Amir H. Payberah
KTH Royal Institute of Technology

Prof. Emanuele Della Valle
Polytechnic University of Milan

Supervisors

Mehrdad Mamaghani
Analytics and Artificial Intelligence Team - Swedbank

Paris Carbone
Conitnuous Deep Analytics Group - RISE

Abstract

This work presents an investigation of tailoring Network Representation Learning (NRL) for an application in the Financial Industry. NRL approaches are data-driven models that learn how to encode graph structures into low-dimensional vector spaces, which can be further exploited by downstream Machine Learning applications. They can potentially bring a lot of benefits in the Financial Industry since they extract in an automatic way features that can provide useful input regarding graph structures, called embeddings. Financial transactions can be represented as a network, and through NRL, it is possible to extract embeddings that reflect the intrinsic inter-connected nature of economic relationships. Such embeddings can be used for several purposes, among which Anomaly Detection to fight financial crime.

This work provides a qualitative analysis over state-of-the-art NRL models, which identifies Graph Convolutional Network (ConvGNN) as the most suitable category of approaches for Financial Industry but with a certain need for further improvement. Financial Industry poses additional challenges when modelling a NRL solution. Despite the need of having a scalable solution to handle real-world graph with considerable dimensions, it is necessary to take into consideration several characteristics: transactions graphs are inherently dynamic since every day new transactions are executed and nodes can be heterogeneous. Besides, everything is further complicated by the need to have updated information in (near) real-time due to the sensitivity of the application domain. For these reasons, GraphSAGE has been considered as a base for the experiments, which is an inductive ConvGNN model. Two variants of GraphSAGE are presented: a dynamic variant whose weights evolve accordingly with the input sequence of graph snapshots, and a variant specifically meant to handle bipartite graphs. These variants have been evaluated by applying them to real-world data and leveraging the generated embeddings to perform Anomaly Detection. The experiments demonstrate that leveraging these variants leads to

comparable results with other state-of-the-art approaches, but having the advantage of being suitable to handle real-world financial data sets.

Keywords

Network Representation Learning, Anomaly Detection, Financial Industry, Graph Neural Networks, Dynamic Graphs, Heterogeneous Graphs

Abstract

Detta arbete presenterar en undersökning av tillämpningar av Network Representation Learning (NRL) inom den finansiella industrin. Metoder inom NRL möjliggör datadriven kondensering av grafstrukturer till lågdimensionella och lätthanterliga vektorer.

Dessa vektorer kan sedan användas i andra maskininlärningsuppgifter. Närmare bestämt, kan metoder inom NRL underlätta hantering av och informantionsutvinning ur beräkningsintensiva och storskaliga grafer inom den finansiella sektorn, till exempel avvikelsehantering bland finansiella transaktioner. Arbetet med data av denna typ försvåras av det faktum att transaktionsgrafer är dynamiska och i konstant förändring. Utöver detta kan noderna, dvs transaktionspunkterna, vara vitt skilda eller med andra ord härstamma från olika fördelningar.

I detta arbete har Graph Convolutional Network (ConvGNN) ansetts till den mest lämpliga lösningen för nämnda tillämpningar riktade mot upptäckt av avvikelser i transaktioner. GraphSAGE har använts som utgångspunkt för experimenten i två olika varianter: en dynamisk version där vikterna uppdateras allteftersom nya transaktionssekvenser matas in, och en variant avsedd särskilt för bipartita (tvådelade) grafer. Dessa varianter har utvärderats genom användning av faktiska datamängder med avvikelsehantering som slutmål.

Nyckelord

Nätverkiskt Representationsinlärning, Avvikelse av Anomali, Finansiell Industri, Grafiskt Neuralt Nätverk, Dynamiska Grafer, Heterogena Grafer

Acknowledgements

I would like to thank all the people who made it possible to develop this project for the support they gave me. First of all, the Analytics and Artificial Intelligence (A&AI) team at Swedbank AB, in particular Mehrdad Mamaghani, who provided and guided me through this challenging project in collaboration with the Continuous Deep Analytics (CDA) at RISE, under the professional supervision of Paris Carbone. I am grateful for all the support and precious guidelines provided by Prof. Amir H. Payberah and Prof. Emanuele Della Valle; it has been particularly motivating and enriching to involve two excellent universities in the development of the project.

Also, I would like to thank EIT Digital for this enriching study programme, which gave me the opportunity not only to deepen my studies but also to be introduced into a vibrant community all across Europe full of inspiring people and initiatives.

Last but not least, I would like to thank my family, my friends, and Simone for all the support, affection, and patience.

Acronyms

AA Adamic-Adar
AML Anti-Money Laundering
BiNE Bipartite Network Embedding
BP Belief Propagation
CART Classification And Regression Tree
CMD Compact Matrix Decomposition
CN Common Neighbours
CPU Central Processing Unit
DANE Dynamic Attributed Network Embedding
DBMM Dynamic Behavioral Mixed-Membership
CNN Convolutional Neural Network
CNNs Convolutional Neural Networks
ConvGNN Graph Convolutional Network
ConvGNNs Graph Convolutional Networks
Evolve-GCN Evolve-Graph Convolutional Network
GAE Graph Autoencoder
GAE* Graph Autoencoder [1]
GAEs Graph Autoencoders
GCN Graph Convolutional Network
GNN Graph Neural Network
GNNs Graph Neural Networks
GPU Graphical Processing Unit
GraphSAGE Graph SAmple and aggreGatE
GRU Gated Recurrent Unit
HITS Hyperlink-Induced Topic Search
HOPE Higher Order Proximity preserved Embedding

ID Iterative Dichotomiser 3

KL Kullback-Leibler

LINE Large-scale Information Network Embedding

LSTM Long-Short-Term-Memory

NRL Network Representation Learning

PCA Principal Component Analysis

RecGNNs Recurrent Graph Neural Networks

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

RNNs Recurrent Neural Networks

RPR Rooted Page Rank

RMSE Root Mean Squared Error

SDNE Structural Deep Network Embedding

Skip-GCN Skip-Graph Convolutional Network

STGNN Spatial-temporal Graph Neural Network

STGNNs Spatial-temporal Graph Neural Networks

SVD Singular Value Decomposition

UTXO Unspent Transaction Output

VGAE Variational Graph Autoencoder

Contents

Chapter 1

Introduction

Discovering unexpected observations in data, namely *anomaly detection*, is a fundamental machine learning and data mining task, which is widely applied in many industrial practices [2]. In the financial industry, anomaly detection can be utilised to fight criminal events such as frauds aimed at customers. Such applications of anomaly detection depend heavily on the usage of financial transaction data.

A *fraudulent transaction* is defined as an unauthorised, deceptive or manipulated transaction intended to provide the fraudster with an unlawful financial gain by depriving a victim of a transaction amount. For example, an unlawful entity who gets hold of the bank account credentials of a victim and transfers the money from the victim's account to another account. In money laundering, specific transactions are conducted in such manner to enable the transfer of funds from or to criminal activities. As easily noticeable, fraudulent transactions and money laundering transactions can be strictly connected, e.g. a criminal can commit fraud to finance criminal activities further. From now on, the term *illicit transaction* is used to generalise both of the anomalous types of transactions just described.

In the financial industry, illicit transactions cannot easily be detected by only looking at the features of single transactions in isolation, such as the exchange of an amount of money. It is crucial to analyse every single transaction together within its context, for example, taking into account the sender and the receiver or other transactions with similar characteristics. In other words, suspicious transaction patterns can only be detected by applying a broader perspective and therefore, by looking at transactions together with their contexts. For these reasons, some of the state-of-the-art methods

for anomaly detection process the financial data represented as graphs [3] [4]. Graphs constitute an enriched data representation schema able to capture the interdependent nature of money transactions and their contexts by incorporating both structural and topological information [5]. For instance, by using graph representation, it would be possible to model a transaction as a directed edge establishing a connection among two nodes: the sender and the receiver nodes.

1.1 Background

Leveraging graph structure information using traditional approaches in anomaly detection is not a straightforward task. Currently, most of the machine learning algorithms available in the literature leverage features represented in vector spaces. They are not able to directly digest data represented as graphs since graph representation, such as the graph adjacency matrix or similar representation, are non-Euclidean, sparse and high-dimensional data structures [6].

In order to overcome this problem, it is possible to extract graph statistics, such as triangle counts or node degrees, or extract ad-hoc features to accomplish the considered machine learning task. However, the drawbacks of such hand-crafted feature extractions are the computationally intensive pre-processing steps, especially considering real-world graphs and the absence of generalizability of such methods for other different graphs due to the data-specific execution of the pre-processing step. For these reasons, a valid alternative to the feature extraction step is employment of network representation learning (NRL) approaches which can learn how to encode graphs into low-dimensional vector spaces, i.e. graph embeddings [7].

The network representation learning literature is flourishing; the research is driven by the intent of improving the quality of embeddings and their ability to encode structural information of graphs to solve downstream machine learning applications better. In particular, to demonstrate the validity of a model, graph-specific machine learning applications are used as a benchmark considering the prediction performance. The most common are:

- Node Classification - predict the type of networks' nodes
- Link Prediction - predict the probability of observing a link among two nodes

- Community Detection - find densely connected groups of nodes
- Network Similarity - measure the similarity among (sub)networks

The most traditional techniques use Graph Factorization [8][9][10][11] or Random Walks [6][12][13] to achieve these goals, but a newer study area is emerging that leverages neural networks with satisfactory results in real-world applications. The work herein concentrates mostly on this latest area [14].

1.2 Problem definition

Although network representation learning represents a thriving research area, the current approaches available in the literature are not able to satisfy all the requirements needed to generate graph embeddings that can be used as input to real-world anomaly detection application in the financial industry.

Indeed, anomaly detection, when applied to financial data, poses several challenges due to the characteristics of the input graph data set, which are further complicated due to the regulatory sensitivity of the application domain.

The most challenging aspect which must be taken into account is the fact that financial transaction graphs are inherently *dynamic*. Every day new transactions are executed meaning that there is a continuous edge addition as well as new users leading to corresponding new nodes in the graph. In the literature, however, the assumption made by many approaches is to have a single fixed graph as input. This dynamic nature requires the NRL model to be able to generate embeddings which can represent new incoming graph entities [15], meaning that the model should be *inductive* to be able to generalize as adequately on unseen data. Along with the assumption that the graph is continuously increasing in order and size, it is also reasonable to assume that the system evolution is driven by certain dynamics, even if such dynamics are not explicit. Therefore, another aspect to address is how to train models to incorporate the system dynamics better; for instance, the prediction performance of a fixed model that does not take into account the system evolution can decrease over time.

Another characteristic of real-world networks is the *heterogeneity* of the nodes. It is not always the case that nodes can be considered as standard entities which share similar properties. Instead, it is more likely to have different types of nodes with

specific properties. For example, from a bank perspective, customers can be modelled as standard entities, but modelling nodes representing accounts held by other banks is not easily possible in the same way since the information for these latter node types is incomplete and out of the bank's reach. In other words, the bank will only be aware of transactions of external accounts which involve the bank's customers, but will not have a complete overview of the entirety of the transaction graph.

Due to the significance of the application domain, the desired property of the NRL model is the ability to generate embeddings in real-time, or at least in near real-time, which can be exploited to perform anomaly detection to detect deviating patterns as soon as possible and prevent many consequences of malicious behaviours. The trade-off among prediction performance and computational complexity is crucial.

The last but not least aspect to consider regards the dimensionality of data. When dealing with real-world data, it is also necessary to consider that the implemented solution should be scalable and efficient to process entire data volumes and correctly allocate resources.

1.3 Research Questions

The main objective of this thesis is to further investigate network representation learning approaches in order to design a model suitable for generating embeddings for real-world financial transaction graphs that can be utilized within an anomaly detection framework. To accomplish this objective, the thesis addresses the following questions:

- Among the existing network representation learning approaches available in the literature, which solutions are best suited for financial transaction data?
- Can graph embeddings improve the prediction performance in anomaly detection over the traditional statistical methods used in finance?
- Is it possible to leverage network representation learning to reflect the evolving nature of financial data?

1.4 Contributions

This thesis project illustrates the research performed in order to tailor a network representation learning model suitable to generate graph embeddings for real-world financial transaction graphs that can be further leveraged by downstream anomaly detection applications. The contributions can be summarized as follows:

- A qualitative analysis of the main techniques available in the literature to generate network embeddings.
  First of all, an overview of state-of-the-art NRL approaches is provided. Each framework in the overview is described in details, and its characteristics are summarised in the qualitative analysis. The qualitative analysis compares the frameworks according to the following criteria: the typology of input graph required, the inductive capacity of the model, the properties of the generated embeddings and the ability of the model to deal with dynamic graphs. These criteria reflect the requirements to handle financial transactions graphs. The qualitative analysis is used to identify the most suitable category of approaches for financial applications, which turns out to be ConvGNN [14], in particular GraphSAGE model [15]. To better understand the properties of the different NRL models, we have generated a synthetic bipartite graph with known characteristics, and it has been encoded into graph embeddings using some of the approaches described. Then, the different embeddings generated have been plotted in 2D space to visualize better how the different approaches reflect the characteristics of the synthetic input data.

- The design and implementation of a GraphSAGE variant meant to evolve accordingly together with the dynamic sequence of graph snapshots, which we call EvolveGraphSAGE-O.
  To evaluate this variant, we made use of the Elliptic Data Set, which includes the transaction data of the Bitcoin Network. This data set is a perfect example of a real-world data set that includes malicious nodes representing the minority class. Our results first demonstrate that via the use of GraphSAGE, it is possible to achieve comparable results with other NRL models presented in this work in terms of the prediction accuracy of the anomalous class. But, GraphSAGE has the advantage of being a more scalable algorithm that does not require to have an input graph with a fixed order. Then, it is also demonstrated that

EvolveGraphSAGE-O, can be trained to leverage recurrent neural networks with the intent of capturing the system dynamics and let the network representation learning model evolve within the input graph. In this case, the accuracy of the predictions decreases compared to the static version, but this dynamic version is more stable in case of concept drift.

- The Design and implementation of a second GraphSAGE variant meant to handle bipartite graphs such as the data provided by Swedbank AB, which includes the transactions of a subset of bank customers collected from January to April 2020. From this perspective, the aim is to detect customers who might have been forced to remain abroad for a long period due to lockdown measures applied to face the COVID-19 pandemic. It is finally demonstrated that the usage of embeddings could facilitate the real-time detection of these customers rather than using only node features. Still, the solution needs further improvement to obtain a satisfactory accuracy.

1.5 Research Methodology

The detailed description of research methods and methodologies presented by Håkansson [16] has been used as a guideline to better structure the degree project's research.

The research methodology of this thesis work consists of a *Case Study*, in fact, the aim is to further investigate if leveraging graph structure information is beneficial to perform anomaly detection in the financial industry.

In the first part of the thesis, an *Analytical Research* method is applied to perform a qualitative analysis of the existing network representation learning approaches in the literature and understand if they meet the requirements in the application domain. The result of this first research section is used to guide the decision-making process in identifying the most suitable NRL model and understand, where needed, the necessary modifications needed to satisfy the requirements.

For the second part of the research, an *Empirical Research Method* is applied. In fact, in this part, the prediction performance of the proposed solutions are quantitatively assessed with a particular focus the accuracy of prediction of the anomalous class, and, where possible, the performances are compared with baseline methods.

1.6 Delimitations

The main delimitation in this work is the fact that the research focuses on the investigation of how graph embeddings can be used for anomaly detection. Hence embeddings are evaluated only on the basis on how they impact the performance of the anomaly detection applications.

Another aspect is that anomalous labels are assigned to nodes, which means that anomaly detection takes place on the node level rather than among edges. Hence, this work concentrates in encoding nodes into low-dimensional vectors, which are specifically called node embeddings, and further analysing them to perform node classification. As a consequence, the proposed models are going to be evaluated considering node classification tasks rather than other graph-specific machine learning tasks, such as link prediction or edge classification, among others.

Furthermore, this research addresses real-world anomaly detection data. This means that the considered data volumes are unbalanced, i.e. the presence of anomalous class is limited compared to the non-anomalous class, and the quality of the labels is no entirely perfect since the labels are assigned using a heuristics approach.

1.7 Benefits, Ethics and Sustainability

The aim of this degree project is to further investigate if network representation learning can be used to improve the accuracy of anomaly detection applications in the financial industry by leveraging graph structural information.

Even if this degree project represents a preliminary investigation, the potential impact and implications of this work are illustrated to raise the awareness.

This work could potentially bring a lot of benefits since improving the performance of anomaly detection in the financial industry could contribute positively to the fight against financial crime. In fact, financial fraud and money-laundering represent severe harm for the whole society, and they have a substantial impact also on the stability and the reputation of the financial sector.

In addition, there have been further ethical issues to consider. Most importantly, the action of labelling a sample as anomalous in practice means suspecting an account holder of performing illicit actions. Moreover, financial records constitute very

sensitive data for analysis. Hence, it is imperative to guarantee the total anonymity of financial records. In particular, there is no information, personal or non-personal, that could in any way be used to reveal identities.

Sustainability represents an important criterion to consider when designing and implementing a model. In particular, when using approaches based on computationally expensive techniques, the training costs must be taken into account and wisely planned.

1.8 Stakeholders

This thesis project has been proposed and supervised by Swedbank AB in collaboration with the Research Institute of Sweden AB (RISE). Swedbank AB is one of the leading banks in Sweden and the Baltic countries, with about 7.3 million private customers and about 615,000 corporate customers. Its mission is to help people and businesses achieve solid financial sustainability. RISE is a public research institute, which plays a key role in driving sustainable development and growth at an international scale which works closely with universities, industry and public sector. The thesis work can be positioned within the *Continuous Deep Analytics* project conducted by the Research Institute of Sweden and the daily work of the Swedbank Analytics and Artificial Intelligence (A&AI) team.

1.9 Outline

The whole document is organized in the following chapters as listed below:

- Chapter 2 - A comprehensive overview of Network Representation Learning is provided, starting from factorization-based and random-walk based methods to the most recent Graph Neural Networks. Then, the approaches described in the overview are compared in a qualitative analysis.
- Chapter 3 - It is presented the taxonomy of graph-based Anomaly Detection methods, and for each category are provided with some examples. Then, are presented auxiliary models which are going to be used to perform Anomaly Detection in the Experiments.
- Chapter 4 - Description of the two Use Case considered: Anti-Money Laundering

in the Bitcoin Network and detection of bank customers in distress during COVID-19 pandemic situation.

- Chapter 5 - The Experiment conducted on the synthetic data set is illustrated. Then, the graph embeddings generator model selected for this work is presented in details; in particular, it is explained how it has been implemented and adapted to fit the two Use Cases.
- Chapter 6 - Presentation of the results achieved for the two Use Cases in terms of prediction performance and fulfilment of requirements, and when possible, the results are compared to previous works available in the literature.
- Chapter 7 - Conclusions of the project summarising the thesis contributions and leading the way for future work and possible improvements.

Chapter 2

Network Representation Learning

Graphs are very commonly used data structures able to model several real-world systems, such us economic networks, the Internet, social networks, and biomedical networks. They provide a level of abstraction suitable for describing complex data belonging to different fields, and to capture intrinsic relationships present within the data. Any real-world network can be modelled as a graph G defined as a tuple of two sets: the first set V consists of vertices (nodes), and the second set E consists of pairs of vertices called edges (links), representing the connection among two vertices.

$$G = (V, E) \tag{2.1}$$

Graph structures represent valuable information as they incorporate existing relationships among vertices. Unfortunately, this type of information cannot be easily exploited by traditional Machine learning techniques, which are mostly designed to analyse feature vectors while graphs data structures are non-Euclidean, sparse and typically with a high number of dimensions [7]. To overcome this problem, traditional approaches perform a feature engineering step and extract graph statistics from the network, e.g. node degrees and counts of triangle participation, among others. A valid alternative to extraction of graph statistics is currently offered by NRL approaches since they are meant to learn how to automatically encode graph structures into low-dimensional representations, namely graph embeddings. A Network Representation Learning model consists of a mapping function Φ that projects the input graph into a low d-dimensional space R^d.

It is possible to discriminate among different embeddings according to the input given to the mapping function Φ, where the model learns how to generate a graph embedding, a node embedding and a link embedding by encoding the entire graph, a particular node and a particular edge, respectively.

$$\begin{aligned} \textit{graph embedding } \Phi &: G \rightarrow R^d \\ \textit{node embedding } \Phi &: v \in V \rightarrow R^d \\ \textit{link embedding } \Phi &: e(u,v) \rightarrow R^d \end{aligned} \tag{2.2}$$

This thesis work focuses on the investigation of Network Representation Learning models able to generate node embeddings since the aim is to detect suspicious nodes by analyzing their embeddings. In the following, a detailed overview of Network Representation Learning techniques available in the literature is provided, from shallow dimensionality reduction approaches to Deep Learning ones. These techniques will be further discussed in Section 2.4 in order to understand whether they meet the requirements of financial transactions graphs.

Before presenting state-of-the-art Network Representation Learning models available in the literature, it is necessary to clearly state which are the requirements that the model has to satisfy for this particular application domain. First of all, the typology of input graph given to the Network Representation Learning model must be considered since real-world networks can not always be modelled as a graph of homogeneous nodes. For example, the transactional data provided by Swedbank AB can be modelled as a bipartite graph, see Section 4.2. In fact, nodes can be divided into two heterogeneous node sets, and they do not have any direct connection with other nodes in the same node set. For this reason, this work focuses as well on models that can also handle heterogeneous nodes. In specific, in this work, we are interested in models able to generalize on bipartite graphs. Secondly, the studied graphs are attributed, which means that nodes have features. For example, for each customer, some demographic information is available and it is also possible to extract some additional information regarding past transactions. Another fundamental characteristic to consider is the fact that the graph is inherently dynamic, hence inductive frameworks are more appropriate since they can generalize on unseen data. As well, the dimensions of the graph play a fundamental role in driving the selection of the approach to be applied.

The number of customers is of the order of millions and the number of transaction points is even larger, in the order of billions. Therefore, generation of the embeddings should be as fast and scalable as possible. In particular, due to the sensitive application domain of the industry, the inference should be online. The model should be able to generate embeddings for some target nodes as soon as new information has been made available. The last prerequisites are in regards to the intent of the generation of such embeddings. From the stakeholder perspective, the main interest concerns node embeddings since the aim is to generate additional features which incorporate the graph structure to augment their model and increase predictive performances. This means that the embeddings generated should be suitable to feed downstream Machine Learning applications.

To recap, the requirements have been summarized in the following Table 2.1, which is going to be used as a reference when comparing the models.

Table 2.1: Use Case Requirements

Requirements	Explanation
Bipartite Graph	The model should be able to take into account the specific characteristics of bipartite graphs when performing the encoding
Attributed Graph	The model should be able to exploit available node features
Dynamic Graph	The model should be inductive, i.e. able to generalize on unseen data, since the graph is dynamic and may evolve over time, e.g. via node addition
Online Inference	Fast and scalable generation of node embeddings as soon as information on new transactions arrive in the stream
Embeddings as *features*	The nodes embeddings should be suitable to be fed into downstream Machine Learning applications

2.1 Factorization-based approaches

This family of approaches relays on matrix factorization techniques, also called *matrix decomposition*, which are used in several Machine Learning applications to reduce the

input volume and high-dimensional matrices into constituent parts. The underlying idea of such methodologies is to decompose a matrix to simplify further processing by considering only the smallest constituent parts of the original matrix. The input matrix should represent the graph and the desired graph properties which need to be encoded. For example, by using the graph Laplacian eigenmaps, it is possible to extract embeddings which represent the pairwise node similarity, alternatively, it is possible to define a node similarity matrix that better reflects the application domain needs.

Singular Value Decomposition

An example of a well-known matrix decomposition technique is called Singular Value Decomposition (SVD) [8], which decomposes the original input matrix $A(m \times n)$ into the three following matrices:

- U Left Singular Vectors
- Σ Singular Values
- V Right Singular Vectors

SVD is closely related to Eigen decomposition, indeed the columns of U are the orthonormal eigenvectors of AA^T while the columns of V are the orthonormal eigenvectors of A^TA. Likewise, Σ is a diagonal matrix containing the square roots of eigenvalues from U or V in descending order. Dimension reduction can be achieved by considering the truncated SVD, taking only the first k columns of U and V, and the sub-matrix Σ_k. One of the most appreciable properties of this low-rank approximation is that SVD is optimal with respect to rank k, i.e. among all rank k matrices, the reconstructed matrix A_k formed by considering only the top k singular values in Σ has the smallest distance to the original input matrix A measured in any unitary invariant norm $||\cdot||_M$. This means that if we want to reduce dimensionality to k dimensions, A_k is the best solution (with associated basis vectors from the corresponding decomposition), which minimizes the reconstruction error. The main drawbacks of such methods are the time complexity, which is $O(mn \min(m, n))$ in the worst case and the fact that the entire matrix is required to compute the decomposition. As a result, these techniques work well in static and transductive scenarios.

HOPE

The paper on Higher Order Proximity preserved Embedding (HOPE) [9] illustrates

the comparative study done by the authors in order to identify which higher-order similarity measures are optimal to generate embeddings which preserve the asymmetric transitivity in directed graphs. They have conducted their experiments by generating different similarity matrices with different high-order proximity measures, see Table 2.1.1, and then they have applied a generalized version of SVD to generate embeddings.

Table 2.1.1: HOPE: Higher-order Proximity Measures

Proximity Measures	Similarity Matrix	Description
Katz	$S^{Katz} = (I - \beta A)^{-1}\beta A$	the Katz similarity between two vertices corresponds to the weighted sum of the paths which connect these two vertices, the path weight decays exponentially with its length with a decay parameter β
RPR	$S^{RPR} = (1 - \alpha)(I - \alpha P)^{-1}$	the RPR similarity between two vertices corresponds to the probability that a random walk starting from one vertex will reach the other in the steady state
CN	$S^{CN} = A^2$	the CN similarity between two vertices corresponds to the counts of vertices connected to both these two vertices
AA	$S^{AA} = ADA$	the AA similarity between two vertices is very similar to the CC similarity with the only difference being that the count of vertices is weighted according to the vertices' degrees

The quality of embeddings has been evaluated in terms of minimization of the reconstruction error using the Root Mean Squared Error (RMSE) and the precision@k for link prediction.

RolX

RolX [10] is an unsupervised learning framework able to generate node embeddings which represent structural roles of the input graph. As a framework, RolX presents a strategy to extract embeddings using matrix decomposition but it is not bound to a particular matrix decomposition technique. Furthermore, it is different since it does not directly require a matrix representation of the graph as input but instead performs feature extraction. This framework is structured in three main components:

- Feature Extraction: the first step consists of extracting relevant features from the graph. The authors propose to use a recursive structural feature discovery algorithm [17] which extracts for each node its local and egonet features. The output of this feature engineering step is a node-feature matrix $V(n \times f)$ since we have n vertices and f features.
- Feature Grouping: The successive step is to reduce the dimensionality of the previous output V using a rank r approximation $GF \approx V$ in order to obtain the matrix $G(n \times r)$, in which each cell represents n$^{\text{th}}$ node memberships in the r$^{\text{th}}$ role, and the matrix $F(r \times f)$, in which each cell represents how membership in the r$^{\text{th}}$ role contributes to estimating the f$^{\text{th}}$ feature value. Feature grouping can for example be performed using Singular Value Decomposition, but the authors used Non-Negative Matrix Factorization [18] in their implementation.
- Model Selection: This last step is used to learn in an automatic way which is the best value for the r parameter in order to preserve as much information as possible within the compressed features but at the same time reduce the necessary memory to store such information. Therefore, the authors decided to apply the Minimum Description Length criterion [19] to automate the model selection which defines the best model as the one which is able to minimize the number of bits required to represent the model and the reconstruction loss.

DANE

The previous techniques demonstrate how it is possible to enrich the information included within the embeddings by defining a similarity matrix at will or by generating a features matrix. Anyhow, it will be interesting to exploit both similarity matrices and node features to encode attributed network. The Dynamic Attributed Network Embedding (DANE) framework [11] is a representative example which aims to

capture evolving patterns of attributed networks. The framework generates a low-dimensional representation that encodes the correlation among node features and the underlying network structure, which can be updated in online manner. Therefore, the framework comprises an offline and online model. The offline model encodes both the graph adjacency matrix and the node features matrix using the Laplacian eigenmaps technique. These generated latent representations are then merged to obtain the final embeddings. The online model incorporates the perturbation in network structure and node attributes which happens in two consecutive time steps. Under the assumption that the graph evolves smoothly within consecutive time steps, the framework DANE exploits first-order matrix perturbation theory [20] to generate in near real time an approximation of the embeddings which reflects the updated graph.

2.2 Random walk-based approaches

The algorithms belonging to this category get inspiration from a technique used in the Natural Language Processing domain, namely word embeddings. These techniques are used to map words in a vector space maintaining the correlation of words within their semantic context. The term ”semantic context” is simply defined as the other surrounding words which are farthest from the target word as specified by a window size parameter. The underlying assumption of this approach is that words which do belong to the same context are likely to be encoded into similar embeddings.

Skip-gram model

The core component of the following techniques is represented by the Skip-gram model [21]. It consists of a neural network with a single hidden layer and multiple output layers that adopt softmax activation functions. There is exactly one output layer for each word in the context; for example in figure 2.2.1 we can see that the model architecture considers a window size equal to two, hence the input word $w(t)$ is surrounded by four words: two for both context sides.

The task used to train the network consists of the prediction of the word context given an input word. This can be considered as an auxiliary task since we are not going to use the trained network to predict the context. Instead, this auxiliary task is used to learn the hidden layer weights that are actually the word embeddings W. For this reason, the training happens in an unsupervised fashion, and the input consists of the text corpus

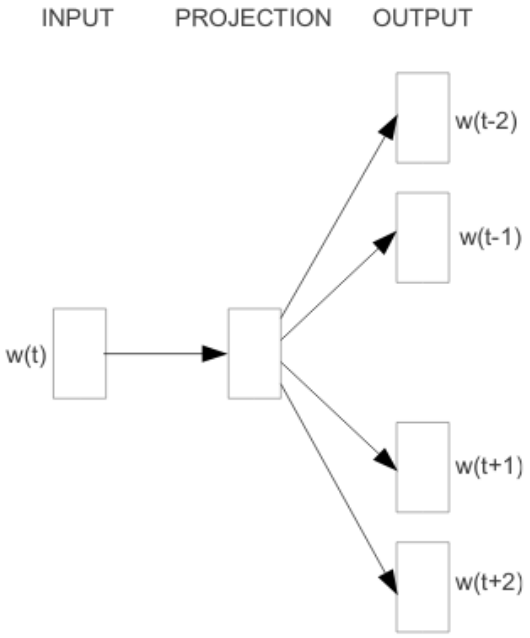


Figure 2.2.1: Skip-gram model architecture as presented by the authors [21]

from which the training samples are extracted.

In addition to the window size, the parameters which need to be defined are the embedding size N and the vocabulary size V, i.e. the number of words considered. These two parameters define the dimension of the neural network weights: the hidden weight matrix W with size $V \times N$ and the shared output weight matrix W' with size $N \times V$.

Of course, words need to be pre-processed before being fed into the neural network, so one-hot encoding is used. Each word is transformed in its one-hot encoded representation, i.e. a binary vector of size $V \times 1$ that has all zeros other except for one value in the ith position, which is also the position of the considered word in the vocabulary. So starting from the one-hot representation of the input word x_i, it is possible to compute its hidden representation h_i applying this formula:

$$h_i = x_i^T W \tag{2.3}$$

which is exactly the ith row of the hidden weight matrix W. And from the hidden representation it will be possible to compute the final vector u:

$$u_i = W'^T h_i. \tag{2.4}$$

Consequently, the softmax activation function is applied to each j[th] element of such final vector u_i:

$$u_{i,j} = \frac{e^{u_{i,j}}}{\sum_{j' \in V} e^{u_{i,j'}}} \tag{2.5}$$

Each j[th] element $u_{i,j}$ of the final vector u_i represents the probability of having the j[th] word in the context of the i[th] word.

The model is trained in order to maximize the probability of words which are in the context. The usage of cross entropy as loss function makes the optimization problem very demanding from a computational perspective since there is a need to iterate all over the vocabulary set at each stochastic gradient descent step.

$$lossfunction = \sum_{i \in V} \sum_{j \in context(i)} -log\left(\frac{e^{u_{i,j}}}{\sum_{j' \in V} e^{u_{i,j'}}}\right) \tag{2.6}$$

To alleviate the training burden, a technique called negative sampling [22] has been adopted. Instead of computing all the probabilities of having a word in a particular context, the objective is to train the network for distinguishing the input word from other k words drawn from a random distribution over all nodes P_V; these sampled words are called negative samples, hence the name.

$$log\left(\frac{e^{u_{i,j}}}{\sum_{j' \in V} e^{u_{i,j'}}}\right) \approx log(\sigma(u_{i,j})) - \sum_{l=1}^{k} log(\sigma(u_{i,n_l})), n_l \sim P_V \tag{2.7}$$

The Skip-gram model can also be applied to graphs by generalizing the concepts of word and its context. In fact, words can be considered as the entities to be encoded into a low-dimensional space, and their context is formed by entities associated with or similar to the target entity. In the graph domain, such entities are representative of nodes and their contexts can be extracted by performing a random walk, for example. The resulting node embeddings are structured and compact latent representations able to capture network topology. This family of approaches is particularly suitable for encoding **homophily** networks since the model is trained to encode nodes within the same context similarly. In other words the network connections should represent a similarity among the nodes which are connected.

DeepWalk

The application of the Skip-gram model in the graph domain was first presented in the

DeepWalk algorithm [6]. The main components of DeepWalk consist of the random walk generator and the Skip-gram neural network. The random walk generator takes as input the entire graph G, and it generates γ random walks of fixed length t for each node by recursively sampling a neighbour of the latest visited node in the walk until the length t is reached. Once the truncated random walks, also known as vertices *corpus*, are generated, they are used to train the Skip-gram model to maximize the probability of having the context of vertex v_i given its encoded representation $\Phi(v_i)$ as defined in the following formula 2.8 which is very similar to the loss function 2.6.

$$\min_{\Phi} -\log Pr(\{v_{i-w}, ..., v_{i+w}\} \setminus v_i | \Phi(v_i)) \tag{2.8}$$

Indeed, the mapping function $\Phi : c \in V \rightarrow R^{|V| \times d}$, as defined by the paper authors, represents the encoded structural information for each vertex $\in V$ and it consists exactly in the weight matrix learned by the neural network W as described in the previous paragraph.

node2vec

Random walk-based approaches are very flexible since they can learn how to encode different structural properties from graphs by simply changing the vertices of the corpus given as input. Indeed, the literature regarding random walk-based approaches explores how to extract different structural properties by biasing the random walk to obtain novel and informative node contexts. The node2vec [12] algorithm, for example, illustrates how to bias the random walks using two parameters p and q that can trade-off between a breadth-first and depth-first neighbourhood exploration.

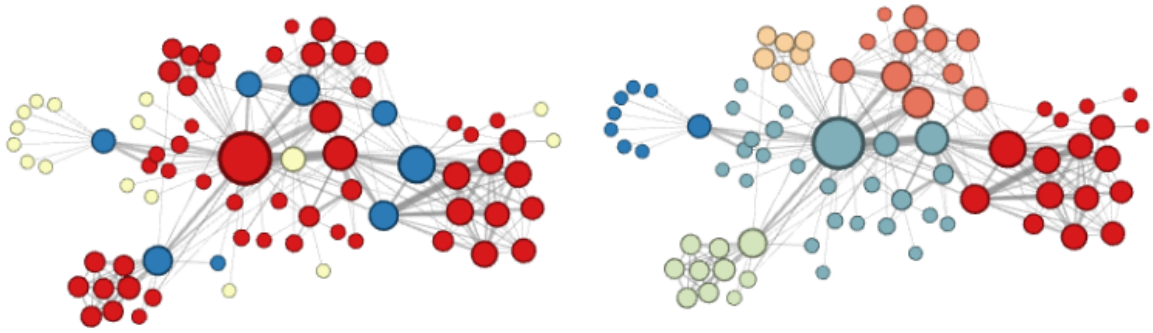
- The return parametThe application of the Skip-gram model in the graph domain was first presented in the DeepWalk algorithm [6]. The main components of DeepWalk consist of the random walk generator and the Skip-gram neural network. The random walk generator takes as input the entire graph G, and it generates γ random walks of fixed length t for each node by recursively sampling a neighbour of the latest visited node in the walk until the length t is reached. Once the truncated random walks, also known as vertices *corpus*, are generated, they are used to train the Skip-gram model to maximize the probability of having the context of vertex v_i given its encoded representation $\Phi(v_i)$ as defined in the following formula 2.8 which is very similar to the loss function 2.6.er p influences

the probability of revisiting a node in the walk, the lower its value the higher the probability of re-sampling immediately an already-visited node leading to a breadth-first exploration.

- The in-out parameter q influences the probability of visiting nodes which are further from the first-hop neighbourhood of the already-visited node, the lower its value the higher the probability to explore in-depth the neighbourhood.

According to the case study performed by the authors [12] on the character network of *Les Misérables* novel, the algorithm can extract node embeddings based on both structural equivalence and homophily. The authors performed clustering on the node embeddings computed with two different parameter combinations learned, and they visualized the 2D network assigning colours to nodes based on the clusters.

In Figure 2.2.2a it is possible to notice how node embeddings extracted with a depth-first sampling strategy are representative of the structural equivalence of nodes. For instance, the characters which act as bridges among other characters have been clearly identified in the blue cluster. In Figure 2.2.2b instead, it is possible to notice how node embeddings extracted with a depth-first sampling strategy are better able to discover clusters of characters which are closely connected among each other that reflects more the homophily.



(a) p = 1, q = 2

(b) p = 1, q = 0.5

Figure 2.2.2: node2vec node classification results using different parameters as presented by the authors [12]

BiNE

The Bipartite Network Embedding (BiNE) algorithm is a particular example of skip-gram based models specifically meant for bipartite graphs [13]. The previous methods can still be applied to bipartite graphs by modelling them as a normal graph, but this

generalization leads to a loss of information. In fact, those methods fail in leveraging the distinguishing structures of bipartite graphs: they do not consider that the nodes of the two sets are heterogeneous, and consequently, the corpus generated by a random walk is not representative of the specific relationships among the nodes of these two sets. In bipartite graphs, there is no direct connection among nodes of the same category. Anyhow, nodes within the same sets are related even if indirectly, and it would be better to exploit even these implicit relations. To address this problem, the authors proposed a joint optimization framework to model the explicit relationships, which connect the two disjoint sets, and the implicit relationships, which connect the nodes within the same set. The modelling of the explicit relationships is inspired by the preservation of first-order proximity defined in the Large-scale Information Network Embedding (LINE) paper [23].

First-order Proximity: The first-order proximity in a network is the local pairwise proximity between two vertices. For each pair of vertices linked by an edge (u, v), the weight on that edge, w_{uv}, indicates the first-order proximity between u and v. If no edge is observed between u and v, their first-order proximity is 0. [23]

To preserve the first-order proximity, the difference between the empirical distribution of vertex co-occurring probability and the reconstructed distribution by the vertex embeddings is minimised, defined in the following formulas:

$$P(i,j) = \frac{w_{ij}}{\sum_{e_{st} \in E} w_{st}} \qquad \hat{P}(i,j) = \frac{1}{1 + exp(-\vec{u}_i^T \vec{v}_j)}$$

The KL-divergence is used as a metric to compare the two distributions.
DeepWalk [6] has inspired the modelling of implicit relationships. However, unlike DeepWalk, BiNE performs random walks on an induced homogeneous graph, which models the second-order proximity among nodes. In this way, there are no periodicity issues and the node sequence generated corpus contains only nodes of the same homogeneous set. The induced homogeneous graphs can be defined as:

$$W^U = [w_{ij}^U] \qquad W^V = [w_{ij}^V]$$

where w_{ij}^U and w_{ij}^V represents the second order proximity among two nodes.

$$w_{ij}^U = \sum_{k \in V} [w_{ik} w_{jk}] \qquad w_{ij}^V = \sum_{k \in U} [w_{ki} w_{kj}]$$

Moreover, random walks are neither of a fixed-length nor of the same number for each node in order to better represent the underlying vertex distribution in bipartite networks. The number of random walks generated for each node is proportional to the importance of the node itself, which can be computed with a centrality metric such as Hyperlink-Induced Topic Search (HITS). The higher the centrality score, the higher the number of random walks generated starting from that particular node. To avoid having fixed size-length *corpus*, the random walk generator considers a probability p to stop the sampling.

2.3 Graph Neural Networks

The previous section illustrated how it is possible to extract *corpora of vertices* from graphs, which can be fed into a particular type of neural network, the skip-gram model. Instead, the techniques presented in this section belong to a new research field that aims to adapt neural networks to directly apply deep learning models on graphs, namely Graph Neural Networks (GNNs). The GNNs research is strictly related to NRL, since many GNNs extract hidden representations from graphs to perform graph-specific Machine Learning tasks, and such embeddings can, as a result, be extracted. An extensive survey [14] in this area proposes a taxonomy to distinguish among the current Graph Neural Networks available in the literature.

- Recurrent Graph Neural Networks (RecGNNs)
  Most of the models within this category represent pioneer work which has led the way for applying other neural network frameworks to graphs. They are based on the assumption of messages passing among graph vertices, the underlying idea is that vertices are in constant communication with their neighbours until an equilibrium is reached.

- Graph Convolutional Networks (ConvGNNs)
  Convolutional Neural Networks (CNNs) have been successfully applied in the computer vision domain since they are particularly suitable to extract compact

representation from images which are grid-like data structure, i.e. matrices of pixels. In order to apply convolutional layers to non-grid like data structures such as graphs, the convolutional operation 2.3.1 has been generalized by relaxing the assumptions that the neighbourhood of a particular node is not of a fixed size and not even ordered, differently from what happens with a pixel matrix in which every pixel has a regular neighbourhood determined by the filter size.

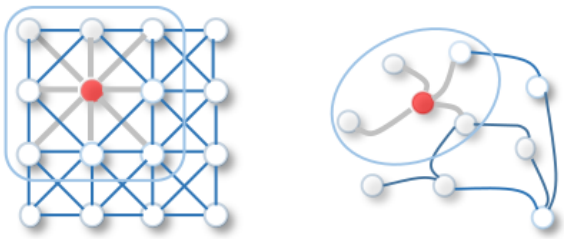


Figure 2.3.1: 2D Convolution compared to Graph Convolution [14]

- Graph Autoencoders (GAEs)
  Graph autoencoders are unsupervised frameworks able to encode graphs into low-dimensional representations. Like traditional autoencoders, the graph autoencoders architecture comprise an encoder and a decoder network. The encoder network encodes the input graph into a latent representation, while the decoder reconstructs the graph data. In order to learn graph embeddings, the autoencoders can be trained to reconstruct the graph structural information, e.g. the adjacency matrix A. Alternatively, they can be trained to learn the graph generative distributions making them particularly suitable for applications that require data augmentation.

- Spatial-temporal Graph Neural Networks (STGNNs)
  This last category of Graph Neural Networks aims to learn patterns by modelling both spatial and temporal dependencies in dynamic graphs. Most of them rely on ConvGNNs to capture the spatial dependency among nodes, while the time correlation is modelled using Recurrent Neural Networks (RNNs) and CNNs.

For the scope of this degree project, the most interesting Graph Neural Networks are ConvGNNs and GAEs. These two types are explained in details in the rest of this section. In addition, an example of Spatial-temporal Graph Neural Network (STGNN) is illustrated which leverages a ConvGNN model.

2.3.1 Convolutional Graph Neural Networks

This category of Graph Neural Networks is called convolutional since they are based on convolutional layers. These layers comprise filters that are generally applied over all the locations in the graph, or over particular graph subsets [24], similarly to traditional CNNs. Among this category, it is possible to distinguish among two different approaches: spectral-based and spatial-based ConvGNNs. The former approaches are based on graph signal processing, and the convolutional operation applied is meant to remove noise from the graph input. They are called spectral convolution since they perform Eigen decomposition of the Laplacian matrix formula 2.9 in order to capture the underlying graph structure, and identify clusters of nodes of the graphs in the *Fourier* space.

$$\begin{aligned} \textit{Laplacian Matrix } L &= I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \\ \textit{with } D_{ii} &= \sum_j (A_{ij}) \end{aligned} \tag{2.9}$$

In other words, the input graph signals are projected in the orthonormal space whose bases are defined by the eigenvectors of the Laplacian matrix and then the graph convolution is applied.

The latter approaches, instead, are inspired by RecGNNs and the convolutional operation is based on the concept of message diffusion also called message propagation. Indeed, the convolutional operation aggregates the information of the nodes' local neighbourhoods and encodes it in a latent space. Unlike computationally expensive spectral convolution, spatial convolutions are efficient and flexible, and nonetheless, they have achieved state-of-the-art results on several graph-specific machine learning tasks. [14]

GCN

Among Spectral Convolutional Networks, the Graph Convolutional Network (GCN) algorithm proposed in this paper [25] is particularly relevant since it bridges the gap among spatial and spectral based approaches. Indeed, GCN has paved the way to spatial based graph convolution, and its architecture is a common standard of the most recent Graph Convolutional Networks available in the literature.

This common architecture consists of the necessary steps to learn a mapping function which produces a node-level output, i.e. node embeddings, starting from a representative description of the graph structure in matrix A.

The graph neural network layer can be generalized as the following non-linear function:

$$H(l+1) = f(H(l), A) \tag{2.10}$$

in particular, the input layer $H(0) = X$ consists in the input features/signals matrix $N \times F$, in which N is the number of nodes and F is the number of input features dimension, and the output layer $H(L) = Z$ of a ConvGNN having L layers consists in the output matrix $N \times D$ in which D is the number of output features per node. The specific ConvGNN model can differ in how $f(,)$ is defined.

The specific propagation rule introduced in [25] is the following:

$$\begin{aligned} f(H(l), A) &= \sigma(\hat{A}H(l)W(l)) \\ \textit{with } \tilde{A} = A + I_n, \tilde{D}_{ii} &= \sum_j (\tilde{A}_{ij}), \hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} \end{aligned} \tag{2.11}$$

where $W(l)$ is the weight matrix of the l^{th} and $\sigma()$ is a non-linear activation function, e.g. Rectified Linear Unit (ReLU). In the formula, they are using $\hat{A}$ rather than the adjacency matrix to guarantee that the convolution operation is applied to all the features vector of the node neighbours and of the node itself. Furthermore, $\hat{A}$ is symmetrically normalized to avoid that the multiplication with $\hat{A}$ completely changes the scale of the feature vectors.

Skip-GCN

Skip-Graph Convolutional Network (Skip-GCN) [26] is a GCN variant, indeed its convolution operation is very similar to Equation 2.11 but it is introduced a *skip connection* [27] among the input features X and the hidden representations $H(l)$ described in the following equation:

$$\begin{aligned} f(H(l), A, X) &= \sigma(\hat{A}H(l)W_1(l) + XW_2(l)) \\ \textit{with } \hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}, \tilde{A} &= A + I_n, \tilde{D}_{ii} = \sum_j (\tilde{A}_{ij}) \end{aligned} \tag{2.12}$$

This variant leverages skip connections to consider at every convolution the input node

features, which brakes the linear dependence among subsequent convolutional layers. Generally, this change leads to better prediction performances as explained in this work [27].

GraphSAGE

The Graph SAmple and aggreGatE (GraphSAGE) algorithm [15], namely GraphSAGE, is considered one of the state-of-art ConvGNN and it is insipired by the GCN [25] neural network framework. Similar ConvGNNs available in the literature have two main limitations: they require the entire graph as input for generating embeddings, e.g. the propagation rule 2.11 takes $\hat{A}$ as input, and they are not really capable of generalizing on unseen data, i.e. they are meant for transductive scenarios. Instead, GraphSAGE stands out for its ability to better suits many real-world applications. In particular, it can be used to create embeddings for very large graph since the forward propagation algorithm can be generalized to work in mini-batch settings without using the entire graph as input [28]. As well, GraphSAGE can generate embeddings for newly observed (sub)graphs since it is trained to learn how to aggregate input features from nodes' local neighbourhoods, rather than directly learn a low-dimensional representation for each node.

In order to better understand how GraphSAGE can work in inductive settings, here it is presented the forward propagation algorithm 1, which generates embeddings given a trained GraphSAGE model. Besides to graph and nodes features, as input are also required the depth L of the neural network, the neighbourhood function and the trained weights. The depth L defines the search depth in the neighbourhood, which corresponds to the number of hops done starting from the target node to aggregate the neighbourhood information, see Figure 2.3.2. The neighbourhood function is used to select the neighbours of the target node at each layer. By setting a limit S_l in the number of nodes which can be sampled by the neighbourhood function at each l^{th} layer it is possible to maintain time and space complexity of the algorithm constant.

The trained the parameters of the model are aggregator functions parameter denoted as $AGGREGATE_l$ and the matrices $W(l)$ used to propagate information.

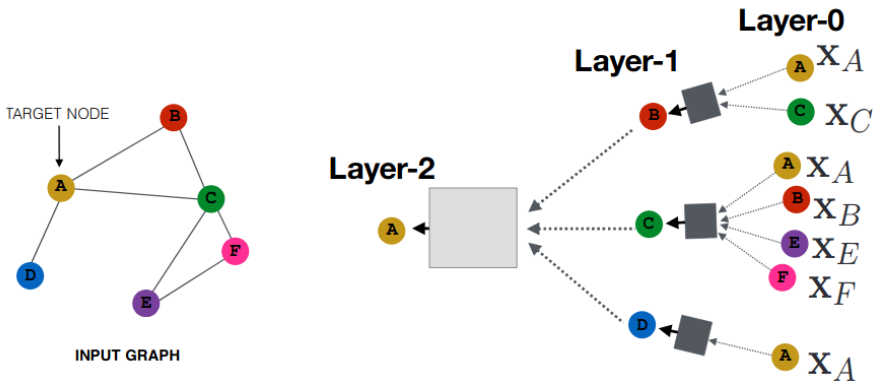


Figure 2.3.2: Computation graph for generating node embedding for a target node with a 2-layer GraphSAGE Network [29]

Algorithm 1 GraphSAGE forward propagation

Input: Graph $G = (V, E)$, input features $x_v, \forall v \in V$, depth L, weight matrices $W(l)$ $\forall l \in \{1, ..., L\}$, non-linearity σ, differentiable aggregation function $AGGREGATE_l$ $\forall l \in \{1, ..., L\}$, neighborhood function $N : v \rightarrow 2^V$

Output: Vector representations $z_v \; \forall \; v \in V$

1: $h_v^0 \leftarrow x_v, \forall v \in V$
2: **for** $l = 1...L$ **do**
3: **for** $v \in V$ **do**
4: $h_{N(v)}^l \leftarrow AGGREGATE_l \left(\{h_u^{l-1}, \forall u \in N(v)\} \right)$
5: $h_v^l \leftarrow \sigma \left(W(l) \cdot CONCAT \left(h_v^{l-1}, h_{N(v)}^l \right) \right)$
6: **end for**
7: $h_v^l \leftarrow \frac{h_v^l}{\|h_v^l\|_2}, \forall v \in V$
8: **end for**

To initialize the algorithm the node features are considered as initial node representations (algorithm 1: step 1). Then, for each layer l in the outer loop, the operations executed for each node are the following:

- aggregation of the node's neighbours representations into a single vector (algorithm 1: step 4)
- concatenation of the node's current representation with its aggregated neighbourhood vector obtained in the previous step and feeding it into a fully connected layer with nonlinear activation function σ (algorithm 1: step 5)
- normalization of the fully-connected layer output (algorithm 1: step 7)

The GraphSAGE aggregator function can be customized at will. Anyhow, the authors proposed three different functions: mean aggregator, LSTM aggregator and pooling aggregator. The simplest architecture consists in the mean operator which aggregates the neighbours' vectors by computing their element-wise mean as illustrated in the following formula which can replace the 4th algorithm 1 line:

$$h^l_{N(v)} \leftarrow MEAN\left(\{h^{l-1}_u, \forall u \in N(v)\}\right) \tag{2.13}$$

The pooling aggregator, instead, uses the neighbours' vectors as input to a fully connected layer before performing the concatenation, and then it applies an element-wise max-pooling operation. The 4th line can be replaced with the following:

$$h^l_{N(v)} \leftarrow max\left(\{\sigma W_{pool} h^l_{u_i} + b, \forall u_i \in N(v)\}\right) \tag{2.14}$$

The last aggregator described is based on a Long-Short-Term-Memory (LSTM) architecture, which can create more complex representations. The only disadvantage of this aggregator is the fact it is not permutation invariant, i.e. it is possible to obtain different embeddings according to the neighbours' vectors order in which they are fed into the LSTM architecture.

The forward propagation algorithm assumes that the parameters are already given, but of course, the network needs to be trained. It is possible to train GraphSAGE both in a supervised and unsupervised manner using stochastic gradient descent.

For the unsupervised training, the graph-based loss function to be optimized is inspired by Random-walk based approaches with negative sampling. Indeed, the loss function 2.15 encourages the embedding of a node z_u to be similar to the embedding of node z_v when they co-occur in random walk *corpus* and t be dissimilar to Q negative samples embeddings extracted from a negative sampling distribution P_n.

$$J_G(z_u) = -log(\sigma(z^T_u z_v)) - Q \cdot E_{v_n} \sim P_n(v) log(\sigma(-z^T_u z_{v_n})) \tag{2.15}$$

It is also possible to opt for supervised training in case the embeddings are meant to be used for a specific downstream Machine Learning task, such as network classification, and the labels are available. In this case, it is sufficient to add the necessary layers to the

GrapSAGE! (**GrapSAGE!**) neural network architecture and optimize a task-specific loss function, for example, cross-entropy for classification, to improve the embeddings expressiveness for further processing.

2.3.2 Graph Autoencoders

This category of GNNs is particularly suitable to learn graphs latent representations. The network comprises two complementary parts: the encoder and the decoder. These two parts are jointly trained to generate a low-dimensional representation that preserves as much as possible the information of the underlying data structures. The encoder generates the latent representation mapping the input graph in the low-dimensional space, i.e. it acts like the mapping function Φ. Instead, the decoder reconstructs the graph data representation starting from the latent representation. Graph autoencoders differ in the selection of the encoder network to handle different input graphs, and in the selection of the decoder network to consider different reconstruction errors.

SDNE

Structural Deep Network Embedding (SDNE) [30] consists of a stacked autoencoder architecture for learning embeddings which preserve structural information of the input graph by considering both first-order and second-order proximity. To preserve first-order proximity is applied a loss function on the embeddings, i.e. the encoder output, which encourages nodes within the same neighbourhood to have similar latent representations. On the other hand, the second-order proximity is preserved by learning how to reconstruct nodes neighbourhoods from the embeddings.

The previous approach uses simple multi-layer perceptrons as encoder and decoder networks, but it is also possible to generate embeddings which do not only consider topological information using different encoder/decoder architectures. In particular, using ConvGNNs as an encoder, it is possible to exploit as well as node features in case of attributed graphs.

GAE* - VGAE

Graph Autoencoder [1] (GAE*) is an example of graph autoencoder architecture which exploits ConvGNNs, in particular GCN [25], to generate embeddings of attributed graphs.

The GAE* network consists of: a 2-layer graph convolutional network as encoder $Z = GCN(X, A)$ that takes as an input the graph adjacency matrix A and the node features matrix $X(N \times F)$, which produces the node embeddings matrix Z, and a simple inner product as decoder $\hat{A} = \sigma(ZZ^T)$. The GAE* decoder objective is to reconstruct a graph adjacency matrix $\hat{A}$ faithful to the original by minimizing the negative cross-entropy among the input object and the reconstructed one.

The task of reconstructing the adjacency matrix could be too simplistic for autoencoders that have great reproduction capacity, this must be taken carefully into account because it could lead to overfitting. [14] The paper authors [1] proposed a solution to this problem by presenting a variational variant of the previously described architecture called Variational Graph Autoencoder (VGAE) inspired by variational autoencoders architecture. The underlying idea of this variational approach is to learn the data distribution parameters matrices μ and σ using a 2-layer GCN as defined in the following formula:

$$\begin{aligned} \mu &= GCN_{\mu}(X, A) \\ \sigma &= GCN_{\sigma}(X, A) \\ \textit{with } GCN(X, A) &= \tilde{A}ReLU(\tilde{A}XW_0)W_1 \textit{and } \tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \end{aligned} \tag{2.16}$$

Given
the trained parameters matrices, the encoder can infer the hidden representation z_v for each node v applying the following formula:

$$q(z_v|X, A) = N(z_v|\mu_v, diag(\sigma_v^2)) \tag{2.17}$$

and the decoder can reconstruct the adjacency matrix $\hat{A}$ using the generative model which consists in the inner product:

$$\begin{aligned} p(A|Z) &= \prod_{u \in V} \prod_{v \in V} p(A_{u,v}|z_u z_v) \\ \textit{with } p(A_{u,v} = 1|z_u z_v) &= \sigma(z_u^T z_v) \end{aligned} \tag{2.18}$$

The loss function optimized to learn the network weights is the variational lower bound L, which consists of two parts: the reconstruction error, and the latent variable restriction loss. The reconstruction loss is used to verify if the reconstructed adjacency

matrix is similar to the original one, while the restriction loss applies Kullback-Leibler (KL) divergence function to measure how similar are the distribution of the latent variables, and the learned normal distribution.

$$L = E_{q(Z|X,A)}[logp(A|Z)] - KL[q(Z|X,A)||p(Z)] \tag{2.19}$$

2.3.3 Spatial-Temporal Graph Neural Networks

This last category of Graph Neural Networks is particularly indicated to capture spatial-temporal dependencies in dynamic graphs. In particular to perform a predictive task such us forecasting nodes labels it is necessary to consider not only existing relationships among graph entities but also how such graph entities have evolved over time.

Evolve-GCN

The GCN ConvGNN model has been used as a base in this paper [26] to create a dynamic version of it called Evolve-Graph Convolutional Network (Evolve-GCN) [26], to capture both spatial and temporal patterns in dynamic graphs. Evolve-GCN can be considered as a STGNN since the underlying idea is to exploit RNNs and their ability to model the temporal dynamics to learn GCN weights. In this way, rather than training directly a single GCN model, the Recurrent Neural Network (RNN) architecture is used to perform a time-dependent model adaption of GCN by regulating its network parameters accordingly with the considered timestep.

The authors proposed two different techniques to learn how to evolve the weights of the network. The first evolution strategy leverages a Gated Recurrent Unit (GRU) [31], a particular type of RNN architecture, to learn how to update the GCN weights. This version is called Evolve-GCN-H, where H stands for hidden, since the GCN weights are treated as the hidden state of the GRU, and the node embeddings as the input of the GRU.

$$W_t^{(l)} = GRU(H_t^{(l)}, W_{t-1}^{(l)}) \tag{2.20}$$

Instead, the second evolution strategy leverages a LSTM [32] to learn how to update

the GCN weights. This version is called Evolve-GCN-O, where O stands for output, since the GCN weights are modelled as the output of this particular RNN, and it takes as input the weights at the previous timestep. It should be noted that in this version nodes embeddings are not considered.

$$W_t^{(l)} = LSTM(W_{t-1}^{(l)}) \tag{2.21}$$

2.4 Qualitative Analysis of Network Representation Learning Models

The models presented in the Factorization-based Category have the main drawback of not being able to generalize over unseen data. For instance, the encoding of a node which was not present during the training phase requires to perform again optimization steps considering the entire matrix. Therefore, node additions are too computationally intensive to handle, in particular when the considered graph is considerably large. The algorithm which satisfies more requirements within this Category is DANE [11] since it relays on matrix perturbation theory to update node embeddings as soon as the graph evolves over time, so it is able to deal with edge addition/deletion under the assumption that such graph variations are not sudden. This means it is possible to retrieve the embeddings of specific nodes through an embedding look up at any time.

Skip-gram-based models partially solve the problem of dealing with node addition since the optimization steps can be computed more efficiently. To generate new embeddings, it is not necessary to consider the entire graph, but it is sufficient to focus on a limited portion of the graph that includes the new nodes and their neighbourhoods. Besides, such optimizations steps can be performed in parallel. However, these approaches have the main drawback of not being suitable to be fed into downstream Machine Learning applications. Indeed, embeddings are learned by optimizing a stochastic node similarity measure that depends on random walks, i.e. the same graph entity can be encoded differently considering a different random walk. And, most importantly, the optimization function is not affected by any arbitrary space embedding orthogonal rotation. This means that the space embeddings obtained

by training separately two distinct graphs can be arbitrarily rotated respect to each other, and worse, given a trained graph and partially retrain it in order to include new graph entities could lead to having space embeddings of the new graph entities rotated. These arbitrary rotations prevent a downstream Machine Learning application, such as a classifier, trained on a particular space embedding orientation to generalize on the newer graph entities if they are rotated. [7]

ConvGNNs and STGNNs are the most relevant categories for the scope of this project since they satisfy most of the established requirements, see Table 2.1. In particular, ConvGNNs present the main advantage of relying on parameters sharing, which means that the encoding functions do not need to learn a unique embedding for each node as in Skip-gram models. Instead, the mapping function learns the weights of each neural networks' layer, and they are shared among all the nodes of the network. These models are inherently suited to leverage node features of attributed graphs since they learn how to encode graph entities in the embedding space by learning a mapping function that takes as input the node features. Accordingly, the quality of the embeddings is strictly related to such features, the more the input features are informative the better low-dimensional representation can be extracted from the attributed graph. GCN [25] is less capable of dealing with dynamic graphs stream. Even if this model can perform prediction on unseen data by updating the Adjacency Matrix A and/or the Features Matrix X, the major constraint that must be guaranteed is to have a fixed order graph as input. If the number of vertices exceeds the one on which the GCN was trained with, it is necessary to retrain the GCN network extending the number of parameters. Moreover, GCN is not suited to perform online inference since is not able to encode only a limited number of target nodes, but it encodes the entire graph. As a result, whenever new information arrives in the graph stream, it is necessary to update all the nodes embeddings, even of those nodes which have not changed. For these reasons, even the dynamic version of GCN, Evolve-GCN, suffers from the same limitations. Instead GraphSAGE does not have the same GCN limitations since it can generate been updated the node embedding of a target node by sampling a limited subgraph constituted by a fixed-size section of the target node neighbourhood.

Graph Autoencoders have the same advantage of parameter sharing as Graph Neural Network (GNN) and also, they inherit the capability of capturing hidden and non-linear pattern in the input data from Autoencoders. However, the GAEs

Table 2.4.1: Network Representation Learning Models Comparison

Model	Category	Bipartite Graph	Attributed Graph	Dynamic Graph	Online Inference	*Embeddings as features*
SVD	Factorization based	✓	✗	✗	✗	✓
HOPE		✗	✗	✗	✗	✓
RolX		✗	✓	✗	✗	✓
DANE		✗	✓	✗	✓	✓
Deepwalk	Skip-gram based	✗	✗	✓	✗	✗
node2vec		✗	✗	✓	✗	✗
BiNE		✓	✗	✓	✗	✗
GCN	ConvGNN	✗	✓	✓	✗	✓
GraphSAGE		✓	✓	✓	✓	✓
SDNE	GAE	✗	✗	✗	✗	✓
GAE*		✗	✓	✗	✗	✓
VGAE		✗	✓	✗	✗	✓
Evolve-GCN-H	STGNN	✗	✓	✓	✓	✓
Evolve-GCN-O		✗	✓	✓	✓	✓

approaches presented are sub-optimal due to the fact they take as an input the entire graph adjacency matrix, hence not only the neural network weights dimension are dependent to the graph order, i.e. the number of nodes n in the graph, but as well they require to retrain the entire network to handle node additions. Indeed, GAEs inherit their ability of handling graph stream from the encoder architecture employed. Both GAE* and VGAE [1] use a GCN encoder, therefore they have the same limitations discussed in the previous paragraph. SDNE [30] relies on a Multi-Layer Perceptron encoder which has not been designed to leverage node features. Besides, it presents the same limitation due to the fact it takes as an input the entire graph adjacency matrix.

To summarise Table 2.4.1 presents a graphical overview of the satisfied properties by each NRL model discussed.

Based on this NRL models comparison, the choice falls on GraphSAGE since it is the most suitable model to handle the dynamic graph stream among the discussed methods. However, for the sake of providing a comprehensive overview, it is worth

to consider as well other NRL models for the early stage of this research to better comprehend the rationale behind these methods, and visualize how the generated embeddings look like using as a reference the same data set. This overview is better discussed in Section 5.1.

Among the proposed methods, the most appropriate are those able to model Bipartite Graphs. The first model is SVD since it is very commonly used in Recommender Systems to perform dimensionality reduction on the so-called user-item matrix [33], which can be considered as a (un)weighted adjacency matrix. Specifically, the two heterogeneous vertices sets consist of the users set and the items set, and the existing edges are the ratings provided by users for specific items. However, it can be generalized to any undirected bipartite graph. As well, BiNE is inherently meant to capture the characteristics of bipartite graphs. The last model considered is RolX even if it has not been specifically designed to deal with bipartite graphs. Anyhow, it is worth to go into it since, among the models presented, it is the only one whose embeddings can be classified as role-based, while the others are all community-based. The embeddings taxonomy to which reference is made has been explained in details in this survey [34], which affirms that it is possible to distinguish among these two approaches to generate embeddings, namely community-based and role-based. The underlying idea behind the former technique is to model the information regarding the community to generate these embeddings. In other words, nodes belonging to the same community are likely to have similar low dimensional representations. Instead, the latter approach aim is to preserve structural similarity among nodes. As a result, nodes with similar topology will have similar representation even if they belong to different communities.

Chapter 3

Graph-based Anomaly Detection

The aim of this Chapter is to further investigate how to leverage Network Representation Learning to perform Anomaly Detection on graphs. First, the main approaches available in the Literature to perform graph-based Anomaly Detection are presented to get inspired and better position the contribution of our work compared to the approaches available in the literature. We will see that our work, see Chapter 5, can be positioned within two categories of Anomaly Detection in dynamic graphs: window-based Anomaly Detection and feature-based Anomaly Detection.

Along with graph-based Anomaly Detection, there is the need to consider as well traditional Anomaly Detection algorithms to analyse the embeddings generated, this is particularly necessary for feature-based Anomaly Detection. For these reasons, in this Chapter are as well presented the auxiliary Anomaly Detection algorithms used to detect anomalies by analysing the embeddings generated with Network Representation Learning models. In particular, since financial transactions graphs are dynamic, the focus is on those models which are able to better model temporal pattern, i.e. RNNs.

3.1 Graph-Based Anomaly Detection

At a high-level, Anomaly Detection can be defined as the retrieval of outlying points in the (high-dimensional) feature space of data points. It must be considered, when dealing with graph representation, that graph-based Anomaly Detection has its own specific challenges which further complicates the detection:

- Inter-dependent objects
  Graph objects can not be considered as independent and identically distributed data points, but they are intrinsically interconnected. Hence, when looking for anomalies, it must be also taken into account the existing correlation among data objects.

- Variety of Definitions
  Since graphs can abstract complex systems, it is possible to define several different types of anomalies such us graph substructures.

- Size of Search Space
  A direct consequence of the previous points is that given a large variety of possible anomalies, as well the search space is huge. And it is even more complicated when dealing with attributed graphs.

In this project, the aim is to generate the most informative node embeddings, which can be further analyzed to perform Anomaly Detection on graphs. In the following section, an overview of graph-based Anomaly Detection techniques is provided not only to better understand the rationale behind these approaches available in the literature, but also to motivate the usage of node embeddings for Anomaly Detection in this project and to position the presented solution respect to the literature. Such techniques are grouped in different categories as proposed by the framework described in this comprehensive survey [5].

The first criterion to distinguish among different graph-based Anomaly Detection algorithms regards the input graph type, i.e. whether if it is dynamic or static. Of course, the thesis focuses mainly on dynamics graphs, but both categories are presented to have a clear picture.

3.1.1 Anomaly Detection in Static Graphs

The main problem addressed in the analysis of Static-Graph Anomaly detection consists in identifying anomalous network entities given a static snapshot of a graph.

- Structure-based methods
  The underlying idea behind those methods resides in the extraction and analysis of structural properties from graphs. Within this category, it is possible to

distinguish among two approaches according to the typology of properties considered. The *feature-based* approach extracts feature from graphs such as degree, centrality measures, egonet features and then it applies traditional Anomaly Detection methods on such features. On the other hand, the *proximity-based* approach focuses on capturing the closeness among graph entities to classify such entities according to their correlation. OddBall [35] is an example of a feature-based approach. It takes into consideration nodes egonet features to identify the normal patterns followed by most of the nodes and to detect the nodes with peculiar patterns. Instead, all the algorithms meant to learn the closeness or the centrality of nodes fall in the proximity-based category. One of the most well known is PageRank [36].

- Community-based methods
  The underlying idea of these approaches is to group nodes in communities, also known as clusters. In this case, the suspicious nodes are isolated or do not belong exactly to a single community. These latter nodes are called *bridges* since they establish a connection among different communities. For example, the approach described in this work [37] leverages Non-Negative Residual Matrix Factorization to detect anomalies. This particular factorization-based approach formulates the decomposition of the original matrix as $A = X \times Y + R$, where X, Y are the low-rank matrices and R is the residual matrix. In particular, the authors demonstrate that X, Y can be used to analyse communities, while the residual matrix can provide useful insights regarding anomalies within the considered graph.

- Relational-learning methods
  These approaches are similar to proximity-based ones since they consider as well the existing relationship among network objects. However, relational classification is different since focuses on inferring the class labels of objects using the correlation among objects and, if available, leveraging further information such as the class labels of neighbours and nodes features, instead of only quantifying the correlation among graph objects. Within this category are included all the approaches based on Belief Propagation (BP), which is a message-passing algorithm for performing inference on graphical models by leveraging the existing relationships. In particular, reference is made to FaBP [38], a linearized approximation of BP with convergence guarantees.

3.1.2 Anomaly Detection in Dynamic Graphs

Anomaly Detection in dynamic graphs consists in identifying anomalous network entities causing the most significant changes in the dynamic graph and/or establishing at which timestamp such significant changes occurred.

- Feature-based methods

  These approaches are based on the assumption that the properties of network entities, or of the network itself, are unlikely to change abruptly over time. A sudden change of such properties is considered suspicious. The framework shared by these approaches consists in extracting features for graph entities that represent the current graph snapshot and monitoring how the similarities of these features evolve over time as the graph evolves. This paper [39] illustrates a pertinent example, which extracts and analyses role-based embeddings, i.e. embeddings that encode nodes structural information, to monitor how the structural dynamics evolve over time. These role-based embeddings are particularly suitable to perform Anomaly Detection when some prior knowledge regarding anomalous behaviours/patterns is available. The embeddings are generated to map accurately the suspicious patterns, which facilitates their detection. The anomalous embeddings represent a sort of signature of their corresponding suspicious behaviour. For this reason, role-based embeddings can generalize over unseen graphs that present similar anomalies.

- Decomposition-based methods

  These approaches share the same framework of feature-based ones, they only differ in the type of *graph summary* they are monitoring. Instead of using feature extraction, they rely on matrix decomposition techniques, e.g. SVD, to generate and analyze the evolution of eigenvectors, eigenvalues or singular values. Compact Matrix Decomposition (CMD) [40] is a factorization-based approach similar to SVD but more efficient, which has been used by its authors to perform Anomaly Detection on dynamic graphs. The reconstruction error obtained by applying CMD is monitored, and in case of anomalous values, the corresponding graph snapshot is marked as suspicious.

- Community-based methods

  These approaches focus on the detection of anomalies by monitoring the evolution of individual clusters over time rather than analysing the overall

network. A representative approach of this category is ECOutlier [41], which can detect anomalies at the node level. In particular, it identifies nodes whose behaviours change over time compared to the other community members, defined as *evolutionary community outliers*. This method consists of two major steps: as soon as is available a new graph snapshot, the communities are matched with the communities identified in the previous graph snapshot to understand the evolutionary trends of communities, and then, the algorithms detect those nodes that have evolved differently from the rest of the community they belong to.

- Window-based methods
  This last category of Anomaly Detection methods aims to detect anomalous patterns in the input graph sequence, hence the anomalies reside in temporal patterns rather than in specific features. These methods are based on the assumption that it is possible to model *normal* temporal patterns. The incoming input sequence patterns are compared to the modelled ones to distinguish among normal and anomalous behaviours. Within this category, there is the Dynamic Behavioral Mixed-Membership (DBMM) [42] algorithm that falls in the role-based embeddings category [39] previously described. The difference resides in the fact that the DBMM algorithm not only encodes nodes roles within embeddings, but it also takes into consideration the temporal dimension by encoding the nodes roles transitions observed in the previous snapshots.

3.2 Auxiliary Anomaly Detection Models

3.2.1 Ensemble Methods

The first category of Anomaly Detection models considered consists of *ensemble methods*. These methods leverage the concept of *wisdom of the crowd*. In other words, the final prediction is nothing but the aggregation of all the predictions of an ensemble of estimators, which generally outperforms the predictions of a single estimator. Despite its simplicity, Ensemble Learning has been selected among possible alternative Anomaly Detection methods since it is very robust, i.e. it maintains good prediction performances even with high-dimensional features spaces, uninformative features, and limited or lacking anomalies in the training set. The following approaches fall

in this category of ensemble methods since they leverage multiple decision trees.

Random Forest

Random Forest [43] is an effective supervised classifier method that exploits the *divide-et-impera* concept to classify samples. It works in a top-down recursive manner, starting from all the training samples, it uses a features selection method to identify a splitting feature and divide the data set in partitions according to the splitting criterion. The algorithm continues to split the data partitions until all the samples in the partition belong to the same class, there are no remaining features that can be used as a splitting feature, or the resulting partition is empty. One of the most important aspects of the algorithm is the feature selection method. It measures how each feature can split the data set in *purer* partitions as possible, i.e. partitions containing samples of the same class. For example, two well-known feature selection methods are Iterative Dichotomiser 3 (ID) and Classification And Regression Tree (CART). The former assigns each feature a measure called Information Gain, Equation 3.3, which is proportional to the decrease in entropy after the data set is split according to that feature. The feature with the highest information gain is selected.

$$Entropy(D) = -\sum_{i=1}^{m} p_i \log(p_i) \tag{3.1}$$

Where m is the total number of classes, p_i is the probability that a sample in the data set D is belonging to class *i*.

$$Entropy(A, D) = \sum_{j=1}^{v} \frac{D_j}{D} Entropy(D_j) \tag{3.2}$$

Where v is the total number of data partitions obtained using A as splitting feature

$$Gain(A, D) = Entropy(D) - Entropy(A, D) \tag{3.3}$$

The latter, instead, assigns to each features a measure called Gini Index, Equation 3.5, which quantifies the impurity of the data partitions which is proportional to the misclassification. The feature with the highest reduction in impurity, Equation **??** is

selected.

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2 \tag{3.4}$$

Where m is the total number of classes, p_i is the probability that a sample in the data set D is belonging to class *i*.

$$Gini(A, D) = \sum_{j=1}^{v} \frac{D_j}{D} Gini(D_j) \tag{3.5}$$

Where v is the total number of data partitions obtained using A as splitting feature.

$$\Delta Gini(A) = Gini(D) - Gini(A, D) \tag{3.6}$$

The ensemble of decision tree classifiers is trained on different random sub sets of the training set. This particular approach of Ensemble Learning is called Bagging.

Isolation Forest

Isolation Forest [44] is a pertinent approach for this work since its underlying working mechanism is based on the simple but effective concept of *isolation*. This approach measures samples susceptibility to being separated from the other samples and marks as suspicious samples those with higher susceptibility values. To isolate samples, this approach leverages binary trees to recursively partition the training samples based on their features in an unsupervised way. The anomalies detection mechanism is based on the assumption that the trained trees are likely to produce shorter paths for data samples that differ a lot from the majority of the other training samples. This algorithm works in two stages: the first step, called the training stage, serves to create and train the isolation trees processing sub-samples of the training set. The second, the so-called evaluation stage, serves to generate the isolation susceptibility of samples in the test set. The main advantage of this method is the fact it is not necessary to define a distance or density measures to detect outliers, but it only analyses the input features and how they are distributed within the entire data set.

3.2.2 RNNs and CNNs for Analysing Sequence Data

Recurrent Neural Networks represent a category of Deep Learning approaches meant to analyse sequences of data and model temporal patterns. The basic concept which characterizes these approaches lies in the augmentation of the networks by giving them *memory* of the information computed so far and by adding connections pointing backwards. In fact, a recurrent neuron retains in its hidden state the memory of what it has seen so far and it has three weights: the weight u for the input $x^{(t)}$, the weight w for the hidden state at the previous time step $h^{(t-1)}$ which is the output vector of the previous time step, and the weight v for the hidden state at the current time step $h^{(t)}$. The output vector $y^{(t)}$ is computed with the following equations:

$$\begin{aligned} h^{(t)} &= tanh(u^T x^{(t)} + w^T h^{(t-1)}) \\ y^{(t)} &= sigmoid(v^T h^{(t)}) \end{aligned} \tag{3.7}$$

Of course, it is possible to stack multiple recurrent neurons in sequence forming a layer, and in turn stack multiple layers to have deeper architecture as needed. Specifically, it is possible to take into consideration the RNN design patterns when modelling the network:

- Sequence-to-Vector: a RNN architecture as long as the input sequence in which only the last output is considered
- Vector-to-Sequence: a RNN architecture as long as the output sequence which takes only one input at the beginning
- Sequence-to-Sequence: a RNN architecture that takes as input a sequence and predicts a sequence of outputs, the two sequences can be of the same length but it is not mandatory and the network length depends on the lengths of the sequences and if they are shifted among each other
- Encode-Decoder Network: a RNN architecture composed by a Sequence-to-Vector network stacked to a Vector-to-Sequence network

RNN may be subjected to the Vanishing Gradient Problem since the number of layers of the network is dependent on the length of the output/input sequences. In Deep Learning, this phenomenon occurs when a network is made of multiple layers and

their activation function is always outputting a value < 1. Hence, whenever multiplied on every layer, the gradient *vanishes* becoming very small. In this thesis, it has been selected a particular type of RNN designed to avoid this problem, namely LSTM, and its simplified version GRU. Along with, it is considered WaveNet, which is a Convolutional Neural Network (CNN)-based architecture. In fact, RNNs are not the only networks which can analyse sequence data, but it is also possible to use CNNs as demonstrated in this comparative study [45].

LSTM

LSTM [32] is an architecture, see Figure 3.2.1, specifically designed to overcome the Vanishing Gradient Problem. As can be guessed from the name, LSTM has two states which represent its memory: long-term state c^t and short-term state h^{t-1}.

LSTM operating principle stands in the processing of the neuron long term state at a given time step t (c^t) by re-weighting it through three gates here described:

- *Forget Gate*: establishes which information has to be forgotten $f^{(t)}$ by applying a *sigmoid* over the previous hidden state $h^{(t-1)}$ and input $x^{(t)}$.

- *Input Gate*: decides what new information should be stored in the cell state $c^{(t)}$ by applying a *sigmoid* over the previous hidden state $h^{(t-1)}$ and input $x^{(t)}$ to decide which values to update $i^{(t)}$, and by applying a *tanh* over the previous hidden state $h^{(t-1)}$ and input x^t to generate a vector of new candidate values $c^{(t)}$.

- *Cell State update*: this is not a proper gate but it is the long-term state update operation which is done by combining the previous state $c^{(t-1)}$ multiplied by the output of the forget gate $f^{(t)}$, and the output of the input gates $i^{(t)}$ and $c^{(t)}$ multiplied together.

- *Output Gate*: decides the final output h^t by applying again a *sigmoid* over the previous hidden state $h^{(t-1)}$ and input $x^{(t)}$, and by multiplying the obtained output by the updated long-term state c^t.

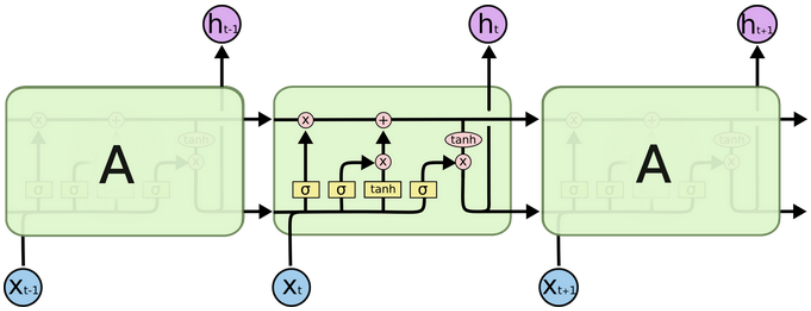


Figure 3.2.1: LSTM Architecture [46]

GRU

GRU [31] is a simplified version of LSTM, it simplifies the two operations of the input gate and forget gate by jointly establishing which information to forget and which to add in the so-called *Update Gate*. As a result, the model forgets about previous information by including newer information in its place, see Equation 3.8.

$$c^{(t)} = f^{(t)} \otimes c^{(t-1)} + (1 - f^{(t)}) \otimes c^{(t)} \tag{3.8}$$

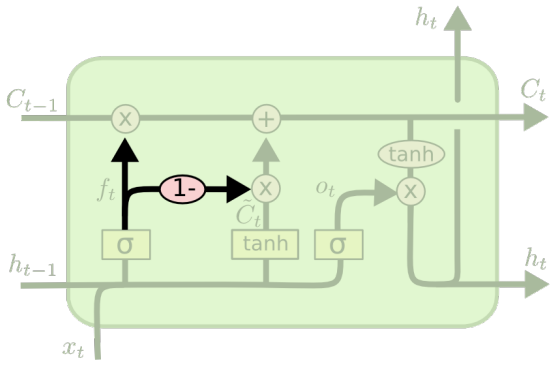


Figure 3.2.2: GRU Architecture [46]

WaveNet

A representative example of CNN applied to temporal sequence is presented by DeepMind researchers, namely WaveNet [47]. The WaveNet architecture comprises stacked 1-dimensional convolutional layers, and at each layer, the dilation rate is doubled as shown in Figure 3.2.3.

This means that the first convolutional layer considers two-time steps of the input sequence, the next layer considers four-time steps since its receptive field is doubled

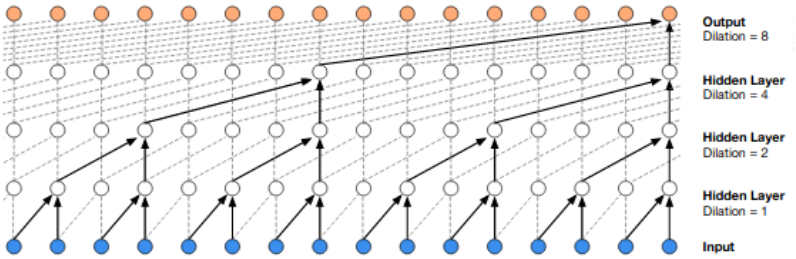


Figure 3.2.3: WaveNet Architecture [47]

compared to the previous one, the layer after that considers eight-time steps, and so on.

This particular architecture allows us to examine both short-term and long-term patterns. The lower layers of the network aggregate the information of subsequent time steps, and as we get to the upper layers, the convolution aggregates information on a wider temporal range until the maximum receptive field capacity is reached. In addition to the ability to learn both short-term and long-term patterns, this architecture is very efficient thanks to the doubling factor. In the original implementation of WaveNet, the authors stacked 10 convolutional layers with dilation rates of 1, 2, 4, 8, ..., 256, 512 and demonstrated that the resulting architecture has the same capability of a convolutional layer with a kernel of size 1024, but with the great advantages of having fewer parameters and consequentially being faster.

Chapter 4

Use Cases

From a high-level perspective, this thesis work aims to design a network representation learning model able to generate high-quality embeddings to feed downstream Anomaly Detection applications. These embeddings are meant to enrich the features analyzed by downstream Machine Learning applications to improve the prediction performances.

The adoption of network representation learning models can be beneficial for many industrial practices which want to leverage graph data representation by using an automatic and data-driven approach to encode graph structural properties in a low-dimensional representation which can be further analyzed and exploited for specific Machine Learning tasks. The main advantage of network representation learning is that it represents a flexible and generalized alternative to *hand-engineered* feature extraction from graph.

In particular, this thesis work aims to demonstrate how node embeddings can be generated and utilized in the financial sector for Anomaly Detection.

In the following chapter a detailed description of the use cases addressed in this thesis project is provided.

4.1 Elliptic Use Case: Anti-Money Laundering in the Bitcoin Network

The first use case addresses the detection of illicit transactions in the Bitcoin Network [48]. The paper [3] introduced this use case, which is the result of a collaborative effort among MIT-IBM Watson AI Lab and Elliptic. The MIT-IBM Watson Lab is a partnership between the Massachusetts Institute of Technology and IBM Research. Elliptic is a company specialised in cryptocurrency intelligence whose aim is to fight criminal activities in cryptocurrency networks. The data set analysed for this use case is the Elliptic Data Set, named after the company that has made it publicly available, and it consists of about two weeks of transactions extracted from the Bitcoin Network.

Even though Bitcoin transactions have their own characteristics and they differ from traditional bank transactions, the data is representative of transactions in the financial industry for several reasons.

First of all, even the Bitcoin network is subject to financial crime. The data set has been specifically provided to investigate further how to accomplish Anti-Money Laundering (AML) in the Bitcoin network, which can be considered as a special case of Anomaly Detection because the positive class, i.e. the class representing deceitful nodes in the network, is the minority class. The labelling process used to mark transactions as licit or illicit is heuristics, which means that labels are assigned according to assumptions that are not guaranteed to be optimal. In the absence of ground truth, heuristics-based labelling is very commonly adopted to process real-world data sets. From this perspective, the Elliptic labels' quality realistically resembles the quality of labels of most of the real-world financial data sets. Another important property of this data set is the fact that it is dynamic. Instead of a unique fixed-size graph, the data includes a time series of consecutive graph snapshots. This allows us to reason about financial systems dynamics. Besides, to the best of our knowledge, this data set is the largest publicly available transaction data set in any cryptocurrency network.

4.1.1 Graph Description

The Elliptic data consists of 49 different graph snapshots. Each snapshot is a connected component, which is nothing but the graph obtained by grouping transactions executed at the same time instant.

Before presenting how the paper's authors have modelled the Bitcoin transactions as a graph, it is essential to provide at a high-level an overview of how transactions work in the Bitcoin network. Bitcoin transactions are specifically designed to guarantee money traceability. In fact, when user A wants to make a payment to user B, A must refer to the previous transaction from which the user A itself has received the money. The reference to the previous transaction is called transaction input, while the actual transfer of money to user B is called transaction output. In turn, the output of a transaction can become the input of a new transaction. For this reason, the transaction operation mechanism is called Unspent Transaction Output (UTXO) model [49]. It should also be specified that multiple inputs and outputs can be included within a single transaction.

The paper's authors decided to model transactions as graph vertices and the reference towards the previous transaction as edges, which actually determine the payments' flows within the Bitcoin network.

The overall dimensions of the graph entities are described here:

- $|E| = 234\,355$ edge payments flows
- $|V| = 203\,769$ transactions

Specifically, the number of illicit transactions is 4545, which corresponds to 2% of the overall number of nodes, and 21% of the overall transactions are licit. Instead, the remaining 77% of the transactions are not labelled. In Figure 4.1.1 it is possible to visualize the distribution of illicit, licit, and unclassified transactions over the different graph snapshots.

The graph is attributed since the paper's authors performed feature engineering and the available features can be divided into two categories: *local features* (94), which consist of the specific information of transactions such as transaction amount, timestamp, number of inputs and outputs included in the transaction, etc., and *aggregated features* (72), which represent similar information extracted in the local features but it is referred to transactions one step backwards and forwards to the current target transaction.

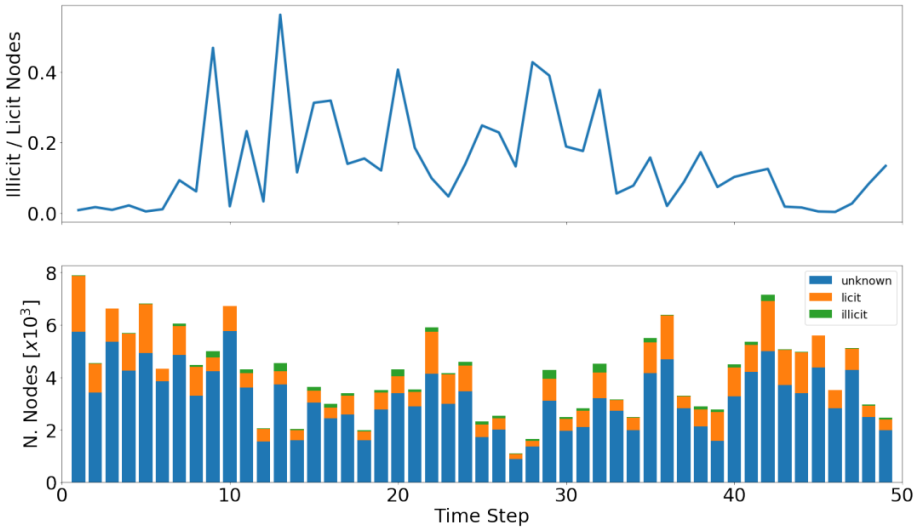


Figure 4.1.1: Typologies of Nodes in the different Graph Snapshots [3]

4.1.2 Types of Anomalies

In this use case, the anomalous class represents harmful financial activities within illicit industries such as black markets, narcotics cartels, human trafficking, terrorism, etc. In the absence of ground truth, the authors have labelled the transactions using a heuristics-based approach. The heuristics considered are based on the assumption that malicious transactions are likely to present similar patterns. For example, it has been demonstrated that it is easier to trace users who execute transactions with a high number of inputs [50] which means that malicious user may prefer executing with a limited inputs number to better preserve their anonymity.

4.1.3 Anomaly Detection

One of the possible approach to address the Anomaly Detection task within this data set consists of leveraging the temporal dimension by training the Anomaly Detection application on the first graph snapshots to predict the labels of the most recent ones. This approach is the most suited since resembles real-world scenario when you rely on past data to detect new incoming information and it is also relevant from an academical perspective since it allows to consider possible strategies to leverage the dynamics of the data set and evaluate how the prediction performance vary along with the period considered for testing. In particular, this data set is peculiar due to the presence of a significant event which consists of a shutdown of a big black market in the Bitcoin Network. From an analytical perspective, this event can be considered as concept drift,

since after it occurred there are no more illicit transactions related with that particular black market. Hence, it is very interesting to evaluate if the model can maintain good prediction performance even in the presence of concept drift.

4.1.4 Limitations

The only limitation of the Elliptic Data Set consists of modelling transactions as graph vertices. Transactions are *volatile* entities, i.e. present in the Bitcoin Network only at the timestamp when they are processed, and it is not possible to evaluate how a single vertex evolve over time. Instead, it is possible to have a better temporal overview of vertices behaviour evolution which is encoded in the node embeddings by modelling transactions as edges and accounts which represent legal entities or private individuals as vertices. Indeed, such accounts can be present in financial transactions in several time instances. For these reasons, in the following section, it is also described the second Use Case which considers a real-world data set which has been provided by Swedbank AB.

4.2 Swedbank AB Use Case: detect stranded customers due to COVID-19 travel restrictions

The second use case addresses the problem of identifying bank customers who might have been forced to remain abroad for a long period due to lockdown measures applied to face the COVID-19 pandemic. This Use Case has been provided by Swedbank AB who has also provided transaction data for a subset of bank customers.

The considered time-period of these transactions is from 20th of January 2020 to 26th of April 2020. This period has been selected since it reflects how the pandemic has affected most of the Swedish population. It is possible to divide this period into two parts: until the 1st of March and after. The first period represents customers' *normal* behaviour while the second part represents a period marked by the onset of restrictions due to the virus' spread. This latter period also reflects many difficulties experienced by customers who due to travel restrictions had a challenging time to get back to Sweden.

The aim of the use case is not to identify financial crime, but is nevertheless relevant because it permits to assess if leveraging node embeddings can be beneficial to perform Anomaly Detection on financial transactions data. In addition, the most interesting aspect resides in the possibility of observing the evolution of customers' behaviour over a time span of four months and to evaluate whether it is possible to model such evolution through the analysis of node embeddings.

4.2.1 Graph Description

In this case, the graph needs to be built from scratch. Hence, it is essential to clarify: the most suitable graph representation, and the most informative features that are associated with the graph entities that can be extracted from the transactions data. These considerations are extremely important since it is possible to extract meaningful embeddings only if the graph has been designed properly.

The input data consists of financial transaction logs of a subset of bank customers.

In particular, the transaction logs' columns are described in the following Table 4.2.1.

The most informative columns are CUSTOMER ID and POINT OF TRANSACTION (POT) ID, which represent the bank's customers and the sender/receiver IDs of the considered transactions, respectively. The CUSTOMER ID entities are homogeneous since they are all bank customers, their information and activity log are kept by the bank in a structured manner, while the POT ID entities are heterogeneous: they could be for example bank accounts (internal or external), tax agencies, or any other imaginable entity. It is worth noting that all personal identifiers have been anonymised in line with Swedbank's operating model and to guarantee adherence to the European Union's privacy regulations.

To generalize the problem we can model it with a bipartite graph $G = (U, V, E)$, see Figure 4.2.1, distinguishing among two sets: the customers set U and the points of transaction set V. These two sets are connected through weighted edges E representing transactions which are weighted with their amount. This distinction is necessary due to the heterogeneity of the two classes.

The dimension of graph entities are described here:

Table 4.2.1: Transaction Dataset

	Table Column	Description
1	DATE	Timestamp
2	EXECUTION TIME	Time in which the transaction has been processed by the system (expressed in seconds)
3	CUSTOMER ID	Customer unique identifier (anonymised IDs)
4	POINT OF TRANSACTION ID	Point of transaction unique identifier
5	AMOUNT	Transaction amount, negative value when the transaction is sent by a customer and positive when it is received by a customer
6	CURRENCY	Currency used for the transaction
7	CHANNEL	Channel used to perform the transaction
8	AGE	Customer age group
9	MUNICIPALITY	Municipality code the customer is registered in

- $|E| = 16.5M$ transactions
- $|U| = 100K$ customers
- $|V| = 2.5M$ points of transaction

Specifically, the number of *"anomalous"* customers which are the ones that might have been stranded abroad is 689 and they represent roughly 0.7% of the considered total number of customers.

4.2.2 Types of Anomalies

In this use case, the anomalous class represents customers who exhibit a higher than usual number of transaction in foreign currencies in the second time period, corresponding to the period in which the impact of the pandemic had started to be noticeable in Europe and many other parts of the world. Similarly, as in the previous case, it is not possible to rely on a ground truth since the labels have been assigned using a heuristics approach. The heuristics considered are based on the assumption that it is reasonable to expect that the number of transactions in foreign currencies decreases

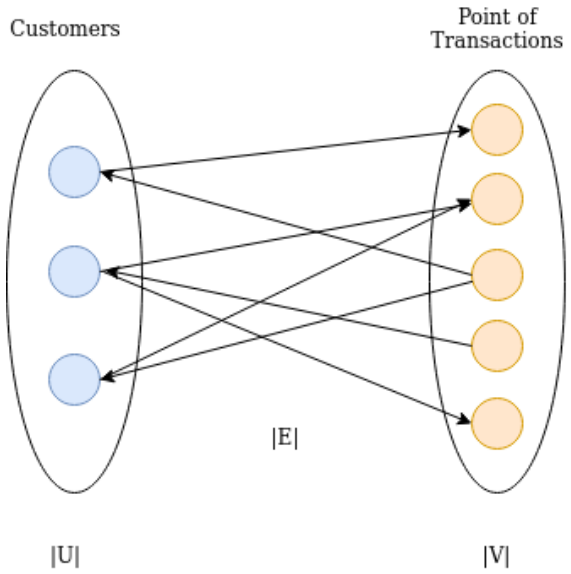


Figure 4.2.1: Bipartite Transactions Graph

in the latter time period, 1st of March onwards. Therefore, customers who exhibit higher numbers of daily average transactions in foreign currencies in the second period compared to the first period are labelled as ”anomalous”. Of course, such heuristics are sub-optimal since paying in foreign currencies does not always automatically translate to being abroad. For this purpose, some of the more international and web retail-related currencies have been filtered out to improve the heuristic labelling.

4.2.3 Anomaly detection

In order to detect customers who might have been forced to remain abroad during the pandemic, it is necessary to generate node embeddings from the financial transactions network and analyse them in order to establish how customers' spending behaviour, in particular in foreign currencies, have been affected.

Even if the focus is on the embeddings of customers, it is worthy to also consider points of transaction embeddings within the network representation learning model to leverage the information of both of the two heterogeneous nodes types. For instance, transactions create relationships among nodes of these two different sets and to generate meaningful customers embeddings it is important to include the information regarding who they having financial transactions with. For Anomaly Detection, however, only customers embeddings can be considered.

To assess the evolution of customers' behaviour over time, a viable option could be

to generate graph snapshots by aggregating the information collected within a time-window and generate the corresponding embeddings. In this way, it is possible to obtain time-series of embeddings which represent the customers' behaviour sampled in different time instances within the considered time window, and further analyse the time series of embeddings where models would be able to detect temporal patterns.

Chapter 5

Methods

5.1 Experiment on the Synthetic Data Set

To better comprehend the properties encoded by different embedding approaches, it has been generated a synthetic data set that resembles the one provided by the bank, but its nodes have known characteristics and it has been simplified. The synthetic data set consists of only three columns: CUSTOMER ID, POINT OF TRANSACTION ID and AMOUNT. The edges are not directed since we consider only outgoing transactions, i. e. customers' spending. In particular, this synthetic data set comprehends four customer categories. Each category is characterized by an average transaction spending since the transaction amount have been established by sampling from Gaussian distributions as described in the following Table 5.1.1.

Table 5.1.1: Synthetic Transaction Data Set Characteristics

Customer Category	Number of Customer within the Category	Transaction Spending *amount (SEK)* $\sim \mathcal{N}(\mu, \sigma^2)$	
Category 0	2000	$\mu = 150$	$\sigma = 20$
Category 1	1000	$\mu = 250$	$\sigma = 30$
Category 2	800	$\mu = 500$	$\sigma = 50$
Category 3	200	$\mu = 1000$	$\sigma = 150$

As well, the points of transaction, 3000 in total, are divided into three categories. Each category is characterized by its average degree to resemble the power-law distribution of the degree of the real data set.

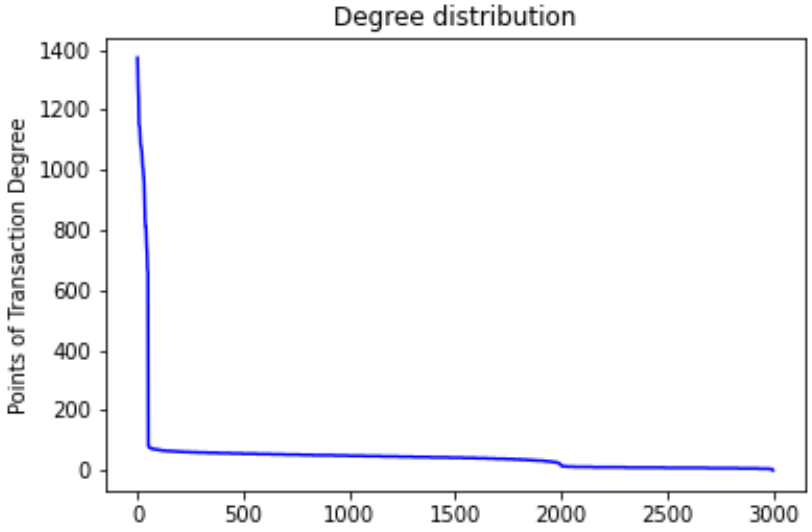


Figure 5.1.1: Degree Distribution of Synthetic Data Set

The pairing of points of transaction with customers is based on the assumption that *rich* customers, i.e. customers within Category 3, mainly spend their money on points of transaction belonging to a niche community, i.e. points of transaction with a limited degree. Instead, common points of transaction are more or less paired evenly with all the customer categories.

In particular, the aim of using this synthetic and domain-specific data set is to compare the embeddings generated by a selection of Network Representation Learning model described in the background chapter and select the most appropriate.

SVD

The first model selected is SVD [8]. From the input row-based transactions, it has been generated a matrix $U \times V$ that has as many rows as customers U and as many columns as points of transaction V. The input matrix consists in the weighted adjacency matrix: each matrix cell $M_{i,j}$ contains the transaction amount spent by the i^{th} customer at the j^{th} point of transaction. Then, this matrix is decomposed using Singular Value Decomposition to obtain the Left Singular Vectors which correspond to customers node embeddings. The only parameter selected in this case is the number of singular values which determines the node embeddings dimension. To make a fair comparison of the models, all the node embeddings generated have a fixed size $d = 64$.

In Figure 5.1.2, it is possible to notice that this model is extremely sensitive to transactions amount. Indeed, the representation of customer of Category 3 is much

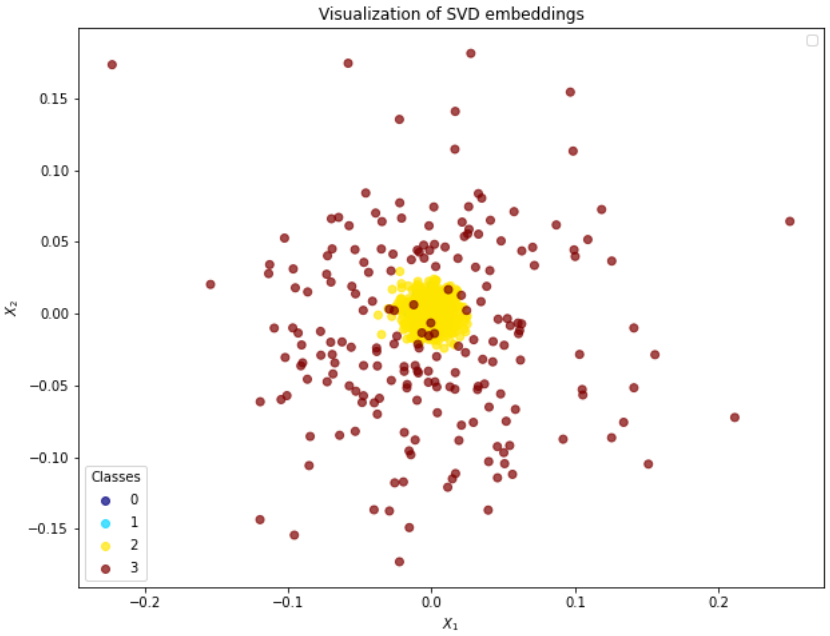


Figure 5.1.2: 2D Visualization of SVD embeddings after applying Principal Component Analysis (PCA)

diverse than the ones of other customers since their amount mean and variation are larger. On the other hand, customers embeddings principal components of the other categories are very close to zero.

RolX

The second model considered is RolX [10], this approach is an improvement compared to SVD since it does not directly decompose the input adjacency matrix, but it first extracts in a recursive manner structural features from the input graph and then applies Non-Negative Matrix Factorization. The only parameters here is again the embedding dimension $d = 64$.

In Figure 5.1.3, it is possible to notice that nodes embeddings generated with this model are more diverse even if it is not possible to identify different clusters of customers categories. The embeddings generated by this approach are meant to encode similarly nodes which have similar structural properties rather than nodes which belongs to the same community. In fact, according to this article [34] RolX falls under the category of role-based embeddings.

BiNE

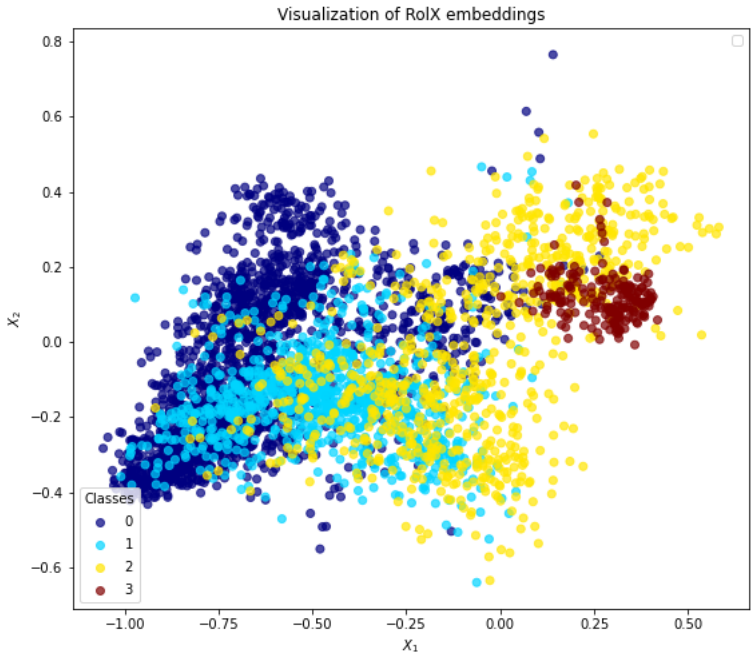


Figure 5.1.3: 2D Visualization of RolX embeddings after applying PCA

The third model BiNE [13] consists in a Skip-gram based approach meant for bipartite graphs. The parameters used for this experiment are listed:

- Embeddings dimension $d = 64$
- Window size 5
- Number of negative samples 4
- Maximum number of random walks per vertex 32
- Minimum number of random walks per vertex 1
- Walk stopping probability 15%
- Centrality Measure $HITS$

In Figure 5.1.4, it is possible to notice that the model can generate nodes embeddings resembling customer categories even if such communities partially overlap. Moreover, the disposition of such customer categories is not casual but it is ordered according to the average transaction amounts. These positive results are expected since BiNE takes into account that the input graph is bipartite and better capture the bipartite-specific properties.

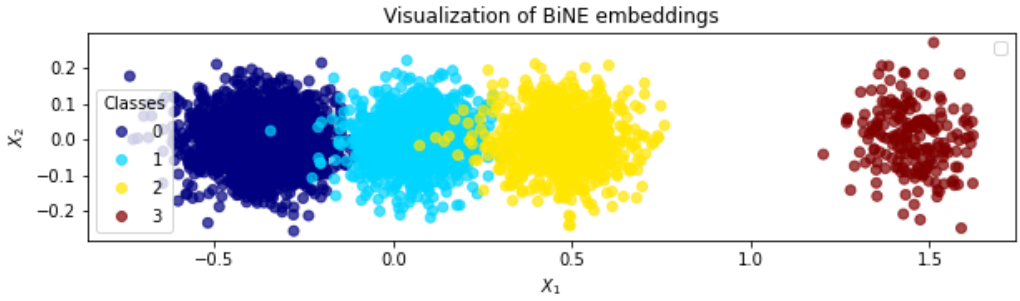


Figure 5.1.4: 2D Visualization of BiNE embeddings after applying PCA

GraphSAGE

The last model considered is GraphSAGE [15], it has been trained in an unsupervised manner, but without using random walks as suggested by its authors. Instead, it has been added a regression layer on top of the GraphSAGE architecture, which aim is to predict transaction amount given as input the concatenated embeddings of both customer and point of transaction involved in that transaction. Since GraphSAGE needs input nodes features, it has been generated a feature vector for each customer representing customers' spending behaviour and a feature vector for each point of transaction representing incomes. Such feature vectors consist of a discrete representation of transactions by defining amount ranges 5.1 and counting the number of transaction in each range for each customer and for each point of transaction.

$$\underset{\textit{Transaction ranges (SEK)}}{[0, 20, 50, 100, 150, 200, 250, 300, 500, 750, 1000, 1250, 1500, 2000, 2500, 5000, \infty]} \tag{5.1}$$

The parameters used for this experiment are listed:

- Embeddings dimension $d = 64$
- Number of layers $L = 2$
- Number of Samples $S_1 = 8, S_2 = 4$
- Epochs 20
- Embeddings dimensions at each layer $d_1 = 64, d_2 = 64$
- Train-test split 70-30%

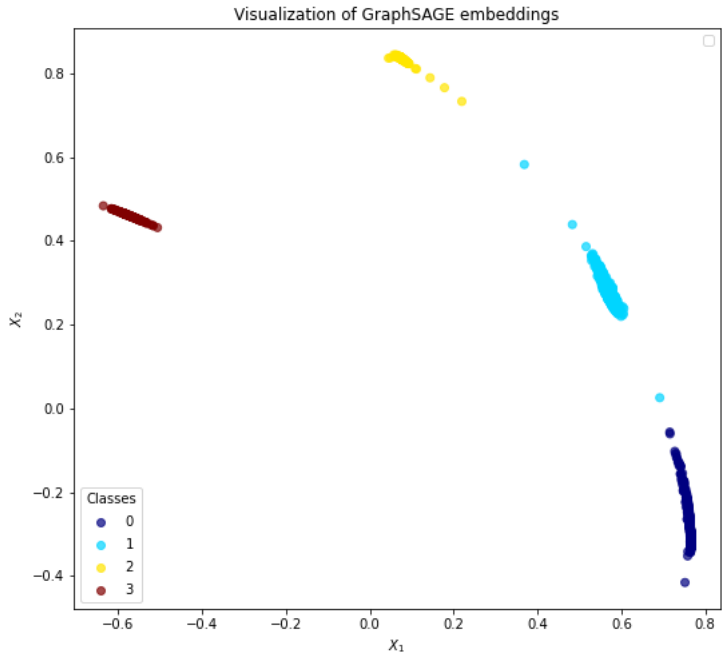


Figure 5.1.5: 2D Visualization of GraphSAGE embeddings after applying PCA

The embeddings generated by GraphSAGE exhibit the clearest distinction among the different customer categories. Moreover, it is possible to notice that the disposition of customers categories follows certain criteria as in BiNE embeddings. For instance, customers of Category 3 are much more distant from the other customers, which are aligned and ordered from top-left to bottom-right according to decreasing average spending.

5.2 Experimental Methodology for Elliptic Data Set

The characteristics of the Elliptic Data set consent to investigate how to train a NRL model in a supervised manner, i. e. leveraging the available node labels. In this case ConvGNNs represent an interesting choice since it is possible to augment the neural network architecture by adding on top of the last layer which generates the embedding a network which learns how to classify the nodes based on their embeddings. In this manner, the NRL model and the Anomaly Detection downstream model are jointly trained, and the node embeddings generated are optimized to perform node classification.

Another aspect considered within these experiments consists of evaluating how to train the model to better capture the system dynamics. For this reason, are both implemented a static and a dynamic variant of GraphSAGE.

A better overview of the models' implementations and the discussion about the achieved results are described in Section 6.1.

5.2.1 Pre-Processing

Most of the pre-processing steps have already been performed by the Elliptic company. These steps are necessary to make the graph snapshots adequate inputs for the model. In particular, the most important task is feature extraction, which requires proper domain expertise to extract relevant information from the Bitcoin Network.

What remains to be done is the data set partitioning among train, validation, and test. For both the GraphSAGE variants has been applied a temporal split. In particular, the first 30 graph snapshots, which corresponds to the 60% of the considered temporal window, are used for training. The subsequent 5 graph snapshots, from 31^{th} to 35^{th} time-steps, are used for validation purposes, e.g. selecting the best weights of the neural network. The last 14 remaining graph snapshots are used to evaluate the prediction performance of the model. The main difference lies in the way in which the training set is passed to the GraphSAGE variants. In the case of the static model, the training snapshots are grouped and passed to the model in batches. Instead, for the dynamic version, the snapshots are passed maintaining the distinction among the several snapshots.

Another aspect to consider is how to leverage the information regarding unlabelled transactions. Even if such nodes are not classified, their features contain important information regarding the graph topology. To retain the information of such data, we also consider unlabelled transactions. In this way, the model learns how to aggregate the information for the entire graph, but a mask is used for the node classification to evaluate the prediction performance only on the labelled nodes.

5.2.2 Models Architecture

The two GraphSAGE variants share the same underlying structure, but the strategy used for learning the weights is completely different. First, the static version is

Algorithm 2 GraphSAGE batch forward propagation

Input: Graph $G = (V, E)$, input features $x_v, \forall v \in$ Batch B, depth L, weight matrices $W(l)\ \forall l \in \{1, ..., L\}$, non-linearity σ, differentiable aggregation function $AGGREGATE_l\ \forall l \in \{1, ..., L\}$, neighborhood sampling functions $N_l : v \rightarrow 2^V, \forall l \in \{1, ..., L\}$

Output: Vector representations $z_v\ \forall\ v \in B$

$B^L \leftarrow B$
for $l = L...1$ **do**
    $B^{l-1} \leftarrow B^l$
    for $v \in B^l$ **do**
        $B^{l-1} \leftarrow B^{l-1} \cup N_l(v)$
    end for
end for
$h_v^0 \leftarrow x_v, \forall v \in B^0$
for $l = 1...L$ **do**
    for $v \in B^l$ **do**
        $h_{N(v)}^l \leftarrow AGGREGATE_l\left(\{h_u^{l-1}, \forall u \in N(v)\}\right)$
        $h_v^l \leftarrow \sigma\left(W(l) \cdot CONCAT\left(h_v^{l-1}, h_{N(v)}^l\right)\right)$
        $h_v^l \leftarrow \frac{h_v^l}{\|h_v^l\|_2}$
    end for
end for
$z_v \leftarrow h_v^L \forall v \in B$

described to facilitate the comprehension of the changes made to implement the dynamic variant.

GraphSAGE Static Version

The network architecture reflects the underlying working mechanism of GraphSAGE as presented in the Forward Propagation Algorithm 1 in the Background Section. Of course, the algorithm needs to be slightly modified to be able to process nodes batches instead of the entire nodes set. Here is described the Forward Propagation Algorithm adapted to batch settings.

The main modification in the algorithm consists of sampling all the nodes needed for the batch computation. In addition, respect to Algorithm 1, there are as many neighbourhood sampling functions as layers to highlight the fact it is possible to limit the number of sampled neighbours at each layer. It is important to notice that the sampling process in the first part of the algorithm is reversed compared to the forward propagation step because it causes some counter-intuitive implications. Let us consider as an example a 2-layered GraphSAGE with S_1, S_2 parameters which

limit the sampling size at each layer. The first implication is that the model starts aggregating the node features from the remotest neighbours, i. e. 2-hop neighbours, and secondly the resulting sampling size will count S_2 1-hop neighbours and $S_1 \cdot S_2$ 2-hop neighbours.

To better comprehend the network architecture, Figure 5.2.1 shows the schema of a 2-layered GraphSAGE model. GraphSAGE takes as input the node features specified in the batch B^1 and the corresponding node features of the sampled 2-hop neighbours. The neighbours' features are then aggregated to obtain a unique vector and such operation depends on the selected aggregator. In our case, we use mean aggregator and pool aggregator among the aggregators proposed in the original implementation [15]. The LSTM aggregator is excluded since it is not permutation invariant, which means that the same neighbours' nodes processed in a different order can lead to different hidden representations. The node features and the aggregated neighbours' vectors are concatenated and fed into a fully-connected layer, which generates the embeddings for the target nodes in the batch B^1. These hidden representations are used as an input for the second layer that takes the embeddings of the target nodes in the batch B^2 and the embeddings of the sampled 1-hop neighbours. As before the neighbours' embeddings are first aggregated, and then concatenated to form the corresponding node embedding. The last fully-connected layer outputs the predicted labels for the target nodes in the batch B^2.

The model weights are learned by minimizing the binary cross-entropy of the node labels where y are the true labels and $\hat{y}$ are the predicted labels:

$$\text{cross-entropy} = -(y)log(\hat{y}) + (1-y)log(1-\hat{y}) \tag{5.2}$$

GraphSAGE Dynamic Version: EvolveGraphSAGE-O

To capture the system dynamics in the embeddings generation, the EvolveGCN-O model has been modified using GraphSAGE as Network Representation Learning model instead of GCN to generate the embeddings in a more scalable manner. This new version is called EvolveGraphSAGE-O, in which the O specifies the type of recurrent neural network architecture used to learn the weights dynamically. The underlying GraphSAGE architecture remains almost unchanged, despite the fact there is a decoupling among the part of the network which learns how to generate the embeddings and the part of the network which learns how to map the node embeddings

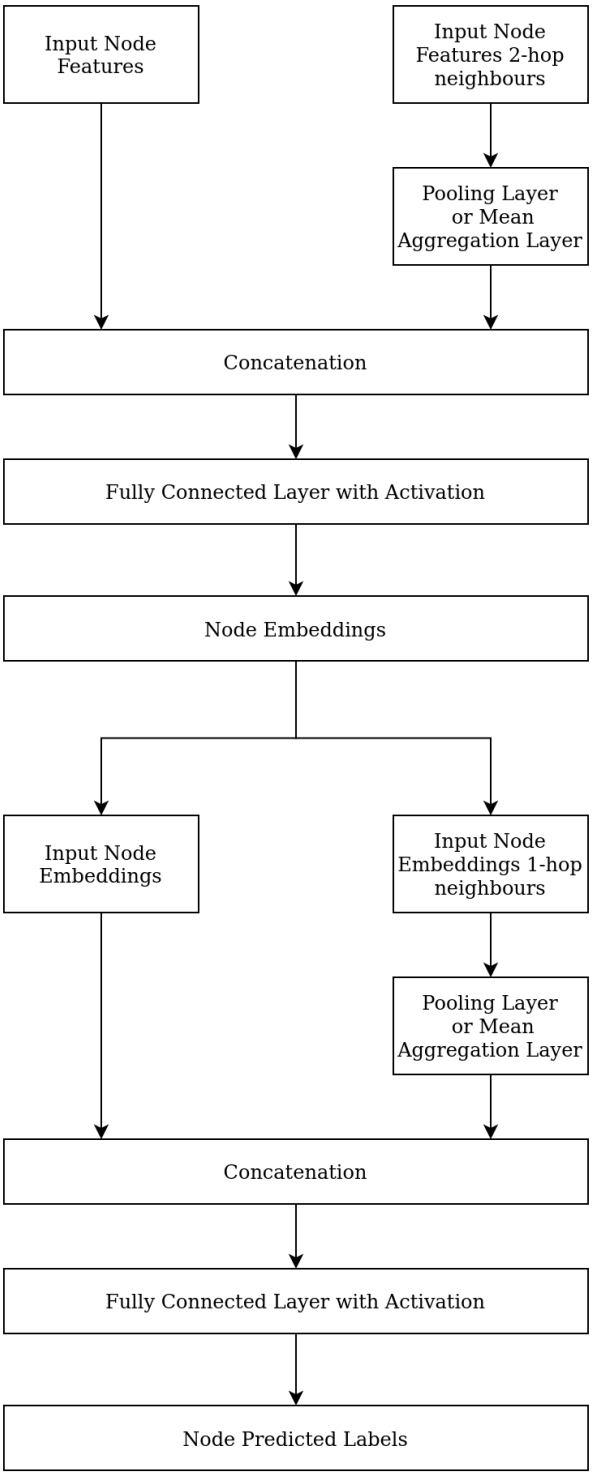


Figure 5.2.1: GraphSAGE Architecture

to their predicted labels. Furthermore, there is the necessity to specify an additional parameter: the number of graph snapshots to consider simultaneously to appraise the graph dynamics. Let us clarify these modifications with an example and the corresponding visual representation. The aim is to generate the embeddings for the t^{th} graph snapshot. Hence, it is needed that specific graph snapshot and n previous

graph snapshots, in our case n is equal to 5. The forward step of the algorithm consists in generating the embeddings for each graph snapshot from t-n until t, letting GraphSAGE weights evolve at each temporal iteration over the graph snapshots window using a GRU as shown in the Figure 5.2.2. Only the embeddings generated for the t^{th} are then used to evaluate the prediction performance and fed into a Multi-Layer Perceptron with two layers. This particular architecture which takes as an input a sequence of input data and ignores all the outputs except for the last one is called Sequence-to-Vector and it is one of the possible Design Patterns used to model RNNs.

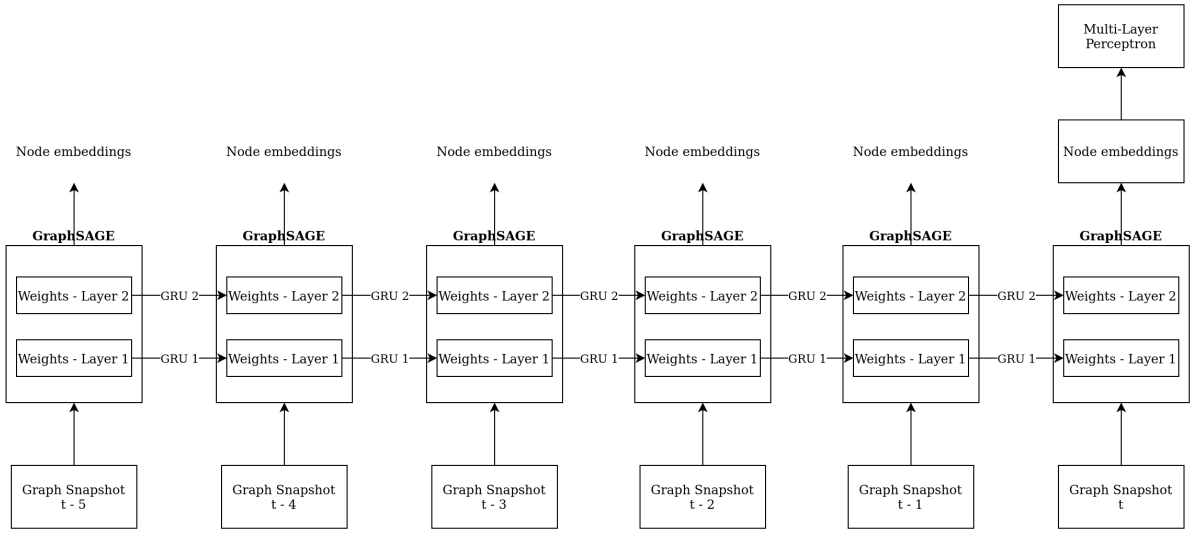


Figure 5.2.2: EvolveGraphSAGE-O - Sequence-to-Vector Design Pattern

As in the previous version, the model weights are updated by minimizing the binary cross-entropy of the node labels.

5.2.3 Metrics

The models considered learn how to classify each transaction as licit or illicit, this can be generalized as a binary classification problem in which the positive class generalizes the illicit class and the negative class generalizes the licit class. To assess the predictive performance of a classifier it is important to consider the values computed in the confusion matrix:

- True Positive - number of samples of the positive class which the model recognize as positive
- True Negative - number of samples of the negative class which the model

recognizes as negative

- False Positive - number of samples of the negative class which the model recognizes as positive
- False Negative - number of samples of the positive class which the model recognized as negative

By analysing the values in the confusion matrix it is possible to obtain the following metrics: **Accuracy** Accuracy represents the ratio of correct predictions over all the samples. This metrics is not suitable in case of unbalanced data set since even with low prediction performance on the minority class the accuracy value remains high.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.3}$$

Precision Precision represents the ratio of correct positive prediction over all the number of positive predictions. Precision is indicative of the number of false positives.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.4}$$

Recall Recall represents the ratio of correct prediction prediction over all the number of positive samples. Recall is indicative of the number of false negatives.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.5}$$

F1-Score The F1-Score is the harmonic mean of Precision and Recall, this score is widely exploited since it is a trade-off among the previous metrics and consent to have a better understanding of the predictive performance of the model.

$$\text{F1-Score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{5.6}$$

The main metric considered is the micro average F-1 score which is used to evaluate the accuracy of the model in predicting the node labels for both the classes.

$$\text{Micro-avg-F1-Score} = \frac{TP_{class0} + TP_{class1}}{TP_{class0} + TP_{class1} + FP_{class0} + FP_{class1}} \tag{5.7}$$

In particular, since this is an example of Anomaly Detection, the minority class is the most interesting one. Therefore, as well the Precision, the Recall and the F-1 score of

the illicit class as considered since the aim is to correctly detect such anomalies. Due to the sensitivity of the application domain, in Financial Industry is more important to have a high Recall to spot as much as anomalies as possible even if it could mean to slightly increase the false positive rate. It is common to further analysed the samples predicted as suspicious by more sophisticated model and/or to submit them to human inspection.

5.3 Experimental Methodology for Swedbank

The characteristics of the Swedbank data allow for an investigation on how to train a NRL model in a unsupervised manner and evaluate whether it is possible to observe the evolution of customers' behaviour by encoding node features and graph structure into node embeddings.

Another aspect considered within these experiments consists of modifying the GraphSAGE architecture in order to handle heterogeneous nodes, since the graph is bipartite and the nodes in the two different sets have different features.

A better overview over the model implementation and achieved results is discussed in Section 6.2.

5.3.1 Pre-Processing

Features Extraction

When dealing with Graph Convolutional Networks, it is extremely important to provide the ConvGNN models with high-quality input node features in order to exploit the representational capacity of this particular type of neural networks as much as possible. Therefore, the pre-processing steps are crucial and they are further complicated due to the necessity to aggregate the transactions and extract graph snapshots in several time instants in order to appreciate how the customers' behaviour changes over the time period. For these reasons, it is important to consider a trade-off among features' quality and computational complexity of extracting such features. Along with node features, it is also necessary to maintain the graph structure since ConvGNNs also need such information to generate embeddings.

A possible solution which allows for extracting graph snapshots, aggregates

transactions which occur in the same time window using a stateful streaming application, see Figure 5.3.1. The main reason to opt for this approach is to deal with concept drift [51], since the financial transaction graphs evolve over time and the normal behaviour is expected to change, and aggregating transactions distant in time could lead to misleading and noisy graph embeddings.

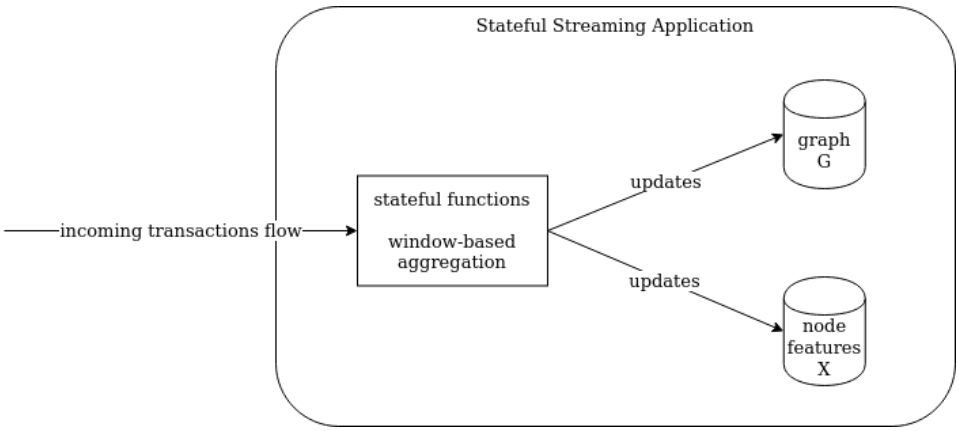


Figure 5.3.1: Features Creation within the Streaming Application

For the scope of this experiment, the development of a streaming application to ingest the input data and generate the snapshots is not necessary, since the experiment has been conducted offline. But in order to asses if this solution could be suitable to really perform real-time Anomaly Detection it is necessary to consider the need to collect graph snapshots from the incoming transaction streams.

In fact, for this experiment two approaches have been simulated to generate graph snapshots, see Figure 5.3.2. Both of these approaches consider a time window of one week, but the first approach consists of a tumbling window strategy which aggregates information week by week. Instead, the second approach consists of a time sliding window strategy which still aggregates transactions within a one-week but the time window slides across the transaction stream according to a specified interval, set to two days, i. e. every two days a graph snapshot is generated which contains the most recent transactions executed in the last seven days.

The input features extracted within the graph snapshot should be representative of customers' spending behaviour and points of transaction amounts, thus the node features are extracted by aggregating the most significant attributes of the transactions considered in the snapshot. The extracted input features for the two heterogeneous classes are shown in Figure 5.3.3. Customers are characterized by static features which consists of selected demographic information, i.e. customer age which has

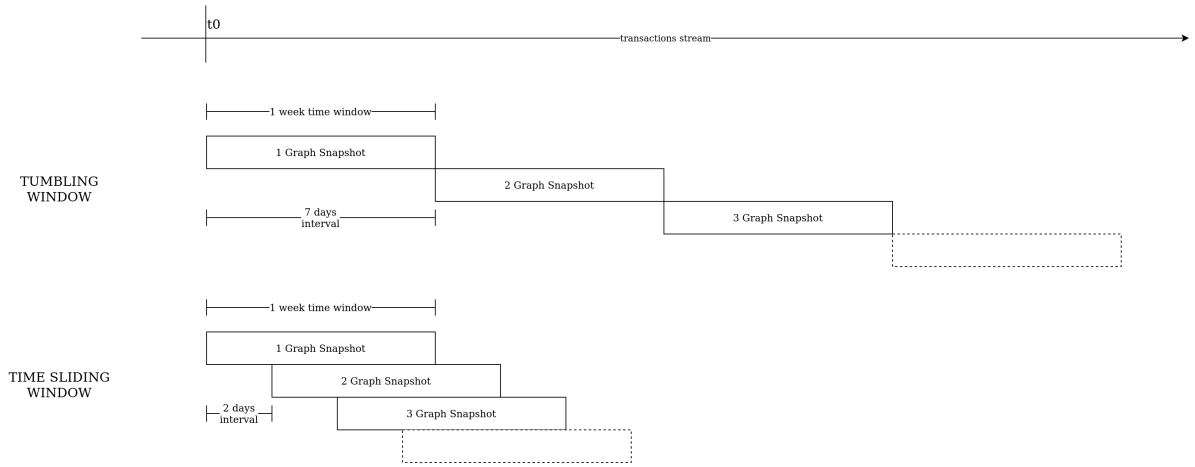


Figure 5.3.2: Time Window Strategies to Generate Graph Snapshots

been subdivided in age groups and the one-hot encoding of customers' municipality belonging. These demographic features are static, since they are unlikely to change in a short period and can be updated only when required. The remaining features represent the spending behaviour of customers in terms of currency and channels used for the payments aggregated by time-window and grouped by categorical values. Points of transaction do not contain the same type of information, but they do have features which represent their income structure similar to customers' aggregated features.

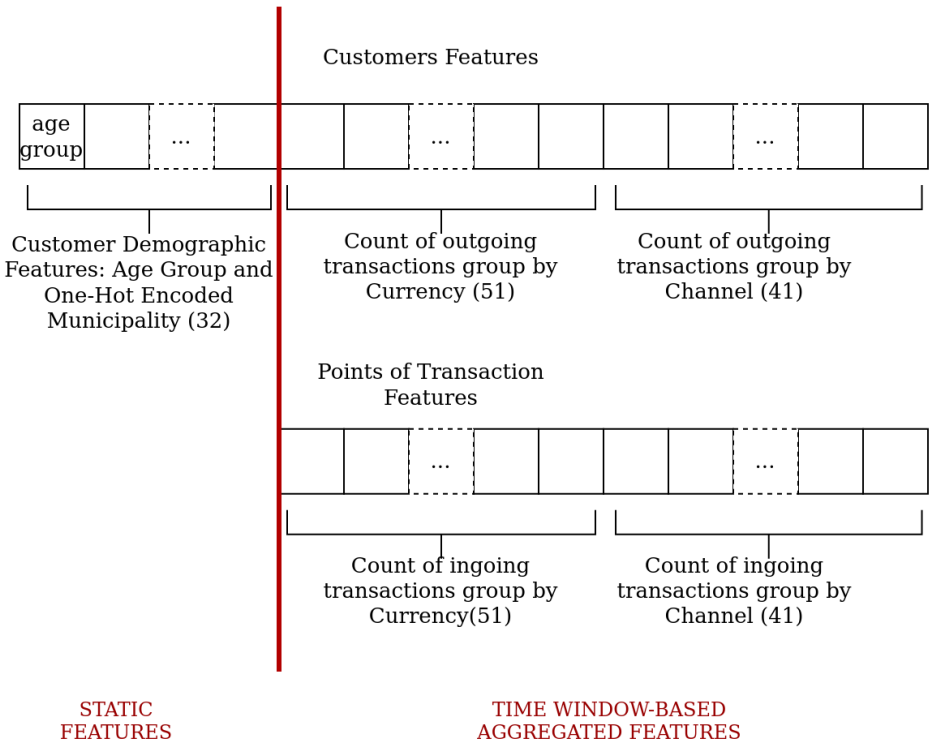


Figure 5.3.3: Extracted Node Features

Data Cleaning and Preparation

Once the most important features needed and their extraction has been established, it is necessary to consider the data cleaning and preparation steps needed to remove noisy and/or unnecessary information and make the input digestible by the models. The data cleaning and preparation operations applied to the input data set are listed:

- **Removal of incoming transactions from the customer perspective.**
  For this particular use case, the most interesting aspect is to investigate patterns of customers' spending. Therefore, only a particular type of transaction direction is considered, from customers to points of transaction, which represents the majority of the edges in the given data.

- **Removal of infrequent categorical values and one-hot Encoding.**
  The transaction data contains categorical attributes which need to be processed. These attributes are currency, transaction channel and municipality, and for each of them, a limited number of distinct values has been considered. For example, there are over a hundred different currencies in the data but almost half of them are very infrequent, i.e. occur less than the 0.05% of the time, so all these infrequent currencies have been assigned a unique code. This adjustment is useful because it allows to limit the node features' dimensionality and at the same time keep such dimension fixed over time. For municipality, one-Hot Encoding is used and the resulting one-hot encoded vectors are utilized as part of the customer static demographic features. For currency and transaction channel, two fixed-size vectors are allocated with the vectors being as long as the number of distinct values within the respective category. Each i^{th} vector element contains the count of the i^{th} value present in the transactions in which a particular node is involved during a time window. The currency and transaction channel counts are kept for both customers and node embeddings. Municipality however, is an attribute applicable only for customers.

- **Removal of Points of Transaction with with frequencies.**
  This last step is performed for improving the quality of the embeddings. The distribution of degrees of points of transaction exhibits a power law pattern, which is very typical for real-world graphs. [52] This particular degree distribution could negatively impact the NRL model performance since very

frequent points of transaction significantly increase the number of two-hop neighbours within the network. In particular, when applying a ConvGNN-based solution, this means that among the neighbours of a target customer it is possible to find other customers that are not very similar from a topological perspective and the only connection among them is due to the very frequent points of transaction. For example, two customers that frequently visit the same popular supermarket chain do not necessarily share the same overall spending pattern. Pruning overly frequent points of transaction and their corresponding transactions is also a best practice from an Anomaly Detection perspective since very frequent transactions are not as meaningful; anomalous behaviours tend to resides in the long tail of distribution.

After these pre-processing steps, the dimensions of the graph entities are the following:

- $|E| = 9.1M$ transactions
- $|U| = 93K$ customers
- $|V| = 1.5M$ points of transaction

Train-Test Split

Due to the considerable size of the Swedbank financial transactions graph, the Network Representation Learning model has been trained using 20% of the customers and all of their corresponding transactions. This 20% has been selected by including all customers who exhibit at least one transaction in an uncommon foreign currency, e.g. excluding the foreign currencies of neighbouring countries which are very frequent. The the remaining customers have been selected randomly. In particular, for the selected customers the first four graph snapshots have been considered; obtained using a tumbling window strategy for the training which includes the transactions from January 20th to February 16th.

For Anomaly Detection, both the entire graph with all the customers and a sampled graph have been considered. The aim is to understand whether the Network Representation Learning model trained over a subset of nodes is still able to generalize over the entire data by generating the embeddings even for unseen graph entities. For the supervised Anomaly Detection models considered, a 70-30% train-test split at the customer level has been selected. In the unsupervised case, all customers are provided

to the models.

5.3.2 Models Architecture

Network Representation Learning Model

This section presents the GraphSAGE architecture modified in order to handle a bipartite graph $G = (V, U, E)$. Figure 5.3.4 represents a 2-layered architecture as an example, it is possible to notice that the architecture presents two GraphSAGE networks which combined together are complementary.

The left-handed GraphSAGE network generates the embeddings for the nodes in the V set and the right-handed one for the nodes in the U set. As intermediate embeddings each GraphSAGE network generates the embeddings for the opposite target set, and these intermediate embeddings need to be exchanged among the GraphSAGE networks in order to retrieve the embeddings of the target nodes set. The two GraphSAGE networks do not directly generate the predictions but instead they generate the embeddings by aggregating the information of two-hop neighbours. In this manner the resulting embeddings encode both of the nodes sets into low-dimensional vectors, and in this manner it is possible to also have the compact representation of an edge by combining the two embeddings of the nodes which are related through that edge, namely link embedding. Having the link embeddings, it is possible to generate a prediction according to the unsupervised task which has been assigned to the architecture.

In these experiments we have considered two unsupervised tasks. The first is the traditional unsupervised learning task which leverages negative sampling, also used by the Skip-Gram model, to assign positive labels to links which actually are present in the graph and sampled through random walks and a negative labels to links which do not exists and connect nodes which are distant from each other. In this case, the network weights are learned by optimizing a binary cross-entropy loss simplified to the below expression:

$$\begin{gathered} \textit{Negative Sampling Loss } = -log(\sigma(z_u^T z_v)) - log(\sigma(-z_u^T z_{v_n})) \\ \textit{where } (u, v) \textit{ and } (u, v_n) \textit{ are positive and negative pairs respectively} \end{gathered} \tag{5.8}$$

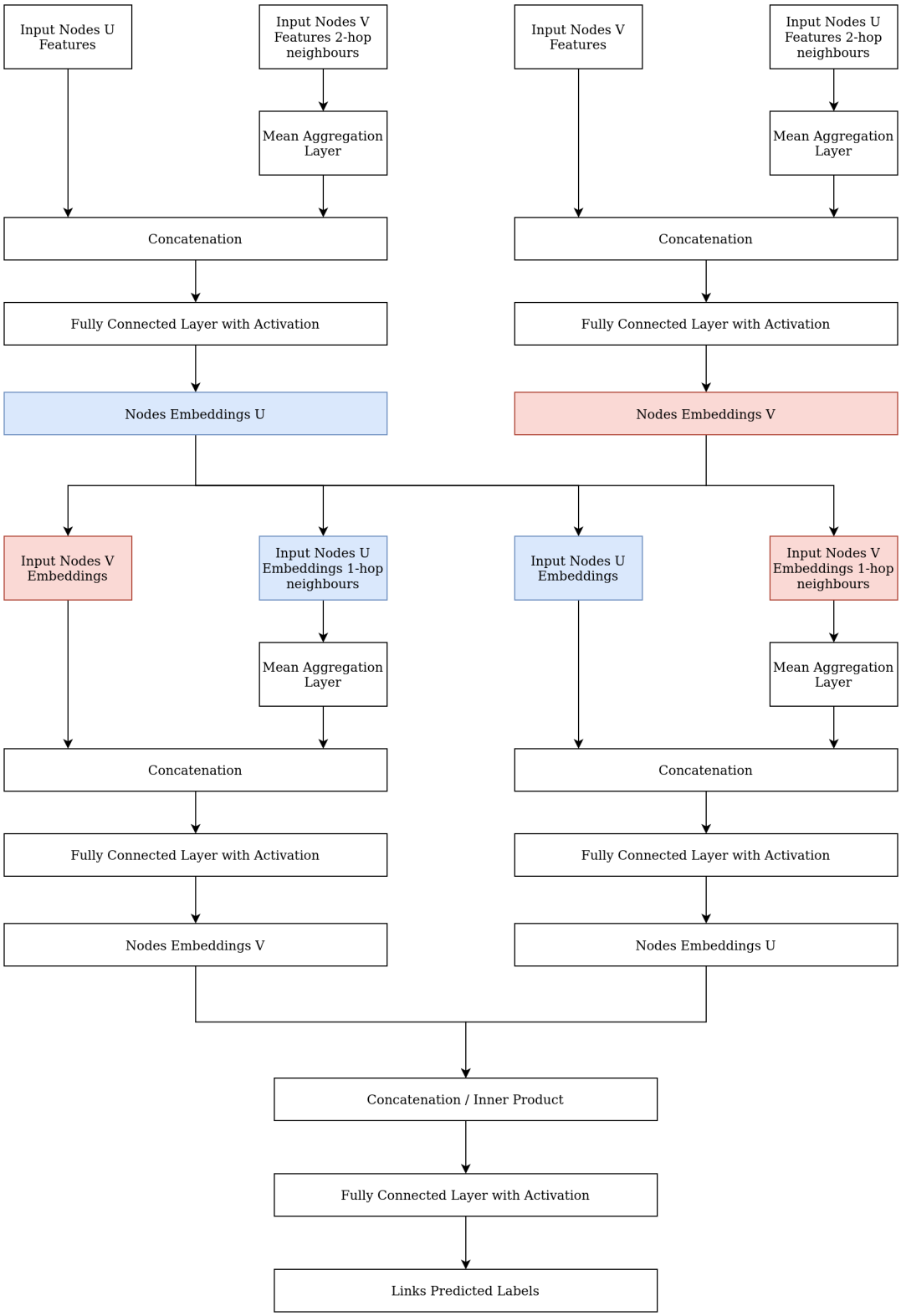


Figure 5.3.4: GraphSAGE Architecture for Bipartite Graphs

The second unsupervised learning task has been specifically designed for this use case. Since the aim is to detect whether customers change spending behaviour in foreign currencies, it is assigned a positive label to a transaction in foreign currency instead

if the transaction is in local currency, i. e. Swedish Crowns SEK, it is considered a negative label. The loss optimized is simplified in the below expression:

$$\textit{Link Classification Loss} = -log(\sigma(concat(z_u, z_v))) - log(\sigma(-concat(z_u, z_{v_n}))) \quad \textit{where } (u, v) \textit{ and } (u, v_n) \textit{ are transaction in foreign currencies and SEK respectively} \tag{5.9}$$

Anomaly Detection Models

Once the node embeddings have been generated, they are used as input for downstream Anomaly Detection applications. The models used, namely Isolation Forest, LSTM and WaveNet, have been implemented as presented by their authors without making any particular adjustments. Their implementation is briefly described.

For Isolation Forest, the implementation provided within the Scikit-learn [53] Python package has been used. As an unsupervised method it takes all the customers in the data as input without dividing them in training and test sets.

LSTM and WaveNet have been implemented using TensorFlow [54] and following the guidelines provided in [55]. They have been trained using 70% of the customers, the remaining part is used for testing.

5.3.3 Metrics

As in the previous use case, the minority class is the most interesting one. Therefore, precision, recall and the F-1 score of the minority class which represent customers who might be stranded are considered.

Chapter 6

Results

6.1 Results on the Elliptic Data Set

6.1.1 GraphSAGE

In order to detect the illicit transactions in the Elliptic Data Set [3], a 2-layered GraphSAGE version has been implemented in Pytorch [56] which aggregates the information of the nodes local neighbourhood up to 2-hops neighbours. Both the mean aggregator and the pool aggregator are considered in the experiment to appreciate if they differently contribute to the prediction performance. The parameters used for this experiment and the specification regarding the computational platform and software libraries used are listed in Appendix A.1.1 and Appendix A.2.1 respectively.

The baseline models selected to compare the prediction performances of Graphsage with are the ConvGNN framework used by the authors, namely GCN and Skip-GCN.

The first results show the prediction performances of the considered models, in particular focusing on the minority class which represents the illicit transactions, on the overall test set which consists of the graph snapshots from 36^{th} until 49^{th} timestamp. It is possible to notice from the results presented in Table 6.1.1 that GraphSAGE can achieve comparable results with the other considered ConvGNNs, and the F-1 score indicates that the GraphSAGE version with pool aggregator is preferable. Skip-GCN is the model with the highest F1-score, this is because it has been designed to consider along with the hidden representation as well the input node features at every

convolutional layer. Hence it considers more information than the others. However, the best performing model among the ones considered is Random Forest, as also highlighted by the paper's authors [43]. Random Forest is an ensemble learning model, which is better suited for this fine-grained Anomaly Detection.

Table 6.1.1: Graph Convolutional Network Prediction Performance

Model	Precision	Recall	F1-score
GCN	0.5158	0.5616	0.5377
Skip-GCN	0.6116	0.5749	0.5927
GraphSAGE mean	0.4738	0.5627	0.5145
GraphSAGE pool	0.6339	0.4650	0.5365
Random Forest (AF)	0.6767	0.6759	0.6763
Random Forest (AF + NE_{gcn})	0.8280	0.6360	0.7194
Random Forest (AF + $NE_{skip-gcn}$)	0.9171	0.6626	0.7693
Random Forest (AF + $NE_{gsage-mean}$)	0.8650	0.6615	0.7497
Random Forest (AF + $NE_{gsage-pool}$)	0.8947	0.6693	0.7657

Anyhow, the usage of Network Representation Learning model can be justified by the fact they are able to generate embeddings that can be used as features for further Anomaly Detection models. For instance, in Table 6.1.1, it is possible to notice that Random Forest achieves good performances leveraging only the input features, i. e. All Features (AF). But, its predictive performances increase considerably when it is trained considering the features in combination with Node Embeddings (NE) generated with the ConvGNNs models.

Besides, it is of interest to evaluate how the prediction performances vary over the test time window since the data set comprises a sequence of graph snapshots. Furthermore, the peculiarity of this data set is that it contains a very interesting example of concept drift which can be located at the 43^{th} timestep. From this precise timestep forward, all the considered models are not robust enough to maintain good prediction performances, see Figure 6.1.1. The only exception regards the 46^{th}, by analysing the prediction for this graph snapshot, it is possible to appreciate different prediction results. In this case, the best model is Random Forest that leverages the embeddings generated with GraphSAGE pool aggregator.

From the results of this experiment, it is possible to draw some conclusions. First of

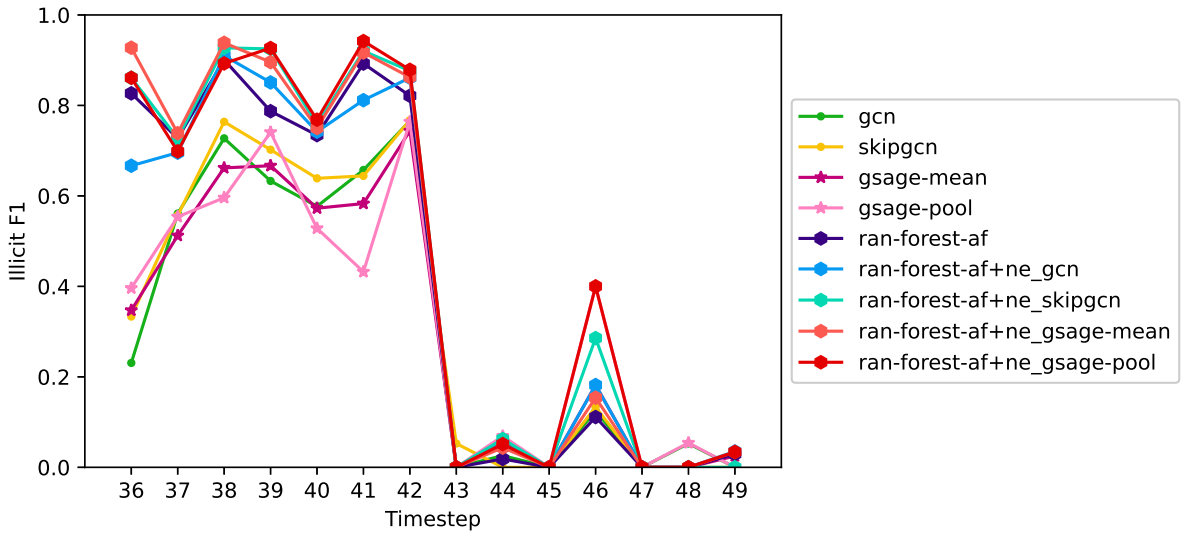


Figure 6.1.1: Illicit class F1-score over test timesteps span

all, leveraging Network Representation Learning can lead to better predictions. Still, it is necessary to use the generated embeddings in combination with input node features, since embeddings and features are not mutually exclusive but complementary. To perform Anomaly Detection it is better to use both these two representations to include as many different aspects as possible. The usage of GraphSAGE does not lead to significantly better prediction results than GCN-based models, but they are comparable. In fact, GraphSAGE embeddings can lead to better results when exploited by the Random Forest model rather than GCN, but considering the ConvGNNs model individually, GCN F-1 score is better. The most significant advantages of using GraphSAGE lie in the model design: GraphSAGE allows to relaxed assumptions on the input order, i.e. a fixed number of nodes, and its weights are dependent on the input features and hidden representation dimensions rather than the graph order.

6.1.2 EvolveGraphSAGE-O

Another principal aim of this experiment is to investigate if it is meaningful to train the selected Network Representation Learning model dynamically to capture the intrinsic dynamic nature of financial transactions graph. For this purpose, it has been implemented a 2-layered GraphSAGE dynamic version, namely EvolveGraphSAGE-O, in Pytorch [56] whose weights evolve at each timestamp accordingly with the sequence of graph snapshots given as input. Both the mean aggregator and the pool aggregator are considered in the experiment to appreciate if they have a different impact on the

prediction performances. The specif parameters used for this experiment are listed in Appendix A.1.2.

The baseline models selected to compare the prediction performances of the GraphSAGE dynamic variant with are EvolveGCN-O and EvolveGCN-H. These two models are ConvGNN dynamic frameworks proposed by the authors [26], previously described in Section 2.3.3.

The first results show the prediction performances of the considered models, on the overall test set consisting of the graph snapshots from 36^{th} until 49^{th} timestamp. In particular, the focus is on the minority class that represents illicit transactions. Even in these dynamic settings, it is possible to notice from the results in Table 6.1.2 that the EvolveGraphSAGE-O variant, which leverages the pool aggregator, can achieve better performances than the mean variant. The EvolveGraphSAGE-O models, compared to their static versions, exhibit a similar recall but with a worse precision. For this reason, the F1-scores of EvolveGraphSAGE-O variants are lower, anyhow all the ConvGNNs and STGNNs considered can identify a similar amount of illicit transactions but in general STGNNs exhibit higher false-positive rates. Considering the accuracy obtained with the baseline STGNNs models, namely EvolveGCN-H and EvolveGCN-O, it is evident that the EvolveGCN-H variant is significantly worse. For this reason, to implement the dynamic version of GraphSAGE we have focused on the EvolveGCN-O architecture, which leverages only the GCN weights at the previous timestep to update the GCN weights instead of considering as well the generated node embeddings. If we compare, EvolveGraphSAGE-O-pool with the EvolveGCN-O considered, we can notice that EvolveGraphSAGE-O accuracy is slightly worse but still able to detect anomalies with comparable results.

Another essential aspect to consider is how the prediction performances change at each timestep, see Figure 6.1.2. In general, we can observe that the concept drift penalizes even the dynamic models. Anyhow, we can observe some slight improvements by leveraging the STGNNs models. The accuracy of both EvolveGraphSAGE-O-pool and EvolveGCN-H does not drop to zero at the 43^{th} timestep. Instead, their F-1 scores are around 5% when the concept drift happened. With EvolveGraphSAGE-O-mean and Evolve-GCN-H, it is possible to observe a significant improvement in the accuracy at the 48^{th} timestep. In conclusion, we can say that the main advantage of using STGNNs is that they retain only the information of the most recent snapshots to make

Table 6.1.2: Graph Convolutional Network Prediction Performance

Model	Precision	Recall	F1-score
GCN	0.5158	0.5616	0.5377
GraphSAGE mean	0.4738	0.5627	0.5145
GraphSAGE pool	0.6339	0.4650	0.5365
EvolveGCN-H	0.3099	0.5438	0.3948
EvolveGCN-O	0.4910	0.5749	0.5297
EvolveGSAGE-O mean	0.3388	0.5272	0.4125
EvolveGSAGE-O pool	0.3944	0.5516	0.4600

predictions. This can be beneficial in the presence of concept drift since it is necessary to forget previous anomalous patterns that are not anymore present.

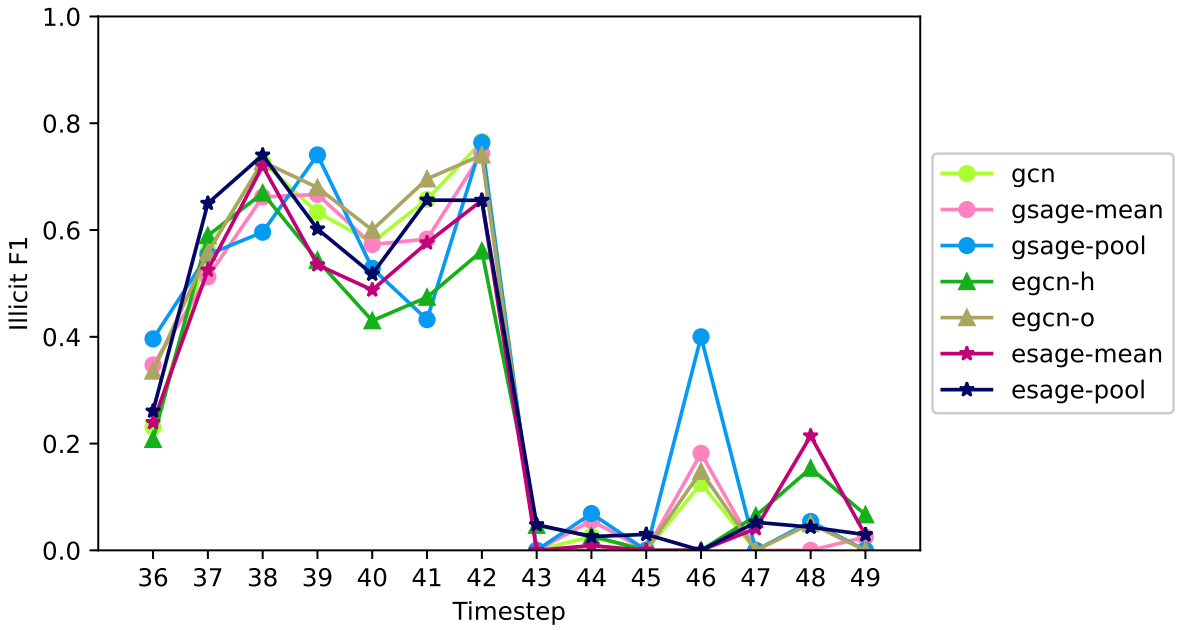


Figure 6.1.2: Illicit class F1-score over test timesteps span

6.2 Results on the Bank Data Set

This section presents the results obtained by using the GraphSAGE version specifically designed to handle bipartite graphs implemented in TensorFlow [54]. This architecture is trained by optimizing an auxiliary unsupervised task to encode node features and graph topology information. The parameters used for this experiment and the specification regarding the computational platform and software libraries used are

listed in Appendix A.1.3 and Appendix A.2.2 respectively.

The GraphSAGE model is used to generate the embeddings which are later fed to Anomaly Detection models, specifically the models considered are Isolation Forest, LSTM and Wavenet. Isolation forest has been used with the intent to perform Anomaly Detection in near real-time. In fact, as soon as new transactions from March onwards are processed, predicting anomalous patterns turns out to be rather straightforward. On the other hand, the recurrent neural network models are used to perform a more accurate evaluation considering the entire period offline.

6.2.1 Results on the Training Sub-Graph

The first results reported consider the prediction performance over the sub-graph which has been selected to train the NRL model. Even if the most interesting results regards the entire graph, they have been considered in order to understand if the NRL model is able to generalize over unseen graph entities.

For the sake of providing a comprehensive overview, the anomaly Detection models are provided different inputs:

- **Node Features** - these are the features extracted in the pre-processing which are also fed in the NRL model
- **Node Embeddings Link Classification (LC)** - these are the embeddings generated using the GraphSAGE version which optimize the link classification task, i.e. predict whether if a transaction is in foreign currency or not
- **Node Features and Node Embeddings Link Classification (LC)** - these are the combination of node features and node embeddings for link classification
- **Node Embeddings Random Walks (RW)**- these are the embeddings generated using the GraphSAGE version which optimize the unsupervised task, i.e. predict whether a transaction really exists among two nodes which have been sampled through random walk or not
- **Node Features and Node Embeddings Random Walks (RW)** - these are the combination of node features and node embeddings for unsupervised training

The isolation forest algorithm has been trained in two different ways. The first training

uses as input data features and/or embeddings extracted in the first half of the time period representing normal behaviour of customers prior to the pandemic's onset in Europe. The second version, does not take into account previous behaviours, rather, the model is continuously updated by training it on the same features and/or embeddings obtained in the time window for which it provides a prediction.

Table 6.2.1: Classification Performance on the Anomalous Class - Small Graph

Input	Model	Precision	Recall	F1-score
Node Features	Isolation Forest	0	0	0
	Isolation Forest (weekly training)	0.0417	0.0015	0.0028
	LSTM	0.7207	0.6548	0.6862
	WaveNet	0.8333	0.6853	0.7521
Node Embeddings LC	Isolation Forest	0.0424	0.7155	0.0800
	Isolation Forest (weekly training)	0.0433	0.8200	0.0822
	LSTM	0.4176	0.3604	0.3869
	WaveNet	0.4331	0.3454	0.3842
Node Features and Embeddings LC	Isolation Forest	0.0576	0.0900	0.0702
	Isolation Forest (weekly training)	0.0802	0.2322	0.1192
	LSTM	0.7524	0.6798	0.7143
	WaveNet	0.7709	0.7005	0.7340
Node Embeddings RW	Isolation Forest	0.0620	0.8070	0.1152
	Isolation Forest (weekly training)	0.0526	0.8984	0.0994
	LSTM	0.5135	0.4822	0.4975
	WaveNet	0.5480	0.4924	0.5187
Node Features and Embeddings RW	Isolation Forest	0.2230	0.3599	0.2754
	Isolation Forest (weekly training)	0.1111	0.4906	0.1812
	LSTM	0.7196	0.6904	0.7047
	WaveNet	0.7874	0.6954	0.7385

When analysing the results in Table 6.2.1, it is necessary to distinguish between near real-time prediction and offline prediction. The near real-time prediction model which leverages the Isolation Forest model is able to display that the best results are achieved by using node embeddings and features in combination. In particular, the best strategy consists of using the embeddings generated to optimize the unsupervised task, with an overall F-1 score of 27.54%. Regarding the offline setting, the best performing model is WaveNet trained using only the node features as input. It is worth mentioning that the combination of node features and embeddings using WaveNet are able to achieve comparable results but do not bring a significant gain in prediction performance, just a slight increase in the recall, which means that leveraging embeddings allows to detect a larger number of anomalies, true positives, but with a higher false positive rate. Concerning the LSTM model, the usage of embeddings in combination with node features is beneficial in all cases, but the F-1 scores remain lower compared with the ones achieved with WaveNet.

6.2.2 Results on the Entire Graph

The second part of the experiment on the entire graph has been conducted similar to the previous one, the models have been trained in the exact same way. The only difference is the fact that the entire customer base has been taken into account.

As it is observable from the results reported in Table 6.2.2, the best prediction in near real-time is achieved by using node embeddings and features together, as in the training of sub-graphs. In particular, the best strategy resides in using the embeddings generated to optimize the link classification task, with an overall F-1 score of 15.45%. Regarding the offline evaluation, the best performing model is again WaveNet trained using only the node features as input. The combination of node features and embeddings using WaveNet are able to achieve comparable results but do not bring a significant gain in the prediction performance, only a slight increase in precision, which means that leveraging embeddings allows for decreasing the false positive rate.

Another aspect which must be considered is plausible changes in the quality of embeddings when generating them on the overall graph, and not only on the sub-graphs used for training. Overall, it is possible to notice a decrease in the prediction performance which is particularly evident when considering the nodes embeddings as

Table 6.2.2: Classification Performance on the Anomalous Class - Entire Graph

Input	Model	Precision	Recall	F1-score
Node Features	Isolation Forest	0.0323	0.0029	0.0053
	Isolation Forest (weekly training)	0.0157	0.0044	0.0068
	LSTM	0.7287	0.6954	0.7117
	WaveNet	0.6636	0.7310	0.6957
Node Embeddings LC	Isolation Forest	0.0114	0.7562	0.0225
	Isolation Forest (weekly training)	0.103	0.8302	0.0204
	LSTM	0.2168	0.1574	0.1824
	WaveNet	0.2500	0.1878	0.2145
Node Features and Embeddings LC	Isolation Forest	0.1012	0.3266	0.1545
	Isolation Forest (weekly training)	0.0310	0.3628	0.0571
	LSTM	0.6620	0.7157	0.6878
	WaveNet	0.7053	0.6802	0.6925
Node Embeddings RW	Isolation Forest	0.0145	0.8578	0.0286
	Isolation Forest (weekly training)	0.0120	0.8940	0.0237
	LSTM	0.3846	0.3807	0.3827
	WaveNet	0.4000	0.4162	0.4080
Node Features and Embeddings RW	Isolation Forest	0.0493	0.4253	0.0884
	Isolation Forest (weekly training)	0.0282	0.5965	0.0539
	LSTM	0.7182	0.6599	0.6878
	WaveNet	0.6789	0.6548	0.6667

the only input. In fact, the node embeddings computed for the link classification tasks allow to achieve a F-1 score around 20% using LSTM and WaveNet, which is half as large as the results achieved considering the sub-graph. When considering the nodes embeddings in combination with features, nodes embeddings for link classification

lead to the best performances when using Isolation Forest.

It is evident from the results reported in Table 6.2.3 that using embeddings for link classification leads to a lower false positive rate using both the windowing strategies.

Table 6.2.3: Classification Performance on the Anomalous Class - Entire Graph

Input	Model	Window	Precision	Recall	F1-score
Node Features and Embeddings LC	Isolation Forest	Tumbling	0.1012	0.3266	0.1545
	Isolation Forest	Sliding	0.1403	0.1263	0.1329
Node Features and Embeddings RW	Isolation Forest	Tumbling	0.0493	0.4253	0.0884
	Isolation Forest	Sliding	0.0554	0.6328	0.1019

To summarise, the experiment results are encouraging but they do not conclusively state that leveraging node embeddings for this particular use case yields better accuracy. The following conclusions can be drawn:

- Regarding the offline evaluation, i.e. when using RNN models to analyse the customer behaviour over the entire period, it is sufficient to consider the node features as input. Leveraging embeddings in combination with node features does not lead to better accuracy. This can be due to the fact that topological information encoded into embeddings can be a bit noisy for this particular task. In fact, making transactions in foreign currencies does not necessarily mean that a person is abroad, and vice versa: it is possible to make transactions in your own currency while staying abroad. In addition, the characteristics of the bipartite graph further complicate the underlying graph topology.
- Regarding the online evaluation, i.e. when considering customers' spending behaviours week by week, it is not possible to detect almost any anomaly by analysing only the node features. In this case, combining features with node embeddings allows for increasing the predictions but the F-1 score does not perform optimally, around 15%. Nevertheless, a lower accuracy can be justified considering that node labels have been assigned by assessing changes in number of transactions in foreign currencies. On the other hand, in the real-time setting, spending behaviours are considered over a time window of one week.

- In general, it is possible to notice that using the unsupervised task, which encodes nodes which belong to the same context obtained with random walks similarly, allows for extracting more meaningful node embeddings. We also observe that although embeddings generated for link classification are not very informative when considered alone, they still yield comparable accuracy with the embeddings generated to optimize the other unsupervised task when used in combination with node features.

Chapter 7

Conclusions

7.1 Discussion

In this section are presented the drawn conclusions answering the research questions presented in the Introduction.

Among the existing network representation learning approaches available in the literature, which solutions are best suited for financial transaction data?

The qualitative analysis indicates ConvGNNs as the most suitable category of Network Representation Learning approaches able to satisfy most of the requirements needed for this particular application domain.

ConvGNNs are meant to leverage nodes features, which in this case include relevant information regarding the financial behaviour of individual and legal entities that use financial system. Some of the approaches within this category, such as GraphSAGE, are inductive that consent them to generalize even on unseen data, and the number of parameters is not linearly dependent with the graph order; instead it is only dependant on the dimensionality of the input nodes features. Also, ConvGNNs inherit by Deep Neural Networks, in general, the flexibility to adjust the architecture according to the specific requirements of use cases. For example, they can be modified to handle heterogeneous graphs, and additional layers can be stacked to directly learn how to classify the generated embeddings and jointly optimize the entire network.

Even if the other Network Representation Learning approaches have not been studied in-depth in this thesis, they are essential to understand the rationale behind NRL.

For example, the analysis of the Skip-gram model with negative sampling allows us to understand better how to train ConvGNNs in an unsupervised manner. In fact, unsupervised ConvGNNs as well leverages random walk to extract samples, and they consider the same loss function.

Can graph embeddings improve the prediction performance in anomaly detection over the traditional statistical methods used in finance?

Of course, there is no silver bullet, and leveraging graph embeddings can not always lead to better prediction performances. The answer profoundly depends on the data set considered and the types of anomalies. It is possible to answer with confidence only regarding the two different real-world data sets considered in this project. In the case of AML on the Bitcoin Network, it is demonstrated that leveraging node embeddings permits to improve the prediction performances. It must be noticed that the best results can be achieved using an auxiliary Anomaly Detection algorithm, namely Random Forest, which takes an input both the input node features and the embeddings. This means that nodes embeddings are not substitutes of input node features, but they are complementary and able to capture different aspects of the data set. In the case of detecting customers in distress, the generated embeddings do not provide a significant improvement when analysing the customers' behaviour offline over the entire period taken into consideration. Instead, analysing week by week the customer behaviours included in the node features and embeddings, it is possible to notice that the combination of the two allows to detect some of the anomalies, but the false positive rate is high. Still, it is an improvement because only considering features was not possible to detect almost anything. The solution needs to be further improved before having results aligned with industry standards.

As a rule of a thumb, we can infer that when the anomalies are dependant as well to the graph structure, graph embeddings are likely to provide meaningful insights which can be exploited for Anomaly Detection. In particular, it must be considered, which are the properties encoded within the embeddings and if they can provide useful insights. In our case, ConvGNNs aggregate the information of nearby nodes in the neighbourhood, and the information coming from this nearby could lead to misleading results if nodes context it is not informative for the anomalies.

Is it possible to leverage network representation learning to reflect the evolving nature of financial data?

Even in these dynamic settings, it is possible to turn financial data mining into a network representation learning problem. For instance, Spatial-Temporal Convolutional Networks represent a valid category of approaches that can model graph topological information and how it evolves.

This type of Graph Neural Networks can be easily modified to model adequately the underlying evolving graphs; indeed, in the literature there exist a variety of different approaches. For these reasons, it is crucial to understand the characteristics of the graph we want to model and have clear in mind, or at least an idea, of how it will evolve.

In this work, the experiment is conducted on the Elliptic data set. This data set has the main limitation of having a new nodes set for each new graph snapshot, i.e. the nodes set is always changing completely. Because of that, the research focused on STGNNs that do not resort to node embeddings for learning the system dynamics. Instead, we focused on learning how to evolve the weights of a Graph Convolutional Network through a RNN architecture. Specifically, we have used GraphSAGE as ConvGNN to model the spatial information, and its weights are adapted accordingly with the input sequence of graph snapshots using a GRU architecture modelled in Sequence-to-Vector manner.

The prediction results in terms of F-1 Score are not as good as the results achieved using ConvGNN trained statically. Anyhow, the evolving models have the advantage of being more robust in the presence of concept drift. The higher F-1 Scores on some timesteps after the concept drift are achieved with the proposed method.

7.2 Future Work

Deep Learning represents a fast-growing Machine Learning area, leveraging deep models consent to achieve human-level prediction performances, but their main limitation is due to the fact they work like in a black-box fashion [57].

Before deploying a Deep Learning model for critical applications such as raising the suspect against an individual for illicit behaviour, it is not sufficient to assess the predictive performances of a model quantitatively. Before using Deep Learning models in critical decision-making processes, you should make sure to be able to positively reply to the following questions: *Can you justify the predictions of your Deep Learning*

model? If so, are your justifications legally and ethically defensible?

Answering to the previous questions was out of the scope for this degree project, but in this section, it is made reference to relevant works, which can be used as a starting point for future work to comprehend better how ConvGNNs make predictions.

In this paper [58], it is presented a theoretical framework to analyze the discriminative power of ConvGNNs, i.e. their ability to distinguish among different graph structures.

The insights are generalized from a mathematical perspective by proceeding from the assumption that each node has a multi-set comprising the feature vectors of the node's neighbours. Such multi-sets are then aggregating according to the aggregation function specified by the considered ConvGNN framework. The aggregation function is the core component of ConvGNN, which is used to incorporate the information regarding the graph structure, i.e. each node representation is updated iteratively by aggregating neighbours' representations.

The authors state that this category of GNNs exhibits a strong representational power when the model is able to aggregate different multi-sets into different representations. In other words, GNNs are as powerful as their aggregation functions are discriminative.

In particular, the drawn conclusions regarding the discriminative power of *max* and *pool* aggregators, which are of extreme interests for this research. The mean aggregator is suitable when there is the need to encode distributional and statistical information in the embeddings rather than encode the exact graph structure. Its discriminative power is limited since is not able to distinguish among two multi-sets with same elements in the sets but in different numbers, i.e. if the first multi-set contains n times the same elements of the second multi-set. Anyhow, exploiting the mean aggregator can lead to satisfactory results to perform node classification if there are high-quality input node features aligned with the classification task.

Similarly, the pool aggregator it is not able to identify the exact graph structures. Anyhow, it is particularly suitable to identify particular elements in the multi-set that consents to have at least an idea of the graph structure backbone. This aggregator is preferable when embeddings robust to noise are required.

A further step towards explainability for GNNs has been made with the work presented

in this paper [59]. The authors presented an approach that can shed the lights over the predictions made by GNNs called GNNExplainer. This framework leverages the concept of *mutual independence* to trace the subgraph, and the node features subset which contributes the most for the predictions given a trained GNN model and the input data set. Indeed, the mutual independence has been specifically defined to quantify how the probability of obtaining a particular prediction for a node v, e.g. the class predicted if the network has been trained for classifying nodes, varies accordingly with the considered subgraph and node features subset. This becomes an optimization problem, and the objective is to maximize the mutual independence. Even if it is not possible to provide a convexity assumption to this optimization problem, the authors were able to produce good explantations for their experiments, and often the optimization reached a local minimum.

Bibliography

[1] Kipf, Thomas and Welling, Max. “Variational Graph Auto-Encoders”. In: *ArXiv* abs/1611.07308 (2016).

[2] Chandola, Varun, Banerjee, Arindam, and Kumar, Vipin. “Anomaly Detection: A Survey”. In: *ACM Comput. Surv.* 41.3 (July 2009). ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: https://doi.org/10.1145/1541880.1541882.

[3] Weber, Mark et al. “Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics”. In: *arXiv preprint arXiv:1908.02591* (2019).

[4] Jiang, J. et al. “Anomaly Detection with Graph Convolutional Networks for Insider Threat and Fraud Detection”. In: *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*. 2019, pp. 109–114.

[5] Akoglu, Leman, Tong, Hanghang, and Koutra, Danai. “Graph-based Anomaly Detection and Description: A Survey”. In: *CoRR* abs/1404.4679 (2014). arXiv: 1404.4679. URL: http://arxiv.org/abs/1404.4679.

[6] Perozzi, Bryan, Al-Rfou, Rami, and Skiena, Steven. “DeepWalk: Online Learning of Social Representations”. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 701–710. ISBN: 9781450329569. DOI: 10.1145/2623330.2623732. URL: https://doi.org/10.1145/2623330.2623732.

[7] Hamilton, William L., Ying, Rex, and Leskovec, Jure. *Representation Learning on Graphs: Methods and Applications*. 2017. arXiv: 1709.05584 [cs.SI].

[8] Menon, Aditya Krishna and Elkan, Charles. “Fast Algorithms for Approximating the Singular Value Decomposition”. In: *ACM Trans. Knowl. Discov. Data* 5.2 (Feb. 2011). ISSN: 1556-4681. DOI: 10.1145/1921632.1921639. URL: https://doi.org/10.1145/1921632.1921639.

[9] Ou, Mingdong et al. “Asymmetric Transitivity Preserving Graph Embedding”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1105–1114. ISBN: 9781450342322. DOI: 10.1145/2939672.2939751. URL: https://doi.org/10.1145/2939672.2939751.

[10] Henderson, Keith et al. “RolX: Structural Role Extraction & Mining in Large Graphs”. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’12. Beijing, China: Association for Computing Machinery, 2012, pp. 1231–1239. ISBN: 9781450314626. DOI: 10.1145/2339530.2339723. URL: https://doi.org/10.1145/2339530.2339723.

[11] Li, Jundong et al. “Attributed Network Embedding for Learning in a Dynamic Environment”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM ’17. Singapore, Singapore: Association for Computing Machinery, 2017, pp. 387–396. ISBN: 9781450349185. DOI: 10.1145/3132847.3132919. URL: https://doi.org/10.1145/3132847.3132919.

[12] Grover, Aditya and Leskovec, Jure. “Node2vec: Scalable Feature Learning for Networks”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 855–864. ISBN: 9781450342322. DOI: 10.1145/2939672.2939754. URL: https://doi.org/10.1145/2939672.2939754.

[13] Gao, M. et al. “Learning Vertex Representations for Bipartite Networks”. In: *IEEE Transactions on Knowledge and Data Engineering* (2020), pp. 1–1.

[14] Wu, Z. et al. “A Comprehensive Survey on Graph Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2020), pp. 1–21.

[15] Hamilton, William L., Ying, Rex, and Leskovec, Jure. “Inductive Representation Learning on Large Graphs”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1025–1035. ISBN: 9781510860964.

[16] Håkansson, Anne. “Portal of research methods and methodologies for research projects and degree projects”. In: *The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing WORLDCOMP 2013; Las Vegas, Nevada, USA, 22-25 July*. CSREA Press USA. 2013, pp. 67–73.

[17] Henderson, Keith et al. “It’s Who You Know: Graph Mining Using Recursive Structural Features”. In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’11. San Diego, California, USA: Association for Computing Machinery, 2011, pp. 663–671. ISBN: 9781450308137. DOI: 10.1145/2020408.2020512. URL: https://doi.org/10.1145/2020408.2020512.

[18] Lee, Daniel D. and Seung, H. Sebastian. “Algorithms for Non-Negative Matrix Factorization”. In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. NIPS’00. Denver, CO: MIT Press, 2000, pp. 535–541.

[19] Rissanen, J. “Paper: Modeling by Shortest Data Description”. In: *Automatica* 14.5 (Sept. 1978), pp. 465–471. ISSN: 0005-1098. DOI: 10.1016/0005-1098(78)90005-5. URL: https://doi.org/10.1016/0005-1098(78)90005-5.

[20] Stewart, G. W. *Matrix Perturbation Theory*. 1990.

[21] Mikolov, Tomas et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[22] Mikolov, Tomas et al. “Distributed Representations of Words and Phrases and Their Compositionality”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 3111–3119.

[23] Tang, Jian et al. “LINE: Large-Scale Information Network Embedding”. In: *Proceedings of the 24th International Conference on World Wide Web*. WWW ’15. Florence, Italy: International World Wide Web Conferences Steering

Committee, 2015, pp. 1067–1077. ISBN: 9781450334693. DOI: 10.1145/2736277.2741093. URL: https://doi.org/10.1145/2736277.2741093.

[24] Duvenaud, David et al. “Convolutional Networks on Graphs for Learning Molecular Fingerprints”. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’15. Montreal, Canada: MIT Press, 2015, pp. 2224–2232.

[25] Kipf, Thomas N and Welling, Max. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[26] Pareja, Aldo et al. “Evolvegcn: Evolving graph convolutional networks for dynamic graphs”. In: *arXiv preprint arXiv:1902.10191* (2019).

[27] Orhan, A. Emin. “Skip Connections as Effective Symmetry-Breaking”. In: *ArXiv* abs/1701.09175 (2017).

[28] Ying, Rex et al. “Graph Convolutional Neural Networks for Web-Scale Recommender Systems”. In: *CoRR* abs/1806.01973 (2018). arXiv: 1806.01973. URL: http://arxiv.org/abs/1806.01973.

[29] Hamilton, William L. et al. *WWW-18 Tutorial - Representation Learning on Networks*. The Web Conference, 2018 (WWW). Lyon, France, 2018. URL: http://snap.stanford.edu/proj/embeddings-www/index.html#bio.

[30] Wang, Daixin, Cui, Peng, and Zhu, Wenwu. “Structural Deep Network Embedding”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1225–1234. ISBN: 9781450342322. DOI: 10.1145/2939672.2939753. URL: https://doi.org/10.1145/2939672.2939753.

[31] Cho, Kyunghyun et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[32] Hochreiter, Sepp and Schmidhuber, Jürgen. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[33] Aggarwal, Charu C. *Recommender Systems: The Textbook*. 1st. Springer Publishing Company, Incorporated, 2016. ISBN: 3319296574.

[34] Rossi, Ryan A. et al. “From Community to Role-based Graph Embeddings”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* (2019).

[35] Akoglu, Leman, McGlohon, Mary, and Faloutsos, Christos. “oddball: Spotting Anomalies in Weighted Graphs”. In: *PAKDD*. 2010.

[36] Brin, Sergey and Page, Lawrence. “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. In: *Proceedings of the Seventh International Conference on World Wide Web 7*. WWW7. Brisbane, Australia: Elsevier Science Publishers B. V., 1998, pp. 107–117.

[37] Tong, Hanghang and Lin, Ching-Yung. “Non-Negative Residual Matrix Factorization with Application to Graph Anomaly Detection”. In: *Proceedings of the 2011 SIAM International Conference on Data Mining*, pp. 143–153. DOI: 10.1137/1.9781611972818.13. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611972818.13. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972818.13.

[38] Koutra, Danai et al. “Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms”. In: *Proceedings of the 2011th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*. ECMLPKDD’11. Athens, Greece: Springer-Verlag, 2011, pp. 245–260. ISBN: 9783642237829. DOI: 10.1007/978-3-642-23783-6_16. URL: https://doi.org/10.1007/978-3-642-23783-6_16.

[39] Rossi, Ryan et al. “Role-Dynamics: Fast Mining of Large Dynamic Networks”. In: *Proceedings of the 21st International Conference on World Wide Web*. WWW ’12 Companion. Lyon, France: Association for Computing Machinery, 2012, pp. 997–1006. ISBN: 9781450312301. DOI: 10.1145/2187980.2188234. URL: https://doi.org/10.1145/2187980.2188234.

[40] Sun, Jimeng et al. “Less is More: Sparse Graph Mining with Compact Matrix Decomposition”. In: *Stat. Anal. Data Min.* 1.1 (Feb. 2008), pp. 6–22. ISSN: 1932-1864.

[41] Gupta, Manish et al. “Integrating Community Matching and Outlier Detection for Mining Evolutionary Community Outliers”. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’12. Beijing, China: Association for Computing Machinery, 2012, pp. 859–867. ISBN: 9781450314626. DOI: 10.1145/2339530.2339667. URL: https://doi.org/10.1145/2339530.2339667.

[42] Rossi, Ryan A. et al. “Modeling Dynamic Behavior in Large Evolving Graphs”. In: *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*. WSDM ’13. Rome, Italy: Association for Computing Machinery, 2013, pp. 667–676. ISBN: 9781450318693. DOI: 10.1145/2433396.2433479. URL: https://doi.org/10.1145/2433396.2433479.

[43] Liaw, Andy, Wiener, Matthew, et al. “Classification and regression by randomForest”. In: *R news* 2.3 (2002), pp. 18–22.

[44] Liu, Fei Tony, Ting, Kai Ming, and Zhou, Zhi-Hua. “Isolation-Based Anomaly Detection”. In: *ACM Trans. Knowl. Discov. Data* 6.1 (Mar. 2012). ISSN: 1556-4681. DOI: 10.1145/2133360.2133363. URL: https://doi.org/10.1145/2133360.2133363.

[45] Yin, Wenpeng et al. “Comparative study of cnn and rnn for natural language processing”. In: *arXiv preprint arXiv:1702.01923* (2017).

[46] Olah, Christopher. *Understanding LSTM Network*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[47] Oord, Aäron van den et al. “WaveNet: A Generative Model for Raw Audio”. In: *Arxiv*. 2016. URL: https://arxiv.org/abs/1609.03499.

[48] Nakamoto, Satoshi. “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: *Cryptography Mailing list at https://metzdowd.com* (Mar. 2009).

[49] LTD, R3. *Description of the Unspent Transaction Output used in Distributed Ledger*. 2020. URL: https://docs.corda.net/docs/corda-os/4.4/key-concepts-transactions.html (visited on 06/02/2020).

[50] Harrigan, M. and Fretter, C. “The Unreasonable Effectiveness of Address Clustering”. In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People,*

and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld). 2016, pp. 368–373.

[51] Gama, João et al. “A Survey on Concept Drift Adaptation”. In: *ACM Comput. Surv.* 46.4 (Mar. 2014). ISSN: 0360-0300. DOI: 10.1145/2523813. URL: https://doi.org/10.1145/2523813.

[52] Leskovec, Jure et al. “Kronecker Graphs: An Approach to Modeling Networks”. In: *J. Mach. Learn. Res.* 11 (Mar. 2010), pp. 985–1042. ISSN: 1532-4435.

[53] Pedregosa, F. et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[54] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[55] Gron, Aurlien. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O’Reilly Media, Inc., 2017. ISBN: 1491962291.

[56] Paszke, Adam et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[57] Xie, Ning et al. “Explainable deep learning: A field guide for the uninitiated”. In: *arXiv preprint arXiv:2004.14545* (2020).

[58] Xu, Keyulu et al. “How powerful are graph neural networks?” In: *arXiv preprint arXiv:1810.00826* (2018).

[59] Ying, Zhitao et al. “Gnnexplainer: Generating explanations for graph neural networks”. In: *Advances in neural information processing systems*. 2019, pp. 9244–9255.

[60] LLC, Google. *Colaboratory*. URL: https://colab.research.google.com/notebooks/intro.ipynb.

[61] Wang, Minjie et al. “Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019). URL: https://arxiv.org/abs/1909.01315.

[62] Data61, CSIRO's. *StellarGraph Machine Learning Library*. `https://github.com/stellargraph/stellargraph`. 2018.

[63] Chollet, Francois et al. *Keras*. 2015. URL: `https://github.com/fchollet/keras`.

Appendix - Contents

Appendix A

Appendix

A.1 Models' Parameters

A.1.1 GraphSAGE - Elliptic Data Set

The specific parameters used for training GraphSAGE on the Elliptic Data Set are listed in the following Table A.1.1.

Table A.1.1: GraphSAGE parameters: Elliptic Data Set

Parameter	Value
Embeddings dimension	$d = 256$
Number of layers	$L = 2$
Number of Samples at each layer	$S_1 = 10, S_2 = 25$
Loss Function	$Crossentropy$
Optimizer	$Adam$
Learning Rate	0.001
Batch size	32
Epochs	1000
Class weights	$Illicit\ Class = 0.7$ and $Licit\ Class = 0.3$
Train-Validation-Test split	$60 - 10 - 30\%$

A.1.2 EvolveGraphSAGE-O - Elliptic Data Set

The specific parameters used for training EvolveGraphSAGE-O on the Elliptic Data Set are listed in the following Table A.1.2.

Table A.1.2: EvolveGraphSAGE-O parameters: Elliptic Data Set

Parameter	Value
Embeddings dimension at each layer	$d_1 = d_2 = 256$
Number of layers	$L = 2$
Loss Function	$Crossentropy$
Optimizer	$Adam$
Learning Rate	0.001
Epochs	1000
Class weights	$Illicit\ Class = 0.7$ and $Licit\ Class = 0.3$
Train-Validation-Test split	$60 - 10 - 30\%$
Number of previous graph snapshots taken as input	5

A.1.3 GraphSAGE for Bipartite Graph - Swedbank AB Data Set

The specific parameters used for training the GraphSAGE variant meant for Bipartite Graph on the Swedbank AB Data Set are listed in the following Table A.1.3.

Table A.1.3: GraphSAGE for Bipartite Graph parameters: SwedBank Data Set

Parameter	Value
Embedding dimension at each layer	$d_1 = d_2 = 64$
Number of layers	$L = 2$
Number of samples at each layer	$S_1 = 10, S_2 = 5$
Loss function	$Cross\ entropy$
Optimizer	$Adam$
Learning rate	0.001
Epochs	10
Class weights	$Illicit\ class = 0.7$ and $Others = 0.3$
Train-Test split	$70 - 30\%$

A.2 Hardware and Tools

A.2.1 Hardware and Tools - Elliptic Use Case

Google Colaboratory [60] is the computational platform used to run the experiments for the Elliptic Use Case. It provides a pre-configured Python virtual environment and computational resources.

The experiments have been implemented using Python 3. The main software libraries

used to implement the models are PyTorch [56] and Deep Graph Library [61]. The former is an optimized tensor library specifically meant to develop deep learning models using Graphical Processing Unit (GPU)s and Central Processing Unit (CPU)s. The latter is a graph-specif library that facilitates the usage of deep learning on graph-structured data. As well, the well-known numpy, scipy, sklearn, and matplotlib Python libraries have been used.

The hardware and software tools specification are summarised in the following box.

Development environment for the Experiment on the Elliptic Data Set

```
Platform: Google Colab
GPU specification: NVIDIA Tesla K80 - CUDA version 10.1
RAM specification: 25 GB
HDD specification: 68 GB

Main Software Libraries
PyTorch: version 1.5.1 - CUDA version 10.1
Deep Graph Library: version 0.4.3
```

A.2.2 Hardware and Tools - Swedbank AB Use Case

The computational platform used to run the experiments for the Swedbank AB Use Case is the bank on-premises platform due to local regulations.

The experiments have been implemented using Python 3, in particular the most important software libraries used to implement the models are TensorFlow [54] and Stellargraph [62]. The former is a library specifically meant to develop Machine Learning models which provide high-level APIs, e.g. Keras [63], with eager execution. The latter is a graph-specific library which facilitates usage of deep learning on graph-structured data. Additionally, the well-known numpy, scipy, sklearn, and matplotlib Python libraries have been used.

The hardware and software tools specification are summarised in the following box.

Development environment for the Experiment on the Swedbank Data Set

Platform: Swedbank Cluster

GPU specification: Tesla V100-SXM2

RAM specification: 32 GB

HDD specification: 1 TB

Main Software Libraries

TensorFlow: version 1.14.0

StellarGraph: version 1.0.0