ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing

# Graph Neural Networks in Finance

Semester Thesis

Kirill Meisser

kushakov@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

**Supervisors:**
Lukas Faber, Béni Egressy, Pál András Papp
Prof. Dr. Roger Wattenhofer

January 11, 2021

# Abstract

Financial networks can find themselves in a fragile state and the 2007-2008 crisis was a big proof of that. Many institutions went bankrupt and thousands of people lost their jobs. In this thesis, we wanted to explore how default is formed, how it propagates in a financial network and how one can save such a network in the most efficient manner. To help us in this endeavour, we make use of GNNs and create different models that can predict default and the severity of default of actors in a financial network. The accuracy of these models is not as good as we had desired, yet they still have predictive power and show that GNNs can and do learn from financial networks.

# Contents

CHAPTER 1

# Introduction

In the past few years we have seen a rise in the deployment of Graph Neural Networks (GNNs) that have been used to successfully learn from graph-structured problems. These problems can be encountered in many diverse fields such as social networks, molecular chemistry and traffic prediction. In this thesis, we leverage the power of GNNs and focus on financial networks. We tackle the problem of default propagation in an interconnected network of financial institutions and the best way to save such a network by bailing out key players. The large number of relations and contracts in such a network makes this a difficult problem, in particular one can show that it is $\mathcal{NP}$-hard. A solution to such a problem would help policymakers make the right decisions where the answer is complex and requires many considerations.

## 1.1 Motivation

The financial system can find itself in times of trouble as can be witnessed throughout history, with the latest example being the financial crisis of 2007-2008. Where one of the big disruptions was the initial bankruptcy of Lehman Brothers which caused an unforeseen market plummet and cascade of further bankruptcies. Ultimately, the crisis wiped out many financial institutions.

During these times the Federal Reserve had difficult decisions to make. Like who to bailout and who to leave behind? These decisions are made based on a variety of factors where one of which is how effective the bailout will be in restoring back the financial network and the economy. This area is where we want to help find a solution by using graph theory and GNNs, so that policymakers can make better decisions that benefit the system as a whole.

## 1.2 Goal

In this thesis, we use GNNs and graph theory to train a model that can assign bailouts to defaulted banks and institutions so that the biggest possible portion of the network can be rescued. Such a system would try to avoid default propagation that causes a domino-like effect where the whole financial system is dragged into failure.

Chapter 2

# Financial Network Model

## 2.1 Financial network as a weighted directed graph

Following L.C.G. Rogers and L. C. G. Rogers and L. A. M. Veraart [1] we represent a financial network as a weighted directed graph. In this realisation the nodes represent financial institutions (such as banks, investment firms etc.) and each of them is assigned a unique *ID*. On the other hand the edges correspond to the relations that the banks have. These are unpaid debts, otherwise known as *liabilities* between banks. The amount of each transaction is represented as a weight that is assigned to each edge and these values can be represented in a matrix form called a *liability matrix*. Finally, each organization has its *net assets* that are available to them. In case of debt these can be used to pay off loans. We include this information as a node feature that is assigned to each of the institutions in the network. An example of such a model is illustrated in fig. 2.1 with the values of the assets assigned in the table. Now that we have a rough picture of how a financial network looks like, we can begin to formally define the *financial system* as follows [1].

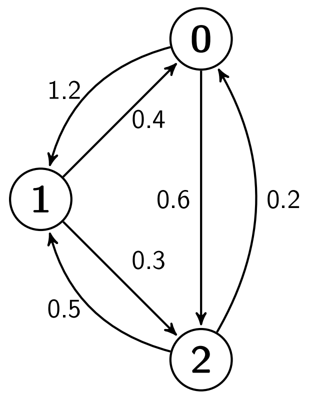
**Definition 2.1** (Liabilities Matrix)**.** The liabilities matrix is given by $\mathbf{L} \in \mathbb{R}^{n\times n}$, where the $ij$th entry $L_{ij}$ represents the nominal liability of bank $i$ to bank $j$. We assume that $L_{ij} \geq 0 \ \forall i, j$ and $L_{ii} = 0 \ \forall i$.

**Definition 2.2** (Obligations)**.** The total nominal obligations of bank $i$ to all other banks in the system are given by $\bar{L}_i = \sum_{j=1}^{n} L_{ij}$ and $\bar{L}$ is the corresponding vector of the total nominal obligations.

**Definition 2.3** (Relative Liabilities Matrix)**.** Let $\mathbf{L}$ be a liabilities matrix and $\bar{L}$ the corresponding vector of total nominal obligations. The *relative liabilities matrix* $\Pi \in \mathbb{R}^{n\times n}$ is defined by

$$\pi_{ij} := \begin{cases} L_{ij}/\bar{L}_i & , \ if \ \bar{L}_i \geq 0 \\ 0 & , \ otherwise \end{cases}$$

**Definition 2.4** (Net Assets)**.** We denote by $e_i \geq 0$ the *net assets* of bank $i$ from sources ouside the banking system. The corresponding vector of net assets is denoted by $e$.



| Node ID | Assets |
|---|---|
| 0 | 1.1 |
| 1 | 0.8 |
| 2 | 0.5 |

Figure 2.1: Simple example of a financial network

**Definition 2.5** (Financial System)**.** We can formally define the *financial system* as a quadruple $(\mathbf{L}, e, \alpha, \beta)$ where $\mathbf{L}$ is the liabilities matrix, $e$ are the net assets and $\alpha, \beta \in (0, 1]$ are two constants which represent the fraction of the face value of net assets realized on liquidation and the fraction of the face value of inter-bank assets realized on liquidation respectively. The first one takes into consideration that the abrupt sell-off of assets will cause that only a fraction $\alpha$ of assets can be recovered. While $\beta$ takes into consideration that early repayment of obligations to the bank in question may not be received in full because of an underlying discount.

Now that we have the main definitions out of the way we are going to introduce the *clearing vector*. A clearing vector determines payments between banks in the system by assuming some rules. These were originally proposed by Eisenberg and Noe [2] and are:

1. *Limited liabilities*: Each node can not pay more than its available cash flow.
2. *Priority of debt claims over equity*: Paying off the liabilities $L_{ij}$ has priority over everything else, even if the net assets $e_i$ have to be used for that.
3. *Proportionality*: In case of default, the defaulting bank pays all claimant banks in proportion to the size of their nominal claims on the assets of the defaulting bank.

Given these we can define the clearing vector as follows.

**Definition 2.6** (Clearing Vector)**.** A clearing vector for the *financial system* $(\mathbf{L}, e, \alpha, \beta)$ is a vector $L^* \in [0, \bar{L}]$ such that

$$L^* = \Phi(L^*)$$

where $\Phi$ is a function defined by

$$\Phi(L)_i \equiv \begin{cases} \bar{L}_i & , \; if \; \bar{L}_i \leq e_i + \sum_{j=1}^{n} L_j \pi_{ji} \\ \alpha e_i + \beta \sum_{j=1}^{n} L_j \pi_{ji} & , \; else \end{cases}$$

The clearing vector can be interpreted as the cash that each bank has available to pay out to other banks. For example $L_i^*$ represents the cash that bank $i$ has available to pay out to others. Thus the sum $e_i + \sum_{j=1}^{n} L_j^* \pi_{ji}$ represents the value of the total assets the bank $i$ has at its disposal. If this value exceeds $\bar{L}_i$, then bank $i$ is able to meet its obligations and is solvent. Otherwise, if this condition is not met then the bank is in default and it must use its assets to pay off its debt. Once the bank finds itself in default it must sell-off its assets and recall the obligations its owed from others. This process causes losses which were considered by including $\alpha$ and $\beta$ in the model, these losses amount to

$$(1-\alpha)e_i + (1-\beta)\sum_{j=1}^{n} L_j^* \pi_{ji}$$

this demonstrates how default can actually cause additional losses.

## 2.2 Greatest Clearing Vector Algorithm (GA)

In order to calculate the resulting *bank values* and the *insolvency sets* we will be using the *Greatest Clearing Vector Algorithm* which was also developed by L. C. G. Rogers and L. A. M. Veraart [1]. This algorithm is defined as follows.

**Definition 2.7** (Greatest Clearing Vector Algorithm (GA))**.** For a financial system $(\mathbf{L}, e, \alpha, \beta)$ the GA algorithm constructs a sequence $(\Lambda_j^{(\mu)})$ as follows (we use all the definitions as before).

1. Set $\mu = 0, \Lambda_j^{(0)} := \bar{L}$ and $\mathscr{I}_{-1}^{+} := \varnothing$.
2. For all nodes $i$, compute $\upsilon_i^{(\mu)} := \sum_{j=1}^{n} \Lambda_j^{(\mu)} \pi_{ji} + e_i - \bar{L}_i$
3. Define $\mathscr{I}_{\mu}^{+} := \{1 \leq i \leq n : \upsilon_i^{(\mu)} < 0\}$ the set containing all indices of insolvent banks, and $\mathscr{S}_{\mu}^{+} := \{1 \leq i \leq n : \upsilon_i^{(\mu)} \geq 0\}$, the set containing all indices of solvent banks.
4. If $\mathscr{I}_{\mu}^{+} \equiv \mathscr{I}_{\mu-1}^{+}$, terminate the algorithm.
5. Otherwise, set
$$\Lambda_j^{(\mu+1)} := \bar{L}_j, \; \forall j \in \mathscr{S}_{\mu}^{+}$$

and determine the remaining clearing payments by finding the unique solution to the system of linear equations

$$x_i = \alpha e_i + \beta\{ \sum_{j \in \mathscr{S}_\mu^+} \bar{L}_j \pi_{ji} + \sum_{j \in \mathscr{I}_\mu^+} x_j \pi_{ji} \} \ \forall i \in \mathscr{I}_\mu^+$$

and setting

$$\Lambda_i^{(\mu+1)} := x_i, \ \forall i \in \mathscr{I}_\mu^+$$

6. Set $\mu \to \mu + 1$ and go back to step 2.

When the algorithm has terminated, the vector $\Lambda^{(\mu)}$ is a clearing vector.

We call $\mathscr{I}_\mu^+$ the accumulative level-$\mu$ insolvency set and $\mathscr{S}_\mu^+$ the accumulative level-$\mu$ solvency set. Where the accumulative insolvency set increases in size at every iteration and has the following interpretation.

**Definition 2.8** (Accumulative Level-$\mu$ Insolvency Set)**.** An accumulative level-$\mu$ insolvency set is a set of nodes that contains the *IDs* of the banks that are insolvent once all the banks in the accumulative level-$(\mu - 1)$ insolvent set have defaulted.

From the accumulative level-$\mu$ insolvency set we can then derive the level-$\mu$ insolvency set which we will then use to categorize banks in respect to the severity of their default. We define these sets as follows.

**Definition 2.9** (Level-$\mu$ Insolvency Set)**.** A level-$\mu$ insolvency set is a set of nodes that contains *only* the *IDs* of the banks that become insolvent once all the level-$(\mu - 1)$ banks have defaulted. That is we can compute the set as

$$\mathscr{I}_\mu = \mathscr{I}_\mu^+ \setminus \mathscr{I}_{\mu-1}^+$$

In addition to the insolvency sets we further define the *bank value.*

**Definition 2.10** (Bank Value)**.** The bank value is calculated by taking the assets of the bank, summing up the liabilities that others are capable of paying towards the bank in question and subtracting the loans that need to be payed out. In other words we can write it as:

$$V = \Pi^\top L + e - \bar{L}$$

where $L$ is the clearing vector that we have calculated by using the GA algorithm. We get the bank value of bank $i$ by taking the $i$th value of the vector and we know if it is in default if $V_i < 0$ or solvent if $V_i \geq 0$.

## 2.3 Default propagation and insolvency levels

Having introduced the GA algorithm we can now calculate the insolvency sets of a given financial network. With it we can see how severely default propagation can damage the financial network and how deep the consequences can become when leaving the system be as is. In the following we will see two simple examples of hypothetical financial networks and the devastating effects of default propagation in action. For both examples we will be assuming $\alpha = 1$ and $\beta = 1$, meaning that we do not consider additional losses in case of a default. In figure 2.2 we can observe a small example of a financial network ($G_1$) with 3 banks. In this case we can witness how the bank with *ID* 1 has too many loans and becomes insolvent with its bank value equal to -2 at the beginning. This insolvency means that bank 0 will only receive a fraction of the money it was due. In fact, because bank 1 has $\bar{L}_1 = 5$ and the obligation towards bank 0 is 2 units of money, bank 0 will only receive $\frac{2}{5}$ of $L_1$. Where $L_1$ represents the cash that bank 1 has available to pay out others and is found by performing the GA algorithm. By not receiving all the money bank 0 was due, it too finds itself in a position of insolvency because of bank 1. To see the defaults more clearly we can write down the mathematical expressions for the bank values as

$$V_0 = e_0 + L_1\pi_{10} + L_2\pi_{20} - \bar{L}_0 = 0 + 2.5 \cdot 0.4 + 1 \cdot 0 - 2 = -1.0$$

$$V_1 = e_1 + L_0\pi_{01} + L_2\pi_{21} - \bar{L}_1 = 1 + 1 \cdot 0.5 + 1 \cdot 1 - 5 = -2.5$$

$$V_2 = e_2 + L_1\pi_{12} + L_0\pi_{02} - \bar{L}_2 = 10 + 1 \cdot 0.5 + 2.5 \cdot 0.6 - 1 = 11.0$$

where we can simplify it further by writing the bank values in the vector form

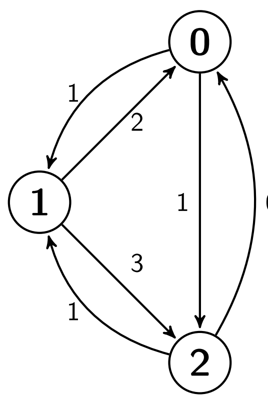
$$V_{G_1} = \begin{bmatrix} -1.0 \\ -2.5 \\ 11.0 \end{bmatrix} \tag{2.1}$$

As we can see the first two elements are negative, this indicates the insolvency of banks 0 and 1. Since bank 1 becomes insolvent without the influence of others, we call it level-0 insolvent. On the other hand, the default of bank 0 is dependent on whether or not bank 1 is solvent, thus per definition it is level-1 insolvent. It is curious to note that we could save this network entirely by bailing out bank 1 with 2.5 units of money.

The second example network ($G_2$) can be seen in figure 2.3. In this case bank 0 is a level-0 insolvent bank. Not being able to pay its liabilities in full it drags in bank 1 which becomes a level-1 insolvent bank. Now, bank 1 also does not manage to pay off all its debts and causes the default of bank 2 which now finds itself in the level-2 insolvency set. The final bank value vector that we obtain is:

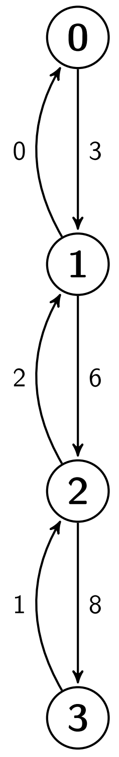
$$V_{G_2} = \begin{bmatrix} -2.0 \\ -1.25 \\ -1.25 \\ 106.0 \end{bmatrix} \tag{2.2}$$

In this case it is also good to note that by injecting bank 0 with only 2.0 units of money we can save all three banks.



| Node ID | Assets |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 10 |

Figure 2.2: Simple example graph $G_1$ where we have node 1 belonging to the level-0 insolvency set and node 0 belonging to level-1 insolvency set.



| Node ID | Assets |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |
| 3 | 100 |

Figure 2.3: Example graph $G_2$ where node 0 is the first to go insolvent, followed by node 1 (level-1 insolvent) and node 2 (level-2 insolvent)

Chapter 3

# Graph Neural Networks

In this chapter we are going to take a quick tour around graph neural networks, the heart of this thesis. We are going to look at the basis of these models, what the principles are that make them work and finally we introduce *NEConv* our custom convolutional layer.

## 3.1 Fundamentals

Graph neural networks (GNNs) are machine learning models that manage to model graph dependencies by using message passing between neighboring nodes and due to this property they have risen in popularity in the recent years [3].

In GNNs we have two types of features: node features $h_i^t$ and edge features $e_{ij}^t$, node features as the name suggests are assigned to nodes of the graph while edge features are assigned to edges. The node features initially describe the properties of the corresponding node they are linked to, but as we apply convolutional layers with message passing these node representations change and begin to contain information about the features of the neighboring nodes. These convolutions allow us to then perform different predictions such as: node classification/regression, graph classification/regression and link prediction.

## 3.2 Message Passing

The message passing paradigm is the backbone of GNNs and can be decomposed into two processing steps. The two steps are an edge-wise and a node-wise computation that is made every round, it looks as follows:

- Edge-wise:
$$m_e^{(t+1)} = \phi(h_v^{(t)}, h_u^{(t)}, w_e^{(t)}), \quad (u, v, e) \in \varepsilon$$
- Node-wise:
$$h_v^{(t+1)} = \psi(h_v^{(t)}, \rho(\{m_e^{(t+1)} : (u, v, e) \in \varepsilon\}))$$

In the above equations, $\varepsilon$ is the set of edges contained in the graph. Then $\phi$ is the *message function* which is defined on each edge. It generates a message by combining the edge feature with the features of its incident nodes. Meanwhile, $\psi$ is an *update function* defined on each node which is used to update the states. It uses the past node features and combines them with the aggregation of the neighbor messages in the *reduce function* $\rho$.

In other words, messages $m_e^{(t)}$ are generated on each edge and take into consideration the features of the neighboring nodes and edges. The nodes then aggregate these messages into a single representation and combine it with their own node representation to form newly updated node features $h_v^{(t+1)}$.

## 3.3 Convolutional Layers

Now that we have familiarized ourselves with the fundamentals and the message passing paradigm we can proceed to take a look at the different convolutional layers that were developed in the past literature which we found to be adapt to the problem we were trying to solve. In addition we introduce a convolutional layer that was specially developed for this thesis.

We began by using the NNConv [4] layer since it is one of the few layers that accounted for edge features and was a good place to start our experiments. We then switched to GraphConv [5] and SAGEConv [6] to see whether we could improve the predictive performance. While these two models are more complex, they only consider node features and thus ignore the information that is contained in the edges of our financial network. To this end we propose a new convolutional layer called NEConv which considers both node and edge features while being more complex and adapt than NNConv for the problem we were trying to solve.

**Definition 3.1** (NEConv)**.**

$$h_{\mathcal{N}(i)}^{t+1} = aggregate(\{FF(concat(e_{ij}, h_j^t)), \forall j \in \mathcal{N}(i)\})$$

$$h_i^{t+1} = \sigma(W \cdot concat(h_i^t, h_{\mathcal{N}(i)}^{t+1}))$$

The NEConv layer takes into consideration both the node features $h_j^t$ of the neighbors and the features of the incoming edges $e_{ij}$ of a given node $i$. It then propagates them through a feed-forward network (FF) and after an aggregation stores the representation in $h_{\mathcal{N}(i)}^{t+1}$. Finally, $h_{\mathcal{N}(i)}^{t+1}$ is concatenated with the node's own representation $h_i^t$ and is sent through a linear layer $W$ and an optional non linearity $\sigma$.

The idea behind NEConv is that by using both types of features we have more information at our disposal and the network would be able to learn better from it.

Chapter 4

# Procedure and Results

Before tackling the main goal of optimizing the bailout strategy given a financial network, we tried to solve less demanding problems. In this chapter we are going to look at the different prediction models we devised for these problems, how we generated the training and test data and finally how we trained and evaluated these models.

## 4.1 Data generation

In order to train and test any machine learning model we need access to data. Yet, for various reasons obtaining real-world data is not possible and thus we resorted to synthetic datasets. To do this we wrote a graph generator that generates random financial networks given the number of desired samples and parameters $\alpha$ and $\beta$. To generate connected graphs, we start with a randomly chosen spanning tree, to which we randomly add edges with uniform probability. In the implementation of the generator we decided to always add two anti-parallel edges between two nodes, which would indicate that both banks have a liability towards the other. One could argue that this must not always be the case and that is correct. We can solve this issue by setting the weight (loan) to zero for the desired edge, in doing so we can also represent one-way relations and maintain our two-way edge representation. The two-way edge representation is important to us because it enables us to have good message passing across the network, where every node can receive messages from its direct neighbors.

Now that we know how we generate the graph structure it is time to assign node features, edge weights and labels to have a complete data sample. We assign the node features and edge weights uniformly at random for each sample in order to represent random financial networks. Finally, we label each of the nodes depending on the task we want to accomplish, for example by using the clearing vector algorithm.

## 4.2 First model: Binary prediction of default

To begin getting to know GNNs and how they work we started to build a model that given the financial network, the loans and the assets would predict whether the bank is in default or not. Thus in this task we are only concerned with a binary classification. The classes being: insolvent (0) or solvent (1). For this task we are going to utilize a two dimensional node feature space that includes the assets and the outgoing liabilities of bank $i$, that means that we can write the features for each node as follows:

$$h_i^{(0)} = \left[assets\ of\ bank\ i\ ,\ outgoing\ liabilities\ of\ bank\ i\right]$$

As for the edge features we are just going to use the weights (loans) that are assigned to each edge. Furthermore we are going to use the NNConv convolutional layer as our model, where for the aggregation function we will take the sum of the incoming messages, for the mapping $f_\Theta$ we are going to use a linear transformation and finally a readout layer to transform the two dimensional node representations into a one dimensional output.

For our training and test data we use the previously discussed data generator. We train our model on 1000 generated graphs which all contain banks that reach level-3 insolvency, we do this by filtering them out during generation and storing them in a dataset. As for our test set we use another 50 graphs that again all contain banks that reach level-3 insolvency. All that is left to do now is train the model, we do this for 30 epochs and use binary cross entropy as our loss function. Finally we evaluate the performance by looking at the accuracy per insolvency level, Table 4.1 shows the results. Here we can notice how we achieve outstanding performance for level-0 insolvent banks and the solvent banks (shown as level -1 in the table). As for the higher-level insolvent banks we can see a noticeable deterioration of performance as we go higher. This was to be expected since we are using only one NNConv layer, in such a configuration each node considers only the information from his direct neighbors. In order to be able to predict higher level insolvencies in a more accurate manner we need to add layers. This is exactly what we are going to do in the next step.

| Level | Total Number of Samples | Total Correct Predictions | Total Incorrect Predictions | Accuracy |
|---|---|---|---|---|
| 0 | 659 | 658 | 1 | 0.998 |
| 1 | 282 | 197 | 85 | 0.699 |
| 2 | 87 | 32 | 55 | 0.368 |
| 3 | 53 | 26 | 27 | 0.491 |
| -1 | 877 | 838 | 39 | 0.956 |

Table 4.1: Performance of the one layer NNConv model using 50 evaluation graphs

## 4.3 Second model: Going deeper

In the last section we have built a classifier that categorizes banks into two classes: solvent and insolvent ones. We have seen how the model manages to predict well the level-0 insolvency or solvency of a given bank. Now what we want to achieve is a better performance in the higher level insolvencies and we hoped to accomplish that by adding more NNConv layers on top of the existing model. Precisely we attempted models with two to four NNConv layers and different architectural variations of these. In the end we found the best results to be achieved with three NNConv layers where the first two layers are exactly the same and share weights, while the third layer is a new one where on top of it we have a readout layer that converts the outputs to a single dimension so that we can perform classification.

In order to combat class imbalance between the solvent banks (of which there are much more) and the different levels of insolvent banks, we implemented a weighted binary cross entropy function so that every level has its corresponding weight assigned to it and thus its contribution appropriately considered. The weights for the different levels were calculated empirically by considering the proportion of the total number of nodes that the corresponding level represents and then taking the inverse of it. In other words we can write the weights in a vector as so:

$$w = \begin{bmatrix} w_0 & w_1 & w_2 & w_3 & w_{-1} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{(\frac{level\ 0\ nodes}{total\ nodes})} & \frac{1}{(\frac{level\ 1\ nodes}{total\ nodes})} & \frac{1}{(\frac{level\ 2\ nodes}{total\ nodes})} & \frac{1}{(\frac{level\ 3\ nodes}{total\ nodes})} & \frac{1}{(\frac{level\ -1\ nodes}{total\ nodes})} \end{bmatrix}$$

$$\approx \begin{bmatrix} 2.5 & 6.0 & 19.0 & 37.0 & 8.0 \end{bmatrix}$$

Where we have assigned a higher weight to the solvent banks because else the class of the insolvent banks was too dominant and the model started predicting only zeros.

Having setup our model we then train it on a data set of 3000 graphs that all contain banks with level-3 insolvency for five epochs. To evaluate our model we will again be using the same 50 graphs that were used before for the first model and in Table 4.2 you can see the corresponding accuracies for each level like before. By looking at these new results we can definitely see a big improvement for the higher level insolvencies, yet we can also witness some significant decline in performance of the solvent bank prediction. This may be likely due to the fact that the information regarding this is lost after propagating through 3 NNConv layers. Overall, we can say that adding more layers gave us a good performance bump which was to be expected given the deeper nature of this network.

| Level | Total Number of Samples | Total Correct Predictions | Total Incorrect Predictions | Accuracy |
|---|---|---|---|---|
| 0 | 659 | 659 | 0 | 1.000 |
| 1 | 282 | 274 | 8 | 0.972 |
| 2 | 87 | 82 | 5 | 0.943 |
| 3 | 53 | 48 | 5 | 0.906 |
| -1 | 877 | 653 | 224 | 0.745 |

Table 4.2: Performance of the three layer NNConv model using 50 evaluation graphs

## 4.4 Third model: Predicting the required bailout

For this last completed model we change course and modify our problem into a regression task. More precisely, we want to be able to predict how much money each bank would need in order to avoid insolvency. That is, how big the bailout has to be in order to elevate the bank's value above zero and thus turn it back operational. These values were calculated with the clearing algorithm previously discussed in Chapter 2, where by using the obtained clearing vector we can calculate the bank values of each node in the graph. Now for the banks with a positive bank value the label is set to zero since these banks are solvent and do not require any bailout. Meanwhile, for the banks with a negative bank value (insolvent banks) the label is created by taking the bank value and multiplying it by a minus. This positive number then represents the amount of money that needs to be injected into the failing bank.

Up until now in the previous models we have always used NNConv layers and at the beginning of this task an attempt was made to reuse the same architecture as in the last section. Unfortunately, such a model performed underwhelmingly and it was time to go back to the drawing board. After some thought we decided to change the underlying NNConv layer in favor of the SAGEConv layer. In contrast to NNConv where edge features and node features are considered, SAGEConv utilizes only node features and omits the edge ones. Because of that we needed to do some feature engineering and try to incorporate the data that is present in the edges into additional node features that can then be used by the SAGEConv based model. To do this we came up with different statistical measures such as the mean, max and the median of the ingoing and outgoing liabilities. These then formed the new 12 dimensional node representations.

Having our data in the right form we can begin to build our model and then commence with the training. For our model we take the SAGEConv layer as our main building block, but we modify it slightly by adding the sum operation as an aggregate function option. Many configurations were experimented with by varying the depth of the network, placing multi layer perceptrons (MLPs) in between the layers, using residual connections and concatenating different types of networks together. Two configurations performed the best in respect to the others. The first one being the 4 layer deep SAGEConv model with single hidden

| Level | Total Number of Samples | MSE |
|---|---|---|
| 0 | 403 | 0.0655 |
| 1 | 148 | 0.0886 |
| 2 | 58 | 0.0425 |
| 3 | 26 | 0.0381 |
| -1 | 338 | 0.0065 |

Table 4.3: Performance of the 4-layer SAGEConv model evaluated on 25 graphs

layer MLPs in between the layers and at the output. Meanwhile, the second one was built by concatenating different intermediate representations which were then fed to another MLP. By intermediate representations we mean the convoluted features which come out after each convolution with SAGEConv. Such a process could be seen as inserting residual connections where we capture the data and do not let it propagate further.

Now that we have the model we can begin training it. We use the same 3000 graph dataset as before but the node features and labels are modified in order to adapt to the new architecture and regression problem. For our cost function we use the mean squared error (MSE) as we want to penalize bigger deviations more heavily than smaller ones. In addition we will be using the early stopping mechanism, so that when the validation error stops decreasing or even begins to increase we stop learning to avoid overfitting. In Table 4.3 and 4.4 we can see the performance of the two respective models that were presented earlier. From the tables we can see how their performance is very similar and that the MSE is very small. Where the small MSE is largely due to the fact that the randomly assigned assets and loans are also quite small, with the assets being distributed uniformly at random between 0 and 1 and the loans being uniformly distributed between 0 and 2. This can give a distorted view of the results and in fact by checking with the fraction of variance unexplained (FVU) the given predictions against the labels we have noticed that the predictions for insolvency levels higher than 1 are not accurate and the FVU can go well above 1 which is very bad since an FVU of 1 would imply that we constantly predict the mean of the variable. In some instances the FVU climbs above 280 for level 2 predictions and 436 for level 3 predictions. Yet, while the predictions for the higher levels are not reliable the model manages to predict level-0 and level-1 insolvencies with a much better degree of accuracy. The FVU stays below 0.1 for the level-0 insolvencies and mostly below 0.5 for level-1 insolvencies with only two instances of it exceeding an FVU of 1.

| Level | Total Number of Samples | MSE |
|---|---|---|
| 0 | 403 | 0.0695 |
| 1 | 148 | 0.0857 |
| 2 | 58 | 0.0440 |
| 3 | 26 | 0.0368 |
| -1 | 338 | 0.0067 |

Table 4.4: Performance of the concatenated 4-layer SAGEConv network evaluated on 25 graphs

## 4.5 Benchmark model: comparing the results

To see how well our regression model really performs we decided to do a sanity check and compare the results from the third model to a very basic one. In such a basic model the label that we predict is done based on a simple computation. For it we take the net assets of the bank, we add to it the incoming liabilities from the other banks and subtract the liabilities the bank in question has towards others. We can also write it in the mathematical form as:

$$h_i = e_i + \sum_{j=1}^{n} \bar{L}_j \pi_{ji} - \bar{L}_i$$

Where the definitions of the different variables are the same as discussed in Chapter 2. Now that we have defined the model we can see how it performs on the evaluation set. For this we take the same set of 25 graphs as in the last section and calculate the MSE for the different levels of insolvency. The results can be found in table 4.5. We can observe how big the MSE is for the lower levels of insolvency and very small for the higher levels of insolvency. In addition the model labels the solvent banks with a 100% accuracy. At a first glimpse one might say that this model performs very well for solvent and higher level insolvency banks but the truth is quite different. In reality the fact is that the model does not consider at all the effects of default (e.g. fractional repayment of liabilities upon failure) and the propagation of the latter. This causes the big MSE values for the $0^{th}$ and $1^{st}$ levels. On the other hand, the very small MSE values of the higher two insolvency levels are given by the fact that the model assigns zero to all the banks that are above level-0 insolvency. This in combination with the true labels being close to zero gets us a very small error even though there is absolutely no predictive component to it.

In conclusion we can say that by comparing the benchmark model to the previously trained GNN models from the last section, the latter fared much better in fitting the first two insolvency levels. In contrast, the predictions for higher level insolvencies were poor in both cases.

| Level | Total Number of Samples | MSE |
|---|---|---|
| 0 | 403 | 1.4893 |
| 1 | 148 | 1.5027 |
| 2 | 58 | 0.0913 |
| 3 | 26 | 0.0018 |
| -1 | 338 | 0.0000 |

Table 4.5: Performance of the basic regression evaluated on the same 25 graphs

## 4.6 Further discussion on the regression task

For the regression task in the last section many other countless attempts were made to create a model with a superior predictive performance. We tried different convolutional layers such as GraphConv and our own custom layer NEConv but at the end to no avail. We hoped that by introducing edge features with NEConv we would get a performance boost but that was not the case and strangely the model that relied on SAGEConv fared much better in comparison to them. Later, we then sought to combine different convolutional layers such as the NNConv and SAGEConv by concatenating them and feeding them through an MLP but that also ended up performing poorly and was in the same accuracy range as the model with just NNConv layers. In conclusion we found it hard to beat the 4-layer SAGEConv model but probably with more time and experimentation this can be further improved.

Chapter 5

# Conclusion

In this semester thesis we used GNNs to learn on financial networks. We trained them on a randomly generated datasets which were labeled with the help of the greatest clearing vector algorithm. Next, we selected three models that had shown the best predictive capability and analyzed them. In these we reached some initial success by predicting whether a bank is solvent or not. Additionally, we managed to predict by how much a bank is insolvent with a decent accuracy measure for the lower levels of insolvency. In both problems we found there to be some limitations to the models. In the first case the accuracy for the solvent banks had decreased and left for some more to be desired. While for the second case the predictions for higher levels of insolvency produced unreliable results. Future work could try to find new architectures and tweak existing ones. Furthermore, the underlying convolutional layer can be potentially improved in many ways and engineered to be more compatible to the problem.

# Bibliography

[1] L. C. G. Rogers and L. A. M. Veraart, “Failure and rescue in an interbank network,” *MANAGEMENT SCIENCE*, vol. 59, 2013.

[2] L. Eisenberg and T. Noe, “Systemic risk in financial systems,” *MANAGEMENT SCIENCE*, 2001.

[3] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, “Graph neural networks: A review of methods and applications,” *arXiv:1812.08434v4*, 2019.

[4] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” *arXiv:1704.01212v2*, 2017.

[5] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv:1609.02907v4*, 2017.

[6] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *arXiv:1706.02216v4*, 2018.