UNIVERSITÉ LIBRE DE BRUXELLES

BRUFACE BRUSSELS FACULTY OF ENGINEERING

VRIJE UNIVERSITEIT BRUSSEL

Graph-based multimodal clinical data for patient prognosis

Is it possible to extract additional information from a population-based representation?

Maxime Bollengier

Master thesis submitted under the supervision of
Prof. Hichem Sahli

The co-supervision of
Dr. Abel Diaz Berenguer

In order to be awarded the Master's programme in
Electrical Engineering

Academic year
2022-2023

ULB UNIVERSITÉ LIBRE DE BRUXELLES

ULB UNIVERSITÉ LIBRE DE BRUXELLES

Exemplaire à apposer sur le mémoire ou travail de fin d'études, au verso de la première page de couverture.

Fait en deux exemplaires, **Bruxelles, le** 16/01/2023

Signature

Réservé au secrétariat :	Mémoire réussi*	OUI
		NON

CONSULTATION DU MEMOIRE/TRAVAIL DE FIN D'ETUDES

Je soussigné

NOM : Bellengier

PRENOM : Maxime

TITRE du travail : Graph-based multimodal clinical data for patient prognosis

AUTORISE*

~~**REFUSE***~~

la consultation du présent mémoire/travail de fin d'études par les utilisateurs des bibliothèques de l'Université libre de Bruxelles.

Si la consultation est autorisée, le soussigné concède par la présente à l'Université libre de Bruxelles, pour toute la durée légale de protection de l'œuvre, une licence gratuite et non exclusive de reproduction et de communication au public de son œuvre précisée ci-dessus, sur supports graphiques ou électroniques, afin d'en permettre la consultation par les utilisateurs des bibliothèques de l'ULB et d'autres institutions dans les limites du prêt inter-bibliothèques.

** Biffer la mention inutile*

** Biffer la mention inutile*

Formulaire version 04/2015

Abstract

Graph-based multimodal clinical data for patient prognosis by Maxime Bollengier, Université Libre de Bruxelles, 2022-2023.

Deep learning models have recently shown a lot of ability to process large amounts of data to make predictions. Specifically, the health domain has progressed through the implementation of these tools. In this domain, data related to a patient's medical information includes many types of data, such as data based on biomedical images, demographic information, symptoms, medication, hospital treatment, laboratory tests, etc... Successfully combining these multimodal data is a challenge to prognosticate a patient. In this thesis, it is proposed to extract information at two levels. A multimodal transformer capable of extracting information from different modalities and capturing their interactions at the patient level on one hand. On the other hand, as a main topic, several graph models are proposed to try to enrich the information extraction by representing the patients in a population graph. Several representation problems are presented and show that it is possible to improve the prediction results. Different ways to represent the interactions between patients are analyzed. Firstly, the interactions based on the definition of a similarity metric and the manual construction of a graph are used as representation. Then, an analysis of how to learn the graph structure using contrastive learning, attention-based modeling, and properties-based structure learning is presented. Finally, hypergraphs are used to model patient interactions in a higher level of representation, giving very encouraging results for extracting information at the population level.

Key words: Graphs, Hypergraphs, Prognosis, Multimodal, Deep learning, Representation learning.

Acknowledgements

First of all, I would like to express my gratitude to Prof. Hichem Sahli for trusting me and pushing me forward by always being honest and exigent with me. He constantly pushed me to my limits to try to extract the maximum of my abilities. He brought me far by constantly offering me opportunities. I would also like to express my gratitude to Dr. Abel Diaz Berenguer, who has been very involved in my learning. He constantly encouraged me and was a great source of motivation and inspiration for me. Despite his busy schedule, he always found time for me. I would also like to acknowledge Dr. Matias Bossa, who helped me comprehend the Bayesian paradigm and the scientific world. He has always been very concerned with my work and has always offered me detailed and pertinent teaching whenever I needed it. He transmitted his passion and his optimism allowing me to enjoy every aspect of the project. These last three mentors have really contributed to making me better and more capable. This fruitful team spirit is fabulous, and I wish anyone could work under these conditions.

Throughout my progression, I have had my ups and downs. Despite everything, my loved ones have always been there to support me and believe in me. I want to thank Ines, who has always been at the forefront and my source of happiness when work overwhelmed me. Also, my parents Stephane and Sandrine who, never stopped believing in me and always put everything they could around me to help me in my success. Similarly, I would like to thank Thomas, Alexis, and Juliette, my little brothers and sister, who have always pushed me to set an example. I would also like to thank my grandparents, and especially Manou, for constantly supporting and encouraging me in difficult moments. I would also like to thank Philipe and Francoise for their support and interest in my projects. In addition, I would like to thank Bruno who has always accompanied me and seen me grow from the lowest to the highest.

I would also like to thank Gauthier, David, P-A, Sacha, Nicolas, Lucas, and Arthur, my lifelong friends who have always been there for me. I would also like to thank Baptiste, Nicolas, Cécil, Leandro, Raymond, and Phrasi, who have been my university friends with whom we have grown and developed our thinking.

I would also like to thank all my university professors and Laurent, my high school teacher, who pushed me to develop my mathematical intuition. I would also like to thank the staff of UZB, who welcomed me into a professional environment and allowed me to understand the medical concepts necessary for my understanding.

Finally, I would like to dedicate this thesis to my grandfather Paul who left us with cancer. Even though he went much too early, I hope he is watching me and that I can make him proud of me.

Contents

1 Introduction **1**
- 1.1 Motivations . . . 1
- 1.2 Objectives and Contributions . . . 1
- 1.3 Context of this thesis . . . 2
- 1.4 Document structure . . . 2

2 Background **3**
- 2.1 Deep learning introduction . . . 3
  - 2.1.1 Learning paradigm . . . 3
  - 2.1.2 Learning objective . . . 4
  - 2.1.3 Learning algorithm . . . 4
  - 2.1.4 Learning evaluation . . . 5
  - 2.1.5 Learning regularisation . . . 6
- 2.2 Representation Learning . . . 7
  - 2.2.1 Multi-Layer Perceptron (MLP) . . . 7
  - 2.2.2 Convolutional Neural Network (CNN) . . . 9
  - 2.2.3 Transformers . . . 11
- 2.3 Graph Neural Network . . . 14
  - 2.3.1 Graph domain . . . 15
  - 2.3.2 Convolution . . . 15
  - 2.3.3 Spectral Graph Convolution . . . 16
  - 2.3.4 Spatial Graph Convolution . . . 19
  - 2.3.5 Graph structure learning . . . 23
  - 2.3.6 Task-based Graph Convolutional Network . . . 24
- 2.4 Hypergraph Neural Network . . . 26
  - 2.4.1 Hypergraph domain . . . 26
  - 2.4.2 Hypergraph convolution . . . 26

3 Related Work **27**
- 3.1 Related Work . . . 27
  - 3.1.1 Methods with manual graph construction . . . 28
  - 3.1.2 Methods with learned graph topology . . . 31

4 Methodology **35**
- 4.1 Introduction . . . 35
- 4.2 Methodology . . . 36
- 4.3 Manual graph construction based on similarity metric . . . 37
  - 4.3.1 Hypothesis . . . 37
  - 4.3.2 Methodology . . . 37
  - 4.3.3 Architecture . . . 38
- 4.4 Contrastive-based similarity learning to reduce graph topology uncertainty . . . 38
  - 4.4.1 Methodology . . . 39
  - 4.4.2 Architecture . . . 39
- 4.5 Attention-based dynamic graph topology learning . . . 40
  - 4.5.1 Hypothesis . . . 40

4.5.2 Methodology . . . 40
4.5.3 Architecture . . . 41
4.6 Properties-based regularisation of graph structure . . . 42
4.6.1 Hypothesis . . . 42
4.6.2 Methodology . . . 42
4.6.3 Architecture . . . 43
4.7 Hypergraph-based model to learn a higher level of interaction . . . 43
4.7.1 Hypothesis . . . 43
4.7.2 Methodology . . . 43
4.7.3 Architecture . . . 44

5 Experiments **45**
5.1 Introduction . . . 45
5.2 Experimental set-up . . . 46
5.2.1 Dataset . . . 46
5.2.2 Experimental pipeline . . . 47
5.3 Patient level . . . 49
5.3.1 Preprocessing . . . 50
5.3.2 Embedding . . . 51
5.4 Population Level . . . 54
5.4.1 Manual graph construction based on similarity metric . . . 54
5.5 Contrastive-based similarity learning to reduce graph topology uncertainty . . . 56
5.5.1 Experiments . . . 56
5.5.2 Results . . . 57
5.6 Attention-based dynamic graph topology learning . . . 57
5.6.1 Experiments . . . 58
5.6.2 Results . . . 58
5.7 Properties-based regularisation of graph structure . . . 58
5.7.1 Experiments . . . 58
5.7.2 Results . . . 59
5.8 Hypergraph-based model to learn a higher level of interaction . . . 59
5.8.1 Experiments . . . 59
5.8.2 Results . . . 59
5.9 Comparison and Discussion . . . 59
5.9.1 Complexity analysis . . . 60
5.9.2 Performance analysis . . . 60
5.9.3 Comparison with the state of the art . . . 60
5.9.4 Experiments-based discussion . . . 61

6 Conclusions and future works **63**
Acronyms . . . 65

References **65**

A Dataset **75**
A.1 STOIC Dataset . . . 75
A.2 SBU Dataset . . . 75

Chapter 1

Introduction

1.1 Motivations

Artificial intelligence (AI) has the potential to revolutionize healthcare by improving the accuracy and efficiency of medical diagnosis, and treatment [1]. With the increase of data collection in the medical field, AI has become a powerful tool to make sense of this data and extract useful information. One area where AI has shown promise is the processing of multimodal clinical data [2]. Multimodal data includes all medical information that can be obtained from a patient. The data can be symptoms, demographics, medication, biomedical images, treatment received during hospitalization, and blood test results. These data are difficult to process and analyze using traditional machine learning methods, but with the advancements in deep learning, the field of AI in healthcare is rapidly expanding [2][3].

Deep learning models are able to learn from large amounts of data and generalize well to new examples [4]. This ability of deep learning to automatically extract features from data and learn complex patterns makes it a powerful tool for processing multimodal clinical data [3]. One of the main challenges in handling multimodal clinical data is understanding the relationships between different modalities and how they influence each other. In addition to extracting information from the interaction of modalities, some works are emerging to extract information from the interaction between patients to increase the model's predictive capacity [5].

Graphs are very promising for representing models based on interactions [6]. Indeed, their representational flexibility allows for coherently expressing many real-world problems [5]. Therefore, they are promising in processing multimodal data and patient interactions. Graph Neural Network (GCN) is a sub-group of deep learning that allows processing data in graph form. They have already proven themselves in many fields, such as chemistry, social networks, neuroscience, and many other areas requiring interaction representation [5]. It is in this context that this thesis is proposed.

1.2 Objectives and Contributions

This thesis aims to develop a predictive model based on GCN for patient-based prognosis using multimodal clinical data. In particular, the objective will be to show how it is possible to extract additional information from a graph representation of a patient population. To begin with, the aim will be to implement a model capable of extracting information from the different modalities. Based on this extraction of information at the patient level, we will have to prove that it is possible to enrich it from that of the other patients.

The contributions of this thesis will be the following:

- A contextualization of mathematical concepts essential to the development of deep learning models.
- A complete synthesis of the aspects related to GCN.
- A review of the literature summarizing work already done in similar contexts.

- An original use of state of the art models to process multimodal data.
- As a main scientific contribution, Several models of population representation by graph are proposed.
- The implementation of an experimental setup allowing the reproducibility and analysis of results.

1.3 Context of this thesis

The initial thesis topic was medical image segmentation with GCN-based models. During the first period of the thesis, I discovered the world of deep learning and graphs. However, as the ULB administration decided to postpone my master's thesis for one year, I had the chance to work as a job student on implementing a state-of-art graph-based covid-19 outcome combining medical images and patient data. I continued on this subject for an extra two months as a job student, as well as for my internship within the framework of a collaboration between ETRO and the Universitair Ziekenhuis Brussel (UZB), where I implemented a Bayesian model, proposed by ETRO, on clinic data to extract risk factors for COVID-19. Based on the experience gained, the subject of the master thesis has been modified to continue in the direction of medical prognosis based on multimodal clinical data.

1.4 Document structure

This thesis is structured based on the following chapters:

- The "Background" chapter is designed to introduce all the mathematical concepts necessary to understand the following chapters. This chapter is based on university courses, books, and articles from the literature.
- The "Related work" chapter is intended to show the existing works in the literature on the same problem. It is structured by summarizing the key points necessary to pose the problem around graphs.
- The "Methodology" chapter is the contribution of this thesis and is structured to pose the problem theoretically. For each proposed approach, a hypothesis will be postulated, and the methodology for translating this hypothesis will be presented. Finally, an architecture synthesizing the method will be shown.
- The chapter "Experiments" represents the practical work done in this area. This chapter will lay the experimental foundations for evaluating the different approaches around graphs. Furthermore, all aspects concerning data, preprocessing, embedding, etc., will be explained in detail. At the end of this chapter, a comparison of the results will be done with each other and with respect to the literature.
- The "Conclusions and future works" chapter will discuss the conclusion of this work and the future work that can be done to continue advancing the problem.

Chapter 2

Background

2.1 Deep learning introduction

Let's represent a deep learning model as a function $f_\theta(x)$ that takes an input x and produces an output $\hat{y}$ by processing x with a series of stacked non-linear transformations parameterized by learning parameters θ. The purpose of deep learning will be to define the suitable model (combination of non-linear transformations) and find the optimal θ values to obtain the expected output based on the input data [7]. Deep learning gets its name from the number of stacked non-linear combinations, commonly named layers; the higher the number of layers, the deeper the model[8]. Deep learning can therefore be defined as a very high dimensional optimization problem [7]. To describe the different aspects necessary to achieve such a modelization, the critical points of the learning process will first be discussed to show how it is possible to obtain suitable model parameters. Then, the aspect of the model representation will be detailed and motivated to solve several problems. Finally, the last two sections will describe the representation by Graph and Hypergraph, constituting the objective of this thesis.

2.1.1 Learning paradigm

An optimization problem is usually posed based on an objective representation that has to be reached under several constraints. In the literature, there are several learning strategies [4][7]:

- Supervised learning: This is the most common type of learning in the literature.[4] This is a type of learning where the model is trained on a labeled dataset where the label is provided for each input. The objective is to learn a function that can map inputs to the given outputs [7].

- Unsupervised learning: In unsupervised learning, the model is trained on an unlabeled dataset. Instead, the objective is to learn patterns and relationships by identifying common features or structures [7].

- Semi-supervised learning: This hybrid approach combines the two previous kinds of learning. The model is trained on labeled and unlabeled samples in the dataset [7].

- Reinforcement learning: In reinforcement learning, the model receives feedback in the form of rewards or penalties based on its actions and learns to select actions that maximize the overall reward [7].

- Transfer learning: Transfer learning is a method that aims to map pre-trained model parameters to another model [4]. It's typically used as an initialization to train a model with a few amount of samples [7].

- Self-supervised learning is a method in which the model has to train itself and aims to use the data to train itself. It is a mix of all the previous learning types. Self-supervised learning is the closest to human learning.

2.1.2 Learning objective

Error representation

To continue, let's assume to be in a supervised learning paradigm. For this purpose, let's define a loss function $L(.)$, representing the error between the predicted output $\hat{y}$ and the ground truth y. Then, following [7], it can be expressed as :

$$L = Loss(\hat{y}, y) \tag{2.1}$$

Many different loss functions can be used depending on the task at hand. For example, in a binary classification problem, the predicted output might be represented as a probability to belong in the positive class $\hat{y} \in [0, 1]$, and the label is a binary label $y \in \{0, 1\}$. In this case, a common loss function is Binary Cross Entropy (BCE) [9], which is defined as :

$$BCE(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \tag{2.2}$$

Objective

Having modeled the model error, the objective is now to find the model parameters that minimize this loss. More formally, following [7], the learning objective can be formulated as follows :

$$\hat{\theta} = \arg_\theta \min \; L(\hat{y}, y; \theta) \tag{2.3}$$

It is not trivial to obtain optimal values of θ. One of the first reasons is that deep learning entails no non-convex optimimization[1] [10], meaning that we cannot guarantee the existence of a global minimum [11][7]. Moreover, even if theta values allow to optimize perfectly the loss function during training, it is not possible to guarantee the same performance for inputs not seen by the model [8]. To implement a robust model, it is desirable to find solutions allowing generalisation capabilities [4].

2.1.3 Learning algorithm

Gradient-based learning algorithms are a class of optimization algorithms commonly used to train deep learning models [12]. These algorithms involve iteratively updating the model parameters to reduce the loss function [7].

Gradient-descent

Gradient descent is an optimization algorithm used to update the values of the model's parameters so that they minimize the loss function [13]. It works by iteratively adjusting the parameters in the direction that reduce the loss [12]. Mathematically, gradient descent updates the model's parameters θ at each iteration t based on the gradients of the loss function L with respect to the parameters [13]:

$$\theta^{t+1} = \theta^t - \eta \nabla_{\theta^t} L(\hat{y}, y; \theta^t) \tag{2.4}$$

where η is the learning rate, which determines the step size of the update of the parameters at each iteration. There are several versions of this technique such as the Stochastic Gradient Descent (SGD)[14] which updates the model parameters after randomly selecting one sample in the dataset, or the mini-batch gradient descent algorithm [15] which updates the parameters after passing a small subset (mini-batch) of the dataset at each iteration [13].

Momentum [13] is a technique that can be used to accelerate SGD by adding a momentum term to the updated rule. The momentum term causes the optimizer to simulate inertia and helps the optimizer to continue moving in the same direction, even if the gradient changes sign [13] :

$$\theta^{t+1} = \theta^t - \eta m_t \tag{2.5}$$

with m_t :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta^t} L(\hat{y}, y; \theta^t) \tag{2.6}$$

[1] Mainly due to the non-linearity introduced by the activation function [10]

Adagrad [16] is an algorithm that uses a per-parameter learning rate, which means that each parameter is updated with a different learning rate depending on how frequently it has been updated in the past [13]:

$$\theta^{t+1} = \theta^t - \frac{\eta}{\sqrt{\hat{v(t)} + \epsilon}} \tag{2.7}$$

with v_t :

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla^2_{\theta^t} L(\hat{y}, y; \theta^t) \tag{2.8}$$

Adaptive Moment Estimation (Adam)

The Adam[17] optimization algorithm is a variant of gradient descent that uses an adaptive learning rate to adjust the model parameters[17]. It combines the ideas of momentum and adaptive learning rates, which helps to improve the convergence of the optimization process [13]. Mathematically, the Adam algorithm updates the model parameters θ at each iteration t based on the gradients of the loss function L with respect to the parameters, as well as the previous updates of the parameters [13]:

$$\theta^{t+1} = \theta^t - \frac{\eta}{\sqrt{\hat{v(t)} + \epsilon}} \hat{m(t)} \tag{2.9}$$

where η is the learning rate, $\hat{m}(t) = \frac{m_t}{(1-\beta_1^t)}$ and $\hat{v}(t) = \frac{m_t}{(1-\beta_2^t)}$ and ϵ is a small constant used to prevent division by zero. $m(t)$ is the moving average of the gradients, $v(t)$ is the moving average of the squared gradients defined in Eq(2.6) and Eq(2.8) respectively. β_1 and β_2 are the decay rates and are often set to 0.9 and 0.99 respectively.

Backpropagation

In practice, the backpropagation algorithm [18] is used to compute the partial derivative efficiently during gradient descent [19]. Here's a brief outline of this algorithm [12]:

- Initialize model parameters randomly.
- Feed forward the input data through the model to obtain the predicted output.
- Calculate the error between the prediction and the ground truth.
- Propagate the error back using gradient descent-based method.
- Repeat this process for multiple epochs (iterations over the entire data set) until we reach the expected behavior.

2.1.4 Learning evaluation

As discussed above, it is desirable to train a model capable of generalizing its knowledge on input data it has never seen. For this, it is necessary to keep aside a part of the data set that will not be used to train the model. This part will only be used to validate a trained model.

Evaluation metric

While the loss function represents the evaluation of the optimisation objective, an evaluation metric is useful to evaluate the estimation performance of the model. In the context of binary classification, the evaluation metrics can be expressed in function of Operational Taxonomic Units (OTUs) [20] namely the true positives (TP), the false positives (FP), the true negatives (TN), and the false negatives (FN). These OTUs are coming from the confusion matrix Figure 2.1. Multiple metrics can be built based on these previous concepts. For example, let's define the accuracy as [20]:

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{2.10}$$

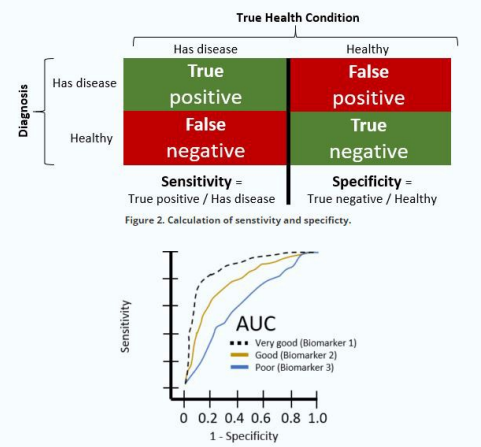

Figure 2.1: **C**onfusion matrix. **Bottom:** ROC curve used to compute the AUC. Illustration taken from `https://medium.com/computer-architecture-club/what-is-the-auc-roc-curve-47fbdcbf7a4a`

The accuracy corresponds to the ratio between the number of correctly classified samples and all the samples [20]. It is important to choose the right evaluation metric to avoid biasing the evaluation of our model [21]. For example, in a case of an unbalanced dataset (e.g., 10% positive, 90% negative label), the accuracy is not optimal because if the model classifies all samples as negative, it will result in an accuracy of 90% even though the model is very bad [20]. Moreover, in the medical field, it is important to evaluate and calibrate the ability of a model to produce negative or positive prediction [21]. For example, in the case of virus detection, it could be preferable to have false positives earlier than false negatives to avoid letting someone out, as such a person may continue spreading the virus or not receive medical attention on time. [21]. To deal with these issues one can evaluate the model based on its sensitivity ($\frac{TP}{TP+FN}$) and its specificity ($\frac{TN}{TN+FP}$). These two metrics inform on the true positive and negative rates, respectively. As described above, the model output is the probability of input belonging to positive class $y \in [0, 1]$. Hence, it's possible to define the probability threshold to obtain discrete prediction $y \in \{0, 1\}$ and calibrate for different true positive/negative rates depending on the application.

Area Under the Curve (AUC) measures the classifier's ability to distinguish between positive and negative instances [22]. The AUC is calculated using the ROC curve (receiver operating characteristic curve) [23], which is a plot of the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification thresholds as illustrated on Figure 2.1 [20]. The TPR is the proportion of positive samples that are correctly classified as positive. In contrast, the FPR is the proportion of negative samples incorrectly classified as positive [20]. AUC score ranges from 0 to 1, with 1 indicating the perfect classification and a value of 0.5 indicating random classification.

2.1.5 Learning regularisation

Overfitting problem

Overfitting is a common problem in machine learning [24]. It occurs when a model is excessively complex, such as having too many parameters relative to the size of the training data [24][7]. During training, the model can learn the noisy details in the training data that negatively impacts the model's ability to generalize to new unseen data [24][7]. The model will perform very well on the training data, but it will perform poorly on new, unseen data [24][7].

L1 regularization , also known as Lasso regularization [25], adds a penalty term proportional to the absolute value of the model parameters. This results in a sparse model, where some parameters are exactly equal to zero, as illustrated in Figure 2.2 [7]. L1 regularization is often used when we want to select a small number of important features or when we want a more compact model [26][7].

L2 regularization , also known as Ridge regularization [27], adds a penalty term proportional to the square of the model parameters. This results in a model where the parameters are small but not necessarily equal to zero, as illustrated in Figure 2.2 [7]. L2 regularization is often used to prevent overfitting and improve the generalization of the model [26][7]. Both L1 and L2 regularization can effectively reduce overfitting, but they have different characteristics and may be better suited for different types of problems [12][4]. It is often necessary to try both and see which works better for a given problem [7].

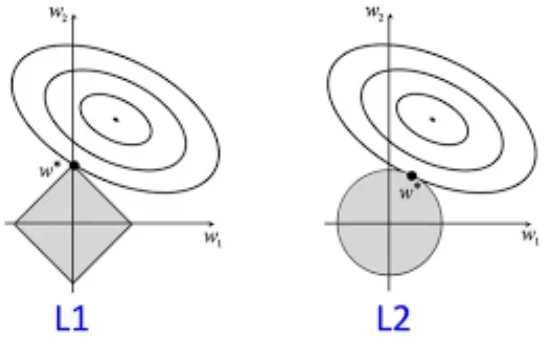

Figure 2.2: Estimation graph showing the intersection between the loss function and the L1 (left) and L2 (right) constraints. The L1 regularisation forces the parameters to be equal to zero (sparsity), while the L2 regularisation forces the parameters close to zero. Illustration taken from `https://www.analyticsvidhya.com/blog/2021/05/complete-guide-to-regularization-techniques-in-machine-learning/`

Dropout is a simple and effective way to reduce overfitting [4]. During training, dropout randomly sets a fraction of the input data or model parameters to zero [28]. This has the effect of ignoring a random part of the elements contributing to the prediction [7]. This helps to prevent overfitting because it forces the model to learn multiple independent representations of the data, rather than relying on a single complex model [28].

2.2 Representation Learning

Representation learning is a fundamental concept in deep learning, which refers to the process of learning a good representation or encoding of the input data in a way that captures the underlying structure and relationships in the data [26][7][29]. In the previous section, the concepts related to learning were introduced. In this section, we will see how to represent the input data using a deep learning model.

2.2.1 Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) is a type of artificial neural network that is composed of multiple layers of interconnected "neurons" [30]. Each neuron receives an input from the previous layer, processes it using an activation function, and then passes the result to the next layer [30]. The output of the final layer is the output of the MLP.

Perceptron

Let's define a perceptron [31] taking an input $x \in \mathbb{R}^{d_f}$, a raw vector of dimension d_f and produce a scalar $z \in \mathbb{R}$ as output by performing a weighted sum of the input feature with weight $w \in \mathbb{R}^{d_f}$ and bias $b \in \mathbb{R}$. Hence, the perceptron can be formulated as follows :

$$z = f(b + \sum_i w_i x_i) \tag{2.11}$$

where $f(.)$ is a non-linear activation function that can be used to apply a threshold (e.g., ReLu [32]) or change the output interval (e.g., Sigmoid [32]).

Multi-Layer Perceptron

A MLP is a combination of layers with multiple perceptrons taking as an input the result of the previous layer [30]. More formally, let's extend the previous equation by defining z_j^l as the output of perceptron j at layer l and define f^l as the perceptron operations at layer l. Hence $f^1(x) = z^1 = [z_0^l, z_1^l, ..., z_j^l]$ represents the raw vector containing the values of the outputs at the first layer. Thus, the MLP operation can be written as :

$$z^L = f^L(...f^2(f^1(x))) \tag{2.12}$$

with $z^L \in \mathbb{R}^{d_o}$ the output of dimension d_o at the last layer L of the MLP.The MLP concepts are described in Figure 2.3.

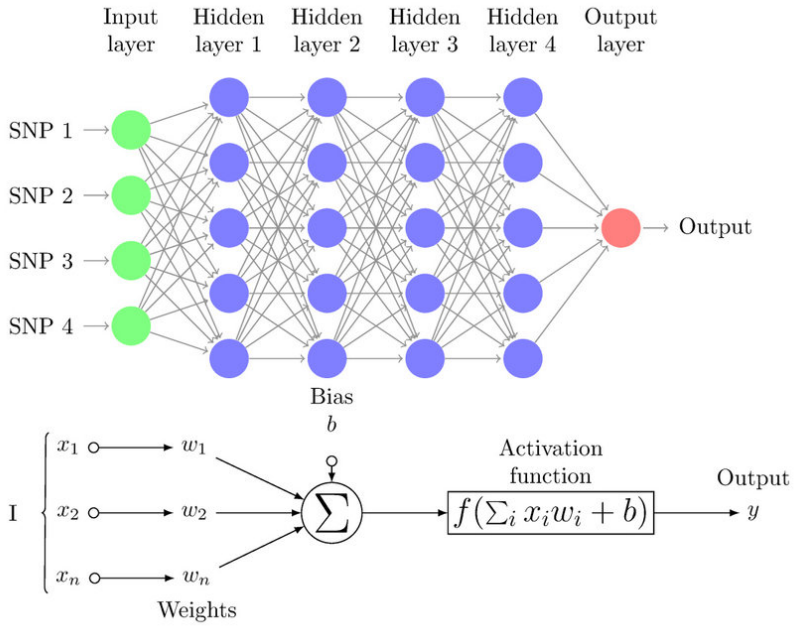

Figure 2.3: Perceptron and MLP illustration. The intermediate layers are called "hidden layers," while the first and the last ones are called the "input layer" and "output layer," respectively. Picture taken from `https://www.researchgate.net/publication/334609713_A_Guide_for_Using_Deep_Learning_for_Complex_Trait_Genomic_Prediction/figures?lo=1`

Representation

Output representation In the case of binary classification, the last layer usually consists of two perceptrons. These two dimensions represent the probability of belonging to a class. To guarantee that the sum of the probabilities is 1, a softmax [33] activation function is usually applied. Let's define $y_i = P(y = i)$ the probability to belong in class $i \in \{0, 1\}$. Thus, the $softmax(.)$ operation can be written as follows [33] :

$$y_i = P(y = i) = \text{softmax}(z^L) = \frac{\exp(z_i^L)}{\sum_i \exp(z_i^L)} \tag{2.13}$$

Input representation In a MLP, the data representation at the input can take several forms depending on the nature of the data and the task at hand. Some common data representations at the input of an MLP include [34][29]:

- Numeric data: Numeric data, such as real-valued or integer-valued data, can be input directly into a MLP. For example, in a classification task, the input data may consist of variables such as age, height, or weight [35].

- One-hot encoded data [36]: Categorical data, such as data that consists of labels or categories, can be input into an MLP by using one-hot encoding. In one-hot encoding, each category is represented as a binary vector with a one in the position corresponding to the category and zeros in all other positions. For example, if there are three categories (A, B, and C), they would be represented as [1, 0, 0], [0, 1, 0], and [0, 0, 1], respectively.

- Embedding: For data consisting of large vocabularies or large numbers of categories, it can be beneficial to use embedding to represent the data [26]. An embedding is a low-dimensional continuous vector representation which can capture the underlying relationships between the features [19]. Embedding can be learned as part of the training process, or they can be pre-trained on a large data set and fine-tuned for the specific task [19].

- Images: Images can be put as input into an MLP by flattening the 2D image into a 1D array and inputting it as a vector [37]. Alternatively, Convolutional Neural Network (CNN) [37] can be used to learn features from the spatial structure of the image, which can then be input into an MLP for further processing.

- Sequences: For data that is structured as a sequence, such as time series data or natural language data, Reccurent Neural Network (RNN) [38] can be used to process the sequential structure and generate a fixed-length representation that can be input into an MLP.

Overall, the choice of data representation at the input of an MLP will depend on the nature of the data [34]. It is essential to choose a representation that captures the relevant information in the data allowing the MLP to learn the desired pattern effectively [29][26].

2.2.2 Convolutional Neural Network (CNN)

Convolutional Neural Network [37] is a type of neural network that is specifically designed to process data with a grid-like structure, such as an image, by learning a set of filters to extract meaningful features [7].

Motivation One of the main reasons for using CNN is to reduce the number of parameters. Indeed, assuming that we have a CT-scan (1024x1024x512) [35], it becomes very difficult to use a MLP to process this biomedical image[34]. We therefore wish to include sparsity in interaction by sharing parameters and get an equivariant representation [7][39].

Convolution To get these properties, let's define the discrete convolution by assuming an input x and a kernel w. The discrete convolution of a one-dimensional signal can be written as follows [7][40] :

$$y_i = \sum_j w_j x_{i-j} \tag{2.14}$$

In practice, most of the standard deep learning packages implement the cross-correlation instead of convolution just by flipping the kernel, which does not change anything to learn his parameters in the backpropagation [39][7][40]:

$$y_i = \sum_j w_j x_{i+j} \tag{2.15}$$

The previous definition can be extended for higher dimensional signals [40][7]:

$$y_{ij} = \sum_{kl} w_{kl} x_{i+k,j+l} \tag{2.16}$$

Eq(2.16) represents the convolution for two-dimensional signals. An intuitive representation is shown in Figure 2.4.

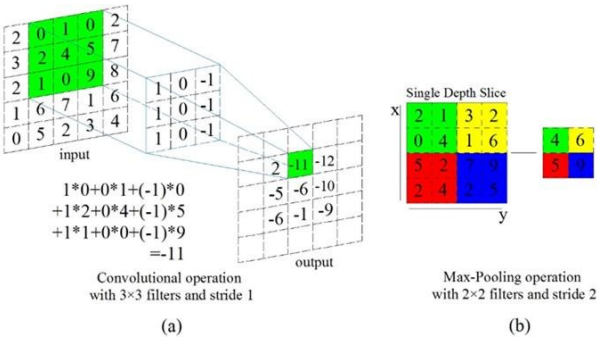

Figure 2.4: Convolution and pooling operation description. Picture taken from `https://towardsdatascience.com/demystifying-convolutional-neural-networks-384785791596`

Pooling In CNN, pooling is a process that downsamples the spatial dimensions of the feature maps produced by the convolutional layers [39]. The main goal of pooling is to reduce the size of the feature maps, which helps reduce the network's computational complexity and makes it more robust to small translations in the input [37]. This concept is illustrated on Figure 2.4

Convolutional Neural Network The typical architecture of a CNN is illustrated in Figure 2.5

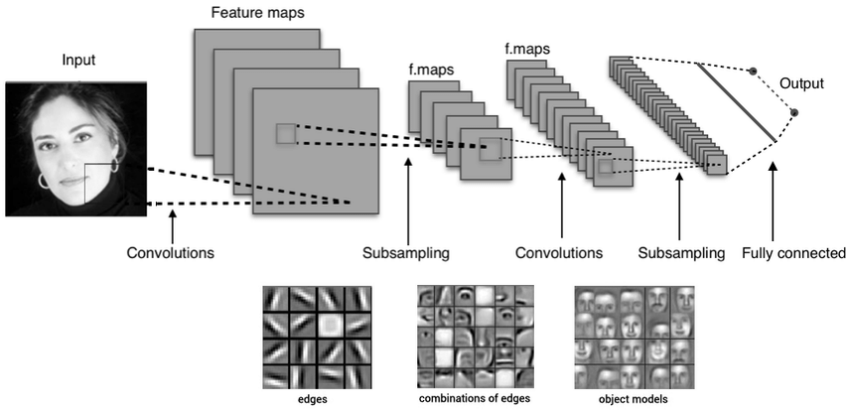

Figure 2.5: Example of CNN architecture description. Picture taken from `https://www.researchgate.net/publication/323218981_Deep_Feature_Extraction_for_Sample-Efficient_Reinforcement_Learning/figures?lo=1`

Input layer The input layer receives an image of size $W \times H \times C$ (width x height x channels).

Convolutional layer This layer applies N kernels (usually of size $3 \times 3 \times C$) to the input images, producing a feature map of size $W \times H \times N$.

Subsampling layer This layer applies pooling (or strided convolution [39]) reducing the size (usually by a factor 2) of the features map ($\frac{W}{2} \times \frac{H}{2} \times N$). Hence, at each stage, the image decreases in size and increases in channels until the number of elements is sufficiently small to be used in a MLP classifier [40].

Representation The goal of representation learning in a CNN for image classification is to learn discriminative features for the prediction [37]. For example, the CNN might learn to extract features such as edges, textures, and shapes that are useful for distinguishing between different belonging patterns [40]. It might also learn to extract more complex features, such as object parts or configurations, that are more specific to certain classes of objects as illustrated on Figure 2.5[41].

2.2.3 Transformers

Transformers gained significant popularity in recent years, especially in the field of Natural Language Processing (NLP)[42]. They were first introduced by the authors of "Attention is all you need" [41], which demonstrated that transformers could achieve state-of-the-art performance on machine translation tasks without using recurrence or convolutions [41]. The critical feature of transformers is their use of self-attention mechanisms, which allow the model to process input sequences of arbitrary length and attend to different parts of the input simultaneously [41][42]. This makes transformers particularly well-suited for tasks such as language translation, where the input and output sequences can be of different lengths and have complex dependencies between different parts of the sequence [41][43][42].

Self-attention

Concept To define the concept of self-attention, let's represent input data as a set of t feature vectors of dimension d_f [40]:

$$\{x_i\}_{i=1}^{t} = \{x_1, ..., x_t\} \tag{2.17}$$

Each $x_i \in \mathbb{R}^{d_f}$ is a tokenized version of the input data and can be represented in matrix form $X \in \mathbb{R}^{d_f \times t}$. The self-attention can be seen as the weight $\alpha \in \mathbb{R}^t$ multiplying each token to represent their impact in the hidden representation h [40]:

$$h = \alpha_1 x_1 + ... + \alpha_t x_t \tag{2.18}$$

$$h = X\alpha \tag{2.19}$$

Hard attention consists in imposing $\|\boldsymbol{a}\|_0 = 1$ to define a mask that cancels specific contributions in the hidden representation [40][43].

Soft Attention consists in imposing $\|\boldsymbol{a}\|_1 = 1$ to include sparsity in combination with more flexibility than hard attention [40][43].

Representation Until now, we represented the hidden representation as the combination of the entire input with the weight matrix for MLP and with a small Kernel for CNN. Here, we are talking about the linear combination of x columns with attention matrix [40]. Then, the attention value α_i can be seen as the impact of the given token x_i with respect to the entire input X and can be written as follows [40]:

$$\alpha_i = \text{softmax}(X^T x_i) \tag{2.20}$$

Where $softmax(.)$ is defined in Eq(2.13) and X^T is the transposed of X. By staking every element of α_i in a matrix $A \in \mathbb{R}^{d_f \times t}$ we can obtain the "attention map" representing the interaction between each token x_i with respect to each other [40]. Hence, the hidden representation of X in the attention paradigm can be written as [40]:

$$H = XA \tag{2.21}$$

Key-value store paradigm In computer science, the key-value store [44] paradigm is a way of storing and organizing data in a database [44][40]. In a key-value store, data is organized as a set of keys; each associated with a specific value [44]. The keys serve as unique identifiers for the values, allowing them to be retrieved and manipulated efficiently [44]. In self-attention, the input to the model is represented as a set of key-value pairs, with each key associated with a corresponding value [43]. The model also receives a set of queries, which represent the output that the model is trying to predict [42]. More formally, Let's define q_i, k_i and v_i the query, key and value of the token x_i as follow [40]:

$$\begin{aligned} q_i &= W_q x_i \\ k_i &= W_k x_i \\ v_i &= W_v x_i \end{aligned} \tag{2.22}$$

Where $W_q \in \mathbb{R}^{d_q \times d_f}$, $W_k \in \mathbb{R}^{d_k \times d_f}$ and $W_v \in \mathbb{R}^{d_v \times d_f}$ are parameters matrix that create 3 different embedding for the token x_i. Now, let's make the assumption that $d_h := d_q = d_k = d_v$ and stack each query, key and value in a $Q, K, V \in \mathbb{R}^{d_h \times t}$ matrices. We can define the self-attention in the Key-value

store paradigm by comparing each query with all keys to obtain a scaled combination of value columns [41]. Then, Eq(2.20) can be written as follow [40]:

$$\alpha_i = softmax(K^T q_i) \tag{2.23}$$

And by staking all α_i element in A, the hidden representation in Eq(2.21) can be written in a complete form as follows [40]:

$$H = softmax(\frac{QK^T}{\sqrt{d_h}})V \tag{2.24}$$

with $\frac{1}{d_h}$ a normalisation factor to get good learning properties [41]. Eq(2.24) can be seen as a "scaled dot-product" [41] as shown on Figure 2.6.

Multi-Head attention Multi-Head attention is a mechanism used to improve the performance of self-attention by allowing the model to attend to multiple parts of the input simultaneously [41]. It simply consists in creating multiple instances of Q,V,K and combining the result at the end (sum, average, ...) as illustrated in Figure 2.6 and explained in [40][43].

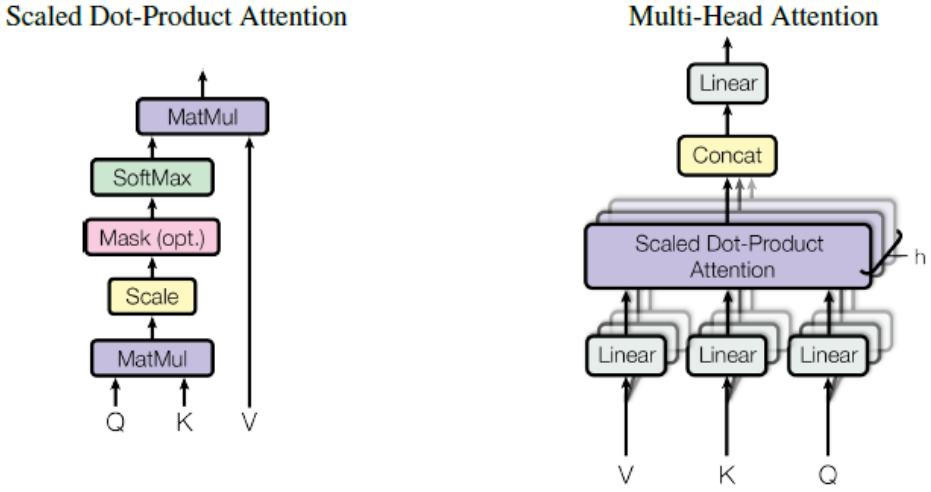

Figure 2.6: **Left:** Illustration of Eq(2.24).**Right:** Illustration of "Multi-Head attention." Picture taken from [41]

Transformer encoder

The authors of [41] propose an architecture for a transformer-based encoder. As illustrated in Figure 2.7, the transformer encoder consists of multiple layers containing two sub-layers. The first one is a "multi-head self-attention" operation as described in Eq(2.24). The second one consists of a MLP -based operation. The authors also use residual connection [45] and normalization layer [46]. The normalization layer is defined as follows [46]:

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta \tag{2.25}$$

Where γ and β are learnable parameters while small ϵ is used to avoid division by zero. The residual connection consists in adding the feature vector (identity mapping) from the precedent sub-layer to improve information flow [45].

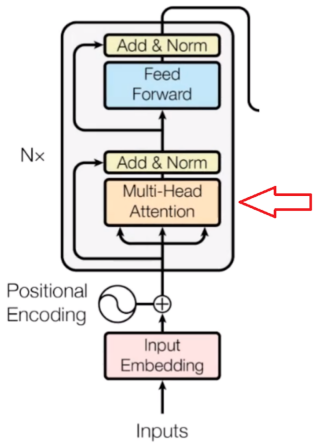

Figure 2.7: Illustration of Transformer encoder [41]. Picture taken from [41]

Tab Transformer

TabTransformer [47] extends the transformer architecture to handle tabular data. Tabular data is often used to represent data in a structured and organized manner, like in an "excel file" [48]. One of the main challenges is that the input of the classical transformer model [41] is a sequence of tokens, while tabular data consists of a combination of numerical and categorical values [48]. To address this problem, the authors of [47] propose to create a special embedding for categorical values while simply scaling numerical values. The authors also extend their model by handling missing data thanks to the column attention [47]. This architecture is effective in a wide range of tabular data tasks, including predictive modeling and data imputation [43], and has achieved state-of-the-art results in many of these tasks [43]. The TabTransformer architecture is illustrated in Figure 2.8

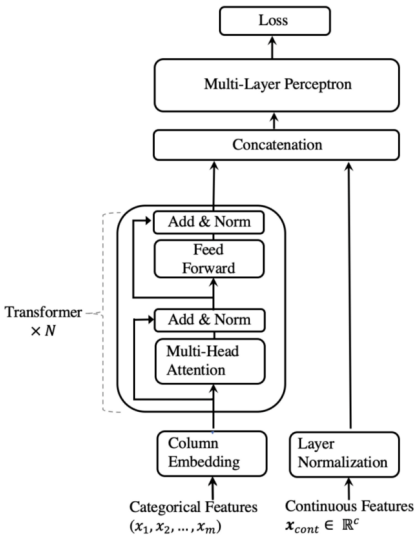

Figure 2.8: Picture taken from [47]

Column embedding Given a categorical column C with N unique values, the column embedding is represented by $E \in \mathbb{R}^{N \times d_e}$, where d_e is the embedding dimension. To convert a unique categorical value v in column C to a token, the corresponding row in the matrix E is retrieved and appended to the input sequence.

Vision Transformer

The authors of [49] propose the Vision Transformer (ViT), a transformer-based image classification model.

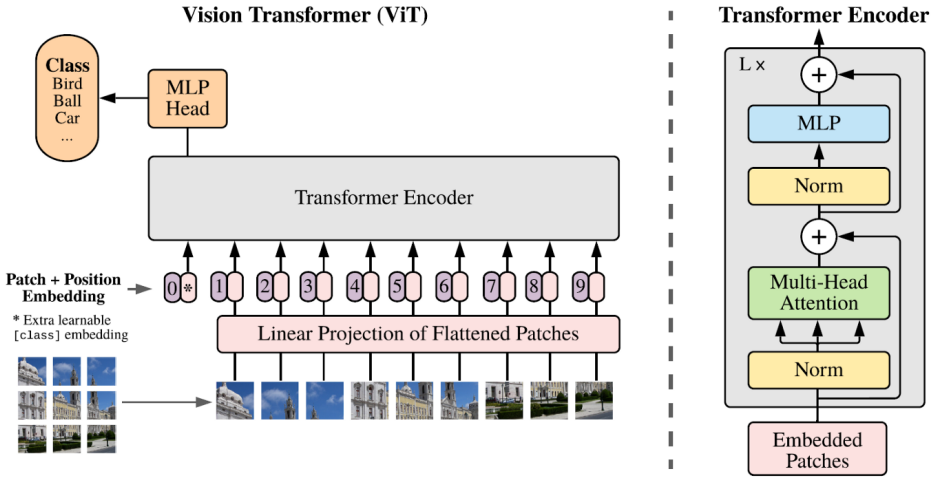

Figure 2.9: Picture taken from [49]

One of the main problems that the vision transformer addresses is the inability of traditional Convolutional Neural Network to process large images at scale [49]. CNN use local receptive fields to extract features from the input image, but these fields are limited in size [37] and might restrict the capture of the semantic context [50][40]. The Vision Transformer addresses this problem by dividing the input image into fixed patches (token) and using the self-attention mechanism to extract features and interaction as illustrated in Figure 2.9.

2.3 Graph Neural Network

So far, we have represented the learning by combining a weighted sum followed by the point-wise non-linearity creating a rotation matrix for our input data (MLP) [7]. We also know how to deal with significant inputs (images) by introducing equivariant representation by sharing the weight (kernel) to extract meaningful feature map while creating a hierarchical model (CNN)[40][7]. Finally, we have seen that it is possible to tokenize the input data and thus capture their interactions through the self-attention mechanism (Transformers) [49][41]. These representations are the basis of most deep-learning models for many tasks. The common point between these representations is that they are all based on a euclidean representation of the input data, meaning a strong spatial dependence [40]. However, it is easy to imagine that semantic understanding of image or tabular data is more complex in the real world [6].

It is in this context that Graph Neural Network (GCN) are introduced. Graph Neural Network allows a higher representation level by modeling interactions without the constraint of euclidean spatial location [6].

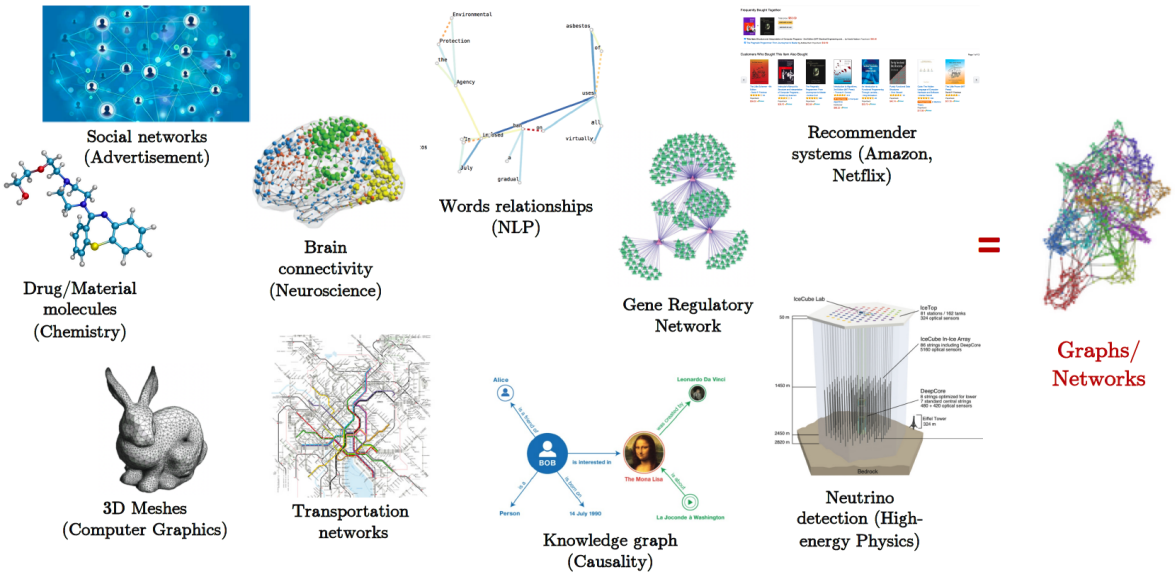

Figure 2.10: Example of using graphs to represent real-world problems. Picture taken from `https://graphdeeplearning.github.io/project/spatial-convnets/`

In this chapter, movivations for using graphs will be presented. First, they will be defined, and then a link between the concepts seen so far in the Euclidean domain will be translated into the graph domain. Finally, different representations for classification tasks will be introduced, such as spectral and spatial convolution, as well as graph structure learning. Moreover, in this section, the robustness, flexibility, and strong representation ability of graphs will be the transversal point driving this presentation [6].

2.3.1 Graph domain

As described in Figure 2.10, a graph is a relational-based representation. It models the relation (edge) between different elements (nodes). More formally, let's define a graph $G = (V, E, A)$ where V, E, A represent [40]:

- **Vertices** $V = \{v_i\}_{i=1}^{n}$ indexing the n vertex/node in the graph.
- **Edges** $E = \{e_{ij} = (v_i, v_j) | \forall e_{i,j} \in g\}$ indexing each pair of vertices that are connected.
- **Adjacency matrix** $A \in \mathbb{R}^{n \times n}$ where $a_{i,j} \in A$ represents the edge connectivity between node i and node j.

The graph can store information as [40]:

- **Node features:** $h_i \in \mathbb{R}^{d_v}$
- **Edge features:** $e_{i,j} \in \mathbb{R}^{d_e}$
- **Graph features:** $g \in \mathbb{R}^{d_g}$

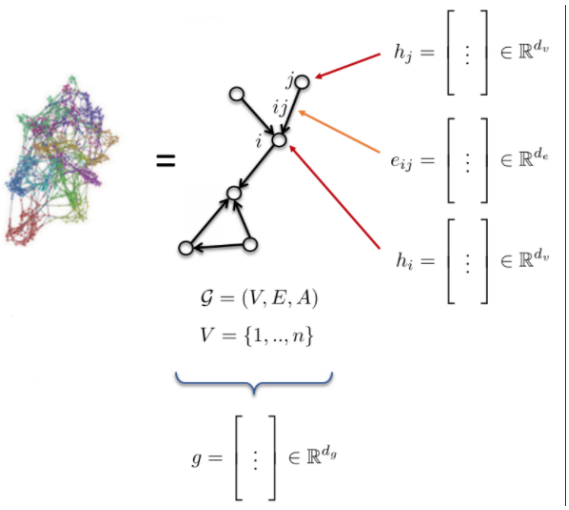

Figure 2.11: Graph definition illustration. Picture taken from [40]

The concepts of graph, node, and edges are now defined, and a clear illustration of the concepts is shown in Figure 2.11. Now we will try to extend the concept of convolution in the graph domain.

2.3.2 Convolution

As described in Section 2.2.2, the convolution operation consists of a dot product between a small kernel w and local region on the input signal [39]. This operation is done over the entire image [39]. Let's rewrite Eq(2.15) as [40]:

$$h_i^{l+1} = w^l * h_i^l = \sum_{j \in \Omega} \langle w_j^l, h_{i+j}^l \rangle \tag{2.26}$$

Where h_i^{l+1} is the hidden representation at layer $l+1$. As illustrated in Figure 2.12 and Eq(2.26), the kernel will always match each pixel with the direct neighbors. Then there is strong spatial interaction between the elements. In the graph, there is no notion of ordering and location [40]. Moreover, in the euclidean representation, the number of neighboring elements is fixed (grid-like structure), while the number of adjacent nodes can vary in the graph domain. Thus, we need to define the concept of graph convolution [39].

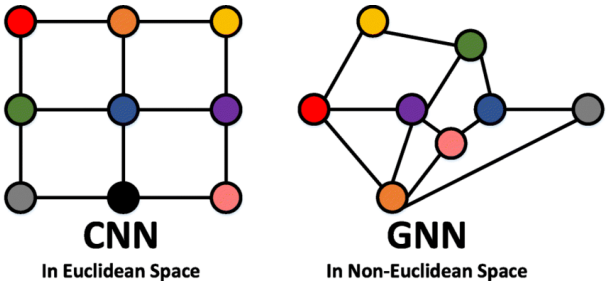

Figure 2.12: Euclidean domain vs. graph domain. Picture taken from `https://www.researchgate.net/publication/343821039_A_Survey_on_Deep_Learning-Based_Vehicular_Communication_Applications/figures?lo=1`

To overcome the precedent issues, there are two main approaches. The first one is named "spectral graph convolution" which uses a Fourier representation [51][52]. The second one is called "spatial graph convolution" which tries to apply Eq(2.26) in the graph domain in an isotropic and anisotropic way.

2.3.3 Spectral Graph Convolution

The first method to define graph convolution involves using a spectral representation of a graph [53]. For that, we need to extend some concepts of spectral theory like the Fourier transform [51][52] and the Laplacian in the graph domain [54].

Graph Laplacian

This is one of the most important concepts in spectral graph theory [53]. The graph Laplacian is defined as follows [40]:

$$\Delta = \mathrm{I} - D^{-1/2} A D^{-1/2} \tag{2.27}$$

Where $\Delta \in \mathbb{R}^{n \times n}$ with n being the number of nodes, A is the adjacency matrix, and $D \in \mathbb{R}^{n \times n}$ is the degree matrix defined as follows [40]:

$$D = \mathrm{diag}\left(\sum_{j \neq i} A_{ij}\right) \tag{2.28}$$

The degree matrix is diagonal and is used to normalize the adjacency matrix to prevent numerical computation instabilities [53][54].

Smoothness The Graph Laplacian measures the smoothness of the graph [53][40]. The smoothness of the graph can be seen as the variation between the features of the nodes that interact between them. For example, by taking a given node i, the graph will be smooth if the adjacent nodes have similar node features[53][40]. In computer science, the discrete Laplacian around an element is represented as the difference between this element and the average of the surrounding elements [54]. Then, in the graph context, the discrete Laplacian around the node i with node feature h_i can be written as follows [40]:

$$(\Delta h)_i = h_i - \frac{1}{d_i} \sum_{j \in \mathcal{N}_i} A_{ij} h_j \tag{2.29}$$

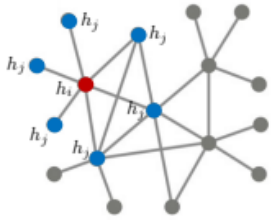

Figure 2.13: Illustration used to interpret graph Laplacian. Picture taken from `https://drive.google.com/file/d/1oq-nZE2bEiQjqBlmk5_N_rFC8LQY0jQr/edit`

Spectrum The graph Laplacian can also represent the spectrum of the graph [53][40]. To do this, let's make an eigen-decomposition of Eq(2.27) as follows :

$$\Delta = \Phi^T \Lambda \Phi \tag{2.30}$$

Where Φ represents the Laplacian eigenvectors ϕ_i and constitutes an orthornormal basis $\Phi\Phi^T = I$ [52]. Λ is a diagonal matrix containing all the Laplacian eigenvalues λ_i. The eigen-decomposition allows us to write the following relation :

$$\Delta \phi_k = \lambda_k \phi_k \tag{2.31}$$

With $k = 1, ..., n$ and with Eq(2.27), we get Laplacian eigenvalues properties $0 \leq \lambda_1 \leq \ldots \leq \lambda_n \leq 2$.

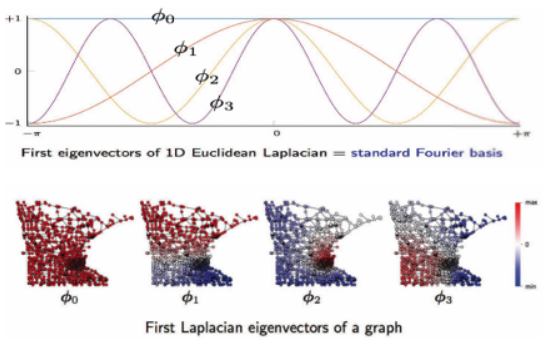

Figure 2.14: Representation of the principal Laplacian eigenvectors in euclidean and graph domain. Picture taken from [40]

The eigen-vectors represent the different basis of the Fourier function, i.e. the "principal directions" [52]. This concept is used in compression where we project the signal on the principal basis (the first ones) to represent the main components of the signal with lower dimension [53]. In the graph domain,

the eigenvectors represent the variation and can capture different topological clusters in the graph as illustrated in Figure 2.14 [40][55]. The graph Laplacian eigenvalues represent the different modes of the graph spectrum and constitute a signature of the graph [54][53].

Fourier analysis

Let's decompose a graph signal h as Fourier series. It can be done by projecting h on each Fourier basis ϕ_k as follow [40]:

$$h = \sum_{k=1}^{n} \langle \phi_k, h \rangle \quad \phi_k \tag{2.32}$$

Hence, the Fourier transform of signal h is defined as follows [40]:

$$\hat{h} = \mathcal{F}(h) = \Phi^T h \tag{2.33}$$

Where $\mathcal{F}(.)$ is the Fourier transform and $\hat{h}$ is the graph signal in the Fourier domain. Thus, the inverse Fourier transform of $\hat{h}$ is defined as follows [40]:

$$\mathcal{F}^{-1}(\hat{h}) = \Phi \hat{h} = \Phi \Phi^T h = h \tag{2.34}$$

Where $\mathcal{F}^{-1}(.)$ represents the inverse Fourier transform. As expressed above we get $\mathcal{F}^{-1} \circ \mathcal{F} = \Phi \Phi^T = \mathrm{I}$.

Convolution Theorem

Let's define the convolution theorem as follows [52][40]:

$$\mathcal{F}(w * h) = \mathcal{F}(w) \odot \mathcal{F}(h) \Longrightarrow w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h)) \tag{2.35}$$

Where $\odot$ is the Hadamard product [56]. This theorem makes the equivalence between the convolution operation in the original domain and in the Fourier domain [52].

Spectral graph convolution

Having defined the spectral concept in the graph domain, we can use them to extend the convolution to spectral-based graph convolution. To do this, let's start with Eq(2.35) :

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h)) \tag{2.36}$$

Where $w \in \mathbb{R}^{n \times 1}$ is a spatial filter and $h \in \mathbb{R}^{n \times 1}$ a graph signal. By using Eq(2.33) and Eq(2.34) we get:

$$w * h = \Phi \left(\hat{w} \odot \Phi^T h \right) \tag{2.37}$$

with $\hat{w} \in \mathbb{R}^{n \times 1}$ a column vector that represents the Fourier transform of w evaluated at each mode of the graph spectrum (eigen-values) as illustrated in Figure 2.15 [52][40]. Thus, by defining $\hat{w}(\Lambda) = diag(\hat{w}) \in \mathbb{R}^{n \times n}$, we can replace the Hadamard product [56] in Eq(2.37) by a matrix multiplication [40]:

$$w * h = \Phi \hat{w}(\Lambda) \Phi^T h = \hat{w}(\Phi \Lambda \Phi^T) h \tag{2.38}$$

And, by using the graph Laplacian $\Delta \in \mathbb{R}^{n \times n}$ described in Eq(2.30), we can define the spectral graph convolution as follows [40]:

$$w * h = \hat{w}(\Delta) h \tag{2.39}$$

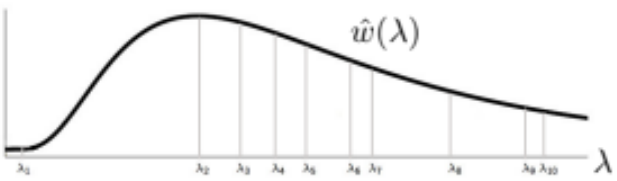

Figure 2.15: Spectral filter illustration. Picture taken from `https://drive.google.com/file/d/1oq-nZE2bEiQjqBlmk5_N_rFC8LQY0jQr/edit`

The spectral graph convolution between spatial filter w and graph signal h consists in transforming the spatial filter into a spectral filter thanks to the Fourier transform and applying it to the graph Laplacian before multiplying it with h [40][53][57].

Spectral graph convolutional layer

We have defined spectral graph convolution. This mathematical concept can be extend in the deep-learning paradigm. Let's extend Eq(2.39) in the form of a graph convolutional layer [57][40]:

$$h^{l+1} = \eta\left(w^l * h^l\right) = \eta\left(\hat{w}^l(\Delta)h^l\right) = \eta\left(\phi\hat{w}^l(\Lambda)\phi^\top h^l\right) \tag{2.40}$$

Where η is a non-linear activation function and h^{l+1} is the node feature at layer $l+1$. In this representation the filter $\hat{w}$ is learned during training.

Representation The spectral graph convolutional network can be defined as a class of graph convolutional network that learn a spectral filter to make prediction based on oscillation pattern in the graph topology [58][55]. Specifically, it defines the convolutional operation in the frequency domain, using the eigenvectors and eigenvalues of the graph Laplacian. The Laplacian matrix encodes the structure of the graph while the eigenvectors and eigenvalues capture the "frequency" content of the graph signal [53]. By applying convolution in the frequency domain, we are able to capture the local neighborhood information of each node in the graph[58][55]. Figure 2.17.left represents a graph signal in the vertex domain with colors representing the magnitude of the graph node (vertex). Figure 2.17.right represents the same graph signal in the spectral domain. We can see the different modes (eigenvalues λ) and observe that the main energy content of the signal is present at low frequency (first principal Fourier basis)[58].

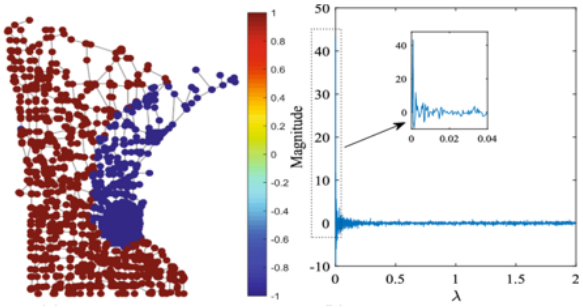

Figure 2.16: Graph signal in vertex domain (left) and in spectral domain (right). Picture taken from [58]

Limitations and improvements

One limitation of Spectral GCN is that it requires the computation of the eigenvectors and eigenvalues of the Laplacian matrix, which can be computationally expensive for large graphs [59].

SplineGCN [60] addresses this limitation by using piecewise polynomial functions, called splines, to approximate the convolutional operation [60]. This allows SplineGCN to perform convolution on graphs more efficiently, without the need to compute the eigenvectors and eigenvalues of the Laplacian matrix [60].

LapGCN [61] uses a more general graph convolutional operation that is defined directly in terms of the graph adjacency matrix, rather than the Laplacian matrix. This allows LapGCN to more accurately capture the structure of the graph and the relationships between nodes [61].

ChebNet [62] proposes to use a Chebyshev polynomials [63] to approximate the convolutional operation. This allows ChebNet to perform convolution on graphs more efficiently without the need to compute the eigenvectors and eigenvalues [62].

CayleyNet [64] defines the convolutional operation in terms of the Cayley transform [65] of the graph adjacency matrix. The Cayley transform is a matrix function that closely connects to the eigenvectors and eigenvalues of the adjacency matrix [65]. By using the Cayley transform, CayleyNet can capture the local neighborhood information of each node in the graph while also having a more flexible formulation than Spectral GCN [64].

In summary, the previous models differ in the way they define the convolutional operation, with some using spectral graph theory (Spectral GCN [57], CayleyNet [64]) and others using polynomial functions (SplineGCN [60], ChebNet [62]) or a more general formulation (LapGCN [61]).

2.3.4 Spatial Graph Convolution

The second method to define graph convolution consists in extending the definition of the convolution in Eq(2.26) from the euclidean domain to the vertex domain. As introduced in Section 2.3.2, the problem is that the numbers of adjacent neighbors can vary and do not have any ordering [40]. To overcome these limitations, we have to find a function $f(.)$ that maps adjacent nodes $j \in N_i$ with the targeted node i. Thus, the spatial graph convolution problem can be written as follows [50] :

$$h_i^{\ell+1} = f\left(h_i^{\ell}, \left\{h_j^{\ell} : j \to i\right\}\right) \tag{2.41}$$

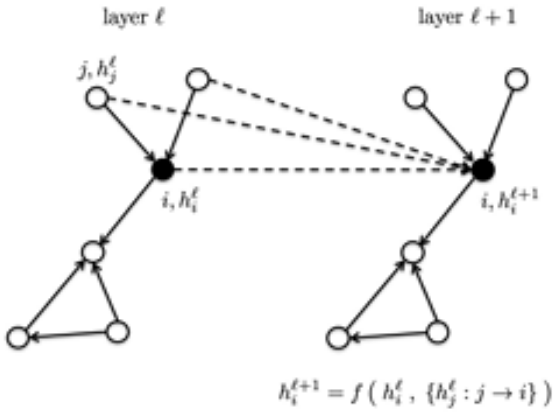

Figure 2.17: Spatial graph convolution formulation. Picture taken from [50]

h_i^{l+1} represents the feature vector in the targeted node i at layer $l+1$. Hence, the function $f(.)$ aims to create spatial graph convolution by using the feature vector h_j^l of nodes j that are connected to the current node i [50][40]. Hence, we leverage the problems introduced in Section 2.3.2 because we are independent of any vertex reparametrization. There are several ways to define this function $f(.)$. However, it is possible to categorize them into two groups: isotropic and anisotropic graph convolution.

Isotropic Graph Convolutional Network

This class of GCN is called "isotropic" because it applies the same convolutional operation to all nodes in the graph, regardless of their number, position, and connectivity strength between them.

Vanilla GCN [66][67]

This simple GCN defines the function $f(.)$ by taking the mean value of the neighboring nodes' feature vectors and convolving the resulting vector before passing in an activation function. One mathematical representation can be written as follows [50][67]:

$$\begin{aligned} h_i^{\ell+1} &= \text{ReLU}\left(U^{\ell}\,\text{Mean}_{j\in N_i}\, h_j^{\ell}\right) \\ &= \text{ReLU}\left(U^{\ell}\frac{1}{\deg_i}\sum_{j\in N_i} h_j^{\ell}\right) \end{aligned} \tag{2.42}$$

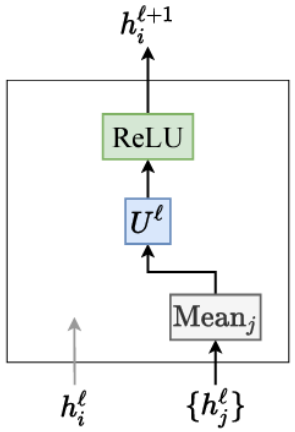

Figure 2.18: Block diagram of Vanilla GCN [66][67]. Picture taken from [50]

Where $U^{\ell} \in \mathbb{R}^{d\times d}$ is a learnable convolutional filter and $\deg_i$ is the degree (number of in-connection) of node i. ReLU is a threshold-based activation function that puts every negative element to zero [32]. Here the convolution consists of a weighted average of the neighboring nodes' features. As illustrated in Figure 2.18, the self-information is not used to update the corresponding node.

Spectral to spatial GCN It is possible to show that the spatial GCN [66] is a simplification version of spectral ChebNets [62][50]. The complete demonstration and additional interpretation are provided in [68].

GraphSage [69]

The authors of [69] propose to define the function $f(.)$ in Eq(2.41) as an aggregation function. The goal is to define a function determining how to aggregate the neighborhood information [6]. To do that, let's define the GraphSage equation as follows [50]:

$$\hat{h}_i^{\ell+1} = \sigma\left(U^\ell \text{ Concat } \left(h_i^\ell, \text{Aggregate}_{j\in\mathcal{N}_i}\, h_j^\ell\right)\right) \tag{2.43}$$

Where $U^\ell \in \mathbb{R}^{d\times 2d}$ is a learnable matrix that can be decomposed in two sub-matrix $W_{neig}^\ell \in \mathbb{R}^{d\times d}$ and $W_{self}^\ell \in \mathbb{R}^{d\times d}$ representing the weighted impact between the aggregation of the features vector of the neighbors with the features vector of the central node. σ is an activation function, and $\text{Aggregate}_{j\in\mathcal{N}_i}$ is the aggregation function applied on the k-hop sampled adjacent nodes as described in Figure 2.19. This figure represents the GraphSage pipeline consisting of the different steps described below.

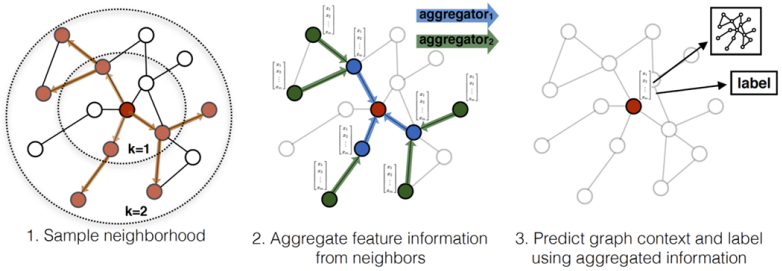

Figure 2.19: Visualisation of GraphSage pipeline. Picture taken from [69]

K-hop sampling Given central nodes, GraphSage samples the neighboring nodes according to the strength of the connection (edges features e_{ij}) or randomly if the edges features are not present [6]. Once done, it is possible to use the features in the nodes within a k-hop distance of the central node. K is a hyper-parameter representing the nodes that can be reached from the central node in k-step [69]. This concept is illustrated in Figure 2.19.

Aggregation function Once the neighboring nodes were chosen, GraphSage would aggregate information from them as defined in Eq(2.43). The authors of [69] define different aggregation functions. The three aggregation functions are defined as follows [50]:

$$\begin{aligned}
\hat{h}_i^{\ell+1} &= \sigma\left(U^\ell \text{ Concat } \left(h_i^\ell, \text{Mean}_{j\in\mathcal{N}_i}\, h_j^\ell\right)\right) \\
\hat{h}_i^{\ell+1} &= \sigma\left(U^\ell \text{ Concat } \left(h_i^\ell, \text{Max}_{j\in\mathcal{N}_i}\, \sigma\left(V^\ell h_j^\ell\right)\right)\right) \\
\hat{h}_i^{\ell+1} &= \sigma\left(U^\ell \text{ Concat } \left(h_i^\ell, \text{LSTM}_{j\in\mathcal{N}_i}^\ell\left(h_j^\ell\right)\right)\right)
\end{aligned} \tag{2.44}$$

Where $V^\ell \in \mathbb{R}^{d\times d}$ is a learnable matrix and $\text{LSTM}_{j\in\mathcal{N}_i}^\ell$ is a LSTM cell [70] widely used in RNN. Thus, we have a mean aggregation that will make a neighbors' average, a max pooling aggregation that aggregates only the more impacting features, and a LSTM-based aggregation that can learn how to aggregate considering adjacent nodes feature as a sequence [38][6].

Normalisation Once the aggregation is made for every node in the graph, the node features are normalized by projection on ℓ_2-unit ball as follows [50]:

$$h_i^{\ell+1} = \frac{\hat{h}_i^{\ell+1}}{\left\|\hat{h}_i^{\ell+1}\right\|_2} \tag{2.45}$$

GraphSage method has shown excellent performance in several graph-based tasks. It gives an excellent paradigm to represent the spatial graph convolution. Moreover, it is well known for processing large graphs with low complexity and with strong inductive capability [69][40][50]. However, it is limited to isotropic capability by treating each neighbor with the same weight [50].

Anisotropic Graph Convolutional Network

To overcome the isotropic limitation of isotropic GCN [69], we want to include anisotropy in the model in order to weigh the aggregation with respect to the importance of neighboring nodes. There are multiple options to include anisotropy. To demonstrate this point, let's continue with GraphSage's definition [69] made in the previous section.

Implicit anisotropy Let's suppose that the graph used to process GraphSage has edges weight $e_{ij} \in \mathbb{R}$ indicating the strength of the connection between node i and node j. Hence, the first way to include anisotropy in the model can be done as follows [71][2] :

$$\hat{h}_i^{\ell+1} = \sigma\left(U^\ell \text{ Concat } \left(h_i^\ell, \text{Mean}_{j\in\mathcal{N}_i}(e_{ij} \odot h_j^\ell)\right)\right) \tag{2.46}$$

By using a mean aggregator, we can see that - in this case - the aggregation will be made by making a weighted average of the neighboring nodes according to the strength of the connection without adding any model parameters [50]. So it can be seen as a natural anisotropy, and the graph topology is kept fixed across the different layers [6].

Explicit anisotropy Now, suppose that the edge weights are not explicitly available. In this case, we can include explicit anisotropy as a joint representation of neighboring node features at each layer as follows [50]:

$$\begin{aligned} \hat{h}_i^{\ell+1} &= \sigma\left(U^\ell \text{ Concat } \left(h_i^\ell, \text{Mean}_{j\in\mathcal{N}_i}(e_{ij}^\ell \odot h_j^\ell)\right)\right) \\ e_{ij}^\ell &= A^\ell\left(h_i^{\ell-1} + h_j^{\ell-1}\right) \end{aligned} \tag{2.47}$$

Where $A^\ell \in \mathbb{R}^{d\times d}$ is a learnable matrix allowing to produce dynamic edge features e_{ij}^ℓ at each layer based on the pair-wise node features $h_i^{\ell-1}$ and $h_j^{\ell-1}$ at the precedent layer [6]. After, we will see that there are more techniques to represent dynamic edge features by, for example, defining a pair-wise similarity metric [6]. It is important to note that in this representation we dynamically change the graph topology across the layers.

Implicit & explicit anisotropy: The precedent concepts can be extended even if the edges weight are available. The idea is to begin with an initial fixed graph topology and change it dynamically as the nodes features change also across the layers [6]. In this case, we will see after that there are more method to do that, but here is an example to represent this concept [50]:

$$\begin{aligned} \hat{h}_i^{\ell+1} &= \sigma\left(U^\ell \text{ Concat } \left(h_i^\ell, \text{Mean}_{j\in\mathcal{N}_i}(\hat{e}_{ij}^\ell \odot h_j^\ell)\right)\right) \\ \hat{e}_{ij}^\ell &= A^\ell\left(h_i^{\ell-1} + h_j^{\ell-1}\right) + B^\ell e_{ij}^{\ell-1} \\ e_{ij}^{\ell+1} &= e_{ij}^\ell + \sigma\left(\hat{e}_{ij}^\ell\right) \end{aligned} \tag{2.48}$$

Where $B^\ell \in \mathbb{R}^{d\times d}$ is a learnable matrix and $\hat{e}_{ij}^\ell$ computes the impact between the joint representation of nodes features and the edge features itself [50].

Flexibility and representation It's important to note that the edge features are not necessarily a scalar. The flexibility of GCN allows it to represent edges features as a n-dimensional vector [72]. The role of the learnable matrix is also to project different information sizes in the same dimensional space. All the previous concepts can be extended for any nodes/edges features size [72][6]. The only constraint is that the node/edges features of every element in the graph need to be constant even if their numbers can change [71]. Hence, we see how to include anisotropy the GraphSage GCN. Anisotropy is very important in treating graph problems, especially in real-world situations where the interaction is, in most cases, different between each entity we want to represent. The authors of [50] show that anisotropic GCN outperform isotropic GCN in most benchmark graph problems. In Isotropic GCN, we learn how a node can aggregate from other nodes while anisotropic GCN can also learn from the graph topology (implicit anisotropy) and also learn the topology dynamically (explicit anisotropy).

[2] `https://docs.dgl.ai/en/0.8.x/generated/dgl.nn.pytorch.conv.SAGEConv.html`

Graph Attention Network (GAT) [73][74]

The concept of anisotropic spatial Graph Convolutional Network can be linked with the concept of attention [75] described in Section 2.2.3 on transformers [41]. To do that, the authors of [74] include explicit anisotropy by using the concept of multi-head attention (explained in Section 2.2.3) introduced by [41]. Hence, the equation of GAT can be written as follows [50]:

$$h_i^{\ell+1} = \text{Concat}_{k=1}^{K} \left(\text{ELU} \left(\sum_{j \in \mathcal{N}_i} e_{ij}^{k,\ell} U^{k,\ell} h_j^{\ell} \right) \right) \tag{2.49}$$

Where $U^{k,\ell}$ represents a k-head learnable matrix and ELU is an activation function [76][32]. Here $e_{ij}^{k,\ell}$ represents the multi-head attention coefficient represented as edges weight. Hence, by the definition of multi-head self-attention defined in Eq(2.20) and the softmax function defined in Eq(2.13), the edge-based attention update layer can be represented as follows [50] :

$$\begin{aligned} e_{ij}^{k,\ell} &= \frac{\exp\left(\hat{e}_{ij}^{k,\ell}\right)}{\sum_{j' \in \mathcal{N}_i} \exp\left(\hat{e}_{ij'}^{k,\ell}\right)} \\ \hat{e}_{ij}^{k,\ell} &= \text{LeakyReLU}\left(V^{k,\ell}\, \text{Concat}\left(U^{k,\ell} h_i^{\ell}, U^{k,\ell} h_j^{\ell}\right)\right) \end{aligned} \tag{2.50}$$

Where $V^{k,\ell} \in \mathbb{R}^{\frac{2d}{K}}$ represents the k learnable matrix and LeakyReLU is an activation function [76][32].

Representation Here, the multi-head attention mechanism is used to know how the central node needs to be "attentive" to the adjacent nodes [6][73]. In the section on transformers, we demonstrate how self-attention can represent the interaction between sequenced input data. For GAT, the idea is exploited in the same way by computing the softmax function to indicate how an edge connection is stronger compared to the others while multi-head is used to increase learning capacity [77][6]. The attention score is computed dynamically and can change the graph topology across the layers.

Variants

Instead of using the LeakyReLU and the concatenation with the same projection matrix $U^{k,\ell}$ to represent $\hat{e}_{ij}^{k,\ell}$, the authors of ([77]) propose to use a different version :

$$h_i^{(l+1)} = \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} W_{right}^{(l)} h_j^{(l)} \qquad \begin{aligned} \alpha_{ij}^{(l)} &= \text{softmax}_1\left(e_{ij}^{(l)}\right) \\ e_{ij}^{(l)} &= \vec{a}^{T(l)}\, \text{LeakyReLU}\left(W_{left}^{(l)} h_i + W_{right}^{(l)} h_j\right) \end{aligned} \tag{2.51}$$

Where $W_{left}^{(l)}$ and $W_{right}^{(l)}$ are learnable matrices representing the impact between the central and adjacent nodes, respectively. The authors of [73] also propose a dot product version of GAT:

$$h_i^{(l+1)} = \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} h_j^{(l)} \qquad \begin{aligned} \alpha_{i,j} &= \text{softmax}_i\left(e_{ij}^{l}\right) \\ e_{ij}^{l} &= \left(W_i^{(l)} h_i^{(l)}\right)^T \cdot W_j^{(l)} h_j^{(l)} \end{aligned} \tag{2.52}$$

Alternatively, the authors of [78] propose to define a projection matrix based attention computed as a Cosine similarity [79] between node features to model pair-wise similarity :

$$H^{l+1} = PH^{l} \quad P_{ij} = \text{softmax}_i\left(\beta \cdot \cos\left(h_i^{l}, h_j^{l}\right)\right) \tag{2.53}$$

Where β is a learnable scalar. Additionally, the authors of [80] propose to use edge embedding to jointly represent the interaction between nodes in a higher dimensional point of view [71]:

$$\begin{aligned} e_{ij} &= \vec{F}\left(f'_{ij}\right) \\ f'_{ij} &= \text{LeakyReLU}\left(A\left[h_i \,\|f_{ij}\|\, h_j\right]\right) \end{aligned} \tag{2.54}$$

Where $f'_{ij}(.)$ is a learnable projection function and f_{ij} is a n-dimensional edge feature vector.

2.3.5 Graph structure learning

We have seen some examples of how to include anisotropy in a GCN layer. We saw anisotropic GraphSage [69] and attention-based anisotropy [41], [73], [74], [77], [80]. These examples show that it is possible to learn graph topology. There are many methods to learn the adjacency matrix of a graph. This section aims to summarize them and constitute an important point for further analysis in this thesis.

Unsupervised graph properties-based structure learning

Here we introduce properties-based constraints to learn the graph topology [81]. These constraints constitute a part of the loss function used to learn model parameters impacting the graph's adjacency matrix. We will see the fitness, fmoothness, connectivity, and sparsity constraints. For the following parts, Let us assume $A \in \mathbb{R}^{n \times n}$ the graph adjacency matrix and $h_i \in \mathbb{R}^d$ the feature vector of node i. $H \in \mathbb{R}^d$ is the matrix representation of the n-node feature vector.

Fitness To define this concept, let us assume that every node feature can be expressed as a linear combination of its neighbors. Then, the fitness constraint aims to minimize [6] :

$$\sum_i \left\| h_i - \sum_j A_{i,j} h_j \right\|^2 \tag{2.55}$$

Where $\sum_j A_{i,j} = 1, A_{i,j} \geq 0$.

Smoothness The smoothness constraint uses the graph Laplacian defined in Section 2.3.3 measuring the smoothness and the spectrum of the graph [40]. Thus, the smoothness constraint aims to impose a smooth graph topology by minimizing the following loss function [6] :

$$\Omega(A, H) = \frac{1}{2} \sum_{i,j} A_{i,j} \|h_i - h_j\|^2 = \operatorname{tr}\left(H^\top \Delta H\right) \tag{2.56}$$

Where Δ is the graph Laplacian defined in Eq(2.27) and $tr(.)$ is the trace operator. $\Omega(A, H)$ is also know as the Dirichlet energy function [82]. Minimizing this energy function will impose similar node features in the graph [81].

Connectivity and Sparsity Theses constraints are in opposition. The first one aims to force graph connectivity, while the second aims to impose sparsity in the connections [81]. These two constraints act as regulators to manipulate the graph topology [81][6]. These two constraints can be defined as follows [6]:

$$-\alpha \overrightarrow{1}^\top \log(A \overrightarrow{1}) + \beta \|A\|_F^2 \tag{2.57}$$

Where α and β are hyperparameters that weigh the impact of each constraint. The Frobenius norm $\|\|_F^2$ will push adjacency matrix elements to zeros while the logarithmic barrier will block the trivial solution $a_{ij} = 0 \forall i, j = 0, ..., n$ [81].

Learning weighted graph structure

We have already seen the attention-based graph learning structure allowing us to learn a weighted graph adjacency matrix. Moreover, it is also possible to define an edge weight based on a similarity metric.

Similarity metric The similarity/distance metric defines the similarity/distance between two nodes. Then, the objective is to find a metric that can project pairwise node information in scale representing how two nodes are similar. Then the problem formulation is the following :

$$a_{ij} = e_{ij} = \operatorname{sim}(h_i, h_j) \tag{2.58}$$

There are a lot of metrics to use. The most used is the Cosine similarity [79] and the Euclidean distance.

Kernel-base similarity This method tries to extend the concept of similarity metric to include statistical distribution properties as follows [6]:

$$\begin{aligned} d\left(\vec{h}_i, \vec{h}_j\right) &= \sqrt{\left(\vec{h}_i - \vec{h}_j\right)^\top M \left(\vec{h}_i - \vec{h}_j\right)} \\ S\left(\vec{h}_i, \vec{h}_j\right) &= \frac{-d\left(\vec{h}_i, \vec{h}_j\right)}{2\sigma^2} \end{aligned} \tag{2.59}$$

Where M is the covariance matrix, σ is the standard deviation of Gaussian distribution, and $d(.)$ represents the Mahalanobis distance [83].

Learning discrete graph structure

Probability-based sampling This method defines a probability function representing the probability that two nodes are connected, and sample edge from it [84]. Some people used the Bernoulli distribution [85] that can be learned in backpropagation [86] while other people used a probability distribution according to pairwise node similarity [6]:

$$p_{i,j} = e^{-t\|h_i - h_j\|} \tag{2.60}$$

Where t is a learnable temperature parameter. To get a discrete graph structure, it is possible to use Bayesian-based sampling technique or other sparsification techniques [84][86][6].

Graph sparsification techniques As introduced in GraphSage [69], it is possible to sample edges according to the strength of the connectivity. If we want to be sure that every node in the graph keeps a minimum number of connections, a KNN-based sampling can be done as follows [6]:

$$A_{i,:} = \text{TopK}\left(e_{i,:}\right) \tag{2.61}$$

Where TopK is an operator that keeps the top k neighboring nodes from every node in the graph [87]. For example, let's suppose that we have a fully connected graph with ten nodes; by applying this operator with k = 3, every node in the graph will have three adjacent connections instead of 10 initially. Then, the number of edges in the graph goes from 100 to 30. Another way to sparsify a graph consists of applying a threshold on the edge's weight as follows [6]:

$$A_{i,j} = \begin{cases} e_{i,j} & e_{i,j} > \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{2.62}$$

Where ϵ is the threshold. Here, we are not guaranteed that every node will have a connection. Then, we can have an isolated node (zero in degree)[87]. However, this method can be compelling when we have an unbalanced graph structure [87].

2.3.6 Task-based Graph Convolutional Network

So far, we have defined the concept of convolution in the spectral domain and the vertex domain. These definitions allow to be invariant to node ordering, and graph topology [50]. We have seen that it was possible to learn a spectral filter thanks to the graph Laplacian while for spatial graph convolution, it is possible to learn a way to aggregate the neighborhood information [58]. We have also seen that including anisotropy in the convolution allows us to learn the graph topology (adjacency matrix) statically and dynamically [6]. We have also seen the different ways to learn the graph structure under multiple criteria [6]. Having learned about these background concepts for graphs, this section aims to show how to include them in a global architecture to perform prediction-based tasks.

Graph Convolutional Network (GCN) pipeline

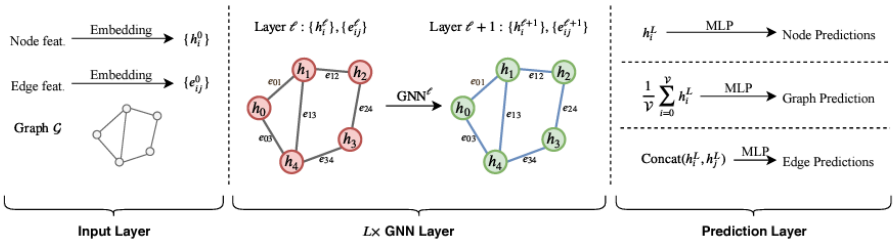

Figure 2.20: Visualisation of typical GCN pipeline. Picture taken from [50]

As illustrated in Figure 2.20, the typical pipeline to use GCN consist in the following layers :

- **Input layer** This layer consists of creating a graph representation. Some datasets provide directly graph-structured data, but it's not always the case. Thus, this layer is used to construct a graph by assigning node/edge/graph features depending on the application. To do that, an embedding can be done to have the expected features dimension/representation.
- **GCN layer:** This layer consist in applying GCN layer multiple times. As we have seen, this layer can dynamically update nodes/edges features.
- **Prediction layer:** Depending on the application, the processed graph is used to make the desired prediction. It could be graph prediction, edge prediction, or node prediction. Depending on the type of prediction, how to represent the input information will differ in the MLP-based predictor.

The prediction can be classification, regression, etc... this prediction can be made in supervised ways as well as other representation learning introduced in Section 2.1.1. The concepts presented in the introduction Section 2.1 are still valid in the graph paradigm. The graph-based model parameters are still learned by backpropagation using the gradient-descent-based method from a loss function. These parameters can be regularised, and the global problem in deep learning is still present in the graph paradigm (overfitting, vanishing gradient, ..).

Graph Convolutional Network (GCN) for node classification

In the case of supervised learning for the node classification task, we have to pose the problem of graph construction/representation and the convolutional layer to choose the model to classify the nodes appropriately.

Graph representation As the objective is to classify the node, we typically represent the node features as an embedding of the self-information associated with the corresponding node. The edge features are used to determine the relationship between nodes. It could be a joint representation of associated nodes features or side information that is not used in the nodes features. Moreover, the edges feature can represent binary connectivity between two nodes without specifically related values.

Graph convolutional layer The idea driving the choice of the GCN layer is to enrich the node features' information by correctly aggregating the information from the other nodes. For that, there are several methods as described in Section 2.3.4, Section 2.3.3, and Section 2.3.5.

Prediction layer At the end of the convolutional part of the network, we expect that each node features vector has a meaningful representation according to the classification task. Hence, we can obtain the final prediction by passing each node features vector in a MLP classifier as follows [50]:

$$y_{i,\text{ pred}} = \text{softmax}(P\,\text{ReLU}\left(Qh_i^L\right)) \tag{2.63}$$

Where $P \in \mathbb{R}^{d\times C}$ and $Q \in \mathbb{R}^{d\times d}$ are the layer-wise projection matrix of the MLP classifier and C is the number of classes. To learn these parameters, a cross-entropy loss is typically used.

2.4 Hypergraph Neural Network

So far, we have represented a graph as a set of nodes connected by edges. So each edge in the graph results from the connection between exactly two nodes. However, extending the graph domain to a higher representation level is possible. It is in this context that Hypergraphs are briefly introduced in this section.

2.4.1 Hypergraph domain

Let's extend the definition of graphs to hypergraphs. To do that, let's define a hypergraph $G = (V, E, H)$ where V, E, H represent [88] :

- **Vertices** $V = \{v_i\}_{i=1}^{n}$ indexing the n vertices v_i in the hypergraph.
- **Hyperedges** $E = \{e_k = (v_{i_1}, ..., v_{i_n}) | \forall v_i \in e_k\}_{k=1}^{m}$ the set of m hyperedges indexing a group of multiple vertices v_i contained in hyperedge e_k.
- **Incidence matrix** $H \in \mathbb{R}^{n \times m}$ where h_{ik} corresponds to the presence of a connection between vertex i and hyperedge k.

The graph can store information as [40]:

- **Vertex features:** $x_i \in \mathbb{R}^{d_v}$
- **Hyperedge features:** $e_k \in \mathbb{R}^{d_e}$
- **Hypergraph features:** $g \in \mathbb{R}^{d_g}$

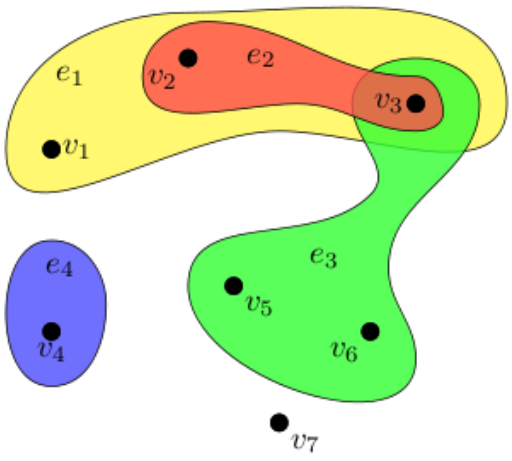

Figure 2.21: Hypergraph representation. Picture taken from `https://tex.stackexchange.com/questions/1175/drawing-a-hypergraph`

The concept of a hypergraph is illustrated in Figure 2.21. We can see that in this representation a hyperedge can connect multiple vertices. The same reasoning in the previous section on graphs can be done to demonstrate hypergraph properties. We can define hypergraph Laplacian and spectral hypergraph convolution as well as isotropic and anisotropic spatial convolution by extending the concept of attention mechanism for hypergraph [88][89]. However, this background chapter aims to introduce the concepts that will be used in the work of this thesis. For more information, the authors of [90],[89], and [88] propose an intuitive presentation of all hypergraph concepts. Nevertheless, one hypergraph convolution will be presented to understand the rest of the work.

2.4.2 Hypergraph convolution

Let's define one hypergraph convolutional layer as follow [89]:

$$X^{\ell+1} = \sigma\left(D^{-1}HWB^{-1}H^{T}X^{\ell}P\right) \qquad (2.64)$$

Where $X^{\ell} \in \mathbb{R}^{n \times d_h^{\ell}}$ contains the n vertex features at layer ℓ with hidden dimension d_h^{ℓ}. $P \in \mathbb{R}^{d_h^{\ell} \times d_h^{\ell+1}}$ represents the projection matrix making the transformation from layer ℓ to layer $\ell+1$. The incidence matrix $H \in \mathbb{R}^{n \times m}$ is represented in Figure 2.22 and defined above. $D \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{m \times m}$ are vertex and hyperedge degree matrices, respectively, and are both diagonal. The concept of degree is the same as in the graph. The vertex degree matrix counts the number of hyperedges connectivity for each vertex while the hyperedge degree matrix counts the number of vertices included in each hyperedge [90]. $W \in \mathbb{R}^{m \times m}$ is the hyperedges weights matrix. It is a diagonal matrix representing the importance of each hyperedge in the hypergraph (hyperedges anisotropy) [88].

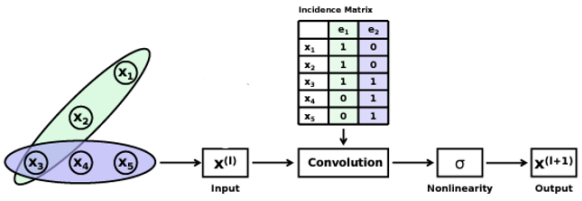

Figure 2.22: Hypergraph convolutional layer representation. Picture modified from [89]

Chapter 3

Related Work

3.1 Related Work

So far, we have seen all the theoretical concepts related to deep learning, graphs, and hypergraphs, as well as the implementation of a general pipeline to build a prediction model. This section will show how some authors have concretized these concepts for implementing a model based on patient prognosis. For this, a literature review was done according to several filter criteria. For that, we will base ourselves on the requirements allowing us to answer the research question of this thesis: "Is it possible to extract additional information from a graph representation of a population level to make a patient prognosis? " Then, the fundamental concepts used for the literature search are:

- "GCN" or hypergraphs
- "Population-based graph representation."
- "Patient prognosis."
- "Multimodal clinical data."

We are looking for work that uses multimodal medical data. That is, data of different types, such as image data, demographic data, symptomatic data, medical history, and other available measures. Based on the data, we will see the techniques used to merge the different modes to extract useful information for prediction. Furthermore, we will see how the authors use the graphs to solve the problem and their research hypotheses. As the work of this thesis is focused on graphs, we will structure this section by classifying the articles into two groups:

- Graph models with manual graph construction
- Graph models with dynamic graph topology

Although there are several works in the literature based on hypergraphs [91][92][93][94], no articles representing a patient population have been found in the literature.

3.1.1 Methods with manual graph construction

In this section, methods based on manual graph construction will be presented.

"Fusion of Imaging and Non-Imaging Data for Disease Trajectory Prediction for COVID-19 Patients"

The authors of [95] propose to fuse imaging and non-imaging data to predict the discharge from the hospital, mortality, and ICU admission. The imaging data are the Chest X-ray collected for patients having at least one positive RT-PCR test for COVID-19. The non-imaging data are demographic information like gender, self-reported race, ethnic group, and age group (categorized by ten-year intervals). To fuse imaging and non-imaging data, the authors propose to construct a graph where the nodes represent a 1024-dimensional vector representing the imaging features extracted from a CNN as illustrated in (Figure 3.1). To do that, the idea is to pre-train a CNN for outcome classification and take the feature vector before the MLP part of the network (as explained in 2.2.2). To represent the edges, the authors use both imaging and non-imaging information. The hypothesis is that patients having the same characteristics should have a similar diagnosis. Mathematically, the authors define a similarity function as follows [95] :

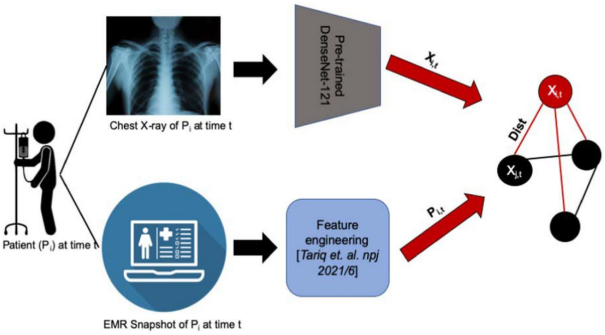

Figure 3.1: Graph representation proposed by [95]. Picture taken from [95]

$$e_{ij} = cos(h_i, h_j) \sum_k \delta(C_i^k - C_j^k) \tag{3.1}$$

Where e_{ij} defines the similarity between patient i and j, cos(.) is the cosine similarity, h_i represent the 1024-dimensional imaging feature vector of patient i and C_i^k represents the non-imaging characteristic k of patient i. Thus, the authors propose to sum the numbers of common patient characteristics by weighing with the imaging feature vector. The authors use GraphSage [69] as a graph convolutional layer and a threshold-based sparsification technique to sample the edges. Here the authors point out that fusing imaging and non-imaging to represent the node embedding doesn't improve the result. They also test different types of non-imaging information to compute the edges and conclude that age and gender are the most expressive characteristics to define the similarity. Then, the authors pose the problem as a node classification problem where each node represents a patient while the edges represent their similarity.

"Multimodal spatiotemporal graph neural networks for improved prediction of 30-day all-cause hospital readmission"

Alternatively, the authors of [96] propose a model able to predict the 30 days of hospital readmission. Similarly to the previous method, the data consist of imaging data (Chest X-rays) and EHR data representing demographic information (age, gender, race, ethnicity) as well as primary diagnoses (heart disease, vascular disease, metabolism disease). In their dataset, the EHR data are more expressive than imaging data for the prediction. Then, the authors propose to create two different graphs. As illustrated in (Figure 3.2), the first one with the imaging feature in the node, and the second one with the EHR features in the nodes. To do that, the imaging feature will be extracted from a pre-trained CNN while the EHR features are represented as a one-hot encoding vector. To represent the edges, the two graphs use the EHR data to define a kernel-based similarity metric as follows [96]:

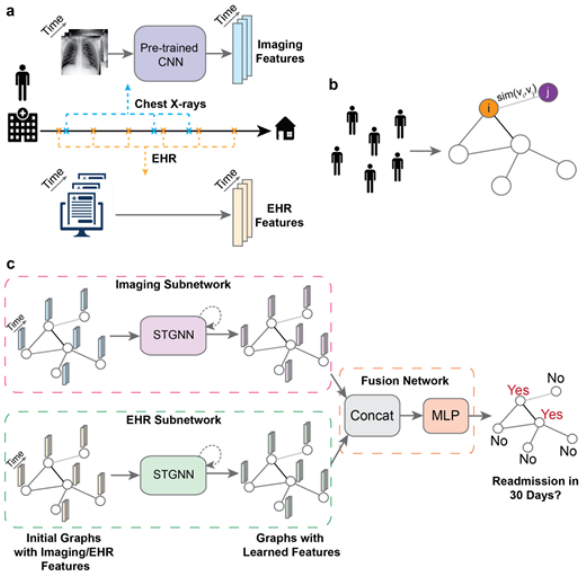

Figure 3.2: Graph model proposed by [96]. Picture taken from [96]

$$e_{ij} = \exp\left(-\frac{\text{dist}\left(v_i v_i\right)^2}{\sigma^2}\right) \text{ if } e_{ij} \geq \text{ threshold, else } e_{ij} = 0. \tag{3.2}$$

Where v_i and v_j are the EHR features for node i and j and $dist(.)$ is the Euclidean distance. σ is the standard deviation of the distances. Here, the authors use both threshold and TopK-based sampling technique to get a sparse graph topology. After the two sub-graph are independently processed by RNN-based graph convolution and the nodes features are concatenated before entering in the MLP classifier.

"Semi-supervised classification of disease prognosis using CR images with clinical data structured graph"

Here, the authors of [97] pose the problem as a semi-supervised learning representation by using a Markov-based graph representation for the population. They use a part of the dataset to include labeled patients in the graph. The model aims to predict ICU admission for COVID patients. Similarly to the previous articles, the dataset contains chest radiography and EHR data for each patient. To construct the graph, the authors use CNN-based technique to represent the nodes with patient imaging features and define a distance metric as follows [97]:

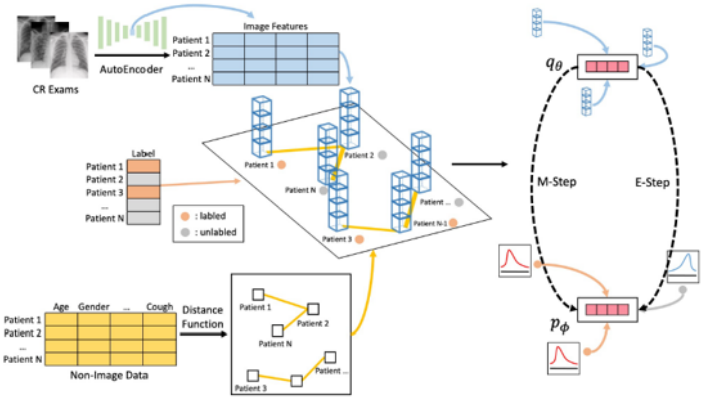

Figure 3.3: Graph model proposed by [97]. Picture taken from [97]

$$e_{ij} = \sqrt{\sum_k (C_i^k - C_j^k)^2} \tag{3.3}$$

This representation is similar to [95] except that they do not use the node's information and choose to use Euclidean distance between the EHR data C^k. In this case, the TopK sampling is used with $k = 3$, meaning that every node is connected with itself and two patients, among which some are labeled or not. The authors also analyze the impact of different distance metrics like Euclidean, Minkowski, and Manhattan distance.

"Predicting Patient Outcomes with Graph Representation Learning"

In this article [98], There is only the presence EHR data. The outcomes are the mortality, length of stay in the hospital, and the ICU admission. To construct the graph, the authors transform the categorical data in a multi-hot vector for each patient [98]. Then each patient's one-hot vector will be assigned as node features. To represent the edges, a similarity score was defined as follows [98]:

$$e_{ij} = a \overbrace{\sum_{\mu=1}^{m} \left(\mathcal{D}_{i\mu}\mathcal{D}_{j\mu}\left(d_{\mu}^{-1}+c\right)\right)}^{\text{Shared Diagnoses}} - \overbrace{\sum_{\mu=1}^{m}\left(\mathcal{D}_{i\mu}+\mathcal{D}_{j\mu}\right)}^{\text{All Diagnoses}} \tag{3.4}$$

Where $\mathcal{D} \in \mathbb{R}^{n\times m}$ represent the multi-hot diagnosis matrix containing the m unique diagnoses of the n patients. d_μ is the occurrence of diagnosis μ. a and c are hyper-parameters ($c = 0.001$ and $a = 5$). The authors try to use both threshold and TopK-based sampling and empirically obtain better performance with TopK sampling (with $k = 3$). Here, the idea is to weight the impact of shared characteristics. Their hypothesis is that: "two patients sharing a rare diagnosis is more significant than a common one" [98]. To do that, they add the d_μ^{-1} term in the "shared diagnoses" part of the above equation. Moreover, they also add the "all diagnoses" part in the similarity score to prevent patients with many characteristics. All these concepts are illustrated in (Figure 3.4) where we can see the strength of the connectivity for multiple patient characteristics.

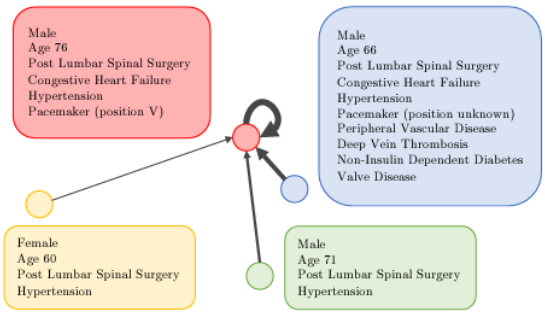

Figure 3.4: Graph construction proposed by [98]. The width of the edge represents the strength of the connection. Picture taken from [98]

"Graph Neural Network Modelling as a potentially effective Method for predicting and analyzing Procedures based on Patient Diagnoses"

The authors of [99] propose a graph-based model for node clustering. The objective is to predict the Therapy Keys (TK) for a patient. The data used for the prediction are the demographic data (age, gender) and the ICD [100] data. The ICD data represent a set of codes that are used as a common reference to classify patient medical status according to the diagnosis or medical procedure applied to a hospitalized patient [100]. Here, the authors use a distance function defined as follows [99]:

$$e_{ij} = \sum_{m=1}^{M} g_{ij}^{m} s_i^m s_j^m + c\left[\delta\left(\mu_i,\mu_j\right)\left(1-\left|\alpha_i-\alpha_j\right|\right)\right] \tag{3.5}$$

Where g_{ij}^m is used to weight to impact between the ICD element m represented by the value s_i^m and s_j^m for patient i and j respectively. $\delta(\mu_i, \mu_i)$ is the Kronecker's delta between binary information μ_i and μ_i (e.g gender). $\alpha_i = \frac{(age)_i}{max(age)}$ is the normalized age of patient i. c is the importance parameter to weight the impact between ICD data and demographic data. The authors also make a complete analysis of the TopK sampling parameters.

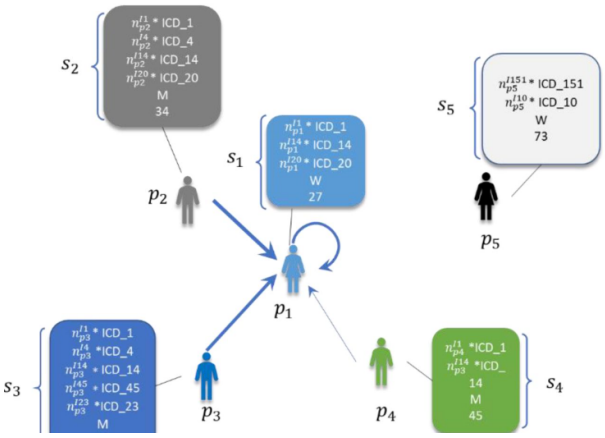

Figure 3.5: Graph model proposed by [99]. Picture taken from [99]

3.1.2 Methods with learned graph topology

"A weighted patient network-based framework for predicting chronic diseases using graph neural networks"

This article [101] is a link between the two sections. The authors propose a manual graph construction but also a dynamic graph topology. They propose a general framework to predict chronic disease with GCN. This framework is described in (Figure 3.6). The authors propose a different approach to constructing the graph. The methodology consists of first creating an undirected bipartite graph between the patient and the disease. After, the graph is projected on the patient side (Figure 3.6 top). Hence, the graph is built by representing edge weight as the number of common diseases [101]. The second part of the framework is the convolutional part. Here the authors analyze two different convolutional layers. The first one is a method based on a fixed adjacency matrix. Then the graph topology is kept fixed across the convolutional step, and the graph convolution layer is defined as follows [101]:

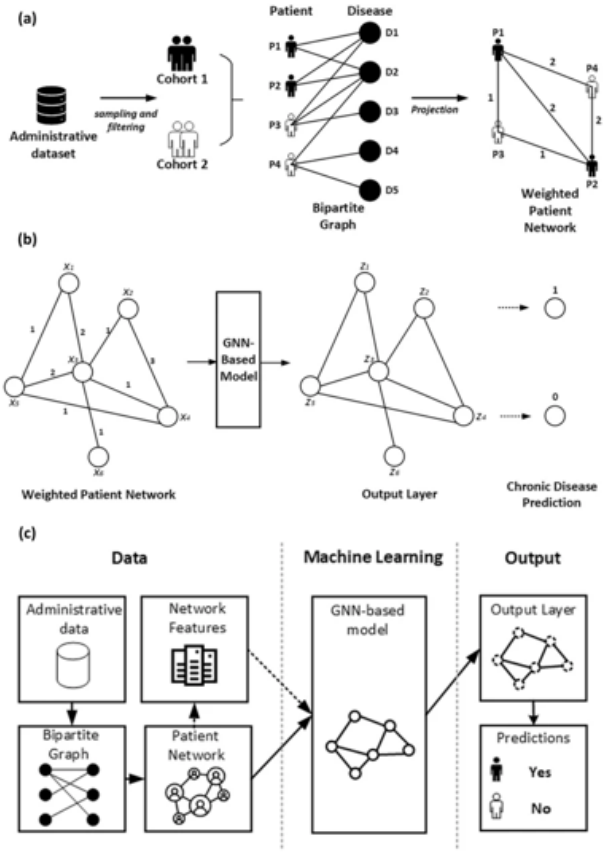

Figure 3.6: Graph model proposed by [101]. Picture taken from [101]

$$H^{(l+1)} = f\left(H^{(l)}, A\right) = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{3.6}$$

Where $H^{(l)}$ is the matrix containing the node feature at layer l, $\tilde{A}$ is the adjacency matrix, and $\tilde{D}$ is the degree matrix. $W^{(l)}$ is the learnable matrix to make the layer-wise transformation with σ the activation function. This equation is a simple spatial GCN where implicit anisotropy is included thanks to the edge weights in the adjacency matrix. The second convolutional layer proposed by the authors allows to change dynamically the graph topology thanks to the attention mechanism [41]. Similar to [73], [74], [77], the author propose the updated edges weight as follows [101] :

$$\alpha_{(u,v)} = \frac{\exp\left(\text{LeakyReLU}\left(a^T\left[Wh_u \| Wh_v\right]\right)\right)}{\sum_{k\in N_u}\exp\left(\text{LeakyReLU}\left(a^T\left[Wh_u \| Wh_v\right]\right)\right)} \tag{3.7}$$

Where $||$ is the concatenation. a and W are learnable weight matrices for projection, this equation represents one of the attention definitions described in (2.3.4). As explained in (2.3.4), the above equation uses explicit anisotropy by learning a joint representation between the features of node u and v as edge weight. Based on the precedent equation, the two nodes' updated rules are defined as follows [101]:

$$\begin{aligned} h'_i &= \|_{k=1}^{K}\sigma\left(\sum_{v\in N_i}\alpha_{uv}^k W^k h_v\right) \\ h'_i &= \sigma\left(\frac{1}{K}\sum_{K=1}\sum_{v\in N_i}\alpha_{uv}^k W^k h_v\right) \end{aligned} \tag{3.8}$$

There is two representation of the use of k multi-head attention in graph convolution. The first equation uses a concatenation $||$ for every head while the second one averages them.

"Latent-Graph Learning for Disease Prediction"

A latent-graph representation is proposed by [102]. The authors defined a general population graph representation. Compared to previous articles, they propose to use the same information to construct nodes and edges in the graph. Moreover, it's important to note that, in this case, the input graph is fully connected (no manual handcraft).

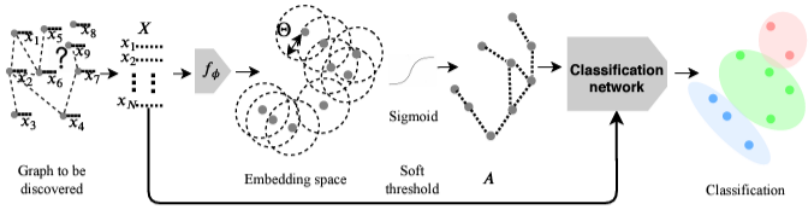

Figure 3.7: Graph model proposed by [102]. Picture taken from [102]

Hence, starting with a fully connected graph, the authors propose to learn an embedding space to project the nodes feature in a lower dimensional space $\tilde{h_i} = f_w(h_i)$. The edge weights are computed as follows [102]:

$$e_{ij} = \frac{1}{1 + e^{-t(\|\tilde{h}_i - \tilde{h}_j\|_2 + \theta)}} \tag{3.9}$$

Where θ and t are learnable threshold and temperature parameters. This function gives edge weight between 0 and 1. One of the novelties here is that the threshold is learned in backpropagation, and there is no need to find it manually. The entire pipeline is described in (Figure 3.7). Similarly, in the previous article, the graph convolutional layer is defined as follows [102]:

$$H^{1+1} = \sigma\left(D^{-1}AH^lW\right) \tag{3.10}$$

The proposed model achieves excellent results in benchmark datasets for multimodal-based disease prediction [102].

"Edge-variational Graph Convolutional Networks for Uncertainty-aware Disease Prediction"

The authors of [103] propose to use spectral convolution for adaptive graphs and design a "pairwise association encoder" to compute edge weight. Similar to previous articles, the authors use imaging data to construct the nodes while using non-imaging data to build the edge weight. To do that, a "pairwise association encoder" is used. As illustrated in (Figure 3.8), it consists of creating an embedding with a MLP composed of a normalization layer, fully connected layer, activation function, and batch normalization layer. After, a scaling version of the cosine similarity between two patients is made as follows [103]:

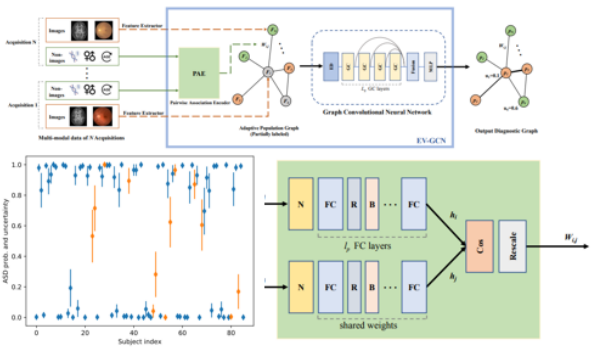

Figure 3.8: Graph model proposed by [103]. Picture taken from [103]

$$e_{ij} = \frac{h_i^\top h_j}{2\,\|h_i\|\,\|h_j\|} + 0.5 \tag{3.11}$$

Where h_i represents the imaging feature vector extracted from CNN-based method. With Eq(3.11), a similarity score between 0 and 1 is computed between each patient to construct a population graph. The authors also propose a graph topology uncertainty analysis thanks to a Monte-Carlo edge dropout as illustrated in (Figure 3.8).

"Multi-Modal Graph Learning for Disease Prediction"

Until now, the feature extraction for multimodal data was made in the preprocessing step. The authors of [104] propose an end-to-end training pipeline by learning the feature extraction and the graph convolutional parameters in the same backpropagation step. Moreover, the authors propose to use a transformer-based multimodal embedding to capture interaction (correlation and complementarity) between different modes (images and EHR data). The full pipeline is illustrated in (Figure 3.9)

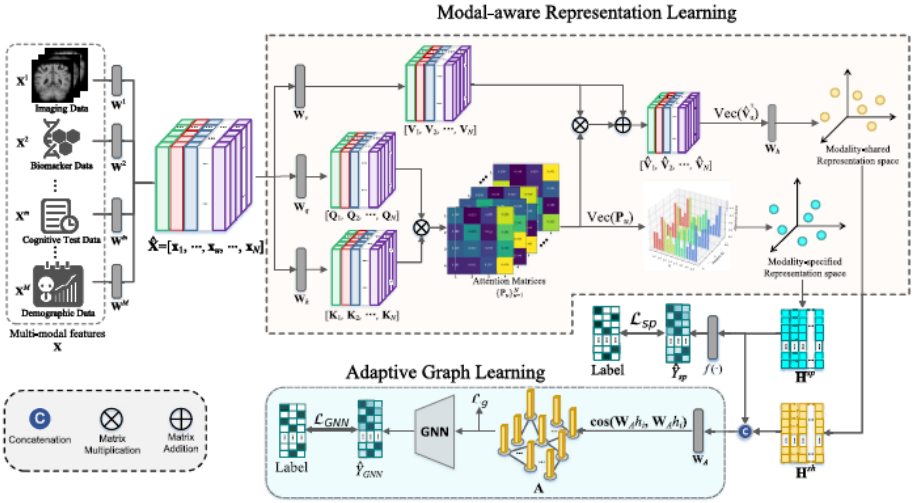

Figure 3.9: Graph model proposed by [104]. Picture taken from [104]

The pipeline consists of a "modal-aware representation learning" step to create a shared h_i^{sh} and a specific h_i^{sp} embedding representation of multimodal patient data. As described in (Figure 3.9), a classification loss will be applied on the "specific" to learn "modal-aware representation learning" parameters. After, the "adaptive graph learning" step is proposed where a population graph is built with node features defined as the concatenation between specific and shared feature representation $h_i = \text{Concat}\left(h_i^{sh}, h_i^{sp}\right)$. To compute the edge eight, the authors propose to project nodes features in a lower embedding space to compute a similarity metric as follows [104]:

$$A_{ij} = \text{Sim}\left(h_i, h_j\right) = \cos\left(W_A h_i, \quad W_A h_j\right) \tag{3.12}$$

Where W_A is a learnable projection matrix. Here the authors apply a manual threshold to cut edges and re-scale the remaining edge weight between zero and one. For the convolutional layer, the authors use the classical GCN layer described in the previous article. To learn the "adaptive graph learning" parameters, the authors proposed to use properties-based constraints to impose graph topological property (smoothness, sparsity, connectivity) as explained in (2.3.5). Then, the loss function proposed by the authors combines a classification loss and a properties-based loss function. The total property loss is defined as follows [104]:

$$\begin{aligned}
\mathcal{L}_{smh}(A, H) &= \frac{1}{2N^2} \sum_{i,j=1}{}_{ij} \left\|h_i - h_j\right\|_2^2 \\
\mathcal{L}_{\text{con}}\left(A\right) &= -\frac{1}{N} 1^\top \log(A \cdot 1) \\
\mathcal{L}_g(A, H) &= \mathcal{L}_{smh}(A, H) + \beta \mathcal{L}_{\text{con}}\left(A\right) + \frac{\gamma}{N^2} \|A\|_F^2
\end{aligned} \tag{3.13}$$

Then, the total loss function is defined as follows [104] :

$$\mathcal{L} = \mathcal{L}_{GNN}\left(Y, \hat{Y}_{GNN}\right) + \lambda \mathcal{L}_g(A, H) + \eta \mathcal{L}_{sp}\left(Y, \hat{Y}_{sp}\right) \tag{3.14}$$

Where $\mathcal{L}_{GNN}$ and $\mathcal{L}_{sp}$ are both a cross-entropy loss function. $\hat{Y}_{sp}$ and $\hat{Y}_{GNN}$ are the predictions (output of the MLP classifier) made at the end of the "modal-aware representation learning" and "adaptive graph learning" part respectively. β, γ, λ, and η are hyperparameters to weight the impact on each loss term.

Chapter 4

Methodology

4.1 Introduction

We have seen how existing works in the literature have formulated the problem of extracting information at the population level. The typical steps to build a model consist in creating a patient feature vector based on EHR data, imaging data, or both. Then it is necessary to represent a graph by choosing the representation of nodes and edges. Next, we have to ask ourselves what we want to learn, what properties we want for the graph, and how to aggregate the information from the other patients (convolutional part). The purpose of this chapter will be to address the problem of our graph creation.

In this section, five approaches to graph creation are proposed and constitute the contribution of this thesis. Here, we will focus on the representation of population graphs. We will make a hypothesis for each approach and establish a methodology based on it. In order to visually illustrate each approach, an architecture will be presented at the end of each method. Each problem will be defined theoretically and validated, and experimented in the chapter "Experiments". Most of the mathematical concepts used to define these approaches have been introduced in the "Background" chapter.

For the sake of lightness, not all terms of the equations will be redefined. However, let's define the starting point we must assume at this stage to understand the rest. Let's assume that we have a dataset containing medical information of several patients, such as image data and EHR clinic data. We will define C_i as a one-hot encoding of each characteristic C_i^c of a patient i. C_i^c can represent the gender, age group, the presence of a disease, the presence of a symptom, the taking of medication, and laboratory results. This characteristic is, therefore, binary and, in the case of a continuous variable, let us assume that we have divided it into several groups to obtain a category. Moreover, let us assume that we have an embedding h_i^0 representing the feature vector of a patient i obtained by extracting image and/or non-imaging data :

$$h_i^0 = \text{encode}_\theta(h_i^{img}, h_i^{EHR}) \tag{4.1}$$

With encode_θ being the proposed pre-trained model at the patient level. It is also important to note that the extraction of information at these two levels was done separately for memory, regularization, and performance reasons. Therefore, each piece of information predicting the patient's level is fixed and the corresponding model is used as an inference. Thus, there will be no backpropagation from the population model to the patient model. In the chapter "Experiments," we will detail the methodology and the data set used to represent the extraction of information at the patient level.

4.2 Methodology

This section will aim to pose the problem of information extraction at the population level by defining our graph representation learning. More formally, let's define a population graph $\mathcal{G} = (V, E, A)$ where the vertices $V = \{v_i\}_{i=1}^{n}$ represent a set of patients that are linked together by a set of edges $E = \{e_{ij} = (v_i, v_j) | \forall e_{ij} \in \mathcal{G}\}$ where the connectivity are represented in the adjacency matrix $A \in \mathbb{R}^{n \times n}$. For each patient in the graph, we have the corresponding vertex information $v_i = \{C_i, h_i^0\}$. In addition, let us define a prognosis as a clinical event of a patient (e.g. mortality or ICU admission). We thus seek to estimate the prognosis of a patient $\hat{y}_i$ from its representation in a population graph. Thus, we propose to use graphs to enrich the information of the patient from the interactions with the information of other patients present in the graph. Let us define $h_i^L \in \mathbb{R}^{d_h}$ as the d_h-dimensional patient feature vector enriched by the GCN-based model :

$$h_i^L = \text{GCN}_\theta((v_i = \{C_i, h_i^0\}, \mathcal{G}) | v_i \in \mathcal{G}) \tag{4.2}$$

Based on this enriched representation of the patient, we seek to estimate a prognostic $\hat{y}_i$. In other words, we try to classify each vertex of the graph, presenting our problem as a node classification problem. To perform the prognosis, a MLP classifier is used as follows:

$$\hat{y}_i = \text{MLP}(h_i^L) \tag{4.3}$$

Where MLP is the MLP introduced in Section 2.2.1 followed by a softmax function. The prognosis-based prediction $\hat{y} \in [0, 1]$ is represented by the probability of having a prognosis. To learn the model parameters, we represent the error as follows:

$$\mathcal{L}_{class} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \tag{4.4}$$

Where $y \in \{0, 1\}$ is the true prognosis, and $\mathcal{L}_{class}$ is the Binary Cross Entropy loss function that increases as the predicted probability diverges from the true label. Based on this loss function, we can backpropagate the error in the network to update the model parameters. The proposed pipeline is illustrated in Figure 4.1

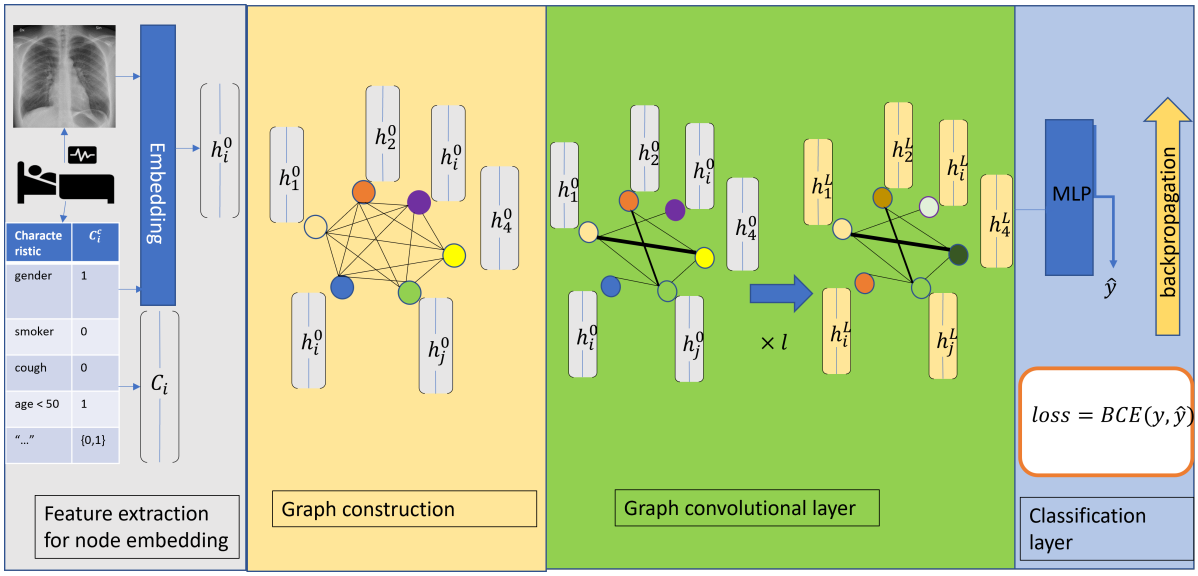

Figure 4.1: Illustration of the proposed model pipeline

In summary, as illustrated in Figure 4.1, we have the feature extraction part (in grey), which will be detailed in the chapter "Experiments" and we have just described the classification part (in blue). The following sections will focus on modeling patient interactions and the graph convolution to aggregate information based on these interactions. (in yellow and green).

4.3 Manual graph construction based on similarity metric

In this approach, we will use an approach consisting in building a graph manually based on a similarity metric between the patients.

4.3.1 Hypothesis

Similar to [95][96][97][98][99][101], we hypothesize that patients with medical similarities are more likely to have the same outcome. In addition, we hypothesize that the ability of a GNN to aggregate neighboring information is more powerful if the connections are representative for the prediction [6].

4.3.2 Methodology

Let's start with a simple model defining a pairwise similarity metric between patients to construct our population graph. The idea is, therefore, to concentrate the learning energy around the aggregation itself. Indeed, by handcrafting the graph, we manually impose its representation and topology. The parameters to be optimized will be those of the graph convolution and those of the MLP classifier. Based on these hypotheses, we will test several methods :

- An isotropic approach consisting in defining a graph sparsification method based on the similarity between nodes. In this approach, the similarity value is just used to determine the presence of a connection and is not used in convolution (i.e., edge weight is either 0 or 1). Each node will aggregate the information of the similar neighboring nodes with the same impact.
- An anisotropic approach to implicitly include anisotropy in our representation by assigning the similarity value in the edges to weigh the impact of the aggregated information from each neighboring node without adding extra model parameters.
- A hybrid approach consisting in sparsifying the graph while keeping its anisotropic representation to better detect similarity fluctuations.

Graph convolution

Since we are building a graph based on the representation of a population, we wish to define a graph convolution method with the high inductive capacity to learn a representation able to generalize to unseen graphs. We have already seen in Section 2.3.4 that it is possible to define an aggregation function able to update each node of the graph while being independent of the number of connections. For these reasons, we use GraphSage[69] as a graph convolution. Let's rewrite the version of GraphSage equation[71] used in this approach :

$$\hat{h}_i^{\ell+1} = \sigma\left(U^\ell \text{ Concat } \left(h_i^\ell, \text{Mean}_{j\in\mathcal{N}_i}\left(e_{ij} \odot h_j^\ell\right)\right)\right) \tag{4.5}$$

Here, the choice of the mean aggregator has been made empirically in Section 5.4. We have already defined h_i^0 and now we have to define the term e_{ij}.

Pairwise similarity metric

Let's pose the problem of the similarity representation. Based on our hypothesis, the problem is represented as follows:

$$e_{ij} = \sum_c \delta(C_i^c - C_i^c) sim(h_i^0, h_j^0) \tag{4.6}$$

Where $sim()$ is a pairwise similarity metric. The Kronecker delta δ is used in the sum to count the number of common characteristics between two patients. Thus, in this representation, we weigh the number of common features between patients by the similarity between their feature vectors. Based on our hypothesis, the criterion for obtaining a measure of patient similarity is $e_{ij} \to 0$ for patients with a different outcome and $e_{ij} \to 1$ for patients with the same outcome[1]. For the similarity metric, we will propose several functions like the Cosine similarity [79], the Euclidean distance, and several variants [105].

[1] In the case of a distance metric, we want the opposite. The objective of representation is the same.

Edge sampling

For approaches using edge sampling, we will try to define a manual edge threshold to keep only relevant connections. The threshold value for these two methods will be determined thanks to a hyperparameter search as described in Section 5.4.

4.3.3 Architecture

The global architecture of the proposed method is illustrated in Figure 4.2.

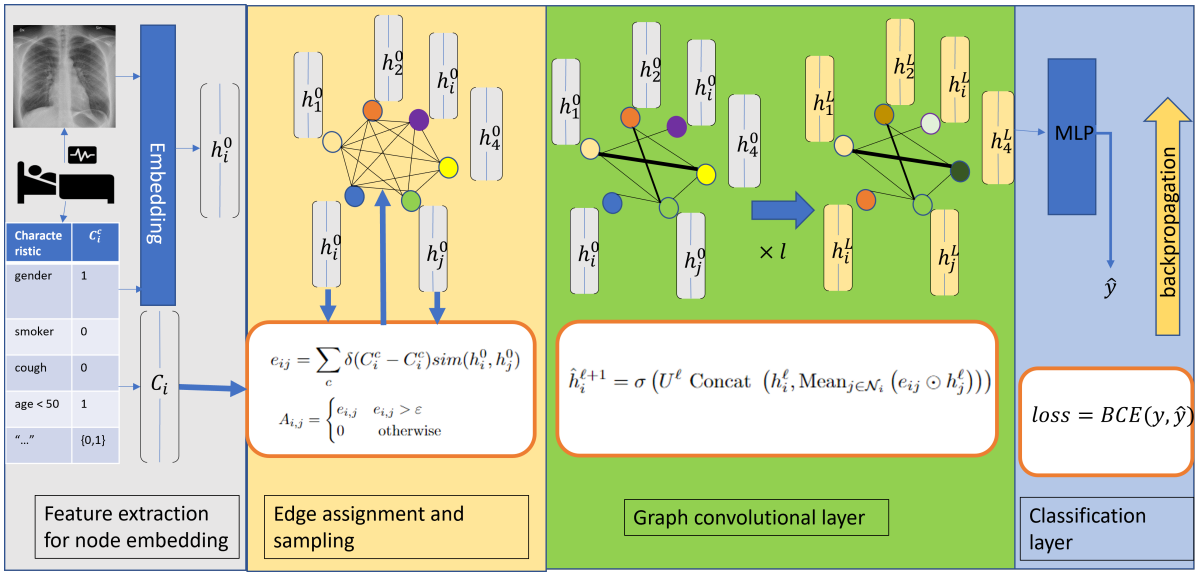

Figure 4.2: Illustration of the proposed method. After the feature extraction (in grey), a fully connected graph is created. From the extracted features and the patient characteristics, the edge weights are calculated. Based on their value, a threshold is defined to remove some connections. In the convolutional part (in green), we have a graph with connectivity represented by the thickness of the links. This graph is used for convolution and will enrich the feature vector of each patient until the last layer. Finally, in the classification part (in blue), a MLP classifier is used to perform the prognosis of each patient in the graph from their feature vector. Based on this prediction, the loss will be calculated allowing to backpropagate the error in the model.

4.4 Contrastive-based similarity learning to reduce graph topology uncertainty

In this section, we will define our problem by trying to force the model to learn its structure.

Hypothesis

Previously, we manually imposed the structure of the graph by defining a similarity metric. However, it is easy to imagine that this approach is highly dependent on the design of the metric as well as the sparsification parameters. Moreover, even if the previous approach could work for a specific dataset, it is each time necessary to redefine an equation for the edges and its parameters which is very time-consuming. In this context, we propose to extend the previous approach by including the learning of the graph structure. Based on our previous hypothesis, we wish to obtain a good contrast between good and bad edge connections, i.e. having a large edge value between patients with the same outcome and a small edge value for patients with a different outcome. Our hypothesis is thus that learning the graph structure by imposing a contrast on the edge representation can reduce the uncertainty of the graph topology [104].

4.4.1 Methodology

To do that, we propose to find an embedding space where the similarity metric is maximal/minimal for positive/negative connections. After reviewing the literature [106][107][108][109][110][111], we propose to use contrastive learning. More formally, let's write the precedent formulation to define edge weight as follows:

$$\hat{e_{ij}} = \text{sim}(W_a h_i^0, W_a h_j^0) \tag{4.7}$$

Where W_a is the projection matrix to embed the node feature in a shared embedding space, i.e. the projection matrix is the same for the two pairs. The objective of our approach will be to optimize W_a to push the values of $\hat{e_{ij}}$ to the extremities according to their label. For this purpose, we define three similarity/distance measures on the basis of which we will build a contrastive loss [106][107][108]. The first one is based on the Cosine similarity [79] and is written as follows:

$$\mathcal{L}_{\text{cont}}(x, y) = \begin{cases} 1 - \cos(x_i, x_j), & \text{if } y = 1 \\ \max(0, \cos(x_i, x_j) - m), & \text{if } y = 0 \end{cases} \tag{4.8}$$

Where x_1 and x_2 are the embedded patient feature vectors $W_a h_i^0$ and $W_a h_j^0$ and y is the true similarity value between patients i and j. Then, we want to push the cosine similarity between similar patients to 1 and to -1 for dissimilar patients. The margin m is used to choose how much we want to push negative pairs [107][106]. This loss function is based on the Hinge loss [112] which is widely used in classification tasks.

Instead of defining a similarity metric, it is possible to define a distance metric. This time, we want to minimize the distance between similar patients and maximize it for different patients. Hence, we can rewrite our constraint as follows:

$$\mathcal{L}_{\text{cont}}(x, y) = \begin{cases} d(x_i, x_j), & \text{if } y = 0 \\ \max(0, m - d(x_1, x_2)), & \text{if } y = 1 \end{cases} \tag{4.9}$$

Where d is a distance function. We propose to use two distance functions, the Euclidean, and the hyperbolic distance [113] which is defined as follows :

$$d(x_i, x_j) = \text{arcosh}\left(1 + 2\frac{\|x_i - x_j\|^2}{(1 - \|x_i\|^2)(1 - \|x_j\|^2)}\right) \tag{4.10}$$

The use of hyperbolic distance is motivated by the literature [102][114][115] where the authors show that it could be easier to contrast the features in a hyperbolic space [114][115].

For the graph convolution, we keep the GraphSage [69] version in Eq(4.5). Additionally, we try TopK as a sparsification technique to be sure that every node keeps connections. The network will be trained with the following loss function :

$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{cont}} + \mathcal{L}_{\text{class}} \tag{4.11}$$

Where $\mathcal{L}_{cont}$ is the contrastive loss used to train the model parameters W_a associated with the graph structure and $\mathcal{L}_{class}$ is the classification loss (Binary Cross Entropy loss in this case) used to train all model parameters of the graph structure, graph convolution, and MLP classifier.

4.4.2 Architecture

The proposed methodology is illustrated in Figure (4.3).

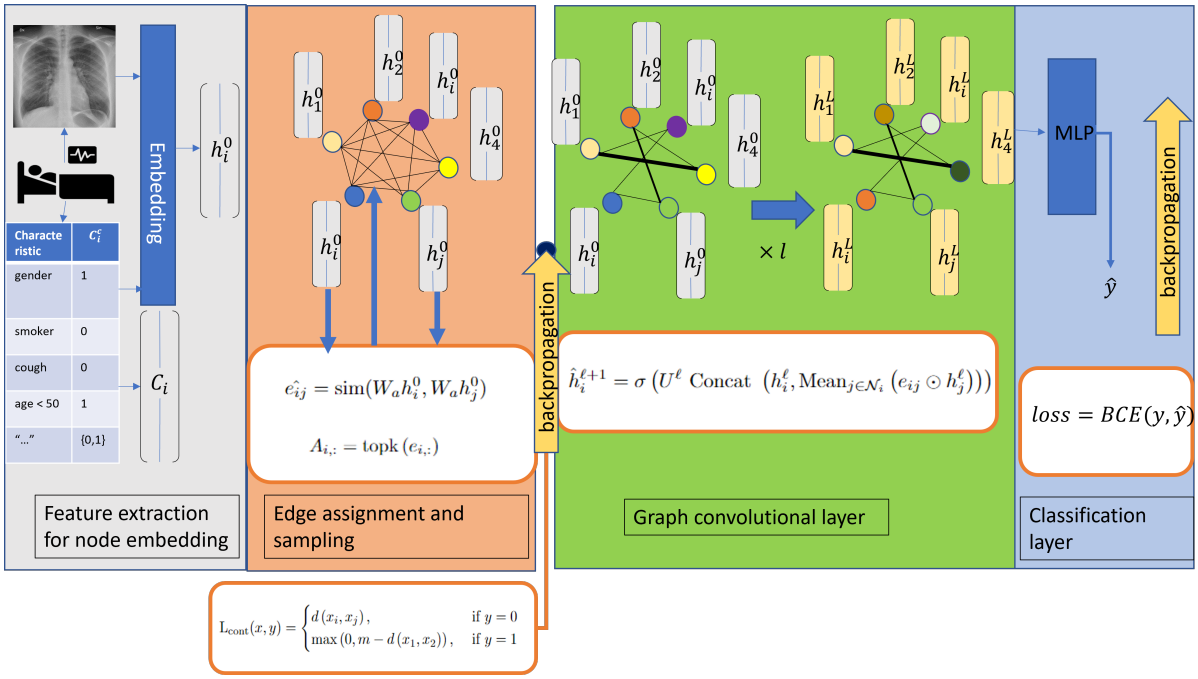

Figure 4.3: Illustration of the proposed method. After the feature extraction (in grey), a fully connected graph is created. The extracted patient feature vectors are projected in a lower dimensional space to compute the edge weight. Based on the edge value, every node will keep a fixed number of connections and delete the others. Based on the edge label, a contrastive loss is computed to backpropagate the error from the end of the graph creation part (in orange). In the convolutional part (in green), we have a graph with connectivity represented by the thickness of the links. This graph is used for convolution and will enrich the feature vector of each patient until the last layer. Finally, in the classification part (in blue), a MLP classifier is used to perform the prognosis of each patient in the graph from their feature vector. Based on this prediction, the loss will be calculated allowing to backpropagate the error in the model.

4.5 Attention-based dynamic graph topology learning

In this section, we propose several approaches to learn the graph structure with attention-based representation.

4.5.1 Hypothesis

So far we have posed our problem by defining a manual and learnable graph structure. This structure is fixed before the convolutional part and remains fixed until the end. However, the definition of the edge weights is made based on the information of the nodes. On this basis, our hypothesis is that a graph structure that changes in conjunction with the change of the node features could be more representative to aggregate neighboring information. We are therefore looking for a way to dynamically change the graph topology in the convolutional layer.

4.5.2 Methodology

As introduced in Section 2.3.4, we know that it is possible to introduce dynamic anisotropy in the graph convolution with the multi-head self-attention mechanism to learn the node interaction. Moreover, we have seen that it is possible to represent the edges in larger dimensions. In this context, a graph-based attention method is proposed. The idea is to start with a fully connected graph and progressively modify its topology according to the node features. We will thus try to use the versions of the graph attention network introduced in Section 2.3.4. In doing so, the aggregation of the information of patient j from patient i will be weighted by the attention coefficient indicating how much the node i should be attentive

to the node j with respect to the other connection [77].

Additionally, we propose to use an edge embedding representation. Then, we wish to represent the edge connectivity as a vector instead of a scalar. For this, we propose to define an edge vectorial representation f_{ij} as follows :

$$f_{ij} = [\delta(C_i^0 - C_j^0), \delta(C_i^1 - C_j^1), ..., \delta(C_i^n - C_j^n)]^T \tag{4.12}$$

f_{ij} is therefore an n-dimensional vector containing binary values indicating the common characteristics between two patients. So, compared to the previous methods where we summed the common characteristics, here we represent them individually in a vector. Based on the previous assumptions, we want to make the features of the nodes interact with the vectorial edge representation as follows 2.3.4:

$$f'_{ij} = \text{LeakyReLU}\left(A\left[h_i \,\|f_{ij}\|\, h_j\right]\right) \tag{4.13}$$

where A is a learnable projection matrix. Here we concatenate edge features f_{ij} with associated node features h_i and h_j and we project the resulting vector in a lower dimension followed by a non-linear activation function. Once we have achieved our vector representation of the interaction between two nodes, we can then project it to obtain a connectivity value as follows:

$$e_{ij} = F\left(f'_{ij}\right) \tag{4.14}$$

Where F is a linear learnable projection matrix. Here, we increase the order of the representation because we can learn the impact of pairwise patient characteristics with corresponding patient feature vectors to represent the similarity.

4.5.3 Architecture

The proposed methodology is illustrated in Figure (4.4).

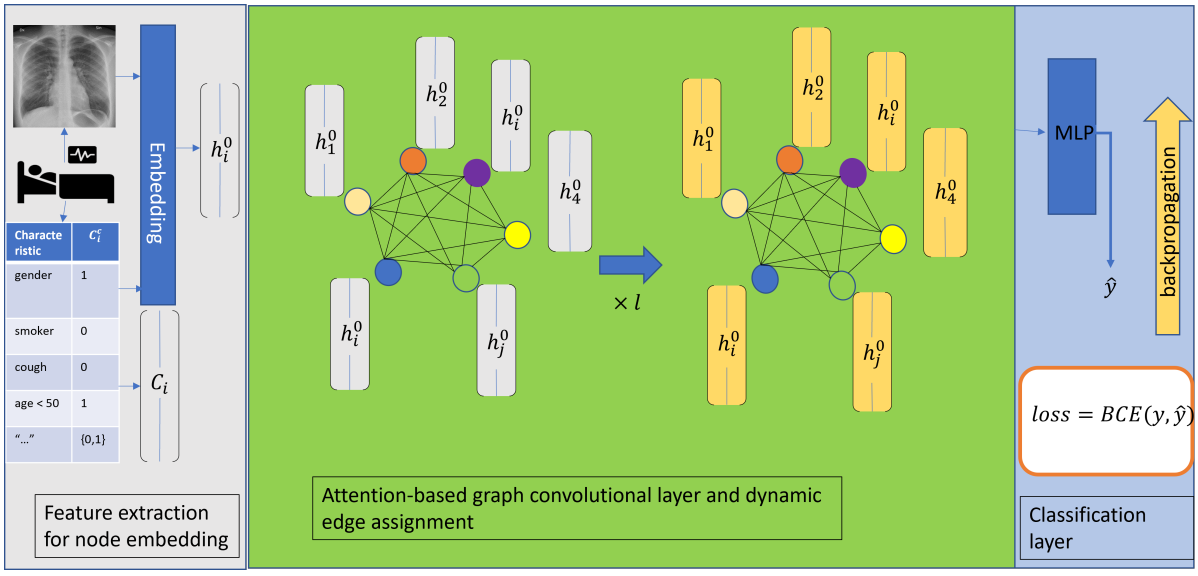

Figure 4.4: Illustration of the proposed method. After the feature extraction (in grey), a fully connected graph is created. In the convolutional part (in green), the weights of the edge are computed from pairwise node features or from vectorial edge jointly with pairwise node features depending on the attention-based GCN model used. The edges and nodes features are computed at each layer and the graph topology changes dynamically until the last layer. Finally, in the classification part (in blue), a MLP classifier is used to perform the prognosis of each patient in the graph from their feature vector. Based on this prediction, the loss will be calculated allowing to backpropagate the error in the model.

4.6 Properties-based regularisation of graph structure

In this section, an alternative approach to constrain the model to learn graph topology is proposed.

4.6.1 Hypothesis

So far, we have posed our problem around the representation of aggregation with the definition of a similarity metric in several forms. Here, we want to focus on the topology of the graph as a whole, i.e. to focus on the adjacency matrix. Our hypothesis is that it is possible to enrich the graph representation by constraining its structure.

4.6.2 Methodology

To do this, we want to get good properties for our graph representation. The idea is to constrain its structure based on property constraints. Similarly to [81][103], we propose to use smoothness, sparsity, and connectivity constraints. Before that, we will represent the similarity as follows:

$$e_{ij} = \text{ReLU}(cos(W_a h_i^0, W_a h_j^0)) \tag{4.15}$$

This definition was chosen empirically and is explained in Section 5.7. Indeed, an adjacency matrix is often symmetric and non-negative [81]. These two properties are provided by the definition in Eq(4.15). Based on the literature [81][103], we know that it is easier for a deep learning model to learn and generalize better based on smooth features. For this, as introduced in Section 2.3.5, we propose to use a graph Laplacian regularizer defined as follows:

$$\mathcal{L}_{\text{smooth}}\,(A, x) = \text{tr}(X^T \left(I - D^{-1/2} A D^{-1/2}\right) X) \tag{4.16}$$

Where $X^T = \left[h_0^0, h_1^0, \ldots, h_N^0\right] \in \mathbb{R}^{d_h \times N}$ and $\left(I - D^{-1/2} A D^{-1/2}\right)$ is the graph Laplacian introduced in Eq(2.27), and $A \in \mathbb{R}^{N \times N}$ is the graph adjacency matrix. Unlike the method based on contrastive learning, here we wish to impose a smooth structure to allow better detection of the fluctuation between the interactions of the nodes. Moreover, we wish to obtain a sparse structure. For this, we will add a sparsity constraint defined as follows:

$$\mathcal{L}_{\text{sparsity}}\,(A) = \|A\|_1 \tag{4.17}$$

where $\|.\|_1$ is the $L1-norm$ and will be used to push adjacency matrix elements to zero. In order to avoid the trivial solution $A = 0$, we add a connectivity constraint to regularize the number of connections. Then, we define the connectivity constraint as follows:

$$\mathcal{L}_{con}(A) = -\frac{1}{N} 1^\top \log(A \cdot 1) \tag{4.18}$$

with $1 \in \mathbb{R}^{N \times 1}$ being a unity vector. The log is used to manage asymptotically the connectivity.

In summary, we propose to pose our graph representation as a whole by learning an embedding space where it is possible to define a similarity allowing us to dynamically change the graph structure under several property constraints. More formally, we wish to optimize the parameters W_a as follows:

$$\hat{A} = \underset{W_a}{\arg\min} \lambda_0 \,\text{tr}(X^T (I - D^{-1/2} A D^{-1/2}) X) + \lambda_1 \|A\|_1 - \lambda_2 \frac{1}{N} 1^\top \log(A \cdot 1) \tag{4.19}$$

And the Graph representation learning loss function :

$$\mathcal{L}_{GRL}(A, x) = \lambda_0 \mathcal{L}_{smooth}(A, x) + \lambda_1 \mathcal{L}_{sparsity}(A) + \lambda_2 \mathcal{L}_{con}(A) \tag{4.20}$$

Where λ_i are hyperparameters to weigh the impact of each constraint. Finally, we propose to use anisotropic GraphSage in Eq(4.5) as a graph convolution and learn these parameters under the classification constraint from the BCE loss. Thus, the total loss function will be defined as follows:

$$\mathcal{L}_{\text{tot}} \;= \lambda_3 \mathcal{L}_{GRL} + \mathcal{L}_{class} \tag{4.21}$$

Where λ_3 is a hyperparameter. The objective will be mainly to find the right regularization through the optimization of these hyperparameters.

4.6.3 Architecture

The proposed methodology is illustrated in Figure (4.5).

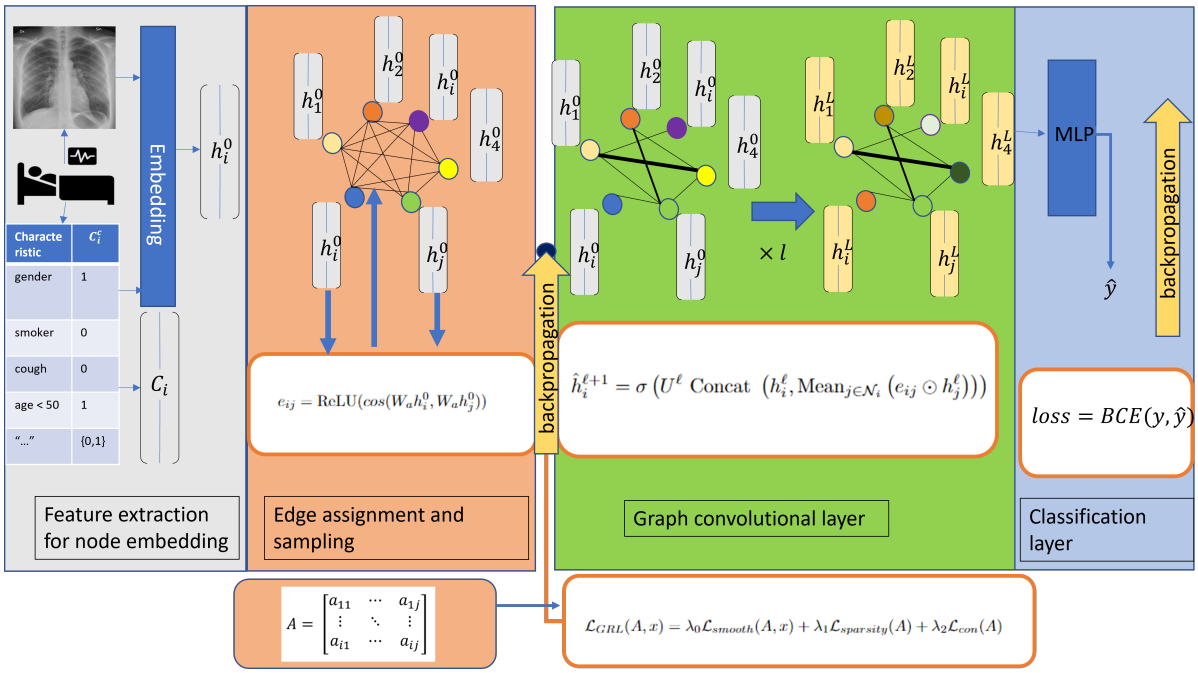

Figure 4.5: Illustration of the proposed method. After the feature extraction (in grey), a fully connected graph is created. The extracted patient feature vectors are projected in a lower dimensional space to compute the edge weight. The edges connectivity with values lower than zero are deleted. Based on the node features and the graph adjacency matrix, the proposed intermediate loss is computed to backpropagate the error from the end of the graph creation part (in orange). In the convolutional part (in green), a graph with connectivity is represented by the thickness of the links. This graph is used for convolution and will enrich the feature vector of each patient until the last layer. Finally, in the classification part (in blue), a MLP classifier is used to perform the prognosis of each patient in the graph from their feature vector. Based on this prediction, the loss will be calculated allowing to backpropagate the error in the model.

4.7 Hypergraph-based model to learn a higher level of interaction

In this section, we try to extend our graph representation in the hypergraph domain.

4.7.1 Hypothesis

So far, we have represented the interaction between patients with a pairwise representation. Here, our hypothesis is that patient interactions are more complex than a simple pairwise similarity. We, therefore, wish to extend the expressiveness of the interactions to a higher level of representation [90].

4.7.2 Methodology

It is in this context that we introduce hypergraphs. As presented in section 2.4, hypergraph interactions are represented by hyperedges. Hyperedges have the ability to connect several vertices at once. To extend the problem posed in Section 4.1, let's define $\mathcal{G} = (V, E, H)$ the population hypergraph where V is a set of vertices representing the patients in the same way as Section 4.1. The problem is still a vertex classification problem. Here we propose to represent the hyperedges $e_k \in E$ by the incidence matrix

$H \in \mathbb{R}^{n \times m}$ where h_{ik} correspond to the presence of a connection between vertex i and hyperedge e_k. Then, in the context of our problem, we define the element of H as follows :

$$h_{ik} = C_i^k \tag{4.22}$$

Where C_i^k is the one-hot encoding EHR characteristics k of patient i. Thus, in this representation, we define each hyperedge as a single characteristic connecting all patients having this characteristic. For the hypergraph convolution, we will use the version introduced in section 2.4 as follows [90][89]:

$$X^{\ell+1} = \sigma\left(D^{-1}HWB^{-1}H^T X^\ell P\right) \tag{4.23}$$

Where $X^\ell \in \mathbb{R}^{n \times d_h^\ell}$ contains the n vertex features h_i^l at layer ℓ with hidden dimension d_h^ℓ. Based on this, we propose to test an isotropic version of hyperedges by assigning $W = I$, and further, we will try to learn this matrix by introducing learnable parameters $\theta \in \mathbb{R}^m$ as follows :

$$W = \text{diag}(\theta) \tag{4.24}$$

4.7.3 Architecture

The proposed methodology is illustrated in Figure (4.6).

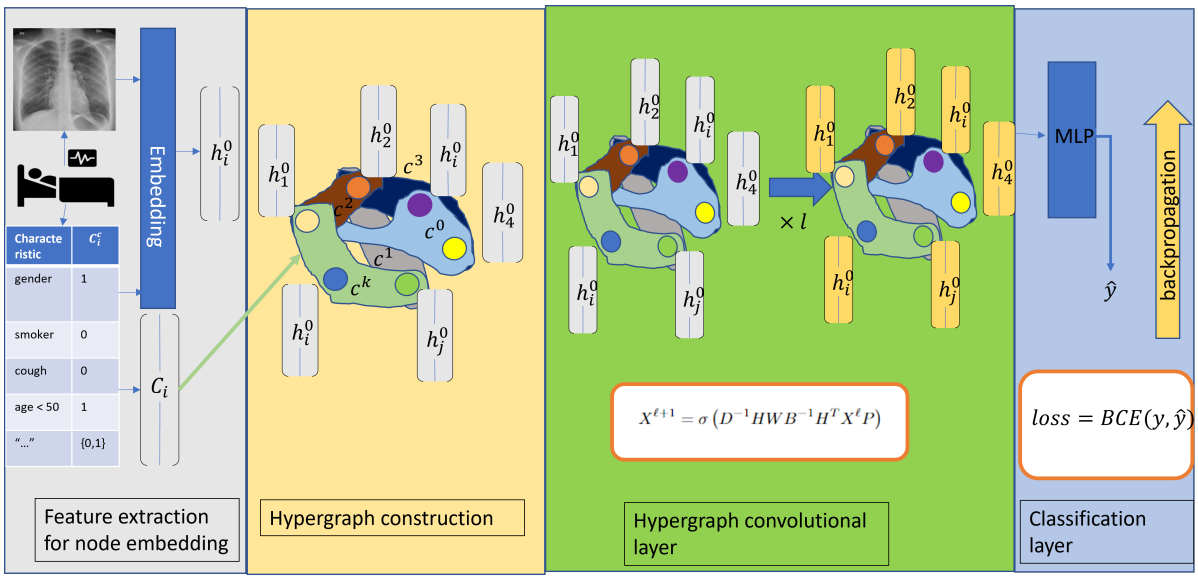

Figure 4.6: Illustration of the proposed method. The first step (in grey) extracts patient feature vectors. In the hypergraph construction part (in orange), the hypergraph is created by representing each vertex as the encoded patient feature vector. The one-hot encoded characteristics are used to construct the hyperedges. Hence, each hyperedge connects the patient/vertex that has the corresponding characteristics. In the convolutional part (in green), the hypergraph is used for convolution and will enrich the feature vector of each patient until the last layer. Finally, in the classification part (in blue), a MLP classifier is used to perform each patient's prognosis in the graph from their feature vector. Based on this prediction, the loss will be calculated, allowing to backpropagate the error in the model.

Chapter 5

Experiments

5.1 Introduction

In the previous chapter, we posed our problem around graph representation learning in a theoretical way. This chapter will detail the experimental method used to implement the proposed approaches. To do so, we will start by defining our experimental set-up. In other words, we will start by defining and describing the data in this thesis. Then, the experimental pipeline will be detailed to understand how the models were created, trained, and optimized. Once this is done, we will describe the methodology used to preprocess the data as well as the methodology used to extract the information at the patient level. At this point, we will have determined all the aspects necessary to build the graph-based models. We will follow the same structure as in the methodology part, detailing the different experiments performed to analyze the proposed models' performance and validate the hypotheses.

5.2 Experimental set-up

5.2.1 Dataset

Stony Brook University COVID-19 Positive Cases (COVID-19-NY-SBU)

We call SBU dataset [116] the dataset acquired at Stony Brook University during the COVID-19 pandemic. This dataset contains multimodal clinical data of 1384 patients who tested positive for COVID-19. It contains imaging data with different modalities and organ sites (chest radiographs, chest CT-scan, brain MRI, etc..) [116]. An example of chest radiography is illustrated in Figure 5.1.

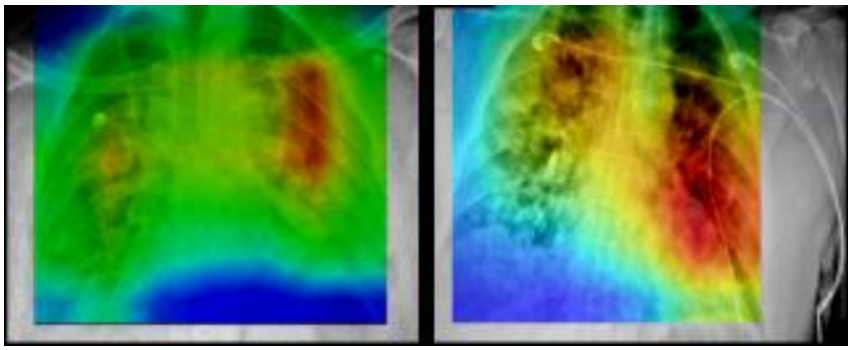

Figure 5.1: Example of chest radiography with an automated region of prognosis importance detection [116]. Picture taken from [116]

As non-imaging data, this dataset contains the EHR data with 130 different variables such as :

- Demographic information (age, gender, ..)
- Patient characteristics (BMI, smoking status,
- Symptoms at hospital admission (cough, nausea, vomiting, ..)
- Medical history (lung disease, kidney tumor, ..)
- Pre-admission medication (antibiotics, anti-inflammatory,..)
- Laboratory result (protein in the urine, blood oxygen, ..)
- Patient measures (temperature, respiration rate, heart rate, ...)
- Different outcomes (mortality, ICU admission, hospital discharge, ..)

A complete list of the variables, their distribution, and their meaning is present in Appendix A.2. This EHR data can be continuous or categorical. Among the 1384 patients available in the dataset, we will use only 1344 for reasons that will be explained in Section 5.3.1.

STOIC2021 - COVID-19 AI Challenge

We call the STOIC dataset [117] the dataset containing the CT scans from 10,735 patients acquired during the COVID-19 pandemic. This dataset has been published in the context of a deep learning challenge[1] where the objective was to predict the positiveness of COVID-19 RT-PCR test and the COVID-19 severity[2] in one month following the positive RT-PCR test. To do that, the goal is to use the lung CT scan of the patient as well as clinical data (age and gender). In this work, we will use the 2000 patient samples that are publicly available for the qualification phase of the challenge. A complete list of the dataset distribution is present in Appendix A.1

[1] `https://stoic2021.grand-challenge.org/stoic2021/`

[2] severity defined as either need for intubation at one point or death

5.2.2 Experimental pipeline

This section will explain the experimental methodology used to train and compare different models.

Graphs creation

Graphs creation			
Dataset	#graphs	#nodes/graph	#edges/graph
STOIC	5	400	1600
SBU	20	67/68	4489/4624

Table 5.1: Description of graphs creation for the dataset

For both SBU and STOIC datasets, we represent a population-based graph where each node represents a patient and each edge represents the similarity between two patients. For that, we create a graph by defining an embedding from patient data to represent node features. The method to generate this embedding will be described in Section 5.3 for the two datasets. To construct the edges, we always begin with a fully connected graph[3] and different techniques will be done in the training pipeline to assign different edge representations. As shown in Table 5.1, we split the population into 5 and 20 graphs for STOIC and SBU datasets respectively. This choice was made empirically to fit the memory during the training. The splitting with stratified sampling[4] was done to ensure that every graph has the same number of positive and negative patients.

Splitting

To evaluate each model correctly, a k-cross validation [118](with k=4) will be made. To do that, we keep out 20% of the dataset and use it as a test set to evaluate the model performance. With the 80% remaining graphs, we split them into four folds. Then, for each model training, we take three folds (60%) to train the model, one fold (20%) to evaluate the model, and one constant fold (20%) to make the final performance test. This process is repeated by changing the folds for validation and training until every fold is used for training and validation. Thus, for each model, we train it four times with different training and validation graphs while the test set is always the same. An example of this procedure is illustrated in Figure 5.2. This method is used to detect overfitting and get model performance from a statistical point of view. Hence, we can see in Table 5.2 the number of graphs and patients used for training, validation, and testing.

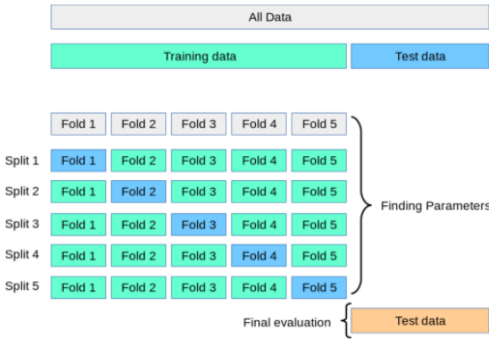

Figure 5.2: Illustration of K-cross fold validation procedure. Picture taken from `https://vitalflux.com/k-fold-cross-validation-python-example/`

Splitting strategy						
Dataset	**Training**		**Validation**		**Test**	
	#graphs	#patients	#graphs	#patients	#graphs	#patients
STOIC	3	1200	1	400	1	400
SBU	12	806	4	269	4	269

Table 5.2: Description of the splitting strategy.

Training

For each model, Adam optimizer [17] with learning rate decay strategy is used. We start with an initial learning rate and reduce it by a factor of two if there is no improvement in the validation loss during a certain number of epochs. We set 3 stopping criteria which are :

[3]Every node is connected with itself and with all the other nodes in the graph with edge weight equal to one

[4]`https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html`

- The minimum learning rate is reached.
- The evaluation metric does not increase during 30 epochs.
- The training time exceeds 12 hours.

Performance measure

We use AUC as an evaluation metric. For each model training, we keep the model's epoch for which the best AUC score is get according to the validation set. Then the model is saved for this epoch and used for inference with the test set[5]. This inference is done 4 times according to the cross-fold validation strategy and the score for the test set is the mean (+/- standard deviation) of this score for the four cross-folds training.

Hyperparameters search

To select the best combination of hyperparameters, we make a hyperparameters search. To do that, we use a Bayesian hyperparameter tuning[6] thanks to "Weights & Biases" library [119]. To perform Bayesian hyperparameter tuning, we first choose a finite set or a probability distribution of hyperparameters we want to optimize. After, we launch the algorithm [119] performing the following steps :

- Randomly choose a step of hyperparameters.
- Perform the training with the cross-fold validation strategy.
- Based-on average validation AUC, choose a new set of hyperparameters.
- Repeat the process until the best combination is obtained according to the average validation AUC.

In the following sections, the score used will be the average AUC of the test set obtained after a search for hyperparameters optimizing the AUC of the evaluation set. Each model will have hyperparameters specific to their network architecture. However, the hyperparameters specific to the training are displayed in Table 5.3.

Hyperparameter	**Mode**	**Search space**	**Description**
batch_size	finite	[1, 2, 3, 4]	The number of graphs used before updating model parameters
init_lr	finite	[1e-2, 5e-3, 1e-3, 5e-4, 1e-4]	Initial learning rate
lr_schedule_patience	finite	[3, 5, 7, 11, 15, 19, 23]	The number of epoch to divide learning rate by two
weight_decay	finite	[0.2, 0.1, 5e-2, 1e-2, 5e-3, 1 e-3, 5e-4, 1e-4, 0]	Impact of the L1-regularisation term in the loss function
dropout	uniform	min:0 ; max:0.95	Dropout applied on model parameters
in_feat_dropout	uniform	min:0 ; max:0.95	Dropout applied on input data

Table 5.3: Example

As described in Table 5.3, there are multiple modes to optimize the hyperparameters. It is possible to define predefined values to test and a probability distribution representing the search space. To choose these modes, we rely on the literature and on the nature of each variable to optimize the searches. For example, the dropout can be a continuous value we seek to define in a uniform space while the batch size is fixed, and a maximum value must be defined to fit the memory.

[5]It is also for this reason that we use a test set in addition to the validation set because the evaluation set is used to choose the best model and for the learning rate decay strategy.

[6]`https://wandb.ai/site/articles/bayesian-hyperparameter-optimization-a-primer`

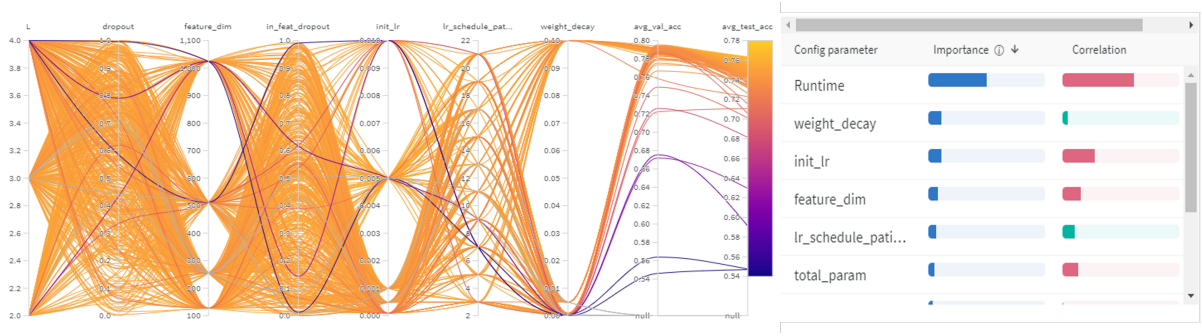

Figure 5.3: Illustration of Wandb framework for hyperparameters search.

Figure 5.3 shows a typical example of the framework used to follow the hyperparameters searches. We can observe all these filaments representing all the combinations of hyperparameters used to train the model. We can also see the results obtained for each model. Moreover, we have access to a panel describing each hyperparameter's importance and correlation with respect to the average score. This feature gives us more sensitivity to the behavior of the network and the impact of each hyperparameter. In addition to these panels, the training curves are also available to analyze the score's evolution and the loss function as a function of the epochs for training, validation, and test set. This functionality lets us detect overfitting and obtain observations to better regularize the model.

Software set-up

The code was made using *Python*[7] programming language on *Anaconda*[8] environment and *Pycharm*[9] as developing tool. Additionally, all the training information is saved on *WandB*[10] framework. All the graphs are created with *DGL*[11] library, and the training is made thanks to *Pytorch*[12]. The data preprocessing for images as made thanks to *MONAI*[13] while, for EHR data, the preprocessing has been done thanks to *Sklearn*[14].

Hardware set-up

The resources and services used in this work were provided by :

- The VSC (Flemish Supercomputer Center), founded by the Research Foundation - Flanders (FWO) and the Flemish Government: 2x Nvidia Tesla P100 16GB with 2x 12-core INTEL E5-2650v4 256GB.
- The server in ETRO (Engineering ICT & Electronics) department located at the Vrije Universiteit Brussel (VUB): NVIDIA TITAN Xp COLLECTORS EDITION 8GB.
- Personal PC : NVIDIA GeForce RTX 3050 GDDR6 4GB with 11th Generation Intel Core i7-11800H Up To 4.60 GHz 8GB.

5.3 Patient level

In this section, the problem of multimodal embedding is posed. As discussed in the previous section, the goal is to create a feature vector representing each node in the graph. So here, we address the extraction of information at the patient level. This means that data preprocessing has to be done and finding a way to merge the different modes. For this, two different approaches will be used for the two datasets.

[7] https://www.python.org/
[8] https://www.anaconda.com/
[9] https://www.jetbrains.com/fr-fr/pycharm/
[10] https://wandb.ai/site
[11] https://docs.dgl.ai/
[12] https://pytorch.org/
[13] https://docs.monai.io/en/stable/index.html
[14] https://scikit-learn.org/stable/

5.3.1 Preprocessing

SBU Dataset

Variable selection As an outcome, the mortality was chosen. The no-informative variables like "patient_id" were deleted, and variables with zeros variances like "covid19_statuses" were also deleted because the model cannot learn anything from them.

Data imputation To handle the missing data, a treatment has been done beforehand. As introduced in Section 5.2.1, there are two types of EHR data. The first is continuous numeric data, while the second is categorical data. To deal with these missing data, the following steps have been done :

- Analyse the missing data rate and delete data with more than 90% of missing rate
- For missing numerical variables, we replace them with the mean value of this variable across the dataset
- For categorical missing variables, we use *KNN Imputer*[15] to replace each variable by the most assigned in the five closest patients.

By doing this preprocessing, a homogeneous dataset is obtained. Some works in the literature used more complex solutions [120]. For categorical data, some authors used the missing value as a category itself [47].

Encoding To encode these variables, the following methods are done :

- For numerical variables, a *Standard Scaler*[16] was used to normalize each variable by subtracting the mean and dividing by the standard deviation.
- For Categorical variables, a *One Hot Encoder*[17] to represent each unique category as a binary variable indicating the presence of this value is used [36].

For categorical variables, the encoding also replaces the category represented by string values with the category defined by binary integer values and allows it to be independent of the variable parametrization [36].

Images preprocessing To fit the memory of future models, the images were resized. In the SBU dataset, multiple modalities and organs are available for each patient. The Chest radiography acquired as close as possible to the hospital admission is selected for each patient. As preprocessing, each image is resized[18] in size (512x512). Moreover, the patients for which Chest radiography is not available are removed from the dataset

STOIC Dataset

Here, there is no missing data for "age," "gender", and "RT-PCR test." The image preprocessing was done in the context of another project [121]. Please refer to this work to get the necessary information [121].

[15] `https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html`
[16] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`
[17] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html`
[18] `https://docs.monai.io/en/stable/transforms.html`

5.3.2 Embedding

This subsection will define the methodology adopted to represent the patient feature vector. As we saw in Section 3.1, some authors use node representation with all the information (imaging and non-imaging data) [103][102]. In contrast, others use only imaging data to represent the nodes and non-imaging data to build the edges [95][96][97]. In the following, we propose to pose two different representations motivated by the nature of the data.

SBU Dataset

Hypothesis Thanks to the analysis made by the authors using this dataset [122][123][124][125][126], we know that the EHR data are the most powerful to predict the mortality. The imaging data are also expressive to get satisfactory predictive results. For this reason, we create an embedding that fuses both imaging and non-imaging data to develop a patient features vector.

Methodology To create a powerful patient feature vector, we have to find an embedding space where the different modalities can be fused to capture their interaction (correlation, complementarity) [103]. There are three levels where we want to extract information :

- The "Modality level" aims at extracting information from each modality individually.
- The "Cross-modality level" aims at extracting interaction between the different modalities.
- The "Population level" aims at extracting information from the other patients.

Here the problem associated with "modality level" and "cross-modality level" will be posed, while the problem of "Population level" is already posed in Section 4.1.

"Modality level" For this level, we want to extract information from each modality to get a powerful representation of each of them. Then we need to find an imaging-based model as well as a tabular-based model. By analyzing the state-of-the-art method, the authors often used a CNN-based approach and an MLP-based system to embed imaging and non-imaging data respectively [95][96][122][123][126]. Here we propose to use a transformers-based approach to encode the different modalities. We propose the following architectures for each modality, as illustrated in Figure 5.4:

- A Vit transformer encoder [49][41] for imaging data. As discussed in Section 2.2.3, the Vit transformer can process images by dividing them into fixed patches and using them as tokens in the transformer encoder. This model achieves state-of-the-art results in most image classification benchmarks. The proposed model uses positional encoding [127] and represents the encoded feature vector as the mean of the encoded patches.
- A TabTransformer encoder [47] for categorical EHR data. By creating a column embedding for each unique categorical variable, the TabTransformer is one of the state-of-the-art models to process tabular data [48].
- For numerical EHR data, the authors of [47] propose to make a concatenation with the categorical feature at the output of the TabTransformer before passing it to a MLP classifier. The proposed approach follows a similar methodology to represent the numerical feature vector by normalizing it.

For the imaging and categorical data, the feature vector will be the vector at the output of the corresponding transformer encoder[49][47] (the input of the MLP classifier in the original architecture[49][47]) as illustrated in Figure 5.4.

"Cross-modality level" For this level, we start with an imaging feature vector from the Vit transformer, a categorical feature vector from the TabTransformer, and a vector containing the normalized numerical EHR data. The objective is to find a way to fuse these different modality feature vectors to capture their interactions. As introduced in Section 2.2.3, Transformers-based models [41] have shown their capabilities to capture data interaction thanks to the multi-head self-attention mechanism [41][43]. For this reason, the proposed approach to combine different modality feature vectors is based on transformer encoder [49][41].

Architecture The full multimodal embedding model for the SBU dataset is illustrated in Figure 5.4

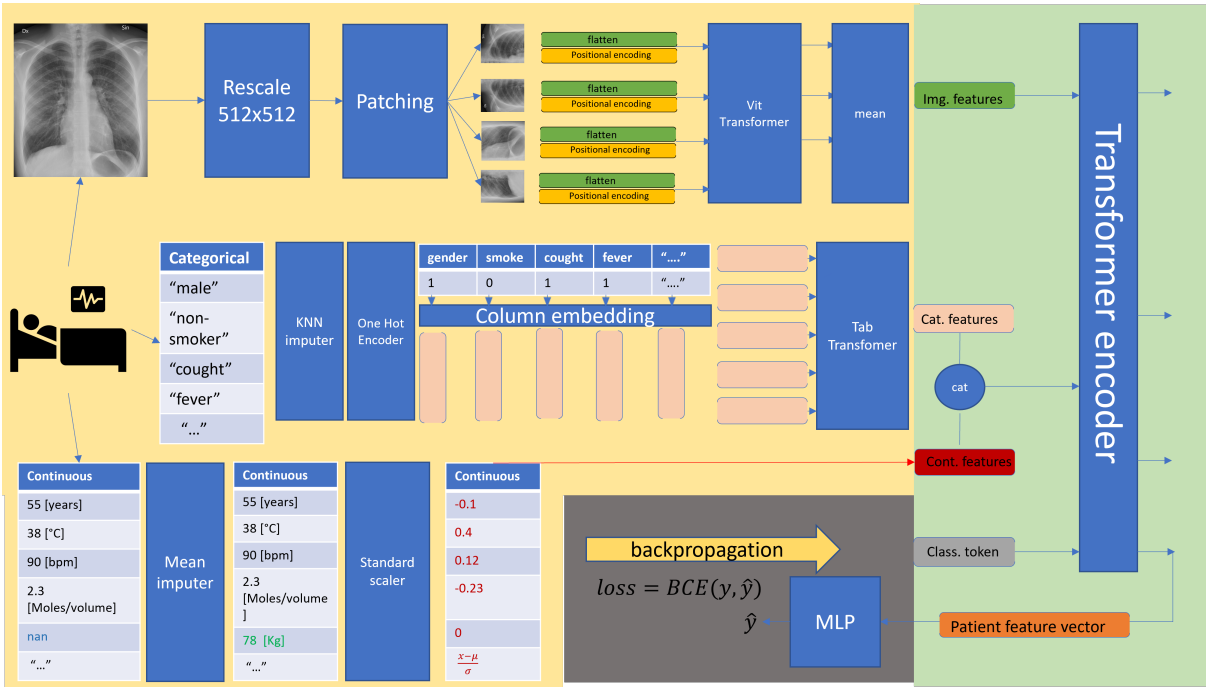

Figure 5.4: Illustration of proposed multimodal patient embedding model. In yellow, is the "Modality level" part. In green, is the "Cross-modality" part. In gray, is the training part.

Figure 5.4 describes a complete overview of the proposed model. Specifically, it consists of the following steps :

- **TabTransformer [47]** is used to process categorical data. The categorical data are imputed, and one-hot encoded. At this step, we have 89 unique binary categories, and each of these values will be embedded in an 8-dimensional vector (column embedding). Each of these embedded columns is used as an input token for the transformer encoder. Hence, we have 89 8-dimensional token passing in 8 layers transformer encoder with 12 multi-head of size 64 and feed-forward of dimension 1024, and no dropout. The final categorical feature vector is obtained by concatenating each output token giving a row vector of size 712.

- **Standard numerical encoder** is used. The 39 numerical variables are imputed and scaled by substrating the mean and dividing the result by the standard deviation, giving the final continuous feature vector of size 39.

- **Vit Transformer [49]** is used to process chest radiography. The image is first rescaled to size 512x512. After, the image is divided into 64 patches with fixed size 64x64. Each patch is flattened and a positional encoding is summed to allow the transformer to be sensitive to position [127]. Each of these vectors is embedded in a lower euclidean space of dimension 751[19] to represent the input token of the transformer. Then, Theses 64 tokens are passed in 8 layers of transformer encoder with 12 multi-heads of size 64 and feed-forward of dimension 1024 and no dropout. The final imaging feature vector is obtained by taking the mean of the output tokens giving a 751-dimensional feature vector.

- **Transformer encoder [41]** is used to fuse the feature vectors of each modality. To do that, a classification token is created. This token is nothing else than a 751-dimensional vector containing trainable model parameters. The categorical and continuous feature vectors are concatenated, giving a 751-dimensional feature vector. Thus, we have three tokens of dimension 751, one from

[19]This dimension is chosen to match the dimension resulting from the concatenation of the continuous and categorical feature vector

imaging data, one from EHR data, and one from the classification token. These three tokens are used as input for the transformer encoder. The cross-modality encoder consists of 8 layers of transformer encoder with 12 multi-heads of size 64 with the feed-forward dimension 1024 and dropout = 0.1. The final patient feature vector is the encoded classification token giving a 751-dimensional patient feature vector (hi^0).

- **MLP classifier** is used to classify the patient outcome based on the encoded patient feature vector. To do that, three layers MLP with the layer sizes (751x350,350x175, 175x2) and Gelu activation function followed by a softmax function is used. Then, the network parameters are trained by backpropagation based on a BCE loss function applied to the outcome prediction and the true patient outcome.

Ablation study To evaluate our model, we propose to study the performance of ViT and TabTransformer individually and in combination. To do this, we first trained the model as a whole. Then, we used it for inference by training a MLP classifier with for input, the feature vector encoded by the corresponding transformer. For TabTransformer, we concatenate the continuous data with the categorical data encoded by the TabTransformer. The results are shown in Table 5.4.

Model	**Test (mean ± std)**	**Val (mean ± std)**	**Input of MLP**	**Data**
ViT[49]	73.45 ± 0.97	74.56 ± 2.03	Imaging feature vector from Vit	SBU
TabTransformer[47]	91.29 ± 0.656	91.64 ± 1.39	Concat(Numerical, TabTransformer output)	SBU
Total model	93.27 ± 0.387	93.60 ± 1.55	Output of the fusion transformer	SBU

Table 5.4: Results obtained for ablation study. As expected, EHR data are more useful for prediction compared to imaging data. However, the ViT help to improve the overall performance of the model.

STOIC Dataset

Hypothesis The challenge associated with this dataset is recent, and no article was using this data at the time of implementation. However, due to the small amount of EHR data, it is implicit that the prediction of COVID severity must come mainly from the image data. Moreover, the challenge is first to predict if the patient is positive for COVID-19 and if so, to predict if the covid will be severe. Therefore, the RT-PCR test information is an intermediate label to the final prediction.

Methodology As the primary source of information comes from the image data, the proposed approach uses image embedding to represent the nodes of the graph and to use the age and gender to construct the edges afterward. This methodology is similar with the methodology adopted in [95][96][98][104]. To do this, the proposed approach uses ConvNext [128] to create the imaging patient feature vector. ConvNext is a hybrid method combining the idea of Vit transformer [49] and incorporating CNN-based techniques. To represent the imaging patient feature vector, we train ConvNext to extract a 1024-dimensional feature vector followed by a MLP classifier that predicts the result of the RT-PCR test. Then, we use the result of the RT-PCR test as the label to train the ConvNext parameters. The severity will be used to train the GCN part in the network.

Architecture The architecture based on ConvNext [128] to extract the patient feature vector has been done in another project. All specifications relating to training and optimization are presented in [121].

Results In addition to the image-only classification, we also tried to concatenate age and gender to the imaging feature vector and train an MLP classifier. The results obtained for this model are shown in Table 5.5.

Model	**Test (mean ± std)**	**Val (mean ± std)**	**Input of MLP**	**Data**
ConvNext + MLP	74.98 ± 0.35	78.58 ± 2.38	Imaging feature vector from ConvNext	STOIC
ConvNext +MLP Cat	74.87 ± 1.41	78.17 ± 2.75	Concat(imaging feature, age, gender)	STOIC

Table 5.5: Results obtained for the proposed method.

5.4 Population Level

In this section, we will detail the experiments that have been done to validate the approaches proposed in the "Methodology" chapter. For that, we will try to validate our hypothesis and find the best versions of our model. So for each model, we will experiment with a search for hyperparameters allowing us to obtain the model with the best performance from a statistical point of view. In addition to optimizing the hyperparameters related to the training as illustrated in Table 5.3, we will also optimize the hyperparameters related to the network architecture. These hyperparameters are detailed in Table 5.6.

Hyperparameter	**Mode**	**Search space**	**Description**
n_layers	discrete	[1, 2, 3, 4, 6, 8]	Number of gnn layers
hidden_dim	discrete	[128, 256, 512, 1024, 2048]	Hidden node feature dimension
residual	discrete	[True, False]	Use of residual connection
batch_norm	discrete	[True, False]	Use of batch normalisation

Table 5.6: Description of network hyperparameters search.

Table 5.3 and Table 5.6 gather the hyperparameters common to all models. Then, depending on the proposed method, we will add some components in our search for hyperparameters.

5.4.1 Manual graph construction based on similarity metric

As introduced in Section 4.3, we proposed three approaches: an isotropic approach with sampling, an anisotropic approach and an anisotropic approach with edge sampling.

Experiments

In order to choose the best approach, we first set the similarity metric and the sampling method. For this, we decided to use the Cosine similarity on the basis of which we defined a threshold from which we assign the edge weight to zero. This approach was done only on the STOIC dataset. For the STOIC dataset, we want to introduce age and gender in our representation. However, as introduced in Section 5.2.1, imaging data is the main source of information. It is therefore necessary to define a representation including both pieces of information. On this basis, let's define the similarity representation introduced in Eq(4.6) as follows:

$$e_{ij} = (\delta(g_i - g_j) + \delta(|a_i - a_j| < \theta))sim(h_i^0, h_j^0) \tag{5.1}$$

Where $\delta(g_i - g_j)$ equals one if gender g_i of patient i is equal to the gender g_j of patient j. $\delta(|a_i - a_j| < \theta)$ equals to one if the age difference $|a_i - a_j|$ between patient i and j is lower than a threshold θ (empirically $\theta = 5$). So, here, we sum the patients similar in age and gender and we will weight this result with a similarity metric based on the imaging features representing the positivity to the RT-PCR test. For practical reasons, we will normalize the edge weight values as follows:

$$e_{ij} = \frac{e_{ij} + 2}{4} \tag{5.2}$$

With this, we get values between zero and one. Then, we try to optimize the hyperparameters in Table 5.7 to determine the best approach.

Hyperparametres	**Mode**	**Search space**	**Description**
edge_weight	discrete	[True, False]	Use of edge weight in convolution (i.e. anisotropy)
aggregator	discrete	[mean, pool, lstm]	GraphSage aggregation function
threshold	uniform	min:0 ; max:1	Threshold value applied to sample edges
k_hop	discrete	[1, 2, 3]	K-hop neighbors used for aggregation
smpl_option	discrete	['lower', 'higher']	Delete edge with value under or higher the threshold

Table 5.7: Description of hyperparameters search

Based on 530 combinations of hyperparameters, we find that the best approach is the hybrid approach where we use the edge weights in the convolution and a threshold that removes the edge weights lower than its value. Moreover, the "Mean" aggregation function is the most efficient for GraphSage with the one-hop neighbors. Having selected our approach, we now want to try to improve it by testing several similarity metrics. For this approach, we will optimize the different similarity metrics for both SBU and STOIC datasets. For the SBU dataset, the representation of the edges is as follows:

$$e_{ij} = \sum_c \delta(C_i^c - C_i^c) sim(h_i^0, h_j^0) \tag{5.3}$$

Where C_i^c represents the categorical EHR data and h_i^0 represents the multimodal embedding from the patient level. The edge values have been normalized to range between zero and one.

We try to find the best similarity metric for these two datasets. For this, several metrics have been selected in the literature as shown in Table 5.8.

Similarity	**Equation**	**Description**
Cosine similarity[79]	$\frac{\langle x_i \cdot y_i \rangle}{\|x_i\| \cdot \|y_i\|}$	Measure of the projection between two vectors
Euclidean distance	$\sqrt{\sum_i (x_i - y_j)^2}$	Measure the Euclidean distance between two points
ISC [105]	$\frac{\sum_{i=1} \sqrt{x_i y_i}}{\sqrt{(\sum_{i=1} x_i)} \sqrt{(\sum_{i=1} y_i)}}$	Variant of Cosine
SoftmaxKL [129]	$\sum_i P(x_i) \log \frac{P(x_i)}{P(y_i)}$	Measure the similarity between probability distribution
SqrtCos [105]	$\frac{\sum_{i=1} \sqrt{x_i y_i}}{(\sum_{i=1} x_i)(\sum_{i=1} y_i)}$	Variant of Cosine

Table 5.8: Illustration of the different similarity metrics used in the edge representation.

Here, we consider the similarity metric as a hyperparameter. To find the best similarity and the best-associated threshold, we reduce the search space by assuming the previous representation (Khop=1, Anisotropic GraphSage with mean aggregator, batch normalization, and residual connection).

Results

For both datasets, we obtain the best performance with the Cosine similarity. The results for both datasets are shown in Table 5.9.

Model	**Test (mean ± std)**	**Val (mean ± std)**	**Specifications**	**Data**
sim+thresh	76.32 ± 0.35	77.63 ± 1.34	Cos + thresh = 0.224	STOIC
sim+thresh	95.11 ± 1.62	94.11 ± 1.25	Cos + thresh = 0.2391	SBU

Table 5.9: Results obtained for the proposed method.

Hypothesis validation

Our hypothesis was that GraphSage is able to better aggregate neighboring information if adjacent connections come from patients with the same outcome. To validate this hypothesis, the following experiment is proposed:

- We define labels for edges. The label of an edge is equal to one if the edge connects two patients with the same outcome and zeros otherwise.
- Based on these labels, we will train the model by imposing a ratio of good and bad connections from each node ("positive rate").
- We repeat the experiment for several positive rates and record the performance of the model.

By doing so, we want to prove that the performance of the model increases as a function of the good connection ratio. The experiment validated our hypothesis as illustrated in Figure 5.5

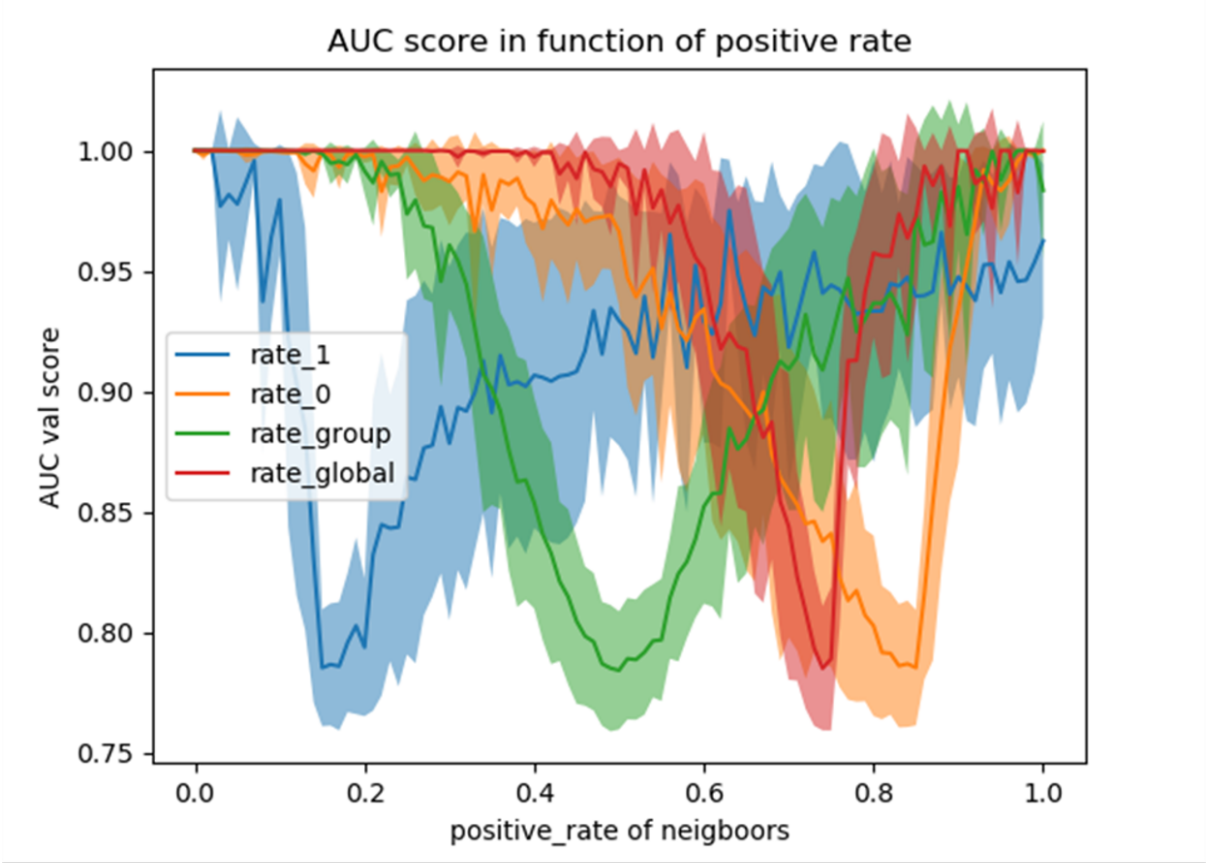

Figure 5.5: Illustration of the impact of the ratio of positive node in-connections on the model performance. The continuous line represents the average score and the continuous area around the lines represents the standard deviation across all folds. In red, we randomly impose a global proportion of good connections in the whole graph. We start from a fully connected graph (i.e. the bottleneck on the figure relative to the class proportion) and we randomly remove connections to reach the desired ratio (i.e. x-axis). In green, we impose a ratio on each node of the graph. In this case, each node (independent of its label) has the same proportion of good and bad connections. In blue and orange, we impose a ratio only on the positive and negative nodes respectively.

5.5 Contrastive-based similarity learning to reduce graph topology uncertainty

As introduced in Section 4.4, the proposed approach aims at contrasting the embedding of edges.

5.5.1 Experiments

To do this a contrastive loss was used. Several definitions have been made based on the Cosine similarity, the Euclidean distance, and the Hyperbolic distance. Moreover, as indicated in Eq(4.7), we project our features in a common lower euclidean space to compute the similarity. In addition to the training and network hyperparameters, we try to optimize the hyperparameters specific to our problem. These hyperparameters are presented in Table 5.10.

Hyperparametres	**Mode**	**Search space**	**Description**
contrastive_sim	discrete	[euclidean, hyperbolic, cosine]	Contrastive-based similarity
sim_dim	discrete	[64, 128, 256, 512, 1024]	Embedding dimension to compute the distance

Table 5.10: Description of hyperparameters search

Finally, the method giving the best results is the one using the hyperbolic distance with an embedding space of dimensions 128 and 256 for SBU and STOIC datasets.

5.5.2 Results

The best results after the hyperparameter search are displayed in Table 5.11.

Model	Test (mean ± std)	Val (mean ± std)	Specifications	Data
contrastive	76.19 ± 0.44	78.87 ± 1.12	Hyperbolic; dim = 256	STOIC
contrastive	93.89 ± 0.78	94.45 ± 1.94	Hyperbolic; dim = 128	SBU

Table 5.11: Results obtained for the proposed method.

Hypothesis validation

In order to validate our hypothesis, we propose to represent our patient feature vector before projection (h_i^0) and after projection ($W_a h_i^0$) in a lower Euclidian space where we compute the distance metric to represent the edges. To illustrate this high-dimensional feature vector, we use t-distributed Stochastic Neighbor Embedding (t-SNE) [130]. In other words, we project our high-dimensional vector in 2-dimensional space as shown in Figure 5.6.

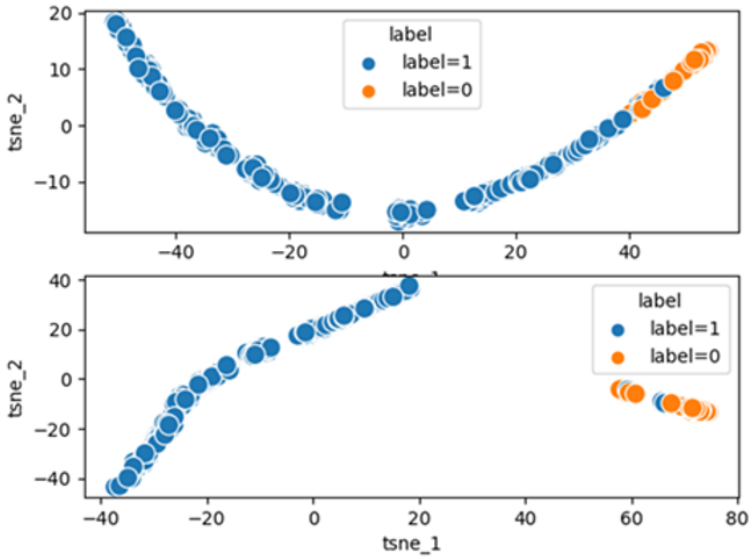

Figure 5.6: t-SNE plot of the patient feature vector for SBU dataset. On top, the representation of the initial feature vector at the input of the GCN (h_i^0). Bottom, the representation of the same feature vector obtained after projection ($W_a h_i^0$) for the distance calculation. Here we can observe that the distance between positive and negative patients has increased due to the hyperbolic contrastive loss. However, the patients mixed with the wrong group remain in the wrong group. So we increase the distance decreasing the uncertainty on the value of the edges but we do not change the classification of the patients who are already misrepresented.

5.6 Attention-based dynamic graph topology learning

As introduced in Section 4.5, we have two different paradigms. The first one uses the multi-head self-attention mechanism to change the graph's structure as the node feature changes. The second one is to try a vector representation of the edge features.

5.6.1 Experiments

There are several variations of the GAT. Here we adopt an empirical approach by considering the variants as hyperparameters. In addition to the hyperparameters related to the training and the network, we try to optimize the hyperparameters of Table 5.12.

Hyperparametres	**Mode**	**Search space**	**Description**
GCN	discrete	[GAT [73], GATv2[77], DotGat[74], AGNN[78]]	Attention-based graph convolution
n_head	discrete	[1, 2, 4, 6, 8, 12, 16]	Number of heads

Table 5.12: Description of hyperparameters search

Finally, we find the best result for DotGat with 4 heads for the SBU dataset. For time reasons, the hyperparameters search was not done for the STOIC dataset and we will use the hyperparameters of the optimized model for SBU to evaluate the performance on the STOIC dataset.

For the second method, we represent the edges with a vector. As indicated in Eq(4.12), the edge features are represented by a binary vector indicating the presence of common characteristics between two patients. This model has been optimized only for the SBU dataset where we use categorical data as characteristics. The model has not been used for stoic data because of time constraints and because the number of characteristics is very small. We used EGAT [80] with the embedding dimension for the edges equal to the hidden dimension of the nodes. The only extra hyperparameter to optimize is the number of heads. For this method, we find the best score with 8 heads. However, the score remains lower than the first model as shown in Table 5.13.

5.6.2 Results

Model	**Test (mean $\pm$ std)**	**Val (mean $\pm$ std)**	**Specifications**	**Data**
Attention	75.28 ± 0.58	78.57 ± 1.19	DotGAT, n_heads = 4	STOIC
Attention	94.40 ± 1.04	95.25 ± 0.83	DotGat, n_heads = 4	SBU
EGAT	94.12 ± 0.78	94.89 ± 0.76	EGAT, n_heads = 8	SBU

Table 5.13: Results obtained for the proposed method.

5.7 Properties-based regularisation of graph structure

As introduced in Section 4.6, we wish to optimize the structure of the graph as a whole. We have defined several property constraints.

5.7.1 Experiments

Experiments were first done manually to find the representation of the edges. Several formulations were proposed but did not give the desired behavior. Finally, Eq(4.15) allowed us to obtain a sparse structure (thanks to the ReLU function) and good connectivity (the edges associated with a self-connection were maximal). Based on this representation, we try to optimize the hyperparameters in Table 5.14

Hyperparametres	**Mode**	**Search space**	**Description**
sim_dim	discrete	[64, 128, 256, 512, 1024]	Embedding dimension to compute the similarity
lambda0	uniform	min:0; max:1	Smoothnes constrain impact
lambda1	uniform	min:0; max:1	Sparsity constraint impact
lambda2	uniform	min:0; max:1	Connectivity constraint impact
lambda3	uniform	min:0; max:2	Total propertie constraint impact

Table 5.14: Description of hyperparameters search

5.7.2 Results

After a search of 634 hyperparameter combinations for the SBU dataset, we get the results in Table 5.15. For time reasons, there was no search of hyperparameters for the STOIC dataset and the optimized model for SBU performs very badly on STOIC. It will be necessary to optimize the hyperparameters for STOIC because they are very sensitive.

Model	**Test (mean ± std)**	**Val (mean ± std)**	**Specifications**	**Data**
Properties	74.72 ± 0.71	77.93 ± 1.19	$\lambda_0 = 0.04$; $\lambda_1 = 0.69$; $\lambda_2 = 0.75$; $\lambda_3 = 0.95$; $sim\_dim = 256$	STOIC
Properties	94.22 ± 1.29	95.33 ± 1.10	$\lambda_0 = 0.04$; $\lambda_1 = 0.69$; $\lambda_2 = 0.75$; $\lambda_3 = 0.95$; $sim\_dim = 256$	SBU

Table 5.15: Results obtained for the proposed method.

5.8 Hypergraph-based model to learn a higher level of interaction

As introduced in Section 4.7, we seek to reach a higher level of representation by using hypergraphs.

5.8.1 Experiments

Here we propose two approaches, one with isotropic hyperedges and one with anisotropic hyperedges. In the first case, we will impose $W = I$ in Eq(4.23) and for the second, we will introduce a vector of parameters to try to learn the impact of the different connections. Thus, in addition to the hyperparameters related to the training and the network, we will test both versions of the model in the search for hyperparameters and for the two datasets. For the STOIC data, the hypergraph is constructed with 7 hyperedges representing the two gender values and 5 age groups created at regular percentiles. For the SBU data, the hypergraph is constructed with 89 hyperedges representing the 89 unique categorical EHR data.

5.8.2 Results

After the hyperparameters search, we obtain the results presented in Table 5.16. For the moment it is the isotropic version that works well and the anisotropic version, as it has been modeled, does not work [20].

Model	**Test (mean ± std)**	**Val (mean ± std)**	**Specifications**	**Data**
HyperGraph	76.29 ± 0.293	78.83 ± 2.568	Isotropic hyperedge	STOIC
HyperGraph	96.29 ± 0.656	95.64 ± 1.15	Isotropic hyperedge	SBU

Table 5.16: Results obtained for the proposed method.

5.9 Comparison and Discussion

In this section, we will analyze the results obtained in order to understand the models and answer the research question. First of all, it is necessary to take a step back on the performance of the models. The summary of the results obtained for each proposed method is presented in Table 5.17.

[20] The training curves are very irregular and the score ends up dropping to 50% for the training, validation, and testing set.

Model	Test (mean ± std)		# params	
	STOIC	SBU	STOIC	SBU
ConvNext	74.98 ± 0.35			
Multimodal Transformer		93.27 ± 0.387		31,447,686
Sim+thresh	76.32 ± 0.35	95.11 ± 1.62	3,718,210	1,613,003
Contrastive	76.19 ± 0.44	93.89 ± 0.78	1,847,398	1,635,364
Attention	75.28 ± 0.58	94.40 ± 1.04	1,956,979	1,733,284
Properties	74.72 ± 0.71	94.22 ± 1.29	1,847,398	1,557,301
HyperGraph	76.29 ± 0.293	96.29 ± 0.656	1,159,236	1,429,554

Table 5.17: Results obtained for the proposed methods.

Before going into the different aspects related to the graph, it is important to analyze the added value of adding a representation of the population by graph in the total pipeline. For this, we can make an analysis in terms of complexity and performance.

5.9.1 Complexity analysis

We observe that the multimodal embedding for the SBU dataset is very expensive in terms of parameters (i.e. 31 million parameters). This is an acceptable order of magnitude when we know that we are dealing with high-dimensional data. Moreover, even if the transformers are heavy in parameters, they are the state of the art in many problems and constitute a powerful baseline to generate the patient feature vector. Graph-based models are much less heavy in terms of parameters. The main reason is that the input data is very small in terms of dimension (i.e. a column vector). Moreover, the library used for graphs implements efficient matrix multiplication methods by reducing the complexity thanks to a sparse representation of the graph components (i.e. adjacency matrix) making the training really fast unlike transformers. In conclusion, from a complexity point of view, adding a graph-based model to the output of our baseline model does not increase so much the total model complexity much (about +5% of parameters).

5.9.2 Performance analysis

In terms of performance, we can observe that the graph methods tend to increase slightly the score obtained for the baseline model. However, from a statistical point of view, we can see that we have a large standard deviation that encompasses the baseline model score. Thus, for some of the proposed models (Contrastive, Attention, Properties), it is difficult to assert a real impacting value. However, the models based on fixed graph structures and the models based on hypergraphs tend to show a significant improvement in the result as illustrated in Table 5.17. These last two models, therefore, add a significant improvement in the result by adding about 5 percent more parameters to our baseline. In this context, it is likely that a population representation based on graphs/hypergraphs can extract additional information from the data.

5.9.3 Comparison with the state of the art

As discussed earlier, the STOIC dataset was used in the context of a deep learning challenge. The dataset is not yet fully public and it is difficult to compare our models with the literature. However, for the SBU dataset, there are several works [122][123][124][125][126]. By analyzing the performance of these works for mortality prediction, we can observe that the proposed models largely outperform most of them [122][123][124][126]. However, it is important to note that some authors do not have the same experimental setup as us. In this work, we try to include all available EHR variables in the dataset while some others select a smaller portion. In addition, we use chest radiography obtained at the admission of

a patient while the dataset proposes several modalities and organs at several time intervals [124]. Among all the works that we found in the literature, only one of them proposes higher performances than us [125]. However, as for the other works, it is difficult to compare because they do not use the same evaluation metrics and the same experimental set-up. Moreover, the authors of [125], do not propose a cross-fold validation and the result has no statistical component making the comparison even more difficult because the score depends greatly on the choice of patients used in the test set. In conclusion, it is difficult to compare our results in a rigorous way but it seems that the approach proposed in this thesis is a suitable competitor in the literature for the SBU dataset.

5.9.4 Experiments-based discussion

During the work of this thesis, it was very difficult to obtain significant results for the graph representation. It is important to note that the method of patient embedding was based on state of the art models that obtain very good performance. It was therefore very difficult to obtain significant results. A lot of effort has been made to pose the problem in several ways in order to enrich our learning representation.

We started by defining a simple model with a fixed graph structure based on a similarity metric where hyperparameters such as the definition of similarity had little impact on the result. So we needed a lot of testing and optimization to get our results. This model has been the most time-consuming in the development of this thesis since it was done in conjunction with the implementation of the experimental set-up and the understanding of the problem.

Based on the validation of the hypothesis of the previous model, we posed the problem differently by trying to learn the structure by contrastive learning. We obtained what we expected from a graph topology point of view, but this did not lead to an improvement in the results, on the contrary. So the method worked, but the problem was not well-posed.

Based on the philosophy of transformers and the attention mechanism, we tried several methods to impose a dynamic graph structure and a higher dimensional edge representation. These models were very complicated to regularize and the training was very complicated because the graph was fully connected which increased very strongly the complexity and thus the training time.

Once we had gone through all these methods, we had to completely revise the way we posed our problem and we decided to focus on the structure of the graph itself. Here, we applied several property constraints. These constraints are all very powerful and have a huge impact on the behavior of the graph. A lot of time has been spent to optimize the impact of the different constraints not giving significant added value.

Finally, the problem of pairwise interaction was questioned and motivated the use of hypergraphs. This method was introduced a few weeks before the end of this thesis and there is still a lot to do in this part. This is the first time that encouraging results have been obtained. The model has been optimized for both datasets but only the model with an isotropic representation of the hyperedges has proven to be successful while the anisotropic version has completely failed (probably due to a bad initialization of the parameters). Nevertheless, little time has been spent to solve these problems but a lot of hope is directed toward this representation.

In general, the simple models showed better performance capabilities. By making the representation more expensive, the results often decreased. It is therefore important to understand what we want to represent and what we want to learn in order to pose the problems properly. The decrease in performance relative to more complex representations may be due to problems of regularization and training rather than the representation itself.

Chapter 6

Conclusions and future works

We have been through a number of representations with different philosophies. However, there is still a lot of work to do in this area. Among the proposed models, only those with a predefined graph structure have shown significant results. It is therefore necessary to conduct more experiments to understand how to learn the graph structure. Sampling models based on probability distributions have not been explored and may be the subject of future work [131][86][6]. Moreover, even if the approaches have been presented differently, some of the models are equivalent to each other. It is therefore necessary to make the right assumptions to really find the key points of the model. The model based on hypergraph representation seems really promising and a lot of things can be done in this direction. The method based on anisotropic hypergraphs was not stabilized due to lack of time and there are many works proposing methods to learn the structure of hypergraphs[91][91][93]. Moreover, hypergraphs can be mathematically expressed in the graph domain. A future analysis could be to bring the proposed hypergraph model to its graph equivalent in the graph domain to understand its representation[132][133][90]. Moreover, even though the subject of this thesis was oriented around the extraction of information from a population-based representation, many works in the literature suggest different approaches to prognosticate a patient [134][135][136][137]. The flexibility of graphs allows for strong representation and the possibilities are numerous [138][139]. A future direction could be to use graphs at the patient level to represent the interaction between different modalities. Once again, it is important to ask the right questions and justify the use of graphs rather than other models to solve this type of problem. Other approaches like multigraphs and diffusion mechanisms are also interesting directions [140][141]. Also, from the beginning of this work, we have assumed to be in supervised learning. However, other types of learning can also solve certain problems [97][81]. Finally, we worked on static data whereas the patient prognosis could be more efficient if we included temporal data in order to capture the evolution of the patient's condition in time. In conclusion, there are a lot of opportunities for future works both at the level of representation and at the level of the problem to be solved. The objective is to continue to bring efficient solutions but also solutions allowing the implementation of a model capable of being used in the real world by combining different capabilities to obtain a generalized prognostic model [1].

In conclusion, we were able to show that it was possible to perform the prognosis of a patient based on medical data in several forms. More specifically, we have shown that it is possible to extract additional information from a graph-based representation of a patient population. The implementation of a rigorous experimental set-up has been proposed to obtain exploitable and reproducible results. This experimental set-up will be a good basis for future projects. During this thesis, a number of concepts related to graph theory have been studied, summarized and mastered. This knowledge will be very useful for future work as graphs are a very promising field for modeling and finding solutions to real-world problems. In addition to graphs, the understanding and implementation of recent state of the art models for patient embedding has been very instructive and has helped to consolidate the general skills in the machine learning field. Moreover, during this thesis, an internship was done in a hospital with similar problems treated in different ways. During this internship, data processing at the source was done as well as a large analysis to extract different risk factors. Moreover, two student jobs were done in the context of a deep learning challenge and cooperation between the university and the university hospital. These two experiences allowed to understand the research environment and to learn from very competent people.

All these experiences allowed to develop strong skills in the processing and analysis of medical data as well as the creativity necessary to elaborate future solutions. These experiences were also sometimes very hard because it was necessary to adapt to the world of research and to understand the expected objectives. A lot of work remains to be done in order to pose problems correctly, structure the research work and develop mathematical creativity. However, ambition and perseverance to develop solutions having a positive impact on the world will overcome all difficulties.

Acronyms

Adam Adaptive Moment Estimation. 5, 47

AI Artificial intelligence. 1

AUC Area Under the Curve. 6, 48

BCE Binary Cross Entropy. 4, 36, 39, 42, 53

CNN Convolutional Neural Network. 3, 9–11, 13, 14, 28, 29, 32, 51, 53

EHR Electronic Health Record. 29, 30, 33, 44, 46, 49–51, 53, 55, 60

FPR False Positive Rate. 6

GAT Graph Attention Network. 22, 58

GCN Graph Convolutional Network. 3, 19, 21–25, 31, 33

GCN Graph Neural Network. 1, 2, 14, 21, 27, 31, 53, 57

ICU Intensive Care Unit. 29, 30

MLP Multi-Layer Perceptron. 3, 7–12, 14, 25, 28, 29, 32, 33, 36–41, 43, 44, 51, 53

NLP Natural Language Processing. 11

OTUs Operational Taxonomic Units. 5

RNN Reccurent Neural Network. 9, 20, 29

SGD Stochastic Gradient Descent. 4

TPR True Positive Rate. 6

UZB Universitair Ziekenhuis Brussel. 2

ViT Vision Transformer. 13

Bibliography

[1] A. Panesar, *Machine learning and AI for healthcare*. Springer, 2019.

[2] J. N. Acosta, G. J. Falcone, P. Rajpurkar, and E. J. Topol, "Multimodal biomedical ai," *Nature Medicine*, vol. 28, no. 9, pp. 1773–1784, 2022.

[3] S. Shamshirband, M. Fathi, A. Dehzangi, A. T. Chronopoulos, and H. Alinejad-Rokny, "A review on deep learning approaches in healthcare systems: Taxonomies, challenges, and open issues," *Journal of Biomedical Informatics*, vol. 113, p. 103 627, 2021.

[4] A. Mathew, P. Amudha, and S. Sivakumari, "Deep learning techniques: An overview," in *Advanced Machine Learning Technologies and Applications*, A. E. Hassanien, R. Bhatnagar, and A. Darwish, Eds., Singapore: Springer Singapore, 2021, pp. 599–608, ISBN: 978-981-15-3383-9.

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

[6] L. Wu, P. Cui, J. Pei, and L. Zhao, *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Singapore, 2022, p. 725.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[8] Z.-H. Zhou, *Machine learning*. Springer Nature, 2021.

[9] L. Liu and H. Qi, "Learning effective binary descriptors via cross entropy," in *2017 IEEE winter conference on applications of computer vision (WACV)*, IEEE, 2017, pp. 1251–1258.

[10] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[11] D. Richards and M. Rabbat, "Learning with gradient descent and weakly convex losses," in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, A. Banerjee and K. Fukumizu, Eds., ser. Proceedings of Machine Learning Research, vol. 130, PMLR, 13–15 Apr 2021, pp. 1990–1998. [Online]. Available: `https://proceedings.mlr.press/v130/richards21a.html`.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, ISSN: 1476-4687. DOI: `10.1038/nature14539`. [Online]. Available: `https://doi.org/10.1038/nature14539`.

[13] S. Ruder, *An overview of gradient descent optimization algorithms*, 2016. DOI: `10.48550/ARXIV.1609.04747`. [Online]. Available: `https://arxiv.org/abs/1609.04747`.

[14] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[15] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan, "Better mini-batch algorithms via accelerated gradient methods," *Advances in neural information processing systems*, vol. 24, 2011.

[16] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[17] A. Tato and R. Nkambou, "Improving adam optimizer," 2018.

[18] R. Rojas, "The backpropagation algorithm," in *Neural networks*, Springer, 1996, pp. 149–182.

[19] B. Moons, D. Bankman, and M. Verhelst, "Embedded deep learning," *Embedded Deep Learning*, 2019.

[20] G. Dunn and B. S. Everitt, *An introduction to mathematical taxonomy*. Courier Corporation, 2004.

[21] S. A. Hicks, I. Strümke, V. Thambawita, *et al.*, "On evaluation metrics for medical applications of artificial intelligence," *medRxiv*, 2021. DOI: 10.1101/2021.04.07.21254975. eprint: https://www.medrxiv.org/content/early/2021/04/09/2021.04.07.21254975.full.pdf. [Online]. Available: https://www.medrxiv.org/content/early/2021/04/09/2021.04.07.21254975.

[22] S. Wu and P. Flach, "A scored auc metric for classifier evaluation and selection," in *Second workshop on ROC analysis in ML, bonn, Germany*, 2005.

[23] M. H. Ferris, M. McLaughlin, S. Grieggs, *et al.*, "Using roc curves and auc to evaluate performance of no-reference image fusion metrics," in *2015 National Aerospace and Electronics Conference (NAECON)*, IEEE, 2015, pp. 27–34.

[24] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[25] J. Mairal and B. Yu, "Complexity analysis of the lasso regularization path," *arXiv preprint arXiv:1205.0079*, 2012.

[26] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *international conference on machine learning*, PMLR, 2017, pp. 2847–2854.

[27] T. Hastie, "Ridge regularization: An essential concept in data science," *Technometrics*, vol. 62, no. 4, pp. 426–433, 2020.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[29] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013. DOI: 10.1109/TPAMI.2013.50.

[30] H. Ramchoun, Y. Ghanou, M. Ettaouil, and M. A. Janati Idrissi, "Multilayer perceptron: Architecture optimization and training," 2016.

[31] S. I. Gallant *et al.*, "Perceptron-based learning algorithms," *IEEE Transactions on neural networks*, vol. 1, no. 2, pp. 179–191, 1990.

[32] P. Ramachandran, B. Zoph, and Q. V. Le, *Searching for activation functions*, 2017. DOI: 10.48550/ARXIV.1710.05941. [Online]. Available: https://arxiv.org/abs/1710.05941.

[33] Z. Tüske, M. A. Tahir, R. Schlüter, and H. Ney, "Integrating gaussian mixtures into deep neural networks: Softmax layer with hidden variables," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 4285–4289. DOI: 10.1109/ICASSP.2015.7178779.

[34] G. Zhong, L.-N. Wang, X. Ling, and J. Dong, "An overview on data representation learning: From traditional feature learning to recent deep learning," *The Journal of Finance and Data Science*, vol. 2, no. 4, pp. 265–278, 2016, ISSN: 2405-9188. DOI: https://doi.org/10.1016/j.jfds.2017.05.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405918816300459.

[35] D. Shen, G. Wu, and H.-I. Suk, "Deep learning in medical image analysis," *Annual review of biomedical engineering*, vol. 19, p. 221, 2017.

[36] P. Rodrıguez, M. A. Bautista, J. Gonzalez, and S. Escalera, "Beyond one-hot encoding: Lower dimensional target embedding," *Image and Vision Computing*, vol. 75, pp. 21–31, 2018.

[37] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[38] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, pp. 64–67, 2001.

[39] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, 2016. DOI: 10.48550/ARXIV.1603.07285. [Online]. Available: https://arxiv.org/abs/1603.07285.

[40] A. Canziani and Y. LeCun, *Nyu deep learning, spring 2020*, 2020. [Online]. Available: https://atcold.github.io/pytorch-Deep-Learning/.

[41] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[42] A. M. Braşoveanu and R. Andonie, "Visualizing transformers for nlp: A brief survey," in *2020 24th International Conference Information Visualisation (IV)*, IEEE, 2020, pp. 270–279.

[43] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Comput. Surv.*, vol. 55, no. 6, Dec. 2022, ISSN: 0360-0300. DOI: 10.1145/3530811. [Online]. Available: https://doi.org/10.1145/3530811.

[44] M. Seeger and S. Ultra-Large-Sites, "Key-value stores: A practical overview," *Computer Science and Media, Stuttgart*, 2009.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[46] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016. DOI: 10.48550/ARXIV.1607.06450. [Online]. Available: https://arxiv.org/abs/1607.06450.

[47] X. Huang, A. Khetan, M. Cvitkovic, and Z. Karnin, "Tabtransformer: Tabular data modeling using contextual embeddings," *arXiv preprint arXiv:2012.06678*, 2020.

[48] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[49] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2020. DOI: 10.48550/ARXIV.2010.11929. [Online]. Available: https://arxiv.org/abs/2010.11929.

[50] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, *Benchmarking graph neural networks*, 2020. DOI: 10.48550/ARXIV.2003.00982. [Online]. Available: https://arxiv.org/abs/2003.00982.

[51] I. N. Sneddon, *Fourier transforms*. Courier Corporation, 1995.

[52] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[53] F. R. Chung, *Spectral graph theory*. American Mathematical Soc., 1997, vol. 92.

[54] R. Merris, "A survey of graph laplacians," *Linear and Multilinear Algebra*, vol. 39, no. 1-2, pp. 19–31, 1995.

[55] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[56] R. A. Horn, "The hadamard product," in *Proc. Symp. Appl. Math*, vol. 40, 1990, pp. 87–169.

[57] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[58] A. Miraki and H. Saeedi-Sourck, "Spline graph filter bank with spectral sampling," *Circuits, Systems, and Signal Processing*, vol. 40, no. 11, pp. 5744–5758, Nov. 2021, ISSN: 1531-5878. DOI: 10.1007/s00034-021-01729-2. [Online]. Available: https://doi.org/10.1007/s00034-021-01729-2.

[59] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[60] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[61] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in neural information processing systems*, vol. 29, 2016.

[62] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[63] J. C. Mason and D. C. Handscomb, *Chebyshev polynomials*. Chapman and Hall/CRC, 2002.

[64] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Transactions on Signal Processing*, vol. 67, no. 1, pp. 97–109, 2018.

[65] F. Diele, L. Lopez, and R. Peluso, "The cayley transform in the numerical solution of unitary differential systems," *Advances in computational mathematics*, vol. 8, no. 4, pp. 317–334, 1998.

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

[67] S. Sukhbaatar, R. Fergus, *et al.*, "Learning multiagent communication with backpropagation," *Advances in neural information processing systems*, vol. 29, 2016.

[68] M. Balcilar, G. Renton, P. Heroux, B. Gauzere, S. Adam, and P. Honeine, *Bridging the gap between spectral and spatial domains in graph neural networks*, 2020. DOI: 10.48550/ARXIV.2003.11702. [Online]. Available: https://arxiv.org/abs/2003.11702.

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

[70] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[71] M. Y. Wang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *ICLR workshop on representation learning on graphs and manifolds*, 2019.

[72] C. Mao, L. Yao, and Y. Luo, "Towards expressive graph representation," *arXiv preprint arXiv:2010.05427*, 2020.

[73] P. V. G. C. A. Casanova, A. R. P. Lio, and Y. Bengio, "Graph attention networks," *ICLR. Petar Velickovic Guillem Cucurull Arantxa Casanova Adriana Romero Pietro Liò and Yoshua Bengio*, 2018.

[74] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, 2017. DOI: 10.48550/ARXIV.1710.10903. [Online]. Available: https://arxiv.org/abs/1710.10903.

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

[76] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in *2018 Chinese control and decision conference (CCDC)*, IEEE, 2018, pp. 1836–1841.

[77] S. Brody, U. Alon, and E. Yahav, *How attentive are graph attention networks?* 2021. DOI: 10.48550/ARXIV.2105.14491. [Online]. Available: https://arxiv.org/abs/2105.14491.

[78] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li, *Attention-based graph neural network for semi-supervised learning*, 2018. DOI: 10.48550/ARXIV.1803.03735. [Online]. Available: https://arxiv.org/abs/1803.03735.

[79] F. Rahutomo, T. Kitasuka, and M. Aritsugi, "Semantic cosine similarity," in *The 7th international student conference on advanced science and technology ICAST*, vol. 4, 2012, p. 1.

[80] K. Kaminski, J. Ludwiczak, M. Jasinski, *et al.*, "Rossmann-toolbox: A deep learning-based protocol for the prediction and design of cofactor specificity in rossmann fold proteins," 2021.

[81] X. Gao, W. Hu, and Z. Guo, *Exploring structure-adaptive graph learning for robust semi-supervised classification*, 2019. DOI: 10.48550/ARXIV.1904.10146. [Online]. Available: https://arxiv.org/abs/1904.10146.

[82] A. Taheri, "Minimizing the dirichlet energy over a space of measure preserving maps," *Topological Methods in Nonlinear Analysis*, vol. 33, no. 1, pp. 179–204, 2009.

[83] G. J. McLachlan, "Mahalanobis distance," *Resonance*, vol. 4, no. 6, pp. 20–26, 1999.

[84] A. Hasanzadeh, E. Hajiramezanali, S. Boluki, *et al.*, "Bayesian graph neural networks with adaptive connection sampling," in *International conference on machine learning*, PMLR, 2020, pp. 4094–4104.

[85] E. W. Weisstein, "Bernoulli distribution," *https://mathworld. wolfram. com/*, 2002.

[86] D. Luo, W. Cheng, W. Yu, *et al.*, "Learning to drop: Robust graph neural network via topological denoising," in *Proceedings of the 14th ACM international conference on web search and data mining*, 2021, pp. 779–787.

[87] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi, "A general framework for graph sparsification," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, 2011, pp. 71–80.

[88] A. Bretto, "Hypergraph theory," *An introduction. Mathematical Engineering. Cham: Springer*, 2013.

[89] S. Bai, F. Zhang, and P. H. S. Torr, *Hypergraph convolution and hypergraph attention*, 2019. DOI: 10.48550/ARXIV.1901.08150. [Online]. Available: https://arxiv.org/abs/1901.08150.

[90] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 3558–3565.

[91] A. I. Aviles-Rivero, C. Runkel, N. Papadakis, Z. Kourtzi, and C.-B. Schönlieb, "Multi-modal hypergraph diffusion network with dual prior for alzheimer classification," *arXiv preprint arXiv:2204.02399*, 2022.

[92] N. S. D'Souza, H. Wang, A. Giovannini, *et al.*, "Fusing modalities by multiplexed graph neural networks for outcome prediction in tuberculosis," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2022, pp. 287–297.

[93] S.-J. Kim, J.-W. Ha, and B.-T. Zhang, "Bayesian evolutionary hypergraph learning for predicting cancer clinical outcomes," *Journal of biomedical informatics*, vol. 49, pp. 101–111, 2014.

[94] Y.-T. Wang, Q.-W. Wu, Z. Gao, J.-C. Ni, and C.-H. Zheng, "Mirna-disease association prediction via hypergraph learning based on high-dimensionality features," *BMC Medical Informatics and Decision Making*, vol. 21, no. 1, pp. 1–13, 2021.

[95] A. Tariq, S. Tang, H. Sakhi, *et al.*, "Fusion of imaging and non-imaging data for disease trajectory prediction for covid-19 patients," *medRxiv*, 2021. DOI: 10.1101/2021.12.02.21267211. eprint: https://www.medrxiv.org/content/early/2021/12/05/2021.12.02.21267211.full.pdf. [Online]. Available: https://www.medrxiv.org/content/early/2021/12/05/2021.12.02.21267211.

[96] S. Tang, A. Tariq, J. Dunnmon, *et al.*, *Multimodal spatiotemporal graph neural networks for improved prediction of 30-day all-cause hospital readmission*, 2022. DOI: 10.48550/ARXIV.2204.06766. [Online]. Available: https://arxiv.org/abs/2204.06766.

[97] J. Bai, B. Li, and S. Nabavi, "Semi-supervised classification of disease prognosis using cr images with clinical data structured graph," in *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, ser. BCB '22, Northbrook, Illinois: Association for Computing Machinery, 2022, ISBN: 9781450393867. DOI: 10.1145/3535508.3545548. [Online]. Available: https://doi.org/10.1145/3535508.3545548.

[98] E. Rocheteau, C. Tong, P. Veličković, N. Lane, and P. Liò, *Predicting patient outcomes with graph representation learning*, 2021. DOI: 10.48550/ARXIV.2101.03940. [Online]. Available: https://arxiv.org/abs/2101.03940.

[99] J. G. D. Ochoa and F. Mustafa, "Graph neural network modelling as a potentially effective method for predicting and analyzing procedures based on patient diagnoses," *medRxiv*, 2021. DOI: 10.1101/2021.11.25.21266465. eprint: https://www.medrxiv.org/content/early/2021/12/02/2021.11.25.21266465.full.pdf. [Online]. Available: https://www.medrxiv.org/content/early/2021/12/02/2021.11.25.21266465.

[100] L. Grassi, S. Sabato, E. Rossi, L. Marmai, and B. Biancosino, "Affective syndromes and their screening in cancer patients with early and stable disease: Italian icd-10 data and performance of the distress thermometer from the southern european psycho-oncology study (sepos)," *Journal of affective disorders*, vol. 114, no. 1-3, pp. 193–199, 2009.

[101] H. Lu and S. Uddin, "A weighted patient network-based framework for predicting chronic diseases using graph neural networks," *Scientific Reports*, vol. 11, no. 1, p. 22 607, Nov. 2021, ISSN: 2045-2322. DOI: 10.1038/s41598-021-01964-2. [Online]. Available: https://doi.org/10.1038/s41598-021-01964-2.

[102] L. Cosmo, A. Kazi, S.-A. Ahmadi, N. Navab, and M. Bronstein, "Latent-graph learning for disease prediction," in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2020*, Springer International Publishing, 2020, pp. 643–653. DOI: 10.1007/978-3-030-59713-9_62. [Online]. Available: https://doi.org/10.1007%5C%2F978-3-030-59713-9_62.

[103] Y. Huang and A. C. S. Chung, *Edge-variational graph convolutional networks for uncertainty-aware disease prediction*, 2020. DOI: 10.48550/ARXIV.2009.02759. [Online]. Available: https://arxiv.org/abs/2009.02759.

[104] S. Zheng, Z. Zhu, Z. Liu, *et al.*, "Multi-modal graph learning for disease prediction," *IEEE Transactions on Medical Imaging*, vol. 41, no. 9, pp. 2207–2216, 2022. DOI: 10.1109/TMI.2022.3159264.

[105] S. Sohangir and D. Wang, "Improved sqrt-cosine similarity measurement," *Journal of Big Data*, vol. 4, no. 1, p. 25, Jul. 2017, ISSN: 2196-1115. DOI: 10.1186/s40537-017-0083-6. [Online]. Available: https://doi.org/10.1186/s40537-017-0083-6.

[106] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 539–546 vol. 1. DOI: 10.1109/CVPR.2005.202.

[107] S. Mo, Z. Sun, and C. Li, *Siamese prototypical contrastive learning*, 2022. DOI: 10.48550/ARXIV.2208.08819. [Online]. Available: https://arxiv.org/abs/2208.08819.

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

[109] L. Faber, A. K. Moghaddam, and R. Wattenhofer, *Contrastive graph neural network explanation*, 2020. DOI: 10.48550/ARXIV.2010.13663. [Online]. Available: https://arxiv.org/abs/2010.13663.

[110] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep Graph Contrastive Representation Learning," in *ICML Workshop on Graph Representation Learning and Beyond*, 2020. [Online]. Available: https://arxiv.org/abs/2006.04131.

[111] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Graph Contrastive Learning with Adaptive Augmentation," in *Proceedings of The Web Conference 2021*, ser. WWW '21, Ljubljana, Slovenia: Association for Computing Machinery, Apr. 2021, pp. 2069–2080, ISBN: 9781450370233. DOI: 10.1145/3442381.3449802. [Online]. Available: https://doi.org/10.1145/3442381.3449802.

[112] C. Gentile and M. K. Warmuth, "Linear hinge loss and average margin," *Advances in neural information processing systems*, vol. 11, 1998.

[113] T. E. Cecil and P. J. Ryan, "Distance functions and umbilic submanifolds of hyperbolic space," *Nagoya Mathematical Journal*, vol. 74, pp. 67–75, 1979.

[114] S. Ge, S. Mishra, S. Kornblith, C.-L. Li, and D. Jacobs, *Hyperbolic contrastive learning for visual representations beyond objects*, 2022. DOI: 10.48550/ARXIV.2212.00653. [Online]. Available: https://arxiv.org/abs/2212.00653.

[115] J. Liu, M. Yang, M. Zhou, S. Feng, and P. Fournier-Viger, *Enhancing hyperbolic graph embeddings via contrastive learning*, 2022. DOI: 10.48550/ARXIV.2201.08554. [Online]. Available: https://arxiv.org/abs/2201.08554.

[116] S. J. S. M. P. P. M. R. H. J. B. E. B. J.; and K. T., "Stony brook university covid-19 positive cases," 2021. DOI: https://doi.org/10.7937/TCIA.BBAG-2923. [Online]. Available: https://doi.org/10.7937/TCIA.BBAG-2923.

[117] M.-P. Revel, S. Boussouar, C. de Margerie-Mellon, *et al.*, "Study of thoracic ct in covid-19: The stoic project," *Radiology*, vol. 301, no. 1, E361–E370, 2021, PMID: 34184935. DOI: 10.1148/radiol.2021210384. eprint: https://doi.org/10.1148/radiol.2021210384. [Online]. Available: https://doi.org/10.1148/radiol.2021210384.

[118] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation.," *Encyclopedia of database systems*, vol. 5, pp. 532–538, 2009.

[119] L. Biewald, *Experiment tracking with weights and biases*, Software available from wandb.com, 2020. [Online]. Available: https://www.wandb.com/.

[120] Y. Luo, "Evaluating the state of the art in missing data imputation for clinical data," *Briefings in Bioinformatics*, vol. 23, no. 1, bbab489, 2022.

[121] A. Dıaz Berenguer, T. Mukherjee, M. Bossa, N. Deligiannis, and H. Sahli, "Representation learning with information theory for covid-19 detection," *arXiv e-prints*, arXiv–2207, 2022.

[122] S. L. Walston, T. Matsumoto, Y. Miki, and D. Ueda, "Artificial intelligence-based model for covid-19 prognosis incorporating chest radiographs and clinical data; a retrospective model development and validation study," *The British Journal of Radiology*, vol. 95, no. 1140, p. 20 220 058, 2022.

[123] P. Soda, N. C. D'Amico, J. Tessadori, *et al.*, "Aiforcovid: Predicting the clinical outcomes in patients with covid-19 applying ai to chest-x-rays. an italian multicentre study," *Medical image analysis*, vol. 74, p. 102 216, 2021.

[124] T. Matsumoto, S. L. Walston, M. Walston, *et al.*, "Deep learning–based time-to-death prediction model for covid-19 patients using clinical data and chest radiographs," *Journal of Digital Imaging*, pp. 1–11, 2022.

[125] H. Aboutalebi, M. Pavlova, M. J. Shafiee, A. Florea, A. Hryniowski, and A. Wong, "Covid-net biochem: An explainability-driven framework to building machine learning models for predicting survival and kidney injury of covid-19 patients from clinical and biochemistry data," *arXiv preprint arXiv:2204.11210*, 2022.

[126] X. V. Nguyen, E. Dikici, S. Candemir, R. L. Ball, and L. M. Prevedello, "Mortality prediction analysis among covid-19 inpatients using clinical variables and deep learning chest radiography imaging features," *Tomography*, vol. 8, no. 4, pp. 1791–1803, 2022.

[127] S. Takase and N. Okazaki, "Positional encoding to control output sequence length," *arXiv preprint arXiv:1904.07418*, 2019.

[128] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, *A convnet for the 2020s*, 2022. DOI: 10.48550/ARXIV.2201.03545. [Online]. Available: https://arxiv.org/abs/2201.03545.

[129] J. M. Joyce, "Kullback-leibler divergence," in *International Encyclopedia of Statistical Science*, M. Lovric, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 720–722, ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_327. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_327.

[130] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html.

[131] S. Jang, S.-E. Moon, and J.-S. Lee, "Eeg-based emotional video classification via learning connectivity structure," *IEEE Transactions on Affective Computing*, 2021.

[132] M. M. Wolf, A. M. Klinvex, and D. M. Dunlavy, "Advantages to modeling relational data using hypergraphs versus graphs," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, 2016, pp. 1–7.

[133] W. Jiang, J. Qi, J. X. Yu, J. Huang, and R. Zhang, "Hyperx: A scalable hypergraph framework," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 909–922, 2018.

[134] J. Gao, T. Lyu, F. Xiong, J. Wang, W. Ke, and Z. Li, "Predicting the survival of cancer patients with multimodal graph neural network," en, *IEEE/ACM Trans Comput Biol Bioinform*, vol. 19, no. 2, pp. 699–709, Apr. 2022.

[135] J. Gao, T. Lyu, F. Xiong, J. Wang, W. Ke, and Z. Li, "Mgnn: A multimodal graph neural network for predicting the survival of cancer patients," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 1697–1700.

[136] Y. Liu, H. Li, T. Luo, *et al.*, "Structural attention graph neural network for diagnosis and prediction of covid-19 severity," *IEEE Transactions on Medical Imaging*, pp. 1–1, 2022. DOI: 10.1109/TMI.2022.3226575.

[137] X. Xing, F. Yang, H. Li, *et al.*, "Multi-level attention graph neural network based on co-expression gene modules for disease diagnosis and prognosis," *Bioinformatics*, vol. 38, no. 8, pp. 2178–2186, 2022.

[138] Z. Sun, H. Yin, H. Chen, T. Chen, L. Cui, and F. Yang, "Disease prediction via graph neural networks," *IEEE Journal of Biomedical and Health Informatics*, vol. 25, no. 3, pp. 818–826, 2020.

[139] X.-M. Zhang, L. Liang, L. Liu, and M.-J. Tang, "Graph neural networks and their current applications in bioinformatics," *Frontiers in genetics*, vol. 12, 2021.

[140] B.-W. Zhao, Z.-H. You, L. Hu, L. Wong, B.-Y. Ji, and P. Zhang, "A multi-graph deep learning model for predicting drug-disease associations," in *International Conference on Intelligent Computing*, Springer, 2021, pp. 580–590.

[141] G. Xie, J. Li, G. Gu, *et al.*, "Bgmsdda: A bipartite graph diffusion algorithm with multiple similarity integration for drug–disease association prediction," *Molecular Omics*, vol. 17, no. 6, pp. 997–1011, 2021.

Appendix A

Dataset

A.1 STOIC Dataset

variable name	variable distribution	variable description
Age	continuous	age of patient
Gender	{male : 5730, 'female': 4270}	gender of patient
RT-PCR results	{'positive': 6448, 'negative' : 3552}	COVID-19 test
Outcome at one month	{'severe': 1575, 'no severe': 8425}	death or need for intubation

Table A.1: Description of the non-imaging clinical data in STOIC dataset

A.2 SBU Dataset

variable_name	variable distribution	variable description
covid19_statuses	{'positive': 1479}	COVID-19 status as determined through either positive PCR test or outside documentation reviewed by clinician
last. status	{'deceased': 194, 'discharged': 1285}	For the selected visit, the status of the patient
age.splits	{'(59,74]': 376, '(74,90]': 297, '[18,59]': 806}	Age intervals (in years) at time of admission; truncated for patients over 90 years of age
gender_concept_name	{'FEMALE': 635, 'MALE': 810}	Documented gender in the EHR (Electronic Health Record); gender is dropped in select cases for de-identification purposes
visit_start_datetime		Date shifted start of selected visit
visit_concept_name	{'Emergency Room Visit': 376, 'Inpatient Visit': 1100, 'Outpatient Visit': 3}	For the selected visit, the type of the visit; note patients can be admitted for observation (multiple days) in emergency department
is_icu	{False: 1201, True: 278}	Patient admitted to the ICU based on documented room charges
was_ventilated	{'No': 1254, 'Yes': 225}	The patient had invasive ventilation
invasive_vent_days		Number of documented days of invasive ventilation support
length_of_stay		Number of calendar days in the facility

Acute.Hepatic.Injury during hospitalization.	{'No': 1166, 'Yes': 48}	ALT or AST was at least 15 times above the test's upper limit
Acute.Kidney. Injury during hospitalization.	{'No': 1206, 'Yes': 273}	Had an increase in serum creatinine of 0.3 mg/dL within 48 hours
Urine protein	{'Abnormal': 387, 'Normal': 211}	Protein in urine
kidney_replacement_therapy	{'Yes': 73}	Did the patient have documented renal replacement therapy
kidney_transplant	{'Yes': 23}	Did the patient have a kidney transplant
htn_v	{'No': 642, 'Yes': 525}	Documented ICD-10 code for HTN (Hypertension), taking anti-hypertensive medications and/or documented SBP>140/90
dm_v	{'No': 880, 'Yes': 289}	Documented ICD-10 code for diabetes type 1 or 2, taking insulin or other oral medications for diabetes
cad_v	{'No': 994, 'Yes': 171}	Documented ICD-10 code for CAD (Coronary Artery Disease), history of stent placement, existing cath report documenting disease
hf_ef_v	{'HFpEF': 46, 'HFrEF': 30, 'No': 1080}	For HFrEF, documented ICD-10 code for HFrEF or echocardiogram documenting reduced ejection fraction (reduced EF is <40%, 40% or higher is preserved EF). For HFpEF, documented ICD-10 code for HFpEF or echocardiogram documenting diastolic dysfunction.
ckd_v	{'No': 1079, 'Yes': 87}	Documented ICD-10 code for CKD (Chronic Kidney Disease), reduced GFR on lab work.
malignancies_v	{'No': 1048, 'Yes': 106}	Documented ICD-10 code for malignancies, receiving treatment for active malignancy
copd_v	{'No': 1096, 'Yes': 69}	Documented ICD-10 code for COPD (Chronic obstructive pulmonary disease), PFTs documenting obstructive defect along with positive smoking history
other_lung_disease_v	{'No': 992, 'Yes': 172}	Documented ICD-10 code for other lung diseases including asthma, ILD, pulmonary hypertension, chronic PE, lung resection
acei_v	{'No': 1010, 'Yes': 147}	Admission medication reconciliation documenting use of ACE Inhibitor as a home medication
arb_v	{'No': 995, 'Yes': 164}	Admission medication reconciliation documenting use of ARB (Angiotensin receptor blockers) as a home medication
antibiotics_use_v	{'No': 858, 'Yes': 276}	Patient on an antibiotic prior to presentation
nsaid_use_v	{'No': 1017, 'Yes': 92}	Admission medication reconciliation documenting use of NSAID (Non-steroidal anti-inflammatory drug) as a home medication
days_prior_sx		The number of days prior to the presentation that symptoms began
smoking_status_v	{'Current': 38, 'Former': 263, 'Never': 805}	Marks patient's smoking status as either current, former, never smoker, or unknown. This refers only to cigarettes and cigars. E-cigarettes and marijuana do not count.
cough_v	{'No': 209, 'Yes': 897}	Reported cough
dyspnea_admission_v	{'No': 313, 'Yes': 807}	Shortness of breath on admission
nausea_v	{'No': 788, 'Yes': 286}	Reported nausea
vomiting_v	{'No': 923, 'Yes': 159}	Reported vomiting
diarrhea_v	{'No': 726, 'Yes': 365}	Reported diarrhea
abdominal_pain_v	{'No': 962, 'Yes': 114}	Abdominal pain symptom

fever_v	{'No': 238, 'Yes': 885}	Yes or no. Only refers to subjective or objective fever at home. Fever in ED does not count.
BMI.over30	{False: 609, True: 417}	Body mass index over 30
BMI.over35	{False: 834, True: 192}	Body mass index over 35
temperature.over38	{False: 1091, True: 348}	Temperature at time of admission over 38 degrees centigrade
pulseOx.under90	{False: 1256, True: 216}	Measured blood oxygen level below 90% using a pulse oximeter
Respiration.over24	{False: 1178, True: 293}	Respiration rate over 24 breaths per minute
HeartRate.over100	{False: 831, True: 640}	Heart rate over 100 beats per minute
Lymphocytes.under1k	{False: 487, True: 557}	Serum lymphocytes
Aspartate.over40	{False: 618, True: 585}	Serum AST measuring liver function
Alanine.over60	{False: 968, True: 235}	Serum ALT measuring liver function
A1C.over6.5	{False: 225, True: 234}	Serum hemoglobin A1C over 6.5
A1C.under6.5	{False: 234, True: 225}	
A1C.6.6to7.9	{False: 348, True: 111}	
A1C.8to9.9	{False: 395, True: 64}	
A1C.over10	{False: 400, True: 59}	
Sodium.above145	{False: 1211, True: 50}	Serum sodium
Sodium.between135and145	{False: 441, True: 820}	
Sodium.below135	{False: 870, True: 391}	
Potassium.above5.2	{False: 1152, True: 49}	Serum potassium
Potassium.between3.5and5.2	{False: 147, True: 1054}	
Potassium.below3.5	{False: 1103, True: 98}	
Chloride.above107	{False: 1214, True: 47}	Serum chloride
Chloride.between96and107	{False: 435, True: 826}	
Chloride.below96	{False: 873, True: 388}	
Bicarbonate.above31	{False: 1240, True: 20}	Serum bicarbonate
Bicarbonate.between21and31	{False: 232, True: 1028}	
Bicarbonate.below21	{False: 1048, True: 212}	
Blood_Urea_Nitrogen.above20	{False: 845, True: 416}	Serum BUN measured for kidney function

Blood_Urea_Nitrogen.between5and20	{False: 433, True: 828}	
Blood_Urea_Nitrogen.below5	{False: 1244, True: 17}	
Creatinine.above1.2	{False: 920, True: 341}	Serum creatinine measured for kidney function
Creatinine.between0.5and1.2	{False: 375, True: 886}	
Creatinine.below0.5	{False: 1227, True: 34}	
eGFR.above60	{False: 340, True: 902}	Serum estimated glomerular filtration rate
eGFR.between30and60	{False: 1003, True: 239}	
eGFR.below30	{False: 1141, True: 101}	
blood_pH.above7.45	{False: 183, True: 60}	
blood_pH.between7.35and7.45	{False: 126, True: 117}	
blood_pH.below7.35	{False: 177, True: 66}	
Troponin.above0.01	{False: 821, True: 222}	Serum indicator of damage to heart tissue
D_dimer.above3000	{False: 961, True: 63}	Serum D-dimer
D_dimer.between500and3000	{False: 739, True: 285}	
D_dimer.below500	{False: 348, True: 676}	
ESR.above30	{False: 187, True: 455}	Serum erythrocyte sedimentation rate
Microscopic_hematuria.above2	{False: 379, True: 209}	Blood in urine
SBP.below120	{False: 958, True: 515}	Systolic blood pressure
SBP.between120and139	{False: 938, True: 535}	
SBP.above139	{False: 1050, True: 423}	
MAP.below65	{False: 1241, True: 25}	Mean arterial pressure
MAP.between65and90	{False: 685, True: 581}	
MAP.above90	{False: 606, True: 660}	
procalcitonin.below0.25	{False: 380, True: 735}	Serum procalcitonin
procalcitonin.between0.25and0.5	{False: 937, True: 178}	
procalcitonin.above0.5	{False: 913, True: 202}	
ferritin.above1k	{False: 613, True: 343}	Serum ferritin
Proteinuria.above80	{False: 149, True: 81}	
8331-1_Oral temperature		Oral temperature in degrees C

59408-5_Oxygen saturation in Arterial blood by Pulse oximetry		
9279-1_Respiratory rate		
76282-3_Heart rate.beat-to-beat by EKG		
8480-6_Systolic blood pressure		
76536-2_Mean blood pressure by Noninvasive		
33256-9_Leukocytes [#/volume] corrected for nucleated erythrocytes in Blood by Automated count		
751-8_Neutrophils [#/volume] in Blood by Automated count		
731-0_Lymphocytes [#/volume] in Blood by Automated count		
2951-2_Sodium [Moles/volume] in Serum or Plasma		
1920-8_Aspartate aminotransferase [Enzymatic activity/volume] in Serum or Plasma		
1744-2_Alanine aminotransferase [Enzymatic activity/volume] in Serum or Plasma by No addition of P-5'-P		
2157-6_Creatine kinase [Enzymatic activity/volume] in Serum or Plasma		
2524-7_Lactate [Moles/volume] in Serum or Plasma		
6598-7_Troponin T.cardiac [Mass/volume] in Serum or Plasma		
33762-6_Natriuretic peptide.B prohormone N-Terminal [Mass/volume] in Serum or Plasma		
75241-0_Procalcitonin [Mass/volume] in Serum or Plasma by Immunoassay		
48058-2_Fibrin D-dimer DDU [Mass/volume] in Platelet poor plasma by Immunoassay		
2276-4_Ferritin [Mass/volume] in Serum or Plasma		
1988-5_C reactive protein [Mass/volume] in Serum or Plasma		

4548-4_Hemoglobin A1c/Hemoglobin.total in Blood		
39156-5_Body mass index (BMI) [Ratio]		
2951-2_Sodium [Moles/volume] in Serum or Plasma.1		
2823-3_Potassium [Moles/volume] in Serum or Plasma		
2075-0_Chloride [Moles/volume] in Serum or Plasma		
1963-8_Bicarbonate [Moles/volume] in Serum or Plasma		
3094-0_Urea nitrogen [Mass/volume] in Serum or Plasma		
2160-0_Creatinine [Mass/volume] in Serum or Plasma		
62238-1_Glomerular filtration rate/1.73 sq M.predicted [Volume Rate/Area] in Serum, Plasma or Blood by Creatinine-based formula (CKD-EPI)		
33254-4_pH of Arterial blood adjusted to patient's actual temperature		
30341-2_Erythrocyte sedimentation rate		
2345-7_Glucose [Mass/volume] in Serum or Plasma		
13457-7_Cholesterol in LDL [Mass/volume] in Serum or Plasma by calculation		
13458-5_Cholesterol in VLDL [Mass/volume] in Serum or Plasma by calculation		
2571-8_Triglyceride [Mass/volume] in Serum or Plasma		
2085-9_Cholesterol in HDL [Mass/volume] in Serum or Plasma		
therapeutic.exnox.Boolean	{False: 1020, True: 355}	Patient had enoxaparin administered
therapeutic.heparin.Boolean	{False: 1190, True: 185}	Patient had heparin administered at a therapeutic dose

Other.anticoagulation.therapy	{'apixaban': 89, 'argatroban': 11, 'dabigatran': 1, 'multiple': 9, 'not documented': 1223, 'rivaroxaban': 20, 'wafarin': 22}	Patient had other anticoagulation therapy as listed

Table A.2: description of the \acrshort{ehr} data in SBU dataset