Otto-von-Guericke University Magdeburg


Department of Computer Science
Data and Knowledge Engineering Group (DKE)

Master Thesis

# Graph representation of patient's data in EHR for outcome prediction

Author:
Nasim Ahmed

February 27, 2023

Supervisors:
Prof. Dr.-Ing. Gunter Saake
(Department of Computer Science, Otto-von-Guericke University)
Dr. Ing. Robert Heyer
(Department of Computer Science, Otto-von-Guericke University)
Dr.-Ing. David Broneske
Daniel Walke
(Department of Computer Science, Otto-von-Guericke University)
Daniel Micheel
(Department of Computer Science, Otto-von-Guericke University)

**Ahmed, Nasim Uddin:**
*Graph representation in electronic health records for outcome prediction*
Master Thesis, Otto-von-Guericke University
Magdeburg, 2022.

# Contents

## Abstract

Prolific growth has been observed recently in the healthcare sector through Artificial Intelligence(AI). Therefore, researchers are working with healthcare data available in the form of Electronic Health Records (EHR) to improve resource management(by predicting mortality and length of stay) as well as improve diagnostics. However, the EHR data are difficult to handle because of their heterogeneous nature and temporal aspects. Furthermore, it is an arduous task to classify and analyze sparse data arising from patient diagnoses, medication, and lab events. This master thesis focuses on addressing the problem of handling time-dependent patient data recorded at specific intervals in the Intensive Care Unit(ICU). To do this, I tested two different models, long short-term memory graph attention networks (LSTM-GAT) and a long short-term memory single layer perceptron (LSTM-SLP). LSTM-GAT was proposed by Rocheteau et al. [38] and represents a hybrid model consisting of an LSTM and GAT model. Similarly, LSTM-SLP is a hybrid model consisting of an LSTM and Single Layer Perceptron. Both models showed comparatively similar performance on a classification task (mortality prediction) and a regression task (length of stay prediction). Therefore, I can conclude that for simple tasks such as mortality and length of stay prediction, complex graph-based modeling is not required. In addition, my experiment showed that adding diagnoses to the features improved the performance of both models. I have also evaluated the query time for specific information in a graph database(Neo4j) and in a relational database(PostgreSQL).

# 1 Introduction

## 1.1 Introduction

Royal Society of Medicine, famously quoted on the use of AI in healthcare:

"*AI has the potential to augment and enhance the capabilities of healthcare professionals, enabling them to provide better care with greater accuracy and increased efficiency.* "[35]

This quote highlights the potential benefits of incorporating AI (Artificial Intelligence) into healthcare. AI has the ability to analyze and interpret large amounts of data quickly and accurately, which can help healthcare professionals make more informed decisions about patient care. However, modern medicines are becoming increasingly complex due to a variety of genetic processes and environmental interactions that the human body undergoes. As a result, the depth of medical knowledge is exponentially developing.[14]. For example, the International Classification of Diseases (ICD)-10, which currently contains approximately 68,000 diagnoses (five times as many as the ICD-9), is constantly expanding [33]. Additionally, research on human genetic, environmental, and lifestyle variations has advanced the notion that medical treatment should be tailored to the unique needs of each patient[32]. Consequently, more patient-specific information needs to be added which requires storing and processing. With existing tools to store and share information in the current era, an aided human can comprehend and utilize the extraordinary amount of knowledge generated in medicine through artificial intelligence (AI).[34]. While there could be direct contributions of AI in this domain where the patient's condition is actively improved through disease prediction or drug prescription, there are also indirect contributions that could be made by AI such as better

resource management, mortality prediction of patients in hospitals, especially intensive care units (ICU).

A clinical decision support system is a computer-based tool that provides healthcare professionals with real-time patient-specific information and knowledge to support clinical decision-making. They can use various sources of information such as electronic health records, laboratory results, medication orders, and patient-specific data to provide evidence-based recommendations for clinical care. For instance, predicting a patient's length of stay in the ICU may help healthcare professionals make medical decisions, and manage medical staff and resources [1]. Therefore, machine learning (ML) when applied to clinical data can offer future insights with a high level of precision, in a healthcare environment to forecast prospective outcomes of interest [4]. Clinicians or physicians are frequently interested in forecasting adverse outcomes so that the required steps may be taken to prevent them (if preventable) or get ready for them. A few examples include the development of an illness, hospital readmission, and mortality.

Sometimes physicians or clinicians can also make false decisions leading to adverse patient outcomes and increased costs. Examples include patients being frequently readmitted after premature discharge from ICU [23], and physicians not considering potential interactions with other medications the patient is taking leading to adverse drug reactions, and/or patients frequently undergoing unnecessary operations, some resulting in immediate death [3]. Similarly, incorrect diagnoses and unnecessary medical testing are becoming more prevalent [5] causing a detrimental effect on patient's health and rising expenses.

The use of ML in healthcare is expected to play a significant role in raising the standard of care [21] through analyzing clinical data like Electronic Health Records (EHR). EHRs are real-time, digital patient records that gives authorized healthcare professionals secure access when needed across different healthcare facilities. The EHR's structure is chronologically ordered and include (a) clinical information about the patient, such as symptoms, treatments, diagnoses, lab results, and medications given to them and: (b) demographic information about the patient (age, gender, etc.)

One of the most important steps in developing ML models is the extraction of features which act as predictor variables (predictors). Predictors can be

any of the input features(e.g., age, gender). In ML, the process of extracting features is frequently referred to as feature engineering. Frequently domain-based feature extraction is used to create prediction models (i.e., a domain expert chooses the relevant features). However, this approach does not scale well, because the selection of predictors from the input features is task-specific, For example, a characteristic such as "blood pressure" is more relevant to the prediction of "cardiovascular disease" than it is to the prediction of "skin allergies".

Another challenge when dealing with a huge amount of clinical data like EHR data is storing them appropriately. The idea of big data is gradually having an impact on the healthcare sector. The relational database model has been commonly used for decades in healthcare data storage and management systems. Given that each type of patient's data has a unique schema and format, storing and managing such diverse, dispersed, and unstructured data presents difficult problems. An individual patient's data may be dispersed across multiple tables and can have different representations (e.g., data can be stored across demographic, admissions, diagnosis table). Doctors and other healthcare professionals typically require uniform access to this data at the same time. Modelling EHR data as a graph is more appropriate and prevents the usage of cost-intensive join operations. To address the problem, I propose an appropriate graph database model in Neo4j which can hold patient data in nodes and edges. I intend to compare the performance of using a graph database model, particularly in Neo4j against the relational database model, PostgreSQL.

Adhering to the above issues, I propose the use of graph data structure to analyze EHR records, for predicting patients' mortality and length of stay. Graphs provide an appropriate way to model EHR data because of high inter-dependencies within the data. After modeling, the data could even be fetched easily even for analysis. My research has shown that graph neural networks are useful tools for analyzing EHR records. Furthermore, graph databases are dynamic and horizontally scalable making them suitable for complex medical data.

The thesis aims to investigate the following research questions related to using graph-based patient representation to predict the outcomes of patients in the ICU using EHR:

Graph Neural Networks(GNN) are a type of neural network that is designed to operate on graph-structured data, such as EHRs. GNNs can capture complex relationships between different elements of the EHRs, such as patients, diagnoses, medications, and procedures(explained in depth in background section). This allows them to model the dependencies and interactions between these elements, which can be important for predicting patient outcomes.

Single layer of perceptrons are a simple type of neural network, that perform a weighted sum of their inputs and apply a nonlinear activation function(explained in depth in background section). Therefore, the first research question I would try to answer is:

**RQ1: Do graph neural networks perform better predictions than single-layer perceptrons(SLP)?**

To address this research question, the thesis will evaluate the performance of GNNs and SLPs for predicting patient outcomes using EHR data. This evaluation will likely involve training and testing both types of models on the same dataset and comparing their performance using metrics such as such as area under the Receiver Operator Characteristic curve (AUROC) and area under the precision recall curve(AUPRC).

Patient diagnoses can be a critical factor in predicting outcomes as they may indicate the severity of a patient's condition, the likelihood of complications, and the appropriate treatment options. However, it is not clear how important diagnoses are relative to other patient features, such as demographics or vital signs, in predicting outcomes. Therefore, the second research question I would try to answer is:

**RQ2: How important are the patient's diagnoses for the prediction with a graph neural network and for the prediction with an SLP?**

To answer this question, the thesis will evaluate the impact of patient diagnoses on outcome prediction using both the hybrid LSTM-GNN model and the hybrid LSTM-MLP model. This evaluation will highlight the relative importance of patient diagnoses in predicting patient outcomes with these models. The methodology of the thesis will build on the work of Rocheteau et al.[38], who proposed a graph-based patient representation technique using common diagnoses and a hybrid LSTM-GNN model to predict patient outcomes.

In recent years, graph databases have become increasingly popular for handling complex, interconnected data, such as social networks, recommendation engines, and biological networks. One advantage of graph databases is that they can efficiently handle queries that traverse relationships between nodes. In contrast, relational databases are optimized for handling queries involving structured data stored in tables. Therefore, the third research question I would try to answer is:

**RQ3: Are Cypher queries in Neo4j faster in fetching data than SQL queries in PostgreSQL?**

To address this research question, the thesis will create a graph model in the Neo4j database and upload the same dataset in a relational database (PostgreSQL). The thesis will then measure the query time taken to fetch data using both Cypher and SQL queries for various query types.

The results of this evaluation will provide insights into the relative performance of graph databases and relational databases for data retrieval. Specifically, it will assess whether Cypher queries in Neo4j are faster than SQL queries in PostgreSQL for retrieving data from a large, complex dataset. This information may be valuable for researchers and practitioners who are considering using graph databases for their applications.

This report is structured into several chapters, each covering a specific aspect of the thesis. Chapter 1 serves as an introduction to the research problem and outlines the objectives of the thesis. Chapter 2 provides a more in-depth background of the research topic, laying the groundwork for the subsequent chapters. Chapter 3 presents a summary of the current state of the art, including its benefits and drawbacks, which will inform the development of the proposed solution. Chapter 4 provides an overview of the dataset used for the research, explaining its relevance and characteristics. In Chapter 5, the graph data model and the proposed model architecture are discussed in detail, highlighting their advantages and limitations. Chapter 6 reports on the results and evaluation of experiments conducted to validate the proposed solution. Finally, Chapter 7 presents a conclusion and future work, summarizing the main findings and outlining potential avenues for future research.

# 2
# Background

In the following section, I will present the basic concepts which were required in my thesis to predict the mortality and length of stay of patients in the Intensive Care Unit(ICU). I will start by discussing the digital version of a patient's medical history and health information which is the electronic health records (EHR), the key target variables are the length of stay (LOS) and mortality prediction of patients. Furthermore, I will explain our choice of using a graph database to store EHR data graphically and provide a comprehensive understanding of graphs and their components. And finally, I will explain recurrent neural networks (RNNs), long short-term memory (LSTM), and graph neural networks (GNNs) in detail, which are the algorithms of primary focus in this research. Because using these algorithms would help me better analyze the EHR data which is being stored in the graph database.

## 2.1 AI in Healthcare: Electronic Health Records (EHR)

Artificial intelligence (AI) has been increasingly applied to healthcare in recent years, aiming to improve patient outcomes and reduce diagnostic costs within application areas such as medical imaging analysis [41], drug discovery [15], precision medicine [26], and virtual health assistants[27]. Leveraging the use of AI in performing calculations and analysis plays a crucial role in analyzing the vast array of data, that is available in the healthcare sector. The healthcare sector gathers, manages, and examines a wide variety of data, including EHR data, claims data, clinical trial data, administrative data, imaging data, research data, and more. EHR analysis is crucial for medical research and can raise the standard of treatment for patients. For instance, the prescriptions in EHRs can aid in medication

recommendations, distribution of cohorts based on patients' phenotypes, help hospitals to manage their resources effectively, and inform patients about possible risks of their underlying disease. Also, would facilitate the doctor to have a clear understanding of their patient's condition and improve their treatment.

Raw EHRs often include a variety of patient characteristics, such as demographics, medical history, diagnoses, prescriptions, diagnostic images, and time-dependent vitals. However, the major challenges of working with raw EHR data include handling the temporal nature, high dimensionality, and sparsity of the data. Temporality is a problem in EHR because it refers to the issue of recording and representing the order and timing of events in a patient's medical history. For instance, the order in which a patient received certain medications or treatments can impact the effectiveness of those treatments, and this information needs to be captured accurately in the EHR data. Also, some data have timestamps which are also important to analyze their time dependence. To address the challenges of temporality in EHR data, various techniques, such as time series analysis and longitudinal data analysis, can be used to model the data over time and identify the connections between various incidents in a patient's medical history. In addition to temporality, high dimensionality can also pose major challenge. High dimensionality in the EHR data means that a patient can have a large number of features(e.g, age, diagnosis, lab results, etc). This high dimensionality can make it challenging to analyze and interpret the data, as there are many potential relationships and patterns that need to be considered. For example, when trying to predict a patient's risk of a certain disease, there may be hundreds or thousands of variables in the data that could potentially impact the outcome. In such cases, it may be difficult to determine which variables are most important. Also, training the model will be slower as the dimensions increase. To overcome the challenges of high dimensionality in EHR data, various techniques, such as dimensionality reduction, feature selection, and machine learning algorithms, can be used to extract meaningful insights from the data and make more accurate predictions. Besides, features in the data often have missing values, which can cause sparsity. This can occur for various reasons, such as the incompleteness of the EHR system, missing information at the time of data entry, or a lack of standardization in the way data is collected and

recorded. The result of this sparsity is that many cells in an EHR data matrix are empty or have no recorded values. This can impact the accuracy and reliability of predictive models, since missing data can cause the model to be biased. To address the issues with EHR data sparsity, various techniques, such as imputation methods, can be used to fill in missing data, or machine learning algorithms that are designed to handle sparse data can be used for analysis and prediction. Modern deep learning models are capable of handling the aforementioned challenges.

### 2.1.1 Length of Stay (LOS)

Length of stay prediction is a task in healthcare where the goal is to predict the length of time that a patient will spend in a hospital or healthcare facility. This target variable is important for several reasons:

1. Resource allocation: Accurate predictions of the length of stay can help healthcare facilities allocate resources more effectively, such as staffing and bed allocation, and minimize the impact of resource constraints.

2. Patient care: Predicting length of stay can help healthcare providers plan for patient care and ensure that patients receive the appropriate level of care while they are in the facility.

3. Quality of treatment: Predicting length of stay can also help healthcare facilities monitor and improve the quality of treatment that patients receive, as shorter lengths of stay are often associated with better outcomes for patients.

4. Financial planning: Accurate predictions of the length of stay can also help healthcare facilities plan their finances and make more informed decisions about resource allocation and budgeting.

Length of stay prediction is typically performed using machine learning algorithms that consider various factors, such as patient demographics, medical history, and information about the patient's condition and treatment, to make predictions about the length of time that a patient is expected to spend in the facility. The state-of-the-art methods for the length

of stay prediction tasks in healthcare have advanced significantly in recent years with the increasing availability of EHR data and the development of sophisticated machine learning algorithms. Traditional statistical models, such as linear regression and time series analysis, are still widely used for length-of-stay prediction. These models are relatively simple to implement and interpret, and they can provide good results when the data is well-behaved and the relationships between variables are linear. For the task of predicting length of stay, machine learning algorithms including decision trees, gradient boosting and random forest are also gaining popularity.Additionally, deep learning models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are being used more frequently for length-of-stay prediction in recent years. Tasks that require processing sequential data are particularly well-suited for these approaches, such as time series data in EHRs, and they can identify intricate patterns in the data that conventional statistical methods would miss.

### 2.1.2 Mortality Prediction

Mortality prediction is a task in healthcare where the goal is to predict the likelihood of a patient dying within a given time period. Predicting mortality can help healthcare facilities monitor and improve the quality of care that patients receive, as earlier detection of declining health can lead to earlier interventions and potentially better outcomes. Also, it can assist healthcare providers to ensure that patients receive the appropriate level of care while they are in the facility, and to manage the resources effectively. Mortality prediction is typically performed using machine learning algorithms that consider various factors, such as patient demographics, medical history, and information about the patient's condition and treatment, to make predictions about the likelihood of a patient dying. Deep learning models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are being used more frequently for mortality prediction in recent years. Additionally, Graph Neural Networks (GNNs) are becoming increasingly popular for mortality prediction in recent years. Tasks that require analyzing graph-structured data are particularly well suited for these models, such as patient co-morbidities and medical history, and they can capture complex relationships in the data.

## 2.2 Data Storage

Choosing the right data storage is crucial for machine learning algorithms because it has a direct impact on the performance and efficiency of the model. A poorly optimized data storage can lead to slow processing times, which can make training and inference computationally expensive. Additionally, if the storage is not scalable, it may become a bottleneck as the size of the data increases. In such scenarios, we are most often left with several options - relational databases, NoSQL databases, graph databases, distributed file systems, object storage, in-memory databases, and columnar databases.

While relational databases are widely used due to their high query performance and scalability, it performs best when it has to deal with multi-row transactions (see Table 2.1). Multi-row transactions in relational databases refer to a group of database operations that are executed together as a single unit of work. This means that either all of the operations in the group are executed successfully or none of them are executed at all, ensuring the integrity of the data in the database. Electronic health records demand more dynamic methods to represent the data within the database due to the unstructured format it follows. Dynamic methods refer to techniques that allow for the creation of flexible and adaptive database structures that can change in response to changing requirements. In an EHR system, data is constantly changing as new information is added, updated, or deleted. Dynamic methods in relational databases help to manage these changes and ensure that the database structure can adapt to new requirements. In comparison to relational databases, non-relational databases have data models that include document-based, key-value, graph-based, and columnar. They can be structured or unstructured. Structured databases use a predefined data structure, while unstructured databases allow for more flexible data storage. Non-relational databases such as NoSQL databases like MongoDB and Cassandra can handle large amounts of unstructured data and can scale horizontally, which means adding more nodes to a database cluster in order to handle the increasing amount of data. This is a great advantage when dealing with the high volume of data generated by medical systems.

| | **Relational** | **Non-relational (Graph)** |
|---|---|---|
| **Query Language** | SQL | cypher, GQL |
| **Scalability** | vertical | horizontal |
| **Data Structure** | table-based | document, key-value, graph etc |
| **Optimal for** | multi-row transactions | unstructured data e.g. JSON |

Table 2.1: Relational vs non-relational database

The choice between a relational and non-relational database for a medical application will depend on the specific requirements of the application, such as :

1. Need to handle large amounts of unstructured data
2. Ability to scale horizontally
3. Need for precise querying and data integrity

In many cases, it may be beneficial to use a combination of both types of databases to take advantage of the strengths of each, such as the ability of relational databases to enforce data consistency and the ability of non-relational databases to handle large amounts of unstructured data. For example, one might use a relational database to store structured data, such as patient demographics, while using a non-relational database to store unstructured data, such as images or diagnostic reports. This approach can help to improve the performance and scalability of the database system.

Lastly, we have a graph database which is a type of non-relational database that is used to store and manage data in the form of nodes and edges. They are preferred over traditional relational databases (SQL databases) or non-relational databases (NoSQL databases) in certain use cases for several reasons:

1. Flexible Data Modeling: Graph databases allow for flexible data modeling, as data can be modeled as a graph and stored as nodes and relationships, which can be easily modified as data evolves over time.
2. Fast Data Retrieval: Graph databases are optimized for fast data retrieval, as they allow for quick traversal of relationships between nodes, which is a common operation in graph-based data.

3. Efficient Queries: Graph databases are efficient at handling complex queries and relationships, as they can quickly traverse the graph to find and retrieve relevant data.

4. Scalability: Graph databases can be easily scaled to handle large amounts of data and processing demands, as they can be distributed across multiple nodes in a cluster.

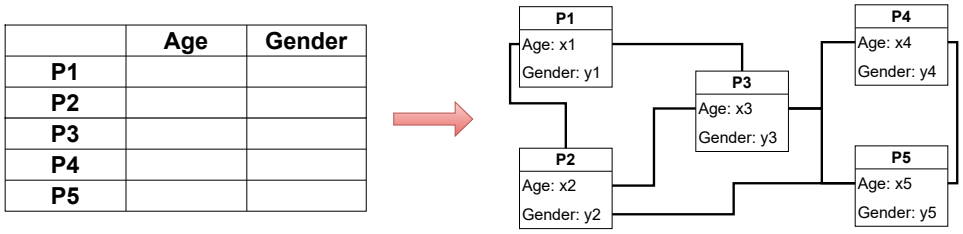


Figure 2.1: Transformation of tabular data to graph data. Rows of patients have been transformed into nodes with node attributes. The edges between nodes can have edge weights which can be the distance/similarity between the rows as per the values of the variables.

In figure 2.1, we demonstrate how tabular data was transformed into the graphical data structure for this research. Every patient row, with values for multiple variables, can be represented as nodes with edge values between them representing relationships between patients in the graph.

## 2.3 Graphs

A graph is a type of non-Euclidean data structure made up of nodes (also known as vertices) and edges between them. A graph can store features or attributes as properties associated with nodes, edges, or both. These properties can be stored as key-value pairs or in more complex data structures, such as arrays or nested objects. Adjacency matrices can be used to model graphs. A two-dimensional array called an adjacency matrix contains rows and columns that represent nodes and values that denote the existence or absence of edges connecting those nodes. Figure 2.2 shows a simple graph having 3 nodes and the lines connecting them as edges. The representation of the graph is shown as an adjacency matrix alongside, where the connection between them is written with a 1 and 0 for no connection. Graphs can also be represented as adjacency lists, where each node is associated with

a list of its neighbouring nodes. Another common representation used in PyG is a feature matrix. It is a 2D tensor that represents the node features of a graph. Each row in the tensor corresponds to a node in the graph, and each column represents a different feature or attribute of the node. The shape of the feature matrix is in the form [num_nodes, num_features]. And to represent the edges of the graph 'edge_index' is used. It is a 2D tensor of shape (2, num_edges), where each column represents an edge and consists of the indices of the two nodes that form the edge. And finally, we represent the edge feature matrix as [num_edges, num_edge_features].

Graphs have become an integral part of our lives, as we use them every day to find the shortest possible route to our destination [31] or receive recommendations from nearby restaurants [19]. Additionally, the common use of graphs comes while using social network profiles to get friend suggestions on Facebook and Twitter. [7]

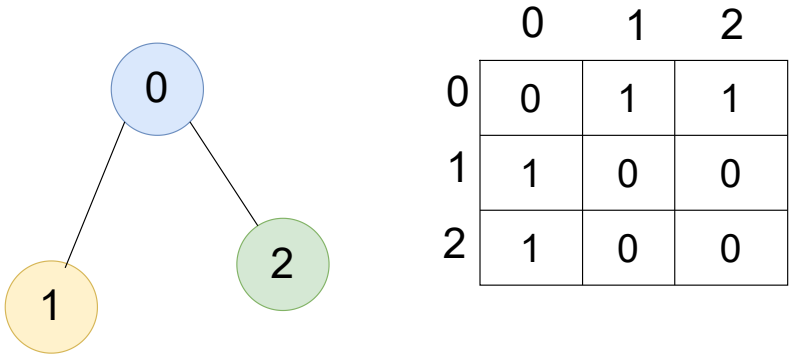


Figure 2.2: Graph and adjacency matrix

### 2.3.1 Types of Graph

We will explain the different types of graphs available with the help of a diagram given below. The first one is an undirected graph in figure 2.3(a). They do not have any direction. An undirected graph is symmetric: if there is an edge connecting node A to node B, then there is also an edge connecting node B to node A. Undirected graphs are often used to model symmetric relationships between objects or entities. For example, an undirected graph can be used to model a social network, where each vertex represents a person and each edge represents a friendship between two people.

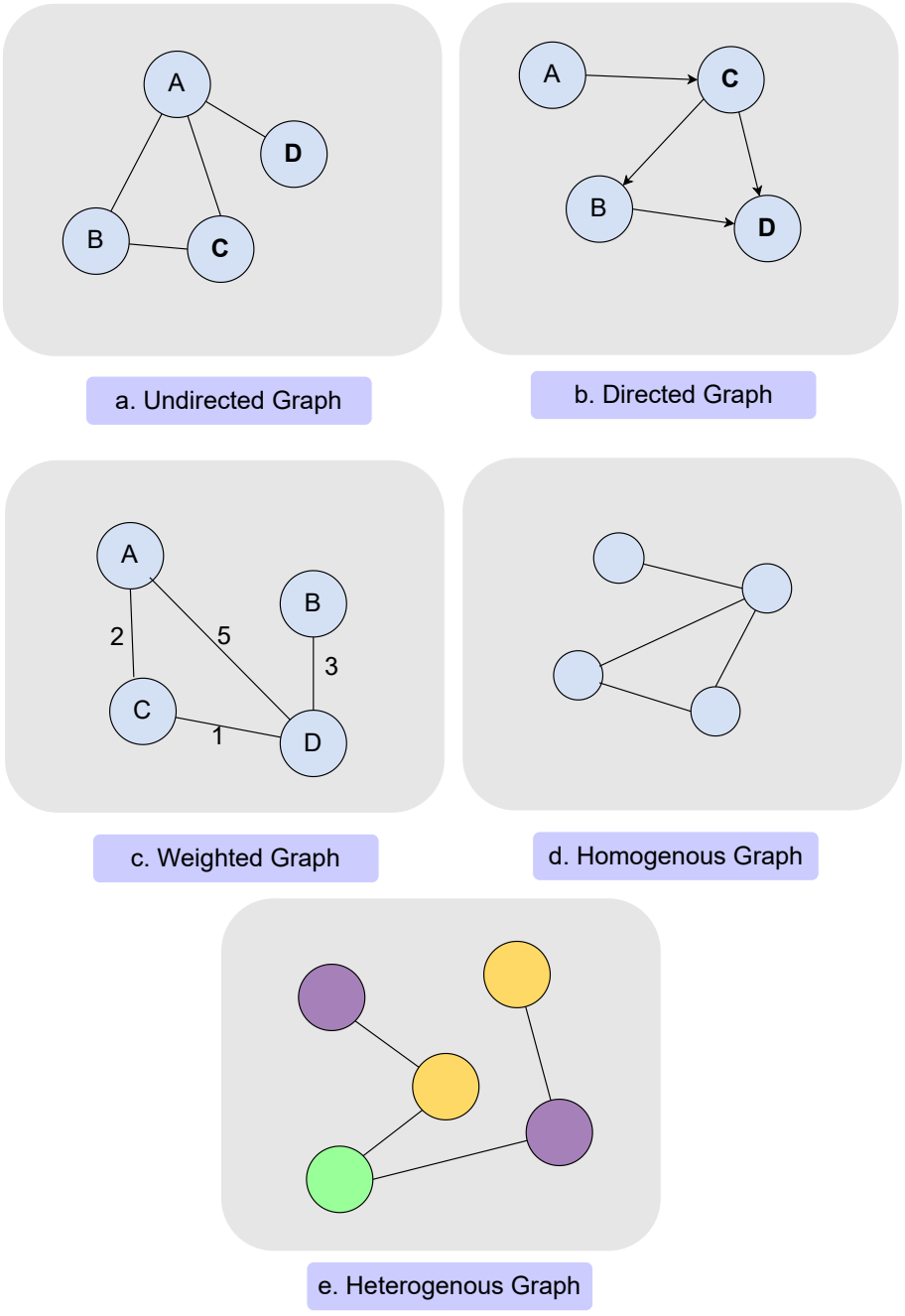


Figure 2.3: Types of Graph

The second one is a directed graph in figure 2.3(b), also known as a digraph, which has a specific direction at all the edges of the graph. Node A is connected to Node C in the forward direction only but the opposite is not

true. Directed graphs are often used to model asymmetric relationships between objects or entities. For example, a directed graph can be used to model a transportation network, where each node represents a city and each directed edge represents a one-way road between two cities.

One key difference between directed and undirected graphs is that directed graphs can have cycles that loop back to a node after following a series of edges. In contrast, undirected graphs are acyclic, meaning that there are no cycles. Directed graphs are often used to model processes, flows, or causal relationships, while undirected graphs are often used to model connections or dependencies between objects or entities.

Figure 2.3(c) is a weighted graph which is a type of graph in which each edge has an associated numerical value, called a weight. The weight of an edge can represent some kind of cost or distance associated with traversing that edge. Weighted graphs are often used to model real-world scenarios where the cost of moving from one node to another is not the same for all edges. In the real world, a weight on an edge might represent the distance in a transportation network or the bandwidth in a communication network.

Weighted graphs can be directed or undirected, and the weights can be positive or negative, depending on the modeled problem. Weighted graphs in comparison to unweighted graphs have the benefit of modeling quantitative relationships and can represent priorities and preferences. For example, weighted graphs can have weights on their edges in a recommendation system, allowing for more personalized and relevant recommendations.

Figure 2.3(d) shows a homogeneous graph. All nodes and edges in a homogeneous graph are of the same kind. A practical example of a homogeneous graph could be a medication-prescription graph where each node represents a medication and each edge represents a prescription of that medication to a patient. In this graph, all nodes would have the same type (medication), and all edges would have the same type (prescription). The nodes would be connected by edges only if a prescription for that medication has been issued to a patient, which creates a homogeneous graph where all edges have the same type and all nodes have the same type. This graph can be useful for analyzing medication usage patterns and identifying potential adverse drug interactions or other medication-related issues for a particular patient or group of patients. In contrast, a heterogeneous

graph has nodes of different types, represented as different color circles in figure 2.3(e). As an example, Liu et al. [28] propose a heterogeneous graph created from electronic health records. The graph is connected by different node types called patient, visit, diagnosis, and medication. Each patient can have multiple visits to a hospital and on every visit to the hospital he may be diagnosed with various conditions and can be prescribed multiple medications based on his conditions. Every node type or edge type can have a different set of attributes. Since one patient can have multiple visits to a hospital, a heterogenous graph can capture the entire journey of a patient. For example, HAS_VISIT, IS_DIAGNOSED, and IS_MEDICATED could be some of the edges which represent the different types of relationships or interactions between the nodes. While homogeneous graphs can be useful for representing simple relationships between entities in an EHR, heterogeneous graphs provide a greater representation of complex relationships, improved analysis, and insights, increased interoperability, and may provide better patient outcomes.

### 2.3.2 Tasks of Graph

Graphs are flexible data structures that can be used to represent a wide variety of relationships [43]. To extract insights or meaningful information from the graph, we can perform different kinds of tasks(e.g. node classification, link prediction, and community detection).

1. **Node classification** is a machine-learning task that involves classifying nodes in a graph based on their features and the features of their neighboring nodes. One popular approach to node classification is to use machine learning algorithms, such as neural networks, to learn a function that maps the features and topology of the graph to the node labels. Classifying paper categories in a citation network is one example of node classification [50].

2. **Edge(also referred to as Link) prediction** is a machine learning task that involves predicting which nodes in a graph are likely to form new connections(edges) in the future based on the existing node and edge properties. Item recommendation is a type of link prediction problem where the goal is to recommend items to users based on their historical preferences or interactions [22].

3. **Community detection**, also known as graph clustering, is the task of identifying groups of nodes in a graph that are more densely connected than the rest of the graph. These groups of nodes, known as communities or clusters, are often assumed to have similar characteristics or to be involved in similar activities. There are several approaches that can be used to perform community detection, including modularity-based methods that maximize a quality function called modularity, which measures the density of edges within a community compared to the density of edges between communities. Deep learning-based methods use neural networks to learn representations of nodes and edges, and then use these representations to identify communities. As an example, community detection can identify groups of web pages that are more densely linked to each other than to the rest of the network [30].

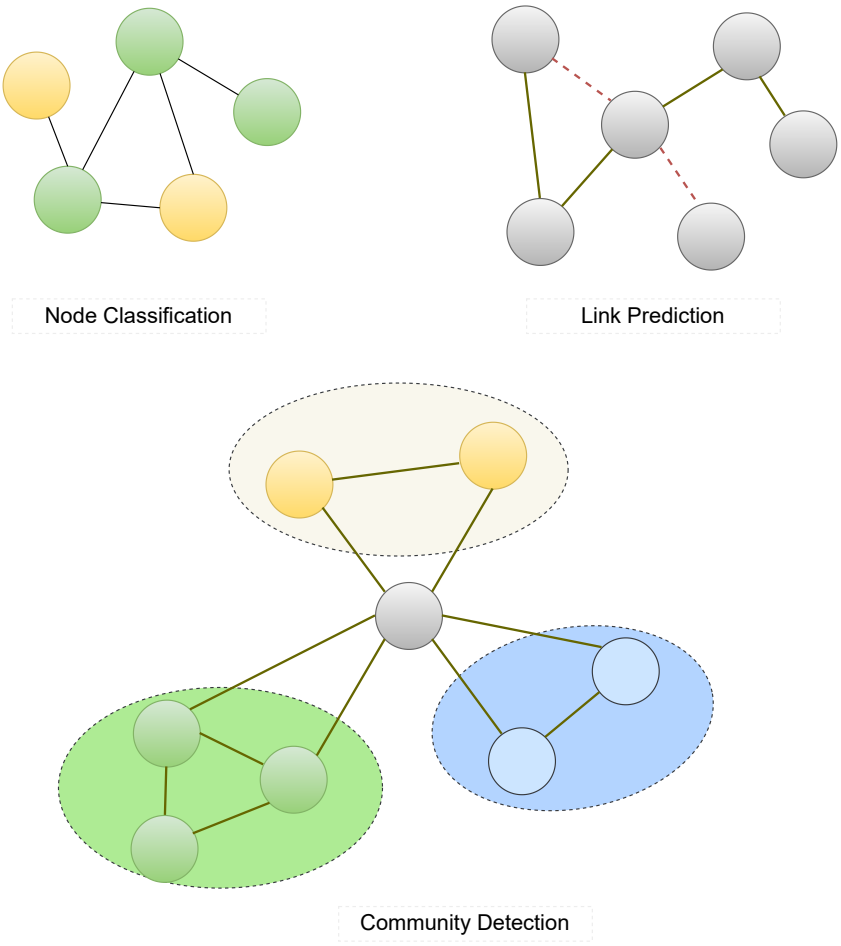


Figure 2.4: Tasks on graph

Figure 2.4 shows the different tasks on the graph which have been briefly explained above.

## 2.4 Machine Learning: Artificial Neural Networks

Machine learning is a subfield of artificial intelligence that is concerned with the development of algorithms and statistical models that enable computers to learn from data. It has enormous applications in several fields, especially in the field of medical science where it is commonly used for diagnostic imaging, drug discovery, and analyzing large amounts of data such as EHR. Machine learning algorithms can be broadly classified into three main categories:

- **Unsupervised Learning:** Algorithms process unlabelled data to learn patterns and groupings. It is used in a number of applications such as anomaly detection, clustering, and dimensionality reduction.
- **Reinforcement Learning:** Algorithms interact with an environment and are trained with punishment or reward. Autonomous driving is an example of reinforcement learning.
- **Supervised Learning:** Algorithms process labeled data by mapping input data to the corresponding target variable. Supervised learning is used in a wide range of applications, such as image classification, speech recognition, etc.

Neural networks are a type of supervised machine learning algorithms that are modeled after the structure and function of the human brain and it is made up of interconnected artificial neurons that process information and learn from it. The strength of the connections between neurons, known as weights along with the bias, is adjusted during the learning process to optimize the performance of the network. There are several forms of neural networks:

- Feed forward neural networks.
- Recurrent neural networks.
- Convolutional neural networks.
- Generative adversarial networks.

### 2.4.1 Neural Networks

Neural Networks consist of layers of interconnected "neurons", which process and transmit information. It is also known as a multi-layer perceptron(MLP) that consists of one or more layers of artificial neurons, where each layer is fully connected to the next layer.

Neural networks have several use cases across many industries, such as:

- Medical image classification for diagnosis [52].
- Using historical financial data to make financial forecasts [20].

- Forecasting of electrical load and energy demand. [39]
- Identification of a chemical compound. [2]

The basic structure of a simple neural network has been explained in the following:

1. Input Layer: The artificial neural network's input layer is the first layer that receives the input data. They consist of one or more neurons. The number of neurons in the input layer is determined by the size of the input data, where each neuron receives one feature from the input data. For example, if the input data consists of a 28x28 image, the input layer of the neural network will have 784 neurons, where each neuron corresponds to a pixel in the image.
2. Hidden Layers: Each neuron in a hidden layer receives inputs from the neurons in the previous layer, and it performs a computation on those inputs using a set of weights and biases. By adding multiple hidden layers, the network can learn more complex and abstract features from the input data, allowing it to model more complex relationships between the inputs and outputs.
3. Output layer: The output layer in a neural network is the final layer of the network that produces the network's output. This layer consists of one or more neurons, where each neuron produces a single output value or a vector of output values depending on the type of problem being solved. For example, in a binary classification problem, the output layer will have a single neuron that produces a value between 0 and 1, representing the probability of the input belonging to one of the two classes. In a multi-class classification problem, the output layer will have one neuron for each class, with each neuron producing a probability value for its corresponding class. In a regression problem, the output layer will have a single neuron that produces a continuous output value.

Forward propagation, backward propagation, and loss are important concepts in neural network training. Forward propagation refers to the process of taking input to a neural network and passing it through the network to

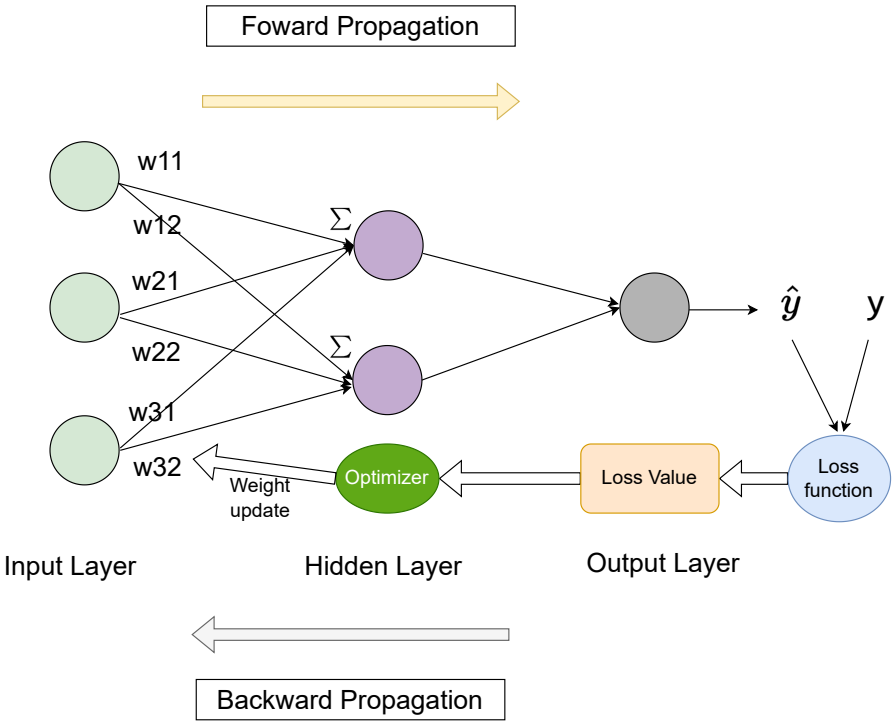


Figure 2.5: Forward and backward propagation

produce an output. During this process, the input is transformed through a series of mathematical operations and activations in the network's layers, eventually resulting in an output that can be used for prediction or classification. Backward propagation (also known as backpropagation), is the process of computing the gradients of the loss function with respect to the weights and biases of the neural network. These gradients can be used to update the parameters of the network during training using an optimization algorithm like stochastic gradient descent. The loss function is a mathematical function that measures the difference between the predicted output of a neural network and the actual output. The goal of training a neural network is to minimize this loss function by adjusting the weights and biases of the network through backpropagation. In every iteration, we compute the loss value and update the weights and biases during the backpropagation stage. The training steps of a feed-forward neural network as shown in figure 2.5 is an iterative process when the weights are

computed both in the forward and backward propagation stage until the loss is brought minimum.

Each layer is made up by the stacking of multiple neurons. The layers communicate with each other by weights and biases of the layers which are represented by arrows in the figure. The level of importance of each input is managed by the weights while the bias is used to offset the results. Initially, non-zero random values are assigned as weights and biases in a step known as parameter initialization. Activation functions are used in artificial neural networks to introduce non-linearity into the output. This non-linearity allows the network to learn complex patterns and model non-linear relationships between inputs and outputs. Without activation functions, a neural network would be just a linear model and would not be able to learn these complex patterns. There are several activation functions as shown in figure 2.6. Activation functions like Sigmoid, Tanh, Relu, and LeakyRelu are in figure 2.6 and are widely used in deep learning models. The Sigmoid function produces output in the range of (0,1) and Relu in (0, infinity), and Tanh in (-1,1).

This generates a predicted value as output in the output layer. The neural network calculates the loss function by comparing the ground truth values with the predicted values. The loss score that is computed is also called the error of the model. Based on the type of machine learning tasks(classification and regression), we have to use different loss functions. In the classification task, the model tries to predict the probabilities of all target classes(categorical variables). Whereas, in the regression task, the model tries to predict continuous values based on a number of features, that have been fed into. The classification loss functions are Binary cross-entropy loss (also known as log loss), Categorical cross-entropy loss, Sparse categorical cross-entropy loss, Hinge loss, and Focal loss. The regression loss functions are: Mean squared error (MSE) loss, Mean absolute error (MAE) loss, Huber loss, and Smooth L1 loss.

**What is Backward propagation?**

Backpropagation is part of the training steps of artificial neural networks. It is used to adjust the weights of the neural network in order to minimize the loss between the predicted output and the true output. At each layer, the error is used to calculate the gradient of the weights with respect to

Table 2.2: Advantages and disadvantages of activation functions

| | Advantages | Disadvantages |
|---|---|---|
| Sigmoid function | • Has a simple and smooth curve, which is easy to compute and differentiate.<br>• It maps the output to a probability-like value, which can be useful for binary classification problems. | • The sigmoid function is prone to saturation, which means that the gradients become very small as the input becomes very large or very small.<br>• It is not zero-centered, which can make it difficult to learn where the input data has a large dynamic range. |
| ReLU (Rectified Linear Unit) function | • It avoids the saturation problem of the sigmoid function.<br>• It is simple and efficient formula, which makes it computationally cheap and easy to implement. | • The ReLU function suffers from the "dying ReLU" problem, which occurs when the input to the function is negative, and the gradient becomes zero.<br>• The ReLU function is not differentiable at zero. |
| Tanh function | • The Tanh function is zero-centered.<br>• It is similar to the sigmoid function but with a steeper gradient, | • The Tanh function still suffers from the saturation problem of the sigmoid function, which can lead to slow convergence or vanishing gradients in deep neural networks. |
| Leaky ReLU function | • The Leaky ReLU function addresses the "dying ReLU" problem by introducing a small slope for negative input values. | • The Leaky ReLU function can introduce some noise in the output. |

the error. The gradient is then used to update the weights in the opposite direction, in order to minimize the error. This process is repeated for many iterations(epoch) until the error is sufficiently small. After completion of the first iteration, usually, the difference between the predicted value and the ground truth value is large, because initially random weights and biases had been assigned to the neural network. So, the process of updating the

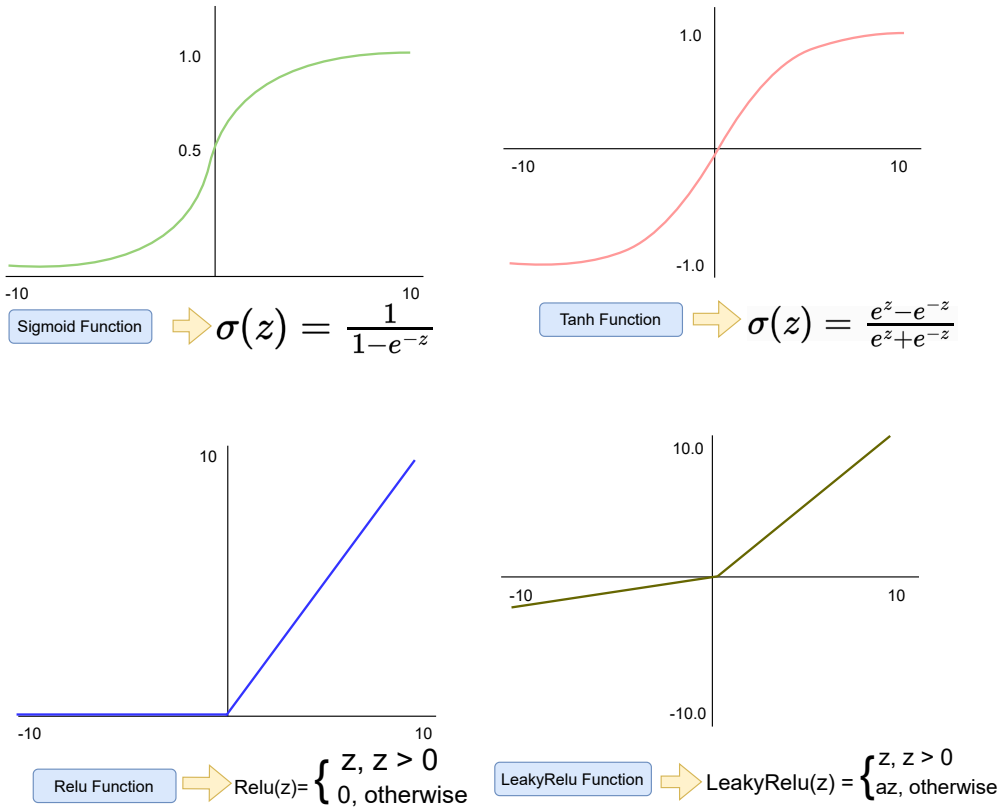


Figure 2.6: Activation Functions

weights is carried out by an optimization algorithm(e.g. gradient descent) that implements the back-propagation algorithm.

The goal of optimization algorithms is to find the global minima or regions with the lowest values of the loss function. However, finding the global minimum of a complex loss function is significantly difficult for an optimization algorithm. In figure 2.7, a global and local minimum is shown. A global minimum is the lowest point on the graph, representing the set of model parameters that result in the smallest loss value across the entire search space. This is the optimal set of parameters that we want to find. On the other hand, a local minimum is a point on the graph that is lower than all the neighboring points but is not the global minimum. Local minima can be problematic for optimization, as they can prevent the algorithm from finding the optimal set of parameters, and can lead to suboptimal performance of the model.

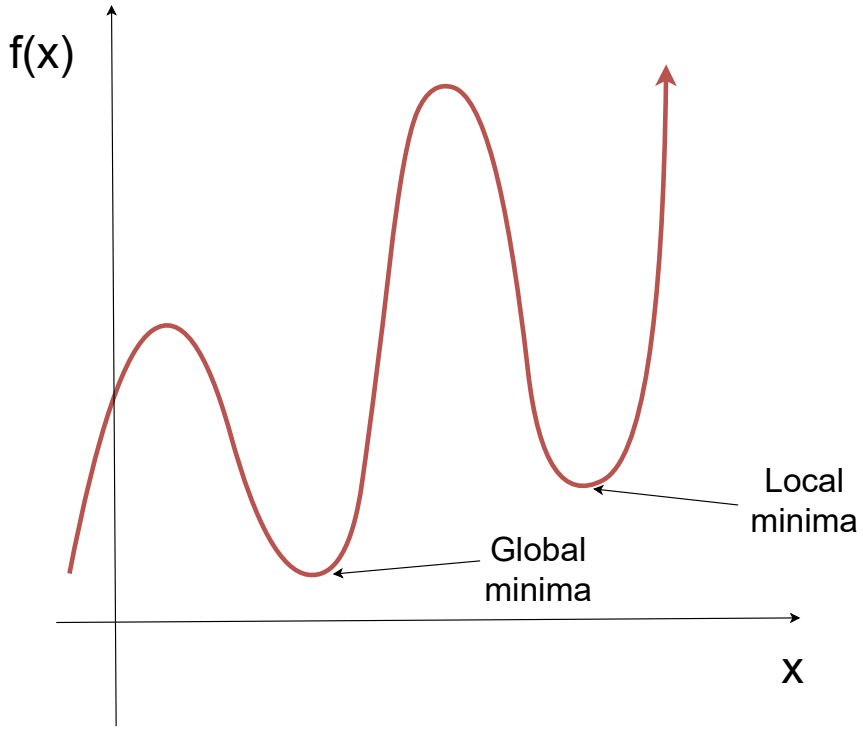


Figure 2.7: Local and Global minima

**Batch Size and epochs** Training a neural network using all of the training data in a single cycle is known as full batch gradient descent or batch gradient descent. While this approach has the benefit of providing the most accurate parameter updates, it can be computationally expensive for large datasets. So we often use mini-batch gradient descent or stochastic gradient descent. In mini-batch gradient descent, the training data is divided into small subsets or "mini-batches". The model is then trained on each mini-batch in turn, with the parameters updated based on the average gradient across the mini-batch. This approach provides a balance between the accuracy of full batch gradient descent and the efficiency of stochastic gradient descent. In stochastic gradient descent, the model is trained on a single training instance at a time, with the parameters updated based on the gradient of that instance. This approach can be much faster than full batch and mini-batch gradient descent, especially on large datasets, but can also be less stable since the parameter updates are based on a single instance. One complete iteration over all batches is called an epoch. Let's consider a dataset of 1000 training samples as an example when we set the

batch size to 10 and the number of epochs to 20. In this instance, we will partition our dataset into 100 (1000/10) batches. Thereby each batch(10 training samples) is passed to the neural network until all batches were passed. The model parameters (i.e., their weights and biases) are optimized after each batch (i.e., 100 times for each epoch). Since there are 20 epochs, the optimizer iterates 20 times over the complete dataset(i.e. 20 times overall 100 epochs).

**Train, validation, and test split** refers to the process of dividing a dataset into three subsets: training, validation, and testing.

- The training set is used to train a machine learning model, and typically consists of the largest portion of the data. [17]
- The validation set is used to tune the hyperparameters (e.g.learning rate and the number of hidden layers) of the model, and to measure its performance during the training process. [6]
- The test set is used to evaluate the final performance of the model and is typically held out from the training process to provide an unbiased assessment of the model's performance. [37]

The division of the dataset into training, validation, and test sets is crucial to ensure that the model is not overfitting or underfitting the data.

**Overfitting and underfitting**

Overfitting is a common problem in machine learning where a model becomes too complex and begins to fit the training data too closely. When a model is overfitting, it has learned the noise and fluctuations in the training data rather than the underlying patterns and relationships. As a result, the model performs very well on the training data but has poor performance on new data. For example, consider a dataset that consists of the number of hours of study per day and the corresponding exam scores for a group of students. If the relationship between study hours and exam scores is actually linear, a simple linear regression model would be appropriate. However, if a more complex model, such as a high-degree polynomial regression, is used, the model may capture the noise and fluctuations in the training data and overfit the data. As a result, the model will perform poorly when used to predict exam scores for new students who were not

part of the training data. To prevent overfitting in this case, techniques such as regularization, early stopping, and cross-validation can be used. For example, regularization can be used to add a penalty term to the objective function that the model is optimizing, which can discourage overfitting by limiting the complexity of the model. Cross-validation can be used to ensure that the model is not overfitting to a specific subset of the data, and early stopping can be used to stop the training of the model before it has converged to prevent it from continuing to fit the noise in the training data.

Underfitting is a phenomenon in machine learning where a model is not complex enough to capture the underlying patterns and relationships in the data, resulting in poor performance on both the training data and new, unseen data. For example, if the true relationship between study hours and exam scores is a quadratic function, a linear regression model would be too simple to capture this relationship. As a result, the model would underfit the data and perform poorly on both the training data and new, unseen data. To prevent underfitting in this case, a more complex model, such as a polynomial regression model with a higher degree, could be used. Additionally, feature engineering could be used to create additional features that capture the non-linear relationship between study hours and exam scores.

**How message passing works?**

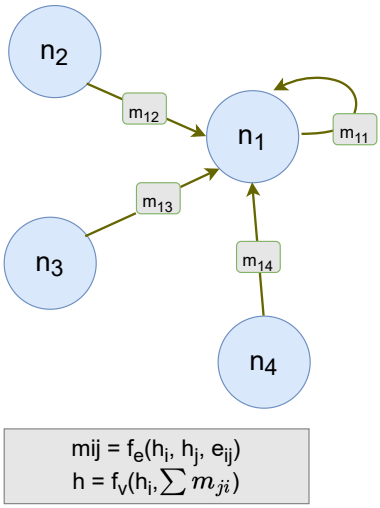


Figure 2.8: Message Passing Layer

Permutation-equivariant layers that convert a graph into a new version of the same graph are known as message-passing layers. A permutation equivariant layer can process the same graph regardless of the order in which the nodes are listed. They can be described formally as "message-passing neural networks" (MPNNs). The message(i.e., information) is usually computed as a function of the node features and the edge connecting the nodes. The information contained in the messages is then aggregated at each node to produce a new representation of the node. This process is repeated several times to produce a final representation of the graph that captures the underlying graph structure.

To explain figure 2.8 for a message-passing network, let us assume we have a graph with n nodes and e edges. The features of each node are contained within the node itself, in the form of a vector or a tensor. The goal of message passing is to allow the GNN to capture the structural information of the graph and to use that information to make predictions. Each node in the graph sends and receives messages to and from its neighboring nodes. The messages contain information about the features of the sender and the relationship between the sender and the receiver and are used to update the features of the receiving node. Messages are passed between nodes i and j and represented by mij, and is calculated by the help of a function fe shown in the first equation from equation 2.1. the edge features between the two nodes are represented as eij. The fe in the formula represents a differentiable function which is usually a small ML) that takes into account both the features of nodes and edges if they have edge features.

$$\begin{aligned} mij &= fe(hi, hj, eij) \\ h &= fv(hi, (\sum m_{ji})) \end{aligned} \tag{2.1}$$

A permutation-invariant function, such as summation, mean or max shown in the second formula of equation 2.1, is then used to aggregate all messages that arrive at each node. Sometimes mean or max is also used in place of the summation. There exists another MLP as fv in the second formula, which joins the aggregated representations and gives an updated feature vector of the node n1 in figure 2.8. An MPNN is considered a basic foundation for GNN[45].

When updating their representation, graph nodes in an MPNN get data from their close neighbors. With n MPNN layers stacked, one node can interact with nodes that are up to n "hops" distant. If the number of MPNN layers is equal to the diameter of the graph(i.e. maximum distance between any two vertices in the graph) would need to be stacked so that each node would capture information from all other nodes in the graph. However, oversmoothing and oversquashing may occur when stacking several MPNN layers. Oversmooting can happen when the graph has too many layers (i.e, propagates information too many times through the graph). Therefore, leading to node representations become too similar and lose their local properties. Additionally, using a propagation function that is too simple or homogeneous can also lead to oversmoothing [51], as it may not effectively capture the complex relationships between nodes in the graph. Oversmoothing can be reduced by reducing the number of GNNs.

Another problem that can occur in GNNs is oversquashing. MPNNs often struggle to handle tasks that rely on long-range connections between nodes in a graph. This is especially problematic when the graph has an exponentially increasing number of distant neighboring nodes, which means that a node's "receptive field" grows rapidly as the radius of the neighborhood increases. In order to deal with these non-adjacent nodes, their messages must be compressed into fixed-size vectors, which can result in an oversquashing of information. For example, using graph pooling methods(e.g., attention-based pooling, adaptive pooling, dimensionality reduction) the size of the graph can be reduced by aggregating information from multiple nodes into a single, coarser node. This can help the GNN process information more efficiently while preserving important structural features of the graph.

### 2.4.2 Graph Neural Networks(GNN)

GNNs are specialized neural networks designed to operate on graph-structured data. They are effective at tackling a variety of graph prediction challenges at the graph, node(e.g. node classification), and edge levels(e.g. link classification). In a GNN, each node is associated with a feature vector, and the edges between nodes represent the relationships or connections between them. GNNs operate by iteratively updating the node features

based on their neighboring nodes and the edges that connect them. The use of pairwise message passing is a critical component in the design of GNNs, allowing graph nodes to iteratively update their representations by exchanging information with their neighbors.

Applications for GNNs include recommending friends in a social network [49]. Also, there are several open-source libraries that implement graph neural networks, including TensorFlow GNN [48], PyTorch Geometric [16].

#### Different types of GNN

There are several forms of MPNN that have been proposed namely Graph Convolutional Network(GCN), Graph attention network(GAT)

| Model | Key features |
|---|---|
| **GCN (Graph Convolutional Network)** | Uses a simple mean or sum aggregation of neighborhood features for node representation learning. |
| **GAT (Graph Attention Network)** | Introduces attention mechanism to weight the contribution of different neighbors when aggregating their features. |
| **GraphSAGE** | Samples a fixed-size neighborhood for each node and aggregates the feature information of these neighbors. |
| **GIN (Graph Isomorphism Network)** | Uses a multi-layer neural network with a learnable aggregation function to update node representations. |

Table 2.3: Comparison of different types of GNN

Here are some advantages and disadvantages of each model:

**Graph Convolutional Networks (GCN)**

Advantages:

- Captures local neighborhood information well
- Simple and easy to implement
- Can be efficient for small graphs

Disadvantages:

- Can struggle with modeling long-range dependencies
- Not very scalable for large graphs

**Graph Attention Networks (GAT)** Advantages:

- Can model fine-grained attention mechanisms
- Can handle long-range dependencies

Disadvantages:

- Computationally expensive
- Can overfit on small graphs Attention mechanisms can be difficult to interpret

**GraphSage**

Advantages:

- Can scale to large graphs Efficient and computationally inexpensive
- Can capture local neighborhood information at different scales

Disadvantages:

- Less expressive than GCN and GAT. May not perform as well on tasks that require modeling long-range dependencies

### 2.4.3 Sequential data handling by Neural Networks

#### Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a type of neural network that is commonly used for sequential data processing tasks. RNNs are designed to operate on sequences of input data, such as time-series data or natural language text, by maintaining a hidden state that encodes information from previous inputs..Rumelhart et al.[40] first introduced vanilla RNN in their paper. To explain how RNN works, it is important to know the difference

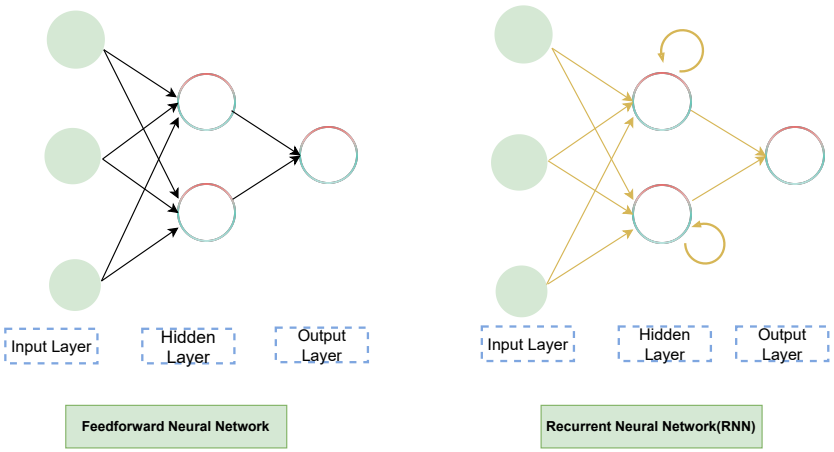


Figure 2.9: Feed-forward neural network vs RNN

between RNNs and feed-forward neural networks first. The following figure would help us to explain the difference.

In the feed-forward neural network, as shown in figure 2.9, information only travels in one direction, from the input layer to the output layer via the hidden layer. These networks take in a fixed number of inputs and use the weights of the connections between neurons to transform the inputs into an output. They are poor at predicting the next outcome of the sequence data because they cannot hold the memory of the previous input that they receive. In contrast, the information passed through an RNN maintains a loop. When it gives an output, it takes into account the current input and what it has learnt from the previous inputs. To give an example, consider feeding the word 'report' as an input to a feed-forward network, one by one. The moment the character reaches 'o', the previous characters are not remembered and it cannot predict that the next character would be 'r'. However, an RNN is fully capable enough to keep those characters in it memory, by copying the generated output and sending it back to the network as a loop. In this way, RNN learns its input and keeps it in the memory. The internal hidden layer allows RNNs to process and make predictions about sequences of variable length. In the feed-forward neural network, as shown in figure 2.9, information only travels in one direction, from the input layer to the output layer via the hidden layer. These networks take in a fixed number of inputs and use the

weights of the connections between neurons to transform the inputs into an output. They are poor at predicting the next outcome of the sequence data because they cannot hold the memory of the previous input that they receive. In contrast, the information passed through an RNN maintains a loop. When it gives an output, it takes into account the current input and what it has learnt from the previous inputs. To give an example, consider feeding the letters of the word 'report' as an input to a feed-forward network, one by one. The moment the character reaches 'o', the previous characters are not remembered and it cannot predict that the next character would be 'r'. However, an RNN is fully capable enough to keep those characters in its memory, by copying the generated output and sending it back to the network as a loop. The internal hidden layer allows RNNs to process and make predictions about sequences of variable length.

#### Long Short-term Memory Networks: LSTM

Long Short-term Memory(LSTM) network is able to retain information over long periods of time because of the memory cell, which acts like a conveyor belt, allowing information to flow through it and be stored over time. Standard RNN architecture has a problem known as the vanishing and exploding gradient. The vanishing gradient problem occurs when the gradients used to update the weights during backpropagation become very small as they propagate through the network. As a result, the weights may not be updated effectively, and the model may have difficulty learning long-term dependencies. Conversely, the exploding gradient problem occurs when the gradients become very large as they propagate through the network. LSTMs solve this problem by introducing a memory cell and three gates (input, output and forget gates) which can retain information over long periods of time. By doing this they control the flow of information into and out of the memory cell. The gates are controlled by sigmoid neural network layers, which decide what information to store in the memory cell, what to discard and what to output. The input gate controls the flow of new information into the memory cell, the forget gate controls the flow of information out of the memory cell(i.e., which information to discard), and the output gate controls the flow of information out of the memory cell and into the output.

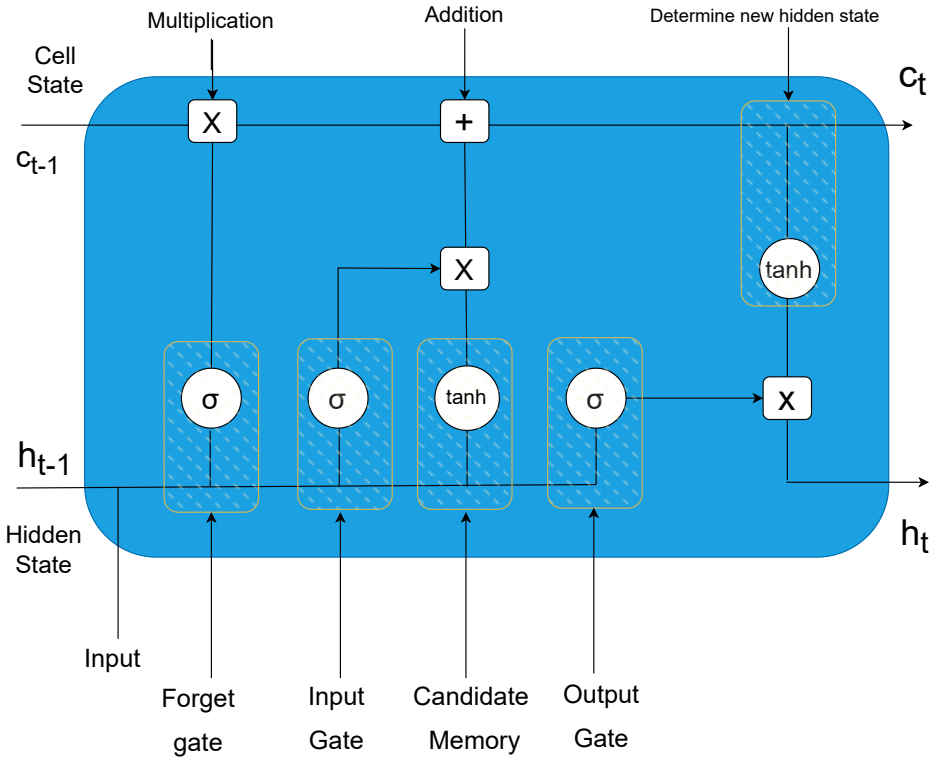


Figure 2.10: LSTM Unit

At each time step, the LSTM takes in an input, x, and the previous hidden state, h(t-1), and computes the current hidden state, h(t), and output, y(t). The computation is done in the following steps(see figure 2.10):

1. Forget gate: The forget gate determines which information in the current memory cell should be discarded. It takes as input the previous hidden state h_t-1 and the current input. A vector created by a multiplying a set of learnable weights and added learnable bias with it. Afterwards this vector is then passed through a sigmoid activation function to produce an output between 0 and 1. This output is then element-wise multiplied by the previous memory cell state value, C_t-1, which allows the network to selectively forget irrelevant or redundant information. Values close to 0 indicating that the corresponding element in the memory cell should be forgotten, and values close to 1 indicating that the corresponding element should be retained.

2. Input gate: the input gate determines which information should be stored in the memory cell. It takes as input the previous hidden state h_t-1 and the current input, and passes them through a sigmoid activation function to produce an output between 0 and 1

3. Candidate memory: The candidate memory cell state value is computed using a tanh activation function. It takes as input the previous hidden state h_t-1 and the current input, and combines them to produce a value between -1 and 1. This value represents the new information that could be added to the memory cell.

4. Output gate: The output gate determines which information in the memory cell should be outputted as the final hidden state of the LSTM network. It takes as input the previous hidden state h_t-1 and the current input, and passes them through a sigmoid activation function to produce an output between 0 and 1. This output is then element-wise multiplied by the current memory cell state value C_t, which allows the network to selectively output the relevant information. The resulting value is passed through a tanh activation function to produce the final hidden state h_t as shown in the figure 2.10.

In this way, LSTMs can selectively read, write, and erase information in the memory cell, which allows them to maintain long-term dependencies in sequential data. However, LSTM networks, like all neural network models, can have several problems, including:

- Overfitting: LSTMs can memorize the training data too well, leading to poor generalization on unseen data.
- Slow training: LSTMs can be computationally expensive to train, especially on large datasets.
- Difficulty in interpret-ability: LSTMs are complex models and it can be difficult to interpret the internal workings of the model, making it challenging to understand why a particular prediction was made.
- Need for a large amount of data: LSTMs require large amount of data to train, which makes it difficult for small datasets to be trained effectively.

**Transformers**

The Transformer architecture was designed to overcome the limitations of LSTMs and RNNs and to provide improved performance on sequential data tasks. A transformer-based encoder-decoder architecture is based on attention layers and was first proposed by Vaswani et Al[44].

The Transformer consists of two main components, the encoder block(see first gray rectangle of figure 2.11) and the decoder block(see second gray rectangle of figure 2.11).

**The Encoder Block**: The encoder processes the input sequence and produces a sequence of hidden representations, known as the encoded sequence. The encoder consists of several identical layers, each containing two sub-layers: a self-attention mechanism and a fully connected feedforward network shown in figure 2.11. Input embeddings are used to represent the input words in a sequence as dense, low-dimensional vectors that can be easily processed by the model. These embeddings are learned through training and are optimized to capture the semantic and syntactic features of the words. Next, positional embeddings are added to the input embeddings to create a representation that encodes both the content of the word and its position in the sequence. And attention mechanism allows the model to focus on different parts of the input sequence when generating each output word.

Self-Attention Mechanism: The self-attention mechanism allows each element in the input sequence to attend to all other elements, weighted by the degree of relevance. The attention mechanism is implemented as a dot-product between a query vector, a key vector, and a value vector for each element in the sequence. The query vectors are used to calculate attention scores for each position in the input sequence, key vectors represent the positions in the sequence being attended to, and value vectors represent the actual information being attended to. The attention weights are then used to compute a weighted sum of the value vectors, which is used as the output for the self-attention mechanism. The "Add and norm" step in the Transformer model combines the output of the attention and feed-forward layers and applies layer normalization.

Feedforward Network: The feedforward network is a simple neural network with a series of fully connected layers. It is used to learn non-linear trans-

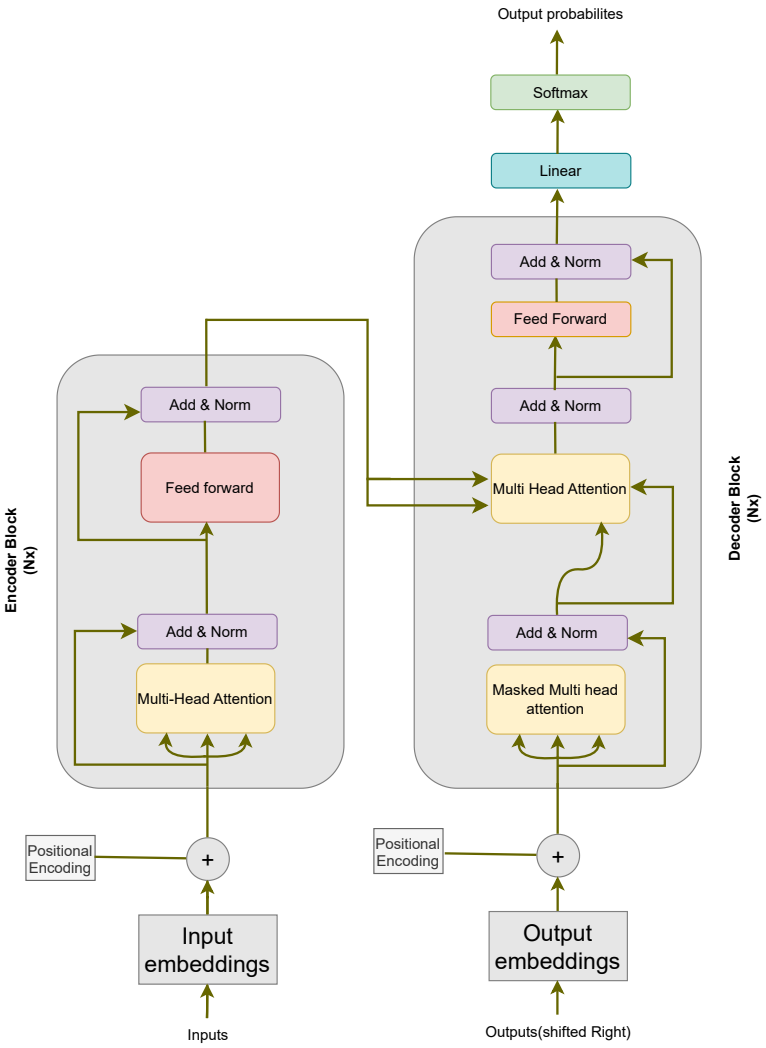


Figure 2.11: Transformer

formations of the input representations and to add non-linearity to the model.

**The Decoder Block**: The decoder processes the encoded sequence and produces the output sequence. The decoder is similar to the encoder, with the addition of an attention mechanism that allows the decoder to attend to the encoded sequence. In the decoder block of the Transformer model, attention is applied over the encoder outputs and the decoder inputs, whereas in the encoder block, attention is applied only over the input sequence. By means of this attention mechanism the decoder take into account the context of the input sequence when generating the output sequence. The decoder also contains a series of identical layers, each containing two sub-layers: a self-attention mechanism, an attention mechanism over the encoded sequence, and a fully connected feedforward network.The attention mechanism is needed in the decoder of the Transformer model to allow the decoder to focus on relevant parts of the input sequence while generating the output, as the decoder's output at each step is influenced by the entire input sequence. In contrast, the encoder only needs to capture the important features of the input sequence and can process the sequence sequentially.

The transformer model have several advantages compared to RNN and LSTM. The self-attention mechanism in the Transformer model allows for parallel processing of input sequences, which can significantly speed up training and inference. In addition, the attention mechanism in the Transformer model allows for visualization of the importance of different parts of the input sequence in generating the output, making the model more interpretable. However, the Transformer model has a large number of parameters, which requires a large amount of training data to avoid overfitting.

## 2.5 Evaluation

Evaluating machine learning models is a crucial step in the development and deployment of machine learning systems. It helps us to determine the performance and robustness of the models. Furthermore, it guides us in making decisions about model selection, optimization, and deployment. In the following, I list several important reasons for evaluating a machine learning model.

- Model Performance: Evaluation allows us to quantify the performance of a model on a specific task and compare it with other models. This helps us determine which model is best suited for the task at hand.
- Model Selection: Evaluation allows us to choose the best model for a specific task based on its performance. This is important because different models may perform better on different types of tasks or datasets.
- Model Optimization: Evaluation helps us determine where a model can be improved and what changes can be made to optimize its performance. This could include adjusting hyperparameters, changing the architecture of the model, or using a different type of model altogether.
- Model Deployment: Evaluation is critical for determining when a model is ready to be deployed in a real-world setting. A model that performs well on a validation set may not perform as well in a real-world setting, so evaluation is necessary to ensure that the model is robust and can generalize well to new data.
- Model Trust: Evaluation helps to build trust in a model by providing transparency into its performance and limitations. This is important because machine learning models are increasingly used to make important decisions, and it's important to ensure that they are trustworthy and accurate.

Accuracy is a popular metric for evaluating machine learning models. However, many real world datasets are imbalanced, i.e., classes differ in their sample size One example where this might occur is in predicting a specific disease(e.y, lung cancer).Only a small fraction of patients have the disease(minority class) while most patients do not have this disease(minority class). In such cases, accuracy can be misleading because a model that simply predicts the majority class for all instances will have a high accuracy, but it will not have any practical value. Therefore, we should also consider other metrics. Other metrics such as precision, recall, F1-score, Area under the ROC curve(AUC), Receiver operating characteristic (ROC) curve and the confusion matrix should also be considered, depending on the specifics

of the task(e.g., using precision for fraud detection, recall for medical diagnosis) and the dataset. Two common metrics used for regression tasks are as follows:

- Mean Squared Error (MSE): MSE measures the average squared difference between the predicted and actual values. It is calculated by taking the average of the squared differences between the predicted and actual values. A lower MSE indicates better performance of the model.

- Mean Absolute Error (MAE): MAE measures the average absolute difference between the predicted and actual values. It is calculated by taking the average of the absolute differences between the predicted and actual values. A lower MAE indicates better performance of the model.

**Area Under the curve**

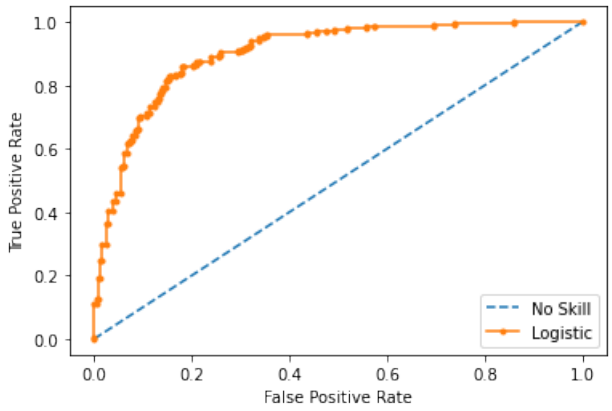


Figure 2.12: Receiver operating characteristic curve

The area under the curve (AUC) is a widely used metric in machine learning that measures the performance of a binary classification model. It is a summary of the model's performance over all possible classification thresholds, expressed as the area under the receiver operating characteristic (ROC) curve (see figure 2.12). AUC measures the model's ability to discriminate between the positive and negative classes, making it a more reliable metric for imbalanced data.

| | | **Prediction outcome** | | |
|---|---|---|---|---|
| | | **p** | **n** | **total** |
| **actualvalue** | **p′** | True Positive | False Negative | P′ |
| | **n′** | False Positive | True Negative | N′ |
| | **total** | P | N | |

Table 2.4: Confusion matrix

The ROC curve is a plot of the true positive rate (TPR) against the false positive rate (FPR). A confusion matrix(see table 2.4) is a table that summarizes the performance of a binary classification model. It has four entries: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). The confusion matrix can be used to calculate the TPR and FPR at different classification thresholds, which can be used to plot the ROC curve. At each threshold, the confusion matrix is calculated, and the TPR and FPR are calculated based on the entries of the confusion matrix. The TPR(also known as sensitivity) is calculated as TP / (TP + FN), while the FPR is calculated as FP / (FP + TN). The TPR and FPR are then plotted on the ROC curve, with the TPR on the y-axis and the FPR on the x-axis. TNR (also known as specificity) measures the model's ability to correctly identify negative instances. The TNR is calculated as TN / (TN + FP). False Negative Rate (FNR) is the ratio of false negatives to the total number of actual positives calculated as FN / (FN + TP).

AUC summarizes the trade-off between TPR and FPR into a single number, where an AUC of 1.0 corresponds to a perfect model and an AUC of 0.5 corresponds to a random model, and AUC of 0 means that the binary classifier's predictions are completely wrong. The AUC provides a measure of how well the model separates positive and negative instances and gives a single metric for comparing models.

# 3
# Related Work

## 3.1 Predictive Analytics in Healthcare

Predictive analytics serves as an efficient tool in the healthcare domain, especially to improve patient outcomes and reduce healthcare costs. In this section, I describe some major case studies with methodologies, on the use of predictive analytics in health care. It is important to note that the quality of results obtained from the analysis is vastly dependent on the quality of data available for analysis. However, as our understanding of data accuracy and its optimal representation increases, a great amount of improvement has been observed in the field of healthcare. For eg:

1. Prediction of patient readmission.
2. Prediction of chronic diseases such as diabetes, heart diseases and cancer.
3. Prediction of medication adherence in patients.
4. Predictive modelling of ICU patients for the length of stay, mortality prediction etc
5. Prediction of sepsis.

My research has primarily focused on predicting the length of stay and mortality of ICU patients. This is generally useful to identify patients at a state of high risk and to enable early interventions and improve their outcomes and is typically achieved by implementing machine learning models on historical patient data, demographic information, clinical variables and treatment information. Some of the most commonly used variables in this case, which I have also considered in my work are:

- Vital signs e.g. heart rate, blood pressure, temperature etc.
- Laboratory test results e.g. blood glucose levels
- Medical history e.g. previous medications
- Treatment information e.g. current medications

### 3.1.1 Mortality Prediction

The main goal of mortality prediction in patients, which is most often applied to patients in ICU and in critical conditions, is to estimate the likelihood of death of patients, thereby assigning priority to the patients with a high likelihood of mortality. Variables such as vital signs, lab test results and medical history could be modelled in logistic regression, decision trees, random forest and/or neural networks to achieve this.

### 3.1.2 Length of Stay (LOS)

Modelled using similar variables as mortality prediction such as vital signs, lab test results etc., LOS prediction aims to estimate the duration of hospitalization in patients, most often for patients in ICU. In addition to helping clinicians plan better care routines for patients, such results also contribute to the efficient management of hospital resources which in turn improves outcomes for other patients.

## 3.2 Neural Networks

Neural networks can be used for efficient mortality prediction and length of stay in patients since they are especially eminent in capturing complex affinities between variables and can be used to model non-linear relationships, which may not identifiable through traditional statistical methods. For both mortality prediction and length of stay, neural networks can be trained from a range of available techniques such as feedforward, convolutional or recurrent, gaining a probability score as an output which can be used to categorize patients into several risk categories.

There are two primary challenges, however, in using neural networks for mortality and LOS prediction: the need for high-quality data in training

and the complexity of the neural network models causing difficulty in interpreting them. Additionally, once generated, the models would also require constant updates and validation to maintain accuracy and ensure their effectiveness over time.

## 3.3 Patient Representation

The important objective of graph representation learning is to produce representation vectors that can encapsulate the important features of large graphs precisely. This is vital for downstream prediction tasks such as node classification, and link(e.g. edge) prediction which essentially relies on the quality of the representation vectors of the graphs. Graphs can contain various kinds of entities(nodes) and an edge connecting them. In contrast, traditional deep learning methods cannot incorporate such diversified properties of the entities and strong correlations between them.

Patient representation learning is a type of machine learning that focuses on analyzing and extracting meaningful features or representations from patient data, such as medical records or imaging data, to improve clinical decision-making and patient outcomes. Traditionally, medical data has been analyzed using statistical methods, which may not capture the complex relationships between different variables of patient data. However, patient representation learning can leverage more advanced machine learning techniques, such as deep learning and neural networks, to identify important patterns and relationships in the data. Huang et. al.[42] discuss various techniques for patient representation learning, including deep learning models such as autoencoders, convolutional neural networks, and recurrent neural networks. They also provide examples of how these techniques have been used in various healthcare applications, such as predicting disease risk and identifying disease subtypes. The authors highlight the potential benefits of patient representation learning for improving healthcare outcomes and call for further research in this area. In the following sections, various patient representation learning methods have been reviewed and compared.

### 3.3.1 Vector-based patient representation

Vector-based patient representation refers to the representation of patients as vectors (i.e., arrays of numerical values) in a high-dimensional feature space. These vectors are learned using machine learning techniques, such as deep neural networks, and can capture complex relationships between different patient data modalities, such as electronic health records, medical imaging, and genomic data. Wang et al.[46] developed a deep-learning model to automatically detect metastatic breast cancer in lymph node specimens. The model was trained on a large dataset of high-resolution digital pathology images and used a convolutional neural network to learn a vector-based representation of the images. The learned representations were then used to classify the images as either positive or negative for metastasis. The model achieved state-of-the-art performance on a public dataset and demonstrated the potential of vector-based patient representation for improving the accuracy and efficiency of a cancer diagnosis.

Word embeddings are a popular vector-based patient representation learning technique that is used to represent textual data. Word embeddings represent each word in a patient's EHR as a vector in a high-dimensional space, where semantically similar words are mapped to nearby vectors. For example, a study by Choi et al. [12] used word embeddings to predict the risk of heart failure for patients based on their EHRs, achieving higher accuracy compared to other machine learning models. Word embeddings are a simple and effective way to represent text data as vectors, which can capture semantic relationships between words and can be used with both structured and unstructured data. However, they can be sensitive to the choice of parameters used to generate them, which can lead to different embeddings for the same words.

Deep autoencoders are a type of neural network architecture that learns to compress and reconstruct input data. A study by Cheng et al. [11] used autoencoders to predict the risk of COVID-19 for patients based on their EHRs, achieving higher accuracy compared to other machine learning models. Autoencoders can learn a low-dimensional representation of high-dimensional data, which can capture important features and patterns. They can be used with various types of data, including both structured and unstructured data, and can be trained in an unsupervised manner.

However, Autoencoders can suffer from overfitting when the dimensionality of the latent space is too small. Additionally, they can be computationally expensive to train, especially for large datasets.

### 3.3.2 Temporal matrix-based patient representation

In this type of representation, a 2D matrix for each patient is created where the x-axis represents time and the y-axis represents all the clinical events. High-dimensional data is eventually reduced to lower dimensions with the aid of a set of non-negative elements generated by non-negative matrix factorization(NMF). The authors [47] have used NMF based framework to extract the temporal features of the patients and therefore created a matrix of all the events. They have validated their methodology on a real-world diabetes dataset which contains diabetes as labels. They have used their framework to mine the temporal features from lab results, medical procedures, etc. They have been shown to outperform the baseline methods without the temporal features using Area Under the curve(AUC). This was one of the earliest research to create a mathematical representation of patients. Although a convolutional matrix factorization method was suggested them to identify shift-invariant patterns across patient EHR matrices, they are unable to select the ideal pattern lengths and must loop every potential value.

Using almost a similar approach mentioned above, authors [10] have proposed a noble methodology to perform predictive modelling of chronic diseases. Initially, every patient is represented as a matrix of events with respect to time. Then a four-layer convolutional neural network was devised to get the phenotypes and finally make the prediction. They have improved upon the work of Zhou et al [53] who also extract phenotypes by grouping temporal trends having similarity. However, their work fails to address the temporal relationships across the different events.

### 3.3.3 Graph-based Patient representation

Graph-based patient representation learning uses graphs to represent patient data and extract meaningful features for various healthcare applications. In this approach, patients are represented as nodes in a graph, and the relationships between patients are represented as edges connecting

the nodes. Graph-based patient representation learning techniques aim to learn a low-dimensional representation of the patient graph that preserves important information about the relationships between patients. GCNs are a type of neural network that can operate on graphs to learn representations of nodes based on their neighbourhood relationships. GCNs have been used in patient representation learning for various healthcare applications, including disease diagnosis, patient similarity measurement, and drug response prediction. For example, a study by Ma et al. [29] used GCNs to predict the disease status of patients based on their EHR data, achieving superior performance compared to traditional machine learning methods. GATs are a type of neural network that can learn attention weights for the edges in a graph, allowing the model to focus on the most important relationships between patients. GATs have been used in patient representation learning for various healthcare applications, including patient clustering, drug response prediction, and disease diagnosis. For example, a study by Zitnik et al. [54] used GATs to predict the effectiveness of drugs for patients with breast cancer. GAEs are a type of neural network that can learn a low-dimensional representation of a graph while preserving its important structural properties. Graph Auto Encoder(GAE) have been used in patient representation learning for various healthcare applications, including disease diagnosis, patient clustering, and drug response prediction. For example, a study by Goyal et al. [18] used graph autoencoders to learn the graph structure of electronic health records (EHRs) and predict the risk of heart failure.

### 3.3.4 Sequence based representation

Each patient's timestamped event sequential characteristics are created using a sequence-based patient representation. RNNs are a type of neural network that can operate on sequences of data by learning a hidden state that represents the contextual information of the previous elements in the sequence. For example, a study by Che et al. [8] used RNNs to predict the risk of heart failure for patients based on their EHR data. Although the authors provide a promising approach for predicting clinical outcomes using RNNs and handling missing data in time-series data, further research is needed to validate the approach in diverse populations and healthcare settings, and to address limitations related to missing data and model com-

plexity. Transformer-based models can operate on sequences of data by attending to all elements in the sequence, without the need for a recurrent structure. For example, a study by Alsentzer et al. (2019) used a transformer-based model to predict the risk of sepsis for patients based on their EHR data. Hidden Markov Model(HMM) is a type of probabilistic model that can operate on sequences of data by modelling the probability of each element in the sequence given its previous elements. For example, a study by Che et al. [9] used HMMs to predict the risk of heart failure readmission for patients based on their EHR data. However, HMMs can be computationally expensive to train and may not be well-suited for large-scale datasets, which can limit the practicality of the approach for use in clinical settings.

### 3.3.5 Tensor-based representation

Tensor-based patient representation learning techniques use tensor data to extract meaningful features and patterns for various healthcare applications. Tensors are multi-dimensional arrays that can represent complex data structures such as multi-modal healthcare data. Tensor-based patient representation learning techniques can be divided into three categories: tensor factorization, tensor regression, and tensor networks. Tensor factorization is a type of matrix factorization technique that can operate on tensors by decomposing them into multiple lower-dimensional tensors. For example, a study by Kim et al. [25] used tensor factorization to predict the risk of heart failure for patients. Tensor regression is a type of regression technique that can operate on tensors by modelling the relationship between a tensor and a set of target variables. For example, a study by Kim et al. [24] used tensor regression to predict the response to chemotherapy for cancer patients based on their multi-modal healthcare data. Tensor networks can operate on tensors by learning a low-dimensional representation of the tensor while preserving its essential structural properties. For example, a study by Lee et al. (2020) used tensor networks to predict the risk of stroke for patients based on their multi-modal healthcare data, achieving higher accuracy compared to other machine learning models.

# 4
# Data

## 4.1 Dataset: EICU

In this section I will discuss about the dataset being used for this thesis and how I have preprocessed the dataset. I have used the eICU Collaborative Research Dataset [36], a database of Intensive Care Unit(ICU) patients from more than 200 hospitals in the USA. The database contains data from about 139,367 patients who spent at least one night in an ICU. 166,355 hospital admissions and 200,859 ICU admissions are included in the data. The dataset contains more admissions than patients because some patients were admitted to the ICU more than once during a single hospital stay or were hospitalized more than once, a disparity between the number of patients and admissions is seen.

Patients' discharge statuses from the ICU or hospital are noted as "Expired" or "Alive." 156,476 entries had people with a discharge status of "Alive," whereas 9,861 patients had a discharge status of "Expired." The patient table includes demographic information such as age, gender, and ethnicity. The median age of patients admitted to the ICU is 65 years old, and mortality rises with the age of admission. ICU admission statistics on gender show that 108 379 of patients identified as male, 92 303 as female, 35 were recorded as Unknown, and 8 chose the status Other. The mortality rate of both males and females are almost similar, 5.46 % for males and 5.37 % for females. However, the mortality rate in the hospital show little difference, as females have a rate of 9.15 % and male 8.9 %.

Diagnosis of patients is an important factor to consider for building out machine learning model. So, we have classified the most important diag-

Table 4.1: Most frequent diagnosis

| **Diagnosis** | **Count** |
|---|---|
| Sepsis, pulmonary | 8862 |
| Infarction, acute myocardial (MI) | 7228 |
| CVA, cerebrovascular accident/stroke | 6647 |
| CHF, congestive heart failure | 6617 |
| Sepsis, renal/UTI (including bladder) | 5273 |
| Rhythm disturbance (atrial, supraventricular) | 4827 |
| Diabetic ketoacidosis | 4825 |
| Cardiac arrest | 4580 |
| CABG alone, coronary artery bypass grafting | 4543 |

noses by filtering diseases with the highest frequencies. The table below shows the diagnoses with more than 4500 patients.

## 4.2 Preprocessing

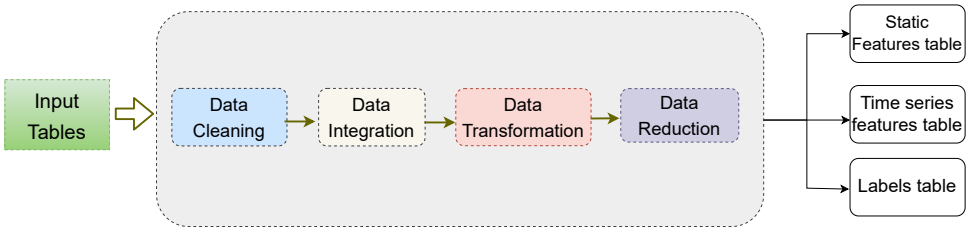


Figure 4.1: Overview of data preprocessing steps

In this project, I have used the data preprocessing steps(see figure 4.1), which I shall explain in the following sections. In the input tables shown in the first block of the diagram, the following four queries were run on data stored in postgresSQL and also in Neo4j graph database to generate labels, diagnosis, flat features and time-series tables.

**Query 1** (Labels Table): The labels table has been generated by running an SQL query to inner join patient and apachepatientresult. The patients from both the tables had been matched by the patientunitstayid. Only the most recent apache prediction model was used, excluding anyone who doesn't have at least 24 hours of data. A selection table is created which considers the first ICU stay of every patient, and satisfying the 24 hours of stay. The

Table 4.2: Static Features extracted from EHR data

| **Attribute** | **Type** |
|---|---|
| Age | Discrete |
| Gender | Binary |
| Height | Continuous |
| Ethnicity | Categorical |
| Weight | Continuous |
| UnitType | Categorical |
| UnitAdmitSource | Categorical |
| UnitStayType | Categorical |
| UnitStayType | Categorical |
| Physician Speciality | Categorical |
| Eyes | Discrete |
| Verbal | Discrete |
| Motor | Discrete |
| Meds | Discrete |
| Intubated | Binary |
| Dialysis | Binary |
| Ventilated | Binary |

idea behind that is the model should be used as soon as the hospital has 24 hours of continuous data. Running this query returns 89143 stays of patients. We have written a similar query to extract the patients from Neo4j Graph database.

**Query 2** (Diagnosis table): We have taken patients only present in the labels table. The diagnosis was entered either before the patient was admitted to the hospital or recorded in the first 24 hours of stay.Also, the past medical history and the primary diagnosis of every patient before being admitted to the ICU was been extracted. The patients in admissiondx table and pasthistory table was matched by using the unique patientunitstayid. The diagnosis string of 89143 patients were been extracted by the help of this query.

**Query 3** (Flat features table): By using this query we have extracted 17 static features of patients from apacheapsvar and apachepatientresult table.Only the patients listed in the labels table were inner joined using the patientunitstayid. The list of the flat features has been listed below

**Query 4** (Timeseries table): Features from multiple tables has been extracted from to create time series table. Firstly, we extract the most common labs present in the dataset, and consecutively the counts of number of patients having result for those labs. The labresultoffset was selectd between -1440 and 1440 because it is important to note down lab results before the start of the unit time. This could potentially help the imputation of missing values forwards. After extraction of all the labs, we limit to those common labs which are present for at least 25 % of the patients. And the counts of patients were in descending order. This results in two columns having labname and count and 47 lab features.

Secondly, we retrieve the time series features from the common lab tests extracted above. By inner joining the common labs and labels table and keeping the labresultoffset between -1440 and 1440 we get four columns of data, namely, patientunitstayid, labresultoffset, labname and labresult.

Thirdly, we retrieve the most common respiratory chart entries and consecutively how many patients have results from respiratory charting table. Similar to above we consider respchartoffset between -1440 and 1440 so that we can do the imputation during the preprocessing stage. By inner joining the labels table and grouping by respchartvaluelabel, we only keep the data that is available for at least 13% of the patients because only few patients are ventilated. This results in a table having only two columns namely, respchartvaluelabel and count.

Then, based on the respiratory chart entries we have extracted in the previous query, we create a table having respchartoffset, respchartvaluelabel and respchartvalue by inner joining commonresp and labels table. The respiratorychart offset is kep between -1440 and 1440. After this, the periodic(regularly sampled) time series data extraction is followed by doing an inner join of vitalperiodic extracing a multiple of features namely, observationoffset, temperature, sao2, heartrate, respiration, cvp, systemicsystolic, systemicdiastolic, systemicmean, st1, st2 and st3. The observation offset range as kept as similar.

Finally, we extract the aperiodic data which are irregularly sampled time series data. A range of features namely observationoffset, noninvasivesystolic, noninvasivediastolic, noninvasivemean is extracted by this last query.

As soon as the tables available after running the SQL and cypher queries, I get raw data in the form of csv's and passed to the main preprocessing block(see figure 4.1)

**Data Cleaning** Missing values in the flat tables are handled by filling the values in the age column with the mean age. There are few ages, which are '>89', for them an extra variable was created to store that information. For the timeseries data, masking was done at 1-hour intervals. The data was irregular with missing values, so the values had been forward-filled. The column where there was % sign present, was been removed and then converted into numbers.

**Data Integration** In the data integration stage, I have combined multiple tables which contains time dependent features into one to perform further preprocessing steps.

**Data transformation** In the data transformation stage, normalization and standaradization are performed. So it is important to rescale the variable to have similar range or variance for the machine learning model to learn. Some features such as admissionHeight, age, eyes, motor, verbal and hour are not normally distributed, so to normalize the data, we subtract the minimum value and then divide by the max value for each of these features. Categorical data has been converted to numerical data using one hot encoding, where each category is represented as a binary vector. The labels has been processed by replacing the actualhospitalmortality with the expired as 1 and the alive as 0. In order to preserve the hierarchical structure of the diagnosis, I have used multi-hot encodings with each spot relating to a different diagnosis and assigned distinct characteristics to each class level. This results in a vector with an average sparsity of 99.5 % and a size of 4,436. Diagnoses having a prevalence of more than 0.5 % are included. If an illness falls below this cutoff, it is included through any parent classes that do.

**Data reduction** Only a selected number of features taken in the data reduction stage for the static features as shown in table 4.2. And the set of time series features has been extracted as shown in table 4.3.

The output of the preprocessing steps are three tables(static features, label and timeseries). Each of the tables were split into 70% for training, 15%

Table 4.3: Timeseries Features extracted from EHR data

| **Attribute** | **Type** |
|---|---|
| Bedside glucose | Continuous |
| FI02 | Continuous |
| SaO2 | Continuous |
| Non-Invasive Diastolic | Continuous |
| Non-Invasive Systolic | Continuous |
| Non-Invasive Mean | Continuous |
| CVP | Continuous |
| Heart Rate | Continuous |
| Respiration | Continuous |
| st1 | Continuous |
| st2 | Continuous |
| st3 | Continuous |
| Systemic Diastolic | Continuous |
| Systemic Systolic | Continuous |
| Systemic Mean | Continuous |
| Temperature | Continuous |

validation and 15% for test. The patient ids were kept unique in each of the different splits of the different splits.

# 5 Methods

The figure 5.1 below depicts the overview of our approach. We start by storing the electronic health records in a graph database storage (Neo4j). The data is stored in the underlying graph storage as nodes(see figure 5.2) and edges before the preprocessing steps. Every patient has multiple unit stays in the ICU. And each unit stay has multiple diagnoses of a patient recorded. Apart from that, labs, vitalperiodic(continuously recorded every time interval, e.g., heart rate), past history, are stored in the respective nodes shown in the diagram. We compare the query performance of Neo4j with SQL query stored in PostgresSQL. The results of the query performance will be compared(section 6).

To start with step A in figure 5.1, we take into account a group of features that describe the patients, including both unchanging(static) features and features that change(temporal)s over time. The features are fetched from Neo4j and stored in a CSV(easier to integrate with Machine learning workflows). After I performed pro-processing(see section4), the set of patients go through a graph construction phase. The resulting graph contains all patient as nodes connected based on their similarity. Next, I train the Temporal(LSTM)-Graph(GNN) with the patient graph in an end-to-end fashion. For the purpose of obtaining per-node predictions of mortality and length of stay task, the Temporal-Graph generates three different types of embeddings: the temporal dependencies, spacial dependencies, and static dependencies(shown in section B of the diagram). Now, lets discuss the procedures of graph construction and the training steps of the Temporal-Graph model.

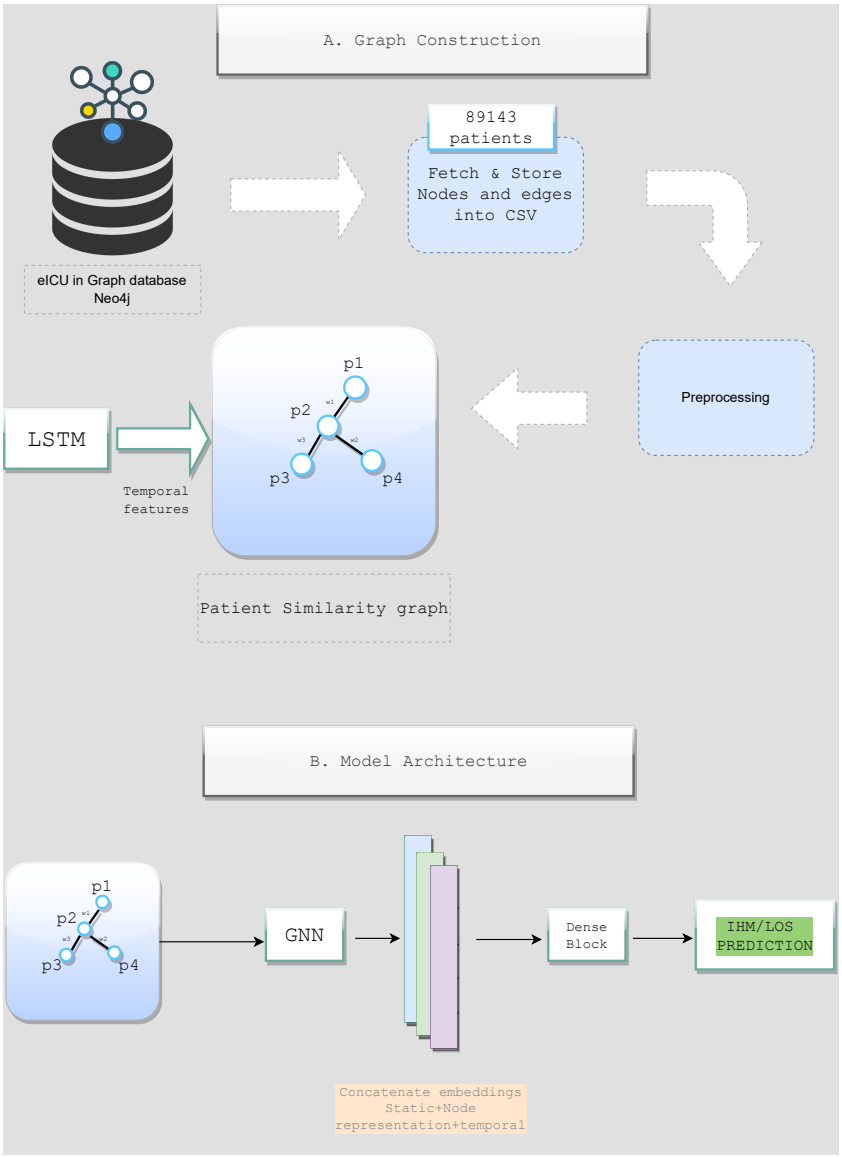


Figure 5.1: Overview of approach.

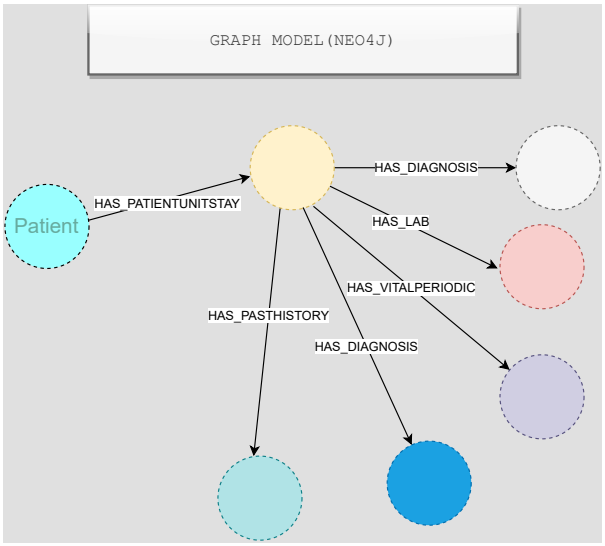


Figure 5.2: Graph storage model.

## 5.1 Patient similarity graph construction

In this section I will discuss the concepts behind the patient similarity graph construction methodology with the help of a mathematical formula. Before the calculation, all the diagnoses of the patients are multi-hot encoded produce a matrix with dimensions m rows and n columns. In this context, m denotes the total number of patients, while n refers to the number of distinct diagnoses. I have used the formula for patient's similarity graph construction which was published by Rocheteau et al.[38].

$$Score(i,j) = a(\sum_{u=1}^{n} D_i \alpha D_j \alpha(\frac{1}{o\alpha} + b)) - \sum_{u=1}^{n} (D_i \alpha + D_j \alpha) \tag{5.1}$$

(5.2)

where

$Score(i,j) is similarity score$

$D_{i\alpha}$ – Diagnosis vector of patient i

$D_{j\alpha}$ – Diagnosis vector of patient j

$n$ – number of distinct diagnosis

$o_\alpha$ – occurrence of a diagnosis

$a, b$ – hyperparameters

The formula above, calculates the similarity score between two patients i and j. The first part(green color) of the formula calculates the score of shared diagnoses and the second part means all the diagnoses encountered between patients i and j. Multiplying the diagnoses in the first part ensures that only the diagnoses which are shared between the two patients are taken into account. And then incorporating the inverse of that unique diagnose taken the rare diagnosis into consideration because it is important to consider not only shared diagnoses but also the rare diagnoses which are also important for calculating the similarity. The constants a and b are used so that we can get a positive value for the similarity score. And in the second part(blue color), subtracting the summation of all the diagnoses benefits the patients from becoming high hubs of connectivity. To determine the number of edges connected with every patient node, we evaluate the calculated Score using the k-Nearest Neighbor (k-NN) technique.

## 5.2 Training steps using LSTM-GNN

After creation of the patient similarity graph, node classification as outcome prediction(alive or expired) is performed. The model we use is a hybrid of two components, Graph(GNN) and Temporal(LSTM) [38], which work together to encode graph and temporal information, respectively. Each patient node consists of static features as well as time series features. In the first step, the time series features are provided as input to a bi-directional LSTM, which outputs a set of hidden state vectors in both the forward and backward directions. To generate a temporal embedding for each node, we concatenate the vectors that correspond to the most recent timestep. The GNN component then propagates the temporal embedding of each node inside its neighborhood. In order to generate a new node representation,

the GNN modifies the weight of each patient node's features by considering the feature vectors of its adjacent nodes in its immediate vicinity. The static features are being given as input to the fully connected layer which also generate a representation for each patient. Finally, all the three different representations(temporal dependencies from the LSTM, spatial dependencies from the GNN and static features from a fully connected layer) are being concatenated and passed through a dense block to give a final prediction for each node(patient).

## 5.3 Baseline Model

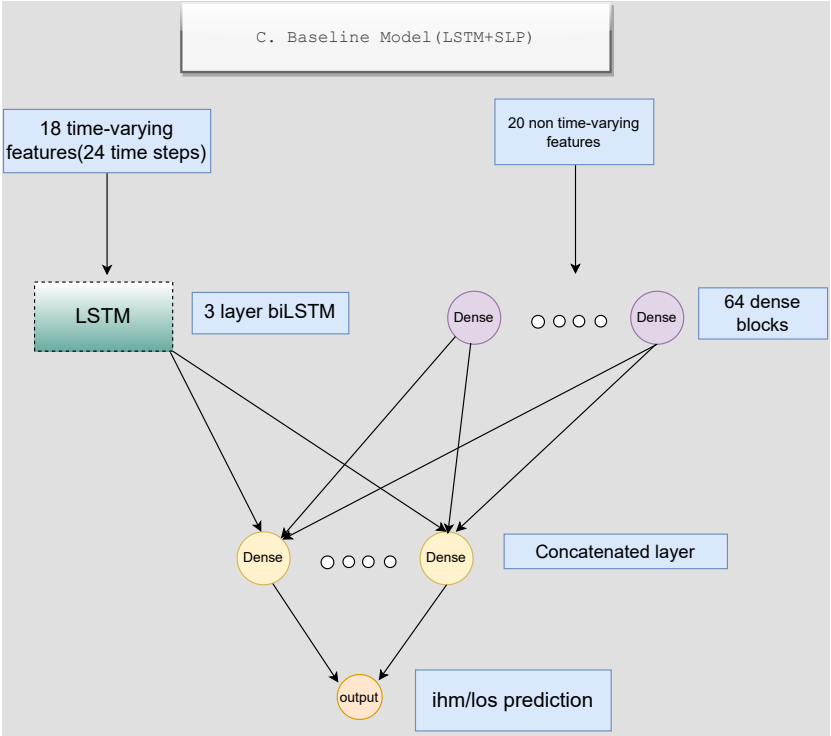


Figure 5.3: Graph storage model.

We have created a baseline model for processing time series features and and static features. Therefore, we have created a model consisting of a bidirectional LSTM layer and a single layer perceptron(see figure 5.3). The idea behind this method is that the single-layer perceptron will concentrate on the model's static characteristics while the bi-directional LSTM learns the relationship between the time-series features. One SLP and three LSTM

layers make up the model architecture, which is followed by a concatenated layer that combines the output from LSTM and SLP layers. This combined output is then sent to a further dense layer, followed by an output layer with an sigmoid activation that can determine the mortality of the patient.

The model summary has been given in the image below:-

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 TIMESERIES_INPUT (InputLayer)  [(None, 24, 34)]     0           []

 BIDIRECTIONAL_LAYER_1 (Bidirec  (None, 24, 256)     166912      ['TIMESERIES_INPUT[0][0]']
 tional)

 DROPOUT_LAYER_1 (Dropout)      (None, 24, 256)      0           ['BIDIRECTIONAL_LAYER_1[0][0]']

 BIDIRECTIONAL_LAYER_2 (Bidirec  (None, 128)         164352      ['DROPOUT_LAYER_1[0][0]']
 tional)

 STATIC_INPUT (InputLayer)      [(None, 447)]        0           []

 DROPOUT_LAYER_2 (Dropout)      (None, 128)          0           ['BIDIRECTIONAL_LAYER_2[0][0]']

 DENSE_LAYER_1 (Dense)          (None, 64)           28672       ['STATIC_INPUT[0][0]']

 CONCATENATED_TIMESERIES_STATIC  (None, 192)         0           ['DROPOUT_LAYER_2[0][0]',
  (Concatenate)                                                   'DENSE_LAYER_1[0][0]']

 DENSE_LAYER_2 (Dense)          (None, 64)           12352       ['CONCATENATED_TIMESERIES_STATIC[
                                                                 0][0]']

 OUTPUT_LAYER (Dense)           (None, 1)            65          ['DENSE_LAYER_2[0][0]']

==================================================================================================
Total params: 372,353
Trainable params: 372,353
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 5.4: Model summary LSTM SLP architecture

The static inputs to the model are static features of individual patients in the ICU, namely, gender, age, height, weight. ethnicity, etc. But time series features include patient features which are varying over time, example, bedside glucose, Heart rate, respiration, systolic, diastolic pressure, etc. These measurements change every hour for the 24 hours periods which is taken under consideration. The number of timesteps considered for the training is 24. The training goes for 50 epochs having a batch size of 64. Smaller batch size have the benefit of regularizing effect and reducing the generalization error. Also, it helps to fit one batch of training data in memory. Batch size of 64 signifies that the 64 samples of the training dataset to calculate the error gradient before the weight of the model is updated. And for each training epochs the, algorithm make one pass through the train data.

Neural networks which are large and trained on smaller datasts cal cause overfitting to the training data. The model learns the noise of the training data which results in poor generalization on new unseen data.As a result, generalizaiton error increases beacause of overfitting. One approach we could follow is to train the same dataset on multiple different neural networks and to average the predictions of every model. This is not feasible due to the time constraints. TO overcome that issue of overfitting, we have used dropout layer in between the LSTM Layer. The way it functions is by ignoring or dropping out some number of layer outputs.

# 6

# Results and Evaluation

In this section of the thesis, I present the results and evaluation of the proposed methods, which aim to answer the research questions defined in the introduction (section 1). The model was trained to predict two patient outcomes: mortality and length of stay. To answer RQ1, I divided the patient outcomes into classification and regression tasks and evaluated the performance of LSTM-GAT and LSTM-SLP using a set of evaluation metrics. In addressing RQ2, I compared the respective models with and without diagnosis features added. Finally, for RQ3, I measured the query times of two different databases.

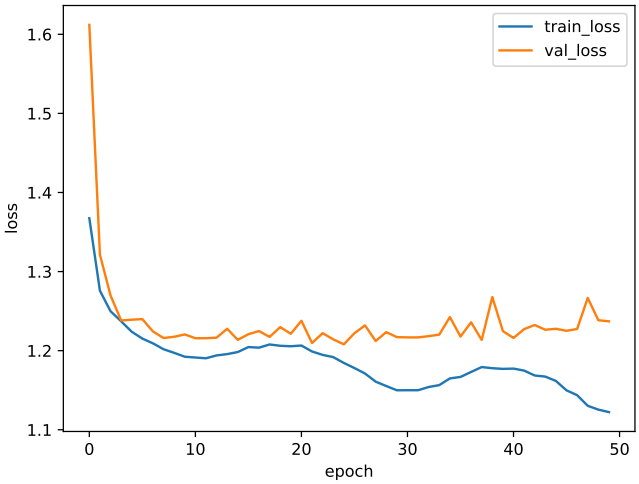


Figure 6.1: LSTM-GAT Loss Function for In-hospital mortality prediction.

## 6.1 RQ1

For the LSTM-GAT model, the batch size was kept as 128 and the training was run for 50 epochs. The activation function for GAT used was 'elu' and the loss function for the classification task was used as a cross-entropy loss. And for the regression task, the loss function was used MSELoss. After every epoch, the training loss and validation loss were monitored(see figure 6.1). As can be seen from figure 6.1, the gap between the training and validation loss is decreasing up to 20 epochs, which means the model has learned the data properly. But after 20 epochs, since the model is very complex, it is overfitting to the training data meaning that it is incapable to generalize well to new, unseen data. In comparison, the LSTM-SLP loss curve(see figure 6.4) is seen to have less gap between the train and the validation data which seen the model being trained without being overfitted for the LSTM-SLP.

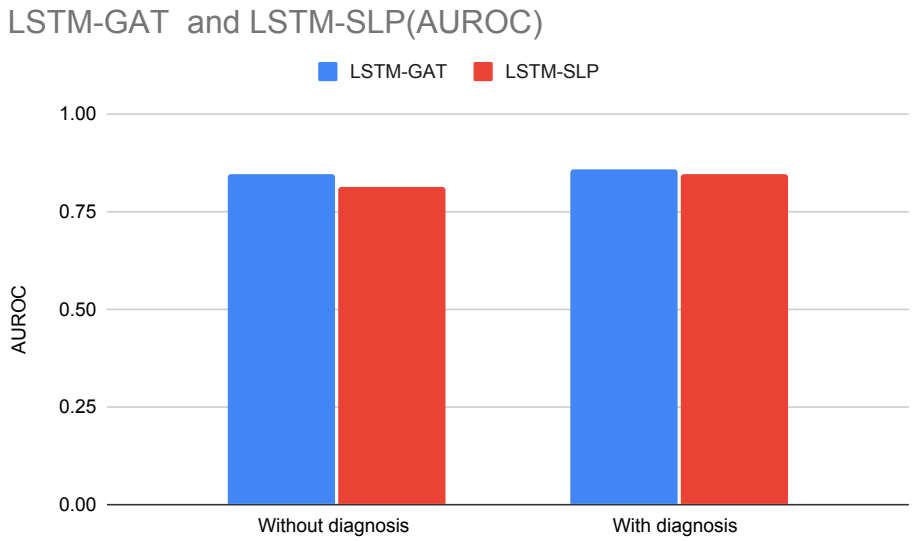


Figure 6.2: LSTM-GAT vs LSTM-SLP for In-hospital mortality prediction(AUROC)

As seen in figure 6.2, the LSTM-GAT model is seen to perform slightly better(3.93%- without diagnosis and 1.77% with diagnosis) for the mortality prediction task. The results being compared in the chart are for the evaluation metric AUROC.

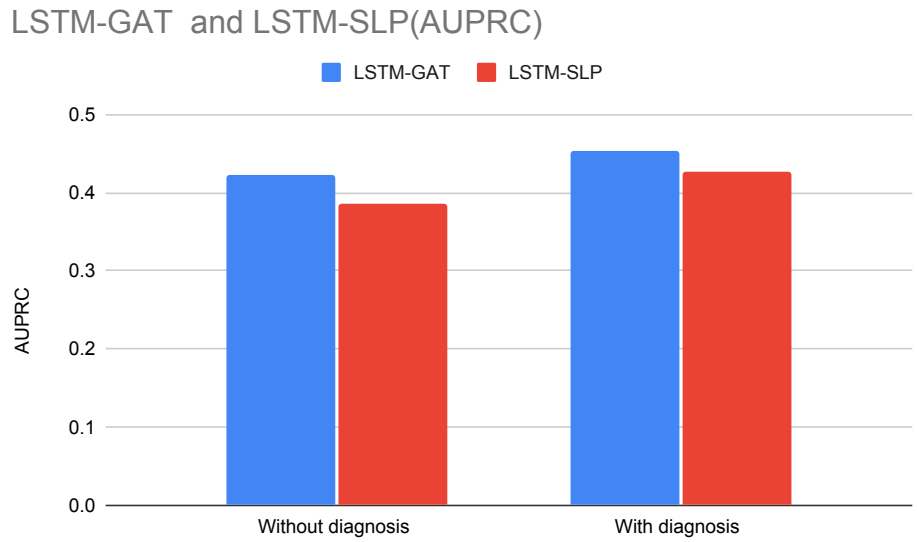


Figure 6.3: LSTM-GAT vs LSTM-SLP for In-hospital mortality prediction(AUPRC)

As seen in figure 6.3, the LSTM-GAT model is performing better for the mortality prediction task(9.8 %- without diagnosis and 6.3 % increase with diagnosis). Therefore, to conclude, graph neural networks perform slightly better predictions than single-layer perceptrons.

## 6.2 RQ2

After doing the preprocessing of the dataset, I have successfully created patient similarity graph. The similarity was defined by the similar diagnoses between patients. And each patient contained flat features and time-dependent features(see section 5). As soon as the graph generation for patient was complete, it is passed through LSTM-GAT model. The model was trained on the objective to find mortality and length of stay. The training was performed for 50 epochs with a learning rate of 0.0005. The evaluation on done on seperate unseen set of patients and is based on AUROC and AUPRC for mortality. The loss function for LSTM-GAT model with diagnosis(see figure 6.4) is given as follows.

Table 6.1 shows the results of the experiment run for the LSTM-GAT model, without and with diagnoses concatenated to the flat features. The higher the AUROC and AUPRC values, the better. It is evident from the diagrams

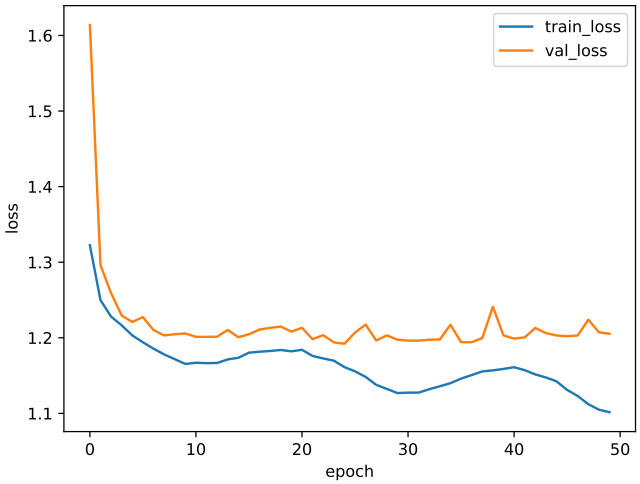


Figure 6.4: LSTM-GAT Loss Function for In-hospital mortality prediction.

| Model | AUROC | AUPRC |
|---|---|---|
| LSTM-GAT( no diagnoses) | 0.846 | 0.422 |
| LSTM-GAT( with diagnoses) | 0.859 | 0.453 |

Table 6.1: LSTM-GAT model (IHM) results

and the table that both values are higher when diagnoses are added to the features(with diagnoses(AUROC: 0.859, AUPRC: 0.453) compared to without diagnosis(AUROC: 0.846, AUPRC: 0.422)) during training allowing the Graph Attention networks to learn better.

The reason behind LSTM-GAT performing slightly better when diagnosis are added is that it uses the attention to weight the importance of neighbouring nodes(patient). Here, each patient's contribution is weighted by the importance of this patient for the final prediction. Thereby, information from other patients with similar diagnoses help in predicting a patients mortality). However, GAT's learning process is computationally expensive and requires a large amount of data and computational resources, especially when working with large graphs.[13]

As another hybrid model, I have used LSTM-Single Layer Perceptron(SLP) to check the importance of graph structure. Here, I have removed the GNN so that I don't have information from patients with similar diagnoses. This

model is also trained and evaluated with and without diagnoses features added to the flat features.

| Model | AUROC | AUPRC |
|---|---|---|
| LSTM-SLP( no diagnoses) | 0.814 | 0.385 |
| LSTM-SLP( with diagnoses) | 0.844 | 0.426 |

Table 6.2: LSTM-SLP model (IHM) results

To compare, (with diagnoses(AUROC: 0.814, AUPRC: 0.385) compared to without diagnosis(AUROC: 0.844, AUPRC: 0.426)), shows an increment of 3.67% of AUROC when diagnoses added and 10.65 % increments with respect to AUPRC.

Since the AUPRC (Area Under the Precision-Recall Curve) value is low for mortality prediction, it suggests that the model's performance in identifying true positive cases (i.e. correctly identifying patients who will die) is poor. The reason could be that in the training dataset, the number of patients who died is low (5865 patients). AUROC is important because it provides a comprehensive summary of the model's performance across all possible threshold values

AUROC is a measure of a model's ability to discriminate between positive and negative cases, regardless of the imbalance in the data. AUPRC, on the other hand, is a measure of a model's ability to correctly identify true positive cases out of all predicted positive cases.

In the case of mortality prediction, the data is highly imbalanced, with a much larger number of negative cases (i.e. patients who will not die) than positive cases (i.e. patients who will die). Considering that, our model has a high AUROC value by correctly identifying a large number of true negative cases, but a low AUPRC value because it is not identifying enough true positive cases.

Figure 6.5 shows that for the LSTM-SLP model that the loss decreases while training. Since the gap between the training and validation loss is less after a few epochs, shows that the neural network was successfully trained. The ROC and PRC (see figure 6.6) curve for mortality prediction is being presented for the LSTM-SLP model.

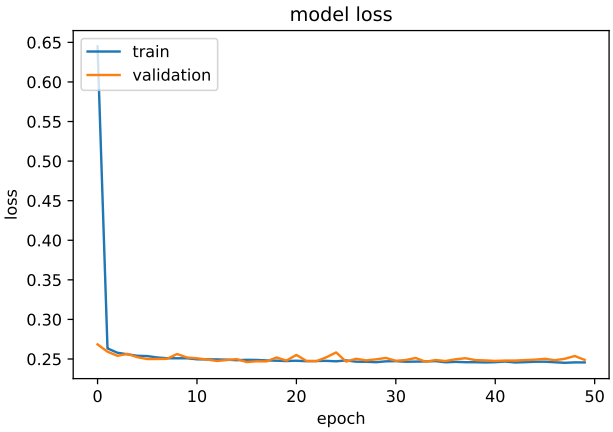


Figure 6.5: LSTM-SLP Loss curve for In-hospital mortality prediction.

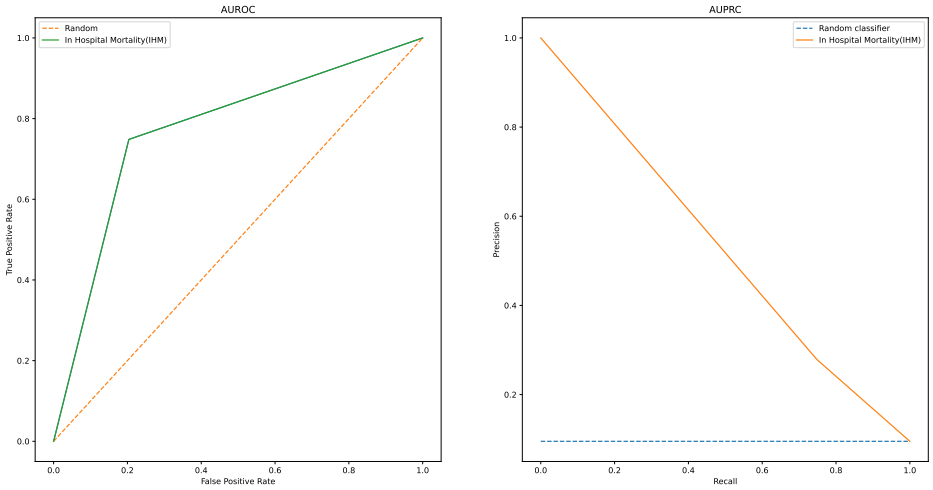


Figure 6.6: LSTM-GAT ROC and PRC curve

The larger area corresponds to a better performing classifier. The green curve in the left figure is the evaluation result from the model and the red dashed line is the what the random classifier performs.

**Length of Stay Prediction Task** I have used mortality prediction to compare the performance of the models with the different inputs on a classification task. But classification tasks are not enough to make valid comparison between different models. Therefore, I have trained the models on another

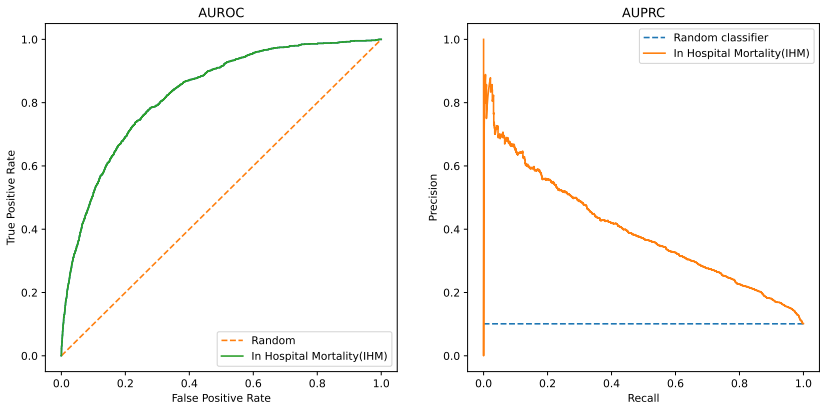


Figure 6.7: AUROC(left) and AUPRC(right) for In-Hospital Mortality(IHM)

objective known as length of stay prediction. I have evaluated the length of stay prediction task based on 5 different evaluation criteria as mentioned in table 6.3:

| Model | KAPPA | MAD | MSE | MAPE | MSLE |
|---|---|---|---|---|---|
| LSTM-GAT( no diagnoses) | 0.242 | 1.908 | 15.105 | 50.911 | 0.388 |
| LSTM-GAT( with diagnoses) | 0.257 | 1.891 | 14.926 | 50.070 | 0.381 |

Table 6.3: LSTM-GAT comparison for length of stay task

For Mean absolute deviation(MAD), Mean Squared Error(MSE), Mean absolute percentage error(MAPE) and Mean Squared Log error(MSLE), lower score is preferred whereas for Kappa, higher score is preferred. As can be seen from table 6.3, all the evaluations have slighty improved upon addition of diagnosis to the static features(MAD: 1.908 to 1.891, MSE: 15.10 to 14.926, MAPE 50.91 to 50.070, 0.388 to 0.381). And for kappa, we are getting higher results with diagnosis concatenated. The results for the hybrid LSTM-SLP model has been shown in table 6.4:

| Model | KAPPA | MAD | MSE | MAPE | MSLE |
|---|---|---|---|---|---|
| LSTM-SLP( no diagnoses) | 0.147 | 2.001 | 16.289 | 53.021 | 0.435 |
| LSTM-SLP( with diagnoses) | 0.283 | 1.91 | 14 | 54.59 | 0.384 |

Table 6.4: LSTM-SLP comparison for length of stay task

With diagnoses features added, the Kappa score has outperformed the LSTM-GAT model. Also, the MSE value has improved in the LSTM-SLP model.However, the MAD, MAPE and MSLE values indicate that LSTM-SLP model performs almost similarly to the LSTM-GAT model. There could be several reasons for GAT(Graph Attention networks) to not perform very well compared to the single layer perceptron(SLP).

- GAT is a more complex model than a single layer perceptron. While GAT can perform well on certain tasks, it may not perform well as SLP or when the data does not contain a lot of complex relationships.

- SLP is a very basic neural network and is less complex than GAT. It can handle linear decision boundaries, but not suitable for more complex non linear boundaries.

- Another reason could be, the length of stay prediction task is simple and the data can be easily separable by a linear decision boundary which makes it suitable for SLP to perform well in certain cases.

The following experiments help us to answer our first and second research questions, that for simple tasks such as mortality prediction and length of stay prediction, both LSTM-GAT and LSTM-SLP perform almost equivalent to each other, because the tasks does not involve complex model and relationships.

## 6.3 RQ3

To answer the third research question I have evaluated the query execution time for both SQL and Cypher in Neo4j Graph Database.

| Queries | Postgres(SQL) | Cypher |
|---|---|---|
| Q1(Fetching labels ) | 841ms | 128ms |
| Q2(Fetching diagnosis ) | 3117ms | 241ms |
| Q3(Fetching flat features ) | 514ms | 187ms |
| Q4(Fetching common labs ) | 6611ms | 55289ms |
| Q5(Fetching timeserieslabs ) | 10677ms | 283ms |
| Q6(Fetching commonresp ) | 3338ms | 25177ms |
| Q7(Fetching timeseriesresp ) | 4399ms | 585ms |
| Q8(Fetching timeseriesperiodic ) | 45028ms | 92ms |
| Q9(Fetching timeseriesaperiodic ) | 10663ms | 228ms |

Table 6.5: SQL vs Cypher Query execution time

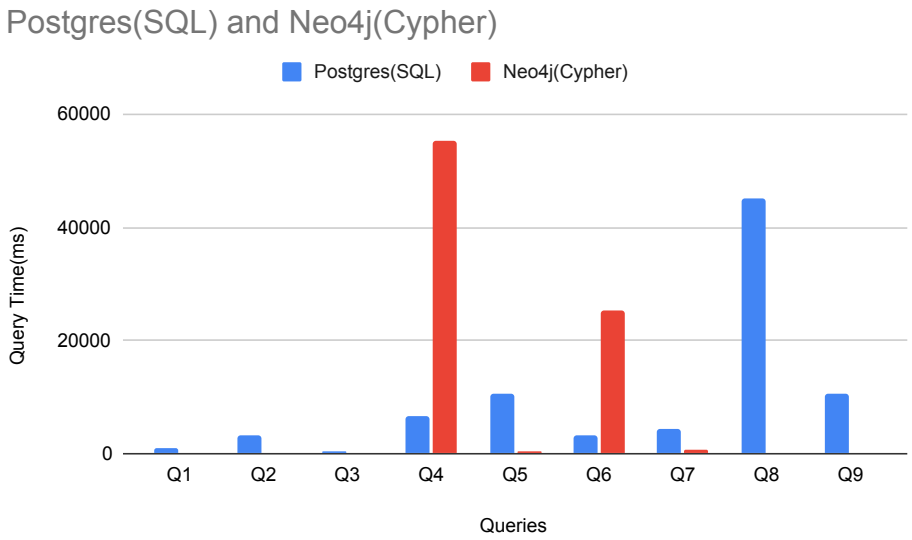


Figure 6.8: Queries execution time between Neo4j and Cypher

As can be seen from table 6.1 and figure 6.3, cypher queries have a better performance mostly compared to SQL queries. Because in most SQL queries, we have 'INNER JOIN' of multiple tables. This is taken care of in Cypher using the relationship between the graph nodes. The graph traversal to find specific nodes are much easier compared to SQL queries which cause the performance to improve in Neo4j. However, Q4 and Q6 performed better compared to SQL compared to Cypher. The size of the dataset used for both of these queries is comparatively larger compared

to the other queries, which can have an impact on the execution time. An example of a Cypher query and an equivalent SQL query for retrieving patient data from a hospital database:

Cypher Query:

```
MATCH (p:Patient)-[:VISIT]->(v:Visit)
WHERE p.name = 'John Smith' AND v.date >= date('2022-01-01')
RETURN p.name, v.date, v.reason
ORDER BY v.date DESC
```

SQL Query:

```
SELECT p.name, v.date, v.reason
FROM Patient p
JOIN Visit v ON p.id = v.patient_id
WHERE p.name = 'John Smith' AND v.date >= '2022-01-01'
ORDER BY v.date DESC;
```

In this example, the query retrieves the names of patients, visit dates, and reasons for visits for patient John Smith. The result set is ordered by visit date in descending order.

The Cypher query starts with the MATCH keyword to specify the pattern of nodes and relationships to match. The WHERE clause filters the results to only those that match the specified conditions. Finally, the RETURN clause specifies which properties to return in the result set and the ORDER BY clause orders the results in descending order.

In the SQL equivalent, the query uses JOINs to connect the Patient and Visit tables. The WHERE clause filters the results to only those that match the specified conditions, and the SELECT clause specifies which columns to return in the result set. The ORDER BY clause orders the results in descending order.

Both Cypher and SQL can be used to retrieve data from a database. While the syntax and structure of the queries are different, they can achieve the same goal of retrieving the desired data from the database. In the

patient data example, both Cypher and SQL queries are relatively simple and involve a single join between the Patient and Visit tables. Therefore, the difference in execution time and performance is likely to be minimal, and it may depend on the specific implementation of the database and query engine. In terms of syntax, Cypher is generally considered more expressive and intuitive for graph-related queries. It allows for a more natural and flexible traversal of nodes and relationships in the graph. SQL, on the other hand, is better suited for querying and manipulating tabular data.

# 7

# Conclusions and Future Work

In this thesis, I have answered the following research questions:

- Do LSTM-GAT perform better predictions than single-layer perceptrons(LSTM-SLP)?
- How important are the patient's diagnoses for the prediction with a graph neural network and for the prediction with an SLP?
- Are Cypher queries in Neo4j faster in fetching data than SQL queries in PostgreSQL?

To answer my research questions, I have used state-of-the-art deep neural networks using electronic health records(EHR) namely the eICU dataset. Initially, I stored the datasets in two different databases, where one is a graph database(Neo4j) and the other is a relational database (PostgresSQL). Different SQL and Cypher queries had been constructed to fetch the data from the two respective databases. The performance of the queries had been compared based on the amount of time each query requires to fetch the result from the database. As observed, Neo4j outperforms PostgresSQL, when the size of the data being fetched is not large enough than 100k. However, more experiments with large storage could have been performed, which were part of the limitations of this thesis. In total 89143 unique patient data were fetched and preprocessed. This gives 3 different patient tables containing labels, flat features, and time-series tables.18 time-varying features and 20 non-time-varying features were used for Machine Learning tasks. The multi-hot encoded diagnoses of the patients were later used to create a patient similarity graph. Every patient is connected to each other by edge weight, based on the number of common diagnoses,

and also the rare diagnoses. An LSTM model was used to create time-related embed-dings from time-series data. The static embed-dings were generated by passing through a fully connected layer. Later, a graph neural network algorithm called GAT is used to propagate the features with the neighbouring patient node to create graph embeddings. The 3 different embeddings are joined together to predict mortality and length of stay in the hospital.

The results of the two tasks were evaluated based on a set of evaluation criteria. Since mortality prediction is a classification task, AUROC and AUPRC were used to evaluate and compare the two different methods. From experiments performed, it is observed that the LSTM-GAT model with diagnoses added, adds predictive power to the model. The time series and static features together can facilitate the GNN to obtain better embeddings. However, the hybrid LSTM GNN is a complex model and needs much computational power during training. Whereas, the LSTM-SLP is a simple model, and performs almost equivalently to the hybrid model. In conclusion, although the graph neural network does not show a considerably large improvement in the evaluation results, the full potential of the GNN is yet to be explored.

As an extension to the current research performed, I suggest to make a comparison of the patient's homogeneous and heterogeneous graphs.The heterogeneous graph will contain both patient and medication node type. I believe that adding further information like medications to the heterogeneous graph would improve the predictive capability of the model. The medication feature could also be added to the homogeneous graph. Finally, it would be interesting to see how the models would perform after replacing the LSTM with the transformer model.

# A

# List of Figures

# B

# List of Tables

# C

# Bibliography


[1] [ABD-ELRAZEK et al. 2021] M. A. Abd-Elrazek, A. A. Eltahawi, M. H. Abd Elaziz und M. N. Abd-Elwhab. **Predicting length of stay in hospitals intensive care unit using general admission features**. Ain Shams Engineering Journal, Vol. 12(4):3691–3702, 2021.

[2] [AGATONOVIC-KUSTRIN und BERESFORD 2000] S. Agatonovic-Kustrin und R. Beresford. **Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research**. Journal of pharmaceutical and biomedical analysis, Vol. 22(5):717–727, 2000.

[3] [ÅHMAN et al. 2018] R. Åhman, P. F. Siverhall, J. Snygg, M. Fredrikson, G. Enlund, K. Björnström und M. S. Chew. **Determinants of mortality after hip fracture surgery in Sweden: a registry-based retrospective cohort study**. Scientific reports, Vol. 8(1):1–10, 2018.

[4] [ANZAI 1992] Y. Anzai. **Pattern recognition and machine learning**. Morgan Kaufmann, 1992.

[5] [BALOGH et al. 2015] E. P. Balogh, B. T. Miller, J. R. Ball, E. National Academies of Sciences, Medicine et al. **Overview of diagnostic error in health care**. In: Improving diagnosis in health care. 2015. National Academies Press (US).

[6] [BENGIO et al. 2013] Y. Bengio, A. Courville und P. Vincent. **Representation learning: A review and new perspectives**. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 35(8):1798–1828, 2013.

[7] [CATANESE et al. 2010] S. Catanese, P. De Meo, E. Ferrara und G. Fiumara. **Analyzing the Facebook friendship graph**. arXiv preprint arXiv:1011.5168, 2010.

[8] [CHE et al. 2018] Z. Che, S. Purushotham, K. Cho, D. Sontag und Y. Liu. **Recurrent neural networks for multivariate time series with missing values**. In: Scientific reports, Vol. 8, p. 6085. 2018, Nature Publishing Group.

[9] [CHE et al. 2017] Z. Che, S. Purushotham, Y. Liu, Y. Bengio, W. Li, X. Li, Y. Wang und Y. Liu. **Hierarchical recurrent state space models for predicting hospital readmissions**. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1305–1314. 2017, ACM.

[10] [CHENG et al. 2016] Y. Cheng, F. Wang, P. Zhang und J. Hu. **Risk prediction with electronic health records: A deep learning approach**. In: Proceedings of the 2016 SIAM international conference on data mining, pp. 432–440. 2016, SIAM.

[11] [CHENG et al. 2020] Y. Cheng, H. Wang, S. Wang, Y. Liu, X. Wan, J. Zhou, W. Zhong, Y. Luo, C. Huang und K. Wang. **Prediction of COVID-19 infeXtion risks using an ambulatory ECG**. IEEE Journal of Biomedical and Health Informatics, Vol. 24(10):2859–2868, 2020.

[12] [CHOI et al. 2016] E. Choi, M. T. Bahadori, J. Sun, J. A. Kulas, A. Schuetz und W. F. Stewart. **Retrieving cohorts of patients from clinical data repositories using bootstrapped learning of medical entity normalizations and index term co-occurrences**. In: AMIA Joint Summits on Translational Science Proceedings, 2016, pp. 41–50.

[13] [DAI et al. 2018] H. Dai, Z. Kozareva, B. Dai, A. Smola und Q. V. Le. **Sparse graph attention networks**. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 4148–4158.

[14] [DENSEN 2011] P. Densen. **Challenges and opportunities facing medical education**. Transactions of the American Clinical and Climatological Association, Vol. 122:48, 2011.

[15] [DREWS 2000] J. Drews. **Drug discovery: a historical perspective**. science, Vol. 287(5460):1960–1964, 2000.

[16] [FEY und LENSSEN 2019] M. Fey und J. E. Lenssen. **Fast graph representation learning with PyTorch Geometric**. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 244–254. 2019, ACM.

[17] [GÉRON 2019] A. Géron. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. O'Reilly Media, 2 Edn., 2019.

[18] [GOYAL et al. 2018] P. Goyal, H. Yang, W. Kim, X. Liu, H. Cheng, S. Ahn, J. Kim, S. Sohn und J. Sun. **Learning the graph structure of electronic health records with graph convolutional denoising autoencoders**. arXiv preprint arXiv:1801.07829, 2018.

[19] [GUPTA und SINGH 2013] A. Gupta und K. Singh. **Location based personalized restaurant recommendation system for mobile environments**. In: 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 507–511. 2013, IEEE.

[20] [HAMID und HABIB 2014] S. A. Hamid und A. Habib. **Financial forecasting with neural networks**. Academy of Accounting and Financial Studies Journal, Vol. 18(4):37, 2014.

[21] [HINTON 2018] G. Hinton. **Deep learning—a technology with the potential to transform health care**. Jama, Vol. 320(11):1101–1102, 2018.

[22] [HUANG et al. 2005] Z. Huang, X. Li und H. Chen. **Link prediction approach to collaborative filtering**. In: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries, 2005, pp. 141–142.

[23] [JOYNT und JHA 2013] K. E. Joynt und A. K. Jha. **Characteristics of hospitals receiving penalties under the Hospital Readmissions Reduction Program**. Jama, Vol. 309(4):342–343, 2013.

[24] [KIM et al. 2021] G. Kim, J. Kim, S. Kim, J. Ha, Y. W. Choi und J. Lee. **Tensor regression for predicting the response to neoadjuvant chemotherapy in breast cancer**. Artificial intelligence in medicine, Vol. 112:102015, 2021.

[25] [KIM et al. 2019] J. Kim, G. Kim, S. Kim, J. Ha, J. Kim, Y. W. Choi und J. Lee. **Tensor factorization-based patient representation for heart failure prediction**. Artificial intelligence in medicine, Vol. 96:89–97, 2019.

[26] [KOSOROK und LABER 2019] M. R. Kosorok und E. B. Laber. **Precision medicine**. Annual review of statistics and its application, Vol. 6:263–286, 2019.

[27] [KRIEGER et al. 2021] J. L. Krieger, J. M. Neil, K. A. Duke, M. S. Zalake, F. Tavassoli, M. J. Vilaro, D. S. Wilson-Howard, S. Y. Chavez, E. B. Laber, M. Davidian et al. **A pilot study examining the efficacy of delivering colorectal cancer screening messages via virtual health assistants**. American journal of preventive medicine, Vol. 61(2):251–255, 2021.

[28] [LIU et al.] **Heterogeneous Similarity Graph Neural Network on Electronic Health Records**.

[29] [MA et al. 2019] J. Ma, J. Gao und S. Wang. **Disease prediction using graph convolutional networks: Application to Autism Spectrum Disorder and Alzheimer's disease**. Bioinformatics, Vol. 35(24):5176–5183, 2019.

[30] [MALLIAROS und VAZIRGIANNIS 2013] F. D. Malliaros und M. Vazirgiannis. **Clustering and community detection in directed networks: A survey**. Physics reports, Vol. 533(4):95–142, 2013.

[31] [MEHTA et al. 2019] H. Mehta, P. Kanani und P. Lande. **Google maps**. International Journal of Computer Applications, Vol. 178(8):41–46, 2019.

[32] [MIRNEZAMI et al. 2012] R. Mirnezami, J. Nicholson und A. Darzi. **Preparing for Precision Medicine**. The New England journal of medicine, Vol. 366:489–91, 2012.

[33] [MONDIALE DE LA SANTÉ et al. 1993] O. mondiale de la santé, W. H. Organization et al. **The ICD-10 classification of mental and behavioural disorders: diagnostic criteria for research**, Vol. 2. World Health Organization, 1993.

[34] [OBERMEYER und LEE 2017] Z. Obermeyer und T. H. Lee. **Lost in thought: the limits of the human mind and the future of medicine**. The New England journal of medicine, Vol. 377(13):1209, 2017.

[35] [OF MEDICINE 2019] R. S. of Medicine. **Artificial Intelligence in Healthcare: Opportunities and Challenges**. Tech. rep., Royal Society of Medicine, 2019.

[36] [POLLARD et al. 2018] T. J. Pollard, A. E. Johnson, J. D. Raffa, L. A. Celi, R. G. Mark und O. Badawi. **The eICU Collaborative Research Database, a freely available multi-center database for critical care research**. Scientific data, Vol. 5(1):1–13, 2018.

[37] [RASCHKA und MIRJALILI 2019] S. Raschka und V. Mirjalili. **Python Machine Learning**. Packt Publishing, 2019.

[38] [ROCHETEAU et al.] **Predicting Patient Outcomes with Graph Representation Learning**.

[39] [ROOS et al. 1994] J. Roos, I. Lane, E. Botha und G. P. Hancke. **Using neural networks for non-intrusive monitoring of industrial electrical loads**. In: Conference Proceedings. 10th Anniversary. IMTC/94. Advanced Technologies in I & M. 1994 IEEE Instrumentation and Measurement Technolgy Conference (Cat. No. 94CH3424-9), pp. 1115–1118. 1994, IEEE.

[40] [RUMELHART et al. 1985] D. E. Rumelhart, G. E. Hinton und R. J. Williams. **Learning internal representations by error propagation**. Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[41] [SHEN et al. 2017] D. Shen, G. Wu und H.-I. Suk. **Deep learning in medical image analysis**. Annual review of biomedical engineering, Vol. 19:221–248, 2017.

[42] [SI et al. 2021] Y. Si, J. Du, Z. Li, X. Jiang, T. Miller, F. Wang, W. J. Zheng und K. Roberts. **Deep representation learning of patient data from Electronic Health Records (EHR): A systematic review**. Journal of biomedical informatics, Vol. 115:103671, 2021.

[43] [TRUDEAU 1993] R. J. Trudeau. **Introduction to Graph Theory**. Dover Books on Mathematics. Dover Pub., 1993.

[44] [VASWANI et al. 2017] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser und I. Polosukhin. **Attention is all you need**. Advances in neural information processing systems, Vol. 30, 2017.

[45] [VELIČKOVIĆ] **Message passing all the way up**.

[46] [WANG et al. 2016] D. Wang, A. Khosla, R. Gargeya, H. Irshad und A. H. Beck. **Deep learning for identifying metastatic breast cancer**. arXiv preprint arXiv:1606.05718, 2016.

[47] [WANG et al. 2012] F. Wang, N. Lee, J. Hu, J. Sun und S. Ebadollahi. **Towards heterogeneous temporal clinical event pattern discovery: a convolutional approach**. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012, pp. 453–461.

[48] [WU et al. 2020] C. Wu, J. Abernethy, S. Zhexuan, Y. Wang, J. Ni, H. Xu, R. Zegarac, X. Xu und P. W. Koh. **TensorFlow graph neural network**. In: Proceedings of the 2nd ACM SIGCAS Conference on Computing and Sustainable Societies, pp. 1–10. 2020, ACM.

[49] [WU et al. 2019] C. Wu, Y. Wang, X. Cui, Q. Wang, J. Pei und H. Zha. **Recommendation in heterogeneous information networks via graph attentional neural networks**. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 135–145. 2019, ACM.

[50] [XIAO et al. 2022] S. Xiao, S. Wang, Y. Dai und W. Guo. **Graph neural networks in node classification: survey and evaluation**. Machine Vision and Applications, Vol. 33:1–19, 2022.

[51] [XU et al. 2019] K. Xu, W. Hu, J. Leskovec und S. Jegelka. **How powerful are graph neural networks?** 2019, In: International Conference on Learning Representations (ICLR).

[52] [YADAV und JADHAV 2019] S. S. Yadav und S. M. Jadhav. **Deep convolutional neural network based medical image classification for disease diagnosis**. Journal of Big data, Vol. 6(1):1–18, 2019.

[53] [ZHOU et al. 2014] J. Zhou, F. Wang, J. Hu und J. Ye. **From micro to macro: data driven phenotyping by densification of longitudinal electronic medical records**. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 135–144.

[54] [ZITNIK et al. 2018] M. Zitnik, M. Agrawal und J. Leskovec. **Modeling polypharmacy side effects with graph convolutional networks**. Bioinformatics, Vol. 34(13):i457–i466, 2018.

# Declaration of Academic Integrity

I hereby declare that I have written the present work myself and did not use any sources or tools other than the ones indicated.

Datum:  .....

(Signature)