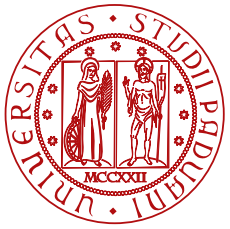


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Control Systems Engineering

A Graph Neural Networks approach to the state estimation of water distribution systems

Master Candidate

Giulia Tancon

Student ID 2054784

Supervisor

Prof. Maria Elena Valcher

University of Padova

Co-supervisor

Prof. Polycarpou Marios, Dott. Vrachimis Stelios

University of Cyprus

Academic Year
2022/2023
Graduation day 20/10/2023

To my firends and family

Abstract

Traditional Machine Learning cannot deal with graph data in a satisfactory way, in fact they are designed to work with simpler data types like images, which can be represented as grids. Graphs are more complex and they are used to model a set of objects (nodes) and the relationships between them (edges). Graphs do not have a fixed shape and each node can have a different number of neighbors; moreover the nodes inside a graph are interconnected and hence they are not independent. These are some of the reasons traditional machine learning algorithms struggle to work with this data. As a consequence Graph Neural networks were developed, with the goal of learning not only from the information encoded in the main elements of a graph, which are the nodes, but also from the connectivity among them, represented by the edges. Graph Neural networks can be applied to different tasks, such as semi supervised classification or regression, where the model learns to label or predict the target values of all nodes, given only a subset of them as training data.
The problem presented in this thesis falls under this category. A Graph Neural network is applied to a water distribution system (WDS) in an attempt to estimate the missing values of the pressures at those junctions where sensors are not present. In fact, these systems are often quite large and sensors too expensive to be installed at all locations; however, it is very important to estimate the pressures everywhere in the WDS, in order to monitor the state of the system and plan the appropriate control actions.

Contents

List of Figures

List of Tables

List of Acronyms

WDS Water Distribution System

WDN Water Distribution Network

NN Neural Network

FFNN Feedforward Neural Network

GNN Graph Neural Network

CNN Convolutional Neural Network

GCN Graph Convolutional Network

GCNN Graph Convolutional Neural Network

MPNN Message Passing Neural Network

ML Machine Learning

ReLU Rectified Linear Unit

GAT Graph Attention Network

NLP Natural Language Processing

MLP Multi Layer Perceptron

SiLU Sigmoid Linear Unit

1

Introduction

This project deals with the application of Graph Neural Networks to water distribution systems. Water distribution systems are fundamental infrastructures which supply clean and safe water to people and facilities, this way ensuring the sustainability of urban areas. For this reason it is of utmost importance to monitor their performance in order to ensure that there are no leakages that can lead to a significant loss of the transported water. Moreover, trying to understand the patterns of the demand of the various parts of the network can be significantly helpful when implementing control strategies that aim to provide a consistent and reliable water supply.
When trying to maintain the system in optimal operating conditions, it is extremely important to obtain sufficient and significant data that efficiently describes the state of the system. In particular, it has been shown that knowing the pressure values at the junctions of the water network can be sufficient to evaluate its performance, considering that from these same values it is also possible to infer the flow rates inside the pipes. Monitoring these parameters is helpful not only to detect anomalies, such as potential leaks or attacks, but also to develop control strategies that optimize the water network management.
However, collecting a sufficient amount of data across the entire water distribution network could turn out to be a too complicated challenge to tackle, because of the network's significant dimensions. In fact, the network size makes it impossible to deploy sensors at every node, because this would translate into very significant costs. As a result, most water distribution networks deploy sensors at only a fraction of their nodes, while the data at the others remains unknown.

This problem has inspired the development of algorithms to find the best possible configuration of sensors inside the water network. This configuration should contain enough elements to obtain a relevant amount of measurements, meanwhile keeping the cost as low as possible.
The goal of this thesis is to propose a strategy to estimate the missing pressure values within a water distribution system when only a subset of the nodes are equipped with sensors. Different techniques have been proposed over time to tackle this problem, but only recently the idea of employing graph neural networks in this field has been proposed.
Machine learning models, neural networks in particular, represent the state of the art in many different fields, such as computer vision, data classification and weather forecasting. However, these methods are tailored for data defined on Euclidean spaces, such as images. However, graphs are not defined in a Euclidean space, and they are characterized by the fact that a lot of information is actually stored in the connections between their elements.
To work with this particular type of data, Graph Neural Networks (GNNs) have been developed. These models are designed specifically to learn how to extract information not only from the nodes and edges present in the graph, but also from the relationships and connectivity patterns between these elements. The challenge in using graph-structured data in machine learning lies in its high dimensionality and non-Euclidean properties. Some of the first approaches that have been proposed to deal with this type of data, involve a first step in which the graph data is preprocessed. The aim is to reframe the graph structure into one which can work better with traditional machine learning models. However, GNNs introduce a new framework tailored specifically to work on data with a graph structure.
These models can be applied to solve different problems which can be at a graph-level, at a node-level or at an edge-level. When dealing with graph-level tasks, the aim of the model is to predict a single property for the entire graph. For example, if a molecule is represented as a graph, a GNN could be used to predict if it can bind to a certain receptor. In node-level tasks, instead, the model predicts some property for each node of the graph. For example, in the field of traffic forecasting if the the road network is modeled as a graph, where nodes represent monitoring stations and edges represent road segments, then a GNN could output some value at each station (node) representing the predicted traffic. Edge-level tasks require the model to predict some property for each link.

An example are recommendation systems which can predict how a user would like a target item based on the preference or rating information of many other users.
A particular subset of GNN are Graph Convolutional Neural Networks (GCNNs), which generalise Convolutional Neural Networks, often employed in image related tasks. While images can be seen as grids, which can be seen as specific types of graphs with regular connectivity patterns, generic graphs exhibit more complex and variable connections. Graph Convolutional Neural Networks bridge this gap by adapting the convolutional operations that characterize CNNs to irregular graph structures.
In particular, based on how the convolution is defined, GCNNs can be divided into two main categories: spectral-based and spatial-based approaches. Spectral-based models have a strong theoretical background based on graph signal processing; however, a significant drawback of this approach is that it can be computationally expensive and hence its use is limited to relatively small graphs. Spatial-based models do not have a strong theoretical background, but they are instead characterized by an intuitive structure which makes them operate on graphs in a way that is similar to how CNNs work on images. In fact they operate by directly aggregating the information contained at the neighborhoods of each node, making this operation localized in space. This property allows spatial GNN models to have a higher computational efficiency compared to spectral approaches, making them especially suited to operate on large graphs.
This master thesis aims to explore the application of different Graph Neural Network models on the state estimation problem discussed above. First, some general background on machine learning and graphs is provided, in order to highlight the challenges of working with graph data. Then the focus shifts specifically to the theory of graph neural networks and to the description of the models used in this project. Finally, the results of the estimation problems obtained with each of these models are presented.

1.1 Thesis structure

After the current introductory chapter, Chapter 2 presents a brief summary of the evolution of Graph Neural Networks. Furthermore, it describes state of the art techniques for the state estimation problem in water distribution systems. Chapter 3 gives some background knowledge about traditional machine learn-

ing tasks which do not involve graph data and then it presents some generic information on graphs and the corresponding notation, which are going to be needed in the next chapter. One of the focuses of this section is to describe how a generic neural network is trained and the challenges that the process may face. Moreover, it describes convolutional neural networks, which are a specific type of NN architecture often used in image related tasks. These models are not suited to operate on graphs but only on data which has the shape of a grid. However they are a good starting point in order to understand the idea behind GCNNs, which is going to be explained accurately in the next chapter. The last part of the chapter recaps the main notions and properties of graphs, describing their characteristic elements, and introduces some notation that will be used throughout the work.
Chapter 4 focuses on graph convolutional neural networks in general and in particular on the two main classes in which they can be divided. The theoretical background on the first class, spectral GCNs, is described in detail and three specific models are then presented. The second main class does not have a rigorous theoretical background however the main idea behind it is described in order to understand how these networks work. Furthermore, the specific architecture employed for this project is described, explaining its elements and how they use the information contained in the graph data to learn how to make estimates.
Chapter 5 presents the framework and tools employed in this project, focusing in particular on PyTorch Geometric and a specific software employed for hydraulic simulations. Moreover, it accurately describes the problem investigated, which is the pressure estimation in a water distribution system which is only partially observable. The dataset used for the task is then presented together with the way the proposed models have been built in PyTorch.
In Chapter 6 the training settings for each model and the results obtained with them are then described. Finally, Chapter 7 draws the conclusions of this work and suggests some possible improvements that can be implemented in order to get better results. Furthermore in this chapter it is highlighted how this estimation problem can be seen as just a starting point for many other tasks such as leakage detection.

2

State of the art

Water distribution systems (WDS) have a fundamental role in the society and it is of upmost importance to guarantee their correct functioning. It has been estimated in [20] that, on average, about 35% of the water produced in the world is lost during transportation, due to leakages present in the networks. For this reason, developing optimal management and monitoring techniques has become a topic of increasing interest. Many tasks regarding WDS have been proven to be successful provided that a sufficient amount of quality data is available in real time. For example in [15], [21] machine learning methods have been employed to forecast the water supply demands, which is important to guarantee users an adequate amount of water. The works [28] and [29], instead, focus on developing algorithms to detect and localize leakages inside a WDS or cyber-physical attacks against it - which is a rising problem considering the adoption of smart water technologies.

However, as stated above, these methods rely on the availability of an adequate amount of measurement data, in particular of nodal pressures inside the network. Given the extension of a WDS, it is impossible to install a sufficient number of sensors to have measurements of every part of the system. Many developments have been achieved in sensors technology, especially by integrating artificial intelligence approaches to traditional control strategies, as proposed in [24] . However, these techniques are not sufficient and there is the need to find a way to estimate the pressures and flow rates at all nodes and pipes, given the available measurements, i.e. techniques to deal with graph signals which are only partially observable.

The concept of graph signal reconstruction plays a crucial role in optimizing the placement of the sensors in the network. An efficient placement in fact leads to reduced operating costs by strategically utilizing only a subset of sensors to estimate signals across all the nodes of the network. Consequently, the need to complete partially observed signals and to optimally select sensors for effective signal reconstruction is a common topic of interest in the literature.
An overview of current trends in graph signal sampling and reconstruction is proposed in [27], which focuses in particular on a widely used linear regression approach to graph signals.

The literature exploring signal completion models specifically applied to water distribution systems (WDS) is however limited. For leak detection purposes, [35] proposes an algorithm that strategically distributes sensors across the WDS, by employing the average mutual coherence as a criterion to evaluate whether the measurements contain sufficient information to identify any potential leaks. In [33] an innovative methodology based on graph Fourier-transform is developed for optimal sensor placement. This approach achieves accurate temporal signal reconstruction by observing only 30% of the nodes, provided that the sensors are placed following a specific algorithm.
The work [23] presents a data-driven real-time method for nodal pressure reconstruction using a multi-layer perceptron (MLP), which is a commonly used NN architecture. Despite the promising results on small networks, applying this model to more realistic and complex scenarios does not lead to good results.
Given the recent renewed interest in graph neural networks, the idea of employing these architectures to deal with WDS has emerged. The new advances in graph signal processing, as highlighted in [26], have led in particular to two formulations of the convolution operator on graphs. One is directly defined in the node domain while the other in the graph spectral domain, which is the domain in which the Fourier transform of a graph is defined. These improvements enhance learning efficiency on irregular domains, which cannot be obtained through traditional ML tools, such as CNNs. An extensive survey on the graph neural networks architectures available to this day is provided in the work [34], while [8] conducts a benchmark study of popular GNN methods, enabling an easier comparison between the models studied.
In contrast to conventional artificial neural networks (NNs) , Graph Neural Networks (GNNs) have the distinct characteristic to gain information not only from node attributes but also from the interconnections among nodes. This unique

feature renders GNNs particularly suitable for learning tasks regarding water distribution systems (WDSs), where the topology of the network contains important information.
Graph convolutional neural networks can be divided in two classes: spectral and spatial GNNs. Spectral GNNs rely upon a good theoretical background on graph theory and they are based on the following idea. A signal associated to the graph is firstly converted to the spectral domain by the graph Fourier transform, then the convolution operation is performed and after the convolution, the resulting signal is converted back using the inverse graph Fourier transform. The first work on these models is [3], where spectral convolution on graphs is defined and used in order to build a graph neural network. After this, many other works on the topic were presented, such as [7] which proposes a model with the same computational efficiency as CNNs and [17] which proposes a spectral model to specifically deal with semi-supervised classification on graphs.
Various architectures for spatial GNNs have been proposed, each exhibiting distinct approaches to exchange and aggregate information contained in the nodes and edges ([2], [38]). Despite these differences, the underlying information exchange mechanisms share a common structure, which can be generalized in a process consisting of two phases, as described in ([10] and [16]). The first phase involves message passing, in which the latent nodes representations are updated based on the information received from neighboring nodes. This step is generally repeated at each layer of the model. The second phase, called readout phase, uses the information contained in the latent representations at the nodes to predict the desired output. For example, if the problem tackled is a binary classification of the nodes, then the sigmoid function is applied in order to determine to which class each node belongs to. Some important spatial GNNs which have been proposed are gated graph neural networks [19], graph attention networks [31] and graph recurrent neural networks [13].
As stated above, GNN have only recently been applied to WDS; for instance, [30] introduced a GNN-based architecture to identify cyber-physical attacks inside the water network. In [39] a GNN model which is able to detect water contamination is proposed. The first work on the estimation of water pressures using Graph Neural Network is [12], in which a simple spectral GNN model is employed, obtaining promising results. The same problem is then tackled in [1], where the same model proposed above is compared to a spatial graph neural network model, which appears to perform better on big networks. Another in-

teresting work is [36] which incorporate the violation of mass balance in the training process.
The work [12] has also been fundamental for [25], in which the state predicted by the model is used to detect leakages inside the network. This appears to be the first work which exploits GNNs not only for state estimation but also for leakage detection.

3

Machine Learning and Graphs background

3.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a branch of Machine Learning models which are inspired by the way the human brain works.
ANNs, often simply called Neural Networks (NNs), are computational models inspired by the structure and functioning of the human brain. ANNs consist of numerous interconnected processing units, each of them called neuron, which exchange information in order to solve specific problems. ANNs try to mimic the way in which biological neurons signal to each other. Neurons are, in fact, the fundamental computational units of the brain; they receive electric input signals via connections with other neurons, process them in the cell body and sum them up together. If the result is above a certain threshold, the neuron 'fires', transmitting a signal to other neurons.
One of the simplest type of NNs in use is the Feedforward Neural Network (FFNN), in which the information can only flow from the input to the output layers, without any form of feedback.
A FFNN is composed of three fundamental layers:

- Input layer: This initial layer receives the input data which is in vector form. The neurons in this layer are referred to as input neurons and their number is equal to the number of input variables in the data to process.

- Hidden layers: Situated in the middle of the ANN, the hidden layers perform diverse mathematical computations on the input data, trying to recognize underlying patterns. ANNs may have a single hidden layer (perceptron) or multiple hidden layers, as in Deep Neural Networks (DNNs), where the number of layers determines the network depth.
- Output layer: The output layer contains output neurons and provides the final prediction.

When a neuron receives signals from other neurons, it performs the following operations. First it weights each of the input signals with a certain parameter, which is the weight of the connection. After this, the weighted signals are summed and an additional value, called bias, is added to the sum. A specific neuron can take several weighted inputs, coming from other neurons, and sum them. If the result of this sum exceeds a certain threshold, defined by the activation function employed, then the neuron sends an output which can reach a different neuron. The way the neuron works can be summarized as:

$$y = f(\sum_{i=1}^{n} w_i x_i + b) \tag{3.1}$$

where y represents the output signal, n is the number of connections to the neuron and f is the activation function; w_i is the weight associated with the ith connection and b is the bias.
By simply putting together different layers, the FFNN is able to perform a lot of different tasks, such as data classification and function approximation. Their main limitation is that they can only work on data represented as vectors. A FFNN which contains more than one hidden layer is also referred to as Multi Layer Perceptron (MLP).

3.2 Convolutional Neural Networks

Feedforward neural networks, however, have a big limitation: when applied to image-related tasks, their performance is not satisfactory because one needs to reduce an image, which is a two dimensional structure, to a vector in order to feed it to a feedforward neural network. This results in losing a lot of information related to the spatial relations of the data. These limitations, together with the growing need to apply deep learning algorithms to computer vision tasks, has led to the development of a particular class of neural networks: convolutional

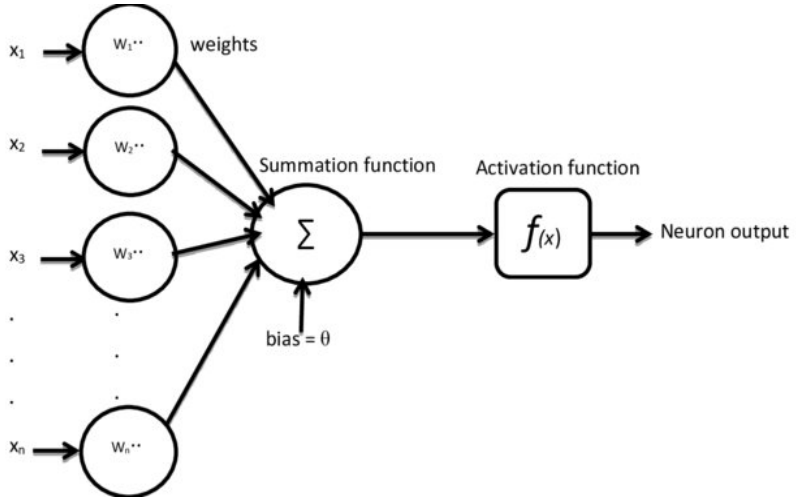


Figure 3.1: Operations performed by a single neuron

neural networks (CNNs).

These neural networks are specialized for working with grid-like structured types of data, such as images, which are represented as tensors. A tensor is a mathematical object which generalizes vectors and matrices and can be thought of as a multidimensional array. An image, in particular, is represented as a 3-dimensional tensor, which is a list of matrices having all the same dimensions. They can be either square or rectangular matrices, however the first ones are typically used to represent images. Each entry of a matrix corresponds to a pixel of the image and it has a certain value associated with it which is related to the intensity of a specific color. For grayscale images, this value represents the intensity of gray of that specific pixel. Colored images, like those in the RGB colorspace, are represented by a list of 3 matrices, each of them associated to the intensity of one of the following colors: red, green and blue. This grid-like structure, in which pixels are arranged in rows and columns, is the foundation for CNNs to perform tasks like image analysis and feature extraction effectively. A convolutional neural network is built by stacking together different blocks (or layers), and the main ones are:

- *Convolutional layer*: it has the task of performing image convolution, a linear operation which allows to extract significant features from an image. In the convolutional layer, a matrix which contains adjustable weights and is known as a ”kernel” or ”filter” is used. This kernel slides across the input image, which is represented as a grid of pixel values in the form of a tensor. At each position where the kernel overlaps with a portion of the image, element-wise multiplication between the pixel values and the corresponding weights in the kernel is performed. The results of these

multiplications are then summed, resulting in one element of the so called "feature map". This process is repeated across the entire image, generating the output tensor, called feature map, which captures important characteristics of the input data.
It is possible to perform on an image the convolution with multiple filters at the same time, and each of them can be seen as a unique feature extractor which focuses on specific characteristics of the input data. By applying different kernels, the CNN is able to capture a wider range of features, and consequently it can learn more complicated patterns and representations from the input data. The convolution operation described above is inspired by the 2D convolution for finite length signals. If one considers two generic finite length signals, s and w, then the convolution between them can be written as:

$$(s * w)(i, j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} s(i - m, j - n)\, w(m, n) \tag{3.2}$$

The operation, in case of images, takes the following form, in which $\mathbf{X}$ is the tensor representing the image and $\mathbf{K}$ is the kernel matrix.

$$\mathbf{K} * \mathbf{X}(i, j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \mathbf{X}(i + m,\ j + n)\, \mathbf{K}(n,\ m) \tag{3.3}$$

- *Pooling layer*: its aim is to reduce the spatial size of the tensors obtained from the previous convolutional layer, i.e. the feature maps. This operation, called downsampling, is obtained by taking small windows of the feature map and applying an operation to reduce the information within each window to a single value. In particular, average pooling makes an average of the values inside the window, while max pooling simply selects the bigger one. This process helps identifying the most important features in the data, while discarding less important details.

- *Activation layer*: since the convolution operation is linear, to introduce some non-linearities and enable the network to learn complex patterns, a non-linear activation function is applied to the feature map. There are different types of functions that can be used for this purpose, some common ones are the rectified linear unit (ReLU), the sigmoid or the hyperbolic tangent (tanh).

- *Output layer*: it is needed only for image classification problems. It is the final layer of the CNN and it takes the result of the convolution and pooling process and takes a classification decision, hence assigning a certain label to the image.

Of all the layers above, the convolutional ones perform the actual feature extraction starting from the image, while the others are designed to perform the specific task at hand, such as classification or regression. This means that the convolutional layers are built in the same way independently of the specific task considered, while the others are the ones that have to be designed based on the

problem requirements.
Convolutional neural networks in particular have three main strengths with respect to feedforward neural networks:

- *Local connectivity*: in CNNs, neurons in one layer are connected only to a small subset of neurons in the previous one, which are referred to as receptive fields. This localized connectivity allows CNNs to efficiently capture local patterns and features within the data. Unlike traditional FNNs, which have fully connected layers where each neuron is connected to all neurons in the previous layer, CNNs are more computationally efficient since each neuron interacts only with a limited number of neurons in the previous layer.
- *Weight sharing*: this property refers to the fact that during the convolution operation, the kernel slides through the whole image, while maintaining the same weights. This significantly reduces the number of parameters needed to train the network. As a consequence, CNNs are more computationally efficient and faster to train than FNNs with fully connected layers, making them the best choice when dealing with high-dimensional data like images.
- *Translational equivariance*: it is a key property which refers to the ability of the network to recognize the same feature regardless of where it appears in the input data. It is particularly useful in object detection tasks, in which the network must be able to recognize a specific object, independently of its location in the image. It is a natural consequence of the weight sharing property of CNNs.

These points of strength make CNNs exceptionally well-suited for tasks involving images, and for this reason they are especially used in the fields of image processing and computer vision. In these cases the spatial relationships, the local patterns, and the computational efficiency of the model have to be accurately considered. Overall, CNNs are efficient feature extractors which are capable of recognizing patterns in the data, while dramatically reducing the computational burden compared to traditional FNNs.

3.2.1 Training a network

Once a neural network is built, either a feedforward or a convolutional one, it needs to be trained in order to learn how to make accurate predictions. At the beginning, in fact, the weights of the different layers are initialized randomly, then in the training phase, the weights are iteratively adjusted in order to minimize a certain loss function.
Regression problems require the model to predict a quantity, for example the

weather temperature. A common choice for the loss function in these problems is the Minimum Squared Error, which computes the squared L2 norm between the predicted values and the corresponding target value. This loss measures how closely the predictions align with the actual values.
Classification problems require the model to assign data points to predefined classes, each of them identified by a label. If the classes are only two, then the problem is a binary classification one; if, instead, there are more than two classes, the problem is referred to as multi-class classification. For binary classification problems a common choice for the loss function is the binary Cross-Entropy, also called Log Loss. This metric is able to measure the extent of incorrect labeling of data as belonging to the right classes.
The optimization algorithm used to train a neural network is called Gradient Descent. It iteratively updates the learnable parameters of the network, such as biases and weights, in order to minimize the loss function, allowing the network to learn from data and refine its predictions. The name of the algorithm derives from the fact that the updates of the parameters are made along the direction given by the gradient of the loss function. The gradient, in fact, provides the direction along which the loss function has the steepest rate of increase. Each learnable parameter is updated along the negative direction of the gradient, which is the direction of steepest decrease, according to a step size determined by a hyperparameter called learning rate. The gradient of the function is computed over all the training examples in the dataset and than it is averaged over all the samples.
The key aspect of Gradient Descent is the update of the parameters of the model, the weights and biases, which can be stored in a vector, w. The first step of the algorithm requires the gradient of the loss function $\mathcal{L}$ to be computed with respect to the value of the parameters vector at the current iteration, w_t . Subsequently, the parameters are moved along the negative direction of the gradient according to the learning rate (η) in order to obtain the updated parameters vector w_{t+1}. In mathematical terms, this step can be formulated as:

$$w_{t+1} = w_t - \eta \, \nabla \mathcal{L} \, (\, w_t \,) \tag{3.4}$$

The learning rate plays a crucial role in the process of converging towards the minimum of the loss function, $\mathcal{L}$. A higher learning rate, in fact, leads to a faster training process but might cause the model to get stuck in a suboptimal set of

weights. A smaller learning rate, on the contrary, slows down the training process but allows the model to learn a more optimal or even globally optimal set of weights.
Gradient Descent algorithm encounters difficulties when dealing with non-convex functions, potentially trapping the optimization process in local minima rather than reaching the desired global minimum. To address this issue and to make the process more efficient, a variant of this algorithm, called Stochastic Gradient Descent (SGD), can be used instead. SGD updates the model parameters by evaluating the gradient of the loss function with respect to a single training data point during each iteration, which is randomly chosen. This method accelerates the computational process by handling one data point at a time, instead of the full dataset. This is particularly advantageous when dealing with a large number of data. However, it is important to note that this approach introduces some level of noise into the weight updates due to its stochastic nature.
A good compromise which is often used to train neural networks is the mini-batch Gradient Descent algorithm. In this approach, the gradient of the function is calculated based on a subset, known as a mini-batch, of the entire dataset. This method helps balancing computational efficiency and stability of the optimization process, since it exploits the benefits of both Stochastic Gradient Descent (SGD) and full-batch Gradient Descent algorithms. Even though it processes only a portion of the dataset at a time, it is able to mitigate some of the noise introduced by the stochastic updates of SGD. This makes it particularly well-suited for efficiently training neural networks on large datasets, helping to achieve convergence towards a good solution while managing computational resources effectively.
Deriving the loss function gradients with respect to each parameter contained in w, within a neural network is a non-trivial task. To solve this problem, the backpropagation algorithm has been proposed, which employs the chain rule of calculus to compute efficiently the gradient of the loss function with respect the parameters w. Once the gradient is available, the weights and biases are updated as in Eq. 3.4.
The updating process is repeated for multiple epochs, where an epoch corresponds to an entire passage of the training data through the algorithm. At each epoch the steps described above are repeated in order to adjust the weights and refine the model performance. The ultimate aim of the algorithm is the convergence toward a minimum point of the loss function, which signifies that the

network has learned to make accurate predictions. In particular, the training process does not stop until a satisfactory minimum is found or until a stopping criterion is met, in order to prevent a common phenomenon, called overfitting, which decreases the performance of the network.

3.2.2 Overfitting: causes and solutions

An effective model should demonstrate strong generalization capabilities, allowing it to perform well when applied to new, unseen data, as in the case, for instance, of the test dataset. However, a common problem arises when the model becomes too complex and starts to memorize noise or outliers present in the training data, rather than learning the underlying patterns that allow the model to generalize to unseen data. This problem manifests itself as the situation where the model achieves a low training loss, indicating strong performance on the training data, but at the same time it yields a high test loss, meaning that the model has a poor performance when confronted with data it hasn't encountered before. The main causes for this phenomenon are:

- model complexity: when a model has many learnable parameters, its capacity to approximate intricate relationships within the data increases. However, if the model becomes too complex, it might detect relationships in the data which are too intricate, since it focuses on capturing every minor variation or nuance present in the training data. As a consequence, the model learns almost perfectly to predict the training data but it is not able to generalise to unseen data.
- dataset dimension: when the dataset is relatively small in size, the model may encounter difficulties in effectively learning the underlying patterns present in the data. Instead, the model may attempt to fit each data point meticulously, effectively memorizing the training data itself rather than abstracting and generalizing the relationships between the data.
- noisy data: a machine learning model should be able to capture and understand the fundamental underlying patterns within a dataset, often referred to as the "signal". However, datasets often contain noise components, which bring irrelevant information, randomness and inconsistencies. This ability to effectively differentiate and disregard noise is essential to obtain a robust model which can generalize to unseen data.
- length of training: when the training process is excessively prolonged, the model can become overly specialized on the particular characteristics of the training dataset. The model adapts too closely to the training data and it begins to replicate not only the meaningful relationships within the data but also any irregularities, inaccuracies, or noise present in the training dataset.

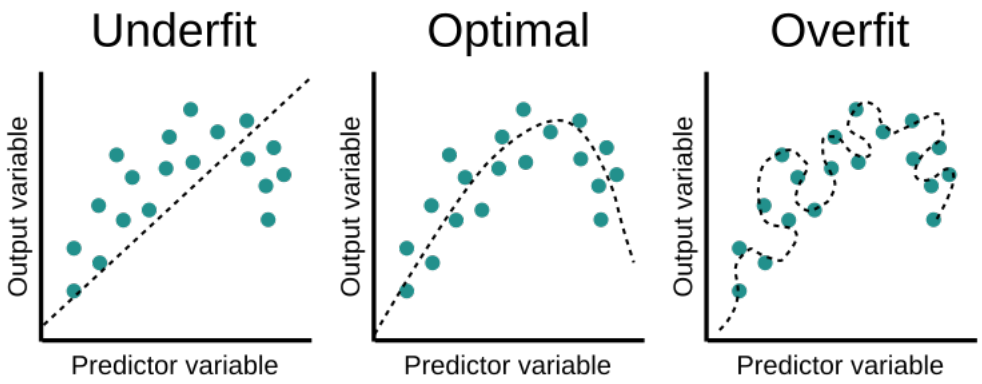


Figure 3.2: A visual explanatipon of the concepts of overfitting, underfitting and optimal fitting.

Some measures can be taken in order to avoid or at least mitigate the effect of overfitting. Early stopping is a rather simple one but still effective; in this case the validation loss is monitored throughout the training phase and when it stops improving, the training process is terminated. This prevents the model from becoming overly specialized on the training data and helps it achieve better generalization.
Another effective technique is to apply some kind of regularization, which constrains the network from learning a model that is too complex. In the regularization process, an additional penalty term is summed to the cost function to prevent the weights from becoming too large. Two regularization techniques are commonly used:

- $L1$ regularization: it encourages some of the model weights to become exactly zero, hence making the weight vector sparse and leading to feature selection. This helps identifying and emphasizing the most important features, while discarding less relevant ones.
- $L2$ Regularization: it encourages the model to have smaller weights but it never forces any of them to become exactly zero. This helps controlling the model overall complexity and prevents any single weight from dominating the others.

3.3 Graphs and notation

Many real-world data-sets can naturally be represented as graphs, which are structures made of nodes and edges. For example, on online platforms such as

social networks, users can be represented as nodes, and follows or likes can be represented as edges. To better understand the peculiarities of this kind of data structure, this section introduces the definition of graph, its main components and its properties.

A graph is a pair $G = (V, E)$, where V is the set of nodes, while $E \subseteq V \times V$ is the set of edges. Typically we denote by n the number of nodes in the graph, namely $|V| = n$, and by m the number of edges, i.e., $|E| = m$. A specific node in the set V is denoted by $v_i \in V$, while an edge is generally indicated by the initial and terminal node, for example $e_{ij} = (v_i, v_j) \in E$.

If the edges have an orientation, which is commonly represented with an arrow, then the graph is said to be *directed*, otherwise it is *undirected*. The set E describes how the nodes are connected, but this information can also be encoded in a formal mathematical way through an object called *adjacency matrix*.

This matrix has dimension nxn and its specific element A_{ij} is equal to one when the two corresponding edges are connected and otherwise it is zero. Specifically:

$$A_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{if } e_{ij} \notin E \end{cases} \tag{3.5}$$

This particular adjacency matrix is referred to as "binary" because it represents the nodes connections in a binary fashion, meaning that connections are either present (encoded as 1) or absent (encoded as 0). Nonetheless it is also possible to consider weighted adjacency matrices, where some kind of weight is associated to the edges, in general representing some sort of cost between connected nodes. It follows from Eq. 3.5 that directed graphs possess symmetrical adjacency matrices, so $A_{ij} = A_{ji}$ for all $i, j \in V$, while undirected ones do not, so $A_{ij} = A_{ji}$ is not necessarily true for all $i, j \in V$. From now on we will consider only undirected graphs, since the data that will be used in this thesis is represented by undirected graphs. For each node i we define the *vertex degree*, d_i, which is the number of edges that are incident to that specific node:

$$d(v_i) = \sum_{j=1}^{n} e_{ij} \tag{3.6}$$

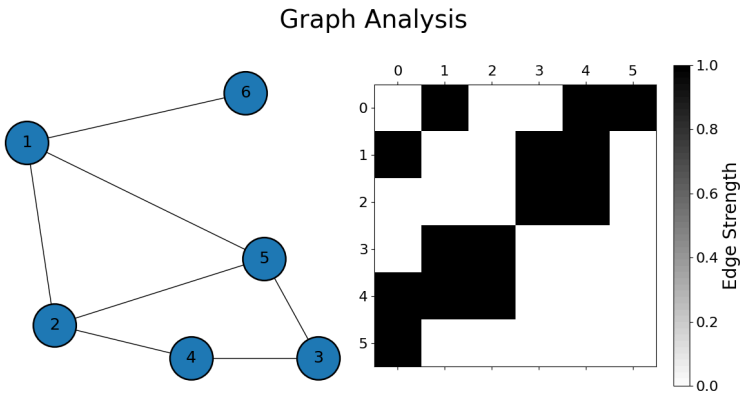


Figure 3.3: Example of an undirected, unweighted graph and the corresponding binary adjacency matrix

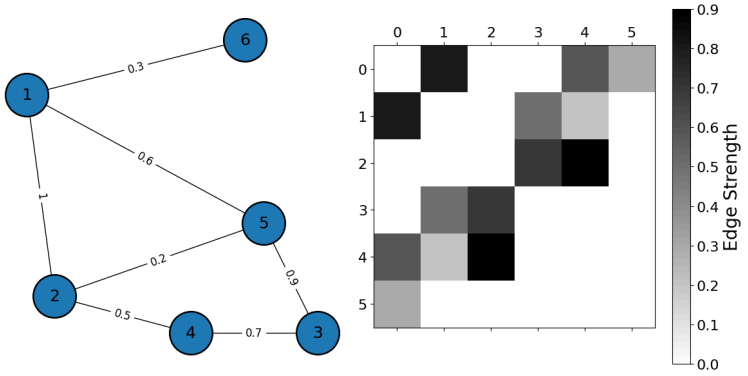


Figure 3.4: Example of an undirected, weighted graph and the corresponding weighted adjacency matrix

Figure 3.5: Main Caption for the Combined Images

The *node degree matrix* $D \in \mathbb{R}^{nxn}$ is a diagonal matrix defined as follows:

$$D_{ij} = \begin{cases} d_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

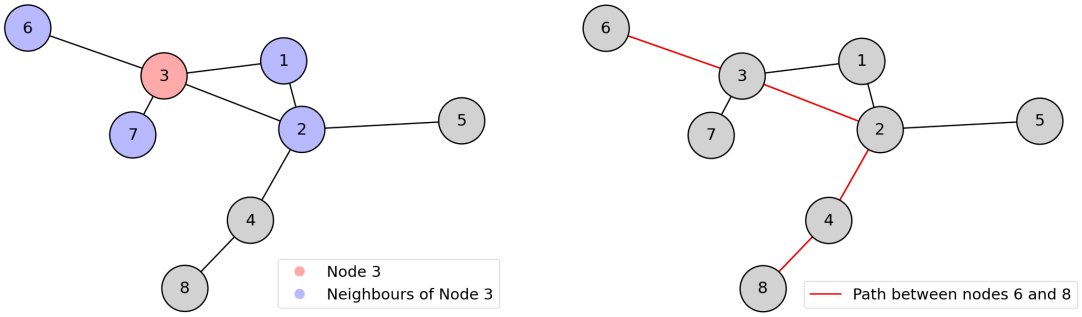
It follows that for an undirected graph, the node degree is equivalent to the number of neighboring nodes of the specific node. The adjacency matrix and node degree matrix are used to define the Laplacian matrix, accurately described in the next subsection.

In general, some properties are associated to the nodes, called node features, which can be encoded in the *node feature matrix* of the graph. The features associated to each node can be either scalar values or vectors; we consider the generic case in which to each node $v_i \in V$ is assigned a row vector of features, $X_i \in \mathbb{R}^{1xd}$. It follows that the node feature matrix associated to the graph is $X \in \mathbb{R}^{nxd}$. Analogously, also edges can have feature vectors associated with them. Let us consider a generic edge $e_{ij} \in E$. We associate to it the edge feature vector $X_i^e \in \mathbb{R}^{1xl}$.

The *edge feature matrix* of the graph is then represented as $X^e \in \mathbb{R}^{mxl}$.
The *neighborhood* of a node is defined as the set of edges which are connected to it, formally:

$$\mathcal{N}(v) = \{u \in V \mid (v,u) \in E\} \tag{3.8}$$

Two nodes are said to be connected if there is an edge between them. A path is a



(a) First Image (b) Second Image

Figure 3.6: In (a) the neighbours of node 3 are highlighted in blue; in (b) an example of a path between node 6 and 8 is shown in red

sequence of consecutive edges, meaning that the starting node of an edge is the ending node of the previous edge in the path. Paths can be of various lengths and can form cycles, meaning that the starting and ending node coincide.
We would like to define a distance between the vertices in a graph, however this concept is difficult to extend to the graph domain.
In an Euclidean space S, the distance between two points is defined in a rigorous way as the function $d : S \times S \longrightarrow \mathbb{R}_+$ such that:

$$d(x,y) = \begin{cases} d(x,y) > 0 & \text{for all } x \neq y, \\ d(x,y) = 0 & \text{if and only if } x = y, \\ d(x,y) = d(y,x) & \text{for all } x, y \\ d(x,y) \leq d(x,z) + d(z,y) & \text{for all } x, y, z. \end{cases} \tag{3.9}$$

In graph theory, the 'distance' between two vertices in a graph is defined in the following way:

$$\text{dist}(v_a, v_b) = \min \sum_{e \in \text{path}} w_e \tag{3.10}$$

where w_e is the weight associated with the edge e, which in the case of unweighted graphs is 1. This distance is also called geodesic distance or shortest-

path distance.
The distance between two vertices, as defined above, does not necessarily satisfy the properties in Eq. 3.9. For example, if a graph is directed, there may be nodes a , b such that $d(a, b) \neq d(b, a)$; if, instead, a weighted graph is considered, then it might happen that $d(a, b) < 0$ for some nodes a , b , if the weights at some edges are negative.
This graph 'distance' is a useful instrument to measure the connectivity or reachability between vertices in a graph, but it cannot be used as a geometric distance. In graph theory, distance is a more abstract concept related to connectivity, while geometric distance in a Euclidean space has a more intuitive geometric interpretation.
Another important concept to define is the *diameter of a graph*:

$$\text{diam}(\mathcal{G}) = \max(dist(v_i, v_j), \text{ for each } v_i, v_j \in V) \tag{3.11}$$

which in the case of unweighted graphs, simply represents the length of the shortest path between the two furthest nodes in the graph.

3.3.1 Laplacian matrix

In this subsection we introduce the Laplacian matrix and describe some of its properties. This mathematical object is in fact essential to build spectral Graph Convolutional Neural Networks which are described in the next chapter. This matrix is going to be defined only for undirected graphs, since they are the ones of interest in this thesis.
The Laplacian matrix of a undirected graph $\mathcal{G}$ is defined as:

$$L = D - A \tag{3.12}$$

where D is the degree matrix, as defined in Eq. 3.7, and A is the adjacency matrix as defined in Eq. 3.5. It follows that L is a matrix of dimension nxn. Given that both the adjacency matrix and the node matrix are symmetric, also the Laplacian is symmetric.
When considering a graph $\mathcal{G}$, it is common for its nodes to exhibit significant variations in their degrees. The presence of a heavy node, characterized by a large degree, can result in a very big diagonal entry in the matrix, which may

dominate the matrix overall properties. To mitigate this imbalance it is possible to normalize the Laplacian matrix. For a graph with no isolated vertices [1], two different normalizations are proposed:

- Random-walk normalization: which is the less common one and is defined as
$$\bar{L} = I - D^{-1}A \tag{3.13}$$
- Symmetric normalization: the most used one, defined as
$$\bar{L} = D^{-1/2}LD^{-1/2} = I - D^{-1/2}AD^{-1/2} \tag{3.14}$$

where $\bar{L}$ identifies the normalized Laplacian. Both normalized versions of the Laplacian have the property that their eigenvalues lay in the interval $[\,0\,,\,2\,]$, as shown in [22]. The symmetric normalization gives a symmetric matrix, obviously, while the random-walk normalization does not in general. In the rest of this work the term normalized Laplacian is going to be used referred to the Laplacian obtained through symmetric normalization.
The Laplacian has the characteristics described in the following theorems.

Theorem 1 *L is symmetric and positive semi-definite, i.e.*

$$L = L^{\top} \geq 0 \tag{3.15}$$

From the symmetric property it follows that the eigenvalues of the matrix are real, while the positive semi-definiteness guarantees that all the eigenvalues are non negative.

Theorem 2 *Let $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the eigenvalues of the Laplacian matrix associated to the graph $\mathcal{G} = (V, E)$. The multiplicity of $\lambda_1 = 0$ corresponds to the number of connected components [2] in the graph.*

[1] A isolated vertex is a vertex of a graph which has no edges connected, hence it has zero degree.

[2] A connected component of an undirected graph is a subset of nodes in which every pair of nodes is connected to each other through a path.

4

Graph Convolutional Neural Networks

Convolutional Neural Networks are models designed to deal with data that naturally lie in euclidean spaces and are organized in grid-like structures. However, many real-world scenarios are better represented through data structures that are naturally defined in non-Euclidean spaces, such as graphs. For example road networks can be represented as graphs, in which certain specific locations are the nodes, and the roads connecting them are the edges. It would be very difficult to accurately represent the data above through a different data structure without losing information about the connectivity of the nodes; however they can be naturally represented as graphs.

Graph Convolutional Neural Networks (GCNNs) originated from the need of generalizing CNNs to work on data structured like graphs (graph data). This, however is not an easy task, since the convolutional layers, which are the fundamental parts of CNNs, are built to work on data structured as regular grids and not on graphs which have an irregular structure.

A GCNN can be thought of as a sequence of differentiable operations that take as input a graph signal, the node feature matrix X, and return a signal containing some property of the graph itself or of its nodes. A GCNN is in general constituted by convolutional layers, pooling layers and readout layers.

Convolutional layers use the attributes of the input graph, which can be in general both the node and the edge feature matrices, and the topology of the graph, encoded in the adjacency matrix, in order to compute a new representation of

the nodes features, X', called latent representation.
Pooling layers work by reducing the number of nodes in a graph; they are not fundamental parts of the architecture and they are not needed in the models proposed, for this reason they are not investigated further.
Readout layers are needed only when dealing with graph-level problems, in which we desire to output a property for the entire graph, for example when we want to classify a graph. The problem tackled in this work, however, is defined at the node-level, since we want to predict a value for each node, and for this reason the readout layer is not needed.
The architecture of a GCNN is obtained by combining these layers, which, as said above, perform differentiable operations. This property allows to train the network using the backpropagation algorithm and gradient descent.
Based on how the convolutional layer is defined, the literature distinguishes two main categories of GCNNs [34]: spectral-based GCNNs and spatial-based GCNNs. Some general information on the two different classes is given here, however they will be described in more detail in the next sections.
GCNNs built following a spectral approach have a solid mathematical foundation in graph signal processing, as stated in [5] and [37]. These models operate by performing convolutions in the spectral domain, as the convolution between two signals is achieved through the simple multiplication of their Fourier transforms. However, there are some important limitations to the models which follow this approach: they can be applied exclusively to undirected graphs and face scalability issues. They are, in fact, based on the eigen-decomposition of the Laplacian matrix, whose dimensions increase with the number of nodes within the graph; for this reason, spectral GCNNs are not suitable for large scale graphs.
Spatial GCNNs, on the other hand, lack a well-established theoretical foundation but offer a more intuitive approach. They try to extend the conventional convolution operator of CNNs (see Fig. 4.1) to work with graphs, by aggregating the information from the neighbouring nodes and using them to update the current state of the node. They are often referred to as Message Passing Neural Networks since they define the convolution on graphs as an exchange of information (messages) between neighbouring nodes. This approach, although less mathematically rigorous, can be more versatile and easier to implement, making it a practical choice for various graph-related tasks, especially when applied to large graphs.

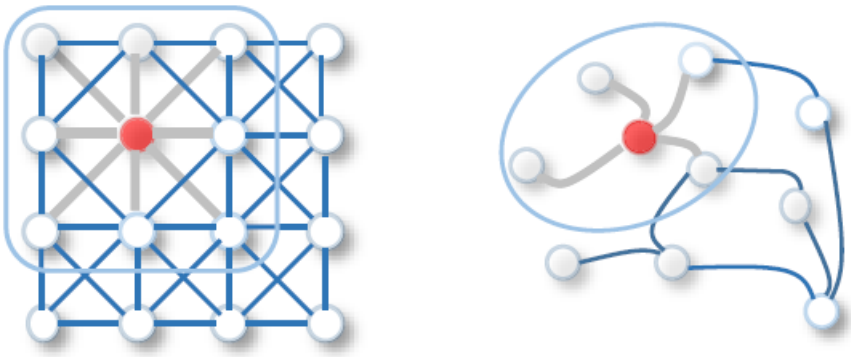


Figure 4.1: Visual representation of the similarities between the spatial convolution on graphs and the image convolution. In (a) the pixels of the image are represented by the nodes. The neighbours of the node are determined by the size of the sliding filter (in this case 3x3), and they are ordered and have a fixed size, given the size of the filter. The convolution consists in taking the weighted average of pixel values of the red node and of its neighbours. In (b) a possible way to define the graph convolution with the spatial approach is represented. This operation on the red node consists of taking the average value of the node and of the neighbouring nodes features. In the case of graphs, each node can have a different number of neighbouring ones and the nodes are unordered.

4.1 Spectral Graph Convolutional Neural Networks

As mentioned above, in spectral GCNNs, the convolutional operation is performed in the spectral domain of a graph. This approach takes advantage of the convolution theorem, which states that the convolution in the time domain between two signals is equivalent to a multiplication in the frequency domain. In particular, given two generic signals f, g in the time domain, it holds that:

$$f * g = \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}(g)) \tag{4.1}$$

The convolution of f and g is equivalent to the multiplication of their Fourier transform ($\mathcal{F}$), which is later re-transformed in the original time domain via the inverse Fourier transform ($\mathcal{F}^{-1}$). In the context of graphs, graph signals are defined in the node domain instead, but the theorem still holds.
In particular, spectral GCNN models, built following this approach, leverage the spectral decomposition of the Laplacian matrix in order to perform convolutions in the Fourier spectral domain. The graph Fourier transform of a graph signal, in fact, can be derived starting from the Laplacian matrix of a graph. In GCNNs we are interested in the convolution of a filter containing learnable weights with the graph signal, which is the node feature matrix $\boldsymbol{X}$.
In order to define the graph Fourier transform of a graph signal we need to first derive the eigenvalue decomposition of the Laplacian.
Let $L \in \mathbb{R}^{n \times n}$ be the Laplacian matrix of a graph $\mathcal{G}$, let $\mathrm{U} = [\, u_1\,,\ u_2,\, \ldots\,,\, u_n\,] \in \mathbb{R}^{n \times n}$ be a basis for $\mathbb{R}^n$ consisting of eigenvalues of the matrix L, where each column u_i represents an orthonormal eigenvector of the matrix. Then the spectral decomposition of the Laplacian can be written as:

$$L = \mathrm{U}\,\Lambda\,\mathrm{U}^{-1} = \mathrm{U}\,\Lambda\,\mathrm{U}^{\top} \tag{4.2}$$

where Λ is the diagonal matrix of the eigenvalues defined as:

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} = \mathrm{diag}\,([\,\lambda_1, \lambda_2, \ldots, \lambda_n\,]) \in \mathbb{R}^{n \times n}. \tag{4.3}$$

The graph Fourier transform (GFT) projects the graph signal into the space

defined by the orthonormal basis formed by the eigenvectors of the graph Laplacian ([6]).
The graph Fourier transform of a generic graph signal $X \in \mathbb{R}^{n \times d}$ is formally defined as:

$$\mathcal{F}(X) = \hat{X} = \mathrm{U}^\top X \tag{4.4}$$

where U is the aforementioned transformation matrix, while the inverse transform is:

$$\mathcal{F}^{-1}(\hat{X}) = X = \mathrm{U}\,\hat{X} \tag{4.5}$$

where $\hat{X}$ is the signal represented in the spectral domain. Given this definition of graph Fourier transform, and considering the convolution theorem, it is now possible to define the convolution between two signals $X, G \in \mathbb{R}^{n \times d}$ as:

$$X * G = \mathrm{U}\,(\,\mathrm{U}^\intercal X \odot \mathrm{U}^\intercal G\,) \tag{4.6}$$

where $\odot$ is the Hadamard (element-wise) product.
Traditionally, signal filtering is described as a convolution between the signal to be filtered and a filter, for this reason Eq. 4.6 can be considered as a graph filtering operation. To better understand how graph filtering works, let us consider two one-dimensional graph signals $x, g \in \mathbb{R}^n$, where g represents the filter. We can consider signals of dimension n instead of $n \times d$ without loss of generality. The filtering of a generic signal x can be written as:

$$\begin{aligned} x * g &= \mathrm{U}\,(\,\mathrm{U}^\top x \odot \mathrm{U}^\top g\,) = \mathrm{U}\,(\,\mathrm{U}^\top g \odot \mathrm{U}^\top x\,) \\ &= \mathrm{U}\,(\mathrm{diag}\,(\mathrm{U}^\top g)\mathrm{U}^\top x) = \mathrm{U}\,\mathrm{diag}\,(\mathrm{U}^\top g)\mathrm{U}^\top x \end{aligned} \tag{4.7}$$

By remembering that $L = \mathrm{U}\,\Lambda\,\mathrm{U}^\top$ and noting that both Λ and $\mathrm{diag}\,(\mathrm{U}^\top g)$ are diagonal matrices, we can think of graph filtering as an operation on the eigenvalues of L, and hence on the matrix Λ; hence, it holds that

$$g(\Lambda) = \mathrm{diag}\,(\mathrm{U}^\top g) \tag{4.8}$$

The derivation above is valid also for multi-dimensional signals, so the general graph filtering operation can be written as:

$$X * G = \mathrm{U}\,g(\Lambda)\mathrm{U}^\top X \tag{4.9}$$

All spectral-based Graph Neural Networks adhere to this definition of convolution. These models use filters, $g_\theta(\Lambda)$, parametrised by some weights θ which are learned in the training phase. The various implementations proposed in time differ based on the choice of parametrisation of the filter.
To sum up, the convolution of the node feature matrix $\boldsymbol{X}$ and a filter containing parameters to be learned can be written as:

$$X * G_\theta = \mathrm{U}\, g_\theta(\Lambda)\mathrm{U}^\top X \tag{4.10}$$

The first spectral GCNN with parametric filters implemented was the one proposed in [4]. The architecture is designed to work with undirected, either binary or weighted graphs. This approach, however, has the main drawback that the convolution as defined in Eq. 4.8 requires a double multiplication for the basis of eigenvector U; this process is computationally expensive, especially for large graphs.
An approach which solves this problem consists in approximating the filter g through various polynomial, with coefficients to be learned. In this case we are talking about employing polynomial filters. Two models which use this approach are presented in the next section.

4.1.1 Spectral GCNN with polynomial filters

In an attempt to solve the computational cost issue, [7] proposed the use of the following polynomial filters:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \tag{4.11}$$

where $\theta_k, k = 10, 1, \ldots, K-1$, are the polynomial coefficients which need to to be learned, and can be thought as a v forming vector $\theta \in \mathbb{R}^K$.
Using this filter allows to avoid the eigenvalue decomposition of the Laplacian. In fact, it holds that:

$$\mathrm{U}\, g_\theta(\Lambda)\mathrm{U}^\top = g_\theta(\mathrm{U}\Lambda\mathrm{U}^\top) = g_\theta(L) \tag{4.12}$$

Indeed, if we assume that

$$g_\theta(\Lambda) = \theta_0 I + \theta_1 \Lambda + ... + \theta_{K-1}\Lambda^{K-1} \tag{4.13}$$

and multiply this quantity by U and $U^\top$, we obtain:

$$\begin{gathered} U g_\theta(\Lambda) U^\top = U(\theta_0 I + \theta_1 \Lambda + ... + \theta_{K-1}\Lambda^{K-1})U^\top = \\ \theta_0 I + \theta_1 L + ... + \theta_{K-1} L^{K-1} = g_\theta(L) \end{gathered} \tag{4.14}$$

As a consequence of Eq. 4.12, it is now possible to apply the filter directly to the Laplacian entries, avoiding the multiplication by U and $U^\top$.
Another advantage of using these filters is that the number of parameters to be learned only depends on the number of polynomial coefficients K and not on the dimension of the input graphs, so the learning complexity is $O(K)$, the same as traditional CNNs. This generic polynomial filter has solved many issues, however models with more specific polynomial filters have been proposed which have shown better performances.
For example, the model proposed in [7] uses the the Chebyshev polynomials, which can be computed recursively, to approximate the graph spectral filter; for this reason is commonly called ChebNet. The spectral filter assumes the following form:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) \tag{4.15}$$

where

$$\tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - I_n \tag{4.16}$$

is the matrix of scaled eigenvalues that lie in $[-1, 1]$, θ_k are the learnable Chebyshev coefficients and $T_k(\tilde{\Lambda}) \in \mathbb{R}^{nxn}$ is the Chebyshev polynomial of degree k evaluated in $\tilde{\Lambda}$. These polynomials can be computed following the recursive algorithm:

$$\begin{aligned} T_0(\tilde{\Lambda}) &= I, \\ T_1(\tilde{\Lambda}) &= \tilde{\Lambda}, \\ T_{k\geq 2}(\tilde{\Lambda}) &= 2\,\tilde{\Lambda}\,T_{k-1}(\tilde{\Lambda}) - T_{k-2}(\tilde{\Lambda}) \end{aligned} \tag{4.17}$$

The filtering operation between two signals of dimension n, becomes:

$$\begin{aligned} y = g_\theta * x = \mathrm{U} g_\theta(\Lambda)\,\mathrm{U}^\intercal x \\ = \sum_{k=0}^{K-1} \theta_k \mathrm{U} T_k(\tilde{\Lambda})\,\mathrm{U}^\intercal x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L}) x = \sum_{k=0}^{K-1} \theta_k \bar{x}_k \end{aligned} \tag{4.18}$$

where $\bar{x}_k = T_k(\tilde{L})x$ and these quantities are computed recursively using:

$$\begin{aligned} \bar{x}_0 &= x \\ \bar{x}_1 &= \tilde{L}x \\ \bar{x}_k &= 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2} \end{aligned} \tag{4.19}$$

with

$$\tilde{L} = \frac{2\Lambda}{\lambda_{max}} - I_n \tag{4.20}$$

The setting above can be generalised to multi-dimensional graph signals; in particular, the ChebNet model transforms the node feature matrix $X \in \mathbb{R}^{n \times d}$ into $X' \in \mathbb{R}^{n \times d'}$ as:

$$X' = \sigma\Big(\sum_{k=0}^{K-1} \bar{X}_k \Theta_k\Big) \tag{4.21}$$

where $\Theta_k \in \mathbb{R}^{d \times d'}$ and $\bar{X}_k = T_k(\tilde{L})X$.
Also in this case the filtering operation does not rely on the computation of the eigenbasis U, resulting in a lower computational complexity.
The GCN model proposed in [17] employs a first order approximation of the Chebyshev filters. It reduces the computational complexity by considering only first order polynomials, i.e. by imposing $K = 2$. As a consequence, each convolutional filter has only two parameters to learn. The filter in this case is linear in $\tilde{\Lambda}$ and hence in $\tilde{L}$:

$$g_\theta(\Lambda) = \sum_{k=0}^{1} \theta_k T_k(\tilde{\Lambda}) = \theta_0 + \theta_1 \tilde{\Lambda} \tag{4.22}$$

In the model proposed in the paper the approximation $\lambda_{max} \approx 2$ is made, obtaining then

$$\tilde{L} = \frac{2}{\lambda_{\max}} L - I_n \approx L - I_n \tag{4.23}$$

in which L is the symmetric normalized Laplacian matrix as in (3.14). Then we can write

$$\begin{aligned} y = g_\theta * x = \sum_{k=0}^{1} \theta_k T_k(\tilde{L})\, x = \theta_0 x + \theta_1 \tilde{L} x \\ = (\theta_0 + \theta_1(L - I_n))\, x = (\theta_0 - \theta_1(D^{-1/2} A D^{1/2}))\, x \end{aligned} \tag{4.24}$$

where the last passage comes from inverting Eq. 3.14, hence obtaining $L - I_n = -D^{-1/2} A D^{1/2}$. To further reduce the number of parameters inside the network, which helps mitigating overfitting and reduces the computational complexity, an additional constraint can be introduced. In particular, the authors of the paper impose $\theta = \theta_0 = -\theta_1$, hence obtaining:

$$y = g_\theta * x = \theta(I_n + D^{-1/2} A D^{1/2}) \tag{4.25}$$

In this way only one parameter, θ, has to be learned at each convolutional layer. However, it has been shown that performing this operation multiple times can lead to the problem of vanishing gradient, hence making the NN perform in a worse way. To avoid this, a simple renormalization operation can be performed on the adjacency matrix, by simply adding self loops $\tilde{A} = A + I_n$ and assuming $\tilde{D}_{i,i} = \sum_{j=1}^{n} \tilde{A}_{i,j}$, thus obtaining

$$g_\theta * x = \theta(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}) x \tag{4.26}$$

The passages above hold also for the general case of multi-dimensional signals. In particular, given $X \in \mathbb{R}^{nxd}$, the matrix of nodes features, then it is possible to compute the graph signal value at the next layer of the GCN using:

$$X' = \sigma(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X \Theta) \tag{4.27}$$

where $\Theta \in \mathbb{R}^{dxd'}$ is the matrix containing the parameters of the filter to be learned and σ is the non-linearity applied. The great strength of the GCN model is that, given the lower number of parameters needed at each layer, a larger number of convolutional layers can be employed.

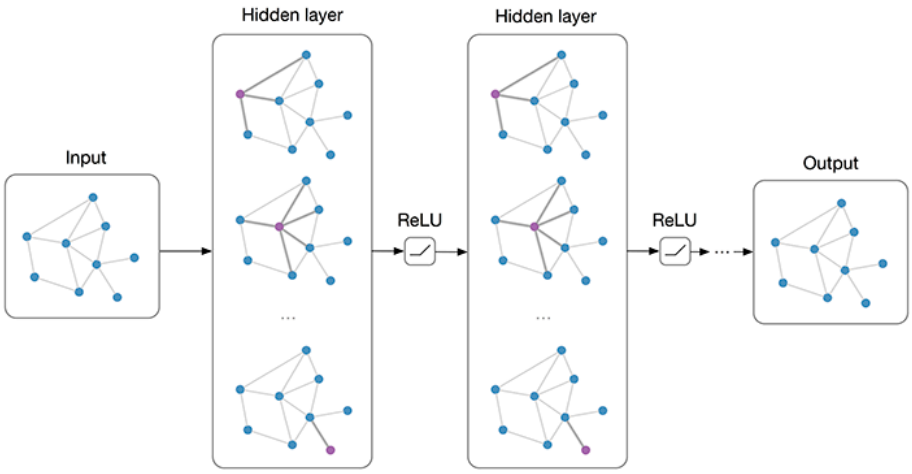


Figure 4.2: Architecture of a generic multi-layered GCN, the non-linear activation function in this example is a ReLU, as proposed in [17].

4.2 Spatial Graph convolutional networks

Spatial GCNNs are inspired by traditional CNNs and operate directly on the graph spatial domain, where the nodes are defined in a similar way to how CNNs process data graph are organized in grids.
In fact, models defined using spatial-based approaches define graph convolutions based on the spatial relations of the nodes. In images, the convolution is based on the computation of a weighted sum of neighboring pixel features. An image can be considered a graph in which the nodes are the pixels, and the nodes have a fixed number of neighbours. In a similar way, in spatial GCNNs the convolution is formulated as an exchange of information between neighbouring nodes, however in this case the neighborhood size is not fixed. The key idea is to use the graph structure to define a local neighborhood for each node and update the node current feature based on those of its neighbors. This approach is computationally efficient and allows GCNNs to handle graphs of varying sizes and structures. By focusing on local neighborhood aggregation, there is no need for global operations on the entire graph, like the eigenvalue decomposition of the Laplacian. Because of the way they work, these models are also called Message Passing Neural Networks (MPNN).
To sum up, the key difference between spatial and spectral graph neural networks lays in how the convolution operation is defined. Spatial GNNs directly operate on the graph spatial domain (node domain) to gather information from neighboring nodes which is used to update the current node representation.

Spectral GNNs, instead, perform the convolution in the graph spectral domain, hence using directly the Laplacian matrix.
Different kind of spatial models have been proposed, such as the Graph Attention Network model [31] or the Graph SAGE [14]. MPNN normally work with

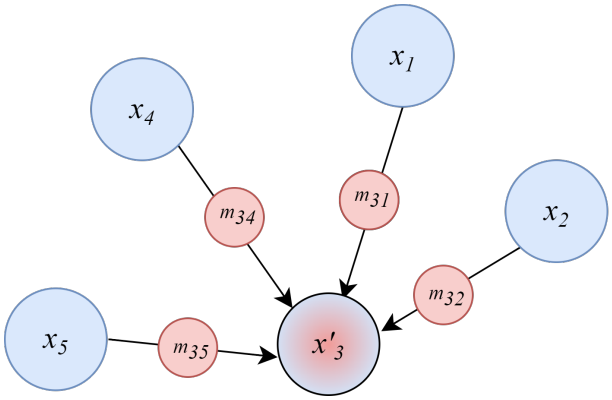


Figure 4.3: The update of the representation of node 3 uses both the representations of the neighbours (blu elements) and the messages of the edges (red elements)

undirected graphs but they can be generalised to work also with directed ones. Each layer of an MPNN follows three important steps: message creation at the nodes neighbours, aggregation of the messages and update of the current node features. These three steps are now described in more detail.
The message creation step uses a message function f_m, which is often modeled by an MLP (an FFNN with more than one layer), to compute for each edge e_{ij} a message m_{ij}, using both the features of the nodes and of the edges:

$$m_{ij} = f_m(h_i, h_j, e_{ij}) \tag{4.28}$$

Here h_i and h_j are the features of the generic nodes i and j, while e_{ij} is the feature of the corresponding edge. Once the messages have been built, they are used to update the feature of the nodes. Let us consider a specific node u, then in the first place the messages of the neighbors $\mathcal{N}(u)$ are aggregated via a permutation-invariant function, which in most cases is either a summation, as in Eq. 4.29, or an average, as in Eq. 4.30. The result is a message associated with the node u, named m_u. At the generic layer $k+1$ of the NN, the message associated to node

u is created as follows:

$$m_u^{(k+1)} = \sum_{v \in \mathcal{N}(u)} f_m(h_u^{(k)}, h_v^{(k)}, e_{uv}) \tag{4.29}$$

or

$$m_u^{(k+1)} = \frac{1}{|\mathcal{N}(u)|} \sum_{v \in \mathcal{N}(u)} f_m(h_u^{(k)}, h_v^{(k)}, e_{uv}) \tag{4.30}$$

In the second place, an update function, modelled in general by a different MLP, f_u, is used to update the features of the specific node starting from the current feature of the node $h_u^{(k)}$ and the message $m_u^{(k+1)}$ which has been created at the previous step:

$$h_u^{(k+1)} = f_u(h_u^{(k)}, m_u^{(k+1)}) \tag{4.31}$$

All the previous passages can be summarized in the following equation which updates the feature of the node u from layer k to layer $k+1$:

$$h_u^{(k+1)} = \phi\,(\, h_u^{(k)}\,,\, \oplus_{v \in \mathcal{N}(u)}\, \psi(\, h_u^{(k)}\,,\, h_v^{(k)}\,,\, e_{uv}\,)) \tag{4.32}$$

where ϕ is the update function, $\oplus$ is the aggregation scheme and ψ is the message function. This process described in (4.32) describes how the message passing layer works. This framework has been used in order to define the spatial GCNN used in this thesis.

4.2.1 Spatial GCNN : proposed architecture

In this subsection the spatial GCNN proposed in this project is going to be described.
The model is inspired by an encoder-decoder architecture, which is typically employed in many ML tasks, especially when dealing with images. This architecture is characterized by three main components. The first one is an encoder-like element which is responsible for transforming the input data into a representation of a certain size, called latent representation. In this case the input data are the node features of the graph. The second component is the 'body' of the model, in which different convolutional layers, each of them followed by a non linear activation function, are applied in order to capture the important information contained in the graph data. Once these operations have been performed, the last layer (the decoder), which is a fully connected neural network, is applied

in order to recover the output of the desired shape. The main difference from a typical encoder-decoder architecture is that the convolutional layers are built following the Message Passing Neural Networks structure described in the previous section. Now, the model is going to be described using a mathematical formulation.

As a first thing, the node feature matrix $X \in \mathbb{R}^{n \times d}$ and the edge feature matrix

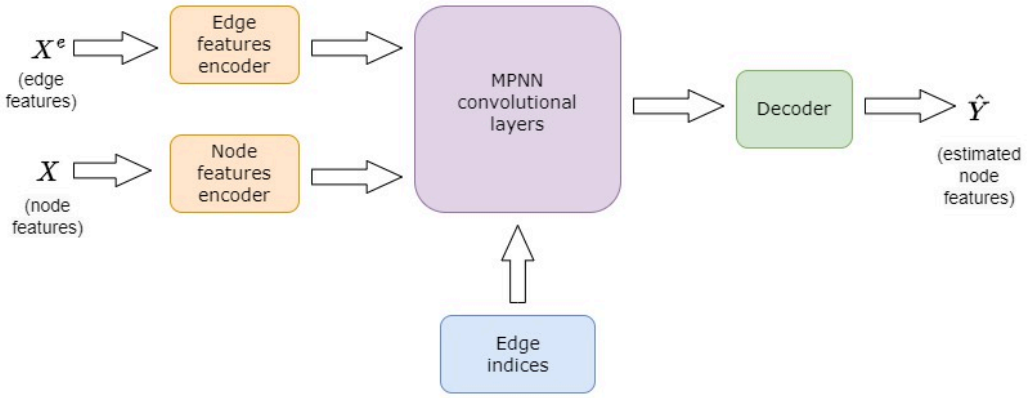


Figure 4.4: Simplified architecture of the proposed model: the features encoders are coloured in yellow, the decoder in green and the convolutional layers are in violet. The edge indices are needed for the convolutional layers to build and aggregate the messages

$X^e \in \mathbb{R}^{m \times l}$ are passed through two different neural networks with one single layer each, so that their output dimensions are the same. The output dimension of this first step and of all the outputs of the message-passing steps is called latent dimension. The outputs of these intermediate layers are called latent states/latent representations.

At the intermediate layers, the latent states are denoted by: h_i for the nodes, with $i \in V$, and $h_{e_{ij}}$ for the edges, with $e_{ij} \in E$.

After the nodes and edge features have been embedded in the first latent representation, a series of message passing layers follows. Each of these layers implements the three steps typical of a message passing neural network: message construction, aggregation of the generated messages and update of the node features.

Following the last convolutional layer, the output is passed through a decoder, a neural network with one single layer, which returns the estimated value of the node features: $\hat{Y}$.

The first latent states, which are the embedded node and edge features, are de-

noted by $h_i^{(1)}$ and $h_{e_{ij}}^{(1)}$. The message corresponding to the edge e_{ij}, coherently with Eq. 4.28, is created summing the latent state of node j and the latent representation of the edge, which now have the same dimension (latent dimension). The result of the sum is then passed through a Multi Layer Perceptron. Let us consider the generic layer $k+1$ of the GCNN. The message corresponding to the edge m_{ij} is obtained as:

$$m_{ij}^{(k+1)} = \text{MLP}\left(h_{e_{ij}}^{(k)} + h_j^{(k)} \right) \quad j \in \mathcal{N}(i). \tag{4.33}$$

where $h_{e_{ij}}^{(k)}$ and $h_j^{(k)}$ are the latent representations of the edge and of the node obtained from the previous layer k. The messages are created for all the neighbouring nodes j of the node i, and then they are aggregated by simply summing them:

$$m_i^{(k+1)} = \sum_{j \in \mathcal{N}(i)} m_{ij}^{(k+1)} \tag{4.34}$$

The last step consists in updating the node features, thus obtaining the latent representation of the node i at the layer $k+1$. This is done through a MLP in the following way:

$$h_i^{(k+1)} = \text{MLP}\left(m_i^{(k)} \right) \tag{4.35}$$

As previously stated, this process is repeated at each layer and the last activation is passed through the decoder in order to recover the desired output, which in the specific case is a vector $\hat{Y} \in \mathbb{R}^n$, containing a predicted value for each node. This is a very simple architecture to implement, but it still provides good results.

5 Problem setting

This chapter describes the problem tackled in this thesis, which is the state estimation in a Water Distribution System (WDS). It also describes in detail what is a WDS, what are its components and how these water networks can be represented as graphs. It then proceeds to describe the data employed, in particular how it has been generated, and then the characteristics of the specific water network employed. In the last part of the chapter the technical tools employed to build and train the different models are presented. Moreover a brief description of how these models have been built in PyTorch is given.

5.1 Water Distribution Systems and estimation problem

Water distribution systems are intricate networks made of interconnected pipes, storage units, and various other components, like valves and pumps. These systems are responsible for transporting potable water to urban areas, medical facilities, industrial sites and many other locations. Public water supply systems rely on these distribution systems to ensure a continuous flow of pressurized, clean drinking water for all users. It appears evident how much these systems and their correct performance is a problem of fundamental importance.
The control and monitoring problem of a WDS has as objective the minimization of water losses in the network while at the same time ensuring a sufficient supply water at the demand points. Water losses in the water network in most cases

are caused by leaks at some locations, which often happen because of damaged pipelines. The damage can be due to 'natural' causes, such as excessive loads in the pipes and material defects, or caused by some specific event, like an earthquake or a flood.
Another problem that has to be treated is the increased vulnerability of WDSs to cyberattacks, which may cause system failure. The water sector is experiencing a significant digital transformation which involves the deployment of digital devices within the water distribution network to enhance the overall network efficiency. However, this same digitalization makes the systems vulnerable to cyber-physical threats. For example, in 2020 Israel has been the subject of an attack in which the operational points of the pumps were changed in order to increase the pressure in the pipes, hence increasing leakages in the network.
The behaviour of a WDS revolves around the alteration of hydraulic states, which are generally considered to be the hydraulic head at nodes (defined in Eq. 5.1), which is a measure strictly related to the pressure, and the flow rates within the pipes. The description of how these states change over time is also called hydraulic dynamics. The most significant impact on the hydraulic behaviour of a system comes from the consumers water demands. These quantities are in practice unknown, however different algorithms have been proposed to try and estimate them. In general, the demands are assumed to follow a periodic behaviour over the hours of the day, over the days of the weeks and over the months in different seasons. For example, in a city the demands are going to be higher during the day and in the evening, while lower during the night.
A water distribution system has different components which are connected in a specific way, such as pipes, valves, pumps, storage tanks, and other components. A WDS is defined by the topology of the network, which is defined as the position of its components and the way they are connected. It is clear then that a very efficient way to represent WDSs is through graphs, which, as discussed in Section 3.3, can be represented as sets of nodes and edges.
To represent the WDS as a graph, we need to specify a set containing the nodes and one containing the edges; moreover the topology of the network is in general represented through the adjacency matrix. The set of nodes in the graph contains the following hydraulic elements:

- pipe joints (or junctions): they are the part of the network in which different pipes intersect or merge. Their role ensures that water can flow smoothly through different sections of the distribution system.

- consuming points: they represent locations where water is drawn from the distribution system for various uses such as homes, businesses or other facilities that require a water supply.
- tanks and reservoirs: they are storage facilities that hold a reserve supply of water. They are used to store additional quantities of water for peak demand periods, and provide an emergency supply if a leak in the system is present.

The edges set, instead, contains the following elements:

- pipes: they are the conduits through which water flows within the distribution system. They connect junctions, consuming points and reservoirs to ensure the transportation of water to different destinations. Different types of pipes can be employed in these systems and they are chosen based on factors such as their size, the materials they are made of and the pressures they can withstand without breaking. They are characterized by some attributes such as their length, diameter and the roughness coefficient of the walls. The roughness coefficient depends on the material used to build the pipe and is related to the pressure loss caused by friction that the fluid experiences when flowing in the pipe.
- pumps: they are mechanical devices used to pump water from lower elevations to higher elevations, and to maintain an adequate quantity of water inside the tanks.
- valves: they are control devices used to regulate the flow of water within the distribution system; they can be opened or closed to manage water distribution, isolate sections for maintenance, and control water pressure.

Pumps and valves allow to directly act on the system to modify its behaviour, hence they are the main hydraulic actuators in a water network.
Each component of the graph used to represent a specific WDS is associated with certain parameters that describe different physical properties of the elements; these features can be used in the hydraulic state estimation. The attributes of the pipes are the roughness coefficients, the lengths and the diameters, while the junctions are instead associated to a certain elevation. In general all these properties are considered to be time-invariant.
In order to guarantee a correct and efficient operation of WDSs, it is important to know the pressures at each junction and the flow rates in each pipe. These quantities can be measured through sensors which are deployed at different locations in the water network. However, given the large dimension WDSs can have, in practice it is not possible to employ a big amount of sensors which would translate to huge costs.
The need to face this problem has led to the development of different approaches

to perform state estimation of the system given the measurements at only a limited number of locations where the sensors are placed. Moreover, many different algorithms have been proposed to determine a way to optimally place the sensors available.
The state of the water network is defined by both the pressures at the junctions and the flows in the pipes, however in this work only the nodal pressures are used as states of the graph. This simplification can be done because, once the pressures at all nodes have been determined, the flows in the pipes can be uniquely recovered using the Hazen-Williams formula. This formula allows to compute the head loss of a pipe p, which is the difference of head values at the starting and ending nodes of the pipe. The *head* of a node is a measure which represents the total energy of fluid present at that specific location. It is computed as the sum of a p_j component, which is proportional to the pressure P_j at node j, and of the node elevation with respect to a geodesic reference z_j:

$$H_j = p_j + z_j \tag{5.1}$$

In particular p_j is determined as the pressure over the density of the fluid (ρ) and the gravital acceleration (g), namely

$$p_j = \frac{P_j}{\rho g} \tag{5.2}$$

Consider a generic pipe p which connects the nodes i and j, the Hazen-Williams equation allows to compute the head loss $h_p = |H_j - H_i|$ in the following way:

$$h_p = |H_j - H_i| = c_p |\, Q_p \,|^{1.852} \tag{5.3}$$

where Q_p is the flow in the pipe and c_p is the loss coefficient of the pipe, which can be computed as

$$c_p = \frac{10.67 \times l_p}{r_{HW,p}^{1.852} \times d_p^{4.871}} \tag{5.4}$$

in which l_p is the length of the pipe in meters, d_p is the diameter in millimeters and $r_{HW,p}$ is the roughness or Hazen-Williams coefficient, which is an adimensional value. In order to compute the flows given the heads, it is sufficient to

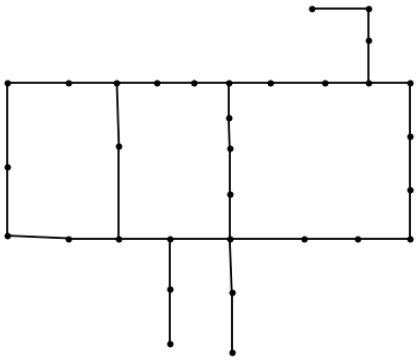
invert Eq. 5.3, hence obtaining

$$Q_p = (\frac{1}{c_p} \mid H_j - H_i \mid)^{0.54} \tag{5.5}$$

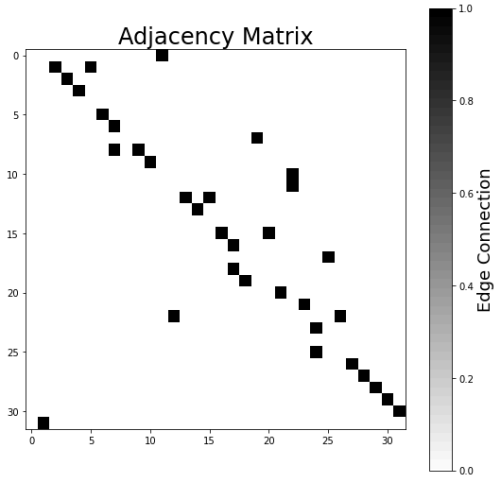
For simplicity from now on the term pressure will be used to refer to the hydraulic head of the nodes, as is commonly done in the field of WDS state estimation. This can be done because in the Hanoi water network, which is used in this thesis, all the components are put at the same height, so pressures and heads are almost equivalent.
In this work, three Graph Neural Network models are proposed in order to deal with the problem of pressure estimation; two are built following a spectral approach and the other one with a spatial approach. In both cases the models use the data available at the nodes with sensors and the topology of the network to learn how to estimate the pressure values at each node. The models are trained and tested on the Hanoi network, described in the next section, which is represented as a graph. The models proposed use the measurements at all nodes only during the training phase of the neural network. After the models have been trained, they are able to receive as input a graph with only few measurements and output the estimated pressure values at each node.
It is worth noting that estimating the state of a Water Distribution System is the starting point to tackle more complicated problems. For example, all methods used nowadays to localise leakages in Water Distribution Systems rely on the availability of real time measurements of the states of the network.

5.2 Water network employed and Dataset

The water network used in this study is the Hanoi network. The Hanoi network is a part of the total Hanoi Water Distribution Network and it consists of one reservoir which is the water source, 31 water demand nodes (which coincide with the junctions), and 34 pipes. The network, first presented in [9] , is pretty small and simple, in fact it does not contain any valves or pumps and all the junctions are at the same elevation. For this reason it has been chosen as a starting point to train and evaluate the models. The components of the network are described in Table 5.1.
The data needed in order to train and evaluate the models are the pressure values at all the junctions of the network, while there is no need for the demands



(a) Hanoi water distribution network configuration



(b) Binary adjacency matrix representing the connectivity of the Hanoi network

and the flows in the pipes. The data used is part of the Leakage Diagnosis Benchmark (LeakDB) dataset proposed in [32], which contains a large number of artificially created but realistic scenarios, on different water distribution networks, under varying conditions. This benchmark has been created specifically for the task of detecting leakages in the network and contains different scenarios, all unique, both with and without leakages. However, in our case, only the scenarios which do not contain leakages in the pipes are used, making the problem easier to work with.

The GitHub repository associated to the LeakDB paper contains a folder with the different scenarios created and simulated via the EPANET software (described in the next section) for the Hanoi network. The hydraulic behaviour of the network is simulated for 8760 hours, i.e. 365 days, with a hydraulic time-step of t = 30 minutes. This leads to 17520 different pressure values for each node, and hence to the creation of a dataset containing 17520 graphs; however in the training and testing of the models only a subset of the entire dataset is used, due to memory reasons. The nodal pressures are assumed to undergo gradual changes over time and they do not have swift changes which may be caused by leaks and other problems inside the network.

In order to simulate the hydraulic behaviour of the system, the EPANET software requires the model of the Hanoi network, contained in a '.INP' file. This model contains information about the elevation of the nodes, the roughness, length and diameters of the pipes. Moreover, the simulator needs the demands

at the nodes to be specified.
The demands pattern generation is accurately described in [32] and now is briefly presented. The demands are generated based on historical real-world data collected from water utilities. These demands are obtained through the multiplication of three signal components:

- Weekly Periodic Component (C_w): this component reflects the demand fluctuations happening throughout a week. It is influenced by a range of social and economic factors related to consumers, the values are on average higher during the week days and lower in the weekends. Moreover, there are two peaks during the day, one in the morning and one in the evening.
- Yearly Seasonal Changes Component (C_y): This component accounts for variations in water consumption resulting from seasonal changes within a year. It captures the impact of different seasons on demand patterns; in general summer months have higher water demands.
- Random Component (C_r): The random component represents high- frequency variations in demand; these fluctuations can arise from unpredictable consumer behaviors, transient events, repairs, and other activities within the network.

The demand at a specific node is hence generated as

$$d = \beta \cdot C_w \cdot C_y \cdot C_r \tag{5.6}$$

where β is a scalar value, different for each node, representing the base demand of the node. By using the formula above it is possible to obtain specific demand patterns for each node. Once the pressures are available at the different

Metric	Hanoi Network
Number of Junctions	32
Number of Pipes	34
Number of Pumps	/
Number of Reservoirs	1
Number of Tanks	/
Number of PSV Valves	/
Network Diameter	23
Degree (min, avg, max)	(1, 2.125, 4)

Table 5.1: Topology of Hanoi Water Network

time steps, the dataset is ready to be created. The dataset is composed of elements represented as undirected graphs, each of them having the topology of

the Hanoi water network. The nodes have as features the pressures at the different time steps, these values form the node feature matrix, which in this case is a vector, since the pressures are scalar values. Moreover, the length, diameter and roughness coefficient of the pipes are associated as features to the edges. These are stored in the edge feature matrix which has dimension mx3, where m is the number of edges. The edge attributes are used only in the spatial GCNN. The pressures at the nodes where there are no sensors are then put to zero, in order to create the node feature vector of the graph that is the input of the Graph Neural Network models. However, to each graph is also associated a vector Y, containing the original pressures of all the nodes. This vector is needed in the training phase of the models to compute the value of the loss function, which is used to modify the inner parameters of the models in order to make more accurate predictions. To sum up, each element of the dataset is a graph with n nodes and m pipes, to which we associate:

- X: the node feature vector of dimension nx1. Each element represents the pressure at the specific node if it has a sensor, otherwise the element is put to zero.
- X^e: the edge feature matrix of dimension mx3. Each pipe has as features its length, its diameter and its roughness coefficient.
- A: the adjacency matrix of dimension nxn which represents the connectivity between the nodes.
- Y: the vector containing the pressure values at all nodes, either with or without sensor. It is needed only in the training phase.

Notice that the vector Y containing the pressures at all nodes is used only in the training phase, however, during the evaluation phase, the models are able to estimate the missing pressures solely based on the sparse values corresponding to the sensors locations, contained in the node feature vector.

5.3 Tools and framework employed

All the technical tools and frameworks used in the project are going to be described briefly, in order to understand how the problem has been set. At first the focus will be on Python an its related libraries, then the EPANET software, widely used in the field of hydraulic simulations, is presented.
Python is a high-level programming language known for its simplicity and readability. Nowadays it is the most widely used language in the fields of machine

learning and deep learning. Python's main strengths are not only the simplicity of its syntax, which makes it a fairly easy language to learn, but also the great and increasing number of libraries available which are specialized for very different tasks. In the field of machine learning and deep learning, PyTorch, an open-source deep learning framework directly related to Python, plays an even bigger role. Its peculiarity is that it introduces the use of tensors beside vectors and matrices, making it specifically suited for image related problems. However it is not able to work on data structured in graphs and for this reason PyTorch Geometric has been developed. PyTorch Geometric makes it possible to handle graph data and to develop graph neural networks and other graph-based machine learning models. The key advantages of PyTorch Geometric include:

- Graph-Centric Features: PyTorch Geometric provides specialized features for working with graph data, allowing it to handle node attributes, edge attributes and the graph connectivity efficiently.
- Pre-Built Graph Layers: It offers a wide range of pre-built convolutional layers which have been presented in many papers. For example the GCN and ChebNet convolutional layer are already implemented and available for use.
- Custom Dataset: PyTorch Geometric offers a InMemoryDataset class, which can be customised in order to create a dataset composed of graph structured data.

Overall PyTorch Geometric is a great choice considering that it is easy to use, provides a good performance and offers a wide range of pre-built graph neural network layers.

To simulate the water networks hydraulic behaviour, EPANET is an open-source software developed by the U.S. Environmental Protection Agency. Its purpose is to model and simulate water distribution systems, making it an invaluable tool in the fields of engineering and water utility management. EPANET is a hydraulic modeling software that specializes in analyzing and understanding the behavior of water distribution systems over time. These systems are complex networks composed of pipes, pumps, valves, storage tanks, and various other components that, together, ensure the reliable delivery of water to consumers (see Section 5.1 for details). EPANET central use is hydraulic analysis, in fact it excels in calculating water flow rates and pressures, which are the quantities that define the state of a WDS. Furthermore, EPANET can be used in water quality analysis, since it can simulate the movement and transformation of water quality constituents within the distribution system. This is particularly significant to

monitor water safety in the first place, but also water quality standards and to investigate potential contamination events. EPANET is characterized by a user-

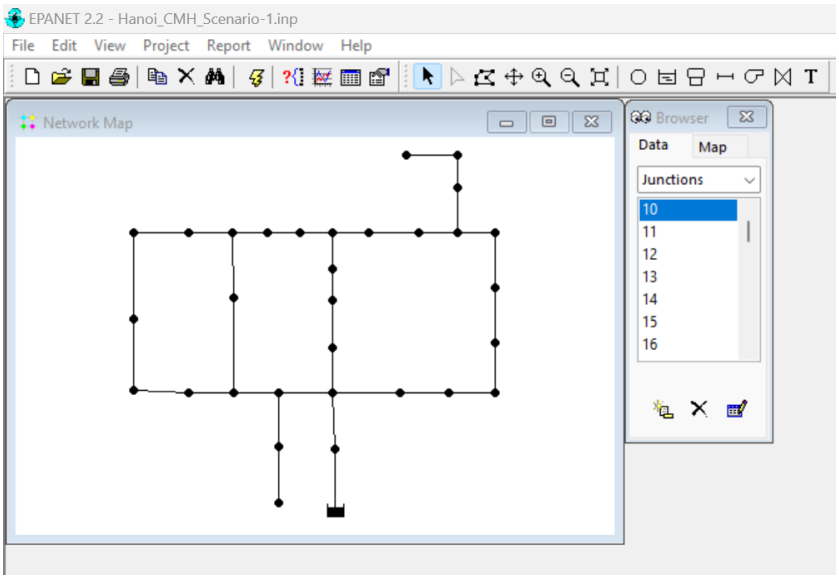


Figure 5.2: Example of the Hanoi network visualized on EPANET; it allows to visualize and modify its components and their attributes

friendly graphical user interface that simplifies the creation and visualization of complex network models. It is able to compute hydraulic and water quality parameters over time, enabling the possibility to simulate different scenarios. Other tools employed in this project are:

- networkx: a Python library used for the creation and analysis of graphs.
- matplolib: a popular Python library which provides a wide range of plotting functions and customization options to efficiently visualize data.
- Optuna: an open-source Python library designed for hyperparameter optimization.
- Panda: a library used for data manipulation and data analysis tasks.

5.4 Models built in PyTorch

In this section a brief description of how the models used for this estimation task have been built in PyTorch is given.
All of the proposed models take as input the node feature matrix, which in the

specific case is a vector, and output a vector of the same dimension containing the estimated values at all nodes. Let n be the number of nodes present in the Hanoi network, then the input node feature vector is $X \in \mathbb{R}^n$. The elements corresponding to the nodes where a sensor is present are simply the pressures at the corresponding nodes. In the other nodes, where the measurements are not observed, the features are put to zero.
The ChebNet model and the GCN model are easy to implement in PyTorch Geometric, because their corresponding convolutional layers are already implemented in a specific library. The 'ChebConv' layer takes as input the node feature vector and a tensor containing the initial and ending nodes for each edge in the graph. This tensor is used inside the layer to build the normalized Laplacian matrix used by the layer to perform the convolution described in Eq. 4.21. After each 'ChebConv' layer, except for the last one, the SiLU (Sigmoid Linear Unit) activation function is applied, which is defined as:

$$SiLU(x) = \frac{x}{1 + e^{-x}} = x\sigma(x) \tag{5.7}$$

and is the classic sigmoid function $\sigma(x)$ multiplied by the input.
The hyperparameters of this model that can be optimized are the number of convolutional layers, the output dimension of the feature vector after each convolution (latent dimension) and the value of K. The GCN layer is equivalent to the ChebConv layer when imposing that the polynomial degree is 1, hence putting $K = 2$. Differently from the ChebNet model, the layers are followed by the LeakyReLu activation function instead of the SiLU, since it has produced better results. It is defined as:

$$\text{LeakyReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{5.8}$$

where α is a small positive constant that determines the slope of the function for negative values of x. The hyperparameters of this model are the number of convolutional layers and the latent dimension after each convolution.
The spatial model proposed is implemented using the MessagePassing class already present in PyTorch Geometric. This model does not directly use the Laplacian matrix as the others, instead uses the tensor containing the edges to define a neighborhood for each node. Then the messages at the edges connecting the

neighbours are built and used to update the central node representation. The 'MessagePassing' base class has three methods (functions) that can be modified accordingly, in order to create the Message Passing layer which performs the passages described in 4.2.1. The user only has to define the message function ϕ, the update function ψ and the aggregation scheme $\oplus$ as described in in 4.32. In this specific case the message function and the update function are two different MLP, containing a different number of layers; while the aggregation scheme employed is the sum of the messages. Each MessagePassing layer is followed by the LeakyReLU activation function.
All the weights of the layers are initialized following the Xavier initialization ([11]) while the biases are initialized to zero. This is a common procedure and it has shown to make the initial phase of the training process more stable. The loss function used to compare the output of the model and the real target is the Mean Squared Error loss.

6

Results and analysis

The models presented in the previous chapter have been trained and tested on the Hanoi water network. Different sensors configurations have been employed, in which the number of observed nodes were different. As it can be expected, the models perform better with higher observation ratios, but in practice we are more interested in scenarios with a lower number of sensors.
Since the specific task treated can be considered a regression problem, the following metrics have been considered to evaluate the models' estimation accuracy. All of these work by examining the residuals or errors, i.e. the difference between the actual and predicted values. For a specific sample i of the dataset, the residual is computed as:

$$e_i = y_i - \hat{y}_i \tag{6.1}$$

where y_i is the real value, or target, and $\hat{y}_i$ is the estimated one.

By this definition it follows that when the residuals get closer to zero, the model predictions become more accurate. The Mean Absolute Error (MAE) is defined as the sum of all the residuals' absolute values averaged by the total number of data in the dataset. It represents the average absolute difference between the model predictions and the real values. It is defined as:

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|^2 \tag{6.2}$$

The Root Mean Squared Error (RMSE) is the square root of the average squared

difference between the actual and predicted values (the residuals). It measures how well the model predictions align with the actual values, similarly to MAE, but RMSE is particularly useful to assess whether there are significant errors or discrepancies that might occur if the model tends to overestimate (predicting values significantly higher than the actual ones) or underestimate (predicting values lower than the actual ones) the predictions. The RMSE not only provides insight into the average closeness of predictions to their actual values, but also highlights the impact of significant errors, as these will contribute significantly to the RMSE score. It is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{6.3}$$

The Mean Relative Absolute Error (MRAE) computes the absolute error for each data point, as in the MAE, and then divides it by the magnitude of the actual value for that data point. This normalization by the actual value magnitude ensures that the error is expressed as a relative fraction of the actual value. It provides a way to assess the model performance in a way that is sensitive to the scale of the data, making it particularly useful when dealing with datasets where the magnitude of the target variable varies significantly. It is defined as

$$MRAE = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{|y_i|} \tag{6.4}$$

The Mean Relative Root Squared error (MRRSE) is the square root of the average of the squared relative errors. It provides a measure of the prediction accuracy relative to the magnitude of the actual values and it provides insights into how well predictions match the scale of the data. RMSE uses the squared error, emphasizing the impact of larger errors, while RMAE uses the absolute error, giving equal weight to all errors.

$$MRRSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{y_i}\right)^2} \tag{6.5}$$

Given that the pressures data that the model has to estimate vary in a significant range, MRAE and MRRSE are the best metrics to evaluate the performance of

the model in this specific task. In fact, it will be seen, that even if the results lead to relatively larger RMSE and MAE, the performance of the model can still be considered accurate. Moreover, these last two metrics are easier to interpret in term of accuracy of the estimates.

6.1 ChebNet model: results

For the ChebNet model an accurate hyperparameter optimization has been performed in order to find the best values. The range in which the hyperparameters have been investigated are indicated in Table 6.1, together with the final choice of the values for both networks. As it can be seen, the model consists of 4 convolutional layers, each of them with a different latent dimension and value of K. In order to prevent overfitting, L2 regularization has been implemented with a factor defined by the hyperparameter weight_decay, settled to the value $1e-4$. It has to be noted that the parameters have been investigated in a relatively small range, for this reason they still might not be the optimal ones; however they led to good results.

The number of epochs the model has been trained on has been set to 1000.

hyperparameter	range of possibility	Hanoi
K1	[32,40]	37
K2	[58 , 63]	62
K3	[42, 48]	43
latent 1	[28 , 35]	30
latent 2	[32 , 38]	32
latent 3	[38 , 44]	40
learning rate	[1e-5 , 1e-4]	2e-5
weight decay	[1e-5 , 2e-4]	1e-4

Table 6.1: The table shows the range of search of the hyperparameters and the final choices made for both networks

Moreover, an early stopping mechanism has been implemented, meaning that the training has been stopped if there was no improvement in the validation loss after 250 epochs. Early stopping is an additional procedure that helps avoiding overfitting.

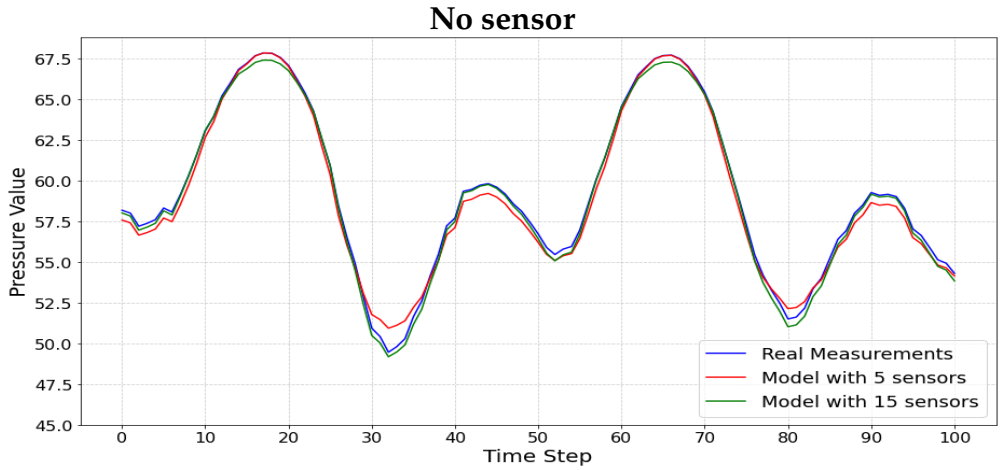
The number of installed sensors in the Hanoi network has also been changed to evaluate the performance of the model when there are different measurements

observation ratios. In particular, the following configurations have been investigated:

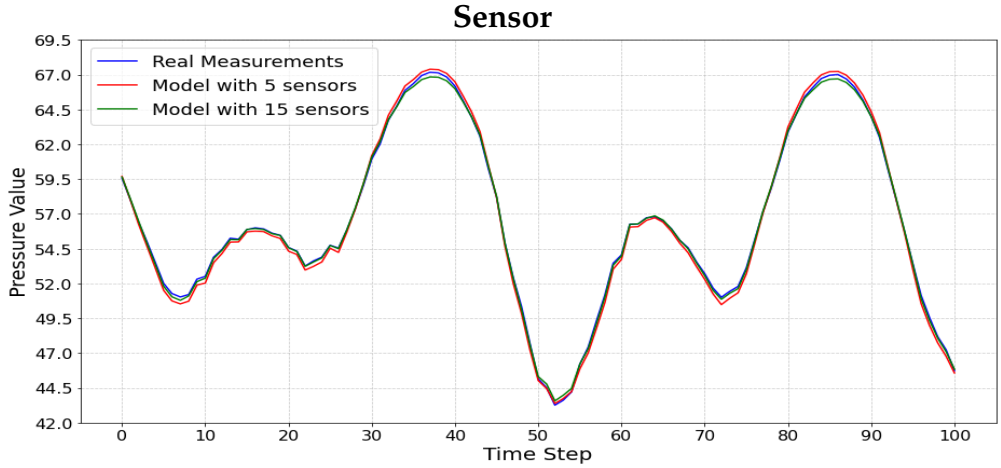
- 1 sensor: observation ratio equal to 0.0323
- 5 sensors: observation ratio equal to 0.1613
- 15 sensors: observation ratio equal to 0.4838.

From the results shown in table 6.2, it appears evident that using one single sensor is not sufficient and the model is not capable of learning how to make the estimates accordingly. Since the performance obtained with one single sensor is evidently insufficient, the plots of the pressures at some nodes have not been reported, differently from the other two configurations. However, it has been noted that the node with the sensor and its immediate neighbours have a decent behaviour (in the table the errors corresponding to the node with sensor are low), while all the others do not learn accordingly. For this reason we can conclude that for the state estimation problem using the ChebNet model using one single sensor is not enough. Let us now see the behaviour of the model on the other configurations.
As it can be expected, employing 15 sensors leads to the best results, however the ratio of observed nodes is quite high. In real life we are more intersted in deploying a smaller number of sensors, so the middle configuration with 5 sensors is the one we are more interested in. The model applied to the Hanoi network under this observation rate is able to lead to satisfactory results while having a smaller number of sensors, and consequently a smaller cost to monitor the system.
The MAE and RMSE are absolute metric, meaning that they are not normalized over the true values. They give an idea of how much the model on average makes a mistake; in the case of the pressures (heads) they are measured in meters. If the RMSE is higher, like in this case, it means that even if overall the model shows a good performance on average, there are some errors of considerable values. In this case the RMSE can still be considered good, however looking at Fig. 6.1, we can see that there are some significant errors (1-2 meters) at some time steps, which increase the RMSE. Moreover, since the metrics using the root mean squared error penalize more large errors, it is sufficient for a single node to perform worse than the others and the overall error will grow significantly.
To better compare the models performance on the network, it is better to consider both absolute and relative metrics. The relative metrics are normalized

over the true values of the pressures and they are easier to evaluate and compare, since they are strictly related to the accuracy of the model.



(a) In this figure the real pressures and the pressures estimated by the ChebNet model corresponding to 5 sensors (in red) and 15 sensors (in green) at a node in which there is no sensor are plotted.



(b) In this figure the real pressures and the pressures estimated by the ChebNet model corresponding to 5 sensors (in red) and 15 sensors (in green) at a node in which a sensor is placed are plotted.

Figure 6.1: Comparison of the pressures estimated with 5 and 15 sensors at two nodes: one with and one without sensor.

To better comprehend the estimation power of the model, the estimated pressures and the real ones (the target values) are plotted in Figure 6.1. It can be noted that the overall estimation is quite good, however the model has some difficulties in modelling the peaks and lows of the pressure signal. Correspond-

ing to these points, in fact, the estimation errors are considerably larger. Moreover, in the first image the pressures at nodes without sensors installed are plotted; it can be noted that in this case the difference in the quality of the estimation is more evident and that having more sensors, as expected, leads to better results. From the images above, where the real and estimated values are plotted corresponding to a specific node, it appears evident that the overall estimation ability is good. However, when the pressures are around the maximum values (the peaks), the model tends to underestimate them, while around the minimum values the model tends to overestimate them. Moreover, we can notice that in the nodes with sensors, the model performs almost in the same way, with both configurations of sensors.

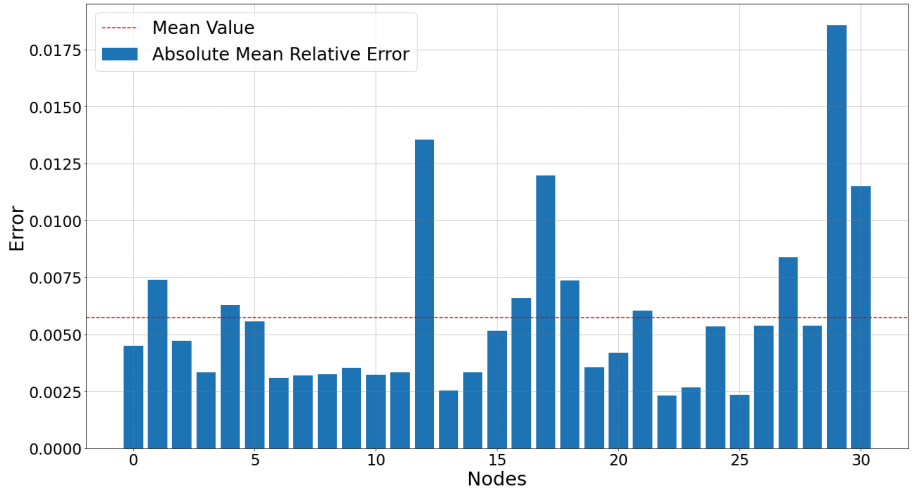


Figure 6.2: Plot of the relative mean absolute error for each node for the ChebNet model tested with 5 sensors.

In Fig. 6.2 the MRAE is plotted for each node in the Hanoi network. From the plot it is evident that the errors are small and almost all in the same range except for some some nodes which have considerably higher errors. Overall, this model can reconstruct the signals of the pressures inside the network with sufficient accuracy.

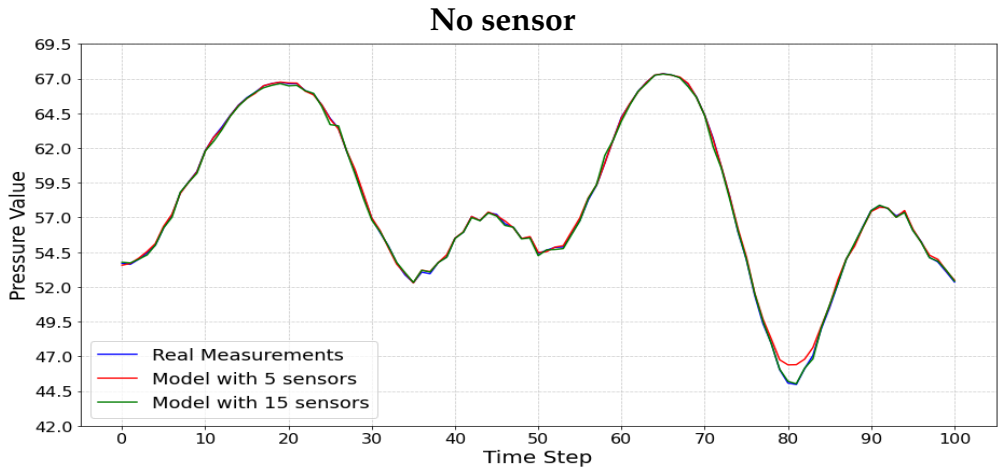
metric	Sensors presence	# of sensors		
		1	5	15
RMSE	all	6.77260	0.47753	0.29897
	sensors	0.23819	0.44397	0.26313
	no sensors	6.88441	0.48371	0.33297
MAE	all	5.77178	0.30438	0.19860
	sensors	0.22751	0.28419	0.17484
	no sensors	5.95659	0.30826	0.22394
MRRSE	all	0.13349	0.01004	0.00634
	sensors	0.00342	0.00937	0.00566
	no sensors	0.13569	0.01017	0.00698
MRAE	all	0.10769	0.00573	0.00375
	sensors	0.00327	0.00512	0.00328
	no sensors	0.11117	0.00584	0.00425

Table 6.2: In the Table the results obtained through the ChebNet model employing 1 sensor, 5 sensors and 15 sensors are reported.

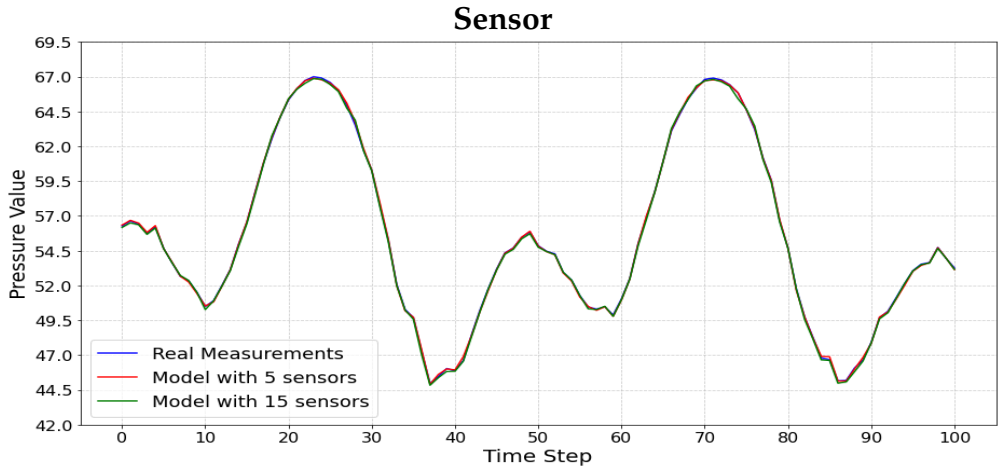
6.2 Spatial model: results

The training phase of the spatial model lasts 2000 epochs in this case, since the model is faster than the ChebNet and early stopping is implemented with a patience of 250 epochs. No diligent hyperparameter optimization has been performed in this case, since it still shows better performance than the ChebNet model. The latent dimension of the embedded node and edge features has been put to 42 and 20 Message Passing layers have been implemented. The learning rate is put to $1e-5$ while the weight_decay parameter which regulates the L2 regularization is put to $1e-6$.
The model has been trained and tested with the same sensor configurations as before, and the results obtained in each case are shown in Table 6.3. It can be noted that this model performs better than the previous one, even in the case of 1 single sensor employed. In this configuration, however, the model still performs badly, and only the nodes close to the one with the sensor obtain good estimates. The estimated values at the node with the sensor and at its close neighbours (around 5 nodes) is actually pretty good, however at the rest of the

nodes the model is not able to reconstruct the signal, which results to be flat.



(a) In this figure the real pressures and the pressures estimated by the spatial model corresponding to 5 sensors (in red) and 15 sensors (in green) at a node in which there is no sensor are plotted.



(b) In this figure the real pressures and the pressures estimated by the spatial model corresponding to 5 sensors (in red) and 15 sensors (in green) at a node in which a sensor is placed are plotted.

Figure 6.3: Comparison of the pressures estimated with 5 and 15 sensors at two nodes: one with and one without sensor.

In the plots in Fig. 6.3, are shown the pressures estimated corresponding to a random node in which the sensor is present and one without, just as in the previous case. In this case the model has a very good and similar performance with both the observation ratios. The main difference in the two configuration can be noted in the node without sensor. In the case of a smaller observation ratio, the

model faces some difficulties in estimating the lower values but overall in both cases the model performs very well.
In Fig. 6.4 the ARE for each node of the Hanoi network are plotted, as done before. Comparing this to the corresponding plot for the ChebNet model we can see how in this case the values are considerably lower.

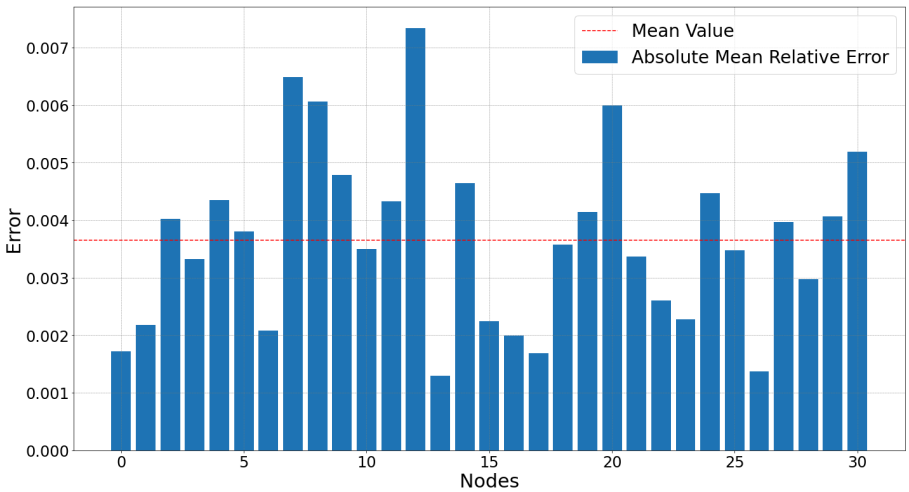


Figure 6.4: Plot of the relative mean absolute error for each node for the spatial model tested with 5 sensors

metric	Sensors presence	# of sensors		
		1	5	15
RMSE	all	5.32999	0.56529	0.47753
	sensors	0.23940	0.55998	0.43651
	no sensors	5.51036	0.57089	0.51771
MAE	all	3.59033	0.18552	0.12571
	sensors	0.21123	0.169744	0.12154
	no sensors	3.82338	0.20236	0.13016
MRRSE	all	0.10353	0.014040	0.010045
	sensors	0.00451	0.013882	0.009188
	no sensors	0.10704	0.014205	0.010886
MRAE	all	0.06672	0.00365	0.00573
	sensors	0.00362	0.00333	0.00549
	no sensors	0.07107	0.00400	0.00597

Table 6.3: In the Table are reported the results obtained through the spatial model employing 1 sensor, 5 sensors and 15 sensors

6.3 GCN MODEL: RESULTS

Since the peculiarity of the GCN model is that it has fewer parameters with respect to the ChebNet model and the spatial one, this results to be faster. For this reason it has 6 layers instead of 4 like the ChebNet and it has higher values for the latent dimensions. Initially, we tried to use a higher number of layers but the performance got worse. It has been shown in [18] that deeper GCNNs do not lead to better results in general, as one might expect.
Considering that the performance of this model is significantly worse than the previous ones, no serious hyperaprameter optimization has been performed. The training setting is the same as in the ChebNet model, so the same learning rate and weight decay parameters have been used. Moreover, like the spatial model, the GCN has been trained for 2000 epochs with early stopping after 250 epochs without improvement of the validation loss.
The results achieved through this model are not satisfactory compared to the ChebNet model, in particular the model really struggles to learn how to estimate lower pressures. The bigger values in the pressure patterns are estimated

significantly better than the lower ones, but still the results cannot be compared to the ones obtained through the ChebNet model. This can be seen in Figure 6.6, where the estimates corresponding to the lower points have a very big error, especially in the case of only 5 sensors available.
The model has been tested using two different sensors combinations, one with 5 sensors and one with 15 sensors. Considering the fact that the model performs in a significantly worse fashion with respect to the ChebNet model, it has not been considered the configuration with only 1 sensor. The main positive aspect of this model is that it needs less time to train, since it contains less parameters. Overall the performance of the model with 15 sensors is on average good, but still considerably worse than the other two models. When only 5 sensors are employed, the model is not able to reconstruct the signal of the pressures with a satisfactory accuracy. As usual the evaluation metrics used are reported in Table 6.4 and the Absolute Relative Error at each node is plotted in Fig. 6.5. Comparing the results obtained with the other two models is hence immediate and it appears clear that the best model is the spatial one, followed strictly by the ChebNet. The GCN model, even if faster than the others is not able to obtain good estimate results, especially in the configuration with 5 sensors, which is the one we are more interested in.

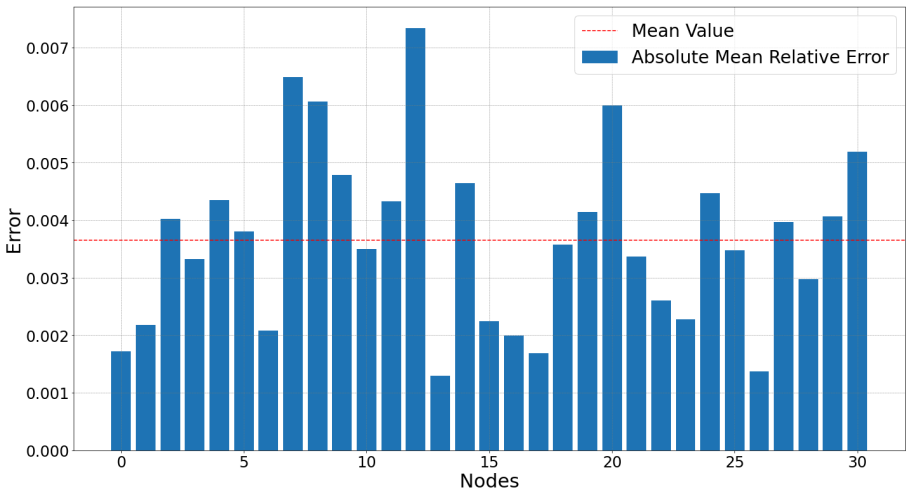
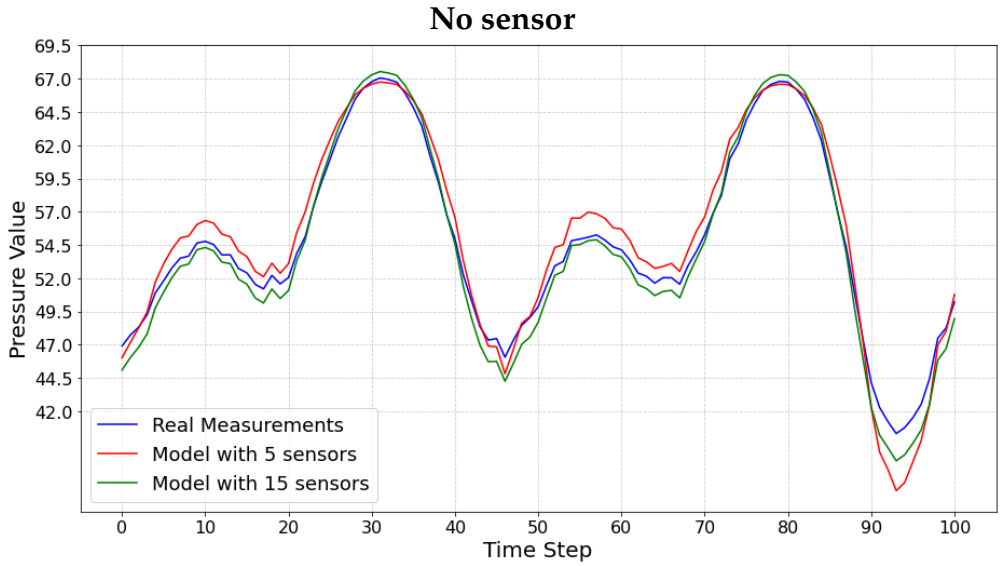


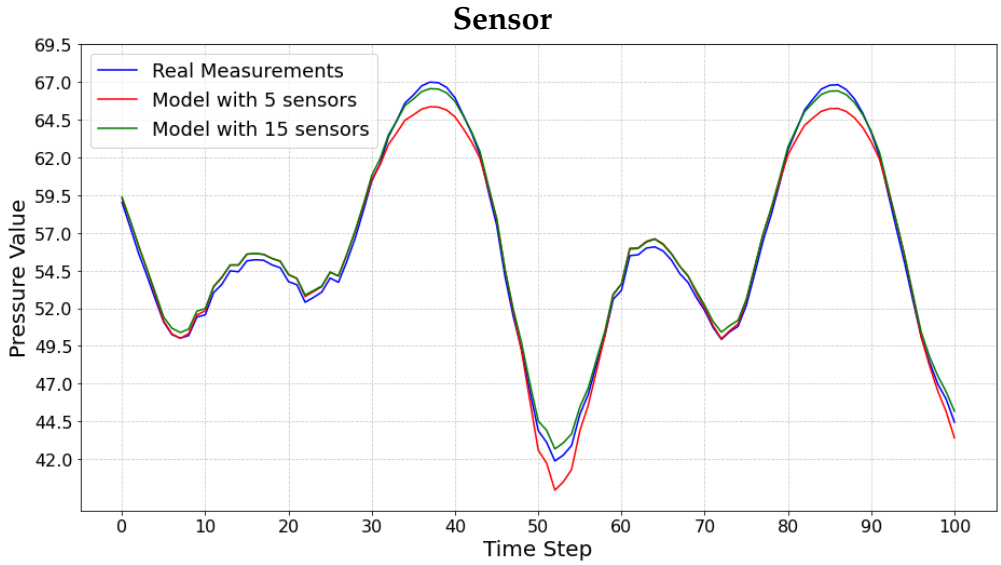
Figure 6.5: Plot of the relative mean absolute error for each node for the GCN model tested with 5 sensors

metric	Sensors presence	# of sensors	
		5	15
RMSE	all	1.55760	1.16720
	sensors	1.00222	0.79013
	no sensors	1.64302	1.43328
MAE	all	1.21484	0.80289
	sensors	0.83439	0.61092
	no sensors	1.28801	0.98285
MRRSE	all	0.029250	0.021896
	sensors	0.016389	0.014444
	no sensors	0.031120	0.027081
MRAE	all	0.021996	0.01459
	sensors	0.014037	0.011059
	no sensors	0.023526	0.017912

Table 6.4: In the Table the results obtained through the GCN model employing 5 sensors and 15 sensors are reported.



(a) In this figure are plotted the real pressures and the pressures estimated by the GCN model corresponding to 5 sensors at a node in which there is no sensor



(b) In this figure are plotted the real pressures and the pressures estimated by the GCN model corresponding to 5 sensors at a node containing a sensor

Figure 6.6: Comparison of the pressures estimated with 5 and 15 sensors at two nodes: one with and one without sensor.

6.4 Considerations

The models employed, which are Graph Neural Networks, seem to be a good approach for this state estimation task. They are able to capture the information contained not only in the node features, which are the pressures, but also in the topology of the water network.
The GCN and ChebNet models use directly the Laplacian matrix of the graph to perform the convolution operation. The dimension of this matrix increases with the number of nodes in the graph; for this reason applying these models to larger graphs might become a computationally expensive task. On the other hand, the spatial model does not depend directly on the Laplacian, however with larger networks the number of layers employed will need to increase, in order to allow each node to receive also the information associated to other nodes.
Even if GCN models perform well in a variety of tasks, they do not seem to be suited for this specific task. However, it is noted that among all the models proposed it has the fastest training time. On average, the GCN model takes around 3.7 seconds to perform an iteration, i.e. to process one batch of data. All the models have been trained using the same batch dimension (50 samples) and for this reason this quantity can be compared. The second faster model is the spatial one with a value of around 7.5 seconds for each iteration, while the ChebNet is definitely the slowest one, taking on average 29.8 seconds to complete a single iteration.

7

Conclusions and Future Works

This thesis has provided an introduction to the field of Graph Neural Networks, in particular focusing on the reasons behind their development. Graph Neural Networks have been implemented in order to work with data which does not have a regular structure, like images have, and they are inspired by Convolutional Neural Networks. According to the way these models define the convolution on graph data, they are classified into spatial and spectral models. Three particular architectures, two of which built following the spectral approach and one the spatial one, have been accurately described. These are the ChebNet, the GCN and the Message Passing Neural Network, which are the models that have been implemented in this work.

This thesis has proposed and evaluated the three different architectures of GCNNs applied to the problem of state estimation in Water Distribution Systems. These systems are of extreme importance in our everyday life and finding ways to improve their monitoring process and optimize their performance is fundamental. In order to do so, it is necessary to have measurements at many locations in the system; this however leads to a high cost, since sensors can be expensive. As a consequence, many algorithms have been developed in order to find the optimal configuration of a given number of sensors inside the WDS.

Once the sensors are deployed inside the network, the problem of estimating the state of the system in the locations without sensors remains. This problem can also be considered as the reconstruction of a signal which is only partially observable. In particular, we are interested in the pressure values at the junctions of the network, which can be considered the state of the system. Since WDSs can

be easily represented as graphs, employing GCNNs for this task has appeared to be a good choice.
The ChebNet model and the GCN model are built following a spectral approach and the way they perform convolution on the graph signals depend explicitly on the graph Laplacian matrix. The size of this matrix depends on the number of nodes of the graph, for this reason these models become computationally expensive when dealing with large graphs. The network they have been tested on, the Hanoi network, has however a small number of nodes so this has not been a problem. Of the two models, the first one has performed significantly better than the other.
The spatial model has obtained the best results overall and, since it does not use directly the Laplacian, its computational cost is not strictly related to the size of the graph. This model has been built using a very simple architecture which for sure can be improved; however, given the good results obtained and the fact that it can use also the information stored in the edges, it seems like the best approach to investigate further.

7.1 Future works

Note that the models proposed have only been tested on the Hanoi network which is a rather small one; for this reason in the future their performance needs to be tested on networks of larger sizes and which contain more complicated components, such as valves and pumps. In particular, we expect the spatial model to still perform better than the others and have a relatively small computational time. The other two models, which follow the spectral approach, might still perform well but their training time will grow significantly.
Moreover, the data provided for training and testing the models has been generated through an hydraulic simulator, which is the most realistic available at the moment. However, real scenarios might show different and unpredictable behaviours, so it is important to test the model on data which has been actually collected in real life situations. Real data might be affected also by sensors noise or errors in their calibration.
Moreover, it is worth noting that state estimation is only the starting point for many other problems, such as detection of leakages and of cyber-physical attacks. It is of interest then to see how the algorithms already available to perform these tasks would actually perform using the estimated measurements.

It is also interesting to extend the state estimation problem to a state prediction one, in which we want to use GCNNs to extract spatial information, together with a temporal NN to extract temporal information. A similar approach has been used in traffic forecasting, obtaining good results.

References

[1] Inaam Ashraf et al. *Spatial Graph Convolution Neural Networks for Water Distribution Systems*. 2022. arXiv: 2211.09587 [cs.LG].

[2] Peter W. Battaglia et al. *Relational inductive biases, deep learning, and graph networks*. 2018. arXiv: 1806.01261 [cs.LG].

[3] Joan Bruna et al. "Spectral Networks and Deep Locally Connected Networks on Graphs". In: *International Conference on Learning Representations (ICLR)*. 2014. URL: https://arxiv.org/abs/1312.6203.

[4] Joan Bruna et al. "vSpectral Networks and Deep Locally Connected Networks on Graphs". In: *arXiv preprint arXiv:1312.6203* (2013). URL: https://arxiv.org/abs/1312.6203.

[5] Siheng Chen et al. "Discrete Signal Processing on Graphs: Sampling Theory". In: *IEEE Transactions on Signal Processing* 63 (24 2015). ISSN: 1053587X. DOI: 10.1109/TSP.2015.2469645.

[6] Xinye Chen. "Understanding Spectral Graph Neural Network". In: *Journal of Graph Analysis* (2021). Department of Mathematics, University of Manchester, Manchester, M13 9PL, United Kingdom.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 3844–3852. URL: http://papers.nips.cc/paper/6081-convolutional-neural-networks-on-graphs-with-fast-localized-spectral-filtering.pdf.

[8] Vijay Prakash Dwivedi et al. *Benchmarking Graph Neural Networks*. 2022. arXiv: 2003.00982 [cs.LG].

[9] Okitsugu Fujiwara and Do Ba Khang. "A two-phase decomposition method for optimal design of looped water distribution networks". In: *Water Resources Research* 26 (1990), pp. 539–549. URL: https://api.semanticscholar.org/CorpusID:120956364.

[10] Justin Gilmer et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG].

[11] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[12] Gergely Hajgató, Bálint Gyires-Tóth, and György Paál. *Reconstructing nodal pressures in water distribution systems with graph neural networks*. 2021. arXiv: 2104.13619 [cs.LG].

[13] Ehsan Hajiramezanali et al. *Variational Graph Recurrent Neural Networks*. 2020. arXiv: 1908.09710 [cs.LG].

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216. URL: http://arxiv.org/abs/1706.02216.

[15] Manuel Herrera et al. "Predictive models for forecasting hourly urban water demand". In: *Journal of Hydrology* 387.1 (2010), pp. 141–150. ISSN: 0022-1694. DOI: https://doi.org/10.1016/j.jhydrol.2010.04.005. URL: https://www.sciencedirect.com/science/article/pii/S0022169410001861.

[16] Elvin Isufi, Fernando Gama, and Alejandro Ribeiro. *EdgeNets:Edge Varying Graph Neural Networks*. 2021. arXiv: 2001.07620 [cs.LG].

[17] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[18] Guohao Li et al. *DeepGCNs: Can GCNs Go as Deep as CNNs?* 2019. arXiv: 1904.03751 [cs.CV].

[19] Yujia Li et al. *Gated Graph Sequence Neural Networks*. 2017. arXiv: 1511.05493 [cs.LG].

[20] Roland Liemberger and Alan Wyatt. "Quantifying the global non-revenue water problem". In: *Water Science and Technology: Water Supply* 19 (July 2018), ws2018129. DOI: 10.2166/ws.2018.129.

[21] Rodrigo López et al. "Multi-Model Prediction for Demand Forecast in Water Distribution Networks". In: *Energies* 11 (Mar. 2018), p. 660. DOI: 10.3390/en11030660.

[22] Jacob Lurie. "Review of Spectral Graph Theory: by Fan R. K. Chung". In: *SIGACT News* 30 (2 1999). ISSN: 0163-5700.

[23] Gustavo Meirelles et al. "Metamodel for nodal pressure estimation at near real-time in water distribution systems using artificial neural networks". In: *Journal of Hydroinformatics* 20 (Sept. 2017). DOI: 10.2166/hydro.2017.036.

[24] Olufemi Omitaomu and Haoran Niu. "Artificial Intelligence Techniques in Smart Grid: A Survey †". In: *Smart Cities* 4 (Apr. 2021), pp. 548–568. DOI: 10.3390/smartcities4020029.

[25] Garðar Örn Garðarsson, Francesca Boem, and Laura Toni. "Graph-Based Learning for Leak Detection and Localisation in Water Distribution Networks*". In: *IFAC-PapersOnLine* 55.6 (2022). 11th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2022, pp. 661–666. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2022.07.203. URL: https://www.sciencedirect.com/science/article/pii/S2405896322005882.

[26] D. I. Shuman et al. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". In: *IEEE Signal Processing Magazine* 30.3 (May 2013), pp. 83–98. DOI: 10.1109/msp.2012.2235192. URL: https://doi.org/10.1109%2Fmsp.2012.2235192.

[27] Yuichi Tanaka et al. "Sampling Signals on Graphs: From Theory to Applications". In: *IEEE Signal Processing Magazine* 37.6 (Nov. 2020), pp. 14–30. DOI: 10.1109/msp.2020.3016908. URL: https://doi.org/10.1109%2Fmsp.2020.3016908.

[28] Riccardo Taormina and Stefano Galelli. "Deep-Learning Approach to the Detection and Localization of Cyber-Physical Attacks on Water Distribution Systems". In: *Journal of Water Resources Planning and Management* 144.10 (2018), p. 04018065. DOI: 10.1061/(ASCE)WR.1943-5452.0000983. eprint: https://ascelibrary.org/doi/pdf/10.1061/%28ASCE%29WR.1943-5452.0000983. URL: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29WR.1943-5452.0000983.

[29] Riccardo Taormina et al. "Battle of the Attack Detection Algorithms: Disclosing Cyber Attacks on Water Distribution Networks". In: *Journal of Water Resources Planning and Management* 144.8 (2018), p. 04018048. DOI: 10.1061/(ASCE)WR.1943-5452.0000969. eprint: https://ascelibrary.org/doi/pdf/10.1061/%28ASCE%29WR.1943-5452.0000969. URL: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29WR.1943-5452.0000969.

[30] Lydia Tsiami and Christos Makropoulos. "Cyber-Physical Attack Detection in Water Distribution Systems with Temporal Graph Convolutional Neural Networks". In: *Water* 13 (Apr. 2021), p. 16. DOI: 10.3390/w13091247.

[31] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].

[32] Stelios G. Vrachimis et al. "LeakDB : A benchmark dataset for leakage diagnosis in water distribution networks". In: *Proceedings of the 2018 Joint Conference on Water Distribution Systems Analysis and Computing and Control for the Water Industry (WDSA/CCWI 2018)* (). DOI: 10.5281/zenodo.1313116.

[33] Zhuangkun Wei et al. "Optimal Sampling of Water Distribution Network Dynamics Using Graph Fourier Transform". In: *IEEE Transactions on Network Science and Engineering* 7.3 (2020), pp. 1570–1582. DOI: 10.1109/TNSE.2019.2941834.

[34] Z. Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32 (2021), pp. 4–24.

[35] Xiang Xie et al. "Compressed sensing based optimal sensor placement for leak localization in water distribution networks". In: *Journal of Hydroinformatics* 20 (Aug. 2017), p. 1286. DOI: 10.2166/hydro.2017.145.

[36] Lu Xing and Lina Sela. "Graph Neural Networks for State Estimation in Water Distribution Systems: Application of Supervised and Semisupervised Learning". In: *Journal of Water Resources Planning and Management* 148 (May 2022). DOI: 10.1061/(ASCE)WR.1943-5452.0001550.

[37] Si Zhang et al. "Graph convolutional networks: a comprehensive review". In: *Computational Social Networks* 6 (1 2019). ISSN: 21974314. DOI: 10.1186/s40649-019-0069-y.

[38] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *AI Open* 1 (2020), pp. 57–81. ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2021.01.001. URL: https://www.sciencedirect.com/science/article/pii/S2666651021000012.

[39] Yujue Zhou et al. "Graph convolutional networks based contamination source identification across water distribution networks". In: *Process Safety and Environmental Protection* 155 (2021), pp. 317–324. ISSN: 0957-5820. DOI: https://doi.org/10.1016/j.psep.2021.09.008. URL: https://www.sciencedirect.com/science/article/pii/S0957582021004742.