UNIVERSITÀ
DI PAVIA

Department of Electrical, Computer and Biomedical Engineering

Ph.D. Program in Electronics, Computer Science and Electrical Engineering

Doctoral Thesis

Neural Network-Based Methods for the Management and Control of Complex Systems

Supervisor:
Prof. Chiara Toffanin

Ph.D. Candidate:
Jorge Lo Presti

CYCLE XXXVII

A. Y. 2024-2025

Acknowledgements

My PhD journey comes to an end with this work, to which many people, directly or indirectly, have contributed. Their support, guidance, and encouragement have been invaluable, and I would like to take this opportunity to express my deepest gratitude.

First and foremost, I struggle to find words that can truly express my gratitude to my mother and my father for their constant support and for instilling in me the values that have shaped my journey and allowed me to reach this milestone. Likewise, I extend my heartfelt thanks to my aunt Maria for her immense love and support.

I am deeply grateful to my supervisor, Professor Chiara Toffanin, and to Professor Lalo Magni for granting me the invaluable opportunity to pursue my doctoral studies and for their academic guidance throughout these years.

I would like to thank Professor Enrico Creaco, for giving me the opportunity to work alongside him, which has been an inspiring and enriching experience.

I am immensely thankful to Professor Marco Forgione and Professor Dario Piga for welcoming me to IDSIA and for offering me an exceptional opportunity for both personal and professional growth.

A special thanks to Giacomo Galuppini and Carlo Giudicianni for their mentorship and wise advice.

I am also grateful to all my colleagues in the ICDS Lab, especially Diego, Francesca, Giacomo, Marco, Nikolas, Simone, Mori, Ayda, Irene, and Edoardo, for the lighthearted moments we shared in the lab.

Additionally, I wish to express my gratitude to all the wonderful people at IDSIA who made me feel welcome and at home during my visiting period in Lugano, especially Manas, Gabriele, Angelo, Marco, Asad, Milhad and Matteo.

I would like to thank my friends Andrea and Giuseppe for their loyal companionship and for always making things a little easier along the way.

I profoundly thank my brother Angelo Manfredi for always being by my side, in both good and bad times.

Finally, I want to thank Konstantina, whose smile, affection, and support have helped me overcome the greatest challenges of this journey.

Abstract

In response to the growing complexity of modern infrastructures and the escalating demand for enhanced reliability, efficiency, and safety, advanced methods for managing and controlling complex systems have become increasingly critical. Artificial intelligence and neural networks have emerged as powerful tools to address these challenges by offering efficient solutions for optimizing performance and mitigating risks.

This thesis investigates neural network-based methods for the management and control of complex systems, focusing on practical applications in critical fields such as water distribution and autonomous marine navigation. The primary objective is to develop advanced strategies for uncertainty management and efficiency improvement in these systems.

The first contribution presents an artificial intelligence-driven methodology for leak detection in water distribution networks, employing neural network regularization techniques to optimize sensor placement. This novel approach in the hydraulic context reduces the number of required sensors while maintaining high fault detection performance, leading to significant cost savings and improved network management.

The second contribution involves modeling nonlinear systems using neural networks integrated with a Bayesian approach for uncertainty quantification. Incorporated into a stochastic model predictive control schema, this method allows accounting for epistemic uncertainty of the model, resulting in more reliable and efficient control actions. The proposed methodology was tested *in-silico* with the ship maneuvering process, demonstrating promising results.

In summary, the methodologies developed in this thesis not only advance the management of uncertainty and enhance the efficiency of complex systems but also offer scalable solutions with significant potential for adaptation in a wide range of critical applications.

Contents

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
AUROC	Area Under the Relative Operating Characteristic
AdaGrad	Adaptive Gradient
Adam	Adaptive Moment Estimation
BGD	Batch Gradient Descent
BNN	Bayesian Neural Network
BNSS	Bayesian Neural State-Space
BPTT	BackPropagation Trough Time
CCE	Categorical Cross-Entropy
CDF	cumulative distribution function
CE	Cross-Entropy
CM	Confusion Matrix
DL	Deep Learning
EPANET	Environmental Protection Agency Network Evaluation Tool
GR	Group Regularization
GRNN	Group Regularized Neural Network
LOOCV	Leave-One-Out Cross-Validation
LSTM	Long Short-Term Memory

MAE	Mean Absolute Error
MAP	Maximum A Posteriori
MBGD	Mini-Batch Gradient Descent
MCMC	Markov Chain Monte Carlo
ML	Machine Learning
MLP	MultiLayer Perceptron
MMG	Maneuvering Modeling Group
MPC	Model Predictive Control
MPC	Model Predictive Control
MSE	Mean Squared Error
NMPC	Nonlinear Model Predictive Control
NSS	Neural State-Space
OCP	Optimal Control Problem
P	Precision
R	Recall
RMPC	Robust Model Predictive Control
RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Network
ROC	Relative Operating Characteristic
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descen
SMPC	Stochastic Model Predictive control
WAA	Weighted Average Accuracy
WCCE	Weighted Categorical Cross-Entropy
WDN	Water Distribution Network
WGN	White Gaussian Noise

List of Figures

List of Tables

Chapter 1

Introduction

During the last decades, Artificial Intelligence (AI) has experienced rapid advancements, revolutionizing various fields such as computer vision [1], natural language processing [2], autonomous systems [3]. The ability of Artificial Neural Networks (ANNs) to model complex, nonlinear relationships ([4–6]) has established them a powerful tool in solving problems that were previously considered intractable [7]. This transformation has been largely driven by the development of deep learning architectures [1, 8–11] and optimization techniques [12], which allow neural networks to automatically learn and extract features from large datasets efficiently.

As the capabilities of AI have expanded, the application of ANNs in system management [13] and control theory [14–17] have emerged as promising area of research. In several fields, as in industrial automation [18–21], resource management [22–25], autonomous vehicles [26, 27] and healthcare [28–31], the use of ANNs has proven to offer enhanced efficiency, reliability, and decision-making accuracy. However, despite the progress made, challenges remain, particularly in dealing with uncertainties inherent in real-world systems [32, 33].

This thesis explores two main challenges: the detection of leaks and optimal sensor placement in Water Distribution Networks (WDNs) and the incorporation of epistemic uncertainty into neural networks for modelling nonlinear dynamical systems. These areas represent critical problems where ANNs, with their advanced modeling capabilities, can provide significant improvements over traditional methods. To support future research, some samples of the code used in this thesis can be found on the following GitHub page: https://github.com/JL0P.

1.1 Objectives

The primary objective of this thesis is to develop innovative methodologies based on ANNs to address complex problems in system management and control. Specifically, this research focuses on:

- **Leak Detection and Sensor Placement in WDNs**: Developing a neural network-based approach that integrates graph theory and spectral clustering to optimize the placement of sensors in WDNs. This aims to enhance leak detection capabilities while minimizing the number of sensors required.
- **Incorporating Epistemic Uncertainty in Constrained Stochastic Model Predictive Control**: Applying Bayesian theory to model epistemic uncertainty in ANNs, with a focus on improving robustness and reliability of ANN-based control algorithm.

1.2 Thesis Structure

To achieve these objectives, the thesis is structured as follows:

- Chapter 2 provides a comprehensive overview of ANNs, tracing their evolution from the *perceptron* to more advanced architectures such as *multilayer perceptrons* and *recurrent neural networks* [7]. This chapter sets the stage for understanding the methods developed in subsequent chapters.
- Chapter 3 focuses on the application of ANNs for leak detection and sensor placement in water distribution networks. By integrating spectral clustering with *group sparse regularization* [34], this chapter presents a novel methodology for optimizing sensor locations simultaneously generating a classification model for leak detection.
- Chapter 4 addresses the challenge of modeling and controlling complex nonlinear systems under uncertainty. It introduces Bayesian Neural Networks [35] for state-space modeling [36, 37] and their incorporation into a Stochastic Model Predictive Control framework [38], with a case study on ship maneuvering.
- Finally, Chapter 5 summarizes the contributions of this work, discusses its limitations, and outlines future research directions.

1.3 Relevance and Contributions

Overall, this thesis proposes several important contributions to the fields of AI, ANNs, and control systems. First, it proposes an innovative approach to simultaneously address sensor placement and leak detection in WDNs, leveraging graph theory and ANNs to improve leak detection efficiency ensuring reduced cost related to sensor installation and maintenance. Secondly, it extends the application of Bayesian Neural Networks to model predictive control, enhancing robustness by incorporating epistemic uncertainty as chance constraint inside a Stochastic Model Predictive Control algorithm. These contributions have the potential to improve resource management in utilities and enhance the reliability of control systems in various industrial applications.

Chapter 2

Artificial Neural Networks

ANNs have revolutionized the field of AI, enabling significant advancements across various domains such as computer vision, natural language processing, and autonomous systems. These technologies have surpassed traditional Machine Learning (ML) methods by providing superior performance in tasks requiring complex pattern recognition and data representation. The ability of Deep Learning (DL) models to automatically extract features from raw data has made them essential for developing intelligent systems capable of human-like performance. As a result, ANNs and DL are at the core of many cutting-edge AI applications, driving innovation and transforming industries in all fields.

In this chapter, the fundamental concepts and structures of ANNs are mentioned, starting with the *perceptron*, the simplest form of a ANN, and gradually moving towards more complex architectures like the MLP and advanced DL models as RNNs. The mathematical foundations, training processes, and practical applications of these models are explored, providing a comprehensive understanding of how they work and their significance in the field of AI.

2.1 Perceptron: The Birth of AI

Introduced by Frank Rosenblatt in 1957 [39], the perceptron is one of the earliest and most fundamental models of ANNs. Inspired by the biological neuron, the perceptron was designed as a simple binary classifier, capable of distinguishing between two classes based on input features. Rosenblatt's work aimed to develop machines that could learn and adapt in a manner analogous to the human brain, laying the groundwork for future developments in ANN research.

Initially, the perceptron garnered significant attention and optimism for its potential to solve various pattern recognition problems. However, its limitations, particularly its inability to solve problems that are not linearly separable (e.g., the XOR problem), were highlighted in [40]. Despite its limitations, the perceptron laid the foundational principles for the development of more sophisticated ANN models and learning algorithms.

2.1.1 Structure of a Perceptron

A perceptron (Fig. 2.1) is a simple neural network unit that models a single neuron. It consists of the following components:

- **Inputs:** The perceptron receives multiple input vector $u \in \mathbb{R}^{n_u}$, where n_u denotes the number of input features. Each component u_i with $i = 1, 2, \ldots, n_u$ represents a feature of the data.
- **Weights:** Each input u_i is associated with a weight w_i which represents the strength or importance of the corresponding input. The weights $w \in \mathbb{R}^{n_u}$ are adjustable parameters that the perceptron learns during training.
- **Bias:** A bias term (b) is added to the weighted sum of the inputs to allow the perceptron to shift the decision boundary. The bias can be considered as an adjustable weight related to an additional input with a fixed value of 1.
- **Activation Function:** The perceptron computes a weighted sum of the inputs and the bias:

  $$z = \sum_{i=1}^{n_u} u_i w_i + b$$

  This sum is then passed through an activation function to produce the output that, in the case of the classical perceptron, is a step function which outputs 1, if $z \geq 0$, and 0 otherwise:

  $$\text{output} = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- **Output:** The output of the perceptron, y, is a binary value (0 or 1), which indicates the class label predicted by the model for a given input.

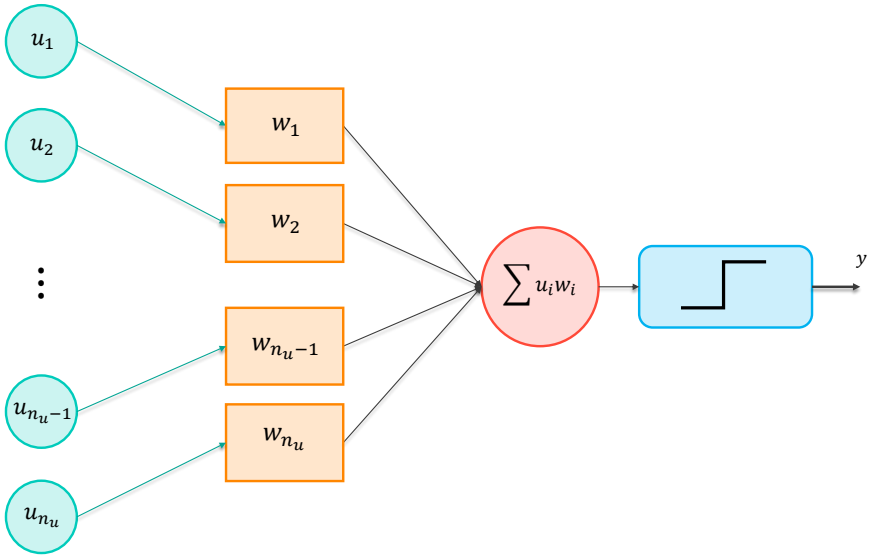

Figure 2.1: Graphical representation of the perceptron structure.

2.1.2 Limitations of the Perceptron

One of the main limitations of the perceptron is its inability to solve problems that are not linearly separable. A classic example is the XOR problem, where the input-output relationship cannot be separated by a straight line. The XOR function outputs true only when the inputs differ:

u_1	u_2	XOR(u_1, u_2)
0	0	0
0	1	1
1	0	1
1	1	0

In a Cartesian plane, no single line can separate the points where the XOR output is true (1) from those where it is false (0) (Fig. 2.2). This limitation is due to the perceptron linear decision boundary, which makes it incapable of solving problems requiring nonlinear separation. Despite its limitations, the perceptron serves as a building block for more complex architectures, such as the MLP, which overcomes many of the perceptron constraints and it is capable of solving a wider range of problems.

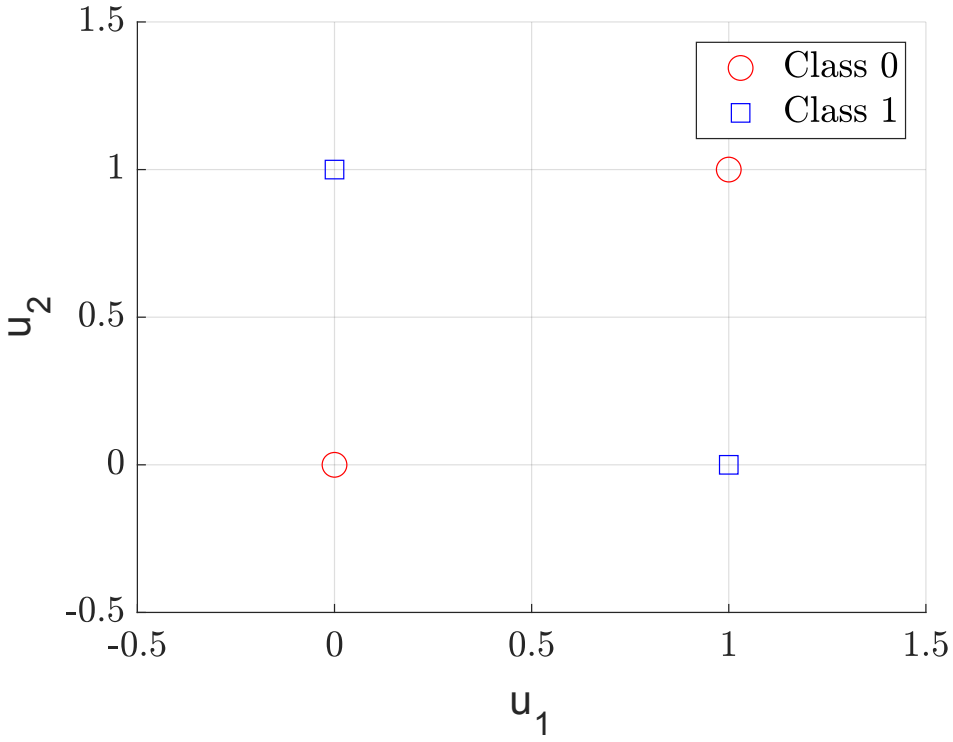

Figure 2.2: Visualization of the XOR problem. The classes are not linearly separable: it is not possible to draw a single straight line to separate the two classes.

2.2 MultiLayer Perceptron

The depth of an ANN, defined by the number of hidden layers of perceptrons (neurons), plays a crucial role in its ability to learn and represent complex patterns in the data [3]. Multiple hidden layers allow the network to build a hierarchy of features, with each layer capturing increasingly abstract representations [41]. This hierarchical feature learning is essential for tasks where raw input data is transformed into high-level concepts. Key advantages of deep networks can be summarized as follows:

- **Hierarchical Feature Learning:** Deep networks can automatically learn features at multiple levels of abstraction, reducing the need for manual feature engineering [42, 43].
- **Modeling Complex Functions:** With more hidden layers, deep networks can approximate complex and nonlinear functions, improving their performance on a wide range of tasks [44–46].

- **Generalization:** Typically, deep networks can achieve better performance on unseen data when trained on sufficiently large and diverse datasets, as their multiple layers allow them to capture a broad range of patterns and relationships. [41, 47].

However, simply increasing the depth of a network does not guarantee better generalization. Deeper networks also tend to overfit, especially if the training data is limited or not diverse [41]. Additionally, increasing the network depth brings new challenges, including higher computational demands and greater training difficulties due to issues like vanishing and exploding gradients. Advances in techniques such as batch normalization [48], dropout [49], and advanced optimization algorithms [50] have helped mitigate these challenges, making DL practical and effective for many applications.

A particularly noteworthy phenomenon observed in training deep neural networks is "double descent". The main idea is that the generalization error of a model does not always decrease monotonically as its complexity increases. Instead, as the number of parameters grows beyond the critical point where the model perfectly fits the training data, the error initially spikes before decreasing again as overparameterization progresses. This counterintuitive behavior challenges traditional views of the bias-variance tradeoff and highlights the unique dynamics of deep learning models, especially under overparameterization regimes [51, 52]. Understanding and leveraging this phenomenon can offer insights into designing more robust and effective networks.

In conclusion, the depth of a neural network is a critical factor in its ability to learn and model complex data. The use of multiple hidden layers in deep learning models has enabled significant breakthroughs in AI, driving advancements across various fields and applications.

2.2.1 Introduction to MLP

The MLP is an extension of the simple perceptron [43], designed to overcome its limitations, particularly its inability to solve nonlinearly separable problems. An MLP consists of multiple layers of neurons: an input layer, one or more hidden layers, and an output layer. Each neuron in a layer is connected to every neuron in the subsequent layer, forming a fully connected network (see Fig.2.3).

The MLP is capable of learning complex patterns and representations from the data due to its layered structure. A graphical representation of a MLP is depicted in Fig. 2.3 and the key components are:

- **Input Layer:** The input layer consists of neurons that are representative of the data input features: each input neuron corresponds to a feature.
- **Hidden Layers:** The hidden layers contain neurons that perform intermediate computations. Each hidden layer applies a linear transformation followed by an activation function to the inputs from the previous layer. The presence of multiple hidden layers enables the MLP to learn hierarchical representations of the data.
- **Output Layer:** The output layer produces the final predictions of the network. Based on the required task, the output function of the output layer should be chosen accordingly to suit the nature of the task, *e.g.*, regression or classification.

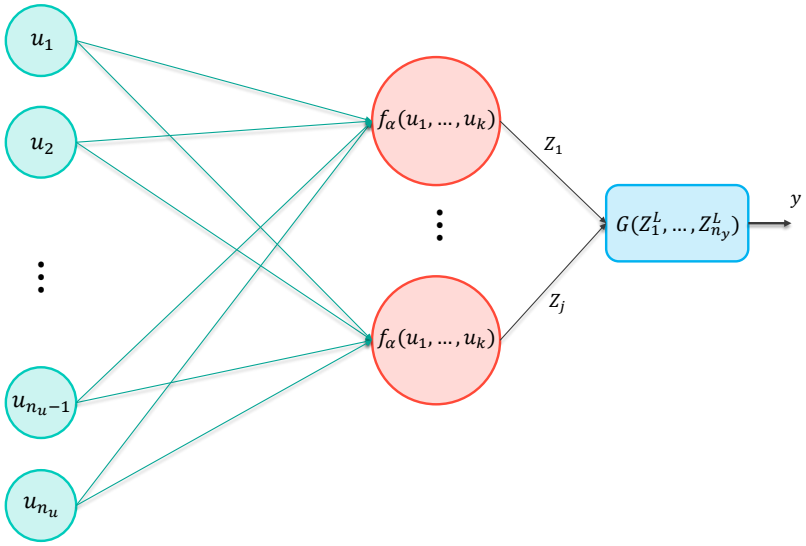

Figure 2.3: Graphical representation of an MLP model.

The primary difference between an MLP and a single perceptron is the presence of one or more hidden layers, which allow the MLP to capture and model more complex relationships within the data. The ability to stack multiple layers of neurons and apply non-linear activation functions enables the MLP to approximate any continuous function [44], making it a powerful tool for a wide range of AI tasks.

In summary, the MLP addresses the limitations of the simple perceptron by introducing multiple layers of neurons, thereby enabling the network to

learn and represent more complex patterns in the data. This capability makes MLPs a fundamental building block in the field of deep learning and neural networks.

2.2.2 Forward Propagation

Forward propagation is the process by which data moves through the layers of the neural network to generate an output. It involves a series of computations from the input layer, through the hidden layers, and finally to the output layer. During forward propagation, each neuron in a layer receives inputs from the neurons in the previous layer, computes a weighted sum of these inputs, adds a bias term, applies an activation function, and passes the result to the next layer.

The model can be mathematically described by the following equations for each layer l:

$$Z_j^{(1)} = \sum_{\kappa=1}^{n_u} w_{\kappa j}^{(1)} u_\kappa + b_j^{(1)} \tag{2.1}$$

$$Z_j^{(l)} = \sum_{i=1}^{N_{l-1}} w_{ij}^{(l)} A_i^{(l-1)} + b_j^{(l)} \tag{2.2}$$

$$A_j^{(l)} = f_a(Z_j^{(l)}) \tag{2.3}$$

$$y = G(Z_1^{(L)}, ..., Z_{n_y}^{(L)}) \tag{2.4}$$

where u_κ is the input feature κ, with $\kappa = 1, \ldots, n_u$, where n_u is the number of inputs of the MLP, $w_{\kappa j}^{(1)}$ is the weight from input κ to neuron j in the first hidden layer, $b_j^{(1)}$ is the bias related to neuron j in the first hidden layer, $w_{ij}^{(l)}$ represents the weight from neuron i in layer $l-1$ to neuron j in layer l, with $l = 2, ..., L$. $b_j^{(l)}$ represents the bias term for neuron j in layer l, $A_j^{(l)}$ represents the activation of neuron j in layer l, considering $l > 0$, and f_a is a proper activation function. $Z_j^{(l)}$ represents the weighted sum of inputs to neuron j in layer l. N_l is the number of neurons in layer l and y is the output of the MLP provided by the output layer described with the function G. In the last layer $j = 1, ..., n_y$, where n_y is the number of outputs of the MLP.

2.2.3 Backpropagation

Backpropagation is a supervised learning algorithm used for training ANNs, including MLP models [43]. It involves computing the gradient of the loss function with respect to each weight by the chain rule [41, 53], allowing the network to update the weights in order to minimize the error. The backpropagation algorithm consists of the following steps:

1. Forward Pass: Compute the predicted output by passing the input data through the network.

2. Compute Loss: Calculate the loss by comparing the predicted output with the true one using a loss function J.

3. Backward Pass: Once the loss is computed, its gradient with respect to the network parameters needs to be determined. This is done by propagating the derivative of the loss backward through the network, starting from the final layer and moving towards the initial layers, using the chain rule. For the output layer ($l = L$):

$$\frac{\partial J}{\partial Z_j^{(L)}} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial Z_j^{(L)}}$$

while, for each hidden layers ($l < L$) in terms of the next layer:

$$\frac{\partial J}{\partial Z_j^{(l)}} = \sum_{k=1}^{N_{l+1}} \frac{\partial J}{\partial Z_k^{(l+1)}} \cdot \frac{\partial Z_k^{(l+1)}}{\partial Z_j^{(l)}} = \left(\sum_{k=1}^{N_{l+1}} w_{jk}^{(l+1)} \frac{\partial J}{\partial Z_k^{(l+1)}} \right) \cdot f'_\alpha(Z_j^{(l)})$$

where j is the index of the neuron in the current layer l, k is the index of the neuron in the next layer $l + 1$, G' and f'_α are the derivatives of the output and activation functions, respectively, N_{l+1} is the number of neurons in the next layer $l + 1$. Finally, it is possible to compute the gradients of the loss with respect to the weights and biases. For each weight $w_{ij}^{(l)}$ and bias $b_j^{(l)}$, the gradients are given by:

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \frac{\partial J}{\partial Z_j^{(l)}} \cdot A_i^{(l-1)}$$

$$\frac{\partial J}{\partial b_j^{(l)}} = \frac{\partial J}{\partial Z_j^{(l)}}$$

where i is the index of the neuron in the previous layer $l-1$, $A_i^{(l-1)}$ represents the activation from the previous layer $l - 1$.

4. Weight and Bias Update: Once the gradients are computed, the weights and biases are updated, commonly with a gradient descent optimization algorithm [41, 54]. Defining $\theta = \{w, b\}$ the set of parameters of an ANN, the updates for basic gradient descent are given by:

$$\theta \leftarrow \theta - \alpha \frac{\partial J}{\partial \theta}$$

where α is the learning rate.

This iterative update process guides the weights in the direction of the negative gradient, aiming to find the parameter values that minimize the loss function There are several variations of gradient descent, including:

- **Batch Gradient Descent (BGD):** In BGD, the gradients are computed using the entire training dataset [41, 54]. This approach provides a stable and accurate estimate of the gradient but can be computationally expensive for large datasets. Additionally, in the context of non-convex optimization problems, BGD is more susceptible to converging to local minima.

- **Stochastic Gradient Descen (SGD):** In SGD, the gradients are computed using a single training example at each iteration [41, 55]. This approach significantly enhances computational efficiency by reducing the processing load per iteration, enabling more frequent updates to the model parameters. Additionally, the intrinsic randomness introduced by this method plays a crucial role in mitigating the risk of converging to local minima, facilitating more robust exploration of the parameter space. However, one notable drawback of SGD is the increased volatility in the gradient estimates. Because the updates are based on individual data points rather than the entire dataset, the gradient direction can vary considerably between iterations. This variability introduces a higher level of noise into the optimization process, which can affect the stability and convergence rate of the algorithm.

- **Mini-Batch Gradient Descent (MBGD):** MBGD is a compromise between BGD and SGD [56]. The gradients are computed using a small batch of training examples. This approach balances the stability of BGD and the efficiency of SGD.

While these variants of gradient descent focus on adjusting the granularity of gradient calculations, more sophisticated optimization algorithms incorporate adaptive learning rate adjustments and momentum terms for more efficient weights update. These advanced methods are discussed in Section 2.5.1.

2.3 Recurrent Neural Networks

RNNs represent a powerful class of neural networks designed specifically for handling sequential data [57–59]. This chapter aims to explore the fundamentals, architectures, challenges, and applications of RNNs, providing a solid foundation for understanding their role and functionality. Sequential data are ubiquitous, encompassing everything from spoken language and written text to time-series. The ability to analyze and predict sequences is therefore crucial in many areas of research and industry. RNNs address this need by leveraging their inherent structure, which connects sequential inputs to form a directed graph along the temporal dimension. This architecture allows them to exhibit dynamic temporal behavior and process inputs of varying lengths, providing a significant advantage over fixed-size input models. MLP models, while effective in many scenarios, fall short when it comes to capture time-dependent structures and relationships within data. RNNs fill this gap by updating a hidden state as the network processes each element of an input sequence. This method not only preserves information from earlier time steps of the input sequence but also uses this information to influence the output, making RNNs particularly suited for tasks such as language modeling, where previous words provide context for understanding or predicting the next words, or time-series, where temporal dependencies are preserved throughout the sequence [60]. However, issues such as the vanishing and exploding gradient problems can hinder the training of traditional RNNs, especially when dealing with long sequences. This section will delve into these challenges and introduce a more sophisticated variant of RNNs, named Long Short-Term Memory (LSTM) network [10], which is designed to overcome these difficulties and improve the network ability to learn long-term dependencies.

2.3.1 Fundamentals of Recurrent Neural Networks

The core architecture of an RNN is described in Fig. 2.4 and involves a loop within the network that passes the hidden state from one step of the sequence to the next. At each time step t, the RNN takes two inputs: the current input u_t and the previous hidden state h_{t-1}. The output of the RNN at each step, h_t, is then computed based on these inputs. This output can serve multiple purposes: it can be used directly as an embedded output of the network, or it can be further processed to produce a final output y_t. The mathematical operations within an RNN are governed by a set of equations that update the hidden state and compute the output. These operations can be summarized as follows:

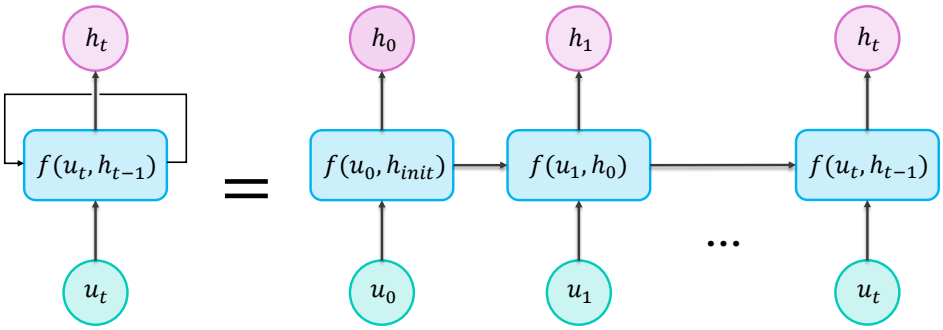

Figure 2.4: Graphical representation of an RNN, where h_{init} is the initialization value of the hidden state.

- **Hidden State Update:**

$$h_t = f(W_{hx}u_t + W_{hh}h_{t-1} + b_h)$$

where f_a is a nonlinear activation function, W_{hu} and W_{hh} are weight matrices for the input and the previous hidden state, respectively, and b_h is the bias term.

- **Output Calculation:**

$$\hat{y}_t = g(W_{hy}h_t + b_y) \tag{2.5}$$

where g is a suitable activation function, W_{hy} is the weight matrix for the hidden state to output transformation, and b_y is the bias term for the output.

2.3.2 Backpropagation Through Time

BackPropagation Trough Time (BPTT) is a technique used for training RNNs [41, 61, 62] and involves calculating the gradients of the loss with respect to the weights and biases of the network, taking into account the entire temporal input sequence.

As in the backpropagation algorithm, also in BPTT the first step is the forward pass, in which the output of the network is computed according

to equation 2.5. Then, the loss function for the RNN, given the predicted outputs $\{\hat{y}_1, \hat{y}_2, ..., \hat{y}_T\}$ is computed as:

$$J = \sum_{t=1}^{T} J_t(\hat{y}_t, y_t),$$

where T is the total number of time steps or sequence elements and J_t is a predefined loss function depending on the task.

Once the loss is computed, the backward pass involves calculating the gradients of the loss with respect to the weights and biases. The key calculations include:

- **Gradient Computation of the output**

  The gradient of the loss with respect to the output weights and biases:

$$\frac{\partial J}{\partial W_{hy}} = \frac{\partial J}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial W_{hy}} = \sum_{t=1}^{T} \left(\frac{\partial L}{\partial \hat{y}_t} \right) h_t^\top,$$

$$\frac{\partial J}{\partial b_y} = \frac{\partial J}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial b_y} = \sum_{t=1}^{T} \frac{\partial L}{\partial \hat{y}_t},$$

  where each term reflects the influence of the hidden state at time t on the prediction error.

- **Direct and Indirect Effects on Hidden State Gradients** Understanding the contributions to the hidden state gradient is crucial for grasping how past and future states influence learning:

  Direct Effect The direct effect measures how changes in the hidden state at time t impact the loss through the output at the same time step:

$$\frac{\partial J}{\partial h_t}\Big|_{\text{direct}} = \frac{\partial J}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} = \frac{\partial J}{\partial \hat{y}_t} W_{hy}^T,$$

  Indirect Effect The indirect effect captures the influence of the hidden state at time t on future states and, subsequently, on the loss:

$$\frac{\partial J}{\partial h_t}\Big|_{\text{indirect}} = \frac{\partial J}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} = \frac{\partial J}{\partial h_{t+1}} f_a'(W_{hh} h_t + W_{hx} u_{t+1} + b_h) W_{hh}^T, \quad (2.6)$$

  where f_a' is the derivative of the activation function.

Total Gradient The total gradient for the hidden state h_t is the sum of direct and indirect effects, enabling comprehensive updates to the model parameters:

$$\frac{\partial J}{\partial h_t} = \frac{\partial J}{\partial h_t}\Big|_{\text{direct}} + \frac{\partial J}{\partial h_t}\Big|_{\text{indirect}}.$$

- **Gradients for Hidden Weights and Biases:** These gradients are vital for tuning the network ability to maintain and transform hidden states across time:

$$\frac{\partial J}{\partial W_{hh}} = \sum_{t=1}^{T} \left(\frac{\partial J}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_{hh}} \right) = \sum_{t=1}^{T} \left(\frac{\partial J}{\partial h_t} \right) h_{t-1}^{\top},$$

$$\frac{\partial J}{\partial W_{hx}} = \sum_{t=1}^{T} \left(\frac{\partial J}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_{hx}} \right) = \sum_{t=1}^{T} \left(\frac{\partial J}{\partial h_t} \right) x_t^{\top},$$

$$\frac{\partial J}{\partial b_h} = \sum_{t=1}^{T} \left(\frac{\partial J}{\partial h_t} \cdot \frac{\partial h_t}{\partial b_h} \right) = \sum_{t=1}^{T} \frac{\partial J}{\partial h_t}.$$

showing how each weight matrix and bias term affects the loss through changes in the hidden states over all timesteps.

2.3.3 Challenges in Training RNNs

Training RNNs presents unique challenges that can significantly affect their training efficiency and performance. Two primary issues are the vanishing and exploding gradient problems and the high computational demands, especially when dealing with long sequences.

Vanishing and Exploding Gradient Problems

The vanishing and exploding gradient problems are prevalent in the training of RNNs due to their inherent use of BPTT for learning. These issues arise primarily because of the way gradients are computed and propagated backward through each timestep in the network ((2.6)). In the case of vanishing gradients, the gradients of the loss function with respect to the network parameters decrease exponentially as they are propagated backward through the network layers when multiple time steps are considered in the computation of (2.6). This occurs because the derivatives of certain activation functions (like the sigmoid or tanh) involve terms that can assume values smaller than 1. Multiplying these small numbers repeatedly through many

timesteps results in even smaller numbers, leading to gradients that effectively vanish. As a consequence, learning long-range dependencies becomes practically impossible [10, 63]. Conversely, exploding gradients occur when the gradients increase exponentially during backpropagation, which can happen with large weight values or high learning rates. This leads to gradient values that become too large, causing updates that make the learning process unstable. The weights may diverge, resulting in a model that fails to converge or exhibits erratic behavior [64]. Both vanishing and exploding gradients severely hinder the network ability to learn, as they affect the stability and speed of the convergence during training. The vanishing gradient problem makes it difficult for the RNN to capture long-term dependencies within the input data, which is critical for tasks such as language modeling and time series forecasting. On the other hand, exploding gradients can lead to a model that doesn't learn anything meaningful at all, as the weights overshoot their optimal values [41].

Computational Demands

The training of RNNs is computationally intensive, which becomes increasingly problematic as the length of the input sequences grows and this is due to several factors. First of all, unlike MLP networks, RNNs must process inputs sequentially according to their temporal order. This inherent characteristic prevents parallel processing of the inputs within a sequence, leading to longer training times as the sequence length increases [41]. Moreover, BPTT involves unrolling (Fig. 2.4) the network across each time step and computing gradients for each state backward from the final output to the start of the sequence. For long sequences, this results in a large computational graph, requiring substantial memory and processing power to store intermediate states and gradients [41]. Furthermore, as the length of the sequences increases, not only the computational requirements increase, but the training also becomes more susceptible to issues like overfitting, especially when there is a limited availability of training data. With long input sequences, maintaining stability in the gradients (avoiding both vanishing and exploding) becomes increasingly difficult [64]. These challenges necessitate the use of optimized hardware and algorithms. Techniques such as gradient clipping are commonly employed to combat exploding gradients, while gated architectures have been designed specifically to mitigate the vanishing gradient problem [10, 65]. Furthermore, in order to reduce the computational demand, more efficient versions of the training algorithm, such as the Truncated BPTT, have been proposed.

2.4 Advanced RNN Architecture: Long Short-Term Memory

LSTM networks represent a crucial development in RNN technology, addressing fundamental challenges that are inherent in traditional RNNs. Developed by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [10], LSTMs are specifically designed to overcome the vanishing gradient problem, enabling these networks to learn and maintain information over long sequences effectively. This capability makes LSTMs ideal for complex tasks in natural language processing, speech recognition, and time series prediction where understanding long-term dependencies is critical.

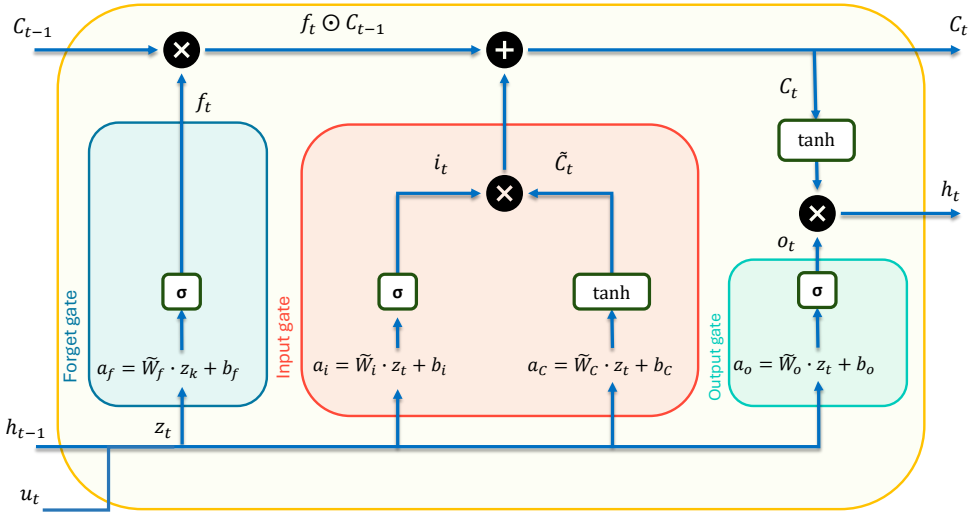

Figure 2.5: Graphical representation of an LSTM unit. Here, the input of the LSTM unit and the parameters matrices are reported using a compact notation $z_t = [h_{t-1}, u_t], \tilde{W}_f = [W_f, U_f], \tilde{W}_i = [W_i, U_i], \tilde{W}_c = [W_c, U_c], \tilde{W}_o = [W_o, U_o]$.

2.4.1 In-depth Analysis of LSTM Architecture

The architecture of a LSTM unit, depicted in Fig. 2.5, is much more structured compared to that of a standard RNN. Each LSTM cell is composed of various gates that meticulously manage the flow of information. These gates have the distinct functions of determining what information to preserve, what

to exclude, and what to ultimately transmit as output, all based on the context provided by incoming data stream.[66]. Considering an LSTM model with n_h hidden units, n_u inputs, and defining the hidden weight matricies as $W_f, W_i, W_o, W_C \in \mathbb{R}^{n_h \times n_h}$, the input weight matrices $U_f, U_i, U_o, U_C \in \mathbb{R}^{n_h \times n_u}$ and the bias $b_f, b_i, b_o, b_C \in \mathbb{R}^{n_h \times 1}$, it is possible to derive the model equations considering the different gates.

- **Forget Gate:** The forget gate, denoted as f_t, plays a critical role in modulating the preservation and omission of information from the previous cell state. It employs a sigmoid activation function to assess h_{t-1} (the prior hidden state) and u_t (the current input), producing a vector with values ranging between 0 and 1 for each element of the cell state:
$$f_t = \sigma(W_f \cdot h_{t-1} + U_f \cdot u_t + b_f)$$
Here, a value of 1 means complete preservation, while 0 indicates total removal of the corresponding component in the cell state.

- **Input Gate:** Concurrently, the input gate, represented by i_t, determines the extent of new information to be included in the cell state. This gate utilizes a sigmoid function to filter incoming data values, complemented by a hyperbolic tangent function that generates a vector of cell candidate values, $\tilde{C}_t$, representing possible additions to the cell state:
$$i_t = \sigma(W_i \cdot h_{t-1} + U_i \cdot u_t + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot h_{t-1} + U_C \cdot u_t + b_C)$$

- **Output Gate:** The output gate, o_t, dictates the composition of the forthcoming hidden state, which directly influences the output sequence generated or serves as input for subsequent LSTM units. This gate filters parts of the cell state to form the output using a sigmoid function:
$$o_t = \sigma(W_o \cdot h_{t-1} + U_o \cdot u_t + b_o)$$
The final hidden state, h_t, is then calculated as the Hadamard product ($\odot$) of the output gate activation and the hyperbolic tangent of the updated cell state:
$$h_t = o_t \odot \tanh(C_t)$$

- **Cell State:** The cell state, C_t, is the core memory mechanism of the LSTM, tasked with carrying relevant information through the sequence. It is updated at each time step by selectively forgetting certain aspects

of the previous state and incorporating new, relevant data as dictated by the forget and input gates:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

This dynamic update allows the LSTM to continuously adapt its stored information, shedding less relevant details and adding new, pertinent data, thus maintaining a critical balance throughout the sequence processing.

2.4.2 Addressing the Vanishing Gradient Problem

The design of LSTM gates allows for the selective remembering and forgetting of information [66], which significantly mitigates the vanishing gradient problem. Due to the incorporation of the cell state, the gradients in LSTMs have paths through time along which they can flow without necessarily passing through non-linear transformations, such as tanh or sigmoid functions. This architecture allows gradients to backpropagate through time more effectively without shrinking to zero, thus enabling LSTMs to learn dependencies from long input sequences, a critical capability for tasks involving extensive historical data [67].

2.5 Training Deep Neural Networks

Training deep neural networks is fundamental to develop models that can effectively process and elaborate data. This section provides an in-depth exploration of the key components involved in the training and validation phases of these complex models. At the heart of this process lies the optimization of network parameters, primarily achieved optimizing a proper loss function through the sophisticated mechanisms of backpropagation and optimization algorithms. This section will articulate the roles of state-of-the-art optimization algorithms that enhance the basic gradient descent framework, leading to improved convergence rates and superior model performance. Moreover, it will elaborate on the pivotal role of loss functions and metrics in order to train and validate the neural network models and the importance of regularization techniques, useful to ensure that the models not only perform well on training datasets but also generalize effectively to new, unseen data, is therefore discussed.

2.5.1 Advanced Optimization Algorithms

Advanced optimization algorithms enhance the basic gradient descent approach (discussed in Section 2.2.3), leading to improved convergence speed and model performance. The most widely used algorithms in DL are Root Mean Square Propagation (RMSProp), Adaptive Gradient (AdaGrad) and Adaptive Moment Estimation (Adam). These algorithms allow to dynamically adjust the global learning rate α and improve the efficiency and effectiveness of optimizing the model parameters θ.

- **AdaGrad** is an optimization algorithm that adapts the learning rate for each parameter based on the historical gradients. By accumulating the sum of the squares of the gradients, AdaGrad provides smaller updates for parameters associated with larger gradients. The update rule for AdaGrad is:

$$\theta \leftarrow \theta - \frac{\alpha}{\sqrt{G_{t,ii}} + \epsilon} g_t$$

  where, G_t is the sum of the squares of the gradients up to time t, and ϵ is a small constant added for numerical stability. AdaGrad is particularly useful for dealing with sparse data and has been successfully applied in natural language processing and other domains [68].

- **RMSProp** is an optimization algorithm that adjusts the learning rate for each parameter by dividing the global learning rate α by an exponentially decaying average of squared gradients. This method helps to address the issue of diminishing learning rates encountered in AdaGrad by allowing the learning rates to be adjusted dynamically based on recent gradient information. The update rules for RMSProp are:

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2$$

$$\theta \leftarrow \theta - \frac{\alpha}{\sqrt{v_t} + \epsilon} g_t$$

  where, v_t represents the moving average of the squared gradients, β is the decay rate, g_t is the gradient at time t. RMSProp has been particularly effective in training RNNs and is known for its simplicity and efficiency [69].

- **Adam** is another popular optimization algorithm that combines the benefits of both AdaGrad and RMSProp. Adam computes adaptive learning rates for each parameter by estimating the first and second

moments of the gradients. The algorithm maintains an exponentially decaying average of past gradients (first moment) and an average the squared gradients (second moment). This approach helps in stabilizing the learning process and achieving faster convergence. The update rules for Adam are as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$\theta \leftarrow \theta - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where, m_t and v_t are estimates of the first and second moments of the gradients, respectively, β_1 and β_2 are decay rates. Adam has been shown to work well in practice and is widely used in training deep neural networks [50].

These advanced optimization algorithms are crucial for the effective training of ANNs. By dynamically adjusting learning rates and incorporating additional techniques to stabilize the learning process, these methods significantly enhance the performance and convergence speed of the training of ANN models.

2.5.2 Loss Functions

Loss functions, also known as cost functions, are used to measure the discrepancy between the predicted outputs of the network and the true values. The objective of training an ANN is to minimize the loss function. Common loss functions include:

Mean Squared Error (MSE): Used primarily for regression tasks, MSE measures the average squared difference over N_m samples between the predicted values and the actual values y_i. It is defined as:

$$MSE = \frac{1}{N_m} \sum_{i=1}^{N_m} (y_i - \hat{y}_i)^2$$

Cross-Entropy Loss: Used for classification tasks, Cross-Entropy (CE) loss measures the difference between the predicted probability distribution and the true distribution. For binary classification, it is defined as:

$$CE = -\frac{1}{N_m}\sum_{i=1}^{N_m}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

Categorical Cross-Entropy Loss: For multi-class classification tasks, the most suitable loss function is the Categorical Cross-Entropy (CCE):

$$CCE = -\frac{1}{N_m}\sum_{i=1}^{N_m}\sum_{c=1}^{N_k}\mathbf{Y}_c^{(m)} \cdot \log\left(\hat{\mathbf{Y}}_c^{(m)}\right)$$

where, for the i-th sample in the dataset, $\mathbf{Y}^{(i)} = \left[Y_1^{(i)}\ Y_2^{(i)}\ \ldots\ Y_{N_k}^{(i)}\right]$ is the one-hot encoded ground truth vector containing the true probability distribution over N_k considered classes (*i.e.* 1 for correct class, and 0 for all the other elements), $\hat{\mathbf{Y}}^{(i)}$ is the predicted probability distribution vector.

In the case of class imbalance in the dataset, the relative frequency of each class can be explicitly considered in the computation of the Weighted Categorical Cross-Entropy (WCCE):

$$WCCE = -\frac{1}{N_m}\sum_{i=1}^{N_m}\sum_{c=1}^{N_k}\mu_c \cdot \mathbf{Y}_c^{(i)} \cdot \log\left(\hat{\mathbf{Y}}_c^{(i)}\right) \tag{2.7}$$

where μ_c is the weight assigned to class c, which is proportional to the inverse of the class frequency. This loss function quantifies the dissimilarity between the true class distribution Y and the predicted class distribution $\hat{Y}$, offering a quantitative measure of the classification effectiveness.

2.5.3 Activation Functions

Activation functions introduce nonlinearity into the ANN, enabling the network to learn and model complex patterns. Here, the common activation functions considered in this thesis are reported:

Sigmoid: The sigmoid (σ) activation function maps any input to a value between 0 and 1, making it suitable for binary classification tasks. It is defined as:

$$\sigma(u) = \frac{1}{1+e^{-u}}$$

Tanh: The hyperbolic tangent (tanh) activation function maps inputs to values between -1 and 1, centering the outputs around zero. It is defined as:

$$\tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

ReLU: The Rectified Linear Unit (ReLU) activation function is widely used in deep learning due to its simplicity and effectiveness. It outputs the input directly if it is positive; otherwise, it outputs zero. It is defined as:

$$ReLU(u) = \max(0, u)$$

These activation functions play a crucial role in the performance of ANNs by introducing nonlinearities that enable the network to learn complex patterns and representations from the data. The choice of activation function can significantly impact the training and convergence of the model.

2.5.4 Overfitting and Regularization

Overfitting occurs when a model learns the noise and details in the training data so that it performs poorly on new, unseen data. Regularization techniques are used to prevent overfitting by adding constraints to the model parameters. Considering an ANN with learnable parameters w, common regularization techniques include:

- **L2 Regularization (Ridge):** L2 regularization adds a penalty equal to the sum of the squared values of the weights to the loss function:

  $$J_{ridge} = J + \lambda \sum_i w_i^2$$

  where J is the original loss function, λ is the regularization parameter [53, 70, 71].

- **L1 Regularization (Lasso):** L1 regularization adds a penalty equal to the sum of the absolute values of the weights [72, 73]. :

  $$J_{lasso} = J + \lambda \sum_i |w_i|$$

- **Group Regularization (GR):** GR applies a penalty to the ℓ_2-norm of groups of weights that belong to the same input of a layer, effectively

encouraging sparsity within the model [34, 72, 74, 75]. Specifically, for a layer l and input u, let

$$\Omega_u = \left[w_{u,1}^{(l)} \, w_{u,2}^{(l)} \, \dots \, w_{u,N_l}^{(l)} \right]$$

be the vector of weights linking input u to all N_l neurons in layer l. Then, the GR penalty is added to the loss function obtaining

$$J_{GR} = J + \lambda \sum_{u=1}^{n_{u,l}} \sqrt{\dim(\Omega_u)} \cdot \|\Omega_u\|_2,$$

where $\dim(\Omega_u)$ is the cardinality of Ω_u, $\mathbf{\Omega} = [\Omega_1, \Omega_2, \dots, \Omega_{N_u}]$ and $n_{u,l}$ is the number of input to the considered layer l.

- **Dropout:** Dropout involves randomly deactivating a fraction of neurons during each training iteration, thereby reducing the network dependency on specific units and promoting a more robust and generalized feature learning process. This technique reduces overfitting and improves generalization [49].

- **Early Stopping:** Early stopping monitors the performance of the model on a validation set and stops training when the performance starts to degrade, preventing overfitting [76].

These regularization techniques help in improving the generalization capability of the ANNs, ensuring it performs well on new, unseen data.

In summary, training an ANN involves choosing an appropriate loss function, utilizing effective optimization techniques, and employing regularization methods to prevent overfitting, thereby enhancing the model's performance and generalization ability.

2.5.5 Normalization and Standardization

Normalization and standardization are crucial preprocessing steps in ML that transform features to a common scale without distorting differences in the ranges of values [41, 73]. These techniques improve the performance and training stability of ML algorithms by ensuring that each feature contributes equally to the model.

- **Standardization**

  Standardization transforms data to have a standard normal distribution with mean of zero and a standard deviation of one. This normalization process involves the following formula:

$$x_{norm} = \frac{x - \mu}{\sigma}$$

where x is the original feature value, μ is the mean of the feature, and σ its standard deviation. Standardization ensures that features with different scales contribute equally when processed by the model. Moreover standardization results effective for algorithms that assume a Gaussian distribution in the data.

- **Min-Max Normalization**

  Min-max normalization scales the data to a fixed range, usually [0, 1]. The formula for min-max normalization is:

$$x_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

  where $\min(x)$ is the minimum value of the feature, and $\max(x)$ its maximum value. Min-max normalization preserves the relationships between data points by maintaining their relative distances. Useful for algorithms that do not assume any distribution in the data.

- **Max-Abs Scaling**

  This normalization scales each feature by its maximum absolute value, which ensures that the transformed data will lie within the range [-1, 1]. The formula for max-abs scaling is:

$$x_{norm} = \frac{x}{\max(|x|)}$$

  where $\max(|x|)$ is the maximum absolute value of the feature. This normalization preserves zero entries in sparse data and maintains the sign of the data.

2.5.6 Initialization Techniques

Proper initialization of the network weights is crucial for effective training of deep ANNs. Poor initialization can lead to slow convergence or getting stuck in suboptimal solutions. Properly initialized ANN weights provides several key advantages that significantly enhance the training process. Firstly, it helps prevent the vanishing and exploding gradient problems, which are common issues that can severely impede the training of ANNs. By ensuring

that gradients do not become excessively small or large, proper initialization leads to more stable training [77, 78]. This stability is crucial for achieving reliable and consistent learning outcomes. Moreover, proper initialization facilitates faster convergence during training. By maintaining the appropriate scale of gradients, it allows the optimization algorithms to make effective updates to the weights, thereby speeding up the learning process [78]. This efficiency in convergence not only reduces the time required for training but also improves the overall performance of the network. Another important benefit is that proper initialization supports the training of deeper networks. By preserving the variance of activations across layers, it ensures that the signal does not diminish or amplify as it propagates through the network [79]. This preservation of variance helps in maintaining the integrity of the information being processed, allowing the network to learn more complex patterns and features effectively. As a result, deeper networks can be trained more efficiently, leading to improved performance on a wide range of tasks [80]. Common initialization techniques include:

- **Xavier Initialization:** Also known as Glorot initialization [77], it sets the weights to values drawn from a distribution with zero mean and a specific variance that depends on the number of input n_{in} and output n_{out} units. Xavier initialization aims to keep the scale of the gradients approximately the same in all layers. This method is particularly useful for networks with sigmoid or tanh activation functions. The formula for Xavier initialization is:

  $$\theta \sim \mathcal{N}\left(0, \frac{2}{n_{in} + n_{out}}\right)$$

  This technique helps maintain the variance of activations throughout the layers, which is crucial for preventing the vanishing or exploding gradient problem.

- **He Initialization:** He initialization [78], was specifically developed for ANNs that use ReLU activation functions. Since ReLU activation functions output zero for negative inputs, almost half of the output in a layer using ReLU are zero, leading to a reduction in the effective signal that propagates forward in the network. He initialization sets the weights to values drawn from a distribution with zero mean and a variance of $\frac{2}{n_{in}}$. By scaling the weights in this way, He initialization compensates for the signal loss caused by ReLU, preserving the input variance and promoting more stable training. The formula for He initialization is:

$$\theta \sim \mathcal{N}\left(0, \frac{2}{n_{in}}\right)$$

- **Orthogonal Initialization:** Orthogonal initialization initializes the weights to be orthogonal matrices. This technique helps to preserve the input variance across layers, which helps to avoid issues like vanishing and exploding gradients. The weights are initialized as follows:

  $$\theta \sim \mathcal{O}\left(n_{in}, n_{out}\right)$$

  where $\mathcal{O}$ represents an orthogonal matrix.

- **Lecun Initialization:** Lecun initialization [80] is another initialization method designed specifically for sigmoid and hyperbolic tangent (tanh) activation functions. It draws the weights from a distribution with zero mean and a variance of $\frac{1}{n_{in}}$. The formula is:

  $$\theta \sim \mathcal{N}\left(0, \frac{1}{n_{in}}\right)$$

  This method helps maintain the variance of activations and gradients.

In summary, proper initialization plays a crucial role in enhancing the training of neural networks by preventing gradient-related issues, ensuring faster convergence, and supporting the effective training of deeper architectures.

2.5.7 Hyperparameter Tuning

Hyperparameter tuning involves selecting the optimal set of hyperparameters for a model to maximize its performance. This process is crucial as it significantly impacts the effectiveness and efficiency of the training process and the final performance of the ANN. Common hyperparameters include learning rate α, batch size B_s, number of epochs N_e, and the architecture of the ANN, such as the number of layers L and the number of neurons per layer N_l. Several methods are used for hyperparameter tuning, each with its own advantages and limitations. Grid search is a traditional method that exhaustively searches over a specified parameter grid, evaluating the model for each combination of hyperparameters. Although comprehensive, this approach can be computationally expensive and time-consuming, particularly

in high-dimensional spaces [81]. Random search, on the other hand, samples hyperparameters from specified distributions. This method is generally more efficient than grid search, especially when dealing with high-dimensional spaces, as it does not evaluate all possible combinations but rather explores the hyperparameter space more broadly [82]. Bayesian optimization represents a more sophisticated approach, using a probabilistic model to find the optimal hyperparameters. This method builds a surrogate model of the objective function and iteratively updates it based on the results of previous evaluations, selecting the most promising hyperparameters to evaluate next. Bayesian optimization is particularly effective for tuning complex models with expensive evaluation functions [83]. In conclusion, training deep neural networks involves a series of critical steps, including data preprocessing, effective initialization, advanced optimization techniques, regularization methods, and careful hyperparameter tuning. Each of these steps plays a crucial role in enhancing the model's performance and ensuring its generalization to new data.

2.6 Evaluation and Metrics

Evaluating ANN models is a critical step in the ML pipeline, as it helps to understand the model performance and identify areas for improvement. Evaluation involves measuring how well the model generalizes to unseen data, ensuring it is not only effective on the training set but also performs well on new, real-world data.

2.6.1 Model Validation Techniques

Model validation techniques are used to evaluate the performance of a model by partitioning the data into multiple subsets, training the model on some subsets, and validating it on the remaining subsets. These techniques help ensure that the model generalizes well to unseen data and prevent overfitting.

- **Held-Out Validation:** Held-out validation, also known as holdout validation, is a straightforward method where the dataset is randomly split into two separate sets: a training set and a validation set. The model is trained on the training set and evaluated on the validation set. This method is simple to implement and fast to compute; however, it can be less reliable because the performance estimate is based on only one partition of the data, which may not be representative of the overall data distribution [73, 84].

- **Train-Validation-Test Split:** In this approach, the dataset is divided into three separate subsets: training, validation, and test sets. The model is trained on the training set, hyperparameters are tuned using the validation set, and the final evaluation is performed on the test set. This method provides a clear distinction between training, validation, and testing phases, which helps prevent overfitting and ensures that the model performance is assessed on unseen data. However, it requires a larger dataset to ensure sufficient data in all three subsets, and the choice of split can still introduce some bias [53].

- **K-Fold Cross-Validation:** In K-Fold Cross-Validation, the dataset is divided into K equally sized folds. The model is trained on K-1 folds and validated on the remaining fold. This process is repeated K times, with each fold used exactly once as the validation data. This method provides a more accurate estimate of model performance by averaging results across multiple folds and reduces bias introduced by a single train-test split. However, it is computationally more expensive than held-out validation and may still lead to overfitting [85, 86].

- **Leave-One-Out Cross-Validation (LOOCV):** LOOCV is a special case of K-Fold Cross-Validation where K equals the number of data points in the dataset. Each data point is used once as the validation data, and the model is trained on all other data points. This method provides an unbiased estimate of model performance and utilizes the maximum amount of data for training in each iteration, but it is very computationally expensive, especially for large datasets, and can be sensitive to outliers [73].

- **Stratified Cross-Validation:** Stratified cross-validation is used when the dataset is imbalanced. It ensures that each fold has a representative proportion of each class, maintaining the original class distribution. This technique is particularly useful for classification tasks where some classes are underrepresented. However, it is slightly more complex to implement than regular K-Fold Cross-Validation [86].

In summary, choosing the appropriate validation technique is crucial for building robust neural network models. Techniques like K-Fold Cross-Validation and Stratified Cross-Validation provide more reliable performance estimates by reducing bias and variance, whereas methods like Held-Out Validation and Train-Validation-Test Split offer simplicity and speed. By carefully selecting and applying these validation methods, we can ensure that our models generalize well to unseen data, leading to better performance and reliability in real-world applications.

2.6.2 Common Metrics

During training, loss functions (discussed in Section 2.5.2) are used to quantify the model performance on the training data. These loss functions are crucial for guiding the optimization process, as they are differentiable and allow for the computation of gradients necessary for updating the model parameters. However, once training is complete, additional evaluation metrics are employed to gain a comprehensive understanding of the model performance [41, 53, 87]. These evaluation metrics, which can be non-differentiable, provide deeper insights into various aspects of the model behavior. By using these metrics, it is possible to assess the model effectiveness in real-world applications and ensure it meets the desired performance criteria. Considering for simplicity a binary classification problem, and differentiating the model outputs as True Positive (TP) and True Negative (TN) as the correct predictions, while False Positive (FP) and False Negative (FN) as wrong predictions, some commonly used metrics are reported:

Average Accuracy: Accuracy is the proportion of correctly predicted instances out of the total instances. It is a useful metric when the classes are balanced.

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

However, when the classes are unbalanced, accuracy can be misleading, as it may not reflect the performance across all classes equally. To address this issue, we can compute the Weighted Average Accuracy (WAA) by accounting for the relative frequency μ_k of each class k

$$WAA = \frac{TP \cdot \mu_P + TN \cdot \mu_N}{TP + TN + FP + FN}$$

Precision: Precision (P) is the proportion of TP predictions out of all positive predictions made by the model. It answers the question: "Of all the instances classified as positive, how many are actually positive?"

$$P = \frac{TP}{TP + FP}$$

Recall: Recall (R) is the proportion of TP predictions out of all actual positive instances. It answers the question: "Of all the actual positive instances, how many did the model correctly identify?"

$$R = \frac{TP}{TP + FN}$$

Confusion Matrix: A Confusion Matrix (CM) is a table that summarizes the performance of a classification model. It shows the number of TP, TN, FP and FN, providing a comprehensive view of the model performance.

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

Table 2.1: CM for a binary classification problem.

F1 Score: The F1 score is the harmonic mean of P and R. It provides a single metric that balances both P and R, making it useful for imbalanced datasets.

$$F1 = 2 \cdot \frac{P \cdot R}{\mathrm{P} + \mathrm{R}}$$

Area Under the Relative Operating Characteristic Curve: The Relative Operating Characteristic (ROC) curve plots the true positive rate against the false positive rate at various threshold settings. The Area Under the Relative Operating Characteristic (AUROC) measures the area under this curve [88], providing an aggregate measure of performance across all thresholds. An AUROC of 1 indicates a perfect model, while an AUROC of 0.5 indicates a model with no discriminatory ability or errors in the original hypothesis.

Mean Absolute Error and Mean Squared Error : For regression tasks, Mean Absolute Error (MAE) and MSE are commonly used metrics. MAE measures the average magnitude of errors in the predictions without considering their direction, while MSE measures the average of the squares of the errors, giving more weight to larger errors.

$$MAE = \frac{1}{N_m} \sum_{i=1}^{N_m} |y_i - \hat{y}_i|$$

$$MSE = \frac{1}{N_m} \sum_{i=1}^{N_m} (y_i - \hat{y}_i)^2$$

R^2 Score (Coefficient of Determination): The R^2 score compares the variance of the residual with the variance of the dataset. By comparing these two quantities, the R^2 score tells us how much of the original variance in the data is explained by the model predictions. It ranges from 0 to 1, where 1 indicates perfect prediction and 0 indicates that the model does no better than the mean of the target variable. The formula for R^2 score is:

$$R^2 = 1 - \frac{\sum_{i=1}^{N_m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N_m}(y_i - \bar{y})^2}$$

where $\bar{y}$ is the mean of the actual values.

FIT Score: The FIT score is similar to the R2 score and measures the goodness of fit of the model, but applying the norm to the variances, it remains more stable to outliers. It is commonly expressed as percentage, ranging from 0% to 100%, where 100% indicates a perfect model and 0 indicates that the model does no better than the mean of the target variable. It is defined as:

$$FIT = 100 \times (1 - \frac{\sum_{i=1}^{N_m} ||y_i - \hat{y}_i||}{\sum_{i=1}^{N_m} ||y_i - \bar{y}||})\%$$

where the norm $||\cdot||$ can be the absolute value.

Chapter 3

Leak Detection and Sensor Placement in Water Distribution Network

This chapter focuses on the critical issue of burst detection and sensor placement within WDNs. Given the increasing challenges posed by aging infrastructure and rising urban demands, the need for efficient and reliable methodologies to manage water distribution is more pressing than ever. The chapter begins by examining the evolution of burst detection techniques, followed by an exploration of advancements in machine learning applications within WDN management. It also discusses the challenges associated with optimizing sensor placement for effective burst detection. The methodologies presented here lay the groundwork for the novel approach developed in this thesis, offering insights that are crucial for both theoretical understanding and practical implementation.

3.1 Introduction

WDNs are critical infrastructure systems that ensure the supply of potable water to urban populations. However, these networks are increasingly vulnerable to failures due to the aging of pipelines and the growing demand resulting from urbanization [89, 90]. These failures, often manifesting as bursts or leaks, contribute significantly to water loss, a problem that in the last decades is gaining global attention. The volume of non-revenue water, water that is produced but not billed to consumers, is estimated to reach 126 billion cubic meters annually, with a corresponding financial loss of approximately USD 39 billion [91]. In Italy, the National Institute of Statistics

(ISTAT) estimated that, in 2022, the total volume of water losses during the distribution to end users amounted to 4.7 billion cubic meters, representing 42.4% of the water introduced into the network [92]. Notably, it is estimated that around 30% of these water losses stem from pipeline bursts and leaks [93]. The consequences of such leaks are far-reaching. Beyond the direct loss of water and revenue, leaks can lead to public health risks through contaminant intrusion into the water supply [94, 95], increased operational costs due to energy wastage [96, 97], and consumer dissatisfaction from supply disruptions [98, 99]. Given these challenges, there is an urgent need for effective strategies to detect, localize, and manage leaks within WDNs. The prompt detection and repair of pipeline bursts not only minimize water losses but also contribute to the sustainable management of water resources [100]. Traditional methods of burst detection, such as manual inspections or basic pressure monitoring, are often insufficient due to their delayed response times and limited accuracy [101]. As a result, there has been a growing interest in leveraging advanced techniques, including ML and optimization algorithms, to enhance the efficiency of burst detection and sensor placement in WDNs. However, many existing approaches focus solely on either the detection of bursts or the optimization of sensor placement, without adequately addressing the integration of these tasks in a way that minimizes costs and maximizes detection accuracy simultaneously. In this chapter a novel methodology that integrates spectral clustering and Group Regularized Neural Networks (GRNNs) to simultaneously optimize burst detection, localization, and the placement of flow and pressure sensors within a WDN is proposed. The approach is designed to handle the complex and dynamic nature of WDNs by accounting for both spatial and temporal uncertainties in water demand and burst scenarios. By pre-clustering the WDN into well-defined detection areas, the method reduces computational complexity while ensuring comprehensive monitoring coverage. Additionally, the GRNN-based approach minimizes sensor redundancy and cost, making it a practical solution for real-world applications. The methodology is tested on a real-world WDN in Parete, Italy, where it demonstrates high detection accuracy across a range of burst sizes, even with a minimal number of sensors. In this chapter the development and application of this methodology will be described in details, offering insights into its potential for enhancing the resilience and efficiency of WDNs.

3.2 Methodology

The primary objective of the proposed methodology is to determine the most effective locations for flow and pressure sensors, facilitating the prompt detection of bursts within specific sub-areas of the WDN. A key aspect of this detection system is its ability to provide comprehensive network coverage while minimizing the number of sensors required, in consideration of economic and practical limitations. To achieve this, the WDN topology is initially utilized to establish a virtual clustering of the network (Section 3.2.1). To address the spatial and temporal uncertainties associated with nodal demands and burst locations, multiple scenarios are generated (Section 3.2.3). Ultimately, the placement of flow and pressure sensors within and between these virtual clusters of the WDN is optimized using a GRNN model, aimed at enhancing burst detection and localization efficiency (Section 3.2.5). A flowchart of the overall methodology is presented in Figure 3.1

3.2.1 Spectral clustering of the Water Distribution Network

The virtual clustering layout is established using a graph theory-based modeling approach, where the WDN is represented as an undirected and unweighted graph $\mathcal{G} = (V, E)$. In this graph, V denotes the set of n nodes (including junctions, reservoirs, and tanks), and E represents the set of m links (such as pipes, valves, and pumps). The creation of virtual clusters is guided solely by the connectivity characteristics of the graph, with the objective of forming well-balanced sub-areas that improve the accuracy of burst detection and localization. Importantly, this virtual clustering does not alter the hydraulic behavior of the WDN, as it does not involve the operation or closure of valves. Instead, the clustering serves to simplify the network and to define detection areas and boundary pipes that can be considered as optimal locations for flow metering devices. Accordingly, the method identifies sub-areas where connections are more concentrated, meaning that pipe burst events within these areas are likely to produce similar effects, such as pressure drops, on nearby nodes. This is because the pressure disturbance caused by a burst diminishes as it moves further from the source [102]. Moreover, the clustering allows for targeted attention on each sub-area individually, particularly for the placement of pressure sensors, and it enhances the precision of burst location classification. Identifying the specific area where an event occurs is highly advantageous for water utility companies, as it significantly reduces the workload, time, and costs associated with managing such incidents [103]. The clustering layout is obtained through the spectral clus-

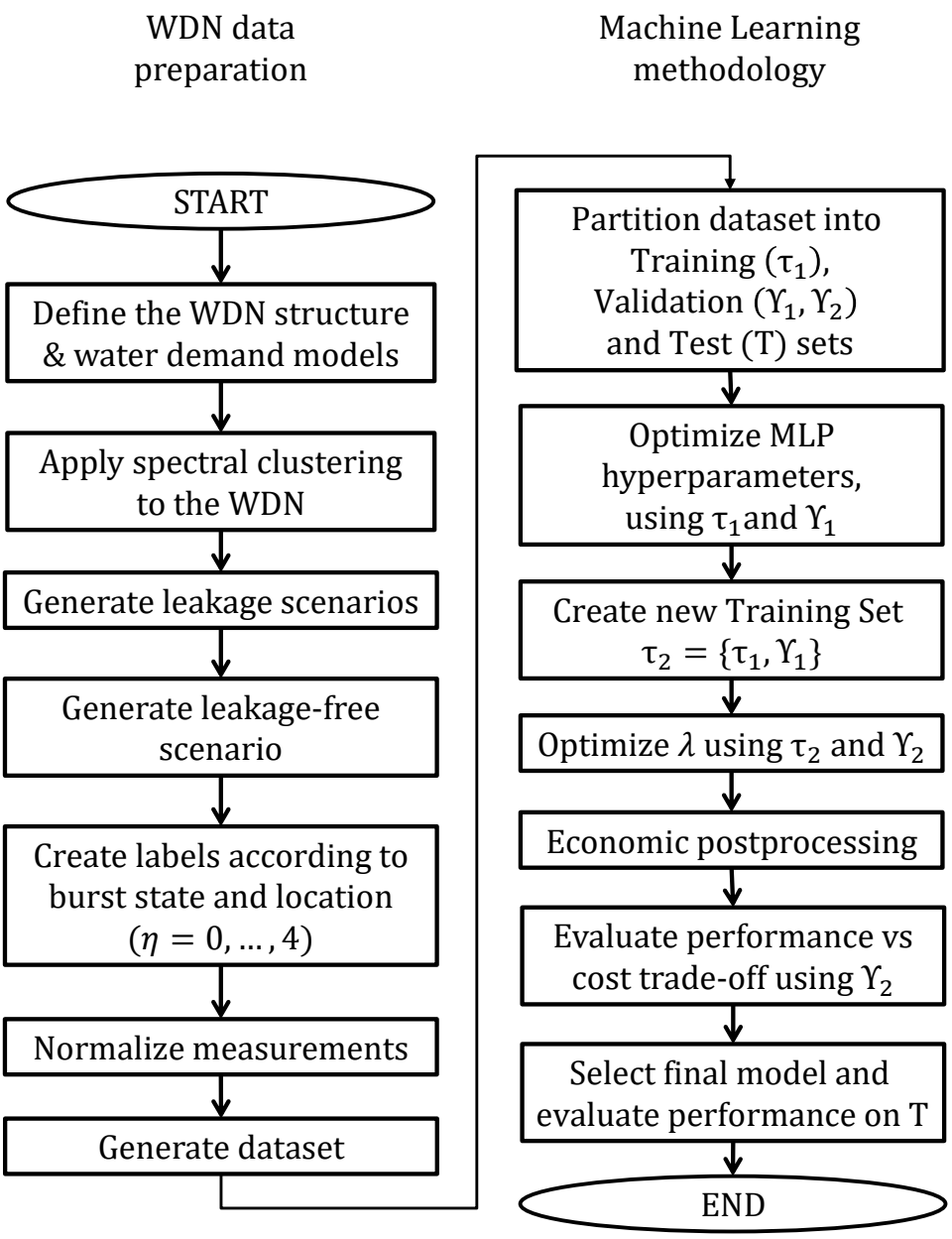

Figure 3.1: Flowchart of the proposed methodology.

tering algorithm [104], which has proved to efficiently assist WDN operators in different management tasks [105, 106]. This approach integrates principles from both linear algebra and graph theory, making use of the mathematical properties of eigenvalues and eigenvectors. The algorithm begins defining the *adjacency* matrix A

$$A_{ij} = f_a(n_i, n_j)$$

where a is representative of the presence of a link between the considered nodes and for unweighted graph

$$A_{ij} = \begin{cases} 1 & \text{if there is a link between nodes } n_i \text{ and } n_j, \\ 0 & \text{otherwise.} \end{cases}$$

In general, f_a is a similarity function that might incorporate nodes characteristic (see [105, 106]). Then, the *degree* matrix $\mathcal{D}$, which is diagonal, is derived

$$\mathcal{D}_{ii} = \sum_j A_{ij}.$$

Finally, the *laplacian* matrix Λ is computed subtracting the *adjacency* matrix from the *degree* diagonal matrix.

$$\Lambda = A - \mathcal{D}.$$

Hence, the spectral decomposition is performed in order to define the eigenvectors associated with Λ, solving:

$$\Lambda q_i = \zeta_i q_i,$$

where ζ_i are the eigenvalues and q_i are the corresponding eigenvectors. Once all the n eigenvectors are computed, the optimal number of clusters is determined by analyzing the eigen-gap, which refers to the difference between consecutive eigenvalues in the Laplacian spectrum. To achieve this, the eigenvalues of the Laplacian matrix are sorted in ascending order, and the optimal number of clusters, N_{cl}, corresponds to the largest eigen-gap between $\zeta_{N_{cl}}$ and $\zeta_{N_{cl}+1}$. Then, the eigenvectors corresponding to the first N_{cl} eigenvalues are selected in order to obtain a reduced embedded representation of the original space, obtaining the matrix:

$$Z = \begin{bmatrix} q_1(1) & q_2(1) & \cdots & q_{N_{cl}}(1) \\ q_1(2) & q_2(2) & \cdots & q_{N_{cl}}(2) \\ \vdots & \vdots & \ddots & \vdots \\ q_1(n) & q_2(n) & \cdots & q_{N_{cl}}(n) \end{bmatrix} \in \mathbb{R}^{n \times N_{cl}},$$

where n corresponds to the number of nodes in the graph and the rows of Z represent the N_{cl}-dimensional embedding of each node in the graph. When all the node information is embedded in the N_{cl}-dimensional space, the K-means algorithm [107] is applied to effectively partition the nodes into N_{cl} distinct clusters. The eigen-decomposition process is essential in revealing the intrinsic clustering structure of the graph, enabling K-means to identify clusters with higher accuracy and efficiency. This method leverages the distribution of the eigenvalues to achieve an optimal clustering configuration (see [108]). In addition to enhancing the accuracy of burst detection and localization, the clustering step significantly impacts the computational efficiency of the subsequent optimization process. Specifically, selecting the optimal number of clusters (based on the eigen-gap in the Laplacian spectrum) defines the boundaries of these clusters, which in turn determines the set of boundary pipes. These boundary pipes, representing critical points of flow communication between clusters, are identified as the most informative locations for the placement of flow meters. Consequently, the optimization of flow meter placement focuses on this smaller, more relevant subset of pipes, rather than on the entire set of WDN pipes, thereby reducing computational complexity without compromising performance. While considering only the boundary pipes in the optimization significantly reduces computational complexity, it is worth noting that this constraint may exclude other potentially informative locations within the network. Allowing sensors to be placed throughout the WDN could, in theory, enhance leak detection performance. However, such an unconstrained approach would also result in a substantial increase in computational effort and potentially higher deployment costs related to flow meters. Indeed, to balance detection performance with practical and economic considerations, this study focuses on boundary pipes as potential locations for flow meters. Concurrently, the placement of pressure sensors can be optimized within each cluster independently, allowing for more precise and efficient sensor deployment. Moreover, this strategy provides the added benefit of achieving a well-distributed spatial arrangement of sensors across the WDN. By concentrating on key locations that offer the most informative data, the methodology ensures comprehensive network coverage with a minimal number of sensors. This not only enhances the overall monitoring capability but also results in a significant reduction in investment costs. Similar strategies have proven effective in optimizing sensor distribution for water quality monitoring, where strategic placement has yielded cost-effective solutions with robust coverage [109].

3.2.2 Hydraulic simulation framework

Environmental Protection Agency Network Evaluation Tool (EPANET) is an open-source software package developed by the United States Environmental Protection Agency's Water Supply and Water Resources Division for modeling water distribution systems [110]. It enables extended-period simulation of the hydraulic and water-quality behavior within pressurized pipe networks, which include pipes, nodes (junctions), pumps, valves, storage tanks, and reservoirs. Using EPANET, users can simulate the flow of water through each pipe, determine the pressure at every node, monitor the water levels in tanks, and trace the movement of chemical substances within the network over time. The software is capable of tracking various water quality parameters, such as age of the water and source tracing, throughout the simulation period. The software features a graphical user interface that simplifies the construction and editing of network models. It provides a suite of data reporting and visualization tools, including color-coded network maps, data tables, time-series graphs, and contour plots, which assist users in interpreting simulation results effectively. EPANET has been extensively used in both research and practical applications related to water distribution systems. For example, it has been utilized in studies on leak detection and localization [111, 112], optimization of network design and operation [113–115], and analysis of water quality [116–118]. In this study, to account for uncertainties related to the spatial and temporal variability of nodal water demands, as well as the size and location of bursts, multiple leak and no-leak scenarios are generated using the EPANET-MATLAB Toolkit [119, 120]. This approach allows for a comprehensive analysis of various conditions within the network, enhancing the effectiveness of burst detection and localization methods.

3.2.3 Nodal water request generation

In order to develop and test the methodology, a realistic scenario describing the water demand across all nods of the WDN is needed. The typical daily demand pattern for the analyzed case study (it will be detailed in Section 3.4) is replicated over a 30-day period, which corresponds to the duration of the simulations. This pattern, usually assumed to be uniform across all nodes, serves as the foundation for generating the time-varying water demands. For each node, 720 coefficients are randomly generated, representing hourly variations over 24 hours across 30 days (24 hours $\times$ 30 days = 720 hours). These coefficients, which range between 0.5 and 1.5, function as amplitude and damping factors, shaping the monthly demand trend for each node.

Additionally, the base demand, which reflects the aggregated nominal water requirement for each specific WDN area, is also replicated over the 30-day period. To calculate the actual monthly water demand at each node, the daily pattern is multiplied by both the randomly generated coefficients (unique to each node) and the corresponding monthly base demand (unique to each node). This process effectively captures the spatial and temporal variability and uncertainty of water demands across the network, as illustrated in Figure 3.2. By integrating the typical daily behavior of the WDN with the randomly generated coefficients and the base demand trends, this approach provides a more realistic simulation of nodal water requests, accounting for both the predictable daily patterns and the inherent variability observed over a month.

The final monthly base demand pattern generated for each node is different, with the daily demand patterns varying potentially from one day to the next due to the influence of the 720 unique amplitude and damping factors. This variability ensures that each node exhibits a unique demand profile over the simulation period. The hydraulic behavior of the WDN, including node pressures and pipe flows, is simulated, incorporating these newly generated monthly water demand trends. This simulation represents the baseline, leak-free scenario (Scenario-0). The resulting data on node pressure and pipe flow are meticulously recorded and will later serve as a benchmark for comparison with various burst scenarios. Notably, since the search for flow meter positions is limited to boundary pipes (as determined in the previous clustering step) and transmission pipes downstream of the reservoirs, the flow data considered pertains exclusively to this subset of pipes.

3.2.4 Burst scenario generation

In this study, bursts are modeled as emitters and are assumed to occur individually at any node within the WDN, with the exception of nodes representing reservoirs. The leakage flow is calculated using the following equation:

$$q_{burst} = C \cdot P^{\gamma} \tag{3.1}$$

where q_{burst} represents the leakage flow rate [L/s], P is the node pressure [m], C is the discharge or emitter coefficient $[L/s/m^{0.5}]$ corresponding to the size or magnitude of the burst, and γ is the pressure exponent, which in this case is assumed to be 0.5. To account for bursts of varying magnitudes, the coefficient C is varied within the range of 0.1 to 0.9. This variation allows for the generation of a range of burst scenarios for each node, with the number of scenarios, E_{burst}, representing different burst sizes. Consequently, if n is the total number of junction nodes in the WDN, the total number of burst

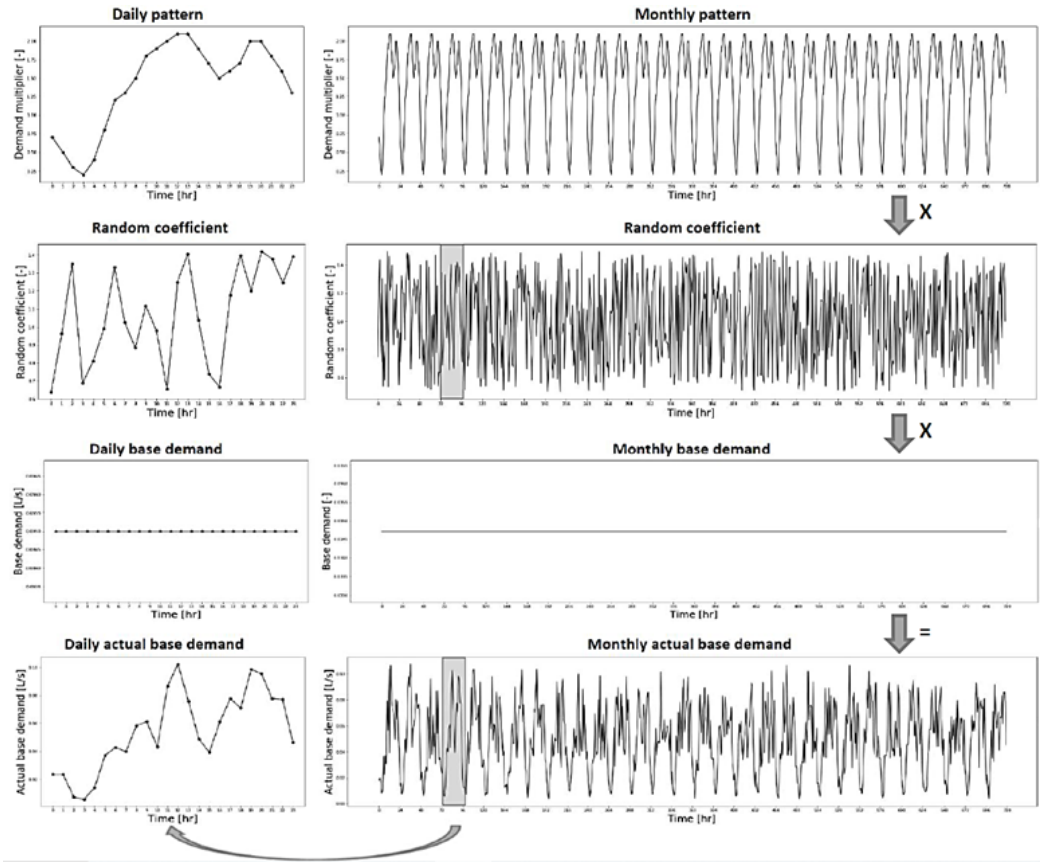

Figure 3.2: Graphical representation of the demand.

scenarios simulated, S_{burst}, is given by

$$S_{burst} = n \cdot E_{burst}. \tag{3.2}$$

Each of these burst scenarios is then simulated in the WDN model for a duration of 30 days, using the actual monthly water demand trends for each node. The resulting node pressure and pipe flow (only for the boundary pipes and transmission pipes downstream the reservoirs) are then stored. The collected data provides a comprehensive basis for analyzing the impact of different burst sizes on the hydraulic behavior of the network and serves as a benchmark for optimizing the placement of sensors.

3.2.5 Machine Learning framework

The primary objective of this research is to develop a robust burst detection model while simultaneously ensuring the optimal placement of sensors within the WDN. The burst detection is framed as a classification problem [75], where each class indicates either the presence or absence of a burst at the cluster level (see Section 3.2.1). Meanwhile, the optimization of sensor deployment is treated as a feature selection problem [75]. In this context, feature selection involves identifying and retaining only the most relevant input data, specifically, the node pressure and pipe flow measurements, while eliminating those that provide redundant information. This reduction in input data not only simplifies the classification model but also enhances its overall performance. By strategically selecting features, the methodology effectively determines the optimal locations for sensors across the WDN, minimizing redundancy and ensuring that only essential sensors are retained. This process is crucial for reducing the complexity of the model and improving the efficiency of both burst detection and sensor placement. The training of the classification model is based on in-silico data generated from the hydraulic simulations detailed in Section 3.2.2, where a MLP is employed for the classification task [7]. The use of an MLP model allows for an analysis of the WDN behavior leading to an accurate and reliable burst detection. To achieve effective feature selection and enhance burst detection accuracy, a GR technique is employed [34]. This technique plays a crucial role in identifying which of the candidate sensors provide the most relevant information based on the in-silico measurements. By applying GR, the methodology ensures that the sensors selected are those that contribute meaningfully to the detection and localization of bursts. A key advantage of incorporate GR technique in the training process is to simultaneously develop the classification model while performing feature selection. This dual functionality means that the selection

of relevant sensors is tightly integrated with the design of the classification model. In other words, the model is specifically tailored to leverage data from the chosen sensors, and the sensors are selected to maximize the model performance. This integrated approach represents a significant innovation in burst detection and localization strategies in WDN.

3.2.6 Multi-layer perceptron for multi-class classification

This section outlines the structure of the MLP employed in this study. As discussed in Section 2.2, the MLP design includes an input layer, several hidden layers, and an output layer. The input layer receives the initial data, while the hidden layers comprise neurons that perform computations using learnable weights and activation functions, as ReLU, to add nonlinearity and process complex patterns in the data. The output layer, using a softmax activation function, transforms the network raw output into a probability distribution over various classes, with the highest probability indicating the final classification decision of the network [7]. The training of the ANN is carried out by minimizing a suitable loss function J, by means of the MBGD optimization procedure with learning rate α [7]. In this study the WCCE (discussed in Section 2.5.2) was used as loss function for the training.

3.2.7 Early stopping and feature selection with group regularization

The primary goal of the methodology developed in this work is to minimize the number of sensors required within the WDN while still ensuring high performance in burst detection and localization. In order to achieve this goal, regularization techniques are employed to perform feature selection and to enhance the generalization capability of the machine learning model. In this work, the GR technique discussed in Section 2.5.4 has been implemented and applied to the input weights of the MLP. In this way, GR penalizes the weights associated with each input feature, which in this case correspond to flow and pressure measurements throughout the WDN. By doing so, it reduces the influence of less significant inputs, ensuring that the model prioritizes the most relevant data for burst detection. This process is visually represented in Figure 3.3, where the application of GR leads to a more streamlined and efficient model by discarding inputs that do not contribute meaningfully to the prediction task.

Through the use of GR, the methodology achieves a balance between

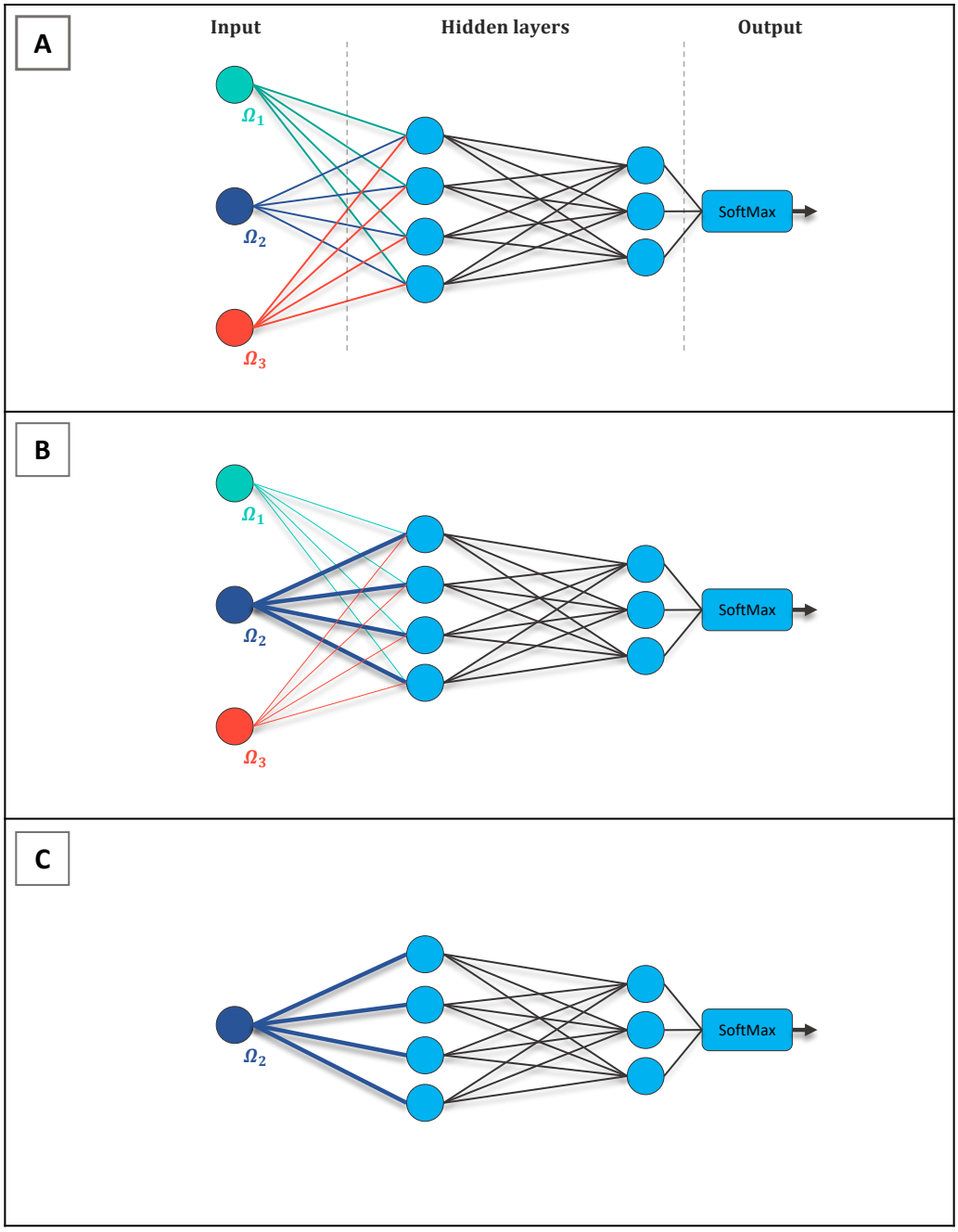

Figure 3.3: A simplified MLP structure, with three inputs (cyan, blue and orange), and two hidden layers with four and three neurons per layer, respectively: a) Dense architecture of the network; b) Effect of GR (weights associated to less informative/redundant inputs, namely cyan and orange, are reduced); c) Sparse architecture of the GRNN resulting from feature selection. Line thickness is representative of the magnitude of the norm-2 of the associated weights.

model simplicity and performance, allowing for the deployment of fewer sensors without compromising the accuracy or reliability of burst detection and localization. This is particularly valuable in the context of WDN management, where cost-effectiveness and operational efficiency are of paramount importance. In this work, the GR term $R_\lambda(\Omega)$ was applied to first hidden layer of the model with the aim of accomplish feature selection:

$$R_\lambda(\Omega) = \lambda \cdot \sum_{u=1}^{n_u} \sqrt{\dim(\Omega_u)} \cdot \|\Omega_u\|_2$$

where $\Omega_u = \left[w_{u,1}^{(1)}\, w_{u,2}^{(1)} \ldots w_{u,N_1}^{(1)}\right]$ is the vector containing the weights related to input feature u, having n_u input features and N_1 the number of neurons of the first hidden layer ($l = 1$). Hence, the R_λ term is added to the loss function WCCE, obtaining the regularized loss, J_{R_λ}:

$$J_{R_\lambda} = WCCE\left(\mathbf{Y}, \hat{\mathbf{Y}}\right) + R_\lambda(\Omega) \tag{3.3}$$

in this loss function, λ is a hyperparameter of the model that needs to be properly tuned. If λ is too small, the effect of the regularization term is minimal. Conversely, if λ is too large, it will result in an excessive penalization of the input weights of the ANN model, and in an excessive filter of the input information. Another regularization technique adopted in this work is the early stopping [76]. With the early stopping, as discussed in Section 2.5.5, the MBGD training algorithm stops when the performance of the ANN model, evaluated on fresh data, does not improve any longer. Early stopping helps reducing overfitting and improving the model generalization performance on unseen data, and reduces the computational time required to carry out the model training.

3.2.8 Model hyperparameters tuning

The performance of a MLP model is highly dependent on the selection of its hyperparameters. In this study, the key hyperparameters include the learning rate α, the number of hidden layers L, the number of neurons per hidden layer N_l, and the regularization constant λ. The learning rate α plays a crucial role in the optimization process by determining the step size during gradient descent, thereby influencing the balance between convergence speed and model stability. The structure of the network, defined by the number of hidden layers and the number of neurons per layer, dictates the model capacity to learn and capture complex patterns from the data. Additionally, the regularization constant λ is critical for controlling the trade-off between

model complexity and effectiveness, as it penalizes large weights to prevent overfitting while optimizing the model using the CCE loss function. To select the optimal hyperparameters, partition in training-validation-test method is employed, which is a widely used approach that avoids reliance on any statistical assumptions [121]. In this method, the available dataset is partitioned into separate training and validation subsets. The training set is used exclusively for training the MLP model, while the validation set is utilized to evaluate and select the best hyperparameter configuration. The performance of various hyperparameter combinations is assessed on the validation set, and the configuration yielding the highest validation performance is chosen. Additionally, a separate test set can be reserved for the final evaluation of the trained model. This test set remains unseen by the model during both training and hyperparameter tuning, providing an unbiased evaluation of the model generalization ability on new data. This partitioning strategy is essential in mitigating the risk of overfitting, ensuring that the model performance metrics accurately reflect its capability to generalize to unseen scenarios.

3.2.9 Model assessment

Evaluating the performance of a multiclass classification model requires specialized metrics that extend beyond those used in binary classification. These metrics provide a deeper understanding of how effectively the model distinguishes between multiple classes. In addition, when dealing with unbalanced classes, it is important to consider class abundance in the calculation of performance metrics to ensure a fair and comprehensive evaluation of the model capabilities. In this study, the WAA, discussed in Section 2.6.2, has been rearranged in order to deal with multiple classes and it has been employed for model validation and hyperparameter tuning. In particular, the WAA for a multiclass problem can be reformulated as follows

$$WAA = \frac{\sum_{k=1}^{N_k} \mu_k \cdot \hat{i}_k}{\sum_{k=1}^{N_k} i_k} \tag{3.4}$$

where N_k represents the total number of classes, $\hat{i}_k$ is the number of correctly predicted instances for class k, i_k is the total number of instances of class k in the ground truth, and μ_k is the weight assigned to class k, which is inversely proportional to the frequency of the class.

Furthermore, a detailed assessment of the model classification performance can be obtained through the use of CM, extended to encompass all classes, allowing for a thorough evaluation of the model predictive capabilities across all possible outcomes [122]. Then, to summarize the model

performance, and to provide insights into its effectiveness, P and R scores are computed.

3.3 Model development and sensor selection

This section provides a comprehensive overview of the unified methodology developed for classification model development and sensor selection within the WDN. The methodology is structured into five distinct steps:

1. Dataset processing and partitioning
2. MLP hyperparameters optimization
3. Regularization hyperparameter optimization
4. Economic post-processing
5. Model testing

A key aspect of this methodology is the two-step hyperparameter optimization process (steps 2 and 3). By separating the optimization of the ANN hidden structure from the regularization term, the methodology achieves two critical objectives: (1) a significant reduction in computational burden, and (2) a detailed evaluation of the GR effect on the ANN model, which is significantly usefull during the economic post-processing phase. Once a suitable model is selected (step 4), the methodology proceeds to the testing phase (step 5), where the model's performance is rigorously evaluated using different test datasets. Initially, model performance is assessed using fresh data generated with the same burst magnitude used during the training dataset generation. Subsequently, the model undergoes testing on new datasets generated with varying burst magnitudes to evaluate its generalization capabilities and robustness.

3.3.1 Dataset Processing and Partitioning

The methodology is built upon a dataset comprising synthetic pressure and flow measurements, as outlined in Section 3.2.2 Additionally, timestamps of the measurements, recorded in a 24-hour format, are incorporated as potential model input. The complete set of candidate inputs for the ANN model includes node pressure measurements at each node, pipe flow measurements in boundary and transmission pipes, and the hour of data collection. For holdout and testing purposes, the 30 days of scenarios per burst class are

partitioned into four different datasets. The first dataset, τ_1, consisting of 15 days per burst class, is used for training. The second dataset, Υ_1, containing 5 days per burst class, is used for model validation during step 2. For steps 3 and 4, another dataset, Υ_2, consisting of 5 days per burst class, is used as a validation dataset. Finally, the test dataset T_1, consisting of 5 days per burst class, is employed to assess model performance in step 5. To further evaluate the model's generalization capabilities, additional secondary test datasets T_2 are utilized. These datasets comprise three datasets T_{c2}, generated using three different burst magnitudes C, each consisting of 5 days per burst class. Note that τ_1 and Υ_1 are combined into τ_2 as the training dataset for steps 3 to 5, as both were used in step 2. A graphical representation of the dataset partitioning is provided in Figure 3.4.

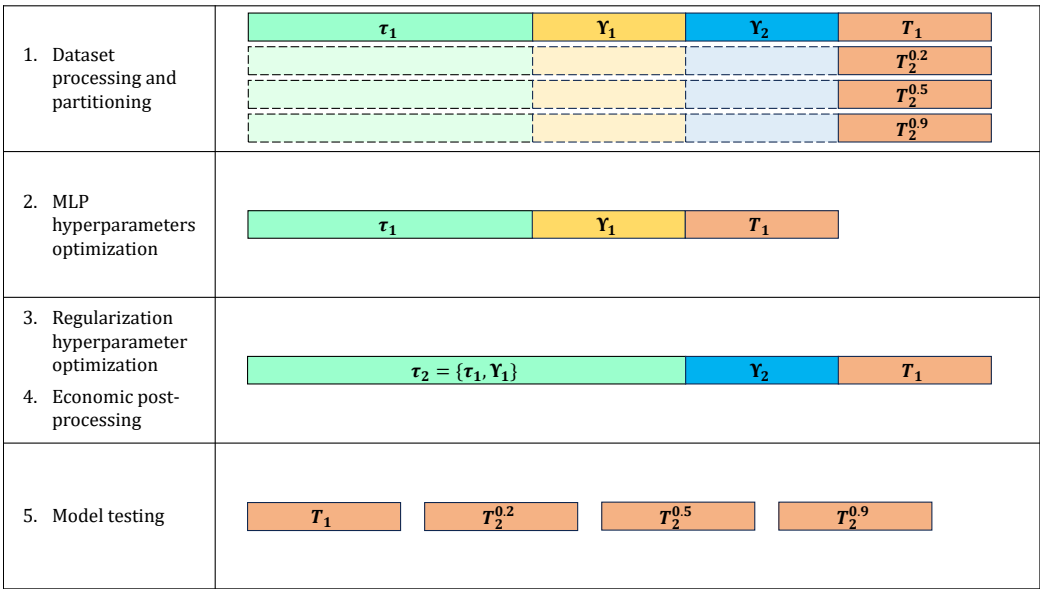

Figure 3.4: Dataset partitioning and usage at different steps of the methodology. τ_1, τ_2 (green) represents the datasets used for training the models. Υ_1 (yellow), the dataset used for the validation duering step 1 and 2. Υ_2 (blue) the validation dataset for step 3 and 4. All the dataset used for testing the models are depicted in orange.

Following dataset partitioning, a scaling process is applied to standardize the magnitude of the model inputs. Specifically, the scaling procedure is based on the maximum absolute value of the quantity as described in Section 2.5.5. The scaling factors are derived from the training dataset and are subsequently used to scale the validation and test datasets appropriately.

3.3.2 MLP hyperparameters optimization

To identify a preliminary structure for the MLP model suitable for the classification problem, an initial optimization process is conducted. Specifically, a Grid Search optimization is performed, which explores various configurations for the number of hidden layers (L) and the number of neurons per hidden layer (N_l). Additionally, the learning rate (α) for the MBGD optimizer is also fine-tuned during this stage. It is important to note that, at this phase of the methodology, all candidate inputs are taken into consideration, and the loss function WCCE is used throughout the training process. This comprehensive search ensures that the selected MLP structure has the capacity to effectively handle the complexity of the classification task.

3.3.3 Regularization hyperparameter optimization

After determining the optimal structure for the MLP and fine-tuning the learning rate, a subsequent optimization step is performed to assess the model performance as a function of the regularization parameter λ. To accomplish this, an exhaustive Grid Search is conducted over a range of λ values, allowing for a thorough exploration of the regularization space. In this phase of the optimization process, the complete loss function J_{R_λ} is employed to ensure that the model fits the data while penalizing excessive complexity. To further refine the selection of sensors, a sensor-specific coefficient $p_u \geq 0$ is incorporated into the regularization term. This coefficient is particularly important because it allows the model to take into account the cost associated with different types of sensors, like pressure and flow sensors. The inclusion of p_u enables the model to favor sensors that are both cost-effective and relevant to the detection and localization tasks. The final loss function $J_{R_{\lambda c}}$ that account for the extended regularization term $R_{\lambda c}(\Omega)$ is expressed as follows:

$$\begin{aligned} J_{R_{\lambda c}} &= WCCE\left(\mathbf{Y}, \hat{\mathbf{Y}}\right) + \lambda \cdot R_{\lambda c}(\Omega) \\ &= -\sum_{m=1}^{N_m}\sum_{k=1}^{N_k} \mu_k \cdot Y_k^{(m)} \cdot \log\left(\hat{Y}_k^{(m)}\right) + \lambda \cdot \sum_{u=1}^{n_u} \sqrt{\dim(\Omega_u)} \cdot \|\Omega_u\|_2 \cdot p_u \end{aligned} \tag{3.5}$$

Here, p_u is designed to represent the cost of input u, ensuring that the model considers the economic aspects of sensor deployment alongside the technical requirements. Although the default interpretation of p_u is related to sensor cost, this term can be adapted to incorporate other sensor-specific characteristics, such as reliability, accuracy, or maintenance requirements. This flexibility allows the methodology to be customized for different operational

contexts, making it a versatile tool for optimizing sensor networks in various scenarios.

As discussed in [34], while GR is effective in promoting sparsity among grouped weights, it does not inherently force the norm-2 of these grouped weights to be exactly zero. Consequently, an additional thresholding step is necessary to ensure that irrelevant inputs are entirely eliminated from the model. Specifically, after applying GR, the norms of the input weights are compared against a predefined threshold. Any input weights with a norm-2 value smaller than this threshold are set to zero, thereby removing the corresponding input feature from the model.

The determination of the threshold value follows a systematic process. First, the model that achieves the highest WAA on the Υ_2 validation dataset is selected as the reference model. Next, a set of candidate thresholds is defined. These candidates may include all available values or a logarithmically spaced grid ranging from the maximum to the minimum norm-2 of the input groups in the reference model. For each candidate threshold, the value is applied to the reference model, after which the remaining weights are fine-tuned to ensure optimal performance [7]. Finally, the highest threshold that results in a model with at least 99% of its original WAA is chosen. Once the optimal threshold is determined, it is applied across all models resulting from the λ optimization process. Then, these models are fine-tuned to ensure that they achieve the best possible performance under the newly imposed input constraints.

3.3.4 Economic Post-Processing

After the models corresponding to the various λ values in the explored grid have been generated and the relevant inputs have been selected as described in Section 3.3.3, the next step involves evaluating the economic implications of sensor deployment. Given that the cost associated with each sensor is known, it becomes possible to calculate the total expenditure for sensors corresponding to each λ value. The investment costs for purchasing the necessary sensors are summarized in Table 3.1.

In addition to the cost analysis, the performance metrics computed on the validation dataset Υ_2 are also considered. By integrating both the cost data and the performance metrics, a comprehensive cost/performance chart can be constructed (see Fig. 3.5). This chart allows for the evaluation of the various models generated across different λ values, enabling the identification of the optimal trade-off between cost and performance. The ability to visualize the balance between economic investment and model effectiveness provides crucial insights for making informed decisions about sensor deploy-

Table 3.1: Flow meter costs used for the case study.

Diameter (mm)	**Flow meter cost (€)**
60	2000
80	2073
100	2187
110	2250
125	2325
150	2586
200	2970

ment strategies.

3.3.5 Model testing

After selecting the model that best balances cost and performance, the next crucial step is to evaluate its generalization capabilities on new, unseen data. Initially, the model is tested using the T_1 dataset, which is generated using the same burst classes as those in the training and validation datasets. For this test, the discharge/emitter coefficient C in the leakage flow equation is set to values of 0.3 and 0.7, providing a range that the model has encountered during its development.

Subsequently, a secondary testing phase is conducted to further assess the robustness of the MLP model. In this phase, additional test datasets are generated using three new burst classes. Specifically, the discharge/emitter coefficient C is set to 0.2, 0.5, and 0.9, corresponding to the secondary test datasets $T_2^{0.2}$, $T_2^{0.5}$, and $T_2^{0.9}$, respectively. The datasets $T_2^{0.2}$ and $T_2^{0.9}$ are designed to challenge the model with burst magnitudes outside the range encountered during training, thereby testing the model ability to generalize beyond its training experience. Conversely, the dataset $T_2^{0.5}$ evaluates the model performance on burst magnitudes within the training range, but with data that was not seen during training, ensuring a reliable assessment of the final model.

For each of these datasets, two different CMs (Section 2.6.2) are generated. The first is a binary CM, which assesses the model ability to detect burst conditions within the WDN. The second is a multi-class CM, which evaluates the model ability to localize the bursts by considering the different clusters as distinct classes. These evaluations provide a comprehensive understanding of both the detection and localization capabilities of the MLP model.

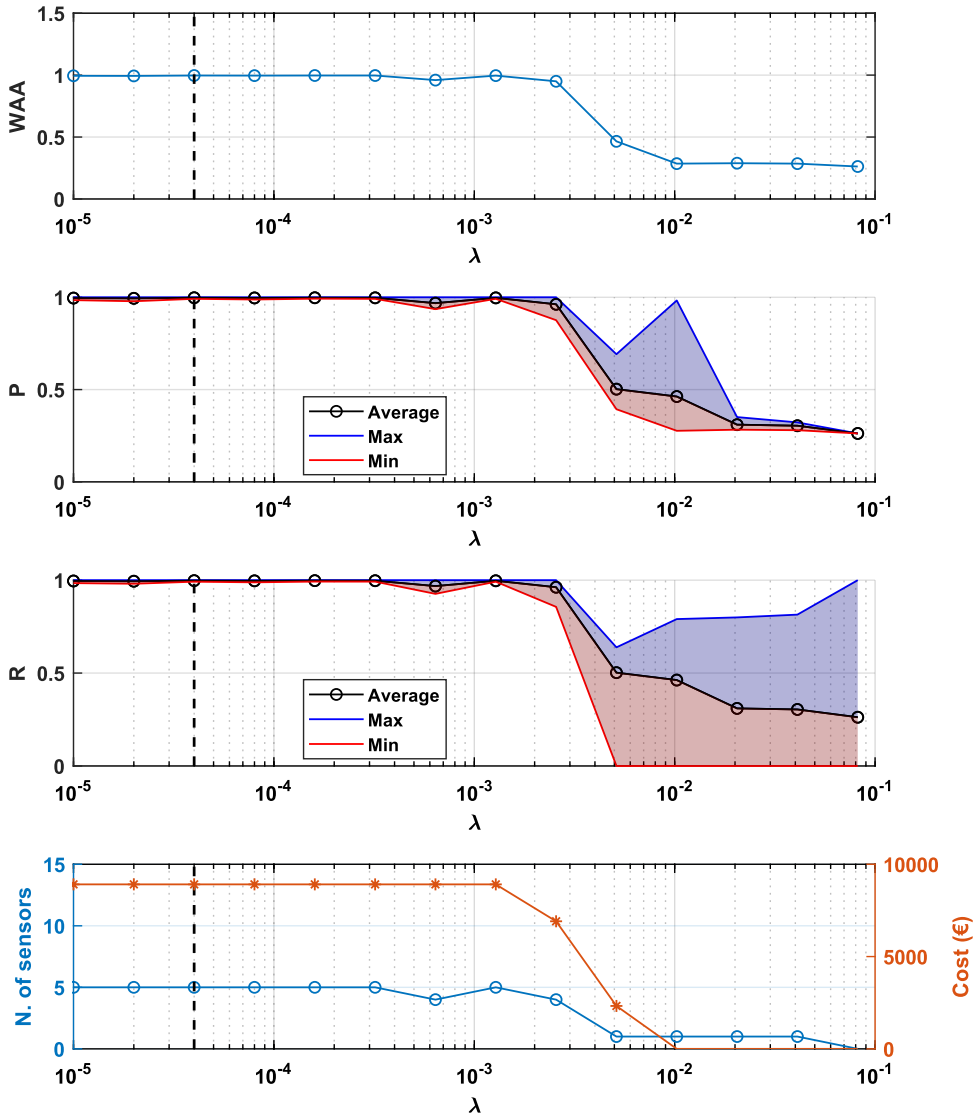

Figure 3.5: Cost/performance chart of the classification models for different values of λ. a) Weighted Average Accuracy; b) Average (black), minimum (red) and maximum (blue) Precision; c) Average (black), minimum (red) and maximum (blue) Recall; d) Number of considered sensors (light blue) and their overall cost (orange). The vertical, dashed line corresponds to the chosen model used in the testing phase.

3.4 Case study

For testing the proposed approach, the adopted case study is the WDN serving the city of Parete [123, 124] in Southern Italy, which has a population of approximately 11,000 inhabitants (see Fig. 3.6 for the layout). The WDN consists of 182 demand nodes with ground elevations ranging from 53 m a.s.l. to 79 m a.s.l., and 281 pipes made of cast iron, featuring a Darcy-Weisbach roughness coefficient of 0.26 mm. The pipe diameters range between 0.06 m and 0.20 m, with lengths varying from 10.4 m to 542 m, totaling 32.4 km in length. The network is supplied by two reservoirs (highlighted with black square symbols in Fig. 3.6) that maintain a fixed pressure head of 110 m a.s.l. The transmission pipes downstream of these reservoirs have lengths/diameters of 150 m/200 mm and 97 m/200 mm, respectively. To model typical daily variations in user demand, an hourly demand pattern is applied, with multiplier values ranging from 0.2 at 4:00 AM to 2.1 at 1:00 PM. Consequently, the total demand at the nodes varies from 7.34 L/s during the night to 77.14 L/s during morning and midday peaks, with an average demand of 51.89 L/s. The hydraulic model is implemented in the EPANET 2.2 environment [120]. For the economic analysis, a purchase cost of 500 € is assumed for each pressure sensor, while the cost of flow meters is variable and depends on the pipe diameter (as shown in Table 3.1).

3.5 Results

The application of a spectral algorithm and the eigengap method for selecting the optimal number of clusters enabled the definition of the clustering layout for the WDN of Parete (see Fig. 3.6). The network was divided into four well-balanced clusters ($N_{cl} = 4$), consisting of 43, 45, 48, and 48 nodes, respectively, along with 16 boundary pipes (colored red in Fig. 3.6). The two reservoir nodes are highlighted with a black square symbol.

For the nodal water requests, a unique monthly actual base demand pattern was generated for each of the 182 demand nodes, as detailed in Section 3.2.3, and used in the subsequent analysis. The results of the hydraulic simulation of Scenario-0 (the leak-free scenario) for the Parete WDN were stored in terms of pressure head at each node and flow through the 16 boundary pipes and the 2 transmission pipes downstream of the reservoirs. Consequently, the following two matrices were generated:

- **Pressure matrix**, 182×720 (number of nodes $\times$ number of hours in a month);

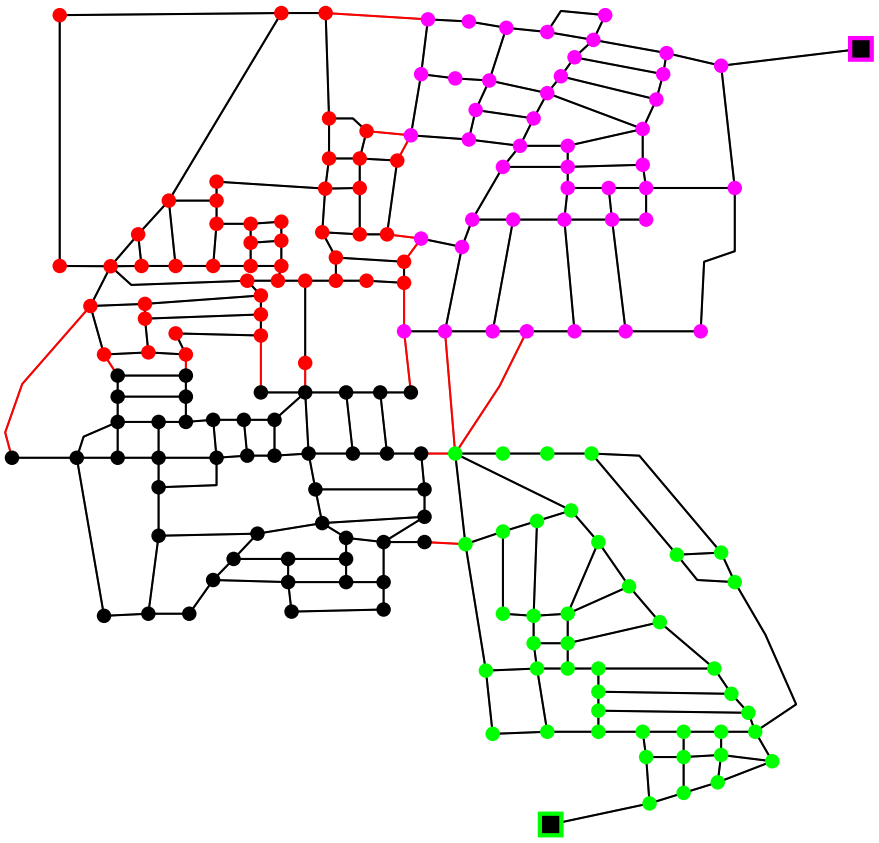

Figure 3.6: Clustering layout for the WDN of Parete: nodes are coloured according to the cluster they belong to; boundary pipes are depicted in red; and reservoir nodes are depicted as black squares.

- **Flow matrix**, 18×720 (number of boundary/transmission pipes $\times$ number of hours in a month).

Five different burst sizes were considered by varying the coefficient C in the leakage flow equation to 0.2, 0.3, 0.5, 0.7, and 0.9. Notably, two of these sizes (specifically, $C = 0.3$ and $C = 0.7$) were used in the first phase of the GRNN application for optimizing flow/pressure sensor placement, aiming to maximize burst detection/localization performance while minimizing information redundancy. It is important to note that, during this initial phase, the dataset was split into training, validation, and testing sets (as explained in Section 3.3.1 and further elaborated in the subsequent sections). Additionally, to further assess the reliability and effectiveness of the proposed approach, the monitoring layout was tested against scenarios generated using the three other never-seen burst sizes ($C = 0.2$, $C = 0.5$, and $C = 0.9$).

The hydraulic simulations of the 182 burst scenarios (one burst per node) were stored as two 3D matrices for each of the five burst sizes, capturing the pressure head at each node and the flow through the 16 boundary pipes and the 2 transmission pipes downstream of the reservoirs:

- **Pressure matrix**, $182 \times 720 \times 182$ (number of nodes $\times$ number of hours in a month $\times$ number of burst scenarios);
- **Flow matrix**, $18 \times 720 \times 182$ (number of boundary/transmission pipes $\times$ number of hours in a month $\times$ number of burst scenarios).

This results in a total of ten 3D matrices storing hydraulic information for $182 + 18 = 200$ elements across a total of $S_{\text{burst}} = 182 \times 5 = 910$ burst scenarios over a 30-day period, with data recorded hourly (720 intervals).

From these 3D matrices, datasets were derived for developing the MLP models. Every measurement from the WDN, along with the corresponding sample time, was used as a candidate input for the model. As discussed in Section 3.3.1, the hour of the measurement (h_{24}) was also included as a free input to the models. Furthermore, since a supervised learning procedure [7] was applied, a corresponding output dataset was generated. Specifically, a categorical variable η ranging from 1 to 4 was used to identify the cluster affected by the pipe burst. Additionally, the possibility of no-burst was accounted for by considering $\eta = 0$.

The input dataset $\mathbf{D}_{\text{in}}$ can be summarized as:

$$\mathbf{D}_{\text{in}} = [f_{R_1}, f_{R_2}, \ldots, f_{R_{18}}, p_{T_1}, p_{T_2}, \ldots, p_{T_{182}}, h_{24}]$$

where f_R represents the flow meter-related measurements and p_T denotes the pressure measurements.

The output dataset $\mathbf{D}_{\text{out}}$ is defined as:

$$\mathbf{D}_{\text{out}} = [\Phi(\eta)]$$

where $\Phi(\eta)$ is the one-hot encoding of η, with $\eta = [0, \ldots, 4]$.

Finally, these datasets were processed as detailed in Section 3.4 to train and test the MLP models. The hyperparameter optimization for the MLP structure was conducted as described in Section 3.3.2, using the loss function J and the dataset τ_1 for training and Υ_1 for validation. After an initial evaluation, the candidate values for the hyperparameters (α, L, N_l) were selected as shown in Table 3.2.

Table 3.2: Hyperparameters values used for the optimization of the NN.

Name	**Hyperparameter**	**Tested values**
Learning rate α	α	[0.0001, 0.00001, 0.000001]
Number of hidden layers	L	[1, 2, 3]
Number of neurons in layer l	N_l	[50, 80, 120]
Regularization factor	λ	$10^{-5}, 2 \times 10^{-5}$, $4 \times 10^{-5}, 5 \times 10^{-5}$, $0.00016, 0.00032$, $0.00064, 0.00128$, $0.00256, 0.00512$, $0.01024, 0.02048$, $0.04096, 0.08192$

The structure with $L = 3$, $N_1 = N_2 = N_3 = 50$, and $\alpha = 0.0001$ achieved a WAA close to 100% on the validation dataset Υ_1, making it a reasonable preliminary structure for the subsequent steps.

After determining a suitable set of MLP hyperparameters, the λ optimization was performed as discussed in Section 3.3.3, using the regularized loss function $J_{R_{\lambda c}}$ and employing the datasets $\tau_1 + \Upsilon_1$ for training and Υ_2 for validating the obtained models.

3.5.1 Selection of the best model

In the cost/performance chart shown in Fig. 3.5, the various models corresponding to different λ values can be compared to identify the optimal cost/performance trade-off. As illustrated in Fig. 3.5, increasing the magnitude of λ tends to reduce the classification performance in terms of a) WAA; b) Precision P; c) Recall R, as the regularization term forces the model to

discard a broader set of sensor measurements. The first three panels allow for an evaluation of the trained models performance, while the last panel depicts the cost and the number of sensors required for each λ. It becomes apparent that as the magnitude of λ increases, the number of sensors decreases, leading to a significant reduction in overall model performance due to the dominance of the regularization term in the computation of $J_{R_{\lambda c}}$. A balanced value of λ ensures that only the necessary number of sensors is selected to achieve satisfactory classification performance. The Cost/Performance chart provides a detailed visualization of how performance and cost evolve with λ. However, the final model selection remains unconstrained to allow for a comprehensive evaluation of each model individually. Notably, starting with 182 + 18 potential sensors, all final models use at most five sensors after optimization. In this study, the model corresponding to $\lambda = 4 \times 10^{-5}$, which ensures both the best performance (WAA = 0.996, $P = 0.997$, $R = 0.997$) and cost (9,500€), with a total of five flow sensors, was selected as the best trade-off.

3.5.2 First testing phase

In order to evaluate the generalization capabilities of the chosen model, it was tested against a test set generated using different burst magnitudes, as described in Section 3.3.1. Initially, the model was tested with the T_1 test set, which was generated using the same burst classes ($C = 0.3$ and $C = 0.7$) as those used for the training and validation datasets. In Fig. 3.7, the detection (left) and localization (right) CMs are presented. As discussed in Section 2.6.2, the diagonal elements of both matrices indicate correct model predictions, while the off-diagonal elements provide data for computing precision (bottom side array) and sensitivity (left side array) for each class.

The chosen model exhibited maximum precision and recall during the detection phase ($P = 100\%$, $R = 100\%$) for both the No-Burst and Burst classes. For this test, scenarios of 5 days for each simulated burst were considered, with 2 different burst magnitudes, resulting in a total of 43680 samples (5 days × 24 hours × 182 nodes × 2 entities). The No-Burst condition was tested by simulating a scenario of 5 days with no fault in the WDN (5 days × 24 hours × 2 entities), resulting in 240 samples.

In terms of localization performance (Fig. 3.7, right), despite a few incorrect predictions (21 for class 1, 93 for class 2, and 11 for class 4), the model consistently achieved high scores across all classes. Specifically, the results were as follows: $P = R = 100\%$ for class 0; $P = 99.2\%$ and $R = 99.8\%$ for class 1; $P = 99.8\%$ and $R = 99.2\%$ for class 2; $P = 99.8\%$ and $R = 100\%$ for class 3; $P = 100\%$ and $R = 99.9\%$ for class 4. These results highlight the model exceptional ability to accurately localize bursts within the network.

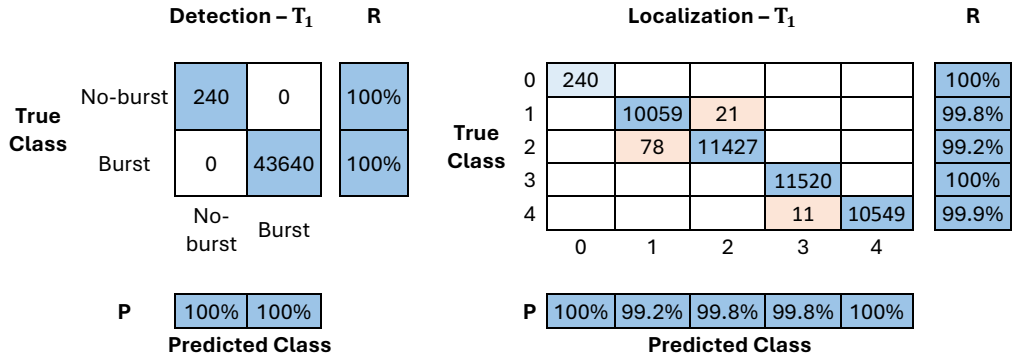

Figure 3.7: Confusion matrices for detection (left) and localization (right), for test sets generated with C = 0.3 and C = 0.7.

In the multiclass CM, each class corresponds to bursts occurring in nodes belonging to the same cluster, accounting for two burst entity values and five days of scenario simulation. For each class, a total of 5 days × 24 hours × 2 entities × N_n samples were considered, where N_n is the number of nodes in the cluster. For instance, the number of samples for which a burst was simulated in cluster 1 can be calculated by summing the elements in the second row of the localization CM, specifically $5 \times 24 \times 2 \times 42 = 10059 + 21 = 10080$ samples. The lowest scores were observed for class 2, where the model misclassified 78 instances of bursts in cluster 1 as being in cluster 2, leading to a precision of 99.2% for class 1 and a recall of 99.2% for class 2.

3.5.3 Second testing phase

Subsequently, the model was further evaluated using test datasets generated with burst coefficients $C = [0.2, 0.5, 0.9]$. Each of these test datasets contained 21840 samples representing burst conditions (5 days × 24 hours × 182 nodes × 1 entity), while only 120 samples were generated for the no-burst condition in the WDN (5 days × 24 hours). The results, illustrated in Fig. 3.8, demonstrate the model robust capability in detecting burst conditions across all the considered values of C.

It is evident, however, that the model performance slightly diminishes when detecting and localizing smaller bursts, as seen with $C = 0.2$ (Fig. 3.8-A), although small burst entities (as $C = 0.2$) are less impactant on the overall efficiency of the WDN. Specifically, the precision in detecting the no-burst condition within the WDN decreases to 80%. Additionally, the

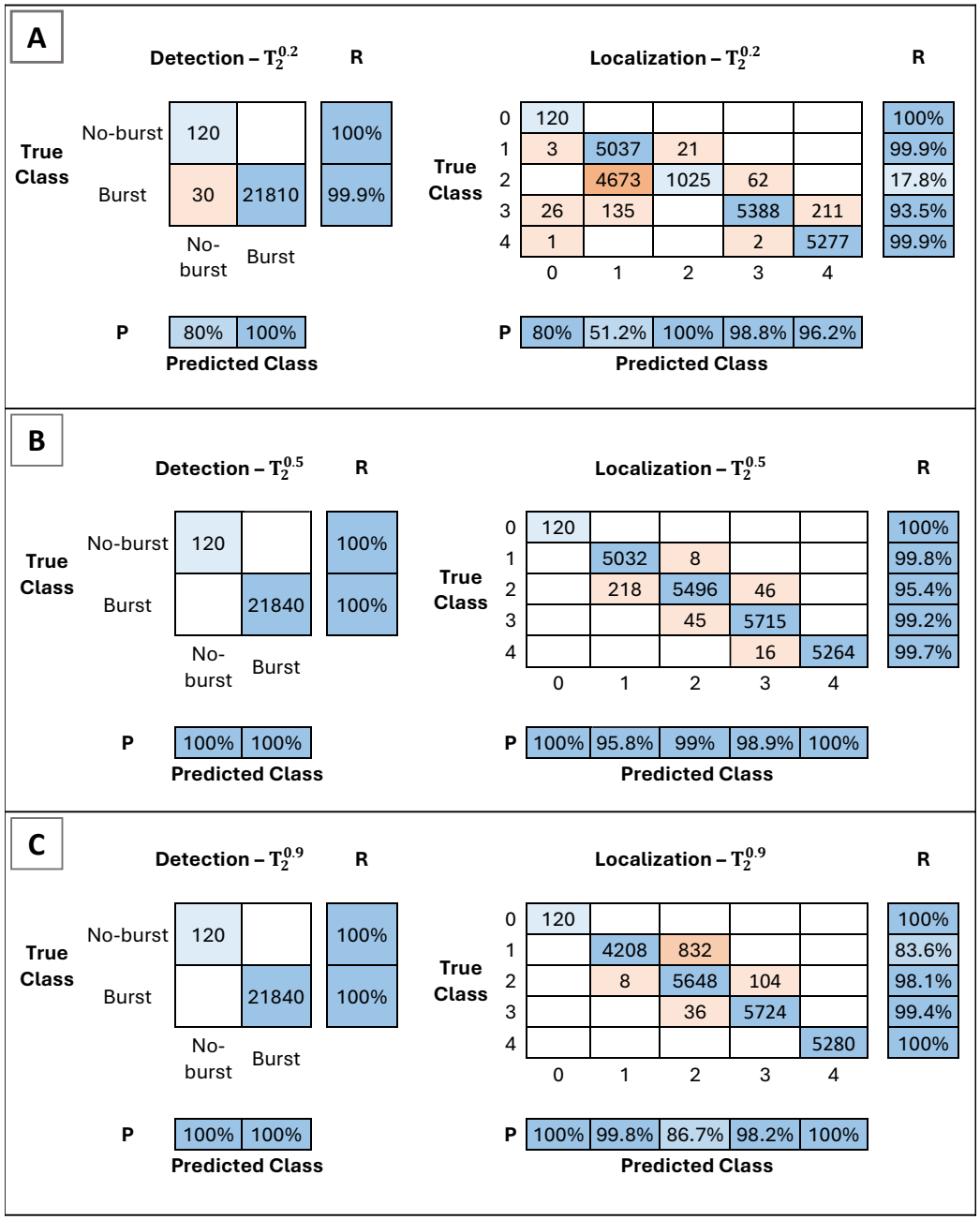

Figure 3.8: Confusion matrices for test sets considering: a) C = 0.2; b) C = 0.5; and c) C = 0.9.

localization performance is notably impacted by the model inaccuracies in distinguishing between bursts in cluster 2 and bursts in cluster 1, leading to a precision of 51.2% for class 1 and a recall of 17.8% for class 2.

Conversely, when tested with $C = 0.5$ (Fig. 3.8-B), the model exhibits high accuracy in detecting bursts, achieving both P and R values of 100%. In terms of localization, the model maintains strong performance across most classes, accurately classifying the location of the burst within the predefined clusters of the network. The high precision and recall values across clusters indicate reliable localization capabilities. These results confirm that bursts with values within the training range ($0.3 \leq C \leq 0.7$) are well-detected and localized.

Even with $C = 0.9$ (Figure 3.8-C), the model sustains its high detection accuracy. This consistency suggests that the model detection capability is robust across a wide range of burst sizes, including larger bursts that were not encountered during the training phase. For $C = 0.9$, both precision and recall remain high in the localization task, indicating the model ability to accurately identify the cluster of the network where the burst has occurred. Overall, the model ensures satisfactory performance on burst entities that were not included in the training dataset, regardless of whether they fall within or outside the range of burst entities used during training.

3.5.4 Results analysis

For each class, a detailed graphical representation of the testing phase, similar to the one shown in Figure 3.9, can be generated. Notably, Figure 3.9 offers comprehensive insights into the MLP model outputs, which are crucial for an in-depth evaluation of the model performance.

In the first panel (Figure 3.9-A), the probabilities assigned to each class are depicted, providing a clear view of the MLP model confidence in its predictions. This visualization also allows for the evaluation of alternative outcomes to the model final decision, offering a better understanding of its decision-making process. The second panel (Figure 3.9-B) compares the model final output to the correct class, enabling a quick overview of how the model predictions align with the true classes over time. Panel (C) in Figure 3.9 illustrates the ID of the WDN node where the burst occurred, which helps identify any potential model weaknesses related to specific areas of the WDN. The hour of the day at which the measurements were taken is shown in panel (D) of Figure 3.9, which is useful for analyzing any correlations between model errors and the time of occurrence. Finally, panel (E) of Figure 3.9 represents the burst magnitude, providing further context for the analysis. It is worth to notice that Figure 3.9 is specifically related to

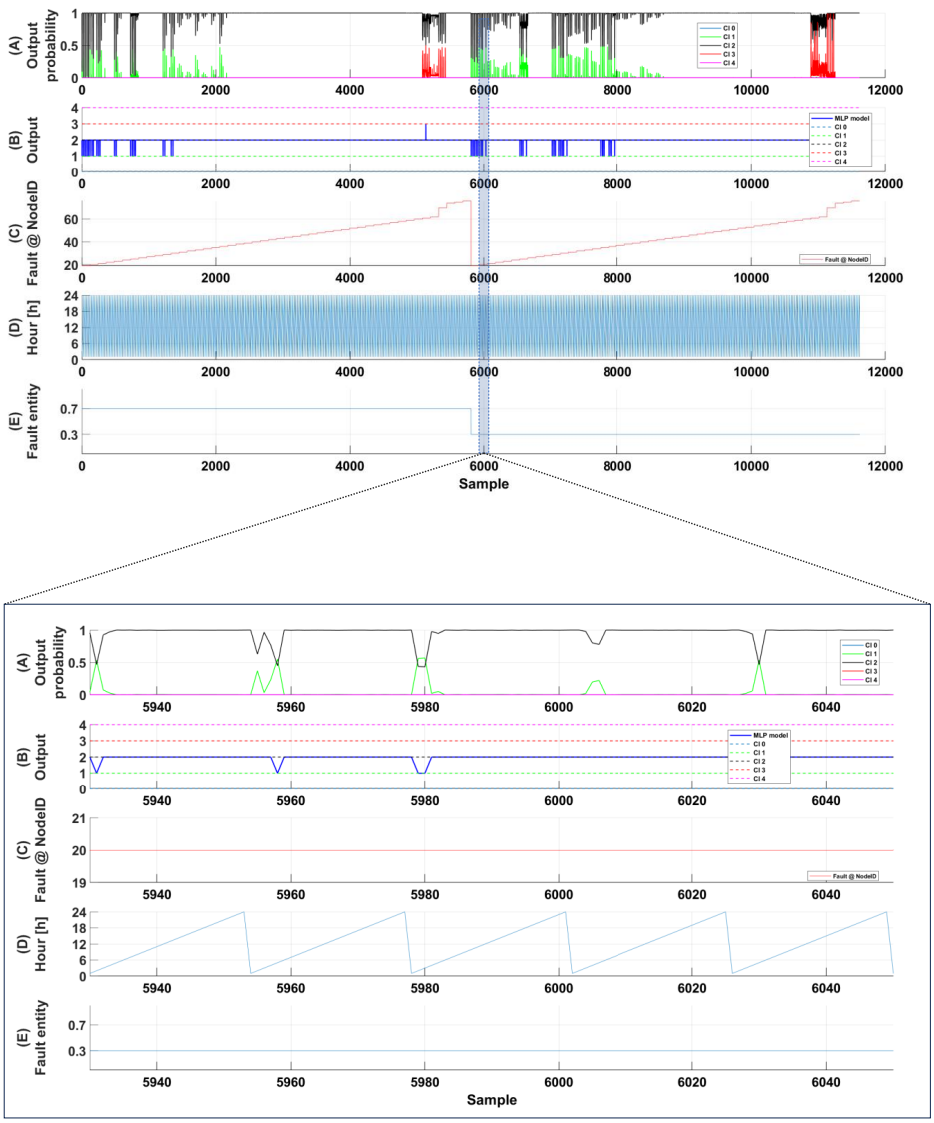

Figure 3.9: Analysis related to test set T1, when burst in cluster 2 is present. A zoomed view related to fault at node ID = 20 is provided. A) Classification probabilities by the MLP model. B) MLP predicted class (maximum classification probability). C) WDN node at which fault occurs. D) Hour of the measurements. E) Fault entity.

the model output for the T_1 test dataset. As previously discussed, with T_1, the model misclassified bursts in cluster 1 as belonging to cluster 2 on 78 occasions. Panel (A) of Figure 3.9 allows for an assessment of the probabilities with which these bursts were classified. In many cases, the probability that the burst belonged to class 1 (green) was very close to the probability for class 2 (black), indicating the model uncertainty. Additionally, it can be observed that this uncertainty is primarily associated with specific locations, such as when the burst is related to node ID = 20 (highlighted in the zoomed chart). As shown in Figure 3.10, node ID = 20 is located in cluster 1 (green) and is distant from the placed sensors.

Moreover, since the graph in Figure 3.9 respects the temporal evolution of the bursts, it allows for the evaluation of time-based correlations between the model errors. Specifically, for the burst at node ID = 20, there is a recurring pattern where the MLP model struggles to accurately predict the correct cluster during the early morning hours. Fig. 10 further clarifies the issue previously discussed regarding the MLP model difficulty in distinguishing bursts in cluster 1 (green) from those in cluster 2 (black). Notably, no sensors were selected between these two clusters for the final model output, which likely contributes to this misclassification. Furthermore, Figure 3.10 provides insights into the physical principles underlying the sensor selection process. Notably, the selected sensors primarily perform flow-balancing between clusters, ensuring effective fault detection within the clusters while maintaining a reasonable trade-off between cost and performance.

It is essential to emphasize that the selection of only flow meters, as determined by the proposed methodology, aligns well with the findings of Wu et al. [125], who highlighted the importance of emphasizing flow data in burst detection within WDN districts. According to Wu et al., flow data are more sensitive to bursts compared to pressure data. This observation is further supported by Mounce et al. [126], who pointed out that pressure is a less reliable indicator for detecting abnormal events, largely due to the fact that the response of a pressure meter is highly dependent on its location. Similarly, Ye and Fenner [127] employed Kalman filtering of hydraulic measurements and concluded that flow measurement data are more responsive to bursts or leaks than pressure measurement data.

Consequently, although pressure sensors are typically less expensive, they require a significantly greater number of measurement stations than flow meters to achieve reliable anomaly detection. This finding further validates the outcomes of the proposed approach, which recommends the exclusive use of flow sensors for the case study under consideration. Additionally, flow meters are often already installed within a WDN, such as downstream of reservoirs, at pump stations, and at district boundaries, where they serve

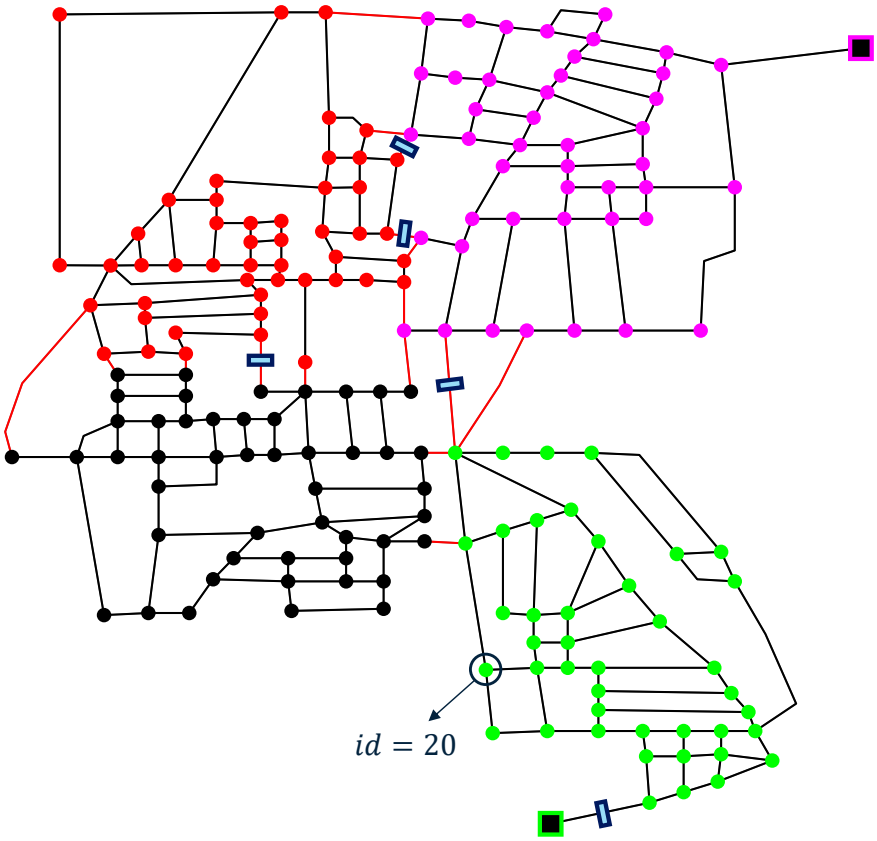

Figure 3.10: WDN layout of Parete and flow meters positioning (rectangular symbols) obtained as result of the proposed methodology.

the regular purpose of monitoring operating conditions. Leveraging data from these pre-existing devices to enhance burst management represents an efficient and cost-effective monitoring strategy. In the specific case study, one of the flow meters is positioned downstream of one of the two reservoirs, indicating that it is likely already present in the network. As a result, its cost can be excluded from the total new investment, leading to an additional cost savings of approximately 20%.

Moreover, it is worth noting that the selection of the number of clusters can significantly influence the results. An increase in the number of clusters leads to a higher number of boundary pipes and, consequently, more potential locations for flow meters, thereby raising the total investment cost as well as the computational workload. However, a greater number of clusters also results in smaller detection areas, which enhances detection accuracy during the modeling phase and simplifies the subsequent burst localization process for utility staff during the managerial phase. This highlights the inherent trade-off between minimizing total costs (both investment and monitoring) and maximizing the efficiency of modeling and management.

Finally, in the context of closed-loop systems for WDN pressure/flow control (see, for example, [128, 129]), it may be necessary to incorporate the current actuator settings into the dataset used to train the machine learning classification model. This consideration is important because closed-loop control systems can reduce the informative value of pressure and flow data, potentially impacting the effectiveness of the burst detection and localization process.

Chapter 4

Uncertainty Quantification in Bayesian Neural State-Space Models for Constrained Model Predictive Control

In this chapter, the potential of ANNs to model and control complex nonlinear dynamical systems is explored. The primary focus is on evaluating the statistical uncertainty associated with the model predictions, which is crucial for ensuring that safety constraints are consistently met in model-based control algorithms. The chapter begins with a detailed discussion about uncertainty classification and estimation for ANNs. Then, its possible implementation in a Model Predictive Control (MPC) scheme is presented. Following the theoretical exploration, the approach is tested through in-silico simulations, on the ship maneuvering process, to assess its practical reliability.

4.1 Introduction to Uncertainty

In recent years, ANNs have permeated every scientific field, becoming a crucial component of numerous real-world applications. As their usage has increased, the importance of assessing confidence in their predictions has significantly grown. However, conventional ANNs function as deterministic models and usually lack the capability to quantify or express this prediction confidence. In the last decades, to overcome this issue, ANN uncertainty and its possible sources have been deeply investigated and various approaches to measure and quantify uncertainty have been proposed [35, 130]. To better

understand and address these uncertainties, it is essential to categorize them based on their origin and nature.

In the field of mathematical modelling and AI [131, 132], uncertainty is typically classified into two main categories:

- **Epistemic Uncertainty**, which arises from the limitations in the model knowledge or in the approximation error due to lack of knowledge. This type of uncertainty is inherent in the model itself or in the training procedure. Epistemic uncertainty is theoretically reducible and can be minimized by improving the model, training procedure, or dataset size and diversity.

- **Aleatoric Uncertainty**, which refers to the uncertainty inherent in the data itself. This type of uncertainty is caused by the inherent variability of the data-generating process and is irreducible even with additional data. The uncertainty related to errors and noise in measurement systems is a prime example of aleatoric uncertainty. This uncertainty arises from the limitations and imperfections in the data collection process, such as sensor noise, low resolution, or erroneous labeling. Unlike epistemic uncertainty, aleatoric uncertainty cannot be reduced by improving the model or collecting more data, as it is a fundamental characteristic of the data.

Understanding the distinction between these two types of uncertainty is crucial for effectively managing and mitigating the risks associated with neural network predictions. While epistemic uncertainty can be addressed by enhancing the model exposure to diverse and comprehensive data, aleatoric uncertainty requires robust methods to handle the inherent noise and imperfections in the data. By carefully considering both types of uncertainty, it is possible to develop more reliable and resilient neural network models that can better handle the complexities and challenges of real-world applications. Having established the distinction between epistemic and aleatoric uncertainty, it is also essential to consider which are the sources of these uncertainties.

4.1.1 Uncertainty Sources

According to [32], the process from raw environmental data to neural network prediction encompass multiple potential sources of uncertainty and error. These uncertainties can significantly impact the final outcomes generated by the network. Among the various factors contributing to uncertainty in deep neural networks (ANNs), five stand out as particularly crucial:

- **Variability in Real-World Situations**: Real-world environments are inherently dynamic and subject to continuous fluctuations. Numerous parameters, such as temperature, lighting conditions, and the size and shape of physical objects are subject to variations. When the conditions in the real world deviate from those represented in the training data, this phenomenon is known as a *distribution shift*. Neural networks are particularly sensitive to such distribution shifts, which can lead to considerable alterations in the model performance. This sensitivity arises because the neural network ability to generalize is closely tied to the distribution of the data it has been trained on. As a result, when faced with new or altered conditions that differ from the training set, the network predictions may become less accurate or reliable.

- **Errors Inherent to Measurement Systems**: Measurement systems themselves can introduce significant uncertainty into the predictions made by a neural network. This uncertainty can stem from various factors, including limitations in the measurement data, such as low image resolution or inadequate sampling rates, which provide only partial or imprecise information. Additionally, noise introduced by sensors, whether due to environmental factors like motion or mechanical stress, can lead to inaccurate measurements. Another critical source of uncertainty, related to classification problems, is erroneous labeling, commonly referred to as label noise. This occurs when the data is incorrectly labeled, which can mislead the network during training by reducing its confidence in correctly identifying the true class. Label noise is particularly problematic because it can degrade the network performance, leading to less accurate predictions. Interestingly, in some cases, this type of noise and error can be strategically leveraged during training to enhance the model robustness and generalization capabilities. By intentionally introducing noise, the training process can be regularized, helping the neural network to better handle variability and uncertainty in real-world data (see [7, 133]).

- **ANN Architectural Specification**: The architecture of a neural network plays a crucial role in determining its overall performance and, consequently, the level of uncertainty associated with its predictions. The design choices made during the construction of the network, such as the number of layers, nodes, and parameters, directly influence the network ability to learn and generalize from the training data. For example, an overly complex network with a large number of parameters might have a high memorization capacity, which can lead to overfitting,

where the model performs exceptionally well on training data but fails to generalize to unseen data. Conversely, a network with too few parameters may underfit, struggling to capture the underlying patterns in the data. In the context of uncertainty, the structure of the neural network is particularly significant. Research has shown that deeper networks, which are more complex, often exhibit overconfidence in their predictions [134] . The challenge lies in striking the right balance in the network architecture to minimize these errors. Properly calibrating the network and adjusting its complexity are essential steps in managing the uncertainty that arises from the model structure itself.

- **Variability in the Training Procedure**: The training process of a neural network involves the careful selection and definition of numerous parameters, each of which can significantly influence the model performance and the resulting uncertainty in its predictions. Key parameters include the batch size, the choice of optimizer, the learning rate, stopping criteria, and regularization techniques, among others. Additionally, the training process involves stochastic elements such as the generation of batches and the initialization of weights, both of which introduce randomness into the training dynamics. These various decisions and stochastic elements contribute to the overall complexity of the training process. This inherent variability in the training process can lead to different local optima, where the model converges to different sets of parameters, potentially impacting the consistency and reliability of the network predictions. Moreover, a particularly critical aspect of the training procedure is the generation of the training dataset. Indeed, if the training dataset is imbalanced or fails to adequately cover certain regions of the data distribution, this can introduce additional uncertainties into the training procedure and, consequently, into the final trained model. For instance, underrepresented classes or regions in the data may not be sufficiently learned, leading to biased or less reliable predictions. To mitigate these issues, techniques such as data augmentation [135–138] can be employed to artificially increase the diversity of the training data. Additionally, methods to balance the impact of individual classes or regions on the loss function can be applied, helping to ensure that the network learns more evenly across the entire data distribution. By carefully managing these aspects of the training process, the uncertainties inherent in the model can be reduced, leading to more robust and reliable neural network predictions.

- **Errors Caused by Unknown Data**: A neural network trained on

data from a specific domain may encounter challenges when presented with samples from an entirely different domain. For instance, consider a neural network that has been trained exclusively on predicting the temperature of a house. If this network is then used to predict temperature in a mall, it may struggle to accurately process the new input data. The uncertainty in this scenario does not derive from errors in data acquisition; rather, it arises because the network was never trained to recognize and elaborate samples from this different domain. In this context, even though the input data is technically valid, it falls outside the scope of the network learned knowledge. As a result, the network might produce unreliable or unpredictable outputs because it lacks the necessary information to handle this new type of data. This type of uncertainty, often referred to as out-of-domain uncertainty, highlights the importance of ensuring that a neural network is either robust enough to manage such unexpected inputs or is equipped with mechanisms to identify when an input falls outside its expected domain. Without such capabilities, the network predictions can become significantly less reliable when confronted with data that diverges from the training distribution.

It is worth to highlight that, based on the first differentiation discussed above, the uncertainties associated with variability in real-world situations, errors in the architectural specification of the ANN, variability on the training procedure, and errors caused by unknown data, all contribute to epistemic uncertainty, while only the uncertainty related to measurement system can be classified as aleatoric uncertainty. Overall, all these sources of uncertainty are inherent to the functioning of neural networks and can have varying impacts depending on the specific context in which the network is applied. Addressing these uncertainties is critical, as they directly influence the reliability and accuracy of the model predictions. Each of the identified factors introduces its own unique challenges, requiring different strategies to mitigate their effects. For instance, while epistemic uncertainties can be reduced through better data collection or model refinement, aleatoric uncertainty is an unavoidable consequence of the data itself.

4.1.2 Predictive Uncertainty Classification in ANNs

Understanding the distinction between aleatoric and epistemic uncertainty is essential for effectively managing predictive uncertainty (the uncertainty associated with the model predictions). Indeed, predictive uncertainty derives from both the model uncertainty due to its limited knowledge (epistemic)

and the variability inherent in the data (aleatoric). By accurately identifying and addressing these two types of uncertainty, it is possible to improve the overall confidence in the model predictions. Predictive uncertainty in ANNs is classified depending on the nature of the input data provided to the model at test time into three main categories [32]:

- **In-Domain Uncertainty**:

  In-domain uncertainty refers to the uncertainty associated with test input data that originates from the same distribution as the training data. Even though the input aligns with the expected data domain, the neural network may still exhibit uncertainty due to its inability to fully capture the complexities within this distribution. This can arise from limitations in the model architecture, training process, or simply from the inherent difficulty of the problem being solved. In-domain uncertainty sources are both epistemic (i.e., model uncertainty) and aleatoric (i.e., data uncertainty). Improving the quality of the training data or refining the training process can help reduce this type of uncertainty. For instance, enhancing data quality or increasing the dataset coverage can allow the neural network to better learn from diverse examples, thereby reducing the ambiguity in its predictions [131].

- **Domain-Shift Uncertainty**:

  Domain-shift uncertainty arises when the test input data, while still somewhat related to the training data, exhibits differences due to shifts in distribution. These shifts can result from variations in real-world situations, where the data encountered during inference differs from the data the network was trained on. For example, consider a neural network trained to model the dynamic behavior of an industrial process using data collected under specific temperature and humidity conditions. During inference, if the industrial process operates in different environmental conditions, such as extreme temperatures or higher humidity levels, its operational characteristics may change, even if the process remains the same. This misalignment between the training and inference environments introduces uncertainty because the network has not been exposed to these specific variations during training. Some causes of domain-shift uncertainty can be modeled and mitigated by augmenting the training data with such variations. However, not all domain shifts can be fully addressed. For example, random motion noise or other unpredictable environmental factors can still introduce uncertainty. From the modeler's perspective, domain-shift uncertainty

is largely influenced by external factors that can, in part, be reduced by increasing the training data coverage of the shifted domain [139].

- **Out-of-Domain Uncertainty**:

  Out-of-domain uncertainty refers to the uncertainty that arises when the network encounters test input data that is completely outside the distribution it was trained on. In this case, the input belongs to an entirely different domain, rendering the model learned knowledge inadequate for processing the new data. Out-of-domain uncertainty is particularly challenging because the model cannot leverage any of its prior learning to make reliable predictions. This type of uncertainty underscores the importance of equipping neural networks with mechanisms to detect out-of-domain data, ensuring that erroneous or overconfident predictions are avoided. The network inability to generalize to these unknown inputs stems from a fundamental lack of knowledge about this new domain, and unless the training data is expanded to include relevant samples, this uncertainty cannot be reduced [140, 141].

Understanding and effectively managing the various categories of uncertainty in ANNs is essential for enhancing the reliability and accuracy of their predictions across a wide range of applications. By distinguishing between epistemic uncertainty, which arises from the model limited knowledge, and aleatoric uncertainty, which is inherent in the data, we can develop more precise strategies to address these challenges. Further classification of predictive uncertainty based on the nature of the test input data (whether it falls within the same distribution as the training data, has undergone a distribution shift, or is entirely out-of-domain) enables us to tailor our approaches more effectively. In this context, it is worth to highlight that domain-shift and out-of-domain uncertainty are more related to epistemic uncertainty, while in-domain uncertainty can be decomposed in both epistemic and aleatoric uncertainty. In-domain uncertainty, which is driven by the complexity of the data and model design, can be mitigated through refined training processes and improved data quality. Domain-shift uncertainty, often caused by variations in real-world conditions, may be addressed through data augmentation and more comprehensive training datasets to ensure the model can adapt to these changes. Out-of-domain uncertainty, on the other hand, underscores the importance of equipping models with mechanisms to detect when they encounter unfamiliar data, thereby preventing erroneous or overly confident predictions. Each type of uncertainty presents unique challenges during both the development and deployment phases of neural networks. By carefully considering and addressing these challenges, the robustness of

neural network models can be significantly improved, ensuring to perform reliably even in diverse real-world scenarios.

4.1.3 Predictive Uncertainty Modelling

Having classified the potential sources of uncertainty in the development of ANNs, it is crucial to understand how to effectively model and manage this uncertainty. To this aim, various methodologies have emerged within the field of uncertainty estimation [32], each offering different approaches to quantify the uncertainty associated with model predictions. These methods can generally be categorized based on whether they employ single or multiple neural networks, and whether those networks are deterministic or stochastic in nature. Below are some of the most widely studied approaches:

- **Single Deterministic Methods**: These methods typically involve a single forward pass through a deterministic network, where uncertainty quantification is derived directly from the network output. For instance, in classification tasks, the softmax probabilities are often interpreted as a measure of confidence or uncertainty in the prediction. However, this approach can be limited as it does not explicitly account for model uncertainty [142–144].

- **Ensemble Methods**: Ensemble methods enhance uncertainty estimation by combining the predictions of multiple deterministic networks. By leveraging the diversity of these models, ensemble methods aim to produce a more reliable estimate of uncertainty. The variance observed among the predictions of the individual models in the ensemble serves as an indicator of the uncertainty associated with the prediction. This approach is particularly useful because it captures both epistemic and aleatoric uncertainties to some extent, though it can be computationally expensive due to the need to train and maintain multiple models [145–148].

- **Test-Time Augmentation Techniques**: Test-time augmentation involves applying a variety of transformations to the input data during inference to generate multiple predictions. These predictions are then analyzed for consistency, which provides an estimate of uncertainty. Like ensemble methods, it can be computationally demanding and may not fully separate the different types of uncertainty [149–151].

- **Bayesian Methods**: Bayesian Neural Networks (BNNs) offer a principled approach to uncertainty estimation by directly modeling the uncertainty in the network parameters. BNNs allow for the estimation

of epistemic uncertainty by integrating over the posterior distribution of the network weights, providing a more detailed and comprehensive framework for uncertainty estimation [152–154].

All these approaches offers valuable insights of the model predictive uncertainty and present their own trade-offs in terms of complexity, computational efficiency and the quality of uncertainty estimation. While the first three methods are more efficient and easy to implement, the structure of the estimated uncertainty is not clearly defined. Instead, BNNs provide a principled and theoretically grounded framework for uncertainty estimation, making them particularly suitable for applications where defining the model confidence is crucial, although they require approximations which can be hard to implement.

4.1.4 Bayesian Neural Networks

BNNs represent a powerful approach at the intersection of neural networks and Bayesian inference, combining the expressive power and scalability of traditional neural networks with the probabilistic foundations of Bayesian theory. Unlike conventional neural networks that typically rely on Maximum Likelihood estimation, BNNs infer a posterior probability distribution over the network parameters, denoted as $\theta = (\theta_1, \ldots, \theta_n)$, rather than learning fixed weight values. This approach allows BNNs to quantify the uncertainty in the model parameters and, as a consequence, to propagate this epistemic uncertainty to the predictions, which is crucial for tasks that require a measure of confidence in the predictions.

In a BNN, given a dataset of training data $\mathbf{D} = (u, y)$, with u and y are respectively the input and output sequence, the posterior distribution over the network parameters $p(\theta \mid D)$ is derived by applying Bayes' theorem. This involves specifying a prior distribution $p(\theta)$ over the parameters and updating it with the likelihood of the observed data $p(D \mid \theta)$ to obtain the posterior distribution:

$$p(\theta \mid D) = \frac{p(D \mid \theta) \cdot p(\theta)}{p(D)} \tag{4.1}$$

The normalization constant $p(D)$, also known as the *model evidence* or *marginal likelihood*, is defined as:

$$p(D) = \int p(D \mid \theta) \cdot p(\theta)\, d\theta$$

The model evidence is obtained by marginalizing $p(D|\theta)$ over the set of parameters θ ensuring that the posterior distribution $p(\theta|D)$ is properly normalized and its integral is unitary. Since the model evidence is a dataset-dependent normalization factor that uniformly scales the posterior distribution density function, it can often be approximated or even omitted without affecting the inference conclusions. Indeed, while the model evidence ensures that the posterior distribution integrates to one, it does not alter the distribution relative structure, such as the distribution peaks, or which parameter values are more likely than others. In other words, it resizes the distribution uniformly, leaving the shape of how probability mass is spread across parameters intact.

Once the posterior distribution over the parameters has been estimated, predictions for a new input sequence u^* can be obtained by integrating the predictive distribution $p(y^* \mid u^*, \theta)$ over the posterior distribution of the parameters obtaining the *posterior predictive distribution*:

$$p(y^* \mid u^*, D) = \int p(y^* \mid u^*, \theta) \cdot p(\theta \mid D)\, d\theta$$

This Bayesian approach allows for the computation of principled predictive uncertainty. However, computing both the posterior distribution $p(\theta \mid D)$ and the posterior predictive distribution $p(y^* \mid u^*, D)$ involves solving integrals over high-dimensional parameter spaces, which are generally intractable for models with a large number of parameters like ANNs.

In simpler Bayesian models, closed-form solutions are often achievable through the use of conjugate priors which are prior distributions that, when combined with the likelihood function, result in a posterior distribution of the same family as the prior [155]. This mathematical convenience allows for closed-form solutions of the posterior inference. However, for neural networks, suitable conjugate priors typically do not exist due to their complex, non-linear nature. As a result, the posterior distribution $p(\theta \mid D)$ cannot be expressed in a closed-form expression.

Similarly, computing the posterior predictive distribution $p(y^* \mid u^*, D)$ requires integrating over the posterior distribution of the parameters, which is itself intractable. This makes direct computation of predictive uncertainties infeasible. Therefore, approximate Bayesian inference techniques are required to address these challenges in both posterior inference and prediction. Among these techniques, three main approaches have emerged as the most widely used:

- **Variational Inference** [156, 157]: This approach approximates the intractable posterior distribution by optimizing over a family of tractable

distributions.

- **Sampling Approaches** [158–160]: These methods represent the target random variable by drawing samples from it, typically using techniques like Markov Chain Monte Carlo (MCMC).
- **Laplace Approximation** [35, 130]: This method approximates the log-posterior distribution and derives a normal distribution over the network weights based on this approximation.

These approaches differ in several important aspects. Sampling methods, such as MCMC, are designed to explore the entire posterior distribution, including multiple modes that may arise due to complex data structures or model ambiguities. By generating samples that capture all high-probability regions in the parameter space, sampling methods provide a comprehensive representation of uncertainty. This is particularly valuable when the posterior distribution is multimodal, as it ensures that all plausible parameter configurations are considered in inference and prediction. However, this thorough exploration comes at a significant computational cost. Sampling methods are inherently stochastic and require a large number of iterations to achieve convergence, especially in high-dimensional spaces, making them computationally expensive during training.

In contrast, deterministic approximations like variational inference and the Laplace approximation offer computational efficiency by simplifying the posterior distribution. Variational inference approximates the posterior within a predefined family of distributions, often leading to unimodal representations that may overlook multiple modes. The Laplace approximation further simplifies the posterior by assuming it is Gaussian, centered around the Maximum A Posteriori (MAP) estimate, and focuses on the local curvature of the posterior. While this assumption of unimodality limits its ability to capture multiple modes, the Laplace approximation stands out for its efficiency, particularly in its ability to be applied to pre-trained networks with minimal additional computational cost.

Although the Laplace approximation may not capture all the complexities of a multimodal posterior, it provides a balance between computational efficiency and a rigorous approach to uncertainty estimation. Given the computational demands and resource constraints of our application, the Laplace approximation has been selected as the preferred method for inferring the posterior distribution in this work. This approach offers a streamlined and effective means of approximating the posterior, making it particularly well-suited for applications where balancing accuracy with computational demands is crucial.

4.1.5 Laplace Approximation

The Laplace approximation is a technique used to approximate the posterior distribution of a model $p(\theta|D)$. The core idea is to estimate the posterior distribution around a local mode and model it as a multivariate normal distribution. To achieve this, the Laplace approximation relies on a second-order Taylor expansion of the log-posterior around the MAP estimate, θ_{MAP}, given a training set of data points D. Indeed, the Taylor expansion provides information about the curvature (i.e., the variance) of the posterior in θ_{MAP}.

The MAP estimate θ_{MAP} is defined as the parameter value that minimizes the negative log-posterior distribution:

$$\theta_{\text{MAP}} = \arg\min_{\theta} L(\theta), \tag{4.2}$$

where $L(\theta) = -\log p(\theta \mid D)$ is the negative logarithm of the posterior distribution. This transformation simplifies the manipulation of products into sums and makes working with exponential terms easier. Indeed, It is possible to decompose the posterior $p(\theta \mid D)$ using Bayes' theorem, and taking the logarithm and rearranging Eq. 4.1:

$$L(\theta) = -\log p(\theta \mid D) = -\log p(D \mid \theta) - \log p(\theta) + C, \tag{4.3}$$

where C is a constant term that incorporates the model evidence $p(D)$ which does not influence the MAP estimation. Therefore, the Taylor expansion of the log-posterior $\log p(\theta \mid D)$ around θ_{MAP} is given by:

$$\begin{aligned} \log p(\theta \mid D) \approx \log p(\theta_{\text{MAP}}) + (\theta - \theta_{\text{MAP}})^\top \nabla \log p(\theta \mid D)\Big|_{\theta=\theta_{\text{MAP}}} \\ + \frac{1}{2}(\theta - \theta_{\text{MAP}})^\top H (\theta - \theta_{\text{MAP}}), \end{aligned} \tag{4.4}$$

where H is the Hessian matrix (the matrix of second-order partial derivatives) of the log-posterior, evaluated at θ_{MAP}. The first-order term vanishes because the gradient $\nabla \log p(\theta \mid D)$ is zero at θ_{MAP}, as θ_{MAP} is a local maximum of the log-posterior. This simplifies the expression to:

$$\log p(\theta \mid D) \approx \log p(\theta_{\text{MAP}}) - \frac{1}{2}(\theta - \theta_{\text{MAP}})^\top H (\theta - \theta_{\text{MAP}}). \tag{4.5}$$

Taking the exponential of both sides, the posterior distribution is approximated as:

$$p(\theta \mid D) \approx \exp\left(\log p(\theta_{\text{MAP}}) - \frac{1}{2}(\theta - \theta_{\text{MAP}})^\top H (\theta - \theta_{\text{MAP}})\right). \tag{4.6}$$

Since $\log p(\theta_{\text{MAP}})$ is constant, the expression simplifies further, and the form of a multivariate normal distribution is recognized. Thus, the posterior distribution can be defined as:

$$p(\theta \mid D) \approx \mathcal{N}(\theta_{\text{MAP}}, H^{-1}), \tag{4.7}$$

where H^{-1} is the covariance matrix. Here, a regularization term, equivalent to the precision of the prior distribution $P(\theta)$ is often added to the Hessian to ensures numerical stability by accounting for possible ill-conditioning in the Hessian matrix. Overall, the Laplace approximation simplifies the problem by assuming a Gaussian form for the posterior distribution centered at the MAP estimate. The uncertainty in the model parameters is primarily represented by the Hessian matrix H, which captures the curvature of the log-posterior around the MAP estimate. This approach balances computational efficiency with a reasonable approximation of uncertainty, making it suitable for applications where computational cost and uncertainty estimation are both important.

4.2 Introduction to Model Predictive Control

MPC refers to a family of control strategies designed to regulate the behavior of dynamic system by utilizing predictions from a system model. Its conceptual simplicity and ability to manage complex system dynamics, including multiple inputs and outputs, state and output constraints, and competing control objectives, have made MPC an attractive choice for multivariable constrained control applications. One of the key factors behind the success of MPC algorithms is their intuitive approach to solve control problems. The main components in the design of a predictive controller are:

- The system model for predicting system future behaviour.
- A loss function that reflects the tracking error relative to the reference signal and the control effort.
- An optimization algorithm to compute a sequence of future control actions that minimizes the performance index, subject to given constraints.
- The receding horizon strategy, in which only the first control action from the optimal sequence is applied in real-time.

Indeed, MPC operates by repeatedly solving an open-loop constrained Optimal Control Problem (OCP) over a finite time horizon N_H, based on the system current state x at each sampling instant k, in a receding-horizon fashion [161–163]. In general, a distinction is made based on the model nature adopted in the control schema. Linear and Nonlinear MPC are differentiated based on whether the system model used in the control scheme is linear or nonlinear, respectively. Linear MPC approaches have found successful applications (see [164–167]) and due to its tractability a wide range of properties and stability guarantee have been derived [163, 168, 169]. Instead, for Nonlinear Model Predictive Control (NMPC) deriving general stability properties is more challenging due to the nonlinear model inherent to the control schema, although in the last decades several works have been carried out [170–175] and various applications have demonstrated its effectiveness [176, 177].

When uncertainties affecting the system or inherent to the model are taken into account in the OCP, a distinction is made based on the type of uncertainty considered. If the uncertainty is deterministic, Robust Model Predictive Control (RMPC) schemes are used, ensuring that the system can handle the worst-case scenario within the given uncertainty bounds. RMPC approaches generally rely on deterministic and bounded representations of uncertainties. On the other hand, if the uncertainty follows a probability distribution, Stochastic Model Predictive control (SMPC) is the preferred approach. SMPC leverages the probabilistic nature of the uncertainty to optimize performance while maintaining a desired level of constraint satisfaction with high probability. Significant advancements have been made in the field of RMPC, with a focus on developing computationally efficient optimal control methods capable of systematically addressing system uncertainties [178]. RMPC typically deals with set-membership uncertainties, where uncertainties are assumed confined within a bounded set. Initial RMPC research largely revolved around min-max OCP formulations, which design control actions based on worst-case evaluations of the cost function, ensuring that the OCP constraints were satisfied for all possible uncertainty realizations [179, 180], although min-max approaches frequently led to conservative control actions, largely due to challenges in managing the range of state trajectories [181]. However, in practical applications, uncertainties are often better represented with a probabilistic approach. When system uncertainties can be described stochastically, it is more effective to account for their probabilistic behavior in the control design. This has led to the development of SMPC, which systematically incorporates probabilistic uncertainty descriptions into a stochastic OCP. SMPC leverages probabilistic uncertainty models to define chance constraints, which require that state/output

constraints are satisfied with a specified probability level [182, 183], or alternatively, be satisfied in expectation [184]. Chance constraints allow for the use of stochastic uncertainty characterizations to manage an acceptable level of constraint violation. In this way, SMPC enables a systematic trade-off between closed-loop performance and constraints satisfaction, which is crucial for handling uncertain systems when high-performance operation occurs near the constraint boundaries [38, 185]. In recent years, SMPC has garnered increasing interest both theoretically [186–188] and in practical applications [182, 189–191].

Over the past decades, there has been a growing interest in data-driven MPC. Traditional model-based control approaches rely on accurate mathematical models of system dynamics, which can be difficult and inefficient to develop for complex or nonlinear systems. Data-driven methods offer an alternative by utilizing system data directly to design controllers without explicit system identification [16, 192–196]. However, a critical challenge in data-driven MPC is ensuring robustness to uncertainties and guaranteeing system stability. Since data-driven models may not capture all system behaviors accurately, especially in the presence of epistemic uncertainties (uncertainties due to lack of knowledge), it is essential to develop control schemes that can handle such uncertainties effectively. Several studies have focused on establishing stability and robustness in data-driven MPC frameworks [197, 198]. In this context, incorporating a probabilistic characterization of epistemic uncertainty becomes crucial for modeling nonlinear systems within data-driven MPC. By representing uncertainties probabilistically, it is possible to design controllers that not only optimize performance but also maintain a desired level of confidence in constraint satisfaction and stability. This work aims to address the integration of probabilistic methods to handle epistemic uncertainties in data-driven MPC, enhancing the robustness and reliability of the control schemes.

4.3 Problem Formulation

This section outlines the complete methodology. First, the formulation of the nonlinear system employed in this work is introduced (Sec. 4.3.1), along with a discussion on the implementation of the ANN-based model (Sec. 4.3.2) employed to approximate the unknown system dynamics when only partial information is available. Then, the Bayesian inference process is detailed (Sec. 4.3.3), outlining the steps taken to estimate model parameters, their uncertainty, and the posterior predictive distribution used in the stochastic control problem (Sec. 4.3.7). Finally, the possible usage of the posterior

predictive distribution in a SMPC control schema is discussed.

4.3.1 Nonlinear Dynamical Model with Epistemic Uncertainty

In many real-world applications, the behavior of dynamic systems is influenced by stochastic components in their states that are usually impossible to model in a deterministic way. These fluctuations can arise from environmental changes, or complex dynamics as discussed in 4.1. A generic nonlinear dynamical system with a stochastic dynamics can be modelled in continuous time as:

$$\begin{cases} \dot{x}(t) = F(x(t), u(t), \rho) + e(t) \\ y(t) = G(x(t)) \end{cases} \tag{4.8}$$

where $x(t) \in \mathbb{R}^{n_x}$ represents the states vector of the system at time t, $u(t) \in \mathbb{R}^{n_u}$ is the input vector (or control signal) at time t, and $\rho \in \mathbb{R}^p$ denotes the model parameters that have to be properly inferred from observations. The term $e(t) \in \mathbb{R}^{n_x}$ is a stochastic disturbance or uncertainty that affects the states evolution. In particular, in this work, $e(t)$ is assumed to be a continuous-time multivariate White Gaussian Noise (WGN) process $\{e(t) : -\infty < t < \infty\}$, characterized by zero mean and an autocovariance matrix $R_e(\tau) = \mathbb{E}[e(t)e(t+\tau)^\top] = \Sigma\delta(\tau)$, where $\Sigma = \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_{n_x}^2) \in \mathbb{R}^{n_x \times n_x}$ is a diagonal covariance matrix, and $\delta(\tau)$ is the Dirac delta function. The function $F : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^p \to \mathbb{R}^{n_x}$ describes the nonlinear states evolution of the system, which maps the current state $x(t)$, the input $u(t)$, and the parameters ρ to the rate of change of the state $\dot{x}(t)$. This function reflects the internal dynamics of the system. The output $y(t)$ is determined by the function $G : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$, which defines the system output as a function of the state $x(t)$. This function maps the state vector to the output vector $y(t)$, where $y(t) \in \mathbb{R}^{n_y}$ represents the observable quantities of the system. This formulation is typical in systems where disturbances, modeled by $e(t)$, play a significant role in affecting the system evolution, making it suitable for modeling complex dynamical systems. The discrete representation of the model representing the nonlinear system in continuous time can be approximated by numerically integrating Eq. 4.8:

$$\begin{cases} x(k+1) = x(k) + \Delta t \cdot (F(x(k), u(k), \rho)) + e(k) \\ y(k) = G(x(k)) \end{cases} \tag{4.9}$$

with Δt the time between two time instants and $e(k)$ is discrete WGN with precision β, $e(k) \sim \mathcal{N}(0, \frac{\Delta t}{\beta})$. Here, the variance of the discrete-time noise is

scaled by the time step Δt, as it represents the integrated effect of continuous-time noise over the time interval Δt.

4.3.2 Neural State-Space Model

The aim of this work is to model nonlinear systems for which a direct formulation, as represented by Eq. 4.9 is not completely available. In particular, the main focus is on those systems in which the function G is known, while the static map F is unknown. A practical approach to address this challenge is to employ ANNs in order to approximate the system dynamics. Indeed, when direct measurements of the real system output or states are accessible, ANNs can be employed to approximate the static maps F from Eq. 4.9 [36, 199–201] and the resulting Neural State-Space (NSS) model can be expressed as:

$$\begin{cases} x_{nn}(k+1) = x_{nn}(k) + \Delta t \cdot (F_{nn}(x_{nn}(k), u(k), \theta)) \\ \hat{y}(k) = G(x_{nn}(k)) \end{cases} \tag{4.10}$$

where θ are the neural network parameters and their probability distribution $P(\theta) \sim \mathcal{N}(0, \frac{1}{\gamma})$, representing prior knowledge, is assumed to be a Gaussian distribution. The function F_{nn}, which approximate the original system dynamics F, is learned by the ANN during the training process. By initializing the NSS model with a known initial condition $x_{nn}(0) = x(0)$ (obtained from measurements) and assuming to know the future input sequence $U_N = [u(0), ..., u(N)]$, the NSS model described in (4.10) allows for the simulation of the system evolution over N discrete time steps. This ability to accurately predict future states based on the current state and input sequence makes the NSS model of Eq. 4.10 particularly well-suited for use in model-based control algorithms, including MPC algorithms. The NSS representation provides a flexible framework that can be directly integrated into control strategies requiring predictive models to optimize system performance over a given horizon.

4.3.3 Bayesian Inference for Neural State-Space Models

To account for the inherent epistemic uncertainty in model 4.9, we employ a Bayesian approach to estimate the posterior distribution of the neural network parameters as discussed in Sections 4.1.4-4.1.5. Given a dataset D, and a NSS model with n_θ parameters, the posterior distribution over the

parameters θ of the ANN can be formulated using Bayes' theorem as:

$$P(\theta|D) = \frac{P(D|\theta) \cdot P(\theta)}{P(D)}, \tag{4.11}$$

where $P(\theta)$ is the prior distribution on the parameters θ, which is assumed to be Gaussian with precision γ:

$$P(\theta_1, \theta_2, \ldots, \theta_{n_\theta}) = \left(\frac{\gamma}{2\pi}\right)^{\frac{n_\theta}{2}} \exp\left(-\frac{\gamma}{2}\sum_{i=1}^{n_\theta}\theta_i^2\right). \tag{4.12}$$

The likelihood function $P(D|\theta)$ is based on the assumption that the noise affecting the state evolution is Gaussian with precision β. For a single state x and its corresponding dataset $D = \{x_1, x_2, \ldots, x_N\}$, with N measurements, we can express the likelihood by comparing the observed state increments to the ones predicted by the neural network F_{nn}, represented as:

$$\begin{aligned} P(D|\theta, \beta) &= \left(\frac{\beta}{2\pi}\right)^{\frac{N}{2}} \exp\left(-\frac{\beta}{2}\sum_{k=1}^{N}\delta_k^2\right), \\ \text{with } \delta_k &= F(x(k), u(k), \rho) + e(k) - F_{nn}(x(k), u(k), \theta) \\ &= \frac{x(k+1) - x(k)}{\Delta t} - F_{nn}(x(k), u(k), \theta), \end{aligned} \tag{4.13}$$

where δ_k represents the discrepancy between the observed and predicted state derivatives. The term $F_{nn}(x(k), u(k), \theta)$ denotes the predicted state derivative from the neural network, while $F(x(k), u(k), \rho) + e(k)$ represents the actual system dynamics. This formulation integrates the temporal evolution of the states into the likelihood function, making it suitable for continuous dynamical systems modeled via ANNs. For systems with n_x states, the likelihood is extended to consider the noise affecting each state independently. Considering noise precisions $\beta_1, ..., \beta_{n_x}$, the joint likelihood can be written as:

$$\begin{aligned} P(D \mid \theta, \beta_1, ..., \beta_{n_x}) &= \prod_{i=1}^{n_x} P(D \mid \theta, \beta_i) \\ &= \prod_{i=1}^{n_x} \left(\frac{\beta_i}{2\pi}\right)^{\frac{N}{2}} \exp\left(-\frac{\beta_i}{2}\sum_{k=1}^{N}\delta_i^2(k)\right) \\ &= \left(\prod_{i=1}^{n_x}\frac{\beta_i}{2\pi}\right)^{\frac{N}{2}} \exp\left(-\frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{n_x}\beta_i\delta_i^2(k)\right), \end{aligned} \tag{4.14}$$

where $\delta_i(k)$ represents the prediction error for state i at time step k.

4.3.4 Posterior Estimation via Laplace Approximation

To estimate the posterior distribution of the parameters $P(\theta|D)$, it is possible to use the Laplace approximation (Section 4.1.5). Hence, $P(\theta|D)$ is assumed to be centered at the MAP estimate θ_{MAP}, which maximizes the posterior distribution. This is equivalent to minimize the negative log-posterior $L(\theta)$, which is composed of two terms: the negative log-likelihood and the negative log-prior as discussed in Section 4.1.5. Then, substituting the expressions both for the prior (Eq. 4.12) and the likelihood (Eq. 4.12) into Eq. 4.3, we get:

$$L(\theta) = -\log\left(\left(\prod_{i=1}^{n_x}\frac{\beta_i}{2\pi}\right)^{\frac{N}{2}}\exp\left(-\frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{n_x}\beta_i\delta_i^2(k)\right)\right)$$
$$-\log\left(\left(\frac{\gamma}{2\pi}\right)^{\frac{n_\theta}{2}}\exp\left(-\frac{\gamma}{2}\sum_{i=1}^{n_\theta}\theta_i^2\right)\right)+C$$

and incorporating in C the effects the prior and likelihood constants $\left(\frac{\gamma}{2\pi}\right)^{\frac{n_\theta}{2}}$ and $\left(\frac{\beta}{2\pi}\right)^{\frac{N}{2}}$, which do not depend on θ, it is possible to derive the loss function useful for the MAP estimate:

$$L(\theta) = \underbrace{\frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{n_x}\beta_i\delta_i^2(k)}_{\text{Likelihood term}} + \underbrace{\frac{\gamma}{2}\sum_{i=1}^{n_\theta}\theta_i^2}_{\text{Prior term}} + C. \tag{4.15}$$

In this equation, the first term captures the prediction error weighted by the precision β_i, while the second term reflects the prior component providing a regularization term in the loss function (equivalent to the L2 regularization). The MAP estimate θ_{MAP} is the parameter set that minimizes $L(\theta)$, and can be found using standard optimization techniques, such as gradient descent, by computing the gradients of $L(\theta)$ with respect to the parameters θ as discussed in Section 4.1.5. The results of the optimization is θ_{MAP} that according to the Laplace approximation is an estimate of the mean of the posterior distribution of Eq. 4.7.

4.3.5 Hessian approximation

To characterize the distribution described in Eq. 4.7, the Hessian $H(\theta)$ of the log-posterior can be obtained computing the second-order partial derivative with respect to the parameters θ:

$$H(\theta) = \frac{d^2L(\theta)}{d\theta^2}. \tag{4.16}$$

The Hessian captures the curvature of the log-posterior and is crucial for estimating the uncertainty around the MAP estimate. Having

$$L(\theta) = \frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{n_x}\beta_i(x_i(k) - F_{NN_i}(k,\theta))^2 + \frac{\gamma}{2}\sum_{i=1}^{n_\theta}\theta_i^2 + C. \tag{4.17}$$

For clarity, we can split the computation into two parts: the first part focuses on the likelihood-related term, and the second part handles the prior-related term. First, considering the likelihood-related term, it is possible to write its derivative with a parameter θ_j:

$$\frac{\partial L}{\partial \theta_j} = -\sum_{k=1}^{N}\sum_{i=1}^{n_x}\beta_i(F_i(k) + e_i(k) - F_{NN_i}(k,\theta))\frac{\partial F_{NN_i}(k,\theta)}{\partial \theta_j}. \tag{4.18}$$

Next, it is possible to compute the second order derivative with respect to another generic parameter θ_l, obtaining:

$$\begin{aligned}\frac{\partial^2 L}{\partial \theta_j \partial \theta_l} = \sum_{k=1}^{N}\sum_{i=1}^{n_x}\beta_i \Bigg[&\frac{\partial F_{NN_i}(k,\theta)}{\partial \theta_j}\frac{\partial F_{NN_i}(k,\theta)}{\partial \theta_l} \\ &+(F_i(k) + e_i(k) - F_{NN_i}(k,\theta))\frac{\partial^2 F_{NN_i}(k,\theta)}{\partial \theta_j \partial \theta_l}\Bigg].\end{aligned} \tag{4.19}$$

In the Gauss-Newton approximation, the term involving the second derivatives $\frac{\partial^2 F_{NN_i}(k,\theta)}{\partial \theta_j \partial \theta_l}$ is typically ignored [202]. Thus, the Gauss-Newton approximation of the Hessian for the sum-of-squared-errors term is:

$$H_{\text{err}} \approx \sum_{k=1}^{N}\sum_{i=1}^{n_x}\Delta t^2 \beta_i \frac{\partial F_{NN_i}(k,\theta)}{\partial \theta}^\top \frac{\partial F_{NN_i}(k,\theta)}{\partial \theta}. \tag{4.20}$$

For the prior-related term, the second derivative is straightforward, yielding γI for the Hessian contribution, where I is the identity matrix. Hence, the Hessian contribution from the regularization term is:

$$H_{\text{reg}} = \gamma I. \tag{4.21}$$

Combining the contributions from both terms, the Gauss-Newton approximation of the Hessian H of the loss function is:

$$H \approx H_{\text{err}} + H_{\text{reg}} = \sum_{k=1}^{N}\sum_{i=1}^{n_x}\beta_i \frac{\partial F_{NN_i}(k,\theta)}{\partial \theta}^\top \frac{\partial F_{NN_i}(k,\theta)}{\partial \theta} + \gamma I, \tag{4.22}$$

where H_{err} captures the contributions from the likelihood and H_{reg} incorporates the regularization term from the prior. This formulation provides an estimate of the covariance matrix of the posterior distribution of the model parameters. Consequently, under the Laplace approximation, the posterior distribution can be expressed as a multivariate Gaussian distribution, with the mean corresponding to the maximum a posteriori (MAP) estimate θ_{MAP} and the covariance matrix given by the inverse of the approximated Hessian H:

$$p(\theta \mid D) \approx \mathcal{N}(\theta_{\text{MAP}}, H^{-1}). \tag{4.23}$$

4.3.6 Posterior Predictive Distribution for State Predictions

Once the posterior distribution of the neural network parameters θ has been characterized, it is important to understand how the uncertainty in θ propagates to the state predictions $x_{nn}(k)$. Since the state update equation depends nonlinearly on θ, we perform a first-order Taylor expansion of the state update equation, evaluated around the MAP estimate θ_{MAP}. In this way, it is possible to approximate the posterior predictive distribution as a Normal distribution around θ_{MAP}. Considering the state update equation of the NSS (Eq. 4.10) and expliciting its dependency from θ:

$$x_{nn}(k+1,\theta) = x_{nn}(k,\theta) + \Delta t \cdot F_{nn}\left(x_{nn}(k,\theta), u(k), \theta\right) \tag{4.24}$$

We perform a first-order Taylor expansion around θ_{MAP}:

$$\begin{aligned} x_{nn}(k+1,\theta) \approx\ & x_{nn}(k,\theta_{\text{MAP}}) + \Delta t \cdot F_{nn}\left(x_{nn}(k,\theta_{\text{MAP}}), u(k), \theta_{\text{MAP}}\right) \\ & + J_\theta(k)\left(\theta - \theta_{\text{MAP}}\right) \end{aligned} \tag{4.25}$$

where:

$$J_\theta(k) = \left.\frac{\partial\left(x_{nn}(k,\theta) + \Delta t \cdot F_{nn}\left(x_{nn}(k,\theta), u(k), \theta\right)\right)}{\partial\theta}\right|_{\theta=\theta_{\text{MAP}}} \tag{4.26}$$

we can compute the expectation w.r.t. the Laplace approximation of $P(\theta|D)$ as

$$\begin{aligned} \mathbb{E}[x_{nn}(k+1)] \approx \mathbb{E}\Big[& x_{nn}(k,\theta) + \Delta t \cdot F_{nn}\left(x_{nn}(k,\theta), u(k), \theta_{\text{MAP}}\right) \\ & + J_\theta(k)\left(\theta - \theta_{\text{MAP}}\right)\Big] \end{aligned} \tag{4.27}$$

Since $\mathbb{E}[(\theta - \theta_{\text{MAP}})] = 0$, the expectation results

$$\mathbb{E}[x_{nn}(k+1,\theta)] \approx x_{nn}(k,\theta_{MAP}) + \Delta t \cdot F_{nn}\left(x_{nn}(k,\theta_{MAP}), u(k), \theta_{\text{MAP}}\right) \tag{4.28}$$

The variance of the posterior predictive distribution can be derived as

$$Var[x_{nn}(k+1,\theta)] \approx Var\Big[x_{nn}(k,\theta_{\text{MAP}}) + \Delta t \cdot F_{nn}\left(x_{nn}(k), u(k), \theta_{\text{MAP}}\right) + J_\theta(k)\left(\theta - \theta_{\text{MAP}}\right)\Big] \tag{4.29}$$

since $Var[x_{nn}(k,\theta_{MAP}) + \Delta t \cdot F_{nn}\left(x_{nn}(k,\theta_{MAP}), u(k), \theta_{\text{MAP}}\right)] = 0$, the variance results in

$$Var[x_{nn}(k+1,\theta)] \approx Var[J_\theta(k)\left(\theta - \theta_{\text{MAP}}\right)] = J_\theta(k) Var[(\theta - \theta_{\text{MAP}})] J_\theta(k)^\top \tag{4.30}$$

obtaining

$$Var[x_{nn}(k+1,\theta)] \approx Var[J_\theta(k)\left(\theta - \theta_{\text{MAP}}\right)] = J_\theta(k)\Sigma_\theta J_\theta(k)^\top, \tag{4.31}$$

where

$$J_\theta(k) = \frac{\partial x_{nn}(k)}{\partial \theta} + \Delta t \cdot \left(\frac{\partial F_{nn}}{\partial x_{nn}} \cdot \frac{\partial x_{nn}(k)}{\partial \theta} + \frac{\partial F_{nn}}{\partial \theta}\right)$$

considering the recursive equation, the Jacobian $J_\theta(k)$ should be properly computed starting from $x_{nn}(0) = x(0)$ that does not depend from θ.

4.3.7 SMPC with Bayesian Neural State-Space Models for Robust Control

In this section, we integrate the Bayesian Neural State-Space (BNSS) model discussed earlier into an SMPC framework to ensure constraint satisfaction in the presence of epistemic uncertainty. By leveraging the posterior predictive distribution derived in the previous section, both the expected value and variance of the model predictions at each time step can be computed. Hence, it is possible to use the predictive uncertainty in the state constraints of the MPC formulation to ensure safety with a specified confidence level.

Consider the discrete-time BNSS model:

$$\begin{cases} x_{k+1} \sim \mathcal{N}\left(x_{k+1}^{\text{mean}}, x_{k+1}^{\text{var}}\right) \\ y_k = G(x_k) \end{cases} \tag{4.32}$$

where the state updates of the neural network at each time step

$$x_{k+1} \sim \mathcal{N}\left(x_{k+1}^{\text{mean}}, x_{k+1}^{\text{var}}\right) \tag{4.33}$$

are derived with the posterior predictive distribution derived earlier, considering

$$\begin{aligned} x_{k+1}^{\text{mean}} &= x_{nn}(k,\theta_{MAP}) + \Delta t \cdot F_{nn}\left(x_{nn}(k,\theta_{MAP}), u(k), \theta_{\text{MAP}}\right) \\ x_{k+1}^{\text{var}} &= J_\theta(k)\Sigma_\theta J_\theta(k)^\top \end{aligned} \tag{4.34}$$

where $x_k \in \mathbb{R}^n$ is the state vector at time step k, $u_k \in \mathbb{R}^m$ is the control input vector at time step k, F_{nn} is the neural network approximation of the system dynamics evaluated at the MAP estimate θ_{MAP} of the network parameters and J_θ is the Jacobian of $F_{\text{NN}}(x_k, u_k, \theta)$ with respect to θ, evaluated at θ_{MAP}.

By recursively applying this procedure over the prediction horizon N_H, we obtain the mean and variance of the predicted states at each future time step. The MPC optimization problem can be formulated while accounting for the uncertainty of the prediction by adding chance constraints to the optimization problem [185]:

$$\begin{aligned} \min_{\{u_k,\ldots,u_{k+N_H-1}\}} \quad & J = \sum_{i=0}^{N_H-1} \|x_{k+i} - x_{\text{ref}}\|_Q^2 + \|u_{k+i} - u_{\text{ref}}\|_R^2 + \|x_{k+N_H} - x_{\text{ref}}\|_P^2 \\ \text{s.t.} \quad & \mathbb{P}\left(x_{\min} \leq x_{k+i} \leq x_{\max}\right) \geq 1 - \epsilon, \\ & u_{k+i} \in \mathcal{U}, \quad i = 0, \ldots, N_H - 1, \end{aligned} \tag{4.35}$$

where, considering the properties of a Gaussian distribution, the chance constraints can be reformulated and implemented deterministically as follows:

$$\begin{aligned} \min_{\{u_k,\ldots,u_{k+N_H-1}\}} \quad & J = \sum_{i=0}^{N_H-1} \left\|x_{k+i}^{\text{mean}} - x_{\text{ref}}\right\|_Q^2 + \|u_{k+i} - u_{\text{ref}}\|_R^2 + \left\|x_{k+N_H}^{\text{mean}} - x_{\text{ref}}\right\|_P^2 \\ \text{s.t.} \quad & x_{k+i+1}^{\text{mean}} = x_{k+i}^{\text{mean}} + \Delta t \cdot F_{\text{NN}}(x_{k+i}^{\text{mean}}, u_{k+i}, \theta_{\text{MAP}}), \\ & x_{k+i+1}^{\text{var}} = J_{\theta,k+i} \Sigma_\theta J_{\theta,k+i}^\top, \\ & \left\| x_{k+i}^{\text{mean}} + k_\sigma \sqrt{\text{diag}\left(x_{k+i}^{\text{var}}\right)} \right\| \leq x_{\max}, \\ & \left\| x_{k+i}^{\text{mean}} - k_\sigma \sqrt{\text{diag}\left(x_{k+i}^{\text{var}}\right)} \right\| \geq x_{\min}, \\ & u_{k+i} \in \mathcal{U}, \quad i = 0, \ldots, N_H - 1, \\ & x_0^{\text{mean}} = x_0, \quad x_0^{\text{var}} = \mathbf{0} \end{aligned} \tag{4.36}$$

In this formulation, x_{k+i}^{mean} represents the expected value of the state at time step $k+i$, while x_{k+i}^{var} is the variance due to epistemic uncertainty in the model parameters. The constraints now include a probabilistic guarantee, ensuring that the system states stay within the bounds $x_{\max}$ and $x_{\min}$ with a probability $1 - \epsilon$. Where ϵ represents the allowable probability of constraint violation and is closely tied to k_σ, which is the parameter that determines the confidence level of constraint satisfaction. For a given ϵ, k_σ can be computed as

$$k_\sigma = \Phi^{-1}\left(1 - \frac{\epsilon}{2}\right), \tag{4.37}$$

where $\Phi^{-1}(\epsilon)$ is the inverse of the Cumulative Distribution Function (CDF) of the standard normal distribution. This approach, based on the Laplace approximation, ensures that the probability of violating the constraints remains below a specified threshold, providing a probabilistic guarantee of constraint satisfaction and enhancing the robustness of the control system in the presence of epistemic uncertainty. Is it worth to notice that the proposed approach could be easily extended to consider asymmetric confidence intervals.

To implement the proposed SMPC framework, several key considerations must be addressed. First, the computation of the Jacobians J_θ at each time step over the prediction horizon is essential for propagating uncertainty, and this can be efficiently achieved using automatic differentiation tools [203]. A significant feature of this approach is the ability to adjust constraints based on the uncertainty, using k_σ to balance performance and safety. A higher k_σ results in tighter constraints and lower risk of constraint violation, while a lower value may force more aggressive control actions but increases the likelihood of constraint violation. In terms of benefits, by incorporating uncertainty into the SMPC framework, it is possible to obtain an efficient balance between performance and safety, offering probabilistic guarantees of constraint satisfaction for controlling systems under epistemic uncertainty.

4.4 Ship Maneuvering Process

In this section, the proposed methodology is tested on a nonlinear benchmark. In particular, to the ship maneuvering process. First, the physical model of the ship is described, followed by an explanation of how the dataset used for training and testing the ANNs were generated. Next, the results of a preliminary comparative study are discussed, focusing on the selection of a computationally efficient ANN model. Subsequently, the selected model is trained to account for epistemic uncertainty in the system, and its application within the SMPC framework is evaluated and analyzed.

4.4.1 Nonlinear Dynamical Model

One of the most appreciated mathematical models used to describe ship maneuvering in open sea was developed by the Japanese Maneuvering Modeling Group (MMG) [204]. According to the 3-DOF MMG model, the ship velocities are described by the following set of differential equations

$$\begin{cases} \dot{v}_x(t) &= \frac{X_H+X_R+X_P+mX_Gv_\psi^2(t)+(m+m_y)v_yv_\psi(t)}{m+m_x}, \\ \dot{v}_y(t) &= \frac{Y_H+Y_R-(m+m_x)v_xv_\psi(t)-\frac{mX_G(N_H+N_R)}{I_{zG}+x_G^2m+J_z}+v_xv_\psi(t)m^2X_G^2}{m+m_y-\frac{m^2+X_G^2}{I_{zG}+x_G^2m+J_z}}, \\ \dot{v}_\psi(t) &= \frac{N_H+N_R-mX_G\left(\frac{Y_H+Y_R-(m+m_x)v_xv_\psi(t)-\frac{mX_G(N_H+N_R)}{I_{zG}+x_G^2m+J_z}+v_xv_\psi(t)m^2X_G^2}{m+m_y-\frac{m^2+X_G^2}{I_{zG}+x_G^2m+J_z}}+v_xv_\psi(t)\right)}{I_{zG}+x_G^2m+J_z}. \end{cases} \tag{4.38}$$

where v_x, v_y, and v_ψ are respectively surge, sway and yaw velocities, m is the ship mass, while m_x and m_y are added masses due to the interaction between ship and water. I_{zG} and J_z are the moment of inertia around the center of gravity x_G of the ship and the added moment of inertia, respectively. X, Y and N are hydrodynamic forces acting on the ship, and are described by

$$\begin{cases} X &= X_H + X_P + X_R \\ Y &= Y_H + Y_P + Y_R \\ N_m &= N_H + N_R \end{cases} \tag{4.39}$$

Subscript H, P, and R means hull, propeller, and rudder, respectively. Equations (4.38-4.39) are represented in a compact form. Indeed, all the forces can be decomposed in several components according to their sources. Among these forces, there are those that are functions of δ and n_p, which are the inputs of the system, and are the rudder angle and the propeller speed, respectively. A detailed description of the mathematical model and the set of parameters used for simulations can be found in [205]. The spatial coordinates of the ship can be achieved by integrating

$$\begin{cases} \dot{\psi}(t) &= v_\psi(t), \\ \dot{x}_0(t) &= v_x(t)\cos(\psi(t)) - v_y(t)\sin(\psi(t)) \\ \dot{y}_0(t) &= v_x(t)\sin(\psi(t)) + v_y(t)\cos(\psi(t)) \end{cases} \tag{4.40}$$

where ψ is the ship heading angle, x_0 and y_0 are the ship coordinates referred to a fixed reference frame.

4.4.2 ANN Model Assessment and Dataset Generation

The flexibility of the simulation environment provides an efficient means of collecting datasets for training and testing models, as well as evaluating

the performance of control algorithms. Indeed, the MMG model was used as reference system for testing the overall methodology. Therefore, velocities measurements were *in-silico* generated simulating the MMG model described before. The study was carried out in two main phases.

In the first phase, two different ANN architectures were evaluated to model the state derivatives of the ship, with the goal of identifying the most efficient model for approximating the ship dynamics. Specifically, a comparison was made between a LSTM model and a FNN model. In this phase, only the rudder angle δ was considered as a control input, while the propeller speed n_p was assumed constant. In this first phase, stochastic components were not introduced in the model dynamics.

The second phase focused on applying the selected model to estimate the ship accelerations while accounting for epistemic uncertainty of the model itself. During this phase, both the rudder angle and the propeller speed were included as control inputs, in order to obtain an effective model capable of managing both the inputs of the ship, with the aim to integrate it into the SMPC scheme (described in Section 4.3.7).

In both the phases, different maneuvers were simulated in order to obtain samples representative of the system under study. Since the models were aimed to describe the state evolution, once the velocity measurements were collected, the accelerations were computed numerically. To this aim, defining a generic measurement of velocity $v(k)$ at sample time k, the state derivative (*i.e.* acceleration) can be approximated as

$$a(k) = \frac{v(k+1) - v(k)}{\Delta t}, \tag{4.41}$$

where Δt is the sampling period and was fixed to $\Delta t = 0.2[s]$. According to Eq.4.10 the input array provided to the models $\mathbf{D}_{\text{in}}$ was defined in the first phase as

$$\mathbf{D}_{\text{in}_1} = [v_x, v_y, v_\psi, \delta], \tag{4.42}$$

while in the second phase also the propeller speed was considered

$$\mathbf{D}_{\text{in}_2} = [v_x, v_y, v_\psi, \delta, n_p], \tag{4.43}$$

while the output dataset used in the training phases, $\mathbf{D}_{\text{out}}$, considered all the accelerations of the ship

$$\mathbf{D}_{\text{out}} = [a_x, a_y, a_\psi], \tag{4.44}$$

In all the experiments, the datasets were properly scaled using the maximum absolute value scaling technique (discussed in Section 2.5.5), the weights matrices of the networks were initialized according to [77], while the bias vectors were initialized to zero. Furthermore, all the network architectures were trained using the Adam optimizer [12].

4.4.3 Analysis of Neural Network Models for Ship Maneuvering System Modeling

In this section, the comparative study on different ANN architectures for modeling the nonlinear dynamics of a ship maneuvering process, based on the results of a previous work ([200]) is summarized. In this first phase, an LSTM model was compared to a FNN to model the states variations of the system under study. The idea was to enhance the model accuracy by incorporating knowledge of the system into the training process. This was achieved by estimating the derivatives of the ship velocities and using them as an approximation of the physical laws governing the system. In this phase, the training and validation datasets (showed in Fig. 4.1 and in Fig. 4.2) were obtained assuming the propeller speed n_p constant ($n_p = 10.28 \quad round/s$) ans simulating the MMG model starting from initial conditions

$$\begin{bmatrix} v_x(0) \\ v_y(0) \\ v_\psi(0) \end{bmatrix} = \begin{bmatrix} 1.26 & m/s \\ 0 & m/s \\ 0 & rad/s \end{bmatrix}.$$

The training dataset was obtained considering several subsequent maneuvers, in particular, a Turning 10° for 125[s], 0° for 125[s], a ZigZag 10°/10° for 230[s] and a ZigZag 20°/20° for 230[s] are simulated. The validation dataset (Fig. 4.2) was aimed for the selection of the best model among those obtained at each training epoch in order to avoid over-fitting on training data. It was built combining several maneuvers: Turning 27° for 125[s], 0° for 125[s], a ZigZag 5°/15° for 230[s],a ZigZag 30°/25° for 230[s], Turning -15° for 125[s] and a Turning 15° for 125[s]. Finally, all the selected models were tested on a single maneuver, a ZigZag 15°/15°. In all the simulations, during rudder angle changes, the absolute value of the rudder steering rate was considered constant $|\dot{\delta}| \leq 2.3[^\circ/s]$. The dataset for training the model were obtained as discussed in Section 4.4.3 and the training loss function $L(\theta)$ was intended to incorporate physical information into network computations, accounting for the difference between the network predicted state derivatives and the outcome provided by the physical laws

$$\begin{aligned} L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \Bigg[& \left(a_x^{(i)} - F_{NN,x}\left(v_x^{(i)},\ v_y^{(i)},\ v_\psi^{(i)},\ \delta^{(i)};\ \theta\right)\right)^2 \\ & + \left(a_y^{(i)} - F_{NN,y}\left(v_x^{(i)},\ v_y^{(i)},\ v_\psi^{(i)},\ \delta^{(i)};\ \theta\right)\right)^2 \\ & + \left(a_\psi^{(i)} - F_{NN,\psi}\left(v_x^{(i)},\ v_y^{(i)},\ v_\psi^{(i)},\ \delta^{(i)};\ \theta\right)\right)^2 \Bigg] \end{aligned} \tag{4.45}$$

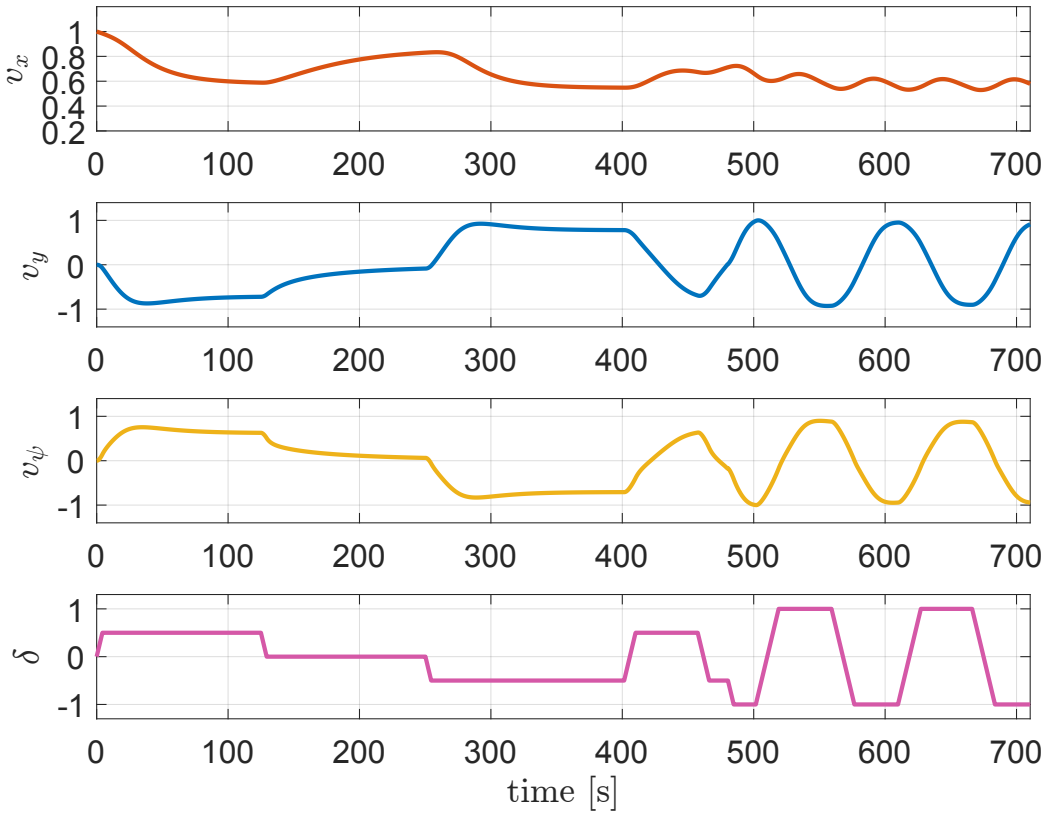

Figure 4.1: Normalized training dataset.

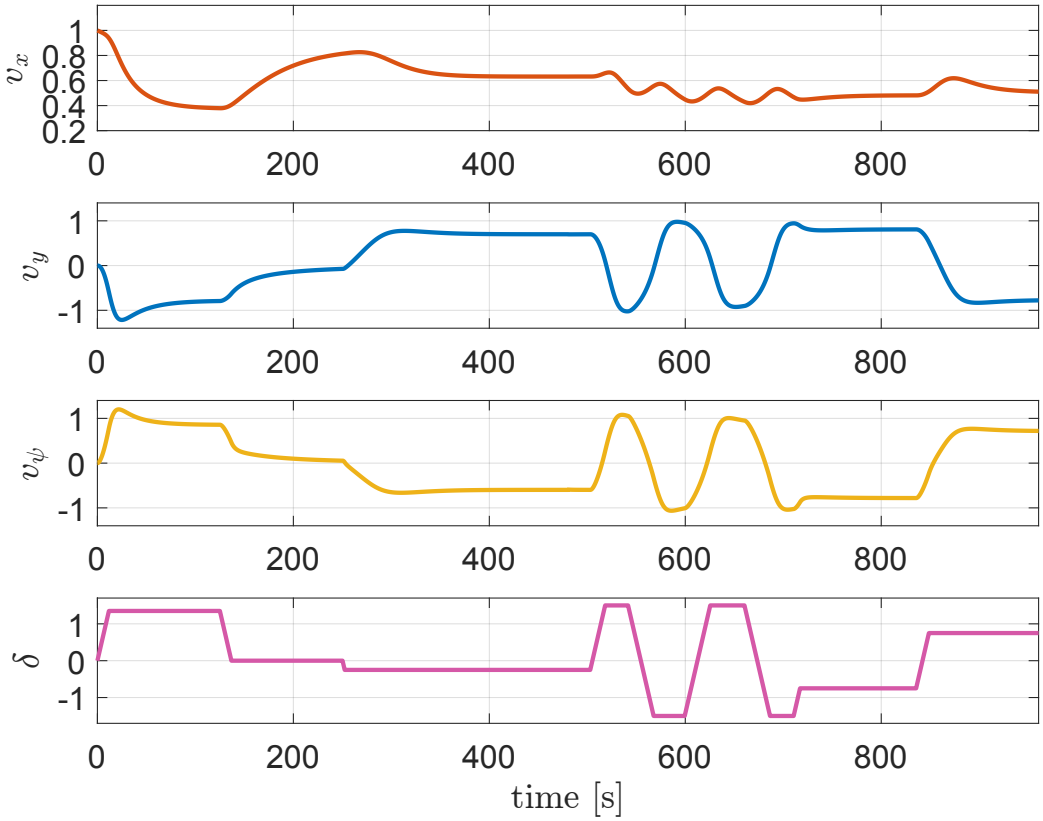

Figure 4.2: Normalized validation dataset.

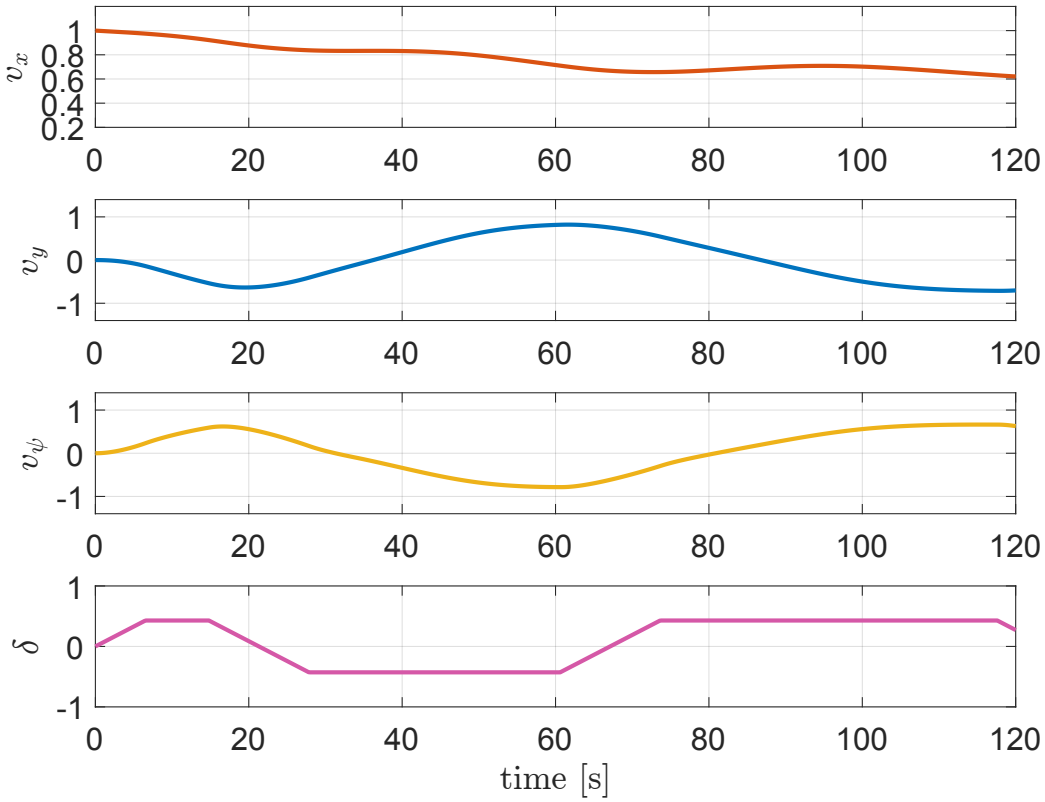

Figure 4.3: Normalized test dataset.

where N is the number of samples considered for the update of the network parameters and F_{nn} is the function expressed by the ANN model. The ANN architectures were selected with a trial and error approach. The LSTM architecture was made up of 4 layers

1. LSTM layer 1 with 100 units (N_1=100),
2. LSTM layer 2 with 100 units (N_2=100),
3. LSTM layer 3 with 100 units (N_3=100),
4. Linear regression layer with 3 units (N_L=3).

while The FFN architecture was composed of 2 layers

1. FNN layer with 100 units (N_1=100),
2. Linear regression layer with 3 units (N_L=3).

Both networks were trained for 3000 training epochs ($N_e = 3000$), with a learning rate $\alpha = 10^{-3}$. Their effectiveness in modelling the states of the ship was accessed incorporating them in a recursive ANN State-Space model (described in Section 4.3.2)

$$\begin{cases} x_{nn}(k+1) = x_{nn}(k) + \Delta t \cdot (F_{nn}(x_{nn}(k), \delta(k), \theta)) \\ \hat{y}(k) = (x_{nn}(k)) \end{cases} \tag{4.46}$$

where x_{nn} represents the three velocities $[v_{x_{nn}}, v_{y_{nn}}, v_{\psi_{nn}}]$ computed with the model starting from a known initial condition

$$\begin{bmatrix} v_{x_{nn}}(0) \\ v_{y_{nn}}(0) \\ v_{\psi_{nn}}(0) \end{bmatrix} = \begin{bmatrix} v_x(0) \\ v_y(0) \\ v_\psi(0) \end{bmatrix}.$$

Providing the input sequence δ and the measured initial conditions of the validation and test dataset, the NSS models were used to simulate the target system. The velocities produced by the NSS models were comparable to those generated by the MMG model showing exceptional performance of the NSS models in modelling nonlinear dynamics. The performance in terms of RMSE and FIT score of the two models in reconstructing the ship states are summarized in Tab.4.1.

The results of the study showed that ANNs can be employed successfully as approximators of the ship accelerations. Moreover, the NSS models, both

Model	Variable	RMSE	FIT (%)
LSTM	v_x	0.0066	95.30
	v_y	0.0039	96.93
	v_ψ	0.0013	96.35
FNN	v_x	0.0010	97.23
	v_y	0.0070	94.77
	v_ψ	0.0045	96.22

Table 4.1: Performance comparison of the LSTM and FNN models based on test set RMSE and FIT scores.

the LSTM and FNN versions, obtained satisfactory prediction accuracy for all velocity components. The one employing the LSTM model achieved FIT scores above 95% across all variables, and the one based on the FNN model, despite being based on a simpler architecture, showed comparable performance with slightly higher accuracy for the surge velocity v_x. Therefore, the FNN architecture was chosen as an efficient choice for modelling the ship accelerations.

4.4.4 Incorporating Noise and Uncertainty Estimation

In order to account for system uncertainties, stochastic disturbances were incorporated into the MMG model, acting on the state equations. These disturbances were introduced to capture real-world phenomena such as environmental variability (e.g., wind, waves, and currents) which are difficult to model deterministically. By introducing stochastic components, the system dynamics better reflect these uncertainties, enhancing the robustness of the used control and estimation strategies. Specifically, for each state in the MMG model (Eq.4.38), a disturbance with a certain magnitude was assumed to affect the system dynamics. The nonlinear dynamical model comprehensive of the stochastic component can be reformulated as

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t), \rho) + e(t) \\ y(t) &= x(t) \end{aligned} \tag{4.47}$$

where

$$x(t) = \begin{bmatrix} v_x(t) \\ v_y(t) \\ v_\psi(t) \end{bmatrix}, \quad u(t) = \begin{bmatrix} \delta(t) \\ n_P(t) \end{bmatrix}, \quad e(t) = \begin{bmatrix} e_{v_x}(t) \\ e_{v_y}(t) \\ e_{v_\psi}(t) \end{bmatrix}.$$

where e(t) was added to the state equations to simulate the stochastic effects, obtaining the formulation described in Eq.4.8. This modification results in a stochastic dynamical system, where the underlying disturbances cannot be fully captured through a deterministic model. Such stochastic dynamics necessitate the use of advanced modeling techniques, such as the Bayesian Neural State-Space model described in Eq.4.32, which accounting for the uncertainty, can more effectively capture the system stochastic behavior. The stochastic component introduced in Eq.4.47 was approximated as band-limited white noise, which assumes a flat power spectral density (PSD) over the relevant frequency range of the system. This approximation reflects the nature of practical disturbances, where true white noise (with infinite bandwidth) is not physically realistic, but disturbances over a limited bandwidth closely match a real application. The band-limited white noise was chosen with a correlation time $t_c = 0.2[s]$ and a power spectral density of 0.01% the maximum amplitude of the velocities of the ship.

$$e(t) = \begin{bmatrix} e_{v_x}(t) \\ e_{v_y}(t) \\ e_{v_\psi}(t) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \beta_{v_x} & 0 & 0 \\ 0 & \beta_{v_y} & 0 \\ 0 & 0 & \beta_{v_\psi} \end{bmatrix} \right) \tag{4.48}$$

where

$$\beta_{v_x} = \frac{t_c}{0.00001 \cdot v_{x_{max}}} \tag{4.49}$$

$$\beta_{v_y} = \frac{t_c}{0.00001 \cdot v_{y_{max}}} \tag{4.50}$$

$$\beta_{v_\psi} = \frac{t_c}{0.00001 \cdot v_{\psi_{max}}} \tag{4.51}$$

This allowed the noise to affect the system in a way that mimics real-world randomness, without overwhelming the slower dynamics of the ship. The noise power was carefully selected to introduce a small, realistic disturbance that affects the system state variables, such as surge, sway, and yaw, without destabilizing the simulation. By incorporating these disturbances, the model more realistically captures the variability and uncertainties that a ship could encounter during operation. This stochastic behavior allows the system to be more effectively modeled and controlled using probabilistic approaches, such as the BNSS model described in Section 4.1.4, ensuring better performance in real-world conditions where perfect deterministic control is not feasible.

4.4.5 Bayesian Neural State-Space training and optimization

For the second phase of this study, the training dataset was generated considering the following maneuvers: a Turning 10° for 125[s], a Turning -10° for 125[s], a ZigZag 10°/10° for 125[s] and a ZigZag 20°/20° for 230[s] (Fig. 4.4). The maneuvers were simulated for three different propeller speed, $n_p = [7, 10, 13]$ and the final training dataset was obtained merging the results of the simulations. Then, multiple maneuvers were combined to generate the validation set (Fig. 4.5), including Turning 27° for 125[s], Turning -27° for 125[s], ZigZag 5°/15° for 250[s], ZigZag 30°/25° for 250[s], Turning -15° for 125[s] and Turning 15° for 125[s]. Finally, The accuracy and generalization capabilities of the best model were tested on a test dataset that accounted for the following maneuvers: Turning 15° for 125[s], Turning -15° for 125[s], a ZigZag 25°/30° for 250[s],a ZigZag 10°/20° for 250[s], Turning -30° for 125[s] and a Turning 30° for 125[s](Fig. 4.6). For both validation and test datasets, the propeller speed was set to change randomly every 50 [s] during the simulation to account for multiple inputs combinations.

Once the dataset were generated, the Bayesian Neural State-Space (BNSS) model was trained using the Automatic Differentiation library of PyThorch [206]. For this purpose, the custom loss function $L(\theta)$ described in Section4.3.4 was derived accounting for the three different states of the system, the precision β of every stochastic component, and the prior distribution with precision γ of the model parameters.

$$L(\theta) = \frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{3}\beta_i\delta_i^2(k) + \frac{\gamma}{2}\sum_{i=1}^{n_\theta}\theta_i^2. \tag{4.52}$$

Here β_i were assumed known and γ was fixed to 0.001, as regularization term of the loss function. The model was optimized using MBGD with a batch size $B_s = 64$ samples per batch. The best model was found after 468 training epochs accounting for high FIT scores for all the velocities (see Table 4.2).

Model	**Variable**	**Validation FIT (%)**	**Test FIT (%)**
BNSS	v_x	84.3	83.9
	v_y	94.9	93.9
	v_ψ	95.6	92.8

Table 4.2: FIT scores obtained with the BNSS model on the validation and test datasets.

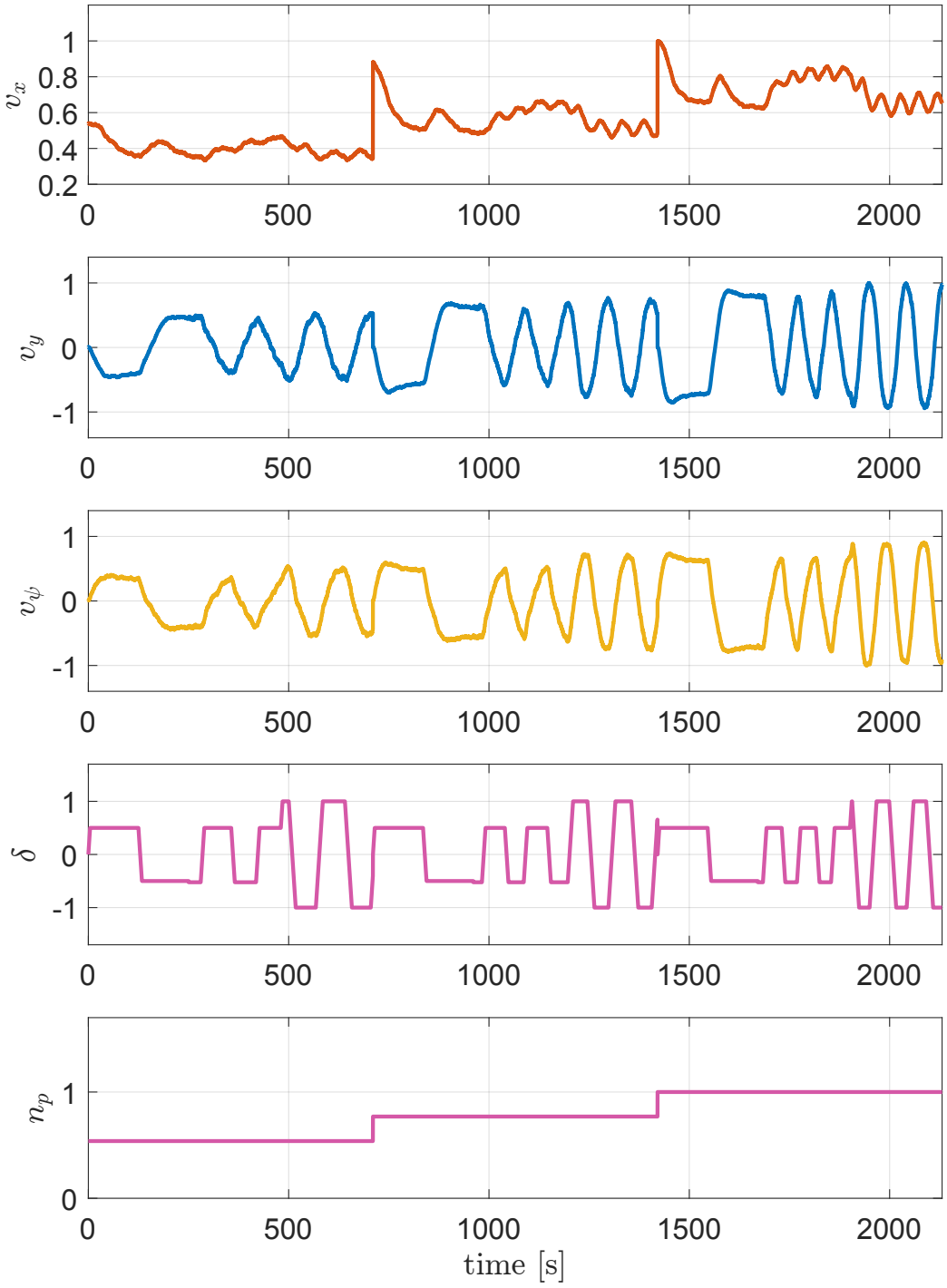

Figure 4.4: Normalized training dataset with uncertainty.

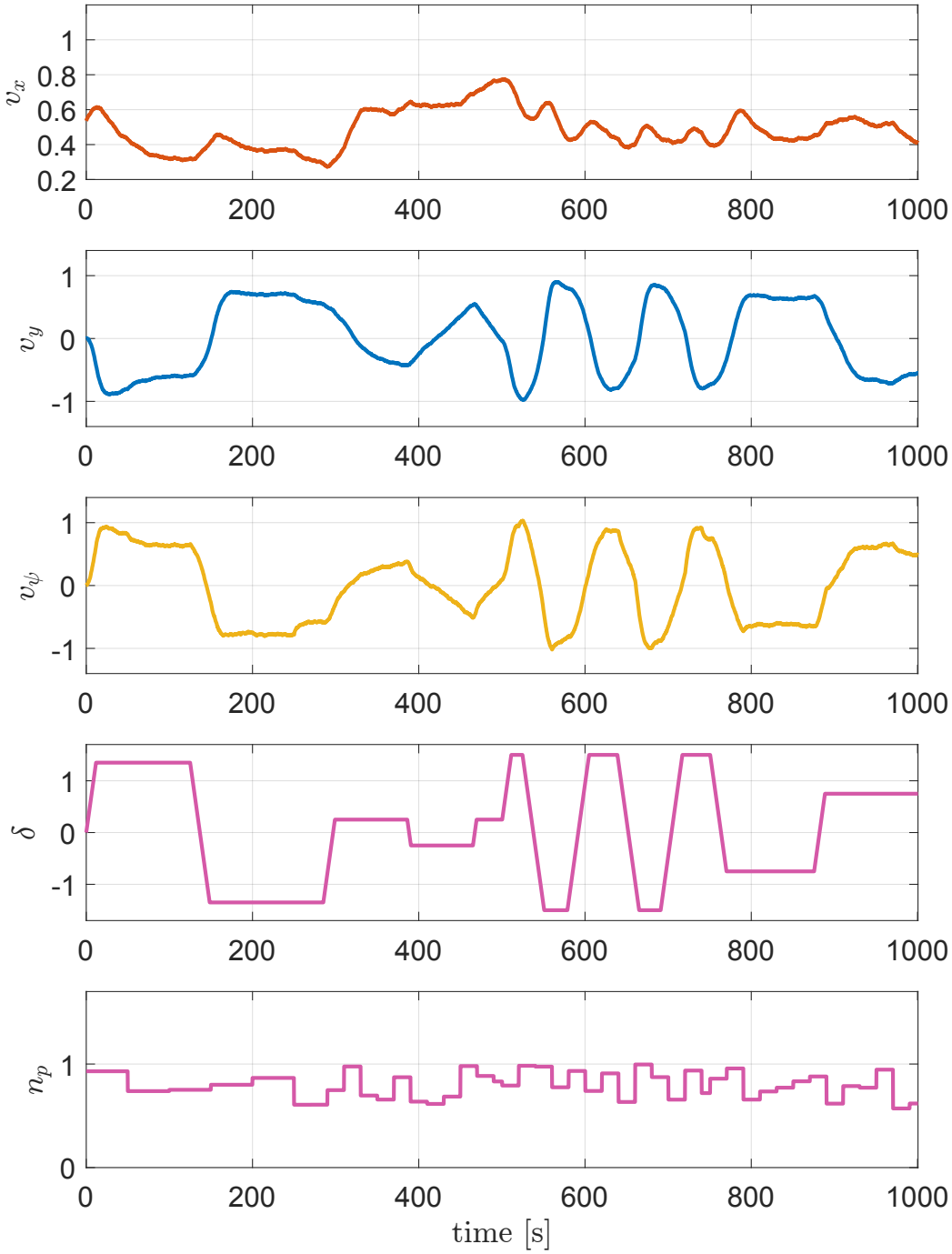

Figure 4.5: Normalized validation dataset with uncertainty.

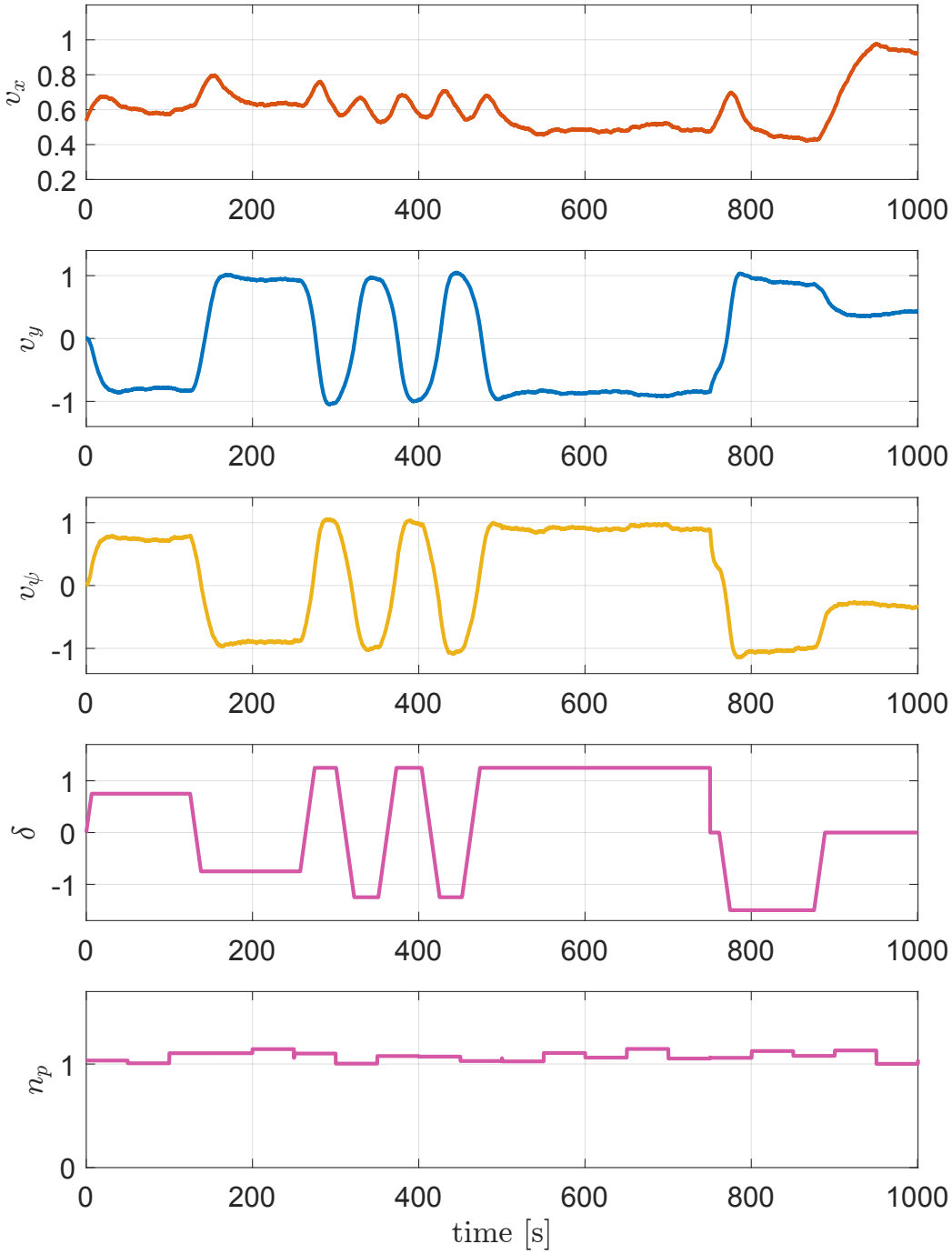

Figure 4.6: Normalized test dataset with uncertainty.

4.4.6 SMPC with state uncertainty constraints for ship maneuvering

The trained BNSS model, was then used inside the SMPC framework described in Section 4.3.7, with the aim of controlling the ship in reaching a desired path ensuring constraints satisfaction while performing the maneuvers. At the beginning of the simulation, the ship was assumed to have the following initial conditions

$$\begin{bmatrix} v_x(0) \\ v_y(0) \\ v_\psi(0) \\ \psi(0) \\ x_0(0) \\ y_0(0) \end{bmatrix} = \begin{bmatrix} 1.26 & m/s \\ 0 & m/s \\ 0 & rad/s \\ 0 & rad \\ 0 & m \\ 0 & m \end{bmatrix}.$$

The control task was to provide an optimal input sequence to the ship in order to reach $y_{0_{ref}} = 3\ m$. Moreover, a constraint on the yaw rate of the ship was imposed $|v_\psi| \leq v_{\psi_{\text{lim}}}$, with $v_{\psi_{\text{lim}}} = 0.03\ rad/s$. Indeed, limiting the yaw rate allows for smoother, safer, and more efficient operation of the ship [207, 208]. Some consideration of physical constraints relative to inputs were made for the simulations. The rudder angle was supposed to vary in a limited range $|\delta| \leq +45°$ with a certain rate $|\dot{\delta}| \leq 2.3\ °/s$, and since the MMG model was aimed to represent ship maneuvering in open sea, the propeller speed was assumed to vary in a proper range $7\ rps \leq n_p \leq 13\ rps$. The optimal input sequence over the considered horizon of length N_H was computed at every time step solving the following optimization problem:

$$\begin{aligned}
\min_{\{n_{p_k},\dots,n_{p_{k+N_H-1}},\delta_k,\dots,\delta_{k+N_H-1}\}} \quad & J = \sum_{i=0}^{N_H-1} \left\| y^{\text{mean}}_{0_{k+i}} - y_{0_{\text{ref}}} \right\|^2_Q + \left\| y^{\text{mean}}_{0_{k+N_H}} - y_{0_{\text{ref}}} \right\|^2_P \\
\text{s.t.} \quad & v^{\text{mean}}_{\psi_{k+i+1}} = v^{\text{mean}}_{\psi_{k+i}} + \Delta t \cdot F_{\text{NN}}(v^{\text{mean}}_{\psi_{k+i}}, \delta_{k+i}, n_{p_{k+i}}, \theta_{\text{MAP}}), \\
& v^{\text{var}}_{\psi_{k+i+1}} = J_{\theta,k+i} \Sigma_\theta J^\top_{\theta,k+i}, \\
& \left\| v^{\text{mean}}_{\psi_{k+i}} + k_\sigma \sqrt{\left(v^{\text{var}}_{\psi_{k+i}} \right)} \right\| \leq v_{\psi_{\max}}, \\
& 7[\text{rps}] \leq n_{p_{k+i}} \leq 13[\text{rps}], \\
& |\delta_{k+i}| \leq 45°, \\
& |\dot{\delta}_{k+i}| \leq 2.3°/\text{s}, \\
& v^{\text{mean}}_{\psi_k} = v_{\psi_k}, \quad v^{\text{var}}_{\psi_k} = \mathbf{0}, \quad i = 0, \dots, N_H - 1
\end{aligned} \tag{4.53}$$

were Q and P are the gains of the SMPC that have been properly tuned. In this work, both the gains were obtained with a trial and error approach ($Q = 100, P = 10$). $v^{\text{mean}}_{\psi_{k+i+1}}$,$v^{\text{var}}_{\psi_{k+i+1}}$ and $y^{\text{mean}}_{0_{k+i}}$ were obtained from the BNSS model and $k_\sigma = 3$ was selected in order to obtain a confidence level of 99.73% for constraint satisfaction. Finally, to evaluate the effectiveness in incorporating the epistemic uncertainty of the model, the control problem (4.53) was solved without considering the uncertainty-related term with a NMPC.

$$
\begin{aligned}
\min_{\{n_{p_k},\dots,n_{p_{k+N_H-1}},\delta_k,\dots,\delta_{k+N_H-1}\}} \quad & J = \sum_{i=0}^{N_H-1} \left\| y^{\text{mean}}_{0_{k+i}} - y_{0_{\text{ref}}} \right\|^2_Q + \left\| y^{\text{mean}}_{0_{k+N_H}} - y_{0_{\text{ref}}} \right\|^2_P \\
\text{s.t.} \quad & v^{\text{mean}}_{\psi_{k+i+1}} = v^{\text{mean}}_{\psi_{k+i}} + \Delta t \cdot F_{\text{NN}}(v^{\text{mean}}_{\psi_{k+i}}, \delta_{k+i}, n_{p_{k+i}}, \theta_{\text{MAP}}), \\
& v^{\text{var}}_{\psi_{k+i+1}} = J_{\theta,k+i} \Sigma_\theta J^\top_{\theta,k+i}, \\
& \left\| v^{\text{mean}}_{\psi_{k+i}} \right\| \leq v_{\psi_{\max}}, \\
& 7[\text{rps}] \leq n_{p_{k+i}} \leq 13[\text{rps}], \\
& |\delta_{k+i}| \leq 45^\circ, \\
& |\dot{\delta}_{k+i}| \leq 2.3^\circ/\text{s}, \\
& v^{\text{mean}}_{\psi_k} = v_{\psi_k}, \quad v^{\text{var}}_{\psi_k} = \mathbf{0}, \quad i = 0, \dots, N_H - 1
\end{aligned}
\tag{4.54}
$$

4.4.7 Results

The closed-loop simulations with NMPC (4.54) and SMPC (4.53) controllers are presented in Fig.4.7 and Fig.4.8. From Fig.4.7, it appears evident that the BNSS-based MPC successfully directed the ship to achieve the desired y_0 coordinate in both cases, demonstrating its effectiveness in guiding the ship to the target despite the presence of inherent stochastic components acting on the system. However, as shown in Fig.4.8-A, this uncertainty led to temporary violations of the yaw rate constraints during the maneuver, particularly in situations where the system dynamics deviated from the nominal model predictions. Nevertheless, by explicitly incorporating the epistemic uncertainty into the yaw rate constraints as chance constraints, the controller adapted to account for system deviations, ensuring the constraints were more consistently satisfied (Fig.4.8-B). This highlights the advantage of uncertainty-aware control strategies, as they enable handling of stochastic system behaviors while maintaining operational safety and respecting dynamic constraints.

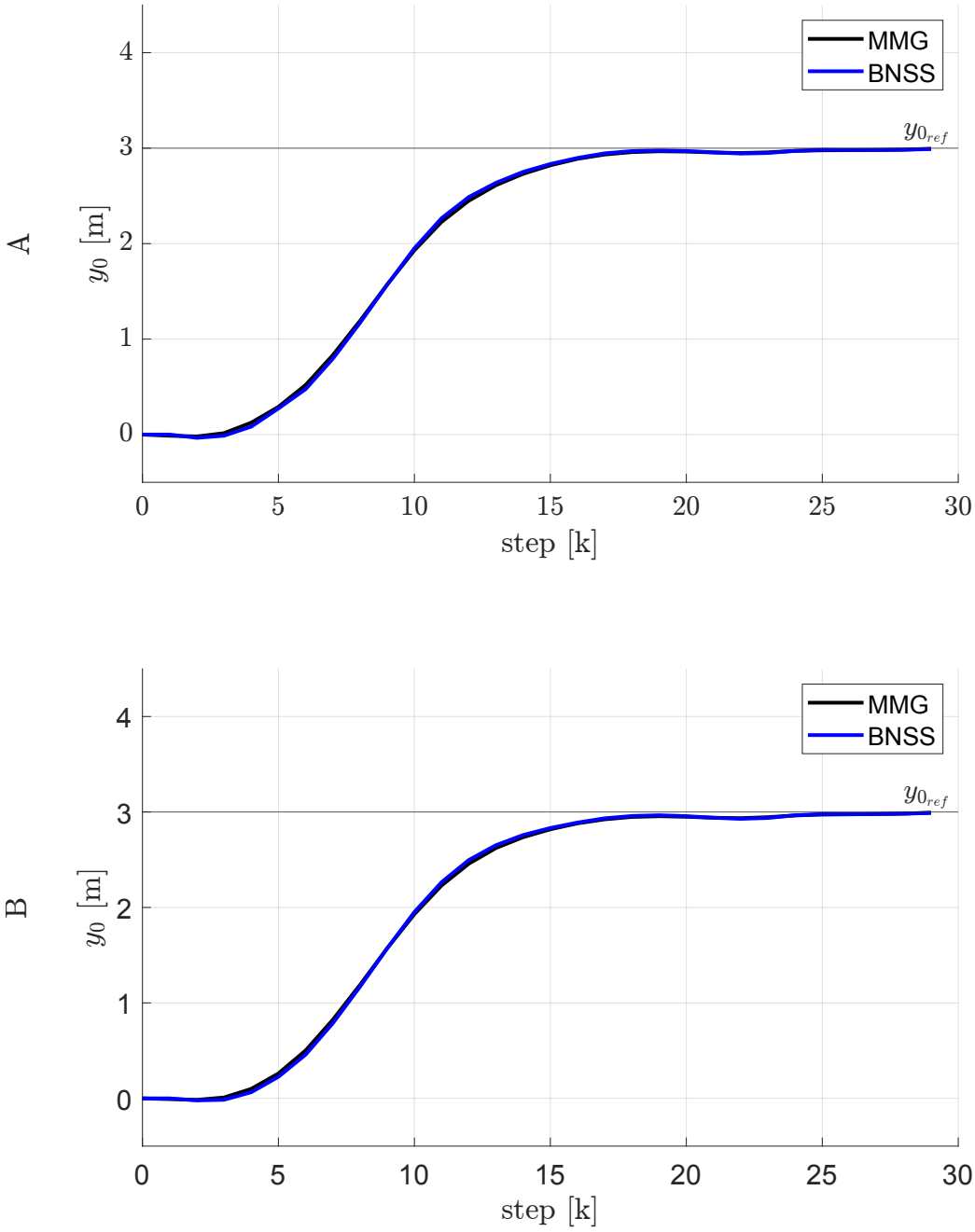

Figure 4.7: Ship y_0 coordinate with NMPC (A) and with SMPC (B).

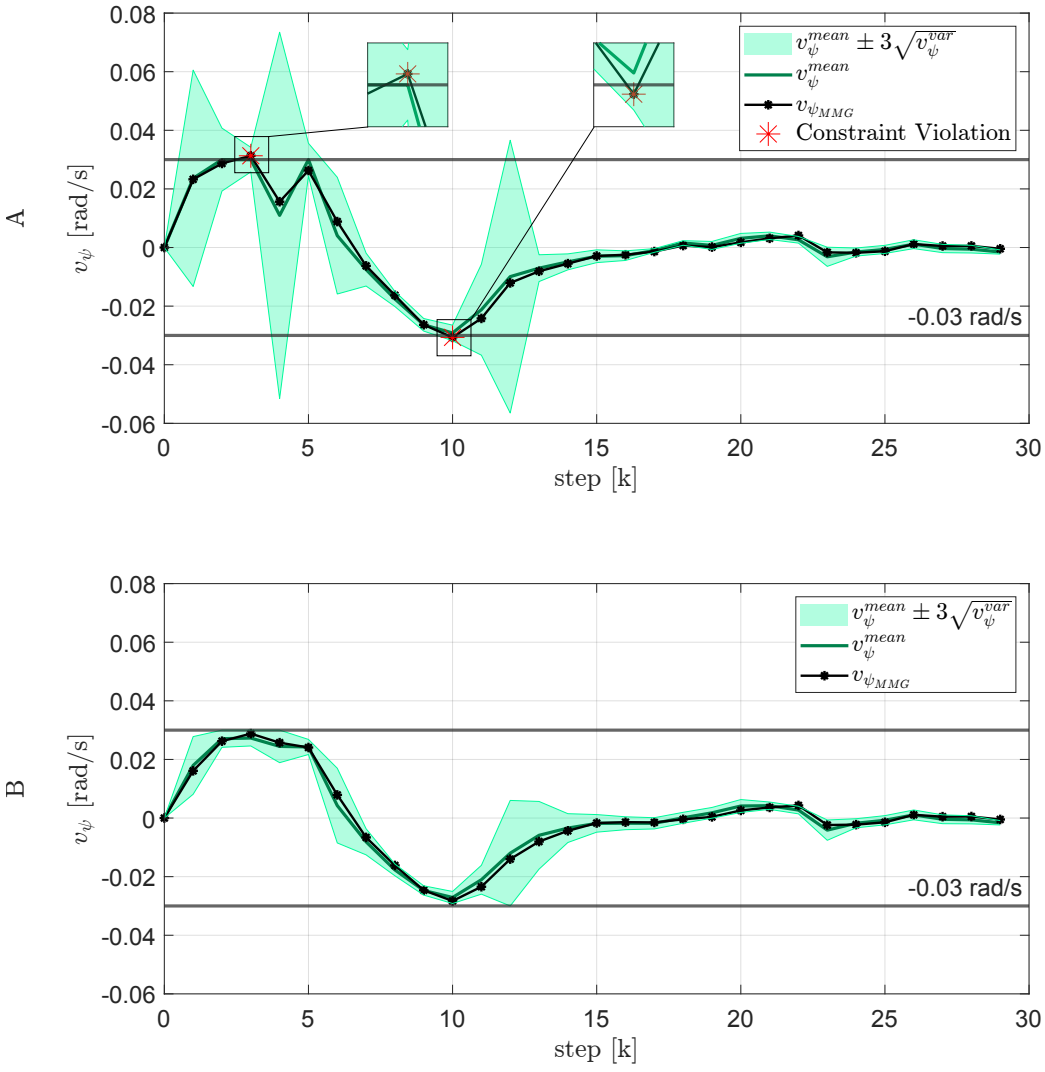

Figure 4.8: Yaw rate predicted by the BNSS (v_ψ^{mean}), the considered uncertainty band ($\pm 3\sqrt{v_\psi^{var}}$) and the MMG Yaw rate ($v_{\psi_{MMG}}$) with NMPC (A) and SMPC (B).

4.5 Concluding Remarks

In this chapter, we demonstrated the potential of ANNs for modeling complex nonlinear systems. By focusing on the estimation and incorporation of epistemic uncertainty, we enhanced the reliability and robustness of ANN models, making them suitable for use in a MPC framework. The theoretical exploration highlighted the critical distinctions between epistemic and aleatoric uncertainty, both of which play essential roles in determining model accuracy. The ability of BNNs to provide a principled approach for capturing and quantifying uncertainty has been discussed. Moreover, by considering the Bayesian framework, the MPC becomes more resilient to unexpected variations in system dynamics, as demonstrated by the integration of uncertainty into the control problem.

Through *in-silico* simulations applied to the ship maneuvering process, we validated the ANN-based models ability to guide the ship to follow a desired trajectory. Notably, it has been observed that stochastic components can lead to constraint violations during maneuvers. However, by incorporating the epistemic uncertainty directly into the MPC cost function, the controller was able to adapt and respect the physical constraints, such as those imposed on the yaw rate.

The results underscore the importance of uncertainty-aware control strategies in enhancing the robustness of MPC algorithms. By effectively managing the trade-offs between control performance and constraint satisfaction, this approach enables safer and more efficient system operation, even in the presence of significant model uncertainty.

In conclusion, this work demonstrates the viability of combining ANN models with Bayesian inference and stochastic control methods to address complex real-world control challenges, providing a powerful framework for future developments in robust control and predictive modeling.

Chapter 5

Conclusion

This thesis has explored the application of ANN-based methods for the management and control of complex systems, focusing on two critical areas: leak detection and sensor placement in WDNs, and the incorporation of epistemic uncertainty in ANN models for constrained SMPC. By leveraging advanced ANN architectures and methodologies, significant real-world problems have been addressed.

The initial chapter, Chapter 2, provided a comprehensive introduction to artificial neural networks, tracing their development from the perceptron to more advanced architectures such as MLP and RNN. This foundational understanding was essential for appreciating the capabilities of ANNs in modeling complex patterns and dynamics, laying the groundwork for their application in the rest of the thesis.

In Chapter 3, the issue of leak detection and sensor placement in WDNs was addressed. Recognizing the substantial water losses and financial implications of pipeline bursts and leaks, a novel methodology was developed, integrating spectral clustering and GRNN. The WDN was partitioned into virtual clusters using graph theory-based spectral clustering, which reduced computational complexity while preserving the hydraulic behavior of the network. Multiple scenarios accounting for spatial and temporal uncertainties in nodal demands and burst locations were generated, providing a robust dataset for model training and validation. A GRNN was employed to optimize the placement of flow and pressure sensors, and the group regularization technique effectively identified the most informative sensor locations, minimizing redundancy and costs. The results demonstrated high detection accuracy, even with a minimal number of sensors, and the exclusive use of flow meters aligns with industry insights regarding the importance of flow data in burst detection.

Chapter 4 delved into modeling and controlling nonlinear dynamical sys-

tems using ANNs, with a particular focus on handling epistemic uncertainty. To this aim, a detailed discussion on the types of uncertainties in ANNs and their sources has been provided. Indeed, a clear understanding of uncertainty is crucial for enhancing model reliability and confidence in predictions. How it is possible to derive the posterior distribution of parametric models, as ANNs, with the Laplace approximation has been addressed. The BNSS framework and its possible integration into a SMPC algorithm have been detailed. Finally, the methodology was tested on the ship maneuvering process, a complex nonlinear system. The incorporation of epistemic uncertainty into the control schema as chance constraints led to more robust and reliable control actions, particularly in the presence of stochastic disturbances.

This thesis makes several significant contributions to the fields of water resource management and control engineering. The integration of spectral clustering and GRNN presents a novel approach to sensor placement in WDNs, balancing cost and performance, and offering practical solutions for utility companies to enhance leak detection and reduce water losses. Moreover, the incorporation of epistemic uncertainty into control algorithms using BNNs enhances the robustness and reliability of control systems, especially for applications in uncertain environments such as autonomous navigation and complex industrial processes. These methodologies were validated through in-silico case studies, demonstrating the applicability and performance of ANN-based methods in managing and controlling complex systems.

While this thesis has made considerable advancements, there are opportunities for future research. The methodology in Chapter 2 primarily recognized flow sensors as most informative inputs, but due to their cost of maintenance and installation future work could explore techniques for a better balance on the selection of flow and pressure sensors. Additionally, the approach could be adapted to different network topologies and configurations to further enhance its general applicability. In Chapter 3, the use of the Laplace approximation, while computationally efficient, may not capture multi-modal posterior distributions. Future research could investigate alternative methods to approximate the posterior distribution. Expanding the application of the proposed methodology to more sophisticated ship models or different environmental conditions could further validate the approach, and real-world experiments would increase the practical relevance of the work. Moreover, a promising extension would involve optimizing the weighting matrices in the SMPC framework based on the derived uncertainty estimates, potentially improving the trade-off between performance and robustness in uncertain environments.

Beyond the specific applications detailed in this thesis, the methodologies developed here can be extended to a broad range of other fields and

challenges. The sensor placement and fault detection strategy that combines graph-based spectral clustering with group-regularized neural networks, for example, could be applied to large-scale networks such as smart grids, gas/oil pipeline networks, industrial plants, and environmental monitoring systems, where strategic sensor deployment is crucial for reliable anomaly detection and efficient resource allocation. Likewise, the uncertainty-aware modeling and control framework based on Bayesian neural networks and stochastic model predictive control has potential use in ground/aerial autonomous vehicles, robotic systems, advanced manufacturing, aerospace, and healthcare, where nonlinear dynamics and unknown parameters are common. In these settings, the ability to estimate epistemic uncertainty, particularly when data are sparse or the environment is highly variable, can significantly enhance the safety, reliability, and performance of automated decision-making.

In conclusion, this thesis has demonstrated the significant potential of ANN-based methods in addressing complex challenges in system management and control. By developing innovative methodologies that leverage advanced ANN architectures and uncertainty estimation techniques, the research has contributed to the advancement of intelligent, efficient, and robust solutions in engineering. The integration of domain-specific knowledge with ANN techniques opens up exciting opportunities for future innovations. As computational resources become more accessible, the use of such methods is expected to grow, driving progress across various industries and applications.

Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[2] A Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[4] Balázs Csanád Csáji et al. "Approximation with artificial neural networks". In: *Faculty of Sciences, Etvs Lornd University, Hungary* 24.48 (2001), p. 7.

[5] Franco Scarselli and Ah Chung Tsoi. "Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results". In: *Neural networks* 11.1 (1998), pp. 15–37.

[6] Anton Maximilian Schäfer and Hans Georg Zimmermann. "Recurrent neural networks are universal approximators". In: *Artificial Neural Networks–ICANN 2006: 16th International Conference, Athens, Greece, September 10-14, 2006. Proceedings, Part I 16*. Springer. 2006, pp. 632–640.

[7] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*. 2017.

[8] Yann LeCun et al. "Learning algorithms for classification: A comparison on handwritten digit recognition". In: *Neural networks: the statistical mechanics perspective* 261.276 (1995), p. 2.

[9] Yong Yu et al. "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7 (2019), pp. 1235–1270.

[10] Sepp Hochreiters and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[11] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems* 27 (2014).

[12] Diederik P Kingma. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[13] Kenneth J Hunt et al. “Neural networks for control systems—a survey”. In: *Automatica* 28.6 (1992), pp. 1083–1112.

[14] Irene Schimperna, Chiara Toffanin, and Lalo Magni. “On offset-free model predictive control with long short-term memory networks”. In: *IFAC-PapersOnLine* 56.1 (2023), pp. 156–161.

[15] Edoardo Vacchini et al. “Design of a deep neural network-based integral sliding mode control for nonlinear systems under fully unknown dynamics”. In: *IEEE Control Systems Letters* 7 (2023), pp. 1789–1794.

[16] Fabio Bonassi et al. “On recurrent neural networks for learning-based control: recent results and ideas for future developments”. In: *Journal of Process Control* 114 (2022), pp. 92–104.

[17] Fabio Bonassi et al. “Nonlinear MPC design for incrementally ISS systems with application to GRU networks”. In: *Automatica* 159 (2024), p. 111381.

[18] Francesca Iacono et al. “Improvement of manufacturing technologies through a modelling approach: an air-steam sterilization case-study”. In: *Procedia Computer Science* 180 (2021), pp. 162–171.

[19] Leandro Pitturelli et al. “FRAN-X: An improved diagnostic transfer learning approach with application to ball bearings fault diagnosis”. In: *IFAC-PapersOnLine* 56.2 (2023), pp. 7716–7721.

[20] Dario Piga, Marco Forgione, and Manas Mejari. “Deep learning with transfer functions: new applications in system identification”. In: *IFAC-PapersOnLine* 54.7 (2021), pp. 415–420.

[21] Jonas Degrave et al. “Magnetic control of tokamak plasmas through deep reinforcement learning”. In: *Nature* 602.7897 (2022), pp. 414–419.

[22] Sen Peng et al. “Pressure sensor placement in water supply network based on graph neural network clustering method”. In: *Water* 14.2 (2022), p. 150.

[23] Soteris A Kalogirou. “Applications of artificial neural-networks for energy systems”. In: *Applied energy* 67.1-2 (2000), pp. 17–35.

[24] Oludare Isaac Abiodun et al. “State-of-the-art in artificial neural network applications: A survey”. In: *Heliyon* 4.11 (2018).

[25] Raffaele G Cestari et al. “Split-Boost Neural Networks”. In: *IFAC-PapersOnLine* 58.15 (2024), pp. 241–246.

[26] Ahmad EL Sallab et al. “Deep reinforcement learning framework for autonomous driving”. In: *arXiv preprint arXiv:1704.02532* (2017).

[27] Bichen Wu et al. “Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2017, pp. 129–137.

[28] Francesca Iacono, Lalo Magni, and Chiara Toffanin. “Personalized LSTM-based alarm systems for hypoglycemia and hyperglycemia prevention”. In: *Biomedical Signal Processing and Control* 86 (2023), p. 105167.

[29] Samir S Yadav and Shivajirao M Jadhav. “Deep convolutional neural network based medical image classification for disease diagnosis”. In: *Journal of Big data* 6.1 (2019), pp. 1–18.

[30] Andre Esteva et al. “Dermatologist-level classification of skin cancer with deep neural networks”. In: *nature* 542.7639 (2017), pp. 115–118.

[31] Riccardo Miotto et al. “Deep learning for healthcare: review, opportunities and challenges”. In: *Briefings in bioinformatics* 19.6 (2018), pp. 1236–1246.

[32] Jakob Gawlikowski et al. “A survey of uncertainty in deep neural networks”. In: *Artificial Intelligence Review* 56.Suppl 1 (2023), pp. 1513–1589.

[33] Marco Forgione et al. “On the adaptation of recurrent neural networks for system identification”. In: *Automatica* 155 (2023), p. 111092.

[34] Simone Scardapane et al. “Group sparse regularization for deep neural networks”. In: *Neurocomputing* 241 (2017), pp. 81–89.

[35] David JC MacKay. “A practical Bayesian framework for backpropagation networks”. In: *Neural computation* 4.3 (1992), pp. 448–472.

[36] Marco Forgione, Manas Mejari, and Dario Piga. “Learning neural state-space models: Do we need a state estimator?” In: *arXiv preprint arXiv:2206.12928* (2022).

[37] Gerben Beintema, Roland Toth, and Maarten Schoukens. “Nonlinear state-space identification using deep encoder networks”. In: *Learning for dynamics and control*. PMLR. 2021, pp. 241–250.

[38] Ali Mesbah. "Stochastic model predictive control: An overview and perspectives for future research". In: *IEEE Control Systems Magazine* 36.6 (2016), pp. 30–44.

[39] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[40] Marvin Minsky and Seymour A Papert. *Perceptrons, reissue of the 1988 expanded edition with a new foreword by Léon Bottou: an introduction to computational geometry*. MIT press, 2017.

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[42] Yoshua Bengio. "Learning Deep Architectures for AI". In: *Foundations and Trends in Machine Learning*. Vol. 2. 1. 2009, pp. 1–127.

[43] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors*. Vol. 323. 1986, pp. 533–536.

[44] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366.

[45] Ronen Eldan and Ohad Shamir. "The power of depth for feedforward neural networks". In: *Conference on learning theory*. PMLR. 2016, pp. 907–940.

[46] Maithra Raghu et al. "On the expressive power of deep neural networks". In: *international conference on machine learning*. PMLR. 2017, pp. 2847–2854.

[47] Matus Telgarsky. "Benefits of depth in neural networks". In: *Conference on learning theory*. PMLR. 2016, pp. 1517–1539.

[48] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the International Conference on Machine Learning*. 2015, pp. 448–456.

[49] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[50] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)* (2015), pp. 1–15.

[51] Rylan Schaeffer et al. “Double descent demystified: Identifying, interpreting & ablating the sources of a deep learning puzzle”. In: *arXiv preprint arXiv:2303.14151* (2023).

[52] Mikhail Belkin et al. “Reconciling modern machine-learning practice and the classical bias–variance trade-off”. In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.

[53] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[54] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[55] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT’2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*. Springer. 2010, pp. 177–186.

[56] Sarit Khirirat, Hamid Reza Feyzmahdavian, and Mikael Johansson. “Mini-batch gradient descent: Faster convergence under data sparsity”. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 2880–2887.

[57] Jerome T Connor, R Douglas Martin, and Les E Atlas. “Recurrent neural networks and robust time series prediction”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 240–254.

[58] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[59] Pearlmutter. “Learning state space trajectories in recurrent neural networks”. In: *International 1989 Joint Conference on Neural Networks*. IEEE. 1989, pp. 365–372.

[60] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks* 61 (2015), pp. 85–117.

[61] Paul J Werbos. “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[62] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, ON, Canada, 2013.

[63] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning Long-Term Dependencies with Gradient Descent is Difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[64] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training Recurrent Neural Networks”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2013, pp. 1310–1318.

[65] Junyoung Chung et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).

[66] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. “Learning to forget: Continual prediction with LSTM”. In: *Neural computation* 12.10 (2000), pp. 2451–2471.

[67] Klaus Greff et al. “LSTM: A search space odyssey”. In: *IEEE transactions on neural networks and learning systems* 28.10 (2016), pp. 2222–2232.

[68] J. Duchi, E. Hazan, and Y. Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research*. Vol. 12. 2011, pp. 2121–2159.

[69] T. Tieleman and G. Hinton. “Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural Networks for Machine Learning*. 2012.

[70] Andrew Y. Ng. “Feature selection, L1 vs. L2 regularization, and rotational invariance”. In: *Proceedings of the twenty-first international conference on Machine learning (ICML)*. 2004, p. 78.

[71] Arthur E. Hoerl and Robert W. Kennard. “Ridge Regression: Biased Estimation for Nonorthogonal Problems”. In: *Technometrics* 12.1 (1970), pp. 55–67.

[72] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58.1 (1996), pp. 267–288.

[73] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[74] Ming Yuan and Yi Lin. “Model selection and estimation in regression with grouped variables”. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 68.1 (2006), pp. 49–67.

[75] Gareth James. *An introduction to statistical learning*. 2013.

[76] Lutz Prechelt. “Early Stopping - But When?” In: *Neural Networks: Tricks of the Trade*. Ed. by G. B. Orr and K. R. Müller. Springer, 1998, pp. 55–69.

[77] 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*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[78] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034.

[79] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. "A simple way to initialize recurrent networks of rectified linear units". In: *arXiv preprint arXiv:1504.00941* (2015).

[80] Yann LeCun et al. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 2002, pp. 9–50.

[81] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2 (2012).

[82] James Bergstra et al. "Hyperopt: a python library for model selection and hyperparameter optimization". In: *Computational Science & Discovery* 8.1 (2015), p. 014008.

[83] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems* 25 (2012).

[84] Weike Sun and Richard D Braatz. "Smart process analytics for predictive modeling". In: *Computers & Chemical Engineering* 144 (2021), p. 107134.

[85] Mervyn Stone. "Cross-validatory choice and assessment of statistical predictions". In: *Journal of the royal statistical society: Series B (Methodological)* 36.2 (1974), pp. 111–133.

[86] Ron Kohavi et al. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *Ijcai*. Vol. 14. 2. Montreal, Canada. 1995, pp. 1137–1145.

[87] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[88] Matthew McDermott et al. "A closer look at auroc and auprc under class imbalance". In: *arXiv preprint arXiv:2401.06091* (2024).

[89] Yehuda Kleiner and Balvant Rajani. "Comprehensive review of structural deterioration of water mains: statistical models". In: *Urban water* 3.3 (2001), pp. 131–150.

[90] Robert I McDonald et al. “Water on an urban planet: Urbanization and the reach of urban water infrastructure”. In: *Global environmental change* 27 (2014), pp. 96–105.

[91] Roland Liemberger and Alan Wyatt. “Quantifying the global non-revenue water problem”. In: *Water Supply* 19.3 (2019), pp. 831–837.

[92] ISTAT. *LE STATISTICHE DELL'ISTAT SULL'ACQUA — ANNI 2020-2023*. Tech. rep. Istituto Nazionale di Statistica, March 2024.

[93] Jinghui Xu et al. “Low-cost, tiny-sized MEMS hydrophone sensor for water pipeline leak detection”. In: *IEEE Transactions on Industrial Electronics* 66.8 (2018), pp. 6374–6382.

[94] Marie-Claude Besner, Michèle Prévost, and Stig Regli. “Assessing the public health risk of microbial intrusion events in distribution systems: Conceptual model, available data, and challenges”. In: *Water research* 45.3 (2011), pp. 961–979.

[95] Mohammad R Karim, Morteza Abbaszadegan, and Mark LeChevallier. “Potential for pathogen intrusion during pressure transients”. In: *Journal-American Water Works Association* 95.5 (2003), pp. 134–146.

[96] Andrew F Colombo, Pedro Lee, and Bryan W Karney. “A selective literature review of transient-based leak detection methods”. In: *Journal of hydro-environment research* 2.4 (2009), pp. 212–227.

[97] Raido Puust et al. “A review of methods for leakage management in pipe networks”. In: *Urban Water Journal* 7.1 (2010), pp. 25–45.

[98] Patrick Thomson et al. “Water Supply Interruptions Are Associated with More Frequent Stressful Behaviors and Emotions but Mitigated by Predictability: A Multisite Study”. In: *Environmental Science & Technology* 58.16 (2024), pp. 7010–7019.

[99] David Roibas, Maria A Garcia-Valiñas, and Roberto Fernandez-Llera. “Measuring the impact of water supply interruptions on household welfare”. In: *Environmental and Resource Economics* 73 (2019), pp. 159–179.

[100] Chan-Wook Lee and Do-Guen Yoo. “Development of leakage detection model and its application for water distribution networks using RNN-LSTM”. In: *Sustainability* 13.16 (2021), p. 9262.

[101] M Bakker et al. “Heuristic burst detection method using flow and pressure measurements”. In: *Journal of Hydroinformatics* 16.5 (2014), pp. 1194–1209.

[102] Lina Sela Perelman et al. “Sensor placement for fault location identification in water networks: A minimum test cover approach”. In: *Automatica* 72 (2016), pp. 166–176.

[103] Hoese Michel Tornyeviadzi and Razak Seidu. “Leakage detection in water distribution networks via 1D CNN deep autoencoder for multivariate SCADA data”. In: *Engineering Applications of Artificial Intelligence* 122 (2023), p. 106062.

[104] Ulrike Von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and computing* 17 (2007), pp. 395–416.

[105] Armando Di Nardo et al. “Applications of graph spectral techniques to water distribution network management”. In: *Water* 10.1 (2018), p. 45.

[106] Joanna A Gutiérrez-Pérez et al. “Application of graph-spectral methods in the vulnerability assessment of water supply networks”. In: *Mathematical and Computer Modelling* 57.7-8 (2013), pp. 1853–1859.

[107] J Macqueen. *Some methods for classification and analysis of multivariate observations*. University of California Press, 1967.

[108] Carlo Giudicianni et al. “Topological taxonomy of water distribution networks”. In: *Water* 10.4 (2018), p. 444.

[109] C Giudicianni et al. “Topological placement of quality sensors in water-distribution networks without the recourse to hydraulic modeling”. In: *Journal of Water Resources Planning and Management* 146.6 (2020), p. 04020030.

[110] Lewis A Rossman et al. “EPANET 2: users manual”. In: (2000).

[111] Wang Lijuan, Zhang Hongwei, and Jia Hui. “A leak detection method based on EPANET and genetic algorithm in water distribution systems”. In: *Software Engineering and Knowledge Engineering: Theory and Practice: Volume 1*. Springer. 2012, pp. 459–465.

[112] Y Shastri and Urmila Diwekar. “Sensor placement in water networks: A stochastic programming approach”. In: *Journal of water resources planning and management* 132.3 (2006), pp. 192–203.

[113] Rasooli Ahmadullah and Kang Dongshik. “Designing of hydraulically balanced water distribution network based on GIS and EPANET”. In: *International Journal of Advanced Computer Science and Applications* 7.2 (2016).

[114] Erika Yesenia Avila-Melgar, Marco Antonio Cruz-Chávez, and Beatriz Martinez-Bahena. "General methodology for using Epanet as an optimization element in evolutionary algorithms in a grid computing environment for water distribution network design". In: *Water Science and Technology: Water Supply* 17.1 (2017), pp. 39–51.

[115] Amirabbas Mottahedin et al. "Efficient Identification Strategy of Isolation Valves to Maintain through Modularity-Based WDN Clustering". In: *Engineering Proceedings* 69.1 (2024), p. 44.

[116] Walter M Grayman, Regan Murray, and Dragan A Savic. "Effects of redesign of water systems for security and water quality factors". In: *World Environmental and Water Resources Congress 2009: Great Rivers*. 2009, pp. 1–11.

[117] Lina Perelman and Avi Ostfeld. "Water-distribution systems simplifications through clustering". In: *Journal of Water Resources Planning and Management* 138.3 (2012), pp. 218–229.

[118] Carlo Giudicianni and Enrico Creaco. "Exploring the Impact of Pulsed Demand Model on the Quality Sensor Placement in Water Distribution Networks". In: *Engineering Proceedings* 69.1 (2024), p. 174.

[119] Demetrios G Eliades et al. "EPANET-MATLAB toolkit: An open-source software for interfacing EPANET with MATLAB". In: *Proc. 14th international conference on computing and control for the water industry (ccwi)*. Vol. 8. The Netherlands. 2016.

[120] Lewis A Rossman et al. *EPANET 2. 2 User Manual (EPA/600/R-20/133)*. 2020.

[121] Weike Sun and Richard D Braatz. "Smart process analytics for predictive modeling". In: *Computers & Chemical Engineering* 144 (2021), p. 107134.

[122] Margherita Grandini, Enrico Bagli, and Giorgio Visani. "Metrics for multi-class classification: an overview". In: *arXiv preprint arXiv:2008.05756* (2020).

[123] A Di Nardo et al. "Simplified approach to water distribution system management via identification of a primary network". In: *Journal of Water Resources Planning and Management* 144.2 (2018), p. 04017089.

[124] Carlo Giudicianni et al. "Multi-criteria method for the realistic placement of water quality sensors on pipes of water distribution systems". In: *Environmental Modelling & Software* 152 (2022), p. 105405.

[125] Yipeng Wu et al. “Burst detection in district metering areas using a data driven clustering algorithm”. In: *Water research* 100 (2016), pp. 28–37.

[126] Stephen R Mounce, Richard B Mounce, and Joby B Boxall. “Novelty detection for time series data analysis in water distribution systems using support vector machines”. In: *Journal of hydroinformatics* 13.4 (2011), pp. 672–686.

[127] Guoliang Ye and Richard Andrew Fenner. “Kalman filtering of hydraulic measurements for burst detection in water distribution systems”. In: *Journal of pipeline systems engineering and practice* 2.1 (2011), pp. 14–22.

[128] Giacomo Galuppini, Enrico Creaco, and Lalo Magni. “Sum-of-delay models for pressure control in Water Distribution Networks”. In: *Control Engineering Practice* 113 (2021), p. 104844.

[129] Ye Wang, Vicenç Puig, and Gabriela Cembrano. “Non-linear economic model predictive control of water distribution networks”. In: *Journal of Process Control* 56 (2017), pp. 23–34.

[130] David JC MacKay. “Bayesian interpolation”. In: *Neural computation* 4.3 (1992), pp. 415–447.

[131] Eyke Hüllermeier and Willem Waegeman. “Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods”. In: *Machine learning* 110.3 (2021), pp. 457–506.

[132] Armen Der Kiureghian and Ove Ditlevsen. “Aleatory or epistemic? Does it matter?” In: *Structural safety* 31.2 (2009), pp. 105–112.

[133] Sam Patterson and Yee Whye Teh. “Stochastic gradient Riemannian Langevin dynamics on the probability simplex”. In: *Advances in neural information processing systems* 26 (2013).

[134] Chuan Guo et al. “On calibration of modern neural networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 1321–1330.

[135] Agnieszka Mikołajczyk and Michał Grochowski. “Data augmentation for improving deep learning in image classification problem”. In: *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE. 2018, pp. 117–122.

[136] Luis Perez and Jason Wang. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. 2017. arXiv: 1712.04621 [cs.CV]. URL: https://arxiv.org/abs/1712.04621.

[137] Qingsong Wen et al. “Time series data augmentation for deep learning: A survey”. In: *arXiv preprint arXiv:2002.12478* (2020).

[138] Dario Piga et al. “Synthetic data generation for system identification: leveraging knowledge transfer from similar systems”. In: *arXiv preprint arXiv:2403.05164* (2024).

[139] Yaniv Ovadia et al. “Can you trust your model’s uncertainty? evaluating predictive uncertainty under dataset shift”. In: *Advances in neural information processing systems* 32 (2019).

[140] Dan Hendrycks and Kevin Gimpel. “A baseline for detecting misclassified and out-of-distribution examples in neural networks”. In: *arXiv preprint arXiv:1610.02136* (2016).

[141] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. “Enhancing the reliability of out-of-distribution image detection in neural networks”. In: *arXiv preprint arXiv:1706.02690* (2017).

[142] Andrey Malinin and Mark Gales. “Predictive uncertainty estimation via prior networks”. In: *Advances in neural information processing systems* 31 (2018).

[143] Jinsol Lee and Ghassan AlRegib. “Gradients as a measure of uncertainty in neural networks”. In: *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2020, pp. 2416–2420.

[144] Maithra Raghu et al. “Direct uncertainty prediction for medical second opinions”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 5281–5290.

[145] Yue Cao et al. “Ensemble deep learning in bioinformatics”. In: *Nature Machine Intelligence* 2.9 (2020), pp. 500–508.

[146] Joao Mendes-Moreira et al. “Ensemble approaches for regression: A survey”. In: *Acm computing surveys (csur)* 45.1 (2012), pp. 1–40.

[147] Kazi Md Rokibul Alam, Nazmul Siddique, and Hojjat Adeli. “A dynamic ensemble learning algorithm for neural networks”. In: *Neural Computing and Applications* 32.12 (2020), pp. 8675–8690.

[148] Wendy S Parker. “Ensemble modeling, uncertainty and robust predictions”. In: *Wiley interdisciplinary reviews: Climate change* 4.3 (2013), pp. 213–223.

[149] Murat Seckin Ayhan and Philipp Berens. “Test-time data augmentation for estimation of heteroscedastic aleatoric uncertainty in deep neural networks”. In: *Medical Imaging with Deep Learning*. 2018.

[150] Divya Shanmugam et al. “Better aggregation in test-time augmentation”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 1214–1223.

[151] Guotai Wang et al. “Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks”. In: *Neurocomputing* 338 (2019), pp. 34–45.

[152] Tishby and Solla. “Consistent inference of probabilities in layered networks: predictions and generalizations”. In: *International 1989 joint conference on neural networks*. IEEE. 1989, pp. 403–409.

[153] Wray L Buntine. “Bayesian backpropagation”. In: *Complex systems* 5 (1991), pp. 603–643.

[154] Cuong V Nguyen et al. “Variational continual learning”. In: *arXiv preprint arXiv:1710.10628* (2017).

[155] Andrew Gelman et al. *Bayesian data analysis*. Chapman and Hall/CRC, 1995.

[156] Geoffrey E Hinton and Drew Van Camp. “Keeping the neural networks simple by minimizing the description length of the weights”. In: *Proceedings of the sixth annual conference on Computational learning theory*. 1993, pp. 5–13.

[157] David Barber and Christopher M Bishop. “Ensemble learning in Bayesian neural networks”. In: *Nato ASI Series F Computer and Systems Sciences* 168 (1998), pp. 215–238.

[158] Radford Neal. “Bayesian learning via stochastic dynamics”. In: *Advances in neural information processing systems* 5 (1992).

[159] Kumar Avinava Dubey et al. “Variance reduction in stochastic gradient Langevin dynamics”. In: *Advances in neural information processing systems* 29 (2016).

[160] Yingzhen Li and Yarin Gal. “Dropout inference in bayesian neural networks with alpha-divergences”. In: *International conference on machine learning*. PMLR. 2017, pp. 2052–2061.

[161] DW Clarke and Riccardo Scattolini. “Constrained receding-horizon predictive control”. In: *IEE proceedings D (control theory and applications)*. Vol. 138. 4. IET. 1991, pp. 347–354.

[162] David Q Mayne and Hannah Michalska. “Receding horizon control of nonlinear systems”. In: *Proceedings of the 27th IEEE Conference on Decision and Control*. IEEE. 1988, pp. 464–465.

[163] Manfred Morari and Jay H Lee. "Model predictive control: past, present and future". In: *Computers & chemical engineering* 23.4-5 (1999), pp. 667–682.

[164] Lalo Magni et al. *Model predictive control of type 1 diabetes: an in silico trial*. 2007.

[165] Paola Soru et al. "MPC based artificial pancreas: strategies for individualization and meal compensation". In: *Annual Reviews in Control* 36.1 (2012), pp. 118–128.

[166] Stefano Di Cairano et al. "An MPC design flow for automotive control and applications to idle speed regulation". In: *2008 47th IEEE Conference on Decision and Control*. IEEE. 2008, pp. 5686–5691.

[167] Giacomo Galuppini, Lalo Magni, and Davide Martino Raimondo. "Model predictive control of systems with deadzone and saturation". In: *Control Engineering Practice* 78 (2018), pp. 56–64.

[168] Kenneth R Muske and James B Rawlings. "Model predictive control with linear models". In: *AIChE Journal* 39.2 (1993), pp. 262–287.

[169] Pierre OM Scokaert and James B Rawlings. "Feasibility issues in linear model predictive control". In: *AIChE Journal* 45.8 (1999), pp. 1649–1659.

[170] G De Nicolao, L Magni, and Riccardo Scattolini. "Stability and robustness of nonlinear receding horizon control". In: *Nonlinear model predictive control*. Springer, 2000, pp. 3–22.

[171] Frank Allgöwer et al. "Nonlinear predictive control and moving horizon estimation—an introductory overview". In: *Advances in control: Highlights of ECC'99* (1999), pp. 391–449.

[172] David Q Mayne et al. "Constrained model predictive control: Stability and optimality". In: *Automatica* 36.6 (2000), pp. 789–814.

[173] Daniel Limon et al. "Input-to-state stability: a unifying framework for robust model predictive control". In: *Nonlinear Model Predictive Control: Towards New Challenging Applications* (2009), pp. 1–26.

[174] Lalo Magni, Davide Martino Raimondo, and Frank Allgöwer. "Nonlinear model predictive control". In: *Lecture Notes in Control and Information Sciences* 384 (2009).

[175] Frank Allgöwer and Alex Zheng. *Nonlinear model predictive control*. Vol. 26. Birkhäuser, 2012.

[176] S Joe Qin and Thomas A Badgwell. "An overview of nonlinear model predictive control applications". In: *Nonlinear model predictive control* (2000), pp. 369–392.

[177] Frank Allgower, Rolf Findeisen, Zoltan K Nagy, et al. "Nonlinear model predictive control: From theory to application". In: *Journal-Chinese Institute Of Chemical Engineers* 35.3 (2004), pp. 299–316.

[178] Johannes Köhler et al. "A computationally efficient robust model predictive control framework for uncertain nonlinear systems". In: *IEEE Transactions on Automatic Control* 66.2 (2020), pp. 794–801.

[179] Alberto Bemporad and Manfred Morari. "Robust model predictive control: A survey". In: *Robustness in identification and control*. Springer, 2007, pp. 207–226.

[180] Davide Martino Raimondo et al. "Min-max model predictive control of nonlinear systems: A unifying overview on stability". In: *European Journal of Control* 15.1 (2009), pp. 5–21.

[181] Wilbur Langson et al. "Robust model predictive control using tubes". In: *Automatica* 40.1 (2004), pp. 125–133.

[182] Ali Mesbah et al. "Stochastic nonlinear model predictive control with probabilistic constraints". In: *2014 American control conference*. IEEE. 2014, pp. 2413–2419.

[183] Marcello Farina, Luca Giulioni, and Riccardo Scattolini. "Stochastic linear model predictive control with chance constraints–a review". In: *Journal of Process Control* 44 (2016), pp. 53–67.

[184] Pu Li, Moritz Wendt, and Günter Wozny. "A probabilistically constrained model predictive controller". In: *Automatica* 38.7 (2002), pp. 1171–1176.

[185] Joel A Paulson et al. "Stochastic model predictive control with joint chance constraints". In: *International Journal of Control* 93.1 (2020), pp. 126–139.

[186] Joel A Paulson et al. "Nonlinear model predictive control of systems with probabilistic time-invariant uncertainties". In: *IFAC-PapersOnLine* 48.23 (2015), pp. 16–25.

[187] Robert D McAllister and James B Rawlings. "Nonlinear stochastic model predictive control: Existence, measurability, and stochastic asymptotic stability". In: *IEEE Transactions on Automatic Control* 68.3 (2022), pp. 1524–1536.

[188] Edward A Buehler, Joel A Paulson, and Ali Mesbah. "Lyapunov-based stochastic nonlinear model predictive control: Shaping the state probability distribution functions". In: *2016 American control conference (ACC)*. IEEE. 2016, pp. 5389–5394.

[189] Tim Brüdigam et al. "Stochastic model predictive control with a safety guarantee for automated driving". In: *IEEE Transactions on Intelligent Vehicles* 8.1 (2021), pp. 22–36.

[190] Kiet Tuan Hoang et al. "Probabilistic forecasting-based stochastic nonlinear model predictive control for power systems with intermittent renewables and energy storage". In: *IEEE Transactions on Power Systems* (2023).

[191] Florian Weissel, Marco F Huber, and Uwe D Hanebeck. "Stochastic nonlinear model predictive control based on Gaussian mixture approximations". In: *Informatics in Control, Automation and Robotics: Selected Papers from the International Conference on Informatics in Control, Automation and Robotics 2007*. Springer. 2009, pp. 239–252.

[192] Zhong-Sheng Hou and Zhuo Wang. "From model-based control to data-driven control: Survey, classification and perspective". In: *Information Sciences* 235 (2013), pp. 3–35.

[193] Lukas Hewing et al. "Learning-based model predictive control: Toward safe learning in control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1 (2020), pp. 269–296.

[194] Lukas Hewing, Juraj Kabzan, and Melanie N Zeilinger. "Cautious model predictive control using gaussian process regression". In: *IEEE Transactions on Control Systems Technology* 28.6 (2019), pp. 2736–2743.

[195] Stephen Piche et al. "Nonlinear model predictive control using neural networks". In: *IEEE Control Systems Magazine* 20.3 (2000), pp. 53–62.

[196] Marco Polver et al. "Artificial Pancreas under a Zone Model Predictive Control based on Gaussian Process models: toward the personalization of the closed loop". In: *IFAC-PapersOnLine* 56.2 (2023), pp. 9642–9647.

[197] Anil Aswani et al. "Provably safe and robust learning-based model predictive control". In: *Automatica* 49.5 (2013), pp. 1216–1226.

[198] Julian Berberich et al. "Data-driven model predictive control with stability and robustness guarantees". In: *IEEE Transactions on Automatic Control* 66.4 (2020), pp. 1702–1717.

[199] Marco Forgione and Dario Piga. “Continuous-time system identification with neural networks: Model structures and fitting criteria”. In: *European Journal of Control* 59 (2021), pp. 69–81.

[200] Jorge Lo Presti, Lalo Magni, and Chiara Toffanin. “Ship Manoeuvering Modelling with a Physics-Oriented Neural Network-Based Approach”. In: *IFAC-PapersOnLine* 56.2 (2023), pp. 3471–3476.

[201] Daniele Masti and Alberto Bemporad. “Learning nonlinear state–space models using autoencoders”. In: *Automatica* 129 (2021), p. 109666.

[202] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[203] Joel AE Andersson et al. “CasADi: a software framework for nonlinear optimization and optimal control”. In: *Mathematical Programming Computation* 11 (2019), pp. 1–36.

[204] Atsushi Ogawa and H Kasai. “On the mathematical model of manoeuvring motion of ships”. In: *International shipbuilding progress* 25.292 (1978), pp. 306–319.

[205] Hironori Yasukawa and Yasuo Yoshimura. “Introduction of MMG standard method for ship maneuvering predictions”. In: *Journal of marine science and technology* 20 (2015), pp. 37–52.

[206] Adam Paszke et al. “Automatic differentiation in pytorch”. In: (2017).

[207] Zhen Li and Jing Sun. “Disturbance compensating model predictive control with application to ship heading control”. In: *IEEE transactions on control systems technology* 20.1 (2011), pp. 257–265.

[208] Yang Liu et al. “Adaptive auto-berthing control of underactuated vessel based on barrier Lyapunov function”. In: *Journal of Marine Science and Engineering* 10.2 (2022), p. 279.