UNIVERSITY OF CAGLIARI

FACULTY OF ENGINEERING AND ARCHITECTURE

MASTER'S DEGREE IN COMPUTER ENGINEERING, CYBERSECURITY AND ARTIFICIAL INTELLIGENCE

Adversarial Learning Attacks on Autoencoder-based models for Control Systems

Supervisor:
Prof. Mauro Franceschelli

Candidate:
Marco Ledda
matr. 70/90/00471

Academic Year 2023/2024

UNIVERSITÀ DEGLI STUDI DI CAGLIARI

FACOLTÀ DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA MAGISTRALE IN COMPUTER ENGINEERING, CYBERSECURITY AND ARTIFICIAL INTELLIGENCE

Adversarial Learning Attacks on Autoencoder-based models for Control Systems

Relatore:
Prof. Mauro Franceschelli

Candidato:
Marco Ledda
matr. 70/90/00471

Anno Accademico 2023/2024

Ringraziamenti

Concludere questo percorso di studi rappresenta per me un grande traguardo, raggiunto grazie al supporto di tante persone che mi sono state vicine in momenti importanti.

Un sentito grazie al Prof. Mauro Franceschelli, che mi ha guidato in questo ultimo anno con competenza e professionalità nello svolgere un lavoro di ricerca in maniera rigorosa e con spirito critico.

Ringrazio i miei genitori, che hanno sempre creduto in me e mi hanno dato la forza necessaria per affrontare ogni sfida, piccola o grande. È grazie al loro esempio che ho imparato il valore del sacrificio e della dedizione.

Ringrazio tutta la mia famiglia, in primis i miei nonni Serafino e Marinella che mi hanno insegnato a guardare la vita da diverse prospettive, e ad affrontarla con coraggio e umiltà.

Ringrazio tutti i miei amici, in particolare Davide e Andrea con cui ho condiviso momenti di leggerezza e risate che mi hanno permesso di ritrovare l'energia nei momenti più intensi e impegnativi.

Vorrei inoltre ringraziare i miei colleghi Christian, Luca e Nicola, con i quali ho condiviso risate, fatiche, successi e momenti di sconforto. Insieme abbiamo affrontato esami, giornate di studio e progetti, sempre con uno spirito di collaborazione e amicizia.

Il grazie più grande va a Jessica, che in questi anni è stata la mia motivazione più grande, mi è sempre stata vicina e mi ha sempre sostenuto anche quando non era facile.

Un pensiero per Stefania, che ci ha lasciati quando ho iniziato questo percorso. Non ti dimenticheremo mai.

In conclusione, desidero condividere una citazione di Davide Nicola che racchiude l'essenza di ciò che ho imparato lungo la strada:

"Io continuo a dire che io non motivo nessuno, perché nessuno può motivare me. Non c'è nessuno che può motivare me, io sono automotivato. Non ho bisogno di una persona che mi motiva. Forse il problema è proprio questo, noi pensiamo che un'altra persona debba motivare noi. In realtà io credo che una persona debba sapere cosa deve fare per poter migliorare se stesso o raggiungere un obiettivo. Molto spesso noi abbiamo bisogno degli altri proprio quando non sappiamo come fare. Ma una volta che sai come fare, devi far da solo sennò è troppo facile, no? È il concetto di carisma che controlla e diventi schiavo. No no, io voglio imparare per camminare con le mie gambe."

Abstract

Autoencoders have been proposed for system identification, offering a flexible alternative to traditional methods by using past input and output data to represent system dynamics without requiring deep knowledge of the system. While this approach simplifies the identification process, it raises concerns about the robustness and reliability of such networks. Although adversarial vulnerabilities are well-known in areas like computer vision, they remain relatively unexplored in the context of control systems. This thesis explores the vulnerabilities of autoencoder-based models in control systems to adversarial attacks. It introduces new attack strategies targeting the bridge network and information vector of the autoencoder, critical components in the state-space identification process. Through an extensive experimental analysis of benchmark control systems, the study demonstrates how these attacks can degrade system performance, emphasizing the need to find formal guarantees for machine-learning-based control systems. The results offer critical insights into the security and reliability of these systems, highlighting the importance of adversarial robustness for future research and practical applications.

Abstract

Per l'identificazione dei sistemi sono stati proposti gli autoencoder, che offrono un'alternativa flessibile ai metodi tradizionali, utilizzando i dati passati di ingresso e uscita per rappresentare la dinamica del sistema senza richiedere una conoscenza approfondita del sistema stesso. Se da un lato questo approccio semplifica il processo di identificazione, dall'altro solleva preoccupazioni circa la robustezza e l'affidabilità di tali reti. Sebbene tali vulnerabilità siano ben note in settori come la computer vision, rimangono relativamente inesplorate nel contesto dei sistemi di controllo. Questa tesi esplora le vulnerabilità dei modelli basati su autoencoder nei sistemi di controllo. Introduce nuove strategie di attacco che prendono di mira i componenti critici della struttura di apprendimento durante il processo di identificazione. Attraverso un'ampia analisi sperimentale su sistemi di controllo ben noti alla comunità scientifica, lo studio dimostra come questi attacchi possano degradare le prestazioni del sistema, sottolineando la necessità di trovare garanzie formali per i sistemi di controllo basati sul machine learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the field of systems and control, the integration of Machine Learning techniques has opened new avenues for improving system robustness and performance. The complex nature of modern control environments often involves nonlinear dynamics for which it is difficult to have an accurate model on which we can rely. Machine Learning, particularly in Deep Learning, has demonstrated significant potential in system identification problems where abundant input/output data is available, thus enabling advanced optimization and control strategies. However, the fundamental issue is the development of formal guarantees on the reliability and robustness of these methods. This thesis aims to explore adversarial learning within the context of control systems, specifically focusing on the vulnerabilities of autoencoder-based models.

1.1 Main Contributions

This thesis contributes to the field of control and systems by exploring the application of adversarial learning to autoencoder-based models, a domain that has previously focused primarily on image recognition and computer vision. While adversarial learning has been investigated in those fields, its implications and challenges in control system theory remain largely unexplored. This research addresses this gap by formulating and investigating adversarial attacks to exploit vulnerabilities in autoencoder-based models used within control systems.

The primary objectives of this thesis include developing a deeper understanding of how adversarial learning can be applied to control systems, designing novel attack strategies that specifically target the unique characteristics of autoencoder-based models, and exploring the potential impact of these attacks on control system performance. Through an experimental analysis, this study aims to highlight the need for greater attention to adversarial robustness in developing and deploying machine learning models in control systems. By extending adversarial learning into this new domain, the thesis offers critical insights that could influence future research and practice in ensuring the security and reliability of control systems.

1.2 Outline of the Thesis

The current work is organized as follows: Chapter 2 provides the necessary background on machine learning, including an overview of neural networks, optimization methods, and dimensionality reduction techniques. In Chapter 3 we show machine learning techniques' integration into control systems, particularly autoencoders. This chapter discusses traditional system identification methods and their limitations, followed by presenting a novel approach based on autoencoders for nonlinear system identification. Chapter 4 discusses adversarial learning, highlighting its relevance and application in control systems. The chapter details the formulation of the proposed adversarial attacks against autoencoder-based models. Chapter 5 shows the experimental results, showcasing the impact of the adversarial attacks on the chosen benchmark systems. Finally, in Chapter 6 there will be a summary of the findings and a discussion of the limitations of the work, including suggestions for future research directions.

Chapter 2

Machine Learning Background

2.1 Fundamentals of Machine Learning

Machine Learning refers to an ensemble of techniques designed to empower computer systems with the ability to learn from data, enabling them to perform specific tasks without being explicitly programmed for those tasks. These tasks are varied, including areas such as *classification*, *regression*, *clustering*, and *dimensionality reduction* problems. The learning process starts with the acquisition of data, which is typically represented by a *feature vector*. This feature vector is a structured representation that includes specific discriminative characteristics or attributes of the data, allowing the machine learning model to identify patterns and relationships. The core of machine learning lies in the algorithms that learn from the collected data, often referred to as the *training set*. This training set is a critical component, as it provides the foundation from which the algorithms can learn and make predictions or decisions. There are different types of learning paradigms within machine learning, primarily categorized into supervised learning and unsupervised learning.

In *supervised learning*, the algorithms are trained using labeled data. This means that each training example is paired with an output label, providing the algorithm with explicit guidance on what the expected outcome should be. This allows the model to learn the mapping from input features to the target output, making it particularly effective for tasks where the desired results are known and can be used to guide the learning process. On the other hand, *unsupervised learning* deals with unlabeled data. In this paradigm, the algorithms must infer the underlying structure of the data without explicit guidance on what the output should be. This is often used for tasks such as *clustering*, where the goal is to group similar data points, or for dimensionality reduction, where the aim is to simplify the data by reducing the number of features while retaining essential information.

2.2 Neural Networks

In Machine Learning, an artificial neural network is a computational model made up of artificial neurons, loosely inspired by the structure of biological neural networks. These artificial neurons are interconnected, with each connection assigned a specific weight. A neural network is organized into layers: an input layer, where nodes receive external data, and an output layer, where nodes deliver the processing results. The layers in between, known as hidden layers, handle the internal processing (Fig. 2.1). When every neuron in a layer is connected to every neuron in the next layer, the network is described as fully connected.

Mathematically, artificial neural networks represent a class of nonlinear functions that map input $\boldsymbol{x}$ to a predicted output $\hat{y}$ [15]. Neural networks can have multiple layers and are viewed as compositions of functions. A neural network with L layers is defined by the following composition map:

$$f = f^L \circ ... \circ f^2 \circ f^1 \tag{2.1}$$

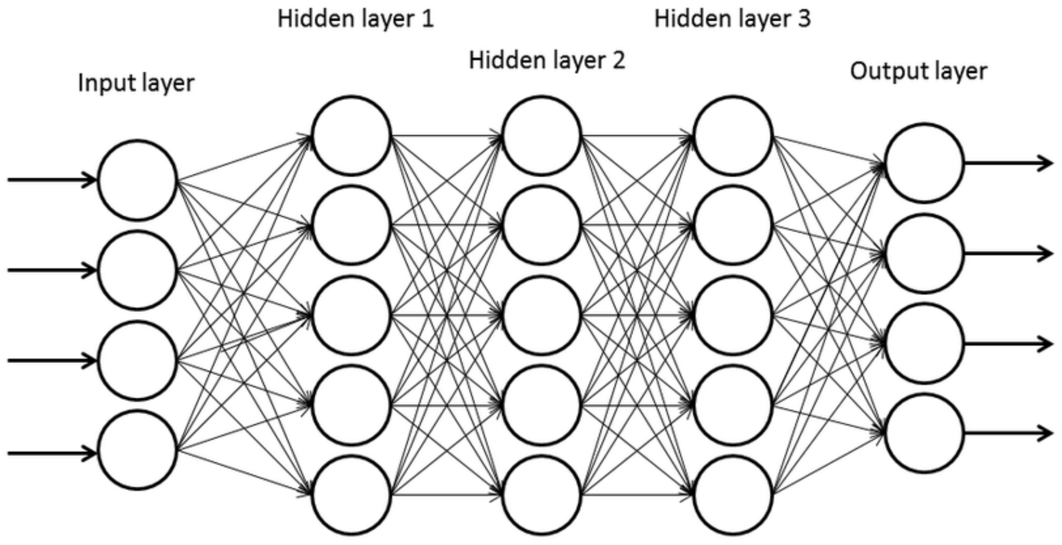


Figure 2.1: Graphical representation of an artificial neural network

where $f^L : \mathbb{R}^{n_L} \to \mathbb{R}^{n_{out}}$ is called the output layer and $f^{L-1}, ..., f^1$ are called intermediate layers, $f^\ell : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_{\ell+1}}$. The dimension of the ℓ^{th} hidden layer, representing the number of nodes or neurons in that layer, is denoted as n_ℓ, and the dimension of the first layer is $n_1 = d$. For instance, if we refer to a 3-layer network, the composition map will be:

$$f(x) = f_3(f_2(f_1(\boldsymbol{x}))) \tag{2.2}$$

Fully connected neural networks were some of the earliest architectures proposed. A fully connected neural network with a single hidden layer is characterized by:

$$\hat{y}(t) = W_2\sigma(W_1x(t) + b_1) + b_2 \tag{2.3}$$

and could be decomposed as $f = f^2 \circ f^1$ which are the layers of the neural network [15]. In fully connected networks, layers are affine transformations followed

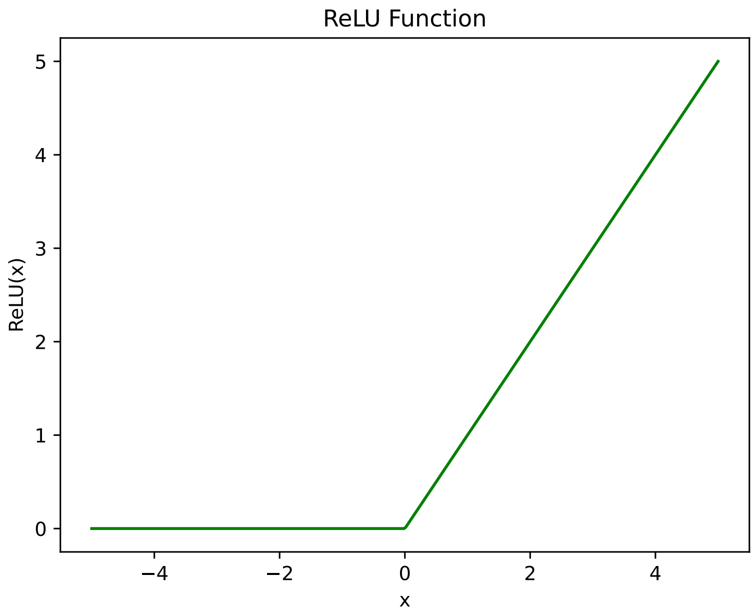


Figure 2.2: Rectified Linear Unit (ReLU) activation function.

by nonlinear transformations:

$$f^{\ell}(z) = \sigma(W_{\ell} z + b_{\ell}) \tag{2.4}$$

where $W_{\ell} \in \mathbb{R}^{n_{\ell+1} \times n_{\ell}}$ are weight matrices, $b_{\ell} \in \mathbb{R}^{n_{\ell+1}}$ are bias vectors and σ is an activation function, which is a nonlinear function that act element-wise on its inputs [15]. The unknown parameters of the neural network are the weight matrices W_i and the biases b_i, for $i = 1, 2, ..., L$. In the literature, several activation functions are employed such as the so-called ReLu (Rectified Linear Unit) $\sigma(x) = \max(x, 0)$ (Fig. 2.2), the hyperbolic tangent $\sigma(x) = \tanh(x)$ (Fig. 2.3) and the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ (Fig. 2.4).

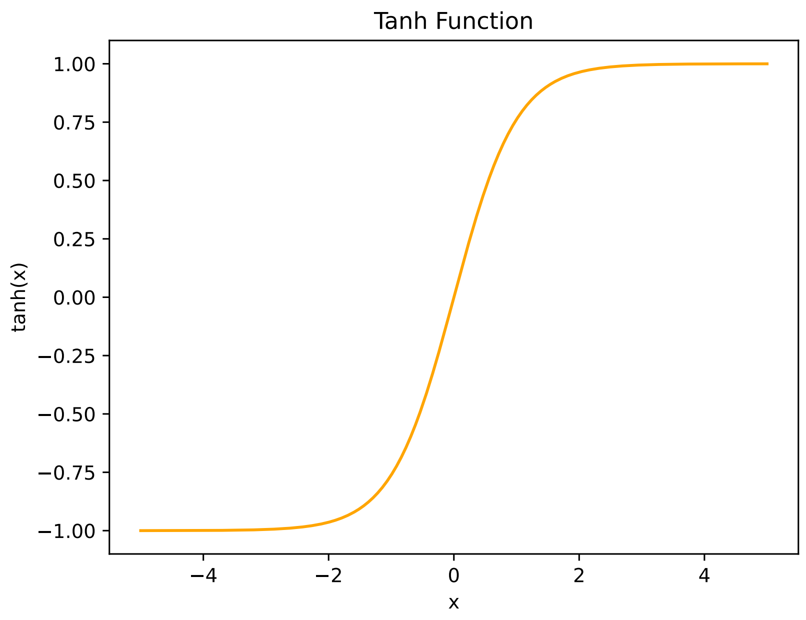


Figure 2.3: Hyperbolic tangent activation function.

2.3 Optimization methods for training Neural Networks

The training process of a neural network focuses on minimizing the loss function $\mathcal{L}$ by adjusting the values of connection weights to produce the desired output. Since the loss function can be very complex, due to the non-convex nature of the loss landscape [12], efficient learning algorithms are employed, namely *backpropagation algorithms*. These algorithms take advantage of the continuous and differentiable nature of neuron activation function, making the network output a continuous and differentiable function of its input. By leveraging the chain rule, these algorithms compute derivatives essential for optimization. The chain rule, applied to the equation above 2.2, helps in computing these derivatives:

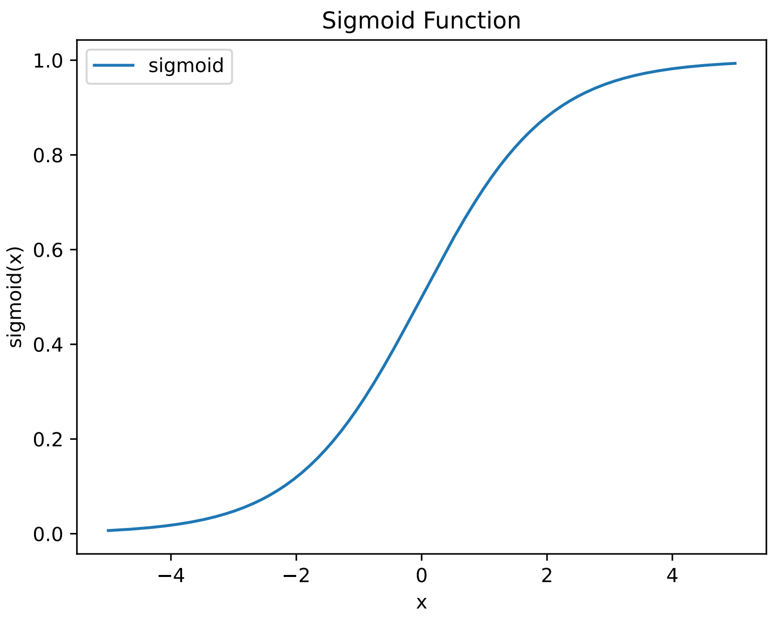


Figure 2.4: Sigmoid activation function.

$$\frac{\partial f}{\partial \boldsymbol{x}} = \frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial f_1}\frac{\partial f_1}{\partial \boldsymbol{x}} \tag{2.5}$$

This formula demonstrates how we can compute the derivatives needed for backpropagation. In neural network training, this rule is applied to a loss function that depends on the network's output. The network's output, in turn, depends on both the input and the weights of each layer. By computing the gradient of the loss function with respect to the weights, we can adjust the weights to minimize the loss function. This gradient descent procedure is iterative: we forward propagate the training data through the network to compute the output, then update the weights based on the computed gradients, and repeat the process. Let's assume that the loss function has the behavior depicted in Fig. 2.5. We need to modify the values w_1 and w_2 so that the initial randomly chosen weights, represented as a black dot, are adjusted following the direction where the descent of the objective function's

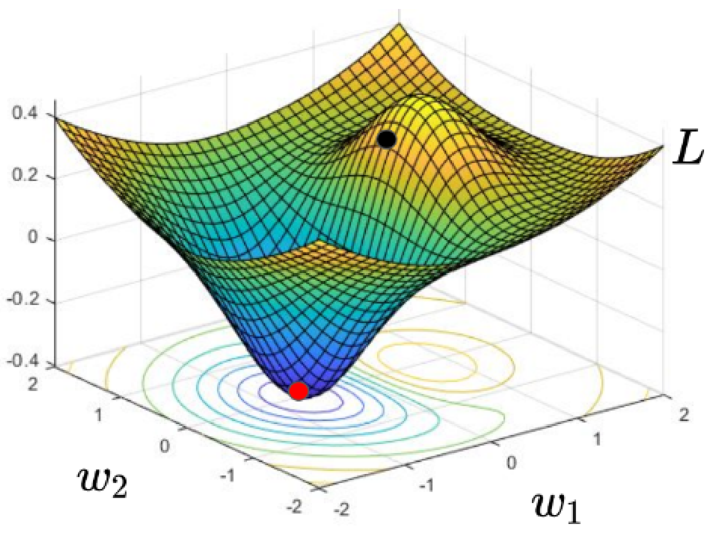


Figure 2.5: Loss function in the feature space.

Algorithm 1 Gradient Descent Algorithm

Input: $\boldsymbol{w}_0$, initial parameters of the network; η, step dimension; *maxiter*, number of iterations.

```
1: $\boldsymbol{w} \leftarrow \boldsymbol{w}_0$
2: $i \leftarrow 0$
3: while $i < maxiter$ do
4:     $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_w L(X, y)$
5:     $i \leftarrow i + 1$
6: end while
7: return $\boldsymbol{w}$
8:
```

value is steepest, reaching the minimum point on the surface (indicated by a red dot). Algorithm 1 provides a pseudocode for the gradient descent algorithm and requires the initial parameters of the network $\boldsymbol{w}_0$, the step size η, and the maximum number of iterations to perform. In lines 3-6, the parameters $\boldsymbol{w}$ are updated by subtracting the gradient of the loss function multiplied by the step size η. This ensures that the point follows the steepest descent direction, which corresponds to -1 for the direction given by the gradient of the objective function with respect to the weights (-$\nabla_{\boldsymbol{w}}$ L) (Fig. 2.6).

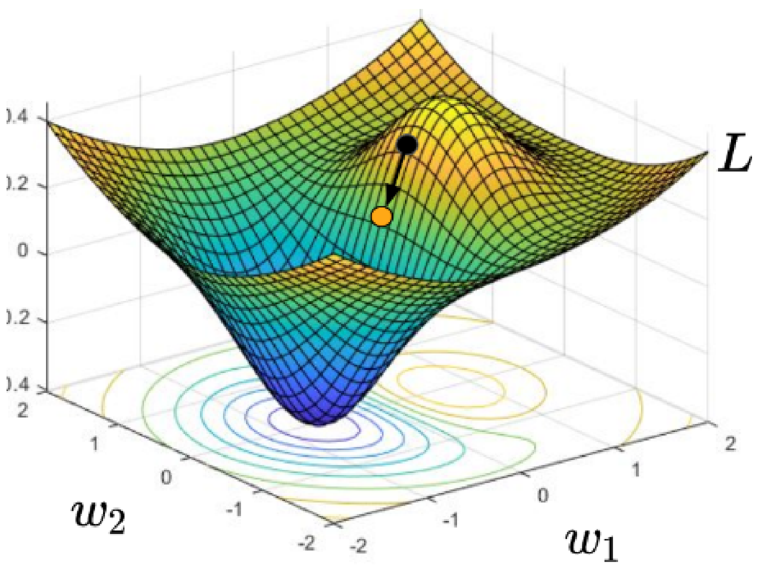


Figure 2.6: Direction to follow iteratively to reach the minimum of the function.

Moreover, there are several variants of gradient descent:

- Stochastic Gradient Descent (SGD): Instead of using the entire dataset to compute the gradient, SGD updates the weights using a single data point. This introduces noise in gradient estimation, which can help escape local minima and improve generalization [23].
- Mini-batch Gradient Descent: This is a compromise between full-batch gradient descent and SGD, where the gradients are computed over small batches of data. This method balances the computational efficiency of SGD with the stable convergence of full-batch gradient descent [23].

2.4 Dimensionality Reduction Techniques

Dimensionality reduction is a critical process in data analytics and machine learning, particularly when dealing with high-dimensional datasets. It involves reducing the number of input variables under consideration, and simplifying the dataset while retaining its essential characteristics.

Datasets with many characteristics are often made up of redundant information, including related or duplicated factors. Feature dimensionality reduction uses existing feature parameters to form a low-dimensional feature space and overcomes the effects of redundant or irrelevant information, thus mapping the effective information contained in the original features to fewer features [9]. In the mathematical sense, suppose there is a n-dimensional vector:

$$X = [x_1, x_2, ..., x_n]^T$$

X is mapped to a m-dimensional vector Y through a map f, where

$$Y = [y_1, y_2, ..., y_m]^T$$

where $m << n$. Vector Y should contain the main features of vector X. Mathematically, the mapping function can be expressed as:

$$Y = f(X)$$

This is the process of feature extraction and selection. This mapping f is the algorithm that we want to find for feature reduction.

2.4.1 Feature selection

Feature selection is the process of selecting feature subsets that are applied to model construction [3]. Methods under feature selection include:

- **Filter Methods**: These methods use statistical techniques to evaluate the importance of features based on their relationship with the output variable.

- **Wrapper Methods**: These methods evaluate the feature subsets based on the performance of a specific machine learning model.
- **Embedded Methods**: These methods perform feature selection as a part of the model training process.

2.4.2 Feature extraction

Feature extraction algorithms are classified into two categories: linear and nonlinear [9]. Linear methods are simpler to handle and interpret, which is why early data dimensionality reduction primarily utilized linear approaches. However, many real-world problems are characterized by nonlinearity and time-variance, prompting increased research into nonlinear feature reduction methods [14].

Feature extraction serves to isolate useful information from redundant data, thereby reducing the computational load on classifiers by lowering the dimensionality. It generates new features from the original ones, meaning the new features are mappings of the original features. The key advantage is that these new features offer more efficient compression. Nonetheless, this process can result in the loss of the physical meaning inherent in the original feature set. The main common techniques include:

1. **Principal Component Analysis (PCA)**: PCA is a linear technique that transforms the data into a new coordinate system. It does so by identifying the directions (principal components) that maximize the variance in the data.
2. **Linear Discriminant Analysis (LDA)**: LDA is used for classification problems and works by finding the linear combination of features that best separate the classes.

3. **Autoencoders**: These are neural networks used for learning efficient codings of input data.

2.5 Introduction to Autoencoders

Autoencoders are a type of artificial neural network used to learn efficient codings of input data, typically for the purpose of dimensionality reduction or feature learning. Introduced by Rumelhart et al. (1986) [17], they have since evolved into sophisticated models capable of capturing patterns in data. High-dimensional data can be transformed into low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors [5]. Dimensionality reduction aids in the classification, visualization, communication, and storage of high-dimensional data. An autoencoder consists of two main blocks [5]:

- An encoder $e(\cdot)$ which maps the input x into a latent space variable z which is much lower dimension.
- A decoder $d(\cdot)$ which tries to reconstruct the input from the latent variable by mapping back to the original space.

The basic structure of an autoencoder is not suited for modeling dynamics. However, it can be extended to a dynamic autoencoder by including past inputs and outputs to the encoder input $x(t)$. Starting from this, the autoencoder will be able to reconstruct a collection of system outputs $\hat{y}(t) = d(z(t))$ that has been obtained by encoding the input $z(t) = e(x(t))$.

The objective is to minimize the reconstruction error, typically measured by the mean squared error (MSE) defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.6}$$

where n is the number of data points, y_i is the actual value of the i-th data point and $\hat{y}_i$ is the predicted value of the i-th data point. In this formula, the term $(y_i - \hat{y}_i)^2$ represents the squared error for each data point, and the MSE is the average of these squared errors over all data points. One of the seminal papers by Hinton and Salakhutdinov (2006) [5] demonstrated that autoencoders, in particular deep autoencoders, can effectively reduce dimensionality while preserving the structure of high-dimensional data. They showed that by pretraining each layer of the network as a Restricted Boltzmann Machine (RBM), the autoencoder could learn a compact representation that outperformed traditional methods like Principal Component Analysis (PCA) in various tasks.

Autoencoders have several applications, including data compression, anomaly detection, and denoising. In this thesis, we will see how they are used in system identification and we will test their robustness to adversarial attacks.

Chapter 3

Learning in Control Theory

In recent years, the field of control systems has experienced a significant transformation due to advancements in machine learning. Traditional control techniques, which relied on precise mathematical models, are now being augmented and, in some cases, replaced by data-driven methods that leverage the power of machine learning. These methods are particularly advantageous for dealing with complex, nonlinear systems. One of the key innovations in this intersection of control and learning is the use of learning architectures for system identification and control [13].

This chapter explores the integration of learning techniques, specifically autoencoders and neural networks, into the field of control systems. The focus is on how these methods can be employed to identify nonlinear state-space models and use these models for control purposes. This chapter will start with a quick overview of dynamical systems and their definition. Then we will examine the problem of system identification, initially discussing traditional methods and then transitioning to those based on learning architectures, with a specific focus on autoencoder approaches as proposed in the paper by [13].

3.1 System identification and traditional methods

A *dynamical system* is an abstract concept representing a physical system that can be described by signals evolving. Such a system is defined by a mathematical model that dictates the laws governing the evolution of these signals. We focus on state-space representations as in Fig. 3.1, which model physical systems through *state variables* and *input/output* signals connected by first-order differential equations (for continuous-time systems) or difference equations (for discrete-time systems).

Every physical system is causal, meaning the output $y(t_0)$ at time t_0 depends only on past and present inputs $u(t)$ for $t \leq t_0$, and not on future inputs. Additionally, the state $x(t_0)$ at time t_0 encapsulates all the information needed, along with the input $u(t)$, to determine the output $y(t_0)$ for all $t \geq t_0$. Knowing the state at time t_0 is equivalent to knowing the inputs applied before t_0 for predicting the output after t_0.

Assume the state $x(0) \in X$ of a system is known at the initial time $t = 0$, where X is the set of all possible states of the system. Letting f denote the law describing the evolution of the state over time, a discrete-time dynamical system is expressed as:

$$x(k+1) = f(x(k), u(k)), \quad k \in \mathbb{N} \tag{3.1}$$

Similarly, a dynamical system evolving in continuous time is expressed as:

$$\dot{x}(t) = f(x(t), u(t)), \quad t \in \mathbb{R} \tag{3.2}$$

In this thesis, we always refer to dynamical systems in discrete time. In some cases, the knowledge of the components that compose a system is not complete or known in advance and the model can be built only using the observation of the input and the output. In the following paragraphs, we will see the main methods of

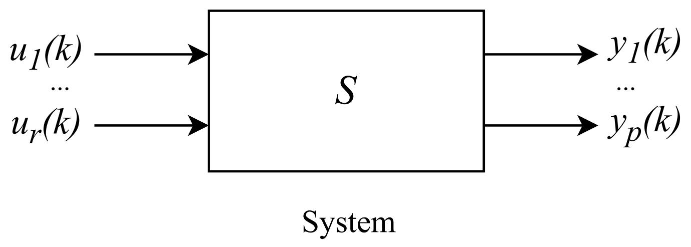


Figure 3.1: Representation of a dynamical system

system identification, starting from the definition of ARX Models and presenting the Least Square Method and then, we will move to modern approaches based on machine learning techniques.

3.1.1 ARX Models and the Linear Least Squares Method

ARX (Auto-Regressive with eXogenous inputs) models are a type of linear dynamic system used in system identification. An ARX model characterizes the system's output as a linear combination of past outputs and inputs, along with an error term. We shall generally denote the system's input and output at time k by $u(k)$ and $y(k)$ respectively [19]. The most basic relationship between input and output is the *linear difference equation*:

$$y(k) = \sum_{j=1}^{n} \alpha_{j-1} y(k-j) + \sum_{j=1}^{r} \beta_{j-1} u(k-j) + \epsilon(k) \tag{3.3}$$

where $\epsilon(k)$ represents some measurement or modeling error. The objective is to estimate a vector of parameters defined as:

$$\theta = [\alpha_1 \; ... \; \alpha_n \; \beta_1 \; ... \; \beta_n]^T \tag{3.4}$$

to build a system in state-space form:

$$\begin{cases} \hat{x}(k+1) = A\hat{x}(k) + Bu(k) \\ \hat{y}(k) = C\hat{x}(k) \end{cases} \tag{3.5}$$

Assuming we do not know the values of the parameter in θ, but have recorded inputs and outputs for at least L sampling intervals, our dataset is:

$$Z = \{u_0, y_0, \; ... \; , u_N, y_N\} \tag{3.6}$$

We then select θ to fit the calculated values as closely as possible to the measured outputs using the least squares method [21]:

$$J(\theta) = \frac{1}{N} \sum_{k=n-1}^{L} (\hat{y}(k) - y(k))^2 \tag{3.7}$$

This objective function $J(\theta)$ represents a quadratic prediction error between the measured signal $y(k)$ and the predicted output $\hat{y}(k)$. Minimizing the objective function allows us to find the optimal parameter vector θ:

$$\arg\min_{\theta} J(\theta) \tag{3.8}$$

3.1.2 System Identification Procedure

The identification of a model from data involves three basic entities:

1. A dataset like Z in 3.6.
2. A set of candidate models: a Model Structure like in 3.3.
3. A rule by which candidate models can be assessed using the data, like the Least Square selection rule.

Once the previous choices have been made, we arrive at a specific model that best describes the data according to the chosen criterion. The next step is to test whether this model is good enough for its purpose. This process is known as *model validation* and involves various procedures to assess the model's performance in relation to observed data and prior knowledge.

3.2 Autoencoder-based models

The identification of nonlinear state-space models is a critical aspect of modern control theory, particularly as systems become more complex and demand more precise control mechanisms [16]. Autoencoders, a type of artificial neural network designed for dimensionality reduction, are capable of learning compressed representations of input data and modeling dynamical systems. In their paper, Masti and Bemporad [13] proposed a novel methodology for the identification of nonlinear state-space models using autoencoders and neural networks.

3.2.1 Nonlinear Identification Problem

Nonlinear system identification has become increasingly popular, driven by advancements in machine learning methods for nonlinear function regression [24] [13]. The concept of using machine learning techniques to identify state-space representations of dynamical systems from input/output data has been extensively explored in the literature. Typically, learning a state-space model involves a nonlinear transformation of a vector composed of past input/output samples into a state vector.

We can define the problem of nonlinear identification from a mathematical point of view. We are given a training dataset of input/output samples $Z = \{u_0, y_0, \dots, u_N, y_N\}$ collected from a dynamical system, where $u_k \in \mathbb{R}^{n_u}$ is the vector of exogenous inputs and $y_k \in \mathbb{R}^{n_y}$ the vector of measured outputs. The goal is to identify a dynamical model in state-space form as:

$$\begin{cases} x_{k+1} = f(x_k, u_k) \\ \hat{y}_k = g(x_k) \end{cases} \tag{3.9}$$

with $x \in \mathbb{R}^{n_x}$, that, starting from an appropriate condition x_{k_0} and excited by the same inputs $u_{k_0}, ..., u_{N-1}$, produces an output signal $\hat{y}_k$ that is as close as possible to the one y_k recorded on the system. Given a number of past outputs $n_a \geq 1$, of past inputs $n_b \geq 1$, and a desired state dimension $n_x \geq 1$, the problem can be recast to the problem of finding a triplet of maps e, f, g defined as:

$$\begin{aligned} e &: \mathbb{R}^{n_I} \longrightarrow \mathbb{R}^{n_x} \\ f &: \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \longrightarrow \mathbb{R}^{n_x} \\ g &: \mathbb{R}^{n_x} \longrightarrow \mathbb{R}^{n_y} \end{aligned} \tag{3.10}$$

where $n_I \triangleq n_a n_y + n_b n_u$, that solves the following optimization problem:

$$\min_{e,\, f,\, g} \mathcal{L}(e,\ f,\ g,\ Z) \tag{3.11}$$

where

$$\begin{aligned} \mathcal{L}(e,\ f,\ g,\ Z) = \sum_{k=k_0}^{N} & L(\hat{y}_k, y_k) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & \hat{y}_k = g(x_k), \quad k = k_0, \dots, N \\ & x_{k_0} = e(I_{k_0-1}) \end{aligned} \tag{3.12}$$

In 3.12, $L : \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \longrightarrow [0, +\infty)$ is a suitable loss function that penalizes the discrepancy between the predicted and the measured output, and I_k is the information vector defined as:

$$I_k = [y'_k \,...\, y'_{k-n_a+1} \; u'_k \,...\, u'_{k-n_b+1}]' \tag{3.13}$$

where $k_0 \triangleq \max\{n_a, n_b\}$. In 3.11, e is the dimensionality-reduction mapping from the vector I_k of past inputs and outputs to the state vector x_k.

3.2.2 Partial Predictive Autoencoders

The approach proposed by [13] is based on the underlying idea of training an Artificial Neural Network to reproduce the identity mapping from a certain information vector $I_k \in \mathbb{R}^{n_I}$ to I_k itself, having as a constraint to learn a description of I_k that lives in the lower-dimensional space $\mathbb{R}^{n_x}$ without losing information. Thus, the final objective is to reduce the fitting error between I_k and the reconstructed I_k. They have introduced the concept of *partial predictive autoencoder* (PPE) that maps I_{k-1} (i.e., the information available up to time $k-1$) into the following vector of outputs:

$$O_k = [y'_k \,...\, y'_{k-m}]' \tag{3.14}$$

with $0 \leq m \leq n_a$. This approach first achieves a compressed representation of the state x_k, which effectively works as a latent space that captures the information from I_k to predict y_k. The Partial Predictive Autoencoder is therefore composed of two distinct Artificial Neural Networks:

- Encoding function $e : \mathbb{R}^{n_I} \longrightarrow \mathbb{R}^{n_x}$, representing the transformation from I_{k-1} (past inputs and outputs) to x_k (state vector);
- Decoding function $d : \mathbb{R}^{n_x} \longrightarrow \mathbb{R}^{mn_y}$, from x_k (state vector) to O_k (output vector).

This method was primarily adopted because employing a traditional autoencoder was not optimal for compressing the information of I_k into a reduced-order vector. The traditional approach would be suboptimal as the samples constituting I_k would be treated as independent samples rather than shifted components that share common elements.

3.2.3 Model Learning

The initial structure described forms the foundation for mapping the information available to time instant $k-1$ into the current state. Subsequently, a function f was introduced to map the current state x_k and the input u_k into the next state x_{k+1}. The proposed approach aims to learn three functions simultaneously: e, f, and d. This problem is also known as a multi-objective learning problem. In Fig. 3.2 there is the representation of the overall structure proposed by Masti and Bemporad [13]. Essentially, two Partial Predictive Autoencoders that share the same set of weights are used: one is fed by I_{k-1} and the other by I_k. The objective is to reproduce O_k and O_{k+1} respectively. By following this approach, the generated state x_k in the first autoencoder and x_{k+1} in the second autoencoder will be coherent. Finally, a third neural network must be trained to map u_k and x_k into the shifted state x_{k+1}, thus obtaining the state-update mapping f. The overall training problem described above has been formulated as the following optimization problem:

$$
\begin{aligned}
\min_{e,f,d} \sum_{k=k_0}^{N-1} \alpha \Big(L_1(\hat{O}_k, O_k) + L_1(\hat{O}_{k+1}, O_{k+1}) \Big) + \beta L_2(x^\star_{k+1}, x_{k+1}) + \gamma L_3(O_{k+1}, O^\star_{k+1}) \\
\text{s.t.} \quad & x_k = e(I_{k-1}), \quad k = k_0, \dots, N \\
& x^\star_{k+1} = f(x_k, u_k), \quad k = k_0, \dots, N-1 \\
& \hat{O}_k = d(x_k), \quad k = k_0, \dots, N \\
& O^\star_k = d(x^\star_k), \quad k = k_0 + 1, \dots, N
\end{aligned} \tag{3.15}
$$

where L_i are loss functions, $\alpha, \beta, \gamma \geq 0$ are scalar weights. Minimizing L_2 and L_3 has the most priority since they capture the one-step ahead properties of the model. Thus, the approach proposed is to start the training process with small values of β, γ, and a high value of α, and then use the solution as the initial guess of another instance of 3.15 with a smaller value of α, possibly reiterating the procedure multiple times for decreasing values of α.

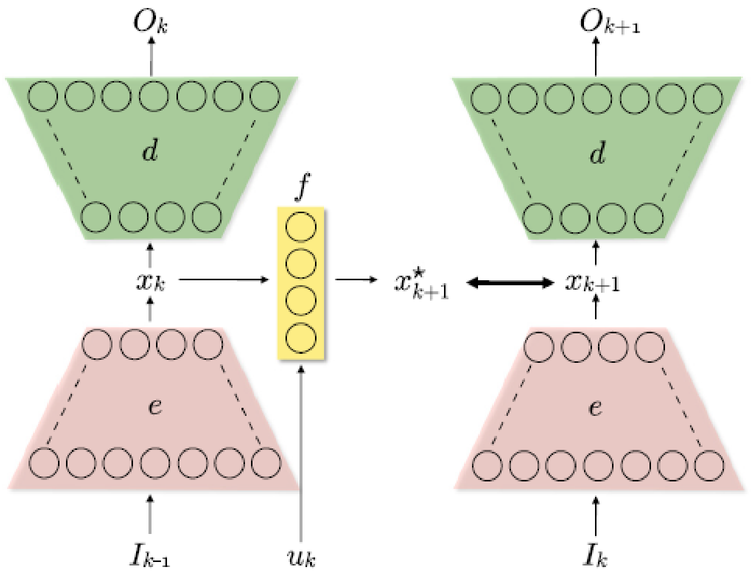


Figure 3.2: Autoencoder structure proposed in [13]

Chapter 4

Adversarial Learning on Autoencoder-based models

In this chapter, we will provide an overview of adversarial learning, a field that has been deeply explored in domains such as computer vision and image recognition but remains relatively unexplored in the context of control systems based on machine learning. Adversarial learning involves techniques designed to deceive machine learning models, exploiting their vulnerabilities. This has brought for sure to build defenses and thus more robust systems. By extending these concepts to control systems, we aim to bridge this gap and explore the implications of adversarial attacks in this novel application area.

We will start with an overview of adversarial learning, drawing on its foundational principles and common techniques. Then, we will move to the core of this chapter focused on the formulation of a novel adversarial attack that aims to fool an autoencoder-based model prediction. We will describe the methodology employed to craft these attacks, using gradient-based techniques to perturb inputs and deceive the model's state estimation and output reconstruction processes.

4.1 Adversarial learning overview

Adversarial learning, an area of research within machine learning, focuses on the development and mitigation of adversarial attacks. The underlying idea of the attack is to craft intentional perturbations to input data to deceive machine learning models into making incorrect predictions. This phenomenon has been already studied in domains such as computer vision and cybersecurity, where adversarial examples have shown vulnerabilities in neural networks and other learning algorithms [2].

Evasion attacks [1] are particularly relevant in the context of machine learning models. These attacks generate inputs that are slightly different from legitimate ones but cause errors in the prediction of the model. The seminal work by Szegedy et al. [22] and Goodfellow et al. [4] demonstrated the susceptibility of deep neural networks to such attacks, giving research ideas for both attack and defense strategies.

In control systems based on machine learning, adversarial attacks can represent significant threats to the stability and safety of the system. Autoencoders, typically used for feature learning and dimensionality reduction, can be vulnerable to adversarial attacks. The introduction of perturbation from the attacker could lead to incorrect reconstructions or predictions, compromising the performance. Thus, in recent years there has been an increasing adoption of neural network-based methods for anomaly detection in industrial control systems [11]. Also, resilient control methods such as those developed in [10] [18] may offer formal guarantees against adversarial attacks on control systems.

Poisoning attacks are particularly famous in industrial control systems. These attacks occur during the training time when the attacker injects malicious data into the training set to influence the model's learning and behavior. As Kravchik et al. have shown, poisoning can allow cyber-attacks to go undetected, especially in systems that rely on continuous online training [11].

In the following sections, we will see a specific attack developed to test the robustness of those systems based on autoencoders, outlining the threat model taken into account and the attack formulation.

4.2 Adversarial Attack Formulation

The adversarial attack developed for this type of neural network is a gradient-based adversarial attack. This technique generates perturbations to the original input by leveraging the gradients of the loss function with respect to the input data. The principle is to exploit the gradients computed during the backpropagation process. As discussed in Chapter 2.1, gradients are typically used to update the model's parameters to minimize the loss function. However, the same gradients can also be used to determine how to alter the input data to maximize the loss function.

This study focuses on attacking the prediction of the output O_{k+1} produced by the autoencoder. The strategy involves applying an iterative attack to craft an adversarial input u_k that is then fed into the bridge neural network f (see Fig. 3.2).

This adversarial attack targets the network f, which maps the encoder's output, which represents the current state, along with the input into the next state. The attack aims to generate an adversarial input that disrupts the model's ability to accurately predict the next latent state representation from the current state and input. This perturbed latent state is then passed to the decoder to produce the estimated output of the real dynamical system.

Gradient-Based Perturbation.

Since the training algorithm employs a stochastic optimization method, we can exploit a gradient-based attack to modify the original input u_k in a direction that increases the loss.

The perturbation is computed using the gradient of the loss function $\mathcal{L}$ with respect to the input u_k. The loss function measures the discrepancy between the true output y_{k+1} and the predicted output O^*_{k+1}:

$$\mathcal{L}(y_{k+1}, O^*_{k+1}) = \text{MAE}(y_{k+1}, O^*_{k+1}) = \frac{1}{n}\sum_{i=1}^{n} |y^{(i)}_{k+1} - O^{*(i)}_{k+1}| \tag{4.1}$$

where MAE is the Mean Absolute Error loss function taken into account. Then, we compute the gradient of the loss with respect to the input u_k is given by:

$$\nabla_{u_{adv}}\mathcal{L}(y_{k+1}, O^*_{k+1}) \tag{4.2}$$

The adversarial input u_{adv} is thus updated according to:

$$u^{(i+1)}_{adv} = u^{(i)}_{adv} + \epsilon * \text{sign}(\nabla_{u_{adv}}\mathcal{L}(y_{k+1}, O^*_{k+1})) \tag{4.3}$$

where i indexes the iteration, ϵ is a small positive scalar, and $\nabla u_{adv}\mathcal{L}$ represents the gradient of the loss with respect to the adversarial input. This process is repeated for N iterations to obtain the final adversarial input u_{adv}, since the attack proposed employs an iterative approach.

4.2.1 Attack on Bridge Network

In this scenario, the perturbation is applied directly to the input u_k provided to the bridge network f to deceive the estimation of the next state x_{k+1} and, consequently, the output O_{k+1} produced by the decoder. As explained in the previous section, the adversarial input u_{adv} is crafted by performing a maximization of the loss function by iteratively adjusting it. The algorithm is the following:

Algorithm 2 Adversarial Attack on Output O_{k+1}

```
Input: Original input $u_k$, current latent state $x_k$, real output of the dynamical
    system $y_{k+1}$, model bridge network $f$, number of iterations $N$, perturbation
    budget $\epsilon$.
Output: Crafted adversarial input $u_{adv}$.
    Initialize $u_{adv} \leftarrow u_k$
    for $i = 1$ to $N$ do
        Calculate next state prediction $x^*_{k+1} = f(u_{adv}, x_k)$
        Calculate the next output prediction $O^*_{k+1} = d(x^*_{k+1})$
        Compute loss $\mathcal{L} = \text{MAE}(y_{k+1}, O^*_{k+1})$
        Calculate gradient of loss $\nabla_{u_{adv}}\mathcal{L}$
        Update adversarial input $u_{adv} \leftarrow u_{adv} + \epsilon \cdot \text{sign}(\nabla_{u_{adv}}\mathcal{L})$
    end for
    return $u_{adv}$
```

4.2.2 Attack on Information Vector

In this scenario, the perturbation is crafted using the same methodology as the scenario presented in 4.2.1, but it is then added to the information vector I_k. The information vector I_k includes past outputs and inputs:

$$I_k = [y_k, y_{k-1}, \dots, y_{k-n_a+1}, u_k, u_{k-1}, \dots, u_{k-n_b+1}]' \tag{4.4}$$

The perturbed information vector is denoted as I_k^{pois}, which means that the information vector is poisoned with perturbed data. The perturbed information vector is defined as:

$$I_k^{pois} = [y_k^{adv}, y_{k-1}, \dots, y_{k-n_a+1}, u_k^{adv}, u_{k-1}, \dots, u_{k-n_b+1}]' \tag{4.5}$$

where y_k^{adv} and u_k^{adv} are respectively, the output produced by the decoder under attack and the input perturbed. Thus, both are added to the information vector.

We have to highlight that the information vector is poisoned in a small portion, thus this attack can give us an idea of how reliable these learning architectures are.

Chapter 5

Experimental Analysis

This chapter presents a comprehensive experimental analysis of the proposed adversarial attack approaches. Both attack scenarios—perturbing the input u_k directly to deceive the bridge network f and poisoning the information vector I_k with perturbed data—have been tested. These attacks were evaluated on the same benchmark problems proposed by Masti and Bemporad [13], ensuring consistency and comparability with the original model's performance.

This experiment's neural network and autoencoder architecture parameters are identical to those specified in Masti and Bemporad's paper [13]. This includes the network topology, the size of the hidden layers, the activation functions, and the training procedure. By maintaining these parameters, we ensure that any observed deviations in model performance can be attributed to adversarial attacks rather than differences in network configurations.

The adversarial attacks were executed during the open-loop validation phase. In this way, we can assess the robustness of the model without any feedback adjustments.

The following sections will detail the experimental setup, the specific benchmark problems used, the implementation of the adversarial attacks, and the resulting performance metrics.

5.1 Experimental Setup

Benchmark problems.

In this experiment, we evaluate the effectiveness of the proposed adversarial attack on two synthetic benchmark systems [20]: the Hammerstein-Wiener system and a Discrete-Time Nonlinear Tank system. Each system represents different types of dynamic behaviors and is described in terms of its input, output, and internal states from a physical perspective.

1. Hammerstein-Wiener System (Σ_{HW})

The Hammerstein-Wiener system [8] is a widely used model in control theory, combining both nonlinear static functions and linear dynamic systems. This system represents processes with nonlinearity at the input and output stages, with linear dynamics governing the evolution of the internal state (Fig. 5.1).

- **Input**(u_k): The input u_k is subjected to a nonlinear transformation v_k before influencing the system's state.
- **State Transition**(x_{k+1}): The state vector x_k evolves according to the linear dynamics governed by the state transition matrix.
- **Output**(y_k): The output y_k is derived from the internal state via a nonlinear function applied to w_k, which is a linear transformation of the state.

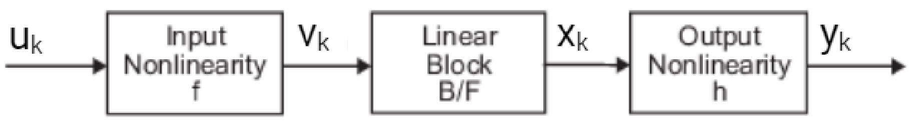


Figure 5.1: Hammerstein-Wiener System - Schematic representation

The system Σ_{HW} is mathematically represented as:

$$\Sigma_{HW} = \begin{cases} x_{k+1} = \begin{bmatrix} 0.7555 & 0.25 \\ -0.1991 & 0 \end{bmatrix} x_k + \begin{bmatrix} -0.5 \\ 0 \end{bmatrix} u_k \\ v_k = \begin{cases} \sqrt{u_k} & \text{if } u > 0 \\ u_k & \text{otherwise} \end{cases} \\ w_k = \begin{bmatrix} 0.6993 & -0.4427 \end{bmatrix} x_k \\ y_k = w_k + 5\sin(w_k) \end{cases} \tag{5.1}$$

2. Discrete-Time Nonlinear Tank System (Σ_T)

The second benchmark system is a discrete-time nonlinear model representing a tank system [6]. This system models the fluid levels in interconnected tanks (Fig. 5.2), where fluid inflows and outflows influence the dynamics.

- **Input**(v_k): The input v_k is defined as $v_k = \text{sign}(u_k)u_k^2$ where u_k represents an external flow. Physically, this input represents the flow rate of fluid into the tank system.

- **State Transition**($x_{k+1,1}$ and $x_{k+1,2}$): The state variables $x_{k,1}$ and $x_{k,2}$ corresponds to the fluid levels in two tanks.
- **Output**(y_k): The output y_k represents the fluid level in the second tank ($x_{k,2}$), which is the primary quantity of interest in this system. This output can be measured and used for monitoring purposes, such as maintaining the tank level within a desired range.

The system Σ_T is mathematically represented as:

$$\Sigma_T = \begin{cases} x_{k+1,1} = x_{k,1} - k_1\sqrt{x_{k,1}} + k_2 v_k \\ x_{k+1,2} = x_{k,2} + k_3\sqrt{x_{k,1}} - k_4\sqrt{x_{k,2}} \\ y_k = x_{k,2} \end{cases} \tag{5.2}$$

where the constants $k_1 = 0.5$, $k_2 = 0.4$, $k_3 = 0.2$ and $k_4 = 0.3$ are system parameters that define the interaction between the tanks.

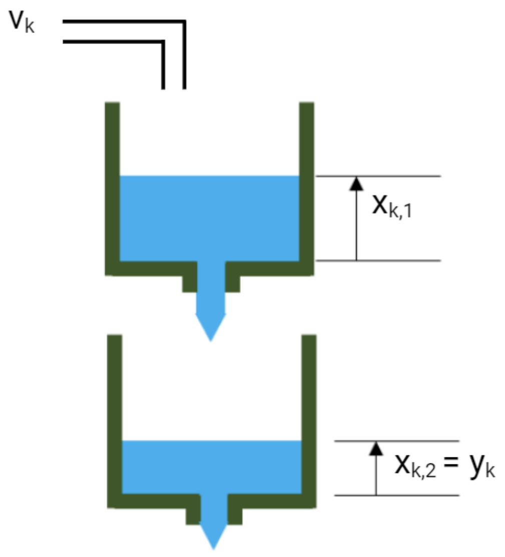


Figure 5.2: Two Tanks System - Schematic representation

Datasets.

For the system Σ_T the dataset is made up of 20.000 training samples generated by exciting the system with a sequence of step signals of length 5 steps with random amplitudes drawn from the Gaussian Distribution $\mathcal{N}(1, 1)$. For Σ_{HW}, we consider a training set of 10.000 samples generated using the same methodology but with amplitudes of the input signals distributed in $\mathcal{N}(0, 1)$ and length equal to 7 steps.

Models' Hyperparameters.

The hyperparameters for both systems in this experiment are consistent with those used in the Masti-Bemporad paper [13]. Specifically, the state dimension n_x is set to 6, and the past input/output window width is defined as $n_a = n_b = 10$.

Additionally, the networks employed in the experiment are fully connected feed-forward networks, each consisting of 3 layers with 30 neurons per layer. The hidden layers utilize ReLU activation functions, ensuring nonlinearity in the model, while the final layer is a linear output layer to map the hidden states to the output. A regularization term $\chi = 0.0001$ is applied to improve generalization across the networks.

Attack's Hyperparameters.

The attack has been run during the open-loop validation. Thus, an adversarial input is crafted at each iteration of the open-loop validation. The attack algorithm requires a set of parameters: the number of iterations N and the magnitude of the perturbation ϵ applied to the input u_k. We combine each value of the sets below

- Number of iterations N: {1, 10, 30}
- ϵ: {0.01, 0.1, 0.3, 0.5}

Furthermore, the open-loop validation needs to set a boolean parameter to choose the nature of the input signal. The input signal can be either of a multi-harmonic nature or not. The attack has been tested for both cases.

Evaluation Criteria.

Performances of the attack are measured in terms of the Best Fit Ratio (BFR)

$$\text{BFR} = \max\left\{0, 1 - \frac{||y_t - \hat{y}_t||_2}{||y_t - \bar{y}||_2}\right\} \tag{5.3}$$

where $\bar{y}$ is the average of the output signal y over all the samples and $\hat{y}_k$ is the open-loop prediction extracted from $\hat{O}_{k+1} = d(\hat{x}_{k+1})$ with

$$\hat{x}_{k+1} = f(\hat{x}_k, u_k), k = k_0, ..., N - 1 \tag{5.4}$$

$$\hat{x}_{k_0} = e(I_{k_{0-1}}) \tag{5.5}$$

It is important to note that the paper proposing this methodology based on autoencoder structures reported a Best Fit Ratio of 0.98 [13] under normal conditions.

5.2 Experimental Results

In this section, we present the experimental results of the adversarial attacks on the autoencoder-based models presented in Chapter 4. The results are categorized into two primary scenarios:

5.2.1 Attack results on Bridge Network (Scenario 1)

In this scenario, the attack is executed directly to the input u_k of the bridge neural network f. The attack is evaluated across both benchmark problems outlined in the previous section. Four specific experiments were conducted:

E1. Attack on the Hammerstein-Wiener Model (System 5.1) using a multi-harmonic input signal.

E2. Attack on the Hammerstein-Wiener Model (System 5.1) using a generic signal.

E3. Attack on the Two Tanks Model (System 5.2) using a multi-harmonic input signal.

E4. Attack on the Two Tanks Model (System 5.2) using a generic signal.

In each experiment, we report a table that details the Best Fit Ratio for each combination of the number of iterations N and the perturbation magnitude ϵ. In addition, several plots are shown, illustrating the model's output under attack along with the corresponding adversarial input. Each row of figures, includes the adversarial signal on the left and the output plot on the right, displaying the actual signal and the reconstructed signal. The results for each experiment, denoted as E1, E2, E3, and E4, are presented below.

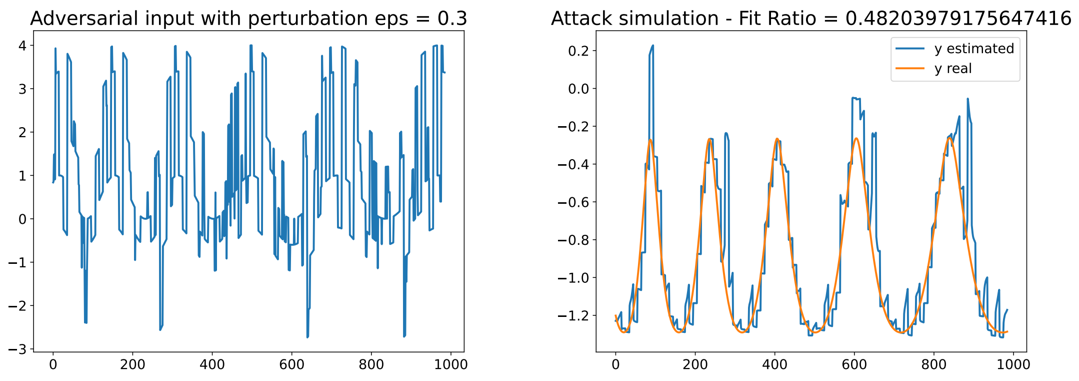
Experiment E1 Results.

In this experiment, the attack is performed on the Hammerstein-Wiener model considering a multi-harmonic input signal applied to the bridge network f and perturbed during the open-loop validation. The table 5.1 shows for each couple of attack's hyperparameters, the Best Fit Ratio, which measures the quality of the output estimation.

Fig. 5.3a and Fig. 5.3b show respectively the adversarial input signal and the estimation of the output. We can observe a massive degradation of the performances in the estimation of the output, despite the small perturbation applied.

Results Experiment E1 | Hammerstein-Wiener Model		
Iterations	**Epsilon**	**Best Fit Ratio**
1	0.01	0.803
	0.1	0.812
	0.3	0.797
	0.5	0.737
10	0.01	0.800
	0.1	0.812
	0.3	0.482
	0.5	0.331
30	0.01	0.805
	0.1	0.348
	0.3	0.439
	0.5	0.319

Table 5.1: Experiment E1 - Performances of the attack on Hammerstein-Wiener Model with Multi-Harmonic input signal.



(a) Adversarial input.

(b) Estimated output.

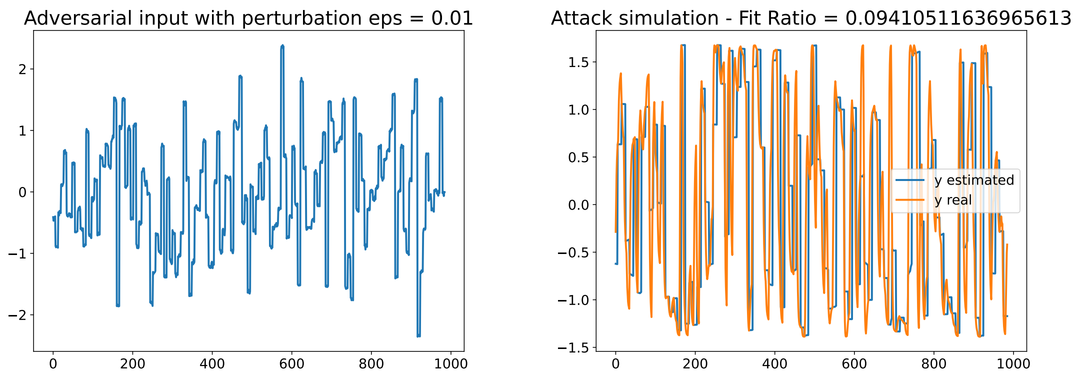
Figure 5.3: Hammerstein-Wiener Model - Multi-Harmonic Input Signal - $N = 10$, $\epsilon = 0.3$.

Experiment E2 Results.

In this experiment, the attack is performed on the Hammerstein-Wiener model considering a generic input signal applied to the bridge network f and perturbed during the open-loop validation. Also in this case, the table 5.2 shows for each couple of attack's hyperparameters, the Best Fit Ratio, which measures the quality of the output estimation. In this case, we can observe in Figs. 5.4a, 5.4b that the Best Fit Ratio decreases much more concerning the previous case. Overall, the application of the generic input signal results in the worst performance.

Results Experiment E2 | Hammerstein-Wiener Model		
Iterations	**Epsilon**	**Best Fit Ratio**
1	0.01	0.093
	0.1	0.091
	0.3	0.088
	0.5	0.090
10	0.01	0.090
	0.1	0.084
	0.3	0.077
	0.5	0.072
30	0.01	0.090
	0.1	0.085
	0.3	0.089
	0.5	0.065

Table 5.2: Experiment E2 - Performances of the attack on Hammerstein-Wiener Model with generic input signal.



(a) Adversarial input.

(b) Estimated output.

Figure 5.4: Hammerstein-Wiener Model - Generic Input Signal - $N = 1$, $\epsilon = 0.01$.

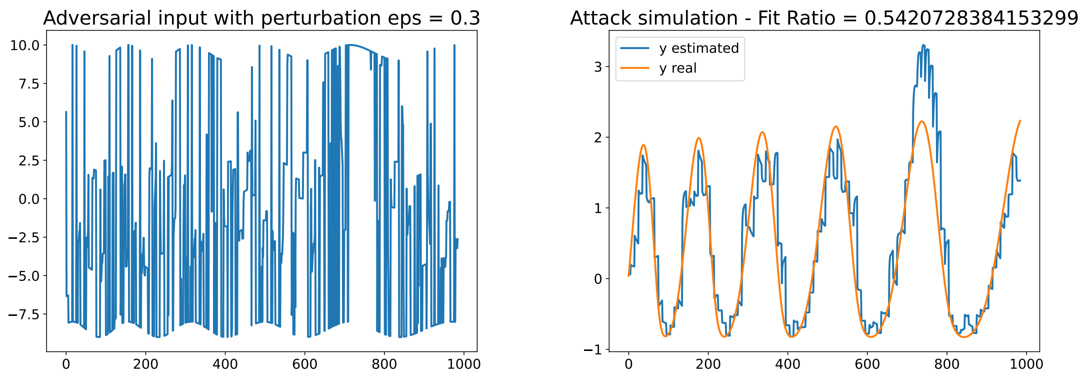
Experiment E3 Results.

The experiment is now moved to the Two Tanks Model (System 5.2). Also in this case a table is reported (Table 5.3), showing the performances of the autoencoder under attack conditions. In this case, the system has a precise physical meaning: the deviation of the output could bring to possible overflow in the tank.

Figures 5.5a, 5.5b show that the deviation output is reached, but in this case, the perturbation is more important if we compare to the previous case. This could be because, in this case, the autoencoder has been trained with a larger dataset concerning the Hammerstein-Wiener model.

Results Experiment E3 | Two Tanks Model		
Iterations	**Epsilon**	**Best Fit Ratio**
1	0.01	0.808
	0.1	0.806
	0.3	0.803
	0.5	0.799
10	0.01	0.806
	0.1	0.795
	0.3	0.793
	0.5	0.713
30	0.01	0.803
	0.1	0.793
	0.3	0.542
	0.5	0.0

Table 5.3: Experiment E3 - Performances of the attack on Two Tanks Model with Multi-Harmonic input signal.



(a) Adversarial input crafted.

(b) Estimated output.

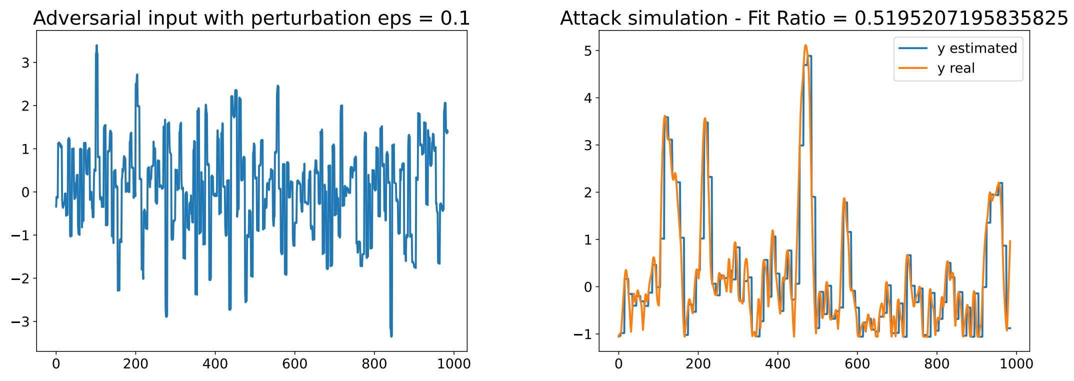
Figure 5.5: Two tanks model - Multi-Harmonic Input Signal - $N = 30$, $\epsilon = 0.3$.

Experiment E4 Results.

Finally, the attack is also evaluated using a generic input signal. The results are summarized in Table 5.4 also report the results in terms of the Best Fit Ratio. Figures 5.6a, 5.6b illustrate the poor reconstruction of the output under this scenario. Notably, as shown previously, the perturbation applied in this case is smaller compared to the scenario with the multi-harmonic input.

Results Experiment E4 | Two Tanks Model		
Iterations	**Epsilon**	**Best Fit Ratio**
1	0.01	0.519
	0.1	0.519
	0.3	0.520
	0.5	0.520
10	0.01	0.519
	0.1	0.524
	0.3	0.485
	0.5	0.465
30	0.01	0.520
	0.1	0.520
	0.3	0.412
	0.5	0.268

Table 5.4: Experiment E4 - Performances of the attack on Two Tanks Model with generic input signal.



(a) Adversarial input crafted. (b) Estimated output.

Figure 5.6: Two tanks model - Generic Input Signal - $N = 1$, $\epsilon = 0.1$.

5.2.2 Attack results on Information Vector (Scenario 2)

In this scenario, the attack is executed, creating the adversarial input u_k^{adv} which produces a mispredicted output y_k^{adv}. These elements are then added to the Information Vector to poison it. The attack is assessed using the benchmark problems described in the previous section. Four experiments were carried out:

P1. Poisoning of the Information vector on the Hammerstein-Wiener Model (System 5.1) using a multi-harmonic input signal.

P2. Poisoning of the Information vector on the Hammerstein-Wiener Model (System 5.1) using a generic input signal.

P3. Poisoning of the Information vector on the Two Tanks Model (System 5.2) using a multi-harmonic input signal.

P4. Poisoning of the Information vector on the Two Tanks Model (System 5.2) using a generic input signal.

With this type of attack, we set the perturbation magnitude $\epsilon = 0.01$ and the number of iterations $N = 1$. This choice was made because we want to know how the performance of the autoencoder varies by varying the poisoning percentage of the information vector by setting a set of attack hyperparameters. This way, we obtain more representative data since we inject data with the minimum possible disturbance.

Experiment P1 Results.

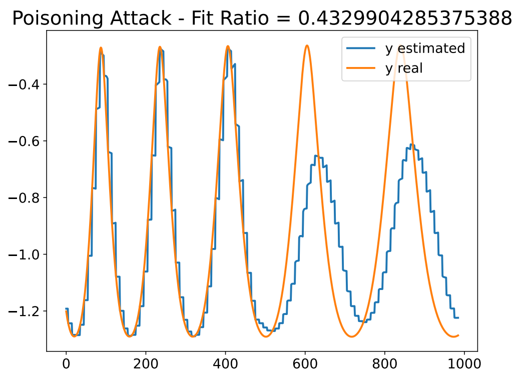
In this experiment, the poisoning attack is performed during the reconstruction of the Hammerstein-Wiener model considering a multi-harmonic input signal. Table 5.5 shows the performances in terms of the Best Fit Ratio when we increase the percentage of poisoning of the information vector. Figures 5.7 show the estimation of the output capability when we poison the information vector.

Results Experiment P1 | Hammerstein-Wiener Model				
Poisoning	**5%**	**10%**	**15%**	**20%**
Best Fit Ratio	0.43	0.17	0.11	0.0

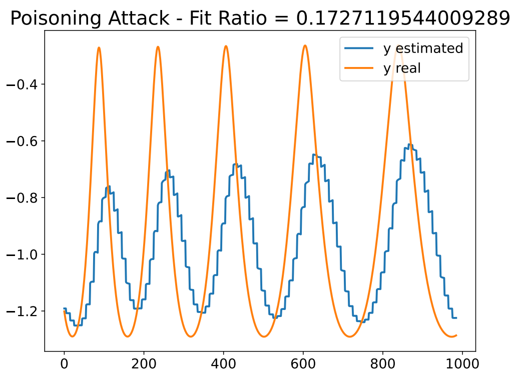
Table 5.5: Experiment P1 - Performances of the poisoning on Hammerstein-Wiener Model with multi-harmonic input signal.

Experiment P2 Results.

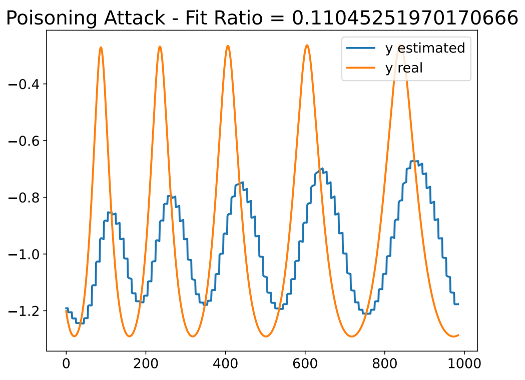
Similarly, it was possible to evaluate the poisoning attack on the same model as the previous experiment, but this time the signal applied to the autoencoder is a generic signal. Thus, Table 5.8 shows the performances as well, and we can observe a massive degradation of output reconstruction, even reaching 0.0 of the Best Fit Ratio in some cases.



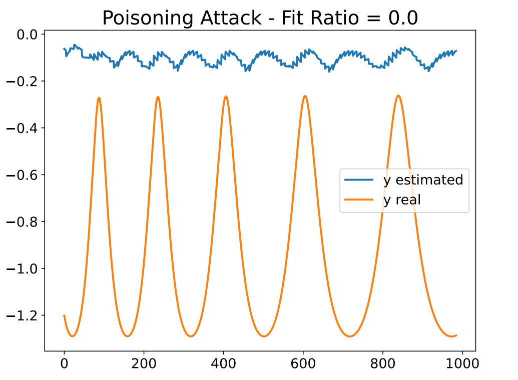
(a) 5% poisoning



(b) 10% poisoning



(c) 15% poisoning

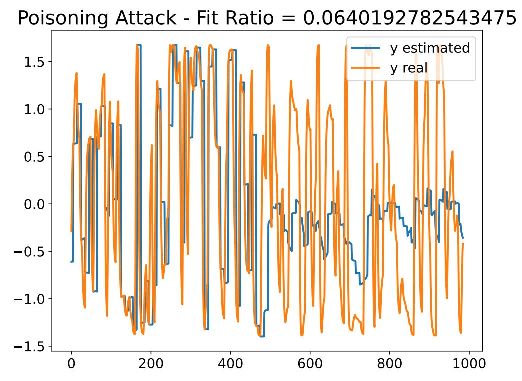


(d) 20% poisoning

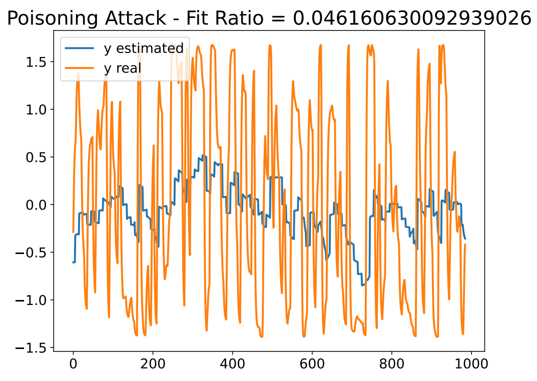
Figure 5.7: Poisoning Attack. Hammerstein-Wiener model - Multi-harmonic Input Signal

Results Experiment P2 | Hammerstein-Wiener Model				
Poisoning	**5%**	**10%**	**15%**	**20%**
Best Fit Ratio	0.06	0.04	0.02	0.0

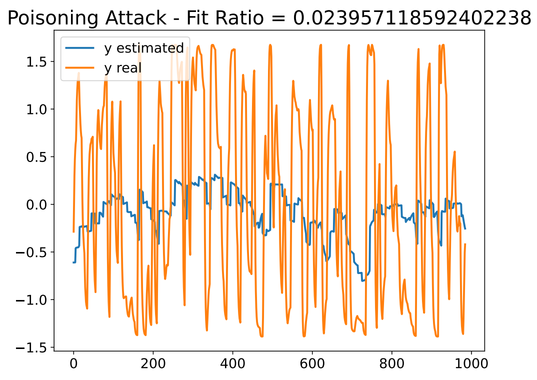
Table 5.6: Experiment P2 - Performances of the poisoning on Hammerstein-Wiener Model with generic input signal.



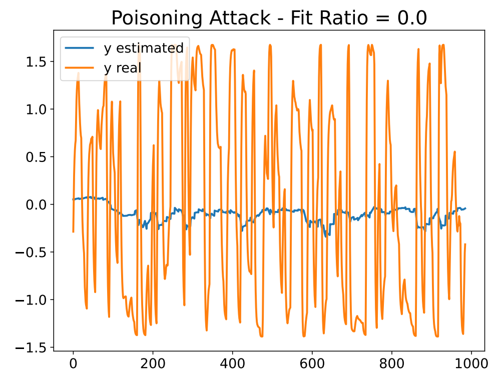
(a) 5% poisoning



(b) 10% poisoning



(c) 15% poisoning



(d) 20% poisoning

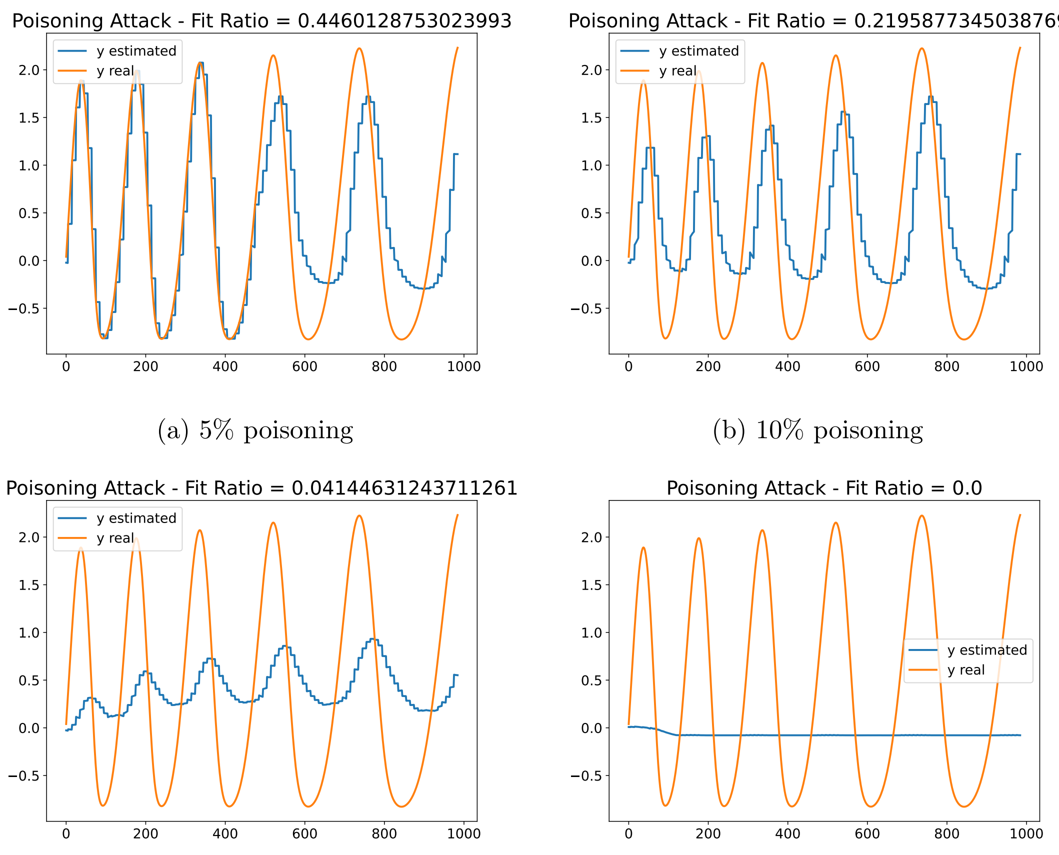
Figure 5.8: Poisoning Attack. Hammerstein-Wiener model - Generic Input Signal

Experiment P3 Results.

This experiment applies the poisoning attack to the Two Tanks model with a multi-harmonic input signal. Table 5.7 displays the performance results regarding the Best Fit Ratio as the percentage of information vector poisoning increases. Figures 5.9 illustrate the output estimation capability under various poisoning levels.

Results Experiment P3 | Two Tanks Model				
Poisoning	**5%**	**10%**	**15%**	**20%**
Best Fit Ratio	0.44	0.21	0.04	0.0

Table 5.7: Experiment P3 - Performances of the poisoning on Two Tanks Model with multi-harmonic input signal.



(a) 5% poisoning

(b) 10% poisoning

(c) 15% poisoning

(d) 20% poisoning

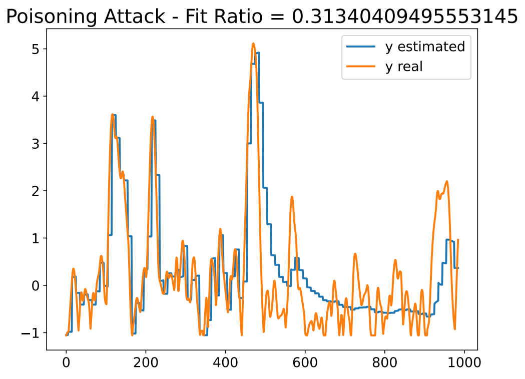
Figure 5.9: Poisoning Attack. Two Tanks model - Multi-harmonic Input Signal

Experiment P4 Results.

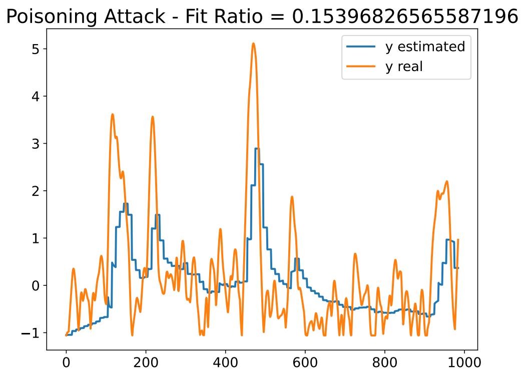
This experiment evaluates the impact of the poisoning attack on the Two Tanks model with a generic input signal. The results, presented in Table 5.8, demonstrate a significant reduction in the Best Fit Ratio as the percentage of poisoning increases. Figures 5.10 provide a visual representation of the output estimation under different levels of poisoning.

Results Experiment P4 | Two Tanks Model				
Poisoning	**5%**	**10%**	**15%**	**20%**
Best Fit Ratio	0.31	0.15	0.0	0.0

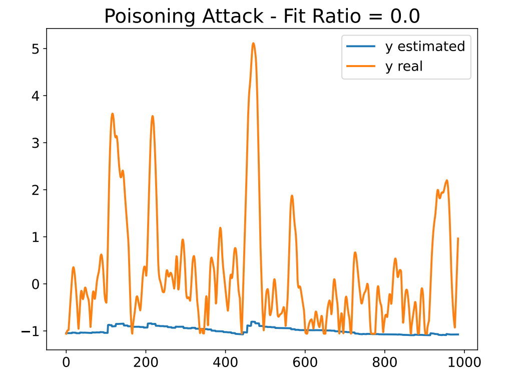
Table 5.8: Experiment P4 - Performances of the poisoning on Two Tanks Model with generic input signal.



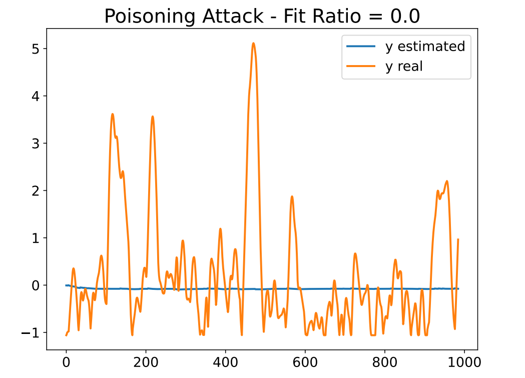
(a) 5% poisoning



(b) 10% poisoning



(c) 15% poisoning

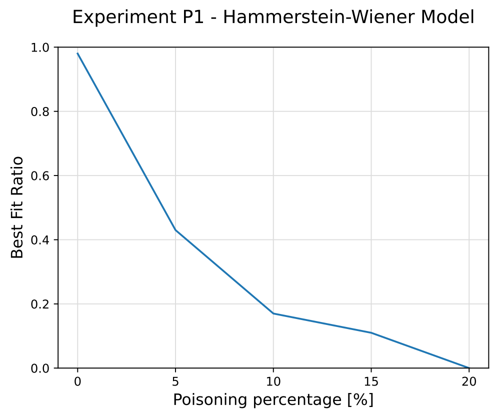


(d) 20% poisoning

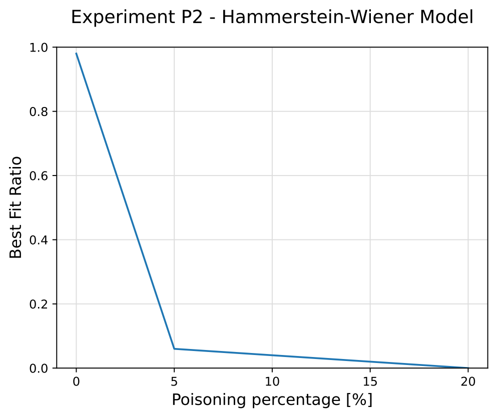
Figure 5.10: Poisoning Attack. Two Tanks model - Generic Input Signal

5.2.3 Impact of poisoning on system performance.

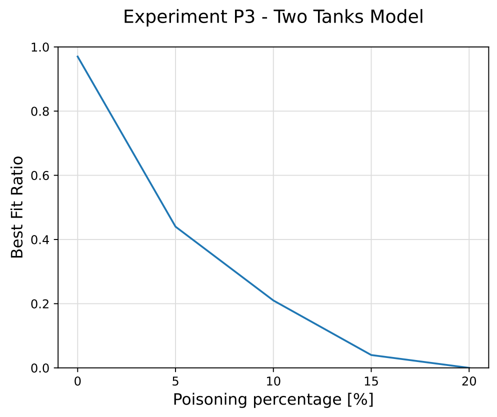
The Best Fit Ratio is a key metric that reflects how closely the model's output matches the expected output. A lower Best Fit Ratio indicated that the system's performance has degraded. We experimented with various poisoning percentages, ranging from 5% to 20%, and observed a consistent decline in this metric.



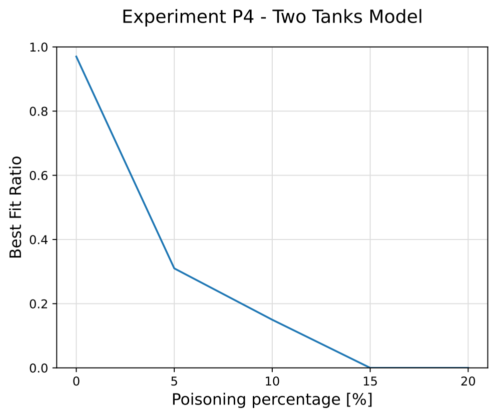
(a) P1 Experiment



(b) P2 Experiment



(c) P3 Experiment



(d) P4 Experiment

Figure 5.11: Best Fit Ratio behavior as poisoning percentage varies.

Interestingly, the experiments demonstrate that the two tanks model (Experiments P3, P4) is more resilient to data poisoning than the Hammerstein-Wiener Model (Experiments P1, P2). This could be another important reason for exploring this field since it could be interesting to know which classes of systems are more exposed to degradation in a data poisoning attack. From the analysis of the plots, we observe a clear negative correlation between the poisoning percentage and the Best Fit Ratio. Even a small percentage of poisoned data has a noticeable impact on system performance, and as the percentage increases, the degradation becomes more severe.

5.2.4 Closed-Loop Feedback and Adversarial Attack Simulation on an Inverted Pendulum

In this final section, an investigation has been conducted into the behavior of an inverted pendulum system (Fig. 5.12) under closed-loop control, focusing on the stability and performance when subject to adversarial attacks. This final contribution has been brought to test the model's properties identified through this learning architecture.

Nonlinear Inverted Pendulum Equations

The dynamics of the inverted pendulum system [7] are nonlinear and can be represented by a set of discrete-time state equations. Let the state vector at time step k be:

$$z_k = \begin{bmatrix} x_k \\ \dot{x}_k \\ \theta \\ \dot{\theta}_k \end{bmatrix}$$

and the input at time step k is u_k, where u_k represents the force applied to the cart. The system evolves over discrete time steps, so the state at time $k+1$ is updated based on the state at time k. The discrete-time state equations can be

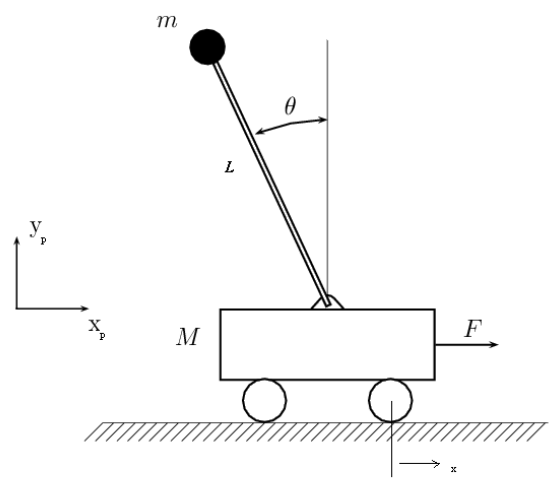


Figure 5.12: Inverted Pendulum System

summarized as:

$$\begin{cases} x_{k+1} & = x_k + \dot{x}_k \\ \dot{x}_{k+1} & = \dot{x}_k + \left(\frac{u_k + ml\dot{\theta}_k^2 \sin(\theta_k)}{M + m\sin^2(\theta_k)}\right) \\ \theta_{k+1} & = \theta_k + \dot{\theta}_k \\ \dot{\theta}_{k+1} & = \dot{\theta}_k + \left(\frac{g\sin(\theta_k) - \left(\frac{u_k + ml\dot{\theta}_k^2 \sin(\theta_k)}{M + m\sin^2(\theta_k)}\right)\cos(\theta_k)}{l}\right) \\ y_k = \theta_k \end{cases}$$

where:

- M is the mass of the cart,
- m is the mass of the pendulum,
- l is the length of the pendulum,
- g is the acceleration due to gravity

Linearization around the equilibrium point

The nonlinear system is linearized around the equilibrium point where the pendulum is upright ($\theta = 0$, $\dot{\theta} = 0$) and the cart is stationary ($x = 0$, $\dot{x} = 0$). Linearizing the system around this point allows for the application of state feedback control. The linearized state-space model is expressed as:

$$\begin{cases} z_{k+1} = Az_k + Bu_k \\ y_k = Cz_k \end{cases}$$

where A, B, C are the linearized system matrices. The following matrices are obtained:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{ml}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{g}{l}\left(1 + \frac{m}{M}\right) & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ -\frac{1}{Ml} \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$$

Open-loop validation

To assess the reconstruction accuracy of the dynamical system by the learning model, an open-loop validation was performed. For the open-loop validation, a dataset of measured inputs and outputs has been composed to train the autoen-

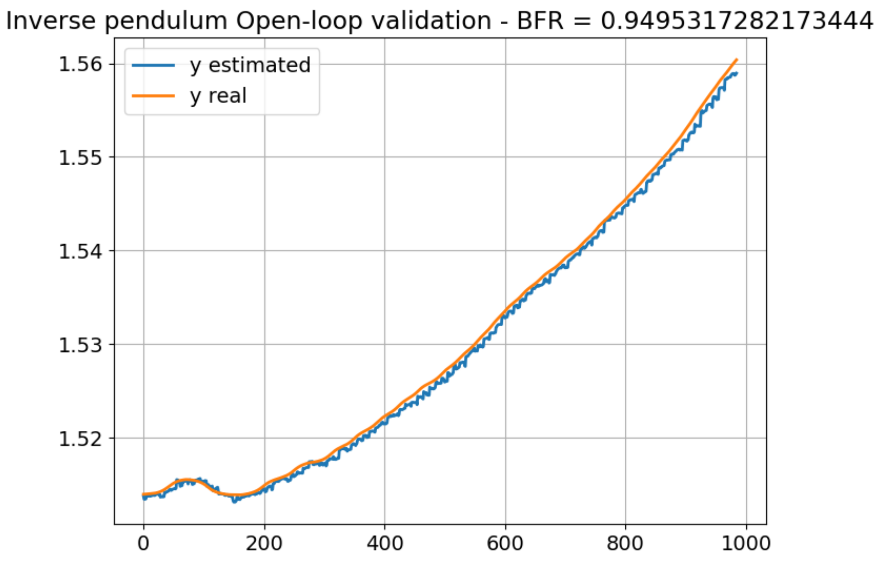


Figure 5.13: Open-loop validation of the identified inverted pendulum system

coders. The dataset consists of 20.000 samples: 10.000 input samples and 10.000 output samples, where the output is the observed pendulum angle θ_k. The input signal u_k is generated using a uniform random distribution within the range $[-5, 5]$, where the magnitude of the force applied to the cart is varied randomly. The dataset is used to train the autoencoder structure with the same hyperparameters used in the previous experiments.

The open-loop validation has been run using the following system parameters:

- Cart mass $M = 1.0$
- Pendulum mass $m = 0.1$
- Pendulum length $l = 0.5$
- Acceleration due to gravity $g = 9.81$

Fig. 5.13 shows the performances of the autoencoder structure in the identification of the system, reaching a Best Fit Ratio of 0.94.

State Feedback Control

Since the inverted pendulum system is typically unstable in open-loop conditions, the first objective is to stabilize it. In a discrete-time system, state feedback control [25] is used to maintain stability by adjusting the control input based on the system's current state. The state feedback law is expressed as:

$$u_k = -K \cdot z_k$$

where u_k is the control input, z_k is the state vector, and K is the feedback gain matrix. The feedback aims to reduce deviations from the desired state by ensuring the closed-loop poles are within the unit circle, thus stabilizing the system.

Closed-loop validation under normal conditions

In this section, we evaluate the behavior of the inverted pendulum system under closed-loop control. The autoencoder, trained on the dataset of inputs and observed outputs, is integrated with the feedback controller to maintain the pendulum's upright position. Fig. 5.14 shows the behavior of the pendulum angle θ wrapped between $-\pi$ and π. We can note that after some oscillations, the autoencoder's output stabilizes at 0 radians.

Closed-loop validation under attack

Finally, we explore the behavior of the inverted pendulum system under closed-loop control when subjected to an adversarial attack. The attack is implemented as described in subchapter 4.2.1. Thus, the control input under attack is given by:

$$u_a dv(k) = -K \cdot z(k) + \delta u(k)$$

where $\delta u(k)$ is the adversarial perturbation computed by the attack.

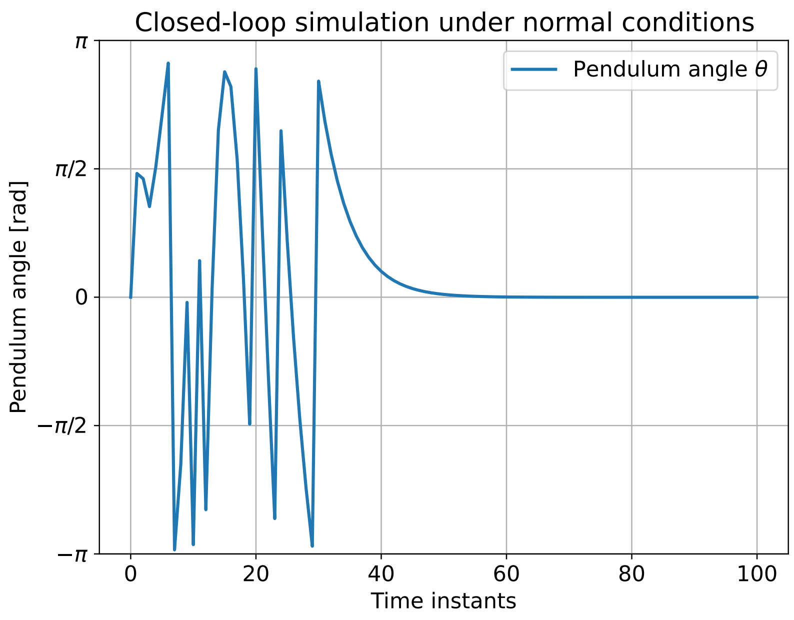
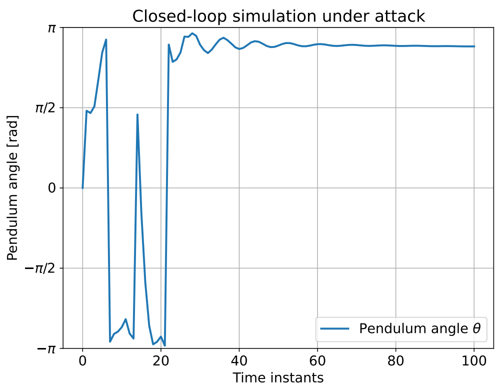
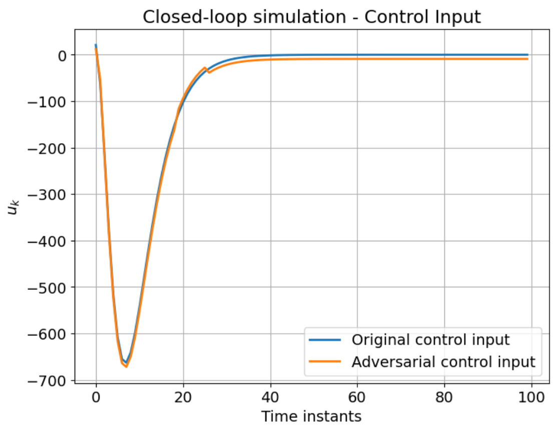


Figure 5.14: Closed-loop simulation under normal conditions.

Fig. 5.15a shows the behavior of pendulum angle θ when the system is subjected to the adversarial attack. The pendulum starts with an oscillatory behavior and then becomes unstable since it is not able to stay with the pendulum in the upright position ($\theta = 0$). Also, Fig. 5.15b shows the original control input and the adversarial one.



(a) Pendulum angle.



(b) Control input and adversarial control input.

Figure 5.15: Closed-loop simulation under attack.

Chapter 6

Conclusions

The increasing integration of machine learning into control systems has introduced opportunities and challenges. While these models provide advanced system identification and control capabilities, they are also vulnerable to adversarial attacks. This thesis examined the robustness of autoencoder-based models [13] against adversarial learning attacks, specifically targeting control systems. The investigation was motivated by the need to ensure that these systems, which often operate in critical environments, can maintain reliability and performance even when subjected to malicious perturbations.

6.1 Summary of Findings

This thesis explored the reliability of autoencoder-based models in control systems to adversarial attacks. Through a series of experiments conducted on the Hammerstein-Wiener and Two Tanks models, the research demonstrated that adversarial attacks can significantly degrade the performance of these models.

The findings showed that both gradient-based and poisoning attacks were effective in reducing the predictive capabilities of these models. In particular, the

Best Fit Ratio metric used to assess model performance exhibited a decrease as the attack intensity increased.

Additionally, the experiments revealed that the impact of adversarial attacks may depend on the complexity of the input signal and the model's architecture. For instance, the Hammerstein-Wiener model, when subjected to generic input signals, showed a more pronounced performance degradation compared to the Two Tanks model.

6.2 Limitations

While this thesis provides insights into the robustness of autoencoder-based models against adversarial attacks in control systems, it is not without its limitations. First, the experimental analysis was conducted using only two benchmark problems. While these systems are representative of common control processes, they may not capture the overall complexity of other real-world systems.

Also, the attack proposed assumes that the attacker has a good knowledge of the model. In practical scenarios, attackers might not have this level of access, thus the feasibility of the attack remains unexplored in black-box conditions. The reason why this attack is set in white-box conditions is that we are testing the robustness of the models identified through autoencoders, thus only with an open knowledge of the learning structure we can evaluate the robustness.

6.3 Future Work Directions

Looking ahead, several potential paths for future research arise from the findings of this thesis. One promising direction would be to implement an observer based on neural networks, which could estimate the system's state even if the knowledge of the model is poor.

Moreover, developing robust defense and detection mechanisms remains an essential area for future research. As adversarial attacks evolve, these defense mechanisms must keep pace to ensure the continued reliability of machine learning-based control systems.

Finally, an important direction for future work is the search for formal guarantees of robustness and resiliency. The system can maintain stability by applying resilient consensus algorithms even when some components fail or provide misleading information.

Bibliography

[1] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13*, pages 387–402. Springer, 2013.

[2] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2154–2156, 2018.

[3] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & electrical engineering*, 40(1):16–28, 2014.

[4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[5] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[6] The MathWorks Inc. Machine-learning-based identification of two-tank system, 2024.

[7] The MathWorks Inc. Pendolo inverso con animazione, 2024.

[8] The MathWorks Inc. What are Hammerstein-Wiener Models? - MATLAB & Simulink - MathWorks Italia, 2024.

[9] Weikuan Jia, Meili Sun, Jian Lian, and Sujuan Hou. Feature dimensionality reduction: a review. *Complex & Intelligent Systems*, 8(3):2663–2693, 2022.

[10] Mojtaba Kaheni, Elio Usai, and Mauro Franceschelli. Resilient constrained optimization in multi-agent systems with improved guarantee on approximation bounds. *IEEE Control Systems Letters*, 6:2659–2664, 2022.

[11] Moshe Kravchik, Battista Biggio, and Asaf Shabtai. Poisoning attacks on cyber attack detectors for industrial control systems. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 116–125, 2021.

[12] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. 2018.

[13] Daniele Masti and Alberto Bemporad. Learning nonlinear state–space models using autoencoders. *Automatica*, 129:109666, 2021.

[14] Deepti Mishra and Saurabh Sharma. Performance analysis of dimensionality reduction techniques: a comprehensive review. *Advances in Mechanical Engineering: Select Proceedings of CAMSE 2020*, pages 639–651, 2021.

[15] Gianluigi Pillonetto, Aleksandr Aravkin, Daniel Gedon, Lennart Ljung, Antonio H. Ribeiro, and Thomas Schön. Deep networks for system identification: a survey. 01 2023.

[16] Gianluigi Pillonetto, Francesco Dinuzzo, Tianshi Chen, Giuseppe De Nicolao, and Lennart Ljung. Kernel methods in system identification, machine learning and function estimation: A survey. *Automatica*, 50(3):657–682, 2014.

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

[18] Matteo Santilli, Mauro Franceschelli, and Andrea Gasparri. Secure rendezvous and static containment in multi-agent systems with adversarial intruders. *Automatica*, 143:110456, 2022.

[19] Johan Schoukens and Lennart Ljung. Nonlinear system identification: A user-oriented road map. *IEEE Control Systems Magazine*, 39(6):28–99, 2019.

[20] Maarten Schoukens and J.P. Noël. Three benchmarks addressing open challenges in nonlinear system identification. *IFAC-PapersOnLine*, 50:446–451, 07 2017.

[21] Alex Simpkins. System identification: Theory for the user, (ljung, l.; 1999)[on the shelf]. *IEEE Robotics & Automation Magazine*, 19(2):95–96, 2012.

[22] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[23] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. https://D2L.ai.

[24] K.J. Åström and P. Eykhoff. System identification—a survey. *Automatica*, 7(2):123–162, 1971.

[25] STANISLAW H. ŻAKJ and CARL A. MACCARLEY. State-feedback control of non-linear systems†. *International Journal of Control*, 43(5):1497–1514, 1986.