University of Mohamed Boudiaf - M'sila

FACULTY OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS

Serial Number: ……

Registration Number: …….

Thesis

A dissertation submitted in partial fulfillment of the
requirements for the degree of
DOCTORATE IN SCIENCES
Specialty: Electronics
Option: Control

TITLE

Contribution to the analysis of the performance of dynamic neural networks in the modeling of nonlinear systems and classification problems

Presented by

MERZOUKA Nouressadate

Presented and publicly defended on **December 17, 2025**

Examined by the jury composed of:

Full Name	Grade	Institution	Position
BENNACER Hamza	Professor	University of M'sila	President
LADJAL Mohamed	Professor	University of M'sila	Supervisor
DAACHI Mohamed El hossine	Professor	University of Sétif 1	Examiner
BEKKOUCHE Tewfik	Professor	University of Bordj Bouarreridj	Examiner
SAOUDI Kamel	Professor	University of Bouira	Examiner
OUALI Mohamed Assam	MCA	University of M'sila	Examiner
KHETTAB Khatir	Professor	University of M'sila	Guest

Academic year: 2025 / 2026

ACKNOWLEDGMENTS

In the name of ALLAH, the Most Gracious and the Most Merciful.

I would like to express my profound gratitude to Almighty Allah, who granted me the strength, patience, and perseverance to bring this doctoral work to completion.

I sincerely thank my supervisor, ***Prof. LADJAL Mohamed****, for his expert guidance and persistent support that greatly contributed to the success of this work.*

*I am also greatly indebted to the members of the defense committee****: Prof. Bennacer Hamza,*** *President of the jury****,*** *for presiding over this defense, and to* ***Prof. Daachi Mohamed Elhossine, Prof. Bekkouche Tewfik, Prof. Saoudi Kamel, Mr. Ouali Mohamed Assam, and Prof. Khettab Khatir****, for the honor they have done me by agreeing to examine this thesis, and for their pertinent remarks and suggestions which have enriched the quality of this work.*

I also reserve a special place for Professor, ***Djamel Chikouche****, to whom I owe deep gratitude for his guidance and inspiration throughout my academic journey. I pray that Allah bestows upon him health, protection, and blessings.*

I wish to express my heartfelt thanks to ***Prof. Mouloud Ayad*** *(University Ferhat Abbas of Setif),* ***Dr. Nadir Boutalbi*** *(University of Béjaïa), and* ***Dr. Abdesslam Belaout*** *(CRTI), for their invaluable assistance and scientific contributions at different stages of this research, which were essential to the success of this project.*

Furthermore, I would like to thank the University of M'sila, Faculty of Science and Technology, Department of Electronics, as well as all the faculty members, for providing me with an excellent academic environment and continuous encouragement.

Finally, I owe my deepest gratitude to my family, especially my parents, whose prayers, sacrifices, and unwavering support have been the foundation of this accomplishment.

DEDICATION

I dedicate this modest work To:

My dearest parents.

My wife and My children Khalil and Zaineb

my sisters and brothers.

All my friends, family, and colleagues.

M. Nour Essadate

TABLE OF CONTENTS

CHAPTER III : Performance Analysis For Modeling Nonlinear Pv Systems

CHAPTER IV: Fault Classification in a Nonlinear PV System: A DWT-BiLSTM Approach

LIST OF FIGURES

CHAPTER I: State of the art on neural networks for dynamic systems

CHAPTER II: Dynamic Neural Network Architectures and Their Evolution

CHAPTER III: Performance Analysis For Modeling Nonlinear PV Systems

CHAPTER IV: Fault Classification in a Nonlinear PV System: A DWT-BiLSTM Approach

LIST OF TABLES

CHAPTER III: Performance Analysis For Modeling Nonlinear PV Systems

CHAPTER IV: Fault Classification in a Nonlinear PV System: A DWT-BiLSTM Approach

NOMENCLATURE

Symbol	Definition
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
BiLSTM	Bidirectional LSTM
GRU	Gated Recurrent Unit
MLP	Multi-Layer Perceptron
NARX	Nonlinear AutoRegressive with eXogenous input
NSSM	Neural State-Space Model
PV	Photovoltaic
DWT	Discrete Wavelet Transform
NCA	Neighborhood Component Analysis
RMSE	Root Mean Square Error
R^2	Coefficient of Determination
MPPT	Maximum Power Point Tracking
I-V	Current-Voltage Characteristic

General Mathematical Symbols

Symbol	Definition
k, t	Discrete time step index
x(k)	Network input vector at time k
u(k)	System external input vector at time k
y(k)	Network/system output vector at time k
$\hat{y}$(k)	Model's predicted output vector at time k
θ	Vector of learnable network parameters (weights and biases)
J(θ)	Cost or loss function to be minimized
W	Weight matrix
b	Bias vector
σ(·)	Sigmoid activation function

tanh(·)	Hyperbolic tangent activation function
⊙	Element-wise (Hadamard) product
∇	Gradient operator

Gated RNN Symbols

Symbol	Definition
$\mathbf{h(k)}$	Hidden state vector at time k
$\mathbf{C(k)}$	Cell state vector (LSTM) at time k
$\mathbf{f_t}$	Forget gate activation vector (LSTM)
$\mathbf{i_t}$	Input gate activation vector (LSTM)
$\mathbf{o_t}$	Output gate activation vector (LSTM)
$\mathbf{z_t}$	Update gate activation vector (GRU)
$\mathbf{r_t}$	Reset gate activation vector (GRU)
$\mathbf{\overrightarrow{h}(k)}$	Forward hidden state vector (BiLSTM)
$\mathbf{\overleftarrow{h}(k)}$	Backward hidden state vector (BiLSTM)

Photovoltaic System Variables

Symbol	Definition
$\mathbf{G}$	Solar Irradiance (W/m²)
$\mathbf{T}$	Cell Temperature (°C)
$\mathbf{P_{max}}$	Maximum Power (W)
$\mathbf{V_{mpp}}$	Voltage at Maximum Power Point (V)
$\mathbf{I_{mpp}}$	Current at Maximum Power Point (A)
$\mathbf{V_{oc}}$	Open-Circuit Voltage (V)
$\mathbf{I_{sc}}$	Short-Circuit Current (A)

ملخص الرسالة

مساهمة في تحليل أداء الشبكات العصبية الديناميكية في نمذجة الأنظمة غير الخطية ومسائل التصنيف.

تتمحور هذه الأطروحة حول استقصاء المزايا التي توفرها نمذجة "الصندوق الأسود" للأنظمة الديناميكية غير الخطية، وذلك بالاعتماد على الشبكات العصبية الموصوفة ضمن إطار "فضاء الحالة" (State-space)؛ وهو نهج يختلف جوهرياً عن النماذج التقليدية القائمة على علاقة "الدخل-الخرج". وتتجلى قوة ومرونة هذا المسعى العلمي من خلال مساهمتين رئيسيتين، تم تطبيقهما على دراسة حالة للأنظمة الكهروضوئية (PV) المعقدة.

تتمثل المساهمة الأولى في تطوير منهجية متكاملة لإنشاء "توأم رقمي" (Digital Twin) عالي الدقة لنظام كهروضوئي باستخدام شبكة BiLSTM. وقد أظهرت النتائج دقة تنبؤية استثنائية (معامل تحديدR2R2 يتجاوز 0.997). والأهم من ذلك، برهنت التحليلات على الاتساق الفيزيائي للنموذج من خلال استقراء حالاته الخفية؛ مما أدى إلى الارتقاء به من نموذج "صندوق أسود" (Black-box)—الذي يقتصر على البيانات الإحصائية دون الإلمام بالميكانيزمات الداخلية—إلى نموذج "شبه فيزيائي" يُعرف بـ "الصندوق الرمادي" (Grey-box models)، الذي يدمج ببراعة بين القوانين الفيزيائية والبيانات التجريبية، مما يجعله نموذجاً قابلاً للتفسير العلمي.

أما المساهمة الثانية، فتُعالج مهمة التصنيف لأغراض تشخيص الأعطال؛ حيث تم اقتراح إطار عمل هجين ومتين يجمع بين "تحويل المويجات المتقطع" (DWT) لاستخلاص السمات، و"تحليل المكونات المجاورة" (NCA) لاختيارها، مع دمج مُصنِّف BiLSTM مُحسَّن. حقق هذا النموذج دقة تصنيف تتجاوز 96%، مبرهناً على كفاءته العالية في التمييز بين الأعطال ذات البصمات الكهربائية المتشابهة جداً.

خلاصة القول، تُفضي هذه الأطروحة إلى تأكيد صحة الفرضية القائلة بأن نهج "فضاء الحالة" يتفوق في نمذجة الأنظمة المعقدة، لقدرته على الجمع بين الدقة العالية والتماسك الفيزيائي الواضح. كما تُثبت النتائج المرونة الملحوظة للشبكات العصبية الديناميكية في التعامل مع مهام النمذجة والتصنيف على حد سواء، مما يفتح آفاقاً واعدة لتطبيقات الصيانة التنبؤية والتحكم الذكي في أنظمة الطاقة.

الكلمات المفتاحية :الشبكات العصبية الديناميكية، BiLSTM، نمذجة الصندوق الأسود، فضاء الحالة، النظام الكهروضوئي، تحديد الأنظمة، تصنيف الأعطال.

ABSTRACT

Contribution to the analysis of the performance of dynamic neural networks in the modeling of nonlinear systems and classification problems.

This doctoral work explores the advantages of "black-box" modeling for nonlinear dynamic systems by using neural networks within a state-space framework, an approach that stands apart from traditional input-output models. The thesis demonstrates the power and versatility of this approach through two main contributions, both applied to the complex case study of photovoltaic (PV) systems.

The first contribution focuses on the task of modeling. We developed a complete methodology to create a high-fidelity "digital twin" of a PV system using a BiLSTM network. The results show exceptional predictive accuracy (with an R^2 score over 0.997) and, more importantly, physical coherence, which we validated by analyzing the model's hidden states. This approach turns the "black box" into an interpretable "grey box."

The second contribution tackles the task of classification for fault diagnosis. We proposed a robust hybrid framework that combines DWT for feature extraction, NCA for feature selection, and an optimized BiLSTM classifier. This model achieves a classification accuracy of over 96%, showing its superiority in distinguishing between faults with very similar electrical signatures.

In summary, this thesis validates the hypothesis that the state-space approach is superior for "black-box" modeling, as it facilitates the development of models that combine precision with a clear physical interpretation. It also confirms the remarkable versatility of dynamic neural networks for both modeling and classification tasks, opening up essential avenues for predictive maintenance and the intelligent control of energy systems.

Keywords: *Dynamic neural networks, BiLSTM, Black-box modeling, State-space, Photovoltaic system, System identification, Fault classification.*

RESUME

Contribution à l'analyse des performances des réseaux de neurones dynamiques dans la modélisation des systèmes non linéaires et les problèmes de classification.

Ce travail de doctorat explore les avantages de la modélisation "boîte noire" de systèmes dynamiques non-linéaires en utilisant des réseaux de neurones décrits dans un cadre d'espace d'état, une approche qui se distingue des modèles entrée-sortie traditionnels. La thèse démontre la puissance et la polyvalence de cette approche à travers deux contributions majeures, appliquées au cas d'étude complexe des systèmes photovoltaïques (PV).

La première contribution se concentre sur la tâche de modélisation. Une méthodologie est développée pour créer un "jumeau numérique" haute-fidélité d'un système PV à l'aide d'un réseau BiLSTM. Les résultats montrent une précision prédictive exceptionnelle ($R^2 > 0,997$) et, plus important encore, une cohérence physique validée par l'analyse des états cachés du modèle, transformant la "boîte noire" en une "boîte grise" interprétable.

La seconde contribution aborde la tâche de classification pour le diagnostic de pannes. Un framework hybride robuste est proposé, combinant l'extraction de caractéristiques par DWT, la sélection par NCA, et un classifieur BiLSTM optimisé. Ce modèle atteint une précision de classification de plus de 96%, démontrant sa supériorité pour distinguer des défauts aux signatures électriques très similaires.

En synthèse, cette thèse valide l'hypothèse que l'approche en espace d'état est supérieure pour la modélisation "boîte noire", en permettant la création de modèles à la fois précis et physiquement cohérents. Elle confirme également la polyvalence remarquable des réseaux de neurones dynamiques pour les tâches de modélisation et de classification, ouvrant des perspectives importantes pour la maintenance prédictive et le contrôle intelligent des systèmes énergétiques.

Mots-clés : *Réseaux de neurones dynamiques, BiLSTM, Modélisation boîte-noire, Espace d'état, Système photovoltaïque, Identification de systèmes, Classification de défauts*

GENERAL INTRODUCTION

General Introduction

In the last ten years, artificial intelligence, especially Deep Learning, has gotten much better. Researchers now widely agree that neural networks are effective for many engineering tasks, including signal processing, system identification, machine control, and rapid fault detection. Because they continue to prove useful, researchers have been motivated to improve them. Today, newer models help us understand how neural networks actually work inside and let us use them in many new ways **[1]**.

While identifying linear systems is now a well-established discipline, identifying nonlinear dynamic systems is still a major scientific challenge **[2]**. This happens mainly because of their inherent complexity, showing behaviors like saturation, limit cycles, and chaos **[3]**, and the difficulty traditional methods have in precisely capturing long-term time dependencies within the internal states. Historically, most methods used to tackle this problem relied on "black-box" approaches, where the model learns the relationship between inputs and outputs without trying to describe the internal mechanisms **[1]**. Although these methods often give good results, they leave one crucial question unanswered for advanced control and diagnostics: What internal mechanism truly governs the system’s dynamics?

The main goal of this doctoral research is to fill this gap by studying the benefits of a more structured modeling framework: dynamic neural networks set up in a state-space representation. This approach specifically includes the idea of the system's internal state. It opens the door to models that are not only accurate and robust against noise, but also physically consistent and interpretable, effectively turning the opaque "black box" into a "grey box." Furthermore, this work also checks how well these models perform on classification tasks, proving their versatility and practical scope for intelligent diagnostics.

To base this study on a real challenge, the photovoltaic (PV) system was chosen as the main case study, due to its dynamic complexity and high technological importance. PV systems are key pillars of the energy transition, but their behavior is highly nonlinear. Their performance relies on detailed semiconductor physics and constantly changing external factors, such as solar irradiance, cell temperature, and the health status of the modules. This complex interaction makes modeling and diagnostics very difficult. Classical physics-based "white-box" models (for example, the single-diode model) quickly reach their limits. Finding their parameters is tough **[2]**, and these models cannot properly capture how

the system changes dynamically or when operating conditions shift, especially if a fault is active. Therefore, data-driven system identification (the "black-box" approach) is now a vital alternative for precisely modeling these complex systems in the real world.

Dealing with the dynamic challenge of sequential data requires the use of Recurrent Neural Networks (RNNs), they are the standard tool. Newer versions, such as the Long Short-Term Memory (LSTM) and the Bidirectional Long Short-Term Memory (BiLSTM), achieve impressive results. They are built precisely to capture the intricate time dependencies needed when working with state functions. The effectiveness of these dynamic models in controlling and diagnosing systems is also well-documented **[4,5]**. Although newer designs, like Transformers and state-based models like Mamba, are excellent at handling very long-term dependencies and optimizing efficiency, they generally require large amounts of data to learn properly. Given that our dataset is structured but of medium size (typical of laboratory test benches), the BiLSTM proves to be the best practical compromise, offering high stability and superior learning capabilities for our specific tasks.

Within PV research, neural network applications have generally followed two main paths, both based on Input-Output modeling. Hybrid models, for example, CNNs combined with LSTMs, are very effective in the first main area: forecasting power output. The second area of focus is the characterisation of static Current-Voltage (I-V) curves. Researchers often employ neural networks or neuro-fuzzy systems in this area to estimate key electrical parameters, which are crucial for tasks such as MPPT **[6]**. However, a shared and major limitation of these Input-Output methods is their structural inability to capture the system's full internal dynamics truly. This severely limits their usefulness for getting deeper physical insights and for developing advanced diagnostics.

Compared to Input-Output models, the state-space framework is much more structured and powerful. Early studies quickly demonstrated the value of using neural networks in this context, highlighting benefits such as simpler models and their suitability for robust control **[7**,**8**,**9]**. The state-space method works by trying to determine the system's internal mechanisms, rather than just memorising the Input-Output mapping. Due to deep learning, this method is gaining renewed attention. Specifically, deep state-space models are designed to follow complex patterns across long sequences of time **[10]**. This is particularly relevant for PV systems, where a state-space formulation is considered necessary to capture full dynamic behaviors **[11].**

Despite this strong theoretical backing, a critical gap remains in the literature: few studies apply a data-driven state-space approach to identify the complete I-V curve of a PV system, and even fewer validate the physical meaning of the learned internal state representation.

Facing this dual challenge, modeling dynamics with physical meaning and classifying faults robustly, this thesis provides two major contributions that complement each other through the use of dynamic neural networks:

1. Modeling and Identification of a PV Digital Twin using State-Space: Our first contribution focuses on system identification. We present a complete method to build a high-fidelity "digital twin" of a PV system using a Bidirectional LSTM (BiLSTM) network set up in an implicit state-space framework. By treating the complete Current-Voltage (I-V) curve identification as a dynamic sequence, we demonstrate that the network's internal hidden state (h(k)) learns a structured and physically coherent representation of the system's dynamics. This process successfully transforms the opaque "black box" into an interpretable "grey box." We validate this physical coherence by successfully extracting key fundamental electrical parameters (such as P_max,V_mpp, and I_sc) from the predicted I-V curves.

2. Robust Fault Diagnosis via Multi-Scale Hybrid Classification: Our second contribution explores the versatility of dynamic networks for advanced classification and fault diagnostics in PV systems. We developed a robust hybrid framework that is especially effective at distinguishing faults with very similar electrical signatures. This framework combines multi-scale feature extraction using the Discrete Wavelet Transform (DWT), smart and focused attribute selection with Neighborhood Component Analysis (NCA), and an optimized BiLSTM classifier (tuned using a Bayesian approach). We prove the superiority of this bidirectional hybrid approach for accurate and robust classification, which is vital for predictive maintenance.

This document is organized as follows:

Chapter 1 lays the theoretical groundwork for our study. It begins by looking at the challenges of modeling nonlinear dynamic systems, highlighting why traditional physics methods often fall short. It then introduces our central case study, the PV system, detailing its physical principles, frequent defects, and explaining why it is an ideal and complex benchmark for our work.

Chapter 2 provides a detailed analysis of the dynamic neural network architectures, the primary tools of this thesis. It traces their evolution from classical input-output models (like NARX) and structured state-space models (NSSM) to the more flexible Recurrent Neural Networks (RNNs). This chapter finishes with a complete review of the newest and most modern gated network architectures, justifying the choice of the Bidirectional LSTM (BiLSTM) for our contributions, and setting the general training framework for these models.

Chapter 3 details our first major contribution: designing and validating a high-fidelity "digital twin" for a PV system. We explain how we used a state-space BiLSTM framework to model the complete I-V curve as a dynamic sequence. The results shown in this chapter demonstrate exceptional predictive accuracy and the physical coherence of the model's learned internal state, transforming the "black box" into an interpretable "grey box."

Chapter 4 presents our second major contribution: a reliable hybrid system built to classify faults accurately. We explain in detail how our method combines feature extraction at different scales using the Discrete Wavelet Transform (DWT) with an optimized BiLSTM classifier. The results demonstrate the framework's high accuracy and its superior robustness compared to other state-of-the-art methods, especially for distinguishing between faults with very similar signatures.

Finally, we conclude the thesis with a General Conclusion, where we summarize the main results achieved for both the modeling and classification approaches. We also offer practical recommendations for researchers and outline promising directions for future work. This conclusion synthesizes all our observations and draws the key findings of this research.

CHAPTER I

State of the art on neural networks for dynamic systems

I.1 Introduction

Artificial intelligence has changed many areas of science and industry, particularly in electronics and control engineering **[1]**. Because of this, understanding, predicting, and controlling complex systems remains a significant challenge for innovation **[2].** For many years, traditional mathematical models grounded on well-known physical laws have been reliable and robust tools **[3]**. However, these methods now show their limits when dealing with strongly nonlinear dynamics **[3**,**4],** unpredictable behaviors **[5]**, or systems for which writing an exact equation is simply impossible **[6].**

It is in this setting that deep learning methods have emerged, bringing a fundamental shift in how we approach modeling **[1]**. Instead of starting from theoretical assumptions, artificial neural networks learn directly from raw data, building functional "black-box" models capable of handling complexities that were out of reach before **[7**, **8]**.

This chapter provides the foundational knowledge for our study by exploring two key areas. First, we will build a comprehensive overview of the "solution space" by detailing the evolution of neural networks for dynamic systems. Second, we will establish the "problem space" by conducting a deep dive into our chosen case study, the photovoltaic system.

Our exploration of the neural network landscape will move step-by-step. We'll explain what nonlinear systems are and why traditional models have a hard time describing them properly. Then, we will take a look at how neural networks have developed, starting with static models like the MLP, and see why they struggle with data that changes over time. This path will naturally lead us to Recurrent Neural Networks (RNNs), which were developed to overcome the problems faced by earlier models, particularly in learning long-term temporal dependencies. We will look at practical applications of these networks in modeling, system identification **[9]**, and sequence classification.

Then, to ground these tools in a real-world challenge, we will conduct a detailed review of our chosen case study: the photovoltaic (PV) system. We will analyze its physical principles, its "white-box" models, and the common fault modes that make it a difficult system to model and diagnose.

Finally, we will talk about current research challenges focused on making these networks, often called "black boxes", more reliable, robust, and easier to understand. This

discussion will highlight gaps in the existing literature and clearly position the original contributions of our study **[10]**.

I.2 Modeling nonlinear dynamic systems

I.2.1 Definition, Ubiquity, and Complexity of Nonlinear Systems

What makes a nonlinear dynamic system different is that the connection between its inputs and state variables isn't proportional, so the principle of superposition doesn't apply. All dynamic systems follow a set of rules as they evolve. But what sets nonlinear systems apart is their lack of proportionality. This key difference is why they can show much richer and more complex behaviors than linear systems **[1,2]**.

These behaviors include sudden regime changes (bifurcations), such as the saturation of a magnetic core in an electrical machine; stable oscillations (limit cycles), which are typical of many electronic circuits; and deterministic chaos, where drastically different responses can emerge from tiny changes in initial conditions. This complexity is not just a mathematical curiosity, but a fundamental characteristic of how many technological and natural systems work **[3]**.

I.2.2. Illustrative Examples of Nonlinearity

I.2.2.1 Weather chaos and the butterfly effect

One of the most famous examples of sensitivity to initial conditions was highlighted by meteorologist Edward Lorenz. He found that in his weather prediction models, even a tiny change in the initial data (down to the sixth decimal place) could lead to entirely different forecasts within just a few days. This phenomenon, known as the "butterfly effect," shows how a slight change in a nonlinear system can have enormous and unpredictable consequences, making long-term predictions very hard **[3]**.

I.2.2.2 Nonlinearity in electrical and electronic systems

Electrical and electronic systems also contain many nonlinear behaviors.

***a- Chua's chaotic oscillator* :**

This is a classic example of a nonlinear system. A straightforward electronic circuit, made up of a few passive components and a single nonlinear component (Chua's diode),

can generate complex and chaotic oscillatory signals. This circuit shows that even a simple nonlinearity is enough to create unpredictable behavior in a deterministic system.

b-Saturation in machines and converters

In power electronics and electrical engineering, magnetic saturation is a pervasive nonlinearity. In an asynchronous machine, the relationship between the excitation current and magnetic flux is linear only within a limited range. Once the ferromagnetic material reaches a certain point, it becomes saturated, and the machine's behavior changes suddenly. Likewise, in a power converter, when an inductor saturates, it can cause a rapid and dangerous increase in current, turning the system from stable to unstable **[4-5]**.

c- Behavior of semiconductor components

The most fundamental element of modern electronics, the transistor (or diode), has a current-voltage characteristic that is inherently exponential, and therefore strongly nonlinear. This nonlinearity is what allows amplification and switching, but it also makes the analysis and precise modeling of analogue circuits particularly complex.

d- Nonlinearity in Photovoltaic (PV) Systems

The reason photovoltaic (PV) systems are the focus of our thesis is their fascinating and complex nonlinear behavior. At its core, the nonlinearity comes from the exponential current-voltage (I-V) relationship within their P-N junctions. But what makes PV systems truly challenging is that this behavior isn't static. The I-V curve itself shifts constantly and unpredictably with changes in sunlight and temperature, a major hurdle for researchers in this field. For example, researchers such as May et al**.[6]** are designing sophisticated controllers for grid-connected photovoltaic systems specifically to manage the DC-link voltage fluctuations that arise directly from these changing conditions **[6].** This dual nonlinearity, both internal to the device and dependent on its environment, makes the modeling, control (especially for maximum power point tracking, or MPPT), and fault diagnosis of photovoltaic systems extremely complex. It is precisely this complexity and the need to manage its impact on power quality and system stability that make photovoltaic systems an ideal case study for the advanced modeling approaches developed in this work.

This strong and rich dynamic behavior causes complex challenges for control and prediction in many vital fields. For example, energy production systems, robotics, and financial markets rely heavily on accurate models to make good decisions and optimize

results. Because nonlinear dynamic systems show complex behaviors like multiple steady states, oscillations, and chaos, they need sophisticated modeling and estimation methods to accurately represent them in real-world situations As these systems are ubiquitous across various scientific and engineering domains, the development of reliable models is essential for improving performance and safety in real-world scenarios **[7].**

These examples, whether from nature or technology, all point to the same conclusion: in nonlinear systems, minor variations can lead to significant and unpredictable changes. The main takeaway here is that nonlinearity is the rule, not the exception **[1]**. Because of this, standard linear methods are often not up to the task of modeling these systems. We need to turn to tools that are specifically designed to handle their complex and frequently surprising behaviors **[8,9]**.

I.2.3 Principles of Dynamic System Modeling

I.2.3.1 Defining the Modeling Process

Modeling a process consists of finding a mathematical description of its operation, which accounts for the relationships between its inputs and outputs**.** This explanation simply shows how what you put into the system (the 'inputs') links up with what you get out of it (the 'outputs'). The exact kind of math description you end up with just depends on how you decide to define that connection.

I.2.3.2 Classification of Models

To better understand the landscape of available models, it is helpful to classify them according to two fundamental axes: the nature of the knowledge used in their design, and the specific task they are built to perform.

a - Classification According to Design Method

Models can be put into three main groups, depending on how they are built:

- Knowledge Models (White Box):

These models are built when you have an obvious and deep understanding of how a system works from the inside... These equations tell us exactly how the system behaves. So, it's about understanding why things happen and putting that knowledge straight into the model **[2],[10]**. Their strength lies in their high interpretability and extrapolation capabilities, but they are often difficult to formulate and solve for complex systems.

- ***Behavioral Models (Black Box)***

In contrast, these models are created when you don't know what's happening internally within a system. You "train" the model to find the best numbers (parameters) to fit the observed data, without worrying about the underlying physical reasons **[1,2,11]**. This approach is highly flexible and powerful but often suffers from a lack of interpretability and a higher risk of overfitting.

- ***Hybrid Models (Gray Box)***

These models bridge the gap between "white box" and "black box" approaches. You can use what science already knows and also use real data to complete the missing parts or improve what you don't fully understand about the system **[1,4]**. This offers a practical balance, leading to models that are both more accurate than purely physical ones and more interpretable than purely black-box ones.

These three approaches are synthesized in Figure 1.1, which provides a conceptual representation of the various modeling approaches. While white-box models primarily draw upon an intrinsic understanding of the system and black-box models are built from statistical insights derived from data, gray-box modeling illustrates how these two methodologies can be combined**.**

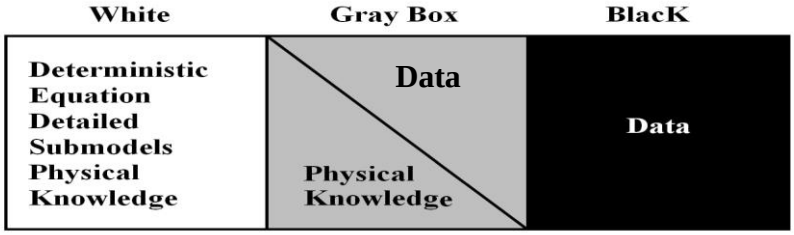

Figure I.1: Conceptual representation of various modeling approaches **[10]**

b- Classification Based on How Models Are Used

A model can be built to do one of two main things: it can act as a predictor or as a simulator. A predictor model works right next to your real system... A simulator, on the other hand, is a system that behaves like the real process but can run independently to test new control devices or to explore operating conditions that would be too expensive or dangerous to test on the real process **[12,13]**. In this thesis, we're developing models that can do both – they can predict future outputs and also simulate different scenarios.

I.2.3.3 The Model Design Process

As we have seen, a wide variety of model types exist. The process of building a specific model for a given task can be broken down into two fundamental stages: structural design and parameter identification **[12].**

The first stage is to select a plausible "hypothesis model." This involves making several key choices about the model's structure—such as whether it should be static or dynamic, linear or nonlinear, input-output or state-based—based on prior knowledge of the system's behavior. For instance, if precise physical laws are known, a white-box structure might be chosen; if not, a black-box approach becomes necessary **[10]**.

The second stage, known as identification, is to estimate the model's parameters. To find the best parameters, we use an optimization process driven by a learning algorithm **[14]**. The goal is to minimize a 'cost function'—a metric that quantifies the discrepancy between the model's output and the actual data. Ultimately, the quality of the final model hinges on three key factors: choosing the right hypothesis model, having access to rich training data, and using an effective learning algorithm **[2]**.

I.2.3.4 Key Choices in Hypothesis Model Selection

Based on the design process described above, several critical choices must be made to define the hypothesis model.

a- Static or Dynamic Model

A model is called 'static' if it just shows an instant connection between what goes in and what comes out. It's good for predicting steady situations, without worrying about time. But if a model uses equations that show how things change over time (like differential equations for continuous changes or recursive ones for step-by-step changes), then it's called 'dynamic'. These models are all about how the system evolves **[2,14,15].**It is this second category that is of interest in this thesis.

A static model describes an instantaneous relationship, Generally, for a static system, the output $y(t)$ at any given time t depends solely on the input u(t) at that same instant, expressible as :

$$y(t)=f(u(t)) \quad \text{(I.1)}$$

where f is a function that defines this direct relationship. An example is Ohm's Law:

$$U=R*I \quad \text{(I.2)}$$

In contrast, a dynamic model incorporates time-evolution. For a continuous-time system, this often involves differential equations:

$$\frac{dy}{dt} = f(y,u) \quad \text{(I.3)}$$

Where *y(t)* represents the system's output variable or state at time *t*, *u(t)* is the system's input variable at time *t*, and f is the (often nonlinear) function that describes how the output variable evolves based on its current state and the input.

For a discrete-time system, recursive or difference equations are used:

$$y_{k+1}=f(y_k,u_k) \quad \text{(I.4)}$$

Here, y_k is the system's output variable or state at discrete time step *k*, uk is the input variable at time step *k*, and f is the (often nonlinear) function that determines the future state y_{k+1} from the current state y_k and the input u_k.

b- Linear or Nonlinear Model

Most real-world processes would require nonlinear models to be described accurately across their entire operating range. In contrast, linear models are only valid approximations within a more or less restricted domain. It is therefore crucial to build a nonlinear model that captures the system's dynamics, not only around its usual operating points but also during transitions from one operating point to another **[14].**

c- Input-Output or State-Space Representation

To model a dynamic system, we can act like two different kinds of observers. The first approach, the input-output model, is like a pure "black-box" observer. It doesn't care what's happening inside the process (system) studied; it simply tries to find a rule, h that connects what went in (past inputs and outputs) to what comes out (the next output). Its general form can be written as :

$$y_p(k)=h\left(y_p(k-1),...,y_p(k-n),u(k),...u(k-m),b(k),...b(k-p)\right) \quad (I.5)$$

Where :

h :is a nonlinear function, *n* and *m:* are the orders of the model, and b(k) represents noise.

The alternative, the state-space model, is more like a mechanic. It assumes there's an internal "state," x(k), that we can't see directly. The goal is then to figure out two things: first, how the state changes from one moment to the next, and second, how that internal state produces the final output we see.

A State Equation, which describes how the internal state evolves over time:

$$x_p(k+1)=f\left(x_p(k),u(k),b1(k)\right) \quad (I.6)$$

An Observation Equation, which describes how the observable output is generated from the current state:

$$y_p(k)=g\left(x_p(k),u(k),b2(k)\right) \quad (I.7)$$

The objective of system identification in this framework is to find suitable approximations for the two nonlinear functions,f and g **[2], [14].** As will be explored throughout this thesis, state-space models can often lead to more compact and robust predictors than their input-output counterparts.

I.2.4 The Limits of Traditional Models for Nonlinear Dynamics

In engineering, we have standard ways of building models, but they run into major issues when dealing with complex, nonlinear systems. These approaches typically fall into two main groups: 'white-box' models, which are created from the laws of physics, and statistical models, which are based on analyzing past data.

I.2.4.1. Models Based on Differential Equations (The "White-Box" Approach)

The conventional approach to modeling physical systems is through differential equations. By applying first principles—such as the laws of mechanics or electromagnetism—a "white-box" model can be constructed to describe the system's

evolution. So, this model's ability to predict things relies entirely on whether we can solve those equations. However, this method frequently runs into some significant difficulties:

a- The Challenge of Finding an Exact Solution

For linear systems, solutions are often well-defined. But for nonlinear systems, finding an exact, analytical solution is rarely possible, usually becoming an intractable problem **[16]**. So, we have to turn to numerical methods instead. But these can cost a lot in computer power, and even tiny changes at the very beginning can make a big difference in the results.

b- The Compromise of Linearization

A common practice to manage this complexity is to linearize the model around a specific operating point. While this technique works for looking at local stability or designing controls, it just can't show the system's full, complex overall behavior. It masks the very essence of nonlinearity, such as limit cycles, bifurcations, and chaotic attractors **[3]**.

***c- Practical Difficulties in Formulation*:**

Furthermore, building a complete physical model can be prohibitively complex if the system's physics are not fully understood or if specific effects (like friction, thermal drift, or component aging**)** are hard to formalize mathematically. In many industrial settings, creating a comprehensive "white-box" model is simply unfeasible **[12]**.

I.2.4.2 Statistical Time-Series Models (The "Linear Black-Box" Approach)

When a white-box model is not viable, engineers turn to statistical models that treat the system as a "black box." Among these, the ARMA (AutoRegressive Moving Average) family and its derivatives are benchmark tools for time-series analysis **[17,18].** You'll find these models work well for linear systems, but they come with a couple of very rigid assumptions, and those often don't fit what happens in the real world:

a- The Linearity Assumption

These models presuppose that the current output is a linear combination of past inputs and outputs. They struggle to model real-world scenarios where **behaviors** are nonlinear, for instance, when a component reaches its operational limit (saturation), a system's behavior shifts suddenly instead of smoothly, or a variable begins to increase at an exponential rate **[18].**

b- The Stationarity Assumption

They also assume that the statistical properties of the time series—such as its mean and variance—remain constant over time. This is often untrue in practice. Transient behaviors, shifting operating conditions, or component aging introduce non-stationarity that ARMA models cannot handle natively without complex data preprocessing **[19]**.

These fundamental limitations highlighted the need for a more flexible approach. The advent of deep learning has provided powerful alternatives to overcome the linearity and stationarity assumptions of classical time-series models **[20]**.

I.3 Sequential classification and its applications

I.3.1. Definition

A sequential or temporal classification problem is all about assigning a class label to an entire data sequence or each of its individual parts. What's absolutely key is that the order of the data matters a great deal, because it contains vital information for making a good decision. Unlike static classification, where data is handled one by one (like classifying a picture), sequential classification is used for time-dependent data such as time series or signals **[21,22].**

I.3.2 Applications of Temporal Classification

The way we understand a sentence is all about the order of the words. The same is true for many real-world systems; their data has to be seen as a sequence to be understood. This is what temporal classification is all about—it's a technique used widely in science and industry to find patterns in data that has a specific order.

Let's look at a few examples:

a- Medical Diagnostics

Doctors use this to analyze ECG signals and tell if a heartbeat is 'normal' or 'arrhythmic'. Instead of just looking at one part of the signal, new methods like the one from Chen et al. **[23]** can cleverly pay attention to multiple parts of the signal at once, which helps make a much more accurate diagnosis.

b- Industrial Fault Detection

This helps figure out if a machine is running normally or about to break down, just by listening to its vibrations **[24]**. The cool part is that modern techniques using deep learning can learn what a "bad" vibration sounds like on their own, directly from the data, giving an earlier warning than older methods.

c- Speech Recognition

This is how our phones and smart speakers understand us. Early on, researchers like Hinton and Graves **[25]** showed that a special type of network, a deep RNN, was amazing at this. It's because these networks have a kind of memory that helps them understand the context of a whole sentence, not just individual words.

These same concepts are key to my thesis, especially since in electronics and electrical engineering, we're constantly working with signals that change over time. Here's how these ideas apply directly to our field:

d- Fault Diagnosis in Electrical Machines:

A prevalent method is Motor Current Signature Analysis (MCSA). As the review by Benbouzid **[26]** explains, the idea is to look at the electrical current of a motor like a sound signature. When something is wrong, like a broken part, it adds tiny, specific "notes" (or harmonics) to this signature that we can detect.

e- Condition Monitoring:

Instead of just one sensor, we now often use data from several (like temperature, vibration, and current) to get a complete picture of a machine's health. The work by Mariani et al. **[27]** shows how we can combine all this sensor data to decide if a system is healthy, getting old, or in immediate danger.

f-Anomaly Detection in Electronic Circuits:

Manually checking circuit signals on an oscilloscope for tiny problems is slow, and you can miss things. That's why researchers like Narra et al. **[28]** are using networks with memory (LSTMs) to scan these signals automatically. The network learns to spot tell-tale signs of trouble, like weird oscillations, much faster and more reliably than a human can.

g-Fault Detection in Photovoltaic (PV) Systems:

This is a crucial application, as faults in PV arrays (like shading, short circuits, or degradation) can significantly reduce energy production. Researchers are developing robust methods, such as hybrid approaches combining signal processing (like DWT) with deep learning models (like BiLSTM), to automatically classify the operational state of PV systems from their voltage and current data **[29]**.

I.3.3 Classical Approaches and Their Limitations

For a long time, the field of temporal classification was shaped by a handful of classical methods. These were the workhorses of early research, proving very effective, especially when the data was relatively clean. Their main weakness, however, became apparent when they were applied to real-world problems, where signals are often complex and noisy. Two of the most common approaches, HMMs and SVMs, particularly illustrate these shortcomings.

I.3.3.1 The Probabilistic Approach: Hidden Markov Models (HMMs)

Many real-world systems reveal only their outputs, not the internal mechanisms. Hidden Markov Models (HMMs) were developed to solve this kind of issue. An HMM is a statistical framework that allows us to work backward, inferring the most likely sequence of hidden, unobservable states that could have generated the sequence of data we see. Rabiner's foundational tutorial provides a detailed mathematical explanation of this inference process **[30]**. This made HMMs the go-to technology for applications like speech recognition for decades. However, HMMs are constrained by two critical limitations.

To make the math manageable, HMMs rely on a simplifying rule known as the '**Markov assumption**': the idea that the future only depends on the present, not the entire past. While this makes the model efficient, it comes at a steep price. The model is left with a **very short memory**, rendering it unable to learn from long-range dependencies in the data. Second, HMMs perform poorly with raw, messy data. They require a careful and time-consuming pre-processing step known as **"feature engineering"** to extract the most essential features from a signal, and the model's overall performance is heavily dependent on the quality of this manual extraction.

I.3.3.2 The Static Classifier Workaround: Support Vector Machines (SVMs)

Another powerful classical method is the Support Vector Machine (SVM), introduced by Cortes and Vapnik as a classifier for individual, independent data points **[31].** Being inherently non-sequential, applying an SVM to time-series data requires a workaround. The standard "trick" is to convert each sequence into a single, fixed-size feature vector by calculating summary statistics like its average, maximum, and standard deviation.
This approach, however, introduces its fundamental flaws. The most significant is the **destruction of temporal information**; by "squashing" the sequence into a single vector, all the crucial information stored in the order and timing of the data is lost. Furthermore, just like with HMMs, the model's success is entirely dependent on **manual feature extraction**. The model doesn't learn from the sequence itself, but only from the statistical features that the user has chosen to represent it.

Ultimately, both HMMs and SVMs were missing two key ingredients for handling real-world sequences: a way to manage long-term memory and the ability to learn directly from raw data. This created a clear opening for a new class of models. Neural networks, especially those with a recurrent design, were developed specifically to provide these missing capabilities **[32]**. Early work on Time-Delay Neural Networks (TDNNs) already showed the promise of using neural architectures for sequential tasks like phoneme recognition **[33]**.

I.4 Neural Networks as a solution: From Static to Dynamic

I.4.1. Introduction to Artificial Neural Networks

Given that both physical (white-box based on differential equations) and linear statistical models (like ARMA) often prove inadequate for elegantly handling complex nonlinear dynamics, Artificial Neural Networks (ANNs) have emerged as a highly promising alternative. Thanks to their inherent ability to capture complex correlations between inputs and outputs, as well as their universal approximation power for nonlinear functions, neural networks have established themselves as a highly effective tool for modeling a wide variety of dynamic systems **[20]**. These models, inspired by the human brain, learn directly from data to model complex nonlinear relationships for tasks like classification and prediction .This flexibility proves particularly valuable where traditional

approaches struggle to account for the inherent complexities of real-world temporal processes. Among the various ANN architectures, the most classic and foundational example in this context is the Multi-Layer Perceptron (MLP).

I.4.2 The Multi-Layer Perceptron (MLP): A first neural approach

An MLP is fundamentally a 'feedforward' network. This architectural design simply means that information flows unidirectionally: it starts at an input layer, progresses through one or more hidden layers where computations are performed, and finally reaches an output layer. This simple yet powerful structure is illustrated in Figure 1.2.

The figure illustrates the basic architecture of an MLP, showing the input layer receiving signals x1,x2,…,xn with synaptic weights w, a hidden layer where the signals are summed (∑) and passed through an activation function, and finally an output layer producing the result Y based on the hidden layer's output w and a bias w_0).

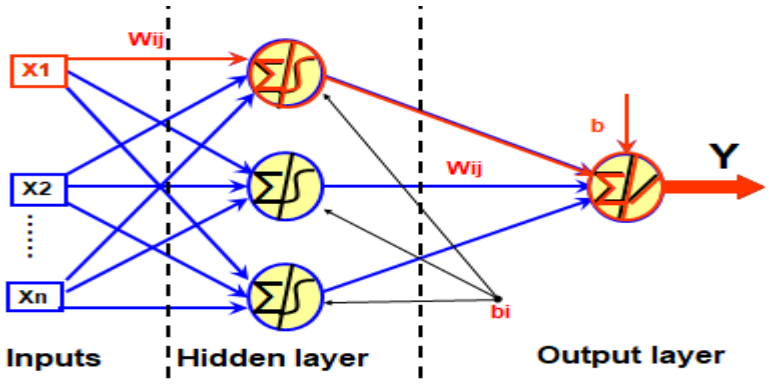

Figure I.2: Schematic Representation of a Multi-Layer Perceptron (MLP) with a Single Hidden Layer.

I.4.2.1 Mathematical Formulation

To understand its operation, a mathematical formulation is essential. Each neuron in the network processes its inputs by computing a weighted sum, adding a bias, and then applying a nonlinear function called an activation function.

Each neuron performs three main steps:

a- Weighted sum of inputs plus bias

For the neuron ***j*** in the hidden layer, the total input signal u_j is calculated by multiplying each input x_i by its associated weight w_{ji}, then summing all these, and finally adding a bias term b_j :

$$u_j = f\left(\sum_{i=1}^{Nin} w_{ji}\, x_i + b_j\right) \quad \text{(I.8)}$$

Where:

x_i *:* are the inputs to the network .

w_{ji} *:* are the weights from input *i* to neuron *j* in the hidden layer.

b_j *:* is the bias of neuron *j*.

N_{in} : is the total number of inputs.

The activation function f(.), is critical, as it introduces the nonlinearity that allows the network to learn complex patterns. Common choices, include the sigmoid function, which squashes values between 0 and 1; the hyperbolic tangent (tanh), which outputs values between -1 and 1; and the computationally efficient ReLU (Rectified Linear Unit).

b- Applying a nonlinear activation function:

To introduce non-linearity, the neuron passes this sum u_j through an activation function ***f***, such as sigmoid, tanh, or ReLU:

$$h_j = f\left(u_j\right) = f\left(\sum_{i=1}^{Nin} w_{ji} x_i + b_j\right) \quad \text{(I.9)}$$

The output h_j : is the activated value from neuron ***j***.

c- Calculating the output layer:

The final output O_k of the network's output neuron ***k*** is a weighted sum of all hidden neuron outputs ***hj***, plus a bias Z_{0k}, passed through another activation function ***g***:

$$O_k = g\left(\sum_{j=1}^{Nh} Z_{kj}.f\left(\sum_{i=1}^{Nin} w_{ji} x_i + b_j\right) + Z_{0k}\right) \quad \text{(I.10)}$$

Where:

Z_{kj} :is the weight from hidden neuron ***j*** to output neuron ***k***,

Z_{0k} :is the bias of output neuron ***k***,

N_h :is the number of neurons in the hidden layer.

f and g : the activation functions for the hidden and output layers, respectively.

O_k: the k-th output of the network.

x_i: the i-th input of the network.

w_{ji}: the connection weight between input i and hidden neuron j.

I.4.3 Architectural Evolution: From Shallow to Deep Networks

We can think of the basic single-hidden-layer MLP as the foundation upon which more complex neural architectures are built. Because of its limited depth, this foundational design is fittingly called a "shallow network". As we will discuss, its strength lies in its ability to approximate static nonlinear functions **[34]**, but its crucial limitation is a static, memoryless nature that makes it ill-suited for modeling systems with temporal dependencies.

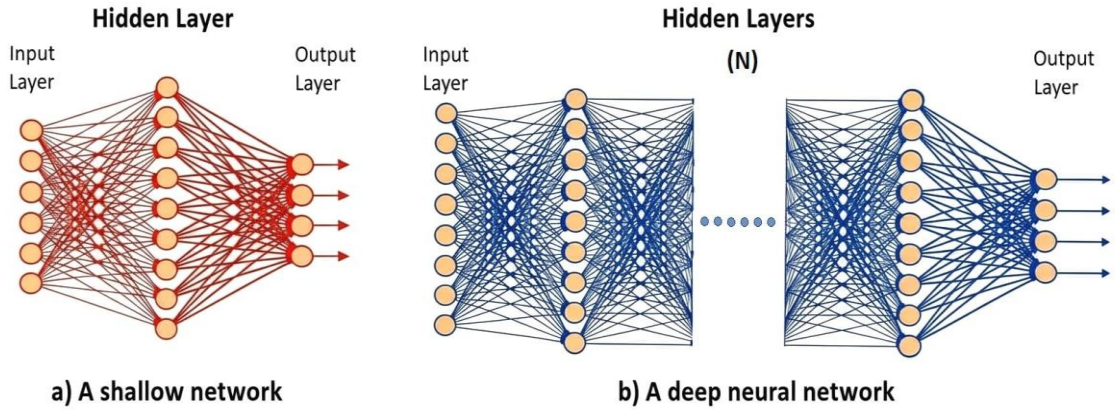

Figure I.3: Architectural Evolution from a Simple MLP to a Deep Neural Network **[1].**

The next logical step in the evolution of these networks was to increase their depth. This led to the development of "deep neural networks" (DNNs) **[35].** As illustrated in Figure 1.3, a deep neural network is conceptually simple: it is an MLP that contains multiple hidden layers stacked one after another.

In contrast to a shallow network, a deep neural network employs its multiple hidden layers to learn a hierarchy of features. This multi-layered structure enables a hierarchical learning process, where the network builds complex representations by composing simpler patterns learned in its earlier layers. This ability to automatically learn a deep hierarchy of features is the key reason why DNNs excel at difficult tasks, such as those found in computer vision **[36]**. By the time we reach the final layers, the network is capable of recognizing highly abstract concepts, a capability that has revolutionized fields like computer vision.

However, it is critical to understand that this evolution in depth does not address the problem of temporal dynamics. Increasing the number of hidden layers makes an MLP a more powerful static function approximator, but it remains fundamentally static and memoryless **[37]**. The challenge of modeling time-based relationships requires a different kind of architectural evolution, which we will address after discussing the MLP's limitations in more detail.

I.4.4 The Learning Process in Neural Networks

The defining characteristic that sets neural networks apart is their ability to "learn" from data. In this context, learning is the process by which the model methodically adjusts its vast number of internal parameters—the synaptic weights and biases—to progressively build a better representation of the relationship between input data and the desired outputs **[38]**. The goal is to reduce the error between the predicted results and the actual values found in the dataset. This learning process is inspired by how the human brain adapts and improves through experience, helping the network get better over time.

In a more formal way, we consider a training set $\{x_k, y_k^p\}_{k=1}^N$

where $\boldsymbol{x_k}$ is the input vector at time step $\boldsymbol{k}$, and $\boldsymbol{y_k^p}$ is the observed output. The learning task is then to find the best parameters $\boldsymbol{\theta}$ of a candidate function $\psi(x,\theta)$ that minimizes a cost function, usually the sum of squared errors:

$$j(\theta) = \sum_{k=1}^{N} \left(y_k^p - \psi(x_k, \theta) \right)^2 \tag{I.11}$$

This cost function tells us how well the model predictions match the real data, and minimizing it is the heart of the learning process.

I.4.4.1 Foundational Concepts: Loss Function and Optimization

At the core of this learning process lies the concept of a loss function (also called an error or cost function). This function serves as a quantitative measure of how "wrong" the model's predictions are. When a prediction results in a high loss, it indicates the model performed poorly, whereas a low loss reflects good performance. The Mean Squared Error (MSE) is a commonly used and reliable loss function for regression problems. It measures the average squared distance between what the model predicts and the actual values for all the data points. More than just a number, this error guides the learning process by showing how the network should adjust its internal settings to fit the data better.

Optimizing this cost function involves using methods called first-order or second-order:

- First-order methods only use the gradient $\nabla J(\theta)$ to update the parameters, following this general rule:

$$\theta^{k+1}=\theta^{k}-\mu_{k}\nabla J\left(\theta^{2}\right) \quad \text{(I.12)}$$

Where $\boldsymbol{\mu_k}$ is the learning rate, this learning rate can be fixed or changed over time to speed up learning and avoid getting stuck near the minimum.

- Second-order methods also make use of the Hessian matrix $H(\theta)=\nabla^2 J(\theta)$ which contains second derivatives and assists in determining more precise directions and step lengths for updating the parameters.

$$\theta^{k+1}=\theta^{k}-H^{-1}\left(\theta^{k}\right)\nabla J\left(\theta\right) \quad \text{(I.13)}$$

However, calculating and inverting this matrix can be expensive, so quasi-Newton methods like BFGS or Levenberg-Marquardt approximate it to save time and still get good results.

I.4.4.2 Paradigms of Learning

The strategy used to guide this minimization process depends on the nature of the data and the problem at hand. In machine learning, there are several distinct learning paradigms **[39]**:

- **Supervised Learning:** This is the most prevalent paradigm for the modeling and classification tasks. The network is trained on a "supervised" dataset, where every input sample X is paired with a known, correct output or label Y. The network's task is to learn the underlying mapping function f: X→Y.

- **Unsupervised Learning:** In this case, the training data is unlabeled. The algorithm must explore the data on its own to discover hidden patterns, structures, or clusters. This is useful for tasks like customer segmentation or anomaly detection.

- **Semi-supervised Learning:** Acting as a bridge between the two, this approach leverages a small amount of labeled data alongside a large amount of unlabeled data. It has become increasingly important in domains where data is plentiful, but labeling it is a costly and time-consuming bottleneck **[40]**.

- **Reinforcement Learning:** Here, a model (or "agent") learns through trial and error by interacting with a dynamic environment. Instead of explicit labels, it receives feedback in the form of rewards or penalties. The agent's goal is to learn a sequence of actions—a policy—that maximizes its cumulative reward over time, a framework particularly suited for robotics and game playing **[41]**.

I.4.4.3 The Backpropagation Algorithm: The Engine of Supervised Learning

For feedforward networks like the MLP, the workhorse algorithm for supervised learning is backpropagation, famously popularized in a seminal paper by Rumelhart, Hinton, and Williams **[42]**. This algorithm provides a computationally efficient method to calculate the gradient of the loss function concerning every single parameter in the network, making learning in even very deep networks feasible. The process can be broken down into a repeatable cycle:

***a-Forward Pass*:** The network receives an input (or a small batch of inputs) that moves forward layer by layer. Each neuron calculates a weighted sum and applies its activation function until the final output $\tilde{\mathbf{Y}}$ is generated.

***b-Loss Computation*:** The error is calculated using the loss function. For regression, the MSE is:

$$E=\frac{1}{N}\sum_{i=1}^{N}\left(Y_i-Y_i\right)^2 \tag{I.14}$$

***c-Backward Pass (Backpropagation)*:** This is the core of the algorithm. The calculated error is propagated backward through the network, from the output layer to the input layer. Using the chain rule of calculus, the algorithm efficiently computes the partial derivative of the error concerning each weight and bias ($\frac{\partial E}{\partial w}$) . This gradient tells us how a slight change in a specific weight would affect the total error.

***d-Weight Update*:** The weights are then adjusted in the direction that minimizes the error, specifically, in the opposite direction of their calculated gradient. The simplest update rule is:

$$w^{k+1}=W^{k}-\eta \nabla E \tag{I.15}$$

Where η is the learning rate.

These steps correspond to the iterative algorithm where:

- At every iteration ***k***, the weights and biases are updated by applying the gradient derived from the current input, demonstrating how the learning progresses gradually.
- The sigmoid function is commonly chosen as an activation function because its derivative is simple to evaluate, which makes computing gradients more efficient :

$$f(x)=\frac{1}{1+e^{x}};\quad f'(x)=f(x)\left(1-f(x)\right) \tag{I.16}$$

Backpropagation uses the chain rule to compute all the necessary derivatives for updating weights, ensuring efficient decrease of the cost function.

This process repeats until the average error over all data is low enough, or the gradients get close to zero, meaning the algorithm has found or is near a minimum.

This cycle of forward pass, loss calculation, backward pass, and weight update is repeated over many thousands or millions of examples for many iterations, called epochs. The choice of the learning rate,

It is critical: if it's too high, the learning process can become unstable and overshoot the optimal solution; if it's too low, the training can be extremely slow or get stuck in a suboptimal solution.

e-Regularization and Optimization

This entire iterative process is managed by an optimization algorithm, such as Stochastic Gradient Descent (SGD) or more advanced adaptive methods like Adam, until the model's performance on a separate validation dataset stops improving. Overfitting occurs when a model simply remembers its training examples rather than generalizing patterns. To prevent this, regularization is crucial. Dropout is a popular technique that helps by randomly dropping neurons during training, which forces the network to develop diverse and stronger features **[43-44].**

Using adaptive optimization algorithms like Adam lets us automatically adjust the learning rate η, based on how the error changes locally, making learning more stable and faster. Dropout regularization works by randomly disabling some neurons during training, which stops them from becoming too dependent on one another and helps the model generalize better to unseen data

I.4.5 Challenges and limitations inherent in using MLPs in a sequential approach

An MLP is comparable to a high-powered camera that captures detailed, static snapshots of data. However, because of its feedforward structure, it has no built-in memory of past inputs. In other words, it cannot naturally perceive the entire “movie” — the history and temporal dynamics of a system — without an external workaround. This fundamental limitation poses significant challenges when applying MLPs to sequential or time-dependent data.

This section highlights the main obstacles encountered when using fixed external memory architectures, such as Multilayer Perceptrons (MLPs) with sliding windows, in modeling complex dynamic systems. Before introducing architectures with integrated memory, it is crucial to fully understand the fundamental limitations of so-called “memoryless” models and the trade-offs imposed by their workaround solutions. Here, we present the critical issues that hinder flexibility, robustness, and learning capacity over long temporal dependencies, especially in the context of static models applied to sequential data. These challenges emphasize how hard it is to find the right balance between memory capacity, computational cost, and learning performance in traditional models, which is why better-suited dynamic architectures have been developed.

I.4.5.1 The "Sliding Window" Technique

The easiest way to help an MLP work with sequential data is to use the 'sliding window' method. This means adding a fixed number of previous inputs and/or outputs to the network's input at each step. By doing this, a problem that changes over time becomes a regular, static regression problem.

This technique is the basis for several classic architectures, including Time-Delay Neural Networks (TDNN) **[45]** and MLP-based NARX models **[46]**. The principle of the NARX (Nonlinear Autoregressive with eXogenous Inputs) model is illustrated in Figure I.4.

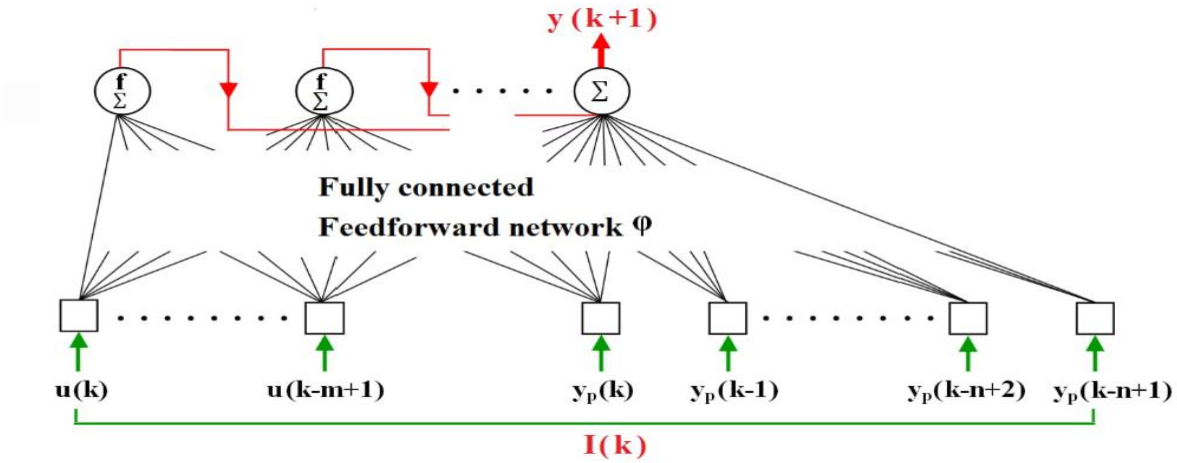

Figure I.4 : Principle of the NARX Architecture for Dynamic Modeling **[2]**

The diagram shows how a window of past input values (u) and past output values (y) is used as a single input vector to predict the future output. Using past outputs as inputs acts like an external memory for the model.

What really makes the NARX model special is its feedback loop. As the diagram shows, it doesn't just consider past inputs (u), but also its previous predictions (y). By feeding its earlier outputs back into its input, the model creates an artificial form of memory outside its core structure.. This is the distinctive trick that allows a static MLP to model dynamic behavior.

To be more specific, the input vector X(k) for this setup is built like this **[2]:**

$$x(k)=\left[u(k),u(k\text{-}1),\ldots,u(k\text{-}n_u+1),y(k\text{-}1),\ldots,y(k\text{-}n_y+1)\right]^T \quad \text{(I.17)}$$

The sliding window approach fundamentally changes the way sequential data is handled by framing the time-dependent problem into a supervised static learning problem, where each input contains a fixed history length of observations. This makes the approach versatile and compatible with various classical machine learning algorithms, including MLPs, but also simpler models like linear regression or decision trees when appropriate.

Because the window 'slides' forward step-by-step over the data sequence, it provides overlapping segments of past states and outputs to the model, enabling it to consider temporal context up to the window size implicitly.

This method is effective and straightforward, with built-in tools available in most programming languages to create sliding windows from time series efficiently.

I.4.5.2 Critical Drawbacks of the Workaround

However, this workaround introduces critical drawbacks:

- **The Curse of Dimensionality:** When the memory window gets longer, the input size expands a lot. This causes a big jump in the number of parameters the model has to learn, needing more data and increasing the chance of overfitting, as well as the computing time **[47].**
- **A Fixed and Arbitrary Memory:** The model's "memory" is rigidly constrained by the window size, which must be defined a priori. This makes it challenging to capture dependencies over long time scales without using a prohibitively large and inefficient window.

- **High Computational Cost:** For long sequences, training MLPs with many delayed inputs becomes computationally intensive. Several studies indicate that the method can quickly become computationally heavy when handling long-duration memory **[15].**

Furthermore, the sliding window method fails to naturally capture long-range temporal dependencies that may exceed the predefined window size, which is a critical limitation in many real-world dynamic systems exhibiting slow or complex temporal dynamics.

Additionally, selecting the window length requires balance, too short, and it may fail to capture key historical details; too long, and it can raise computational complexity and bring in unnecessary or noisy data that complicate training. With memory external and predefined, this architecture struggles to handle sequences with varying time dependencies

or new input patterns dynamically. As a result, it is less flexible and less efficient than models that include memory within their structure **[48]**.

In general, while the MLP solves the nonlinearity problem, it fails to address the challenge of dynamics elegantly. This fundamental gap—the need for a model with intrinsic memory—paved the way for the development of architectures explicitly designed for sequential data: Recurrent Neural Networks (RNNs), which will be the focus of the next section.

I.5 Focus on the case study: the photovoltaic system as a complex nonlinear system

I.5.1 A Brief History and the Growing Importance of PV Systems

The story of the photovoltaic (PV) effect began in 1839 with the French physicist Edmond Becquerel, who first observed that certain materials could produce an electric current when exposed to light. However, it remained a scientific curiosity for over a century until the development of the first practical silicon solar cell at Bell Labs in 1954. This breakthrough marked the true birth of modern solar power technology.

For decades, the high cost of PV cells confined their application mainly to specialized areas like providing power to satellites in space or remote off-grid installations. But in recent years, driven by a combination of technological advances, economies of scale, and a growing global urgency to address climate change, solar energy has undergone an explosive transformation **[1]**.

Today, photovoltaic systems are no longer a niche technology; they are a central pillar of the global energy transition **[47]**. The rapid growth of solar power capacity around the world shows how important it is for reducing carbon emissions. But this growth also brings a new challenge: making sure these large solar farms run as efficiently, reliably, and safely as possible.

As solar power is becoming a bigger part of the electrical grid, any sudden drop in energy from a big solar farm due to a hidden problem can affect the whole grid's stability. Therefore, the ability to accurately model PV systems and to diagnose their faults in real-time is not just an engineering problem—it is a fundamental requirement for the success of

our future energy infrastructure **[49]**. This is the practical context that motivates the research presented in this thesis.

I.5.2 The Photovoltaic Cell: From Physics to the I-V Characteristic

I.5.2.1 The Photovoltaic Effect

At its most basic level, a photovoltaic cell works by harnessing the photovoltaic effect, a physical and chemical phenomenon that occurs in semiconductor materials. A solar cell is essentially a large P-N junction diode. When photons of light with sufficient energy strike the semiconductor material, they can knock electrons loose, creating "electron-hole" pairs.

Due to the internal electric field present at the P-N junction, these freed electrons are swept to the N-side and the holes are swept to the P-side. This separation of charge creates a voltage difference across the cell. If an external circuit is connected, this voltage will drive a current, thus generating electrical power.

I.5.2.2 The I-V Characteristic: The Cell's "Fingerprint"

The fundamental electrical behavior of a PV cell is described by its current-voltage (I-V) characteristic curve**.** This curve, as shown in Figure I.5, is the "fingerprint" of the cell's performance under specific conditions of irradiance and temperature. It shows the amount of current (I) the cell will produce for a given voltage (V) across it.

There are several key points of interest on this curve that define the cell's performance:

- **Short-Circuit Current (I_{sc}):** This is the maximum current the cell can produce, found at the point where the voltage is zero**.** I_{sc} is almost directly proportional to the solar irradiance.

- **Open-Circuit Voltage (V_{oc}):** This is the maximum voltage the cell can produce, found where the current drops to zero. V_oc is primarily influenced by the cell temperature.

- **Maximum Power Point (MPP):** The power produced by the cell at any point on the curve is simply P = V * I. There is a unique point on the "knee" of the curve where this power is maximized.This point is called the Maximum Power Point (MPP), and the corresponding voltage and current are denoted V_mpp and I_mpp. Operating the cell at this

point is the goal of all MPPT controllers. This I-V curve is highly nonlinear and changes its shape dramatically with fluctuations in irradiance and temperature, which is the root of the modeling challenge.

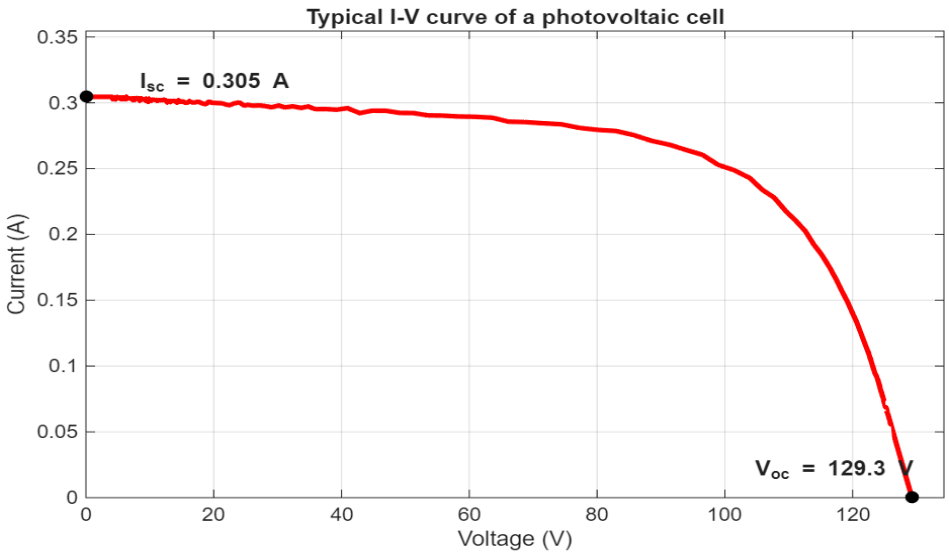

Figure I.5 : A typical I-V characteristic curve of a photovoltaic cell.

I.5.3 Modeling the PV Generator: The Limits of "White-Box" Models

The primary approach to modeling a PV cell based on its underlying physics involves using "white-box" equivalent circuits. The single-diode model (SDM), illustrated in Figure I.6(a), is the most widely used starting point **[98,100]**.

In this representation, the PV cell is modeled as a current source (Iph) that generates power from incident light, combined with a diode (D) and two parasitic resistances representing real-world losses: the series resistance (R_s) and the shunt resistance (R_{sh}). Its behavior is characterised by a five-parameter vector θ:

$$\theta = \left\{ I_{ph}, I_0, V_t, R_s, R_s \right\} \tag{I.18}$$

These parameters are used to describe the cell's behavior in the following equation:

$$I = I_{ph} - I_0 * \left[\exp\left(\left(V + I*R_s \right) / V_t \right) - 1 \right] - \left(V + I*R_s \right) / R_{sh} \tag{I.19}$$

However, this conventional model has a significant limitation. While it performs accurately under normal operating conditions, it fails to account for behavior in reverse bias—a frequent occurrence during partial shading **[50,51]**. To address this, the Bishop model was developed **[100].**

As shown in Figure I.6(b), the Bishop model introduces a non-linear term to represent the avalanche breakdown effect **[99, 100]**. This adds three new parameters: the breakdown voltage (V_{br}) and two adjustment coefficients (m and k). The updated parameter vector Φ becomes:

$$\Phi = \left\{ I_{ph}, I_0, V_t, R_s, R_{sh}, V_{br}, a, m \right\} \quad \text{(I.20)}$$

By adding the breakdown term to the single-diode equation, we get the new, more comprehensive Bishop model equation:

$$I = I_{ph} - I_0 * \left[\exp\left(\frac{V + I*R_s}{V_t} \right) - 1 \right] - \frac{(V + I*R_s)}{R_{sh}} * \left[1 + k * \left(1 - \frac{(V + I*R_s)}{V_{br}} \right)^{-m} \right] \quad \text{(I.21)}$$

Where **[100]**

- **I_{ph} (Photocurrent):** The equivalent current produced by the cell, which is directly proportional to the incident solar irradiance.
- **I_0:** The reverse saturation current of the diode.
- **$Vt=(a.k_b.Tc/q)$ (Thermal voltage):** This voltage depends on the cell temperature (Tc), where:
  - ***a*** is the diode ideality factor (ranging from 1 to 2);
  - ***k_b*** is the Boltzmann constant (1.38×10^{-23} J/K);
  - ***q*** is the electron charge (1.602×10^{-19} C).
- ***Rs*:** The series resistance of the cell.
- ***Rsh*:** The shunt (parallel) resistance of the cell.
- ***k*:** Bishop fraction coefficient (typically around 0.1). It represents the fraction of the shunt current involved in the breakdown.
- ***m*:** Bishop avalanche exponent (typically between 3.4 and 4). it governs the non-linearity of the curve in reverse bias.
- **Vbr**: The breakdown voltage of the cell (typically ranging from -10 V to -30 V).

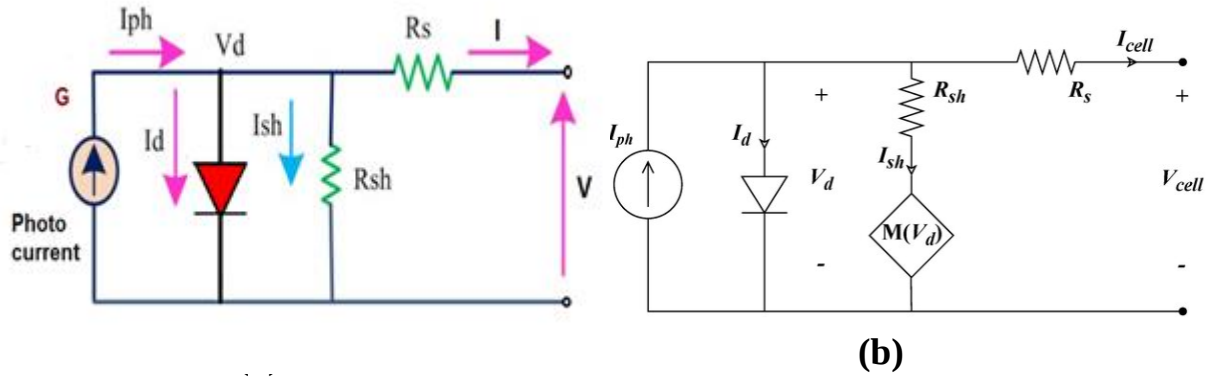

Figure I.6: Equivalent circuit models for a PV cell. **(a)** The standard single-diode model. **(b)** The Bishop model, which adds a term for avalanche breakdown **[100]**.

The Challenge of Parameter Estimation : Herein lies the primary challenge of the "white-box" approach. These five parameters are not constants. They depend heavily on the environmental conditions (irradiance and temperature) and the physical state of the cell (e.g., degradation over time).

Accurately estimating these parameters from datasheets or experimental data is a notoriously difficult and often unreliable optimization problem. Many different methods have been proposed, but finding a set of parameters that works well across all operating conditions is a significant hurdle.

The Limits in Real-World Conditions : Even more importantly, while the single-diode model works reasonably well for a single, healthy cell under uniform conditions, it struggles to represent the complex behaviors of a real-world PV module or array, especially when faults are present.

For example, modeling the effect of partial shading requires complex modifications, often involving multiple interconnected diode models, which makes the parameter estimation problem even harder. Similarly, modeling the effects of degradation or other physical faults is not straightforward.

Our motivation comes from the limits of physically detailed models. They give good insight but are often too heavy to work with in real-time problems. Instead, black-box methods, which figure out how things work by looking at data rather than equations, have become a smart way forward.

I.5.4 The Challenge of Fault Diagnosis in PV Systems

While a perfect model can describe a healthy PV system, the real-world challenge lies in understanding and identifying its behavior when things go wrong. As detailed in comprehensive reviews on the topic **[1,50],** fault diagnosis is a critical task for several key reasons.

I.5.4.1 Why Fault Diagnosis is Crucial

Undetected faults in a PV system are a serious concern because they can lead to significant consequences:

- **Production Losses:** Even minor faults can lead to a substantial loss of energy production over time, directly impacting the economic viability of the plant.
- **Increased Maintenance Costs:** Faults that are not identified and localized quickly can lead to more severe damage, requiring more expensive and complex repairs.
- **Safety Risks:** Certain faults, like poor isolation or short-circuits, can pose serious safety risks, including the danger of fire.

The development of automated and reliable diagnostic methods is therefore essential for the operation of modern PV systems.

I.5.4.2 A Catalogue of Common PV Faults

Photovoltaic systems can experience many different types of faults, which can be broadly categorized as intrinsic (related to the hardware itself) or extrinsic (caused by external factors). As catalogued in numerous studies **[51]**, some of the most frequent and impactful faults include:

- **Partial Shading:** This is one of the most common and damaging conditions. When even a small part of an array is shaded (by a tree, a building, or even just dirt), the affected cells can stop producing power and start acting like resistors. This not only causes a dramatic drop in power but also creates "hot spots" that can permanently damage the cells. This fault creates a highly distorted I-V curve with multiple power peaks, making both MPPT and diagnosis very challenging.

- **Increased Series Resistance (ISR):** This fault is often a sign of degradation, caused by issues like corrosion at the interconnections between cells. It reduces the overall efficiency of the module by dissipating more power as heat.

- **Bypass Diode Faults:** Bypass diodes are safety devices designed to protect the system during partial shading. However, they can fail, either by short-circuiting or by developing a faulty impedance. A failed diode can render a whole section of a module useless or even dangerous.

- **Module-Level Faults:** These include more severe issues like the short-circuiting of an entire module or physical damage like cracks, which can be caused by environmental stress.

I.5.4.3 Traditional Diagnostic Approaches and the Power of I-V Curve Analysis

Traditionally, fault diagnosis in photovoltaic systems has relied on a combination of approaches such as manual inspections and the evaluation of the electrical output signals from the solar array. Although these methods can detect many common problems, they frequently miss subtle or intermittent faults that do not always present clearly. This issue underscores the necessity for more advanced diagnostic techniques.

In this context, leveraging the complete current-voltage (I-V) characteristic curve offers a more robust approach to fault diagnosis. Since each fault type imposes a unique and identifiable 'fingerprint' on the curve’s trajectory, this method enables the detection and differentiation of defects that might otherwise remain latent.

However, the main challenge is to accurately extract and interpret the fault patterns hidden in the I-V data. This task is intricate and requires advanced, data-driven analytical approaches. Chapter IV of this thesis is dedicated to developing and implementing such cutting-edge methods to fully harness the diagnostic potential of I-V curve analysis.

I.6 Conclusion

In this first chapter, we explained the main ideas behind using neural networks to model dynamic systems. We looked at why traditional physics-based models often struggle

to represent complex and changing behaviors accurately. Because of this, deep learning methods—which build models directly from data—have become more popular.

We also looked at how neural networks have changed, moving from basic models like the Multilayer Perceptron (MLP) to smarter ones that remember past information, like Recurrent Neural Networks (RNNs). This shows that simple models struggle to understand data that changes over time. Finally, we discussed the current challenges with these advanced architectures, focusing on their "black box" nature and the need to evolve toward a more interpretable state-space formulation. These limits strongly justify the ongoing research efforts—including our own—to improve the transparency and performance of these networks.

To ground this discussion in a practical context, we then conducted a detailed review of our chosen case study: the photovoltaic (PV) system. We have shown that the PV system is a perfect example of a complex, nonlinear system whose behavior is difficult to model with traditional "white-box" approaches. Our analysis of its operation, its equivalent circuit models, and its common fault modes has highlighted the specific and challenging problems that any advanced modeling and diagnostic technique must overcome.

This structured and critical analysis of both the solution space (dynamic neural networks) and the problem domain (PV systems) frames the challenges we will explore further in this study. With this foundational understanding now in place, the next chapter will take a closer look at the different types of dynamic neural networks, from the traditional RNNs to the advanced gated models that play a key role in our work.

This structured and critical analysis of both the solution space (dynamic neural networks) and the problem domain (PV systems) frames the challenges we will explore further in this study. With this foundational understanding now in place, the next chapter will take a closer look at the different types of dynamic neural networks, from the traditional RNNs to the advanced gated models that play a key role in our work.

CHAPTER II

Dynamic Neural Network Architectures and Their Evolution

II.1 Introduction

In the last chapter, we hit a wall with traditional modeling. We saw that even though static networks like the MLP are great with nonlinear data, they have no memory of the past. This is a real problem for any system that changes over time. So, this chapter picks up right where we left off. We're going to dive into the specialized architectures that were built to solve this exact problem: the family of Dynamic Neural Networks (DNNs).

Our exploration will follow a logical progression, tracing the evolution of these models from simple concepts to the state-of-the-art tools that form the technological core of this thesis. To begin, we will establish the two foundational approaches for representing time in a neural network: the external (spatial) representation and the internal (dynamic) representation.

This conceptual basis will allow us to survey the prominent families of dynamic architectures. First, we will start with Global Input-Output Models, focusing on the NARX architecture, which cleverly uses an external memory mechanism to handle time. Next, we will examine Neural State-Space Models (NSSMs), an approach that introduces a more structured and explicit internal state, borrowing concepts from classical control theory. This emphasis on state representation is central to our hypothesis. This path will then naturally lead us to the more flexible and unified Recurrent Neural Networks (RNNs), where we will cover the classic Elman and Jordan networks.

Finally, by analyzing the limitations of these early models—specifically the fundamental challenge of learning long-term dependencies—we will justify the need for the more sophisticated gated architectures. This final part of our review will provide a detailed look at the modern workhorses of sequence modeling: the Long Short-Term Memory (LSTM), the Gated Recurrent Unit (GRU), and, most importantly for our work, the Bidirectional LSTM (BiLSTM).

Ultimately, this chapter will offer a complete toolkit of dynamic architectures and their learning principles, giving readers the knowledge they need to understand the modeling and classification contributions presented in the following chapters.

II.2 Representing Time in Nonlinear System Modeling

Effective modeling of nonlinear dynamic systems requires incorporating time to reflect their evolving behavior and dependencies **[52]**. Two conceptually distinct approaches to handling time in neural network modeling have been extensively documented in the literature, and were explicitly detailed in the work of M.Zemouzi in **[15]**. These approaches are distinguished as the spatial (external) representation and the internal (dynamic) representation of time.

II.2.1 Spatial Representation (External Memory Mechanism)

In this approach, time is treated as an external parameter to the neural network. Temporal memory is established by constructing a sliding window over the input data. Specifically, as shown in Figure II.1 (left), the network receives a concatenated vector comprising data at the current time step and a series of past time steps (e.g. at times $t, t-\tau_1, \ldots, t-\tau_N$). This extended vector is then processed by a static network simultaneously **[53]**.

This methodology, also referred to as the spatial representation of time by Elman (1990), enables the use of standard static feedforward network architectures. These networks have no built-in memory. They rely entirely on pre-processing to handle time dependencies. Although easy to use, this method does not provide actual dynamic memory and thus cannot grasp complex patterns outside the fixed input window.

II.2.2 Internal Representation (Intrinsic Temporal Mechanism)

By contrast, the internal representation approach presumes that the neural network inherently embodies the capacity to process temporal sequences through an internal memory mechanism. This is termed the dynamic representation by Elman or the internal representation by Chappelier.

In this framework, as illustrated in Figure II.1(right), temporal memory is captured naturally via recurrence in the network's connections without requiring explicit input expansions **[54]**. The network maintains an internal "state" or "memory" that evolves and updates dynamically as the sequence progresses, enabling persistence of information over arbitrary lengths of time.

Such networks can model long-term temporal dependencies, a critical feature for accurately representing the complex and nonlinear dynamics observed in real-world systems **[55]**. This intrinsic memory is what distinguishes dynamic networks, such as recurrent neural networks (RNNs) and their advanced variants, from static feedforward architectures, as highlighted in comprehensive surveys of the field **[56]**.

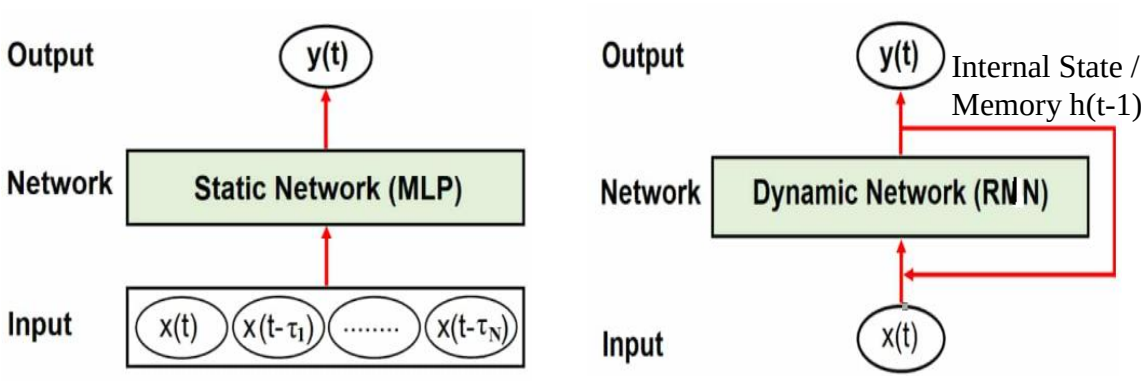

Figure II.1: A Schematic Comparison of Temporal Representation Approaches in Neural Networks **[54].**

-The spatial approach (left) treats time externally by feeding a fixed window of past data to a static network.

-The internal approach (right) handles time intrinsically, using recurrent connections to maintain a dynamic internal state (h(t-1)) that serves as memory.

II.3 Architectures of Dynamic Neural Networks

II.3.1 Introduction to Dynamic Neural Network Architectures

After introducing the two main ways to represent time, the external spatial approach and the internal dynamic approach, we now examine the specific architectures designed for modeling dynamic systems. These architectures provide the means to capture the evolving behavior and long-range dependencies typical of nonlinear systems, a topic that has been widely studied in the scientific literature.

It is important to note that a modeling approach can sometimes borrow a mechanism from another category to achieve its goals. This is particularly true for the first family we

will study, the Global Input-Output Models, which use a spatial mechanism to model a recursive dynamic behavior.

With this in mind, three prominent families of dynamic neural networks stand out, each offering a distinct strategy for modeling system dynamics **[52, 56]:**

Global Input-Output Models, which directly model the relationship between a history of past inputs and the current output, without formalizing a recurrent hidden state.

Neural State-Space Models, where the state vector **x(k)** is explicitly defined and updated through a structured dynamic function, often combining linear and nonlinear components.

Recurrent Neural Networks (RNNs) which incorporate recursive feedback loops directly into their neuron-level architecture, create a powerful and evolving internal memory.

In the following sections, we will elaborate on each of these architectures, discussing their operating principles, advantages, and limitations. We will then conduct a deep dive into the family of Recurrent Neural Networks, as they constitute the most widely studied and highest-performing group for processing complex temporal data and are central to the contributions of this thesis **[55, 57].**

II.3.2 The Input-Output Paradigm: NARX Models

II.3.2.1 The Direct Input-Output Philosophy

Global input-output models represent a foundational and historically significant approach to "black-box" modeling of nonlinear dynamic systems. As highlighted in the pioneering work of Narendra and Parthasarathy **[14],** instead of assuming the existence of an abstract internal state, this paradigm focuses on learning the direct functional relationship between a system's past inputs, its past outputs, and its current output. The most iconic model in this family is the NARX (Nonlinear AutoRegressive with eXogenous inputs) network.

II.3.2.2 Mathematical Formulation The core assumption of the NARX model is that the system's output y(t) at time t can be predicted by a nonlinear function f that takes the d_y past outputs and d_u past inputs as its arguments. The general equation is as follows **[58]**:

$$y(t) = f\left(y(t-1),\ldots,y(t-d_y),u(t-d),\ldots,u(t-d-d_u)\right)+\in(t) \quad \text{(II.1)}$$

Where:

- y(t) :is the system's output at time t.
- u(t): is the exogenous (external) input at time t.
- d_y and d_u : are the memory orders, which set how far back the model looks at past outputs and inputs.
- d :is the system's delay, representing the time lag between an input action and its first effect on the output.
- f(·) :is an unknown nonlinear function that the neural network learns to approximate.
- ε(t) :represents an error or noise term, typically assumed to be white noise.

When we use a Multilayer Perceptron (MLP) to approximate the function f(·), the resulting structure is often called a NARX recurrent neural network **[58]**, because of the feedback loop that feeds the output back to the input.

II.3.2.3 Distinction from TDNN (Time Delay Neural Network)

At this point, it's helpful to distinguish the NARX model from another key architecture in this family: the Time Delay Neural Network (TDNN), which was famously introduced by Waibel et al. **[33]** for speech recognition.

A TDNN also uses a sliding window, but it looks only at the past inputs:

$$y(t) = f\left(u(t),u(t-1),\ldots,u(t-n_u)\right) \quad \text{(II.2)}$$

The critical difference is the absence of the autoregressive terms (y(t-i)). This makes the TDNN a non-recursive model. A great way to think about it is as the neural network equivalent of a nonlinear FIR (Finite Impulse Response) filter. It is inherently stable and excellent for recognizing patterns in an input signal.

The NARX model, in sharp contrast, is defined by feeding its own past outputs back into its input. This feedback loop makes it a recursive system—the nonlinear equivalent of an IIR (Infinite Impulse Response) filter. This recursion gives it a much richer dynamic

memory, making it ideal not just for recognizing patterns, but for truly capturing the internal dynamics of a system. This is precisely why the NARX model is a cornerstone of system identification, and it also explains why its stability must be carefully checked, unlike the TDNN.

II.3.2.4 Architecture and Memory Mechanism

Conceptually, the NARX model cleverly transforms a dynamic modeling problem into a static regression task. It achieves this using an external memory mechanism based on a sliding window (also known as a tapped delay line). At each time step ***t***, a special input vector, called the regression vector ϕ(t)**,** is created by gathering past values of the inputs and outputs:

$$\varphi(t) = \left[y(t-1), \ldots, y(t-d_y), u(t-d), \ldots, u(t-d-d_u) \right]^T \quad \text{(II.3)}$$

This high-dimensional vector φ(t) is then fed into a static neural network, typically a Multilayer Perceptron (MLP), which is tasked with learning the function f. Crucially, the network itself has no internal memory; all the temporal information is contained in the regression vector prepared for it. The general architecture is shown in Figure II.2.

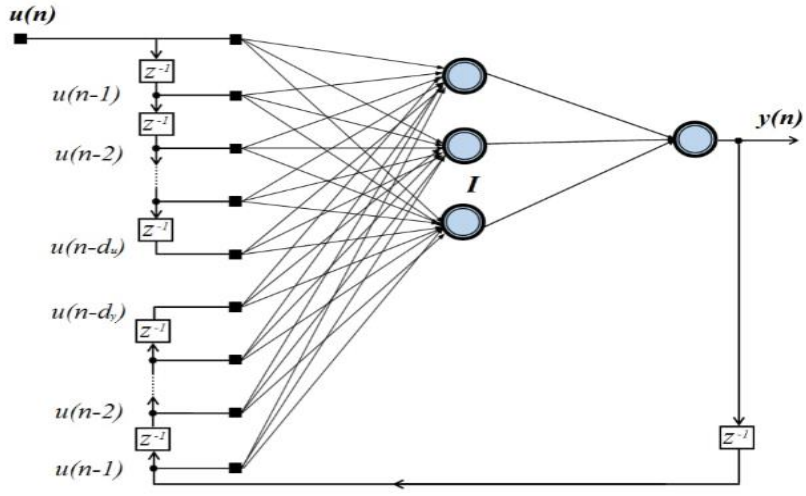

Figure II.2: The architecture of a NARX neural network **[58]**.

Tapped delay lines (the z^{-1} blocks) build the regression vector from past inputs u(n-i) and past outputs y(n-i). A static feedforward network (MLP) then processes this vector to produce the current output y(n).

II.3.2.5 Operating Modes

A crucial distinction must be made between two operating modes for a NARX network. The main difference depends on where the past output values y(t−i) fed back into the model come from. These modes are the Series-Parallel (SP) mode, used for training, and the Parallel (P) mode, used for prediction and simulation. The conceptual difference is illustrated in Figure II.3.

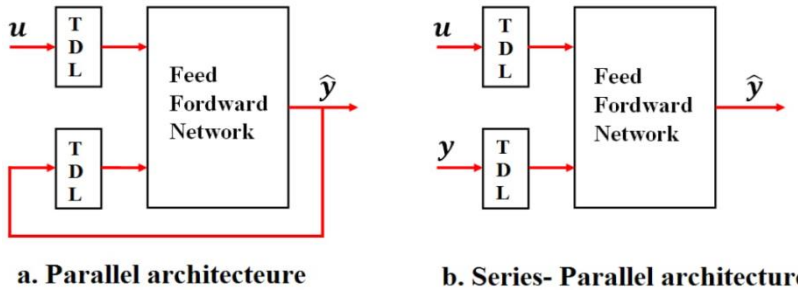

Figure II.3 The NARX architecture with tapped delay lines: (a) parallel architecture, and (b) series-parallel architecture **[59]**

This figure illustrates the two modes, which are distinguished by the signal fed into the output's Tapped Delay Line (TDL). In (a) Parallel Mode, the model's own prediction, ŷ(t), is fed back for simulation and multi-step prediction. In (b) Series-Parallel Mode, the true system output, y(t), is used for training the network. The TDL (Tapped Delay Line) block is the component that implements the "external memory" or "sliding window" mechanism of the NARX model**.[59].**

a- Series-Parallel (SP) Mode

As shown in Figure II.3(b), this mode is used exclusively for training the network. Here, the regression vector is built using the actual, measured output values from the real system. The prediction, which we will denote as $\hat{\mathbf{y}}_{\mathbf{SP}}\mathbf{(t)}$ (where **SP** stands for Series-Parallel), is given by:

$$\hat{y}_{sp}(t) = (f\left(\underbrace{y(t-1),\dots,y(t-d_y)}_{actual\ values},u(t-d),\dots,u(t-d-d_u)\right) \quad \text{(II.4)}$$

The significant advantage of this mode, also known as "teacher forcing," is that the network behaves like a simple feedforward structure at each time step. This makes the

training process stable and allows for the straightforward use of the standard backpropagation algorithm.

b- Parallel (P) Mode

As depicted in figure II.3(a), This mode reflects the real-world use of the model for simulation or multi-step-ahead prediction after it has been trained. Instead of using real data, the model's own predicted outputs are fed back to predict future values. The prediction, now denoted as ŷP(t) (where P stands for Parallel), is calculated recursively:

$$\hat{y}_{\mathrm{p}}(\mathrm{t}) = (\mathrm{f}\left(\underbrace{\hat{y}_p(t-1), \ldots, \hat{y}_p(t-d_y)}_{predicted\ values}, u(t-d), \ldots, u(t-d-d_u) \right) \quad \text{(II.5)}$$

As emphasized in **[58,59]**, the feedback loop operates fully in this mode, rendering the NARX model a genuinely dynamic and recursive system. However, this also introduces the risk of errors building up over time, known as drift, highlighting the critical need for careful stability assessment.

II.3.2.6 Learning Algorithm

Training a NARX network means adjusting its weights and biases so that the difference between what the network predicts $\hat{y}(t)$ and what actually $y(t)$ happens is as slight as possible.

a- Objective Function

The most common objective function is the Mean Squared Error (MSE), computed over a training dataset of N samples. The predictions used for this calculation are generated in the Series-Parallel mode to ensure a stable learning process:

$$J(\theta) = \frac{1}{N}\sum_{t=1}^{N}\left(y(t) - \hat{y}_{SP}(t)\right)^2 \quad \text{(II.6)}$$

b- Gradient Computation: The Backpropagation Algorithm

To minimize this cost function, we need to know how to adjust each parameter in the network. The standard method for efficiently computing the gradient of the cost function

concerning all parameters, $\nabla J(\theta)$, is the Backpropagation algorithm. Because the NARX network in its training (Series-Parallel) mode is a standard feedforward structure, this algorithm can be applied directly **[58].**

For a network with a single hidden layer, the gradient computation for a single training sample ($\varphi(t)$, $y(t)$) can be broken down as follows:

Gradient for the Output Layer:

The process starts by computing the error signal $\delta y(t)$ for the output layer, which measures the sensitivity of the error with respect to the layer's pre-activation $ay(t)$:

$$\delta_y(t) = \frac{\partial E_D}{\partial a_y(t)} = -\left(y(t) - \hat{y}(t)\right).\sigma_y\left(a'_y(t)\right) \tag{II.7}$$

This error signal is then used to find the gradients for the output weights (W_y) and biases (b_y):

$$\frac{\partial E_D}{\partial W_y} = \delta_y(t).h(t)^T \tag{II.8}$$

$$\frac{\partial E_D}{\partial b_y} = \delta_y(t) \tag{II.9}$$

Gradient for the Hidden Layer:

Next, the error is propagated backward to the hidden layer to compute its error signal ***δh(t):***

$$\delta_h(t) = \left(W_y^T \delta_y * \sigma'_h(t)\right) \tag{II.10}$$

This, in turn, allows us to calculate the gradients for the hidden layer's parameters ($\boldsymbol{W_h}$, $\boldsymbol{b_h}$):

$$\frac{\partial E_D}{\partial W_h} = \delta_h(t).\varphi(t)^T \tag{II.11}$$

$$\frac{\partial E_D}{\partial b_h} = \delta_h(t) \tag{II.12}$$

These gradients are typically accumulated over all training samples (in batch training) before a parameter update is performed.

c- Parameter Optimization Algorithms

Once the gradients are computed, an optimization algorithm uses them to update the parameters iteratively θ. While simple Gradient Descent is the foundational method, several more advanced algorithms are commonly used in practice.

- ***First-Order Methods:***

In modern deep learning, adaptive optimizers like Adam or RMSprop are widely used. They adjust the learning rate for each parameter individually, often leading to faster convergence.

- ***Second-Order Methods (Levenberg-Marquardt):***

For networks of moderate size, as is often the case in system identification, more powerful second-order methods can be highly effective. For instance, Hidayat et al. **[59]** employ the Levenberg-Marquardt (LM) algorithm **[59]**. The LM algorithm speeds up training compared to first-order methods by estimating the Hessian matrix. However, it requires more computing power and is usually better suited for problems of a smaller size.

d- Regularization for Improved Generalization

One frequent issue when training neural networks, particularly with small or noisy datasets, is overfitting. This occurs when the model fits the training data too closely and struggles to handle new, unseen examples. To address this problem, regularization methods are commonly applied.

As demonstrated in **[59]**, Bayesian Regularization works well in this situation by automatically adjusting the regularization settings. It changes the objective function to discourage large weights, which helps create models that are smoother and less complex. The new objective function ***E(w)*** combines the sum of squared errors ($\boldsymbol{E_D}$) with the sum of squared weights ($\boldsymbol{E_W}$).

$$E(W) = \beta * E_D + \alpha * E_W = \beta * \sum_{i=1}^{N} \left(y_i - \hat{y}_i \right)^2 + \alpha * \sum_{j=1}^{w} w_j^2 \qquad \text{(II.13)}$$

Where:

- w is the vector of all network weights.
- α and β are regularization parameters that are automatically optimized within the Bayesian framework. The term $\boldsymbol{\beta E_D}$ measures the data fit, while $\boldsymbol{\alpha E_W}$ acts as a penalty for model complexity.

This approach not only prevents overfitting but also offers a systematic method to manage noise in the target data **[59]**.

The direct consequence of this modified objective function is a new update rule for the parameters. The gradient of **E(w)** includes a term from the data error **(∇E_D**, computed via backpropagation) and a term from the weight penalty **(∇E_W**, which is simply 2w). For a basic gradient descent update, this results in the following final update equations:

$$W_y \leftarrow W_y - \alpha_{lr} * \left(\beta * \frac{\partial E_D}{\partial W_y} + \alpha * W_y \right) \quad \text{(II.14)}$$

$$W_h \leftarrow W_h - \alpha_{lr} * \left(\beta * \frac{\partial E_D}{\partial W_h} + \alpha * W_h \right) \quad \text{(II.15)}$$

Where $\boldsymbol{\alpha_{lr}}$ is the learning rate. These equations show how the final update for each weight is a balance between fitting the data (the $\boldsymbol{\beta}$ term) and keeping the weights small to prevent overfitting (the $\boldsymbol{\alpha}$ term).

Summary of the Training Process

To clarify our specific methodology, this section details the step-by-step procedure used to train the NARX network. Our approach relies on the Series-Parallel mode and is guided by the principles of Bayesian Regularization. Algorithm 2.1: Training a NARX Network (Series-Parallel Mode).

Initial Setup:

Before we start, we need our training data (the inputs u(t) and the real outputs y(t)) and the defined MLP architecture. For each time step, the model's input is prepared as a regression vector, φ(t), as defined in **(Eq. II.3)**. We give the network its initial starting point by setting all its weights and biases to small, random numbers. The process also begins with initial estimates for the regularization parameters, α and β.

The Training Loop:

The training itself is an iterative process. Instead of running for a fixed number of times, the loop continues until the model's performance stops improving. In each cycle of the loop, the following steps are performed:

Step 1: Check the model's current fit. First, we run all the training data through the network to get its current predictions (ŷSP(t)). We then measure how much the model's predictions differ from the real data by calculating the sum of squared errors. This term, **E_D**, tells us how well the model is fitting the data.

Step 2: Balance accuracy with simplicity. Next, we look at the size of the network's weights. A model with overly large weights tends to be too complex and might not generalize well. So, we calculate a complexity penalty, **E_W**, which is the sum of all the squared weights. Our final objective is to minimize a combined function, as described in **(Eq. II.13)**, which we represent as **E(W) = β*E_D* + *α*E_W**. This objective pushes the model to be both accurate and simple.

Step 3: Adjust the network's weights. This is the main learning step. Using the combined objective function from the previous step, an optimization algorithm adjusts the network's weights and biases to find a better solution. As mentioned, a powerful second-order method like **Levenberg-Marquardt is highly effective for this part of the process.**

Step 4: Automatically tune the regularization. Finally, in this key feature of the Bayesian approach, the algorithm examines the current weights and intelligently updates the α and β values. This allows the training process to **discover the optimal amount of regularization on its own**, instead of requiring us to set it manually.

This loop repeats, refining the weights and the regularization strength together, until the model converges to a final, stable solution. The result is the set of trained parameters, θ.

II.3.3 Neural State-Space Models (NSSMs)

II.3.3.1 The Principle of Separated Dynamics

The second major family of dynamic networks explicitly adopts a state-space formulation, a standard approach in modern control theory. The core principle of a Neural State-Space Model (NSSM) is to use the network's internal structure to learn a representation of the system's unobservable internal state vector, x(k), and to model its evolution over time.

What we need is a model that can tell us two things: how the system's memory evolves, and what output that memory produces. The standard way to do this in engineering is with the following two equations **[60]:**

$$\begin{cases} x(k+1) = f\left(x(k), u(k), \theta\right) \\ y(k) = g\left(x(k), u(k), \theta\right) \end{cases} \quad \text{(II.16)}$$

Where

$x(k) \in \mathrm{R}^{nx}$: is the internal state vector, representing the system's memory or accumulated knowledge at time step *k*.

$u(k) \in \mathrm{R}^{nu}$: de notes the external input vector at time step *k*

$y(k) \in \mathrm{R}^{ny}$: is the observed output vector.

Θ : represents the set of all learnable network parameters (weights and biases).

$f : \mathrm{R}^{nx} \times \mathrm{R}^{nu} \rightarrow \mathrm{R}^{nx}$ is the state transition function.

$g: \mathrm{R}^{nx} \times \mathrm{R}^{nu} \rightarrow \mathrm{R}^{ny}$: is the output function.

For a "black-box" model, the challenge is to learn the complex, high-dimensional nonlinear mappings *f* and ***g*** directly from data. A common and effective architecture for this task is illustrated in Figure II .4.

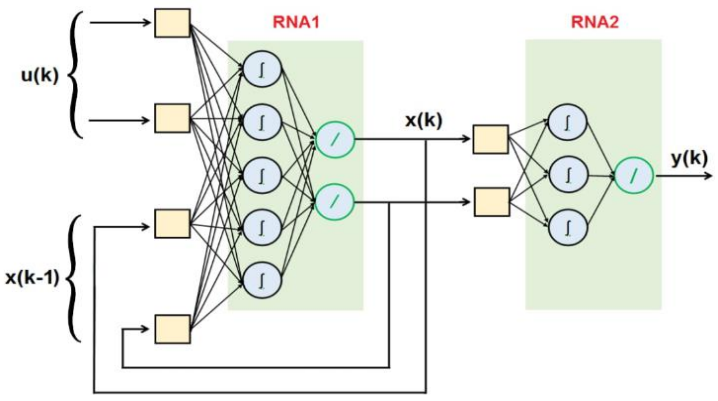

Figure II.4: Architecture of a "black-box" State-Space Neural Network (SSNN) **[61]**:.

It uses a recurrent sub-network (RNA1) to learn the state transition function f(·) and a feedforward sub-network (RNA2) to learn the output function g(·).

As shown in the figure, this architecture cleverly decomposes the problem into two main components, directly corresponding to the functions in our state-space equations **[61]:**

- A **recurrent sub-network** (RNA1) is responsible for updating the state. It takes the previous state $x(k-1)$ and current external input $u(k)$ to compute the new state $x(k)$. This sub-network models the system's state transition function $f(\cdot)$.
- A **static feedforward sub-network** (RNA2), responsible for computing the output y(k)$y(k)$ from the current state $x(k)$. This models the output function $g(\cdot)$.

This explicit separation between state dynamics and output mapping is a hallmark of the state-space modeling approach. Implementations often use advanced recurrent units such as BiLSTMs to create robust "digital twins" of complex nonlinear systems, facilitating tasks like prediction, control, and fault diagnosis.

II.3.3.2 Learning in State-Space Models

The training of Neural State-Space Models (NSSMs) is an optimization problem focused on estimating the parameters θ of the neural functions f and g. The central goal is to minimize the prediction error between the model's output ŷ(k) and the measured output y(k) across an entire time sequence.

To do this, we define an objective function J(θ). The standard choice is the Mean Squared Error, which sums the squared errors over all N time steps:

$$J(\theta) = \frac{1}{2}\sum_{k=1}^{N} \left\| y(k) - \hat{y}(k;\theta) \right\|^2 \qquad \text{(II.17)}$$

Solving this optimization problem is challenging due to the model's recurrent architecture. It requires specialized algorithms capable of handling temporal dependencies. In the following sections, we will detail the most prominent of these, starting with Backpropagation Through Time (BPTT).

II.3.3.2.1 Backpropagation Through Time (BPTT)

The cornerstone algorithm for minimizing the cost function J(θ) in a recurrent context is Backpropagation Through Time (BPTT). The core difficulty it addresses is the temporal dependency: the state x(k) depends on x(k-1), meaning decisions made in the past influence errors that occur in the present.

The BPTT algorithm comes from the foundational work of researchers like Paul Werbos **[62].** His key insight was to stop thinking of the model as a loop and instead to 'unfold' it through time. This effectively turns the recurrent network into a deep, feedforward one—an idea that remains central to modern textbooks on the subject **[63].** This unfolded network consists of N layers, one for each time step, where all layers share the same set of weights. This transformation makes it possible to apply the chain rule for derivatives across the entire temporal sequence.

a- The Two-Stage Gradient Computation

For the specific hybrid architecture of our SSNN, BPTT is implemented as a two-stage process, a procedure well-established in the system identification literature **[15]**. The process traces the error signal backwards through both sub-networks:

1. ***Stage 1: Backpropagation through the Static Output Network (RNA2).***

   The process starts at the final output. For each time step k, the error is first propagated backwards through the static output network, RNA2. This is a standard backpropagation step that computes the gradients for RNA2's parameters (θ_g). Most importantly, this stage calculates the "error signal for the state," $\delta x(k)$, which quantifies how much the calculated state x(k) contributed to the final error.

2. ***Stage 2: Backpropagation Through Time for the State Network (RNA1).***

   This error signal $\delta x(k)$ then becomes the input for training the dynamic state network, RNA1. The BPTT algorithm propagates this error not only backwards through the layers of RNA1 at the current time k, but also backwards *through time* to the previous state x(k-1). To make this work, the calculation starts at the end of the sequence and works its way backward. This step-by-step process is what allows the algorithm to accurately measure how the parameters of RNA1 (θ_f) influenced all the errors that came after.

b- The Inherent Challenge of BPTT: Vanishing and Exploding Gradients

In theory, BPTT is a great idea. But in practice, it hits a major roadblock when dealing with long sequences. This is the well-known problem of 'vanishing and exploding gradients.

What happens is that as the error signal travels backwards in time, it gets multiplied over and over again. After many steps, this repeated multiplication can cause the signal to either fade away to almost nothing (vanish) or become so massive that it's useless (explode).

Mathematically, this issue comes from applying the chain rule to find the gradient of the loss function with respect to a hidden state htht that occurred many steps earlier. The calculation ends up being a long sequence of multiplied Jacobian matrices.

$$\frac{\partial J}{\partial h_t} = \left(\frac{\partial J}{\partial h_N} \right) * \prod_{k=t+1}^{N} \frac{\partial h_k}{\partial h_{(k-1)}} \tag{II.18}$$

If the values in the Jacobian matrices (∂h_k / $\partial h_{(k-1)}$) are consistently smaller than 1, this product shrinks exponentially toward zero, causing the gradient to vanish. If they are consistently larger than 1, they grow exponentially, causing the gradient to explode.

This phenomenon, rigorously analyzed in early work by Hochreiter **[64]** and Bengio et al. **[65],** makes it extremely difficult for the model to learn dependencies between events that are far apart in time. This is the real consequence of a vanishing gradient: the model loses its ability to connect an error to something that happened far in its past. The learning signal that carries this crucial information simply fades out before it can reach its destination.

c- Practical Optimization and Modern Solutions

So, given these serious challenges, how do we make BPTT work in the real world? Researchers have developed a set of powerful techniques that are now a standard part of training any dynamic model.

One of the first and most direct solutions is Truncated BPTT (TBPTT). When you're dealing with very long sequences, running backpropagation all the way to the beginning is not only slow but also increases the risk of the gradient vanishing or exploding. The idea behind TBPTT is simple: we just stop the process after a fixed number of recent time steps. While this makes training much faster and more stable, it comes with an obvious trade-off: the model is now unable to learn from events that occurred outside of this shorter time window.

Beyond simply shortening the sequence, modern adaptive optimizers like Adam **[66]** or RMSprop are essential. Instead of using a single gradient descent update rule :

$$\theta \leftarrow \theta - \alpha * \nabla \omega J(\theta)$$

they intelligently maintain an individual learning rate for each parameter. Adam achieves this by using estimates of the first moment (the mean, m_t) and the second moment (the uncentered variance, v_t) of the gradients.

The final parameter update θ looks like this:

$$\theta_{(t+1)}=\theta_t-\left(\alpha/\left(\sqrt{\hat{v}_t}+\varepsilon\right)\right)*\hat{m}_t \tag{II.19}$$

Here, $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected estimates of the gradient's momentum and its squared magnitude, respectively. This dramatically improves stability and helps the model converge much more quickly.

For moderately sized networks, common in system identification, we can even use more powerful second-order methods. An algorithm like Levenberg-Marquardt **[67]** goes a step further by approximating the curvature of the loss surface using the Hessian matrix H. Its update rule elegantly blends the Newton method with gradient descent:

$$\theta_{(t+1)}=\theta_t-\left(H_t+\lambda I\right)^{-1}*\nabla J\left(\theta_t\right) \tag{II.20}$$

The damping factor λ is the key: when λ is small, the algorithm acts like the fast-converging Newton's method. When λ is large, it safely reverts to a small step in the direction of gradient descent.

Of course, a model that learns well must also generalize well. A significant danger when training any network is that it might just memorize the training examples instead of learning the underlying rules. This problem is called 'overfitting'. To prevent this, we use regularization techniques.

A common way to do this is by adding a 'complexity cost' during training. This idea is formalized by adding a penalty term to the original cost function J(Θ). For L2 regularization (also known as weight decay), the new objective becomes:

$$J_{reg}\left(\theta\right)=J\left(\theta\right)+\left(\lambda/2\right)*\left\|\theta\right\|^2 \tag{II.21}$$

The second term is the L2 penalty (the "L2" refers to the L2 norm, a mathematical way to measure a vector's magnitude. In L2 regularization, the squared version of this norm, Σw^2, is used as a complexity penalty to push the network towards simpler, more robust solutions). This penalty discourages large weights, guiding the network to find a less complicated solution.

These bar charts (Figure II.5) illustrate how L2 regularization shrinks the weight values, helping to simplify the model and reduce overfitting.

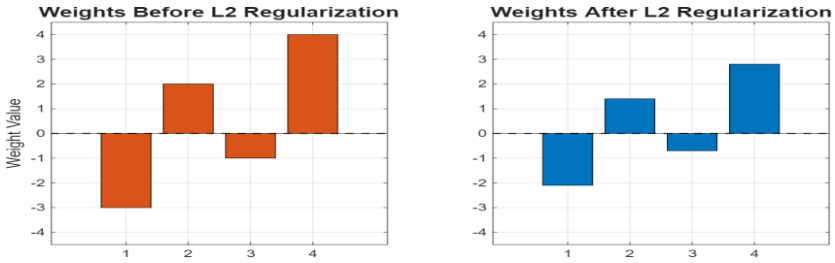

Figure II.5: Comparison of Weight Magnitudes Before and After L2 Regularization

The more sophisticated Bayesian Regularization technique developed by MacKay **[68]** builds on a similar principle but automatically tunes the strength of this penalty during training. The Gaussian curve represents the Bayesian prior distribution on weights (Figure II.6), showing how prior knowledge guides the learning process towards more robust solutions.

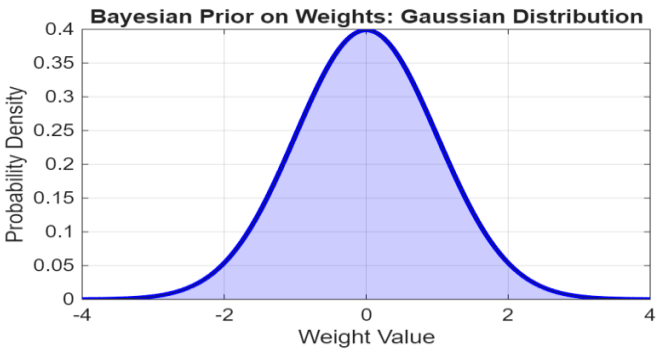

Figure II.6: Bayesian Prior Distribution on Model Weights Represented by a Gaussian

Finally, none of these advanced techniques can work properly if the model isn't set up correctly from the start. Careful initialization of the network's weights and normalization of the input and output data are fundamental steps for ensuring numerical stability throughout the training process.

So, to put it all together, training a Neural State-Space Model is a two-part challenge. On one hand, we need BPTT to handle the way the model's memory flows through time. On the other hand, BPTT on its own isn't enough. We have to combine it with a whole toolkit of modern optimizers and practical tricks to get past its limitations. It's this combination that ultimately allows us to train models that can truly learn the complex dynamics of a nonlinear system.

II.3.3.2.2 Alternative and Advanced Optimization Algorithms

Backpropagation Through Time (BPTT) gives us the gradients needed for training neural state-space models, but how we use these gradients to update the model's parameters Θ really matters. This choice affects how fast and well the model learns, and how stable the training process is. Let's take a look at the main groups of optimization algorithms, starting from basic improvements to gradient descent and moving towards more advanced techniques.

a- Enhancing Gradient Descent

The first group of methods improves upon basic gradient descent by adding smart features to speed up learning and reduce instability.

- **Stochastic Gradient Descent (SGD):** Instead of calculating the gradient over the whole dataset, SGD uses small random chunks called mini-batches. This makes parameter updates faster and introduces some randomness that helps avoid getting stuck in bad local minima. The update rule is:

$$\theta_{(t+1)} = \theta_t - \eta * \nabla J\_B\left(\theta_t\right) \tag{II.22}$$

where B : is the selected mini-batch.

Momentum: Sometimes SGD oscillates and moves slowly. Momentum adds a velocity term v that remembers past gradients exponentially:

$$v_{(t+1)} = \gamma * v_t + \alpha * \nabla J(\theta_t) \quad \text{(II.23)}$$

$$\theta_{(t+1)} = \theta_t - v_{(t+1)} \quad \text{(II.24)}$$

Here, $\gamma \approx 0.9$ controls how much the update "remembers" past directions, smoothing progress.

- **Adaptive Learning Rate Methods (RMSprop & Adam):** These methods tweak the learning rate individually for each parameter based on recent gradient behavior. **RMSprop** keeps track of a running average of the squares of recent gradients to adjust the learning rate for each parameter. **Adam** combines momentum-like term (m_t) and a scaling term based on the squared gradients (v_t) **[66]:**

First, it calculates the moving averages for the first and second moments of the gradient:

$$m_t = \beta_1 * m_{(t-1)} + (1-\beta_1) * \nabla J(\theta_t) \quad \text{(II.25)}$$

$$v_t = \beta_2 * v_{(t-1)} + (1-\beta_2) * (\nabla J(\theta_t))^2 \quad \text{(II.26)}$$

A key innovation in Adam is that it then corrects for the initialization bias of these moving averages, which is especially important during the early stages of training. This results in the bias-corrected estimates $\hat{m}_t$ and $\hat{v}_t$:

$$\hat{m}_t = m_t / (1 - \beta_1 \wedge t) \quad \text{(II.27)}$$

$$\hat{v}_t = v_t / (1 - \beta_2 \wedge t) \quad \text{(II.28)}$$

These corrected estimates are then used in the final parameter update, which scales the momentum by the square root of the variance:

$$\theta_{(t+1)} = \theta_t - \left(\alpha / \left(\sqrt{\hat{v}_t} + \varepsilon \right) \right) * \hat{m}_t \quad \text{(II.29)}$$

Adam is now often the default optimizer because of its robustness and speed. These hyperparameters are typically set as $\beta = 0.9$, $\beta = 0.999$, and $\varepsilon = 10^{-8}$. Adam is now often the default optimizer because of its robustness and speed.

b- Moving Beyond the Gradient: Quasi-Newton Methods

Instead of just relying on the gradient (the slope), quasi-Newton methods also approximate the curvature of the loss surface using the Hessian matrix $H = \nabla^2 J$.

- **BFGS and L-BFGS:** These build an approximate inverse Hessian H^{-1} iteratively. This allows direct, more informed parameter updates and often faster convergence, especially for moderately sized networks.

- **Levenberg-Marquardt (LM):** This algorithm **[67]** combines the fast Newton method and more stable gradient descent:

$$\theta_{(t+1)} = \theta_t - \left(H_t + \lambda I\right)^{-1} * \nabla JB\left(\theta_t\right) \tag{II.30}$$

The damping parameter λ controls the balance: small λ means Newton-like steps, while larger λ makes the steps more cautious.

C. Probabilistic and Heuristic Approaches

For difficult, non-differentiable, or very complex error surfaces, other optimizers work differently.

- **Bayesian Optimization:** This method, building on the ideas of researchers like MacKay [68], does not restrict learning to a single Θ but rather infers a posterior distribution $p(\Theta|D)$. The objective function,

$$J\_Bayes(\theta) = \log p(D|\theta) - \log p(\theta) \tag{II.31}$$

captures both data likelihood and parameter regularization, while offering uncertainty estimates for predictions.

- **Extended Kalman Filters (EKF):** Originating in control theory, EKF can recursively estimate states and parameters as new data arrives. It's ideal for online learning or changing systems where data is sequential.

- **Metaheuristics:** When the loss landscape is rough with many traps, population-based methods like Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO) explore the space broadly and globally. Although these can be computationally heavy, they are good at avoiding poor local minima.

To better understand these differences, Figure II.7 shows how several optimization methods reduce the error during training.

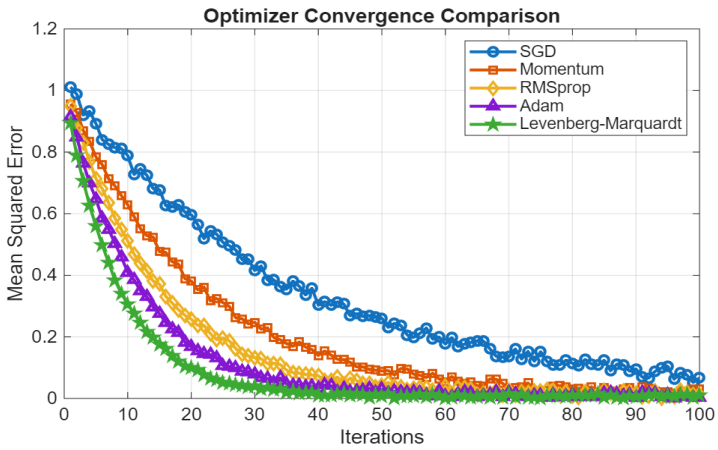

Figure II.7: Convergence Comparison of Different Optimizers on Model Training.

It clearly illustrates that methods like Adam and Levenberg-Marquardt reach lower errors faster than SGD and Momentum. This chart gives a practical view of the trade-offs between speed and effectiveness when tuning dynamic neural network models.

To sum up, choosing an optimization strategy for dynamic neural networks requires balancing computational demands, speed of convergence, and overall robustness. Although adaptive approaches such as Adam have become widely used thanks to modern deep learning frameworks, alternative methods still play an important role in specific applications, particularly in system identification and online learning.

II.3.4 Recurrent Neural Networks (RNNs): The Intrinsic Memory Paradigm

II.3.4.1 The Shift to a Unified, Flexible Architecture

We now arrive at the third and most influential family of dynamic networks: Recurrent Neural Networks (RNNs). While Neural State-Space Models introduce the powerful concept of an internal state, they often do so with a rigid, pre-defined structure (e.g., separating state and output functions into distinct sub-networks). RNNs propose a more flexible and unified approach.

In an RNN, the internal memory is not an explicit, separated state vector but an emergent property of the network's core architecture. This is achieved by incorporating feedback loops directly at the neuron level, allowing information to persist and evolve organically as the network processes a sequence. This creates a powerful, fully learned internal memory.

II.3.4.2 The Simple RNN (Elman Network)

The foundational RNN architecture, often referred to as a Simple RNN or an Elman Network, was introduced by Jeffrey Elman **[69]**, it is defined by a pair of equations that govern its hidden state h(k) and output y(k) at each time step k. The hidden state acts as the network's memory, updated at each step by both the new input and its own previous state.

The general equations for a simple RNN are:

$$\begin{cases} h(k) = \sigma_h(W_{xh} * x(k) + W_{hh} * h(k-1) + b_h) \\ y(k) = \sigma_y\left(W_{hy} * h(k) + b_y\right) \end{cases} \quad \text{(II .32)}$$

Where:

- h(k) ∈ R^nh is the hidden state vector (the memory).
- x(k) ∈ R^nu is the input vector.
- y(k) ∈ R^ny is the output vector.

- W_{xh}, W_{hh}, W_{hy} are the weight matrices for the input-to-hidden, hidden-to-hidden (recurrent), and hidden-to-output connections.
- b_h and b_y are the bias vectors.
- σ_h (e.g., tanh, ReLU) and σ_y (e.g., linear, softmax) are the activation functions.

This architecture is visually represented in Figure II.8. The diagram shows how the input x(k) and the delayed hidden state h(k-1) are combined to produce the new state h(k), which in turn generates the final output y(k) .

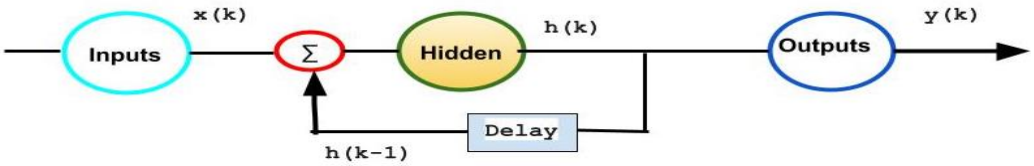

Figure II.8: Architecture of a simple Recurrent Neural Network (Elman Network), *illustrating the recurrent connection via a one-step time delay of the hidden state.*

The key component is the recurrent weight matrix W_hh, which allows the network to control how much of its past state is carried over to the present.

II.3.4.3 Specialized Recurrent Architectures

A basic RNN offers a solid starting point, but over time, researchers have introduced many tweaks to make it work better or suit particular tasks. These changes are usually grouped by how the feedback connections are set up and how complex they are.

- **Locally Recurrent Networks:** This category includes the Elman network. Another classic example is the Jordan network proposed by Michell .Jorden **[70]**. Instead of feeding back the hidden state h(k-1), a Jordan network feeds back the output of the previous time step, y(k-1). This creates a more direct link between the network's past predictions and its current state. The state update equation for a Jordan network becomes:

$$h(k) = \sigma_h\left(W_{xh} * x(k) + W_{yh} * y(k-1) + b_h\right) \quad \text{(II.33)}$$

The specific structure of this output feedback loop is illustrated in Figure II.9. Unlike the Elman network, the recurrent connection originates from the output y(k) before being delayed.

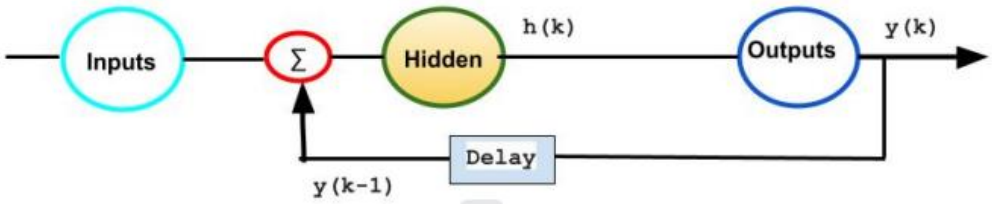

Figure II.9: Architecture of Jorden Network,*which features a recurrent connection from the delayed output y(k-1)*

- **Globally or Deeply Recurrent Networks:** To increase the model's capacity and learn more complex temporal hierarchies, recurrence can be extended across multiple layers. In a **stacked (or deep) RNN**, the output of one recurrent layer becomes the input to the next. This allows the network to process information at different timescales, with lower layers capturing short-term patterns and higher layers capturing longer-term dependencies. The state update for a layer (l) in such a network is:

$$\hat{h}_1(k) = \sigma_h\left(W_{x\hat{h}_l} * \hat{h}_{(l-1)}(k) + W_{h\hat{h}_l} * \hat{h}_l(k-1) + b_{\hat{h}_l} \right) \quad \text{(II.34)}$$

Where : $\hat{h}_0(k)$ is the input x(k).

The architecture of a two-layer Stacked RNN is illustrated in Figure II.10.

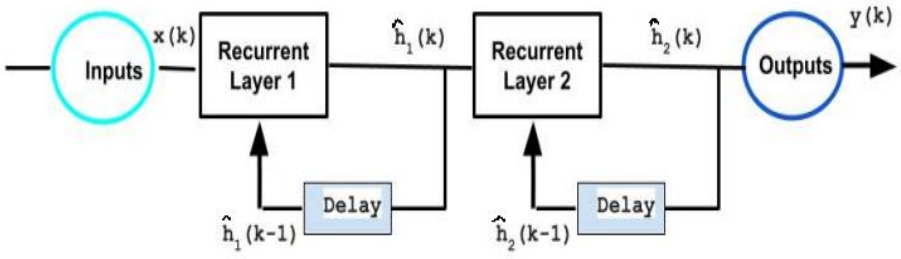

Figure II.10: Architecture of a deep (or Stacked) RNN with two recurrent layers. *The output of the first layer, $\hat{h}_1(k)$, serves as the input sequence for the second layer. Each layer maintains its own internal recurrent connection.*

It clearly shows how the output of the first layer ($\hat{h}(k)$) is passed as input to the second layer, while each layer maintains its own independent recurrent loop via its delayed hidden state ($\hat{h}(k-1)$ and $\hat{h}(k-1)$ respectively).

II.3.4.4 Other Architectural Variants

Beyond the standard RNN structures, Researchers have introduced new hybrid models that aim to bring together the advantages of various approaches:

a- Hybrid Models:

These architectures are designed to explicitly separate and handle different types of complexity by combining static and dynamic components. For example, a static feedforward network might act as a non-temporal feature extractor, processing the current input x(k). This pre-processor does its job first, and then hands off its output to a recurrent part of the network. This second part is the specialist for anything that changes over time.

This kind of modular design works really well when you can clearly separate a system's fixed nonlinearities from its time-varying behavior. We can sketch out this general idea as:

$$y(k) = f\_dynamic\big(g\_static\big(x(k)\big), \dots \dots\big) \tag{II.35}$$

This approach is conceptually linked to the NARX paradigm, where a static network processes a regression vector containing past values.

b- Volterra Networks:

For problems where a rigorous mathematical foundation is essential, Volterra Networks offer a structured approach based on the Volterra series expansion, a classical tool for representing nonlinear systems with memory **[70,71].** A Volterra network takes a different approach from a standard RNN. Rather than learning the system's overall behavior in a black-box way, it's specifically built to model the system's 'DNA'—a set of functions called Volterra kernels.

These kernels are a powerful extension of the impulse response idea from linear systems. They capture not just the direct linear effects, but all the complex, high-order ways the system can behave. Mathematically, y(k) is expressed as a weighted sum of inputs and products of inputs over past time steps, capturing nonlinear dependencies through the Volterra kernels.

$$y(k) = h_0 + \sum_i h_1(i)\,x(k-i) + \sum_i \sum_j h_2(i,j)\,x(k-i)\,x(k-j) + \dots \quad \text{(II.36)}$$

Where :

- h_0 : is a simple bias or offset.
- The first-order kernel h_1 captures the linear dynamics (like a standard FIR filter).
- The second-order kernel h_2 captures the quadratic interactions between past inputs, and so on for higher orders.

The key idea here is that a neural network (often an MLP) can be structured and trained to explicitly approximate the coefficients of these kernels, h_1, h_2,... This establishes a direct link between the network's weights and a formal mathematical description of the system.

c- Conceptual Distinction

In essence, these two approaches offer a powerful trade-off. Hybrid models provide practical architectural flexibility, allowing a designer to combine different building blocks to suit a specific problem. Volterra networks, on the other hand, offer a trade-off: you get a more formal mathematical model instead of just architectural flexibility. This gives you a more transparent window into what the model is doing, since the network's parameters are tied to specific, well-defined nonlinear dynamics.

Ultimately, both hybrid models and Volterra networks are valuable tools that extend what we can do with classic recurrent architectures, especially when we're faced with highly complex systems.

As we will see in the following section, the learning process for all these architectures is unified by a single theoretical principle**.** However, the simple RNN and its variants, despite their power, suffer from significant limitations in learning long-term dependencies. This very challenge motivated the development of more sophisticated architectures like **LSTM and GRU**, which are at the heart of modern deep learning for sequential data and are central to the contributions of this thesis.

II.4 The General Learning Framework: Unfolding and the "Copy" Concept

Now that we have reviewed the main types of recurrent neural networks, a key question comes up: how can we train models that have feedback loops? Most training methods rely on networks without loops, because they make computing gradients easier. The common way to handle this is to imagine the recurrent network as a deep feedforward network. This means unfolding the network over time into a long chain of copies that each represent the network at one moment. This approach allows us to apply standard training techniques to models with loops.

II.4.1 The Modular Representation and the "Copy" Concept

This transformation is achieved by unfolding the recurrent network into its canonical form. This principle, which is central to the system identification literature **[73,74]**, allows us to replace the recurrent network with a cascade of N non-recurrent networks called "copies", where N corresponds to the number of time steps. This modular representation is a concept remains a cornerstone of how recurrent models are trained today, as highlighted in recent deep learning literature **[75].**

Each "copy k" represents the original network at a particular time step k. While the input vectors and neuron activations vary between copies, all copies share the same parameters θ. The state outputs of copy k, noted $S_{out}(k)$, become the state inputs $S_{in}(k+1)$ for copy k+1, bridging the temporal dependency .

The overall structure of this learning network is illustrated in Figure II.11. The figure highlights the two primary configurations that arise, depending on how temporal information is passed between the copies.

As the figure shows, the key distinction lies in the state connections:

Figure II.11(a) represents the case where there is no internal feedback loop between the copies. Each copy is independent in terms of its state. This corresponds to situations where all past information is provided from the dataset (a method often called "teacher-forcing").

Figure II.11(b) represents the truly recurrent case. Here, the state output $S_{out}(k)$ from copy k is passed as the state input $S_{in}(k+1)$ to the next copy. This explicitly models the system's evolving internal memory.

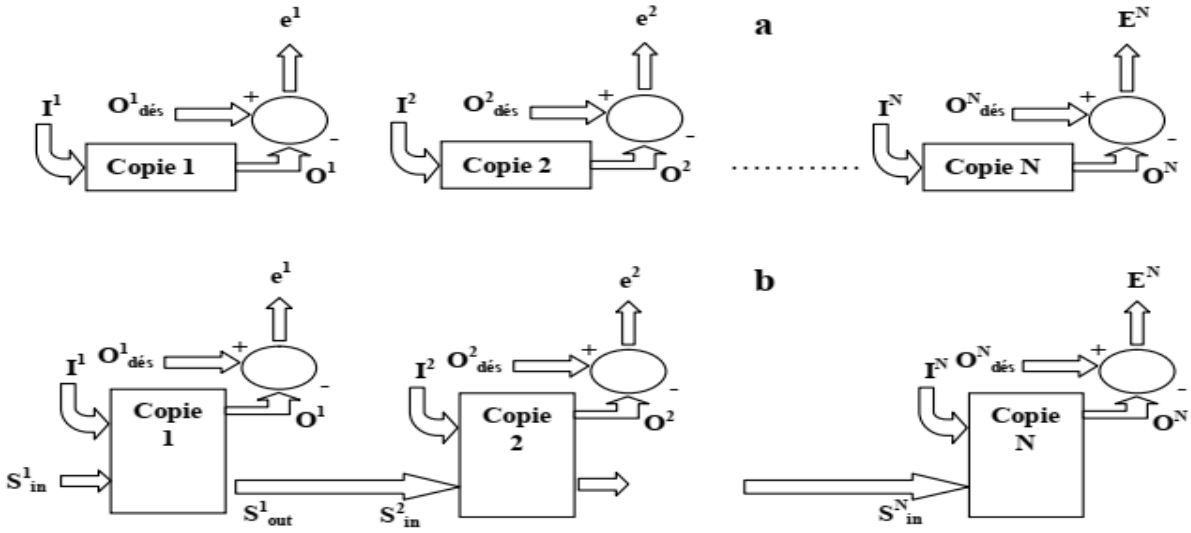

Figure II.11: The learning network configurations formed by N copies: **a)** for a model with external feedback (non-recursive), and **b)** for a model with internal feedback (recursive).

To understand the specific signals involved at each step of this chain, we can examine the detailed structure of a single 'copy k', as shown in Figure II.12.

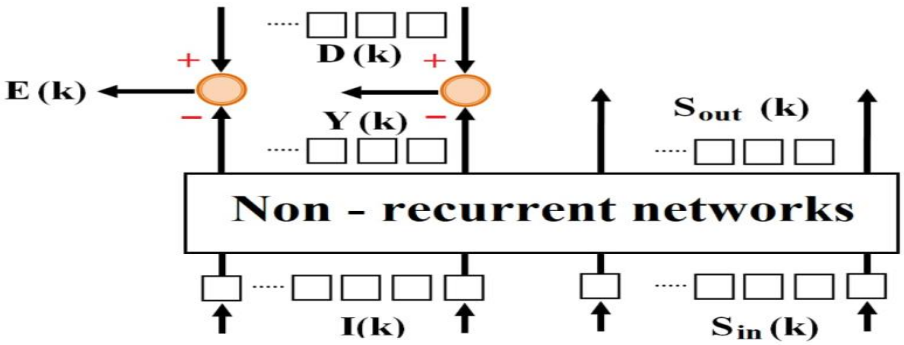

Figure II.12: Modular representation of a recurrent network as a non-recurrent "copy" at time step k

As this detailed diagram shows, the inputs and outputs of each copy in the recurrent case (Figure II.11(b)) can be formally described as follows:

- The inputs to the k-th copy are thus composed of two parts **[75]**:

  ✓ The imposed inputs **I(k),** wich include external signals $\boldsymbol{u}$ and sometimes past measured outputs $\boldsymbol{y_p}$**.** In other words, they can be written as:

$$I(k)=\left(u(k);u(k-1),\ldots,u(k-m+1),y_p(k),\ldots,y_p(k-n+1)\right) \quad \text{(II.37)}$$

  ✓ The state inputs **S $_{in}$(k)** (or looped inputs), which are the outputs and states from previous time steps generated by the model itself:

$$S_{in}(k)=\left(O(k-1),\ldots,O(k-n),x(k-1),\ldots,x(k-n)\right) \quad \text{(II.38)}$$

- Similarly, the outputs of copy k are also of two types:

  ✓ The predicted outputs **O(k)**, which are compared to their desired values O_{des}^k, are used to compute the instantaneous error:

$$E(k)=O_{des}(k)-O(k) \quad \text{(II.39)}$$

  ✓ The state outputs **S $_{out}$(k)**, which are passed to the next copy, satisfying the temporal consistency constraint:

$$S_{in}(k+1)=S_{out}(k) \quad \text{(II.40)}$$

This unfolded structure converts the training of the original recurrent network into training on a deep, feedforward-like network of N copies with shared weights. This robust framework is precisely what allows gradient-based algorithms such as Backpropagation Through Time (BPTT) to be applied effectively.

II.5 The Core Learning Algorithm: BackPropagation Through Time (BPTT)

II.5.1 The General Principle of BPTT

The modular framework of unfolded "copies" provides the theoretical foundation for training all recurrent networks. We will now detail the general algorithm that operates on this structure, known as Backpropagation Through Time (BPTT). While we have seen a

specific two-stage application of this algorithm for the hybrid architecture of NSSMs, we will now describe its fundamental principle, which applies to any unfolded recurrent model **[62].** The primary goal of BPTT is to compute the gradient of the total cost function J(θ) with respect to the network's shared parameters θ. From the "copy" view, a key takeaway is that the overall gradient equals the sum of the gradients calculated at each of the N copies.

$$\nabla\_\theta J(\theta) = \sum_{k=1}^{N} \nabla\_\theta J\left(\theta(k)\right) \tag{II.41}$$

Here, ∇_θJ(θ(k)) represents the gradient contribution from the k-th copy alone. BPTT provides a systematic and computationally efficient way to calculate this total gradient by applying the chain rule of calculus recursively through the unfolded network.

II.5.2 The BPTT Algorithm in Practice

The algorithm follows a cycle of three steps. This cycle repeats again and again for every training epoch.

- **Step 1: The Forward Pass.**

The process begins by feeding the entire input sequence through the learning network, from the first copy (k=1) to the last (k=N). At each time step k, the outputs O(k) and the state outputs S_{out}(k) are computed. A critical requirement for the algorithm is to store all the intermediate activations within each copy, as they will be essential for the backward pass.

- **Step 2: The Backward Pass.**

The backward pass is where the core calculations happen. Starting from the end of the sequence (copy N), it moves backwards towards the first element. We want to find the gradient of the cost function with respect to each hidden state *h(k)*.This gradient, $\delta h(k) = \frac{\partial J}{\partial h(k)}$ is computed recursively as the process goes back through the sequence.

$$\delta h(k) = \left(\frac{\partial J}{\partial y(k)} * \frac{\partial y(k)}{\partial h(k)}\right) + \left(\frac{\partial J}{\partial h(k+1)} * \frac{\partial h(k+1)}{\partial h(k)}\right) \tag{II.42}$$

- Let's break this down:

1. The calculation starts at the end, k=N. The term involving h (k+1) is zero, so the error signal is only influenced by the final output.

2. For each preceding step k (from N-1 down to 1), the error signal δh(k) is a sum of two influences: the error from the output at the *current* time step, and the error propagated back from the *future* time step k+1.

3. Once these error signals δh(k) are known for each copy, they are used to compute the local parameter gradients ∇J(θ(k)), which are then summed up as shown in Eq. (II.41).

- **Step 3: The Parameter Update.**

Once the backward pass is complete, we have the total accumulated gradient, ∇J(θ). This final gradient is then used to update the single, shared set of parameters θ using an optimization algorithm like SGD or Adam:

$$\theta_{new} = \theta_{old} - \alpha * \nabla J(\theta) \tag{II.43}$$

This three-step cycle of forward pass, backward pass, and parameter update is the fundamental mechanism for training virtually all modern recurrent neural networks.

II.5.3 Challenges and Optimization Strategies

The implementation of BPTT for complex architectures such as LSTMs and BiLSTMs faces the same theoretical and numerical obstacles encountered in State-Space models. The most significant challenge remains the vanishing and exploding gradient problem, which limits the network's ability to learn long-term dependencies. The mathematical root of this issue, involving the product of Jacobian matrices, was formally established in Section II.3.3.2.1-b.

To ensure robust training and convergence, the optimization framework utilizes the same advanced tools previously described in Section II.3.3.2.2. This includes the use of adaptive optimizers (Adam), second-order methods (Levenberg-Marquardt), and Bayesian regularization to prevent overfitting while maintaining high predictive accuracy.

II.5.4 Summary of the Learning Process

In summary, the effective training of dynamic neural networks is established as a two-fold process. First, structural frameworks such as unfolding and the BPTT algorithm enable the model to capture how information persists and flows through time. Second, to overcome the inherent limitations of BPTT, these models are integrated with advanced optimizers and regularization tactics. This synergistic combination is what ultimately allows the networks to accurately identify and model the complex, nonlinear behaviors characteristic of the systems studied in this work.

II.6 Gated Architectures for Long-Term Memory

II.6.1 The Architectural Solution to the Vanishing Gradient Problem

In the last section, we saw that the simple RNN, even though it looks elegant in theory, has a major issue called the vanishing gradient problem. This mathematical hurdle makes it nearly impossible for the model to learn dependencies between events that are far apart in time. Although some algorithms can improve the issue, the strongest and most lasting fix turned out to be changing the architecture itself.

This led to the development of a new class of recurrent units designed explicitly to manage information flow over long sequences. Recent thorough studies **[76]** have shown that these so-called "gated architectures" represented a major breakthrough. Rather than letting information flow freely without control, these models use "gates" — small neural networks that decide which information to keep, which to discard, and which to pass along. The most influential of these is the Long Short-Term Memory (LSTM).

II.6.2 Long Short-Term Memory (LSTM)

The Long Short-Term Memory (LSTM) network, a groundbreaking architecture proposed by Sepp Hochreiter and Jürgen Schmidhuber in their seminal 1997 paper **[77]**, was designed with one primary goal: to overcome the long-term dependency problem of simple RNNs.

The genius of the LSTM lies in its core component: the memory cell. Unlike a simple RNN, which merges all information into a single hidden state, an LSTM unit maintains two distinct states that are passed from one time step to the next:

1. **The Hidden State (h_t):** This is the "working memory" used to make predictions at the current time step.

2. **The Cell State (C_t):** This is the key innovation. Think of the cell state as a conveyor belt that carries information through time with only minor, controlled modifications. This structure is what allows information to persist over long periods without degrading.

The flow of information into and out of this cell state is meticulously regulated by three "gates." Each gate is a small neural network that learns to output a value between 0 (block everything) and 1 (let everything through), controlling what information is kept, discarded, or revealed.

The internal structure of an LSTM unit is visually detailed in Figure II.13**.** The diagram illustrates how the gates interact with the cell state to manage the flow of information. The entire process can be broken down into the following steps:

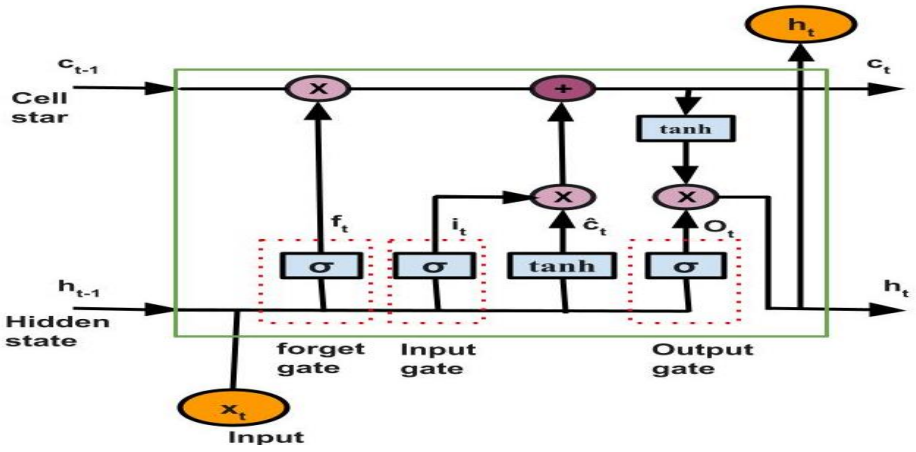

Figure II.13 : The internal structure of a Long Short-Term Memory (LSTM) cell **[77].**

The diagram shows the "cell state" conveyor belt, along with the three gates (Forget, Input, Output) that regulate the information written to, read from, and maintained in the cell.

- **The Forget Gate (f_t):** First, the LSTM decides what information from the past is no longer needed. This gate, a crucial addition to the original concept **[78]**, looks at the current input x_t and the previous hidden state $h_{(t-1)}$ to decide what to "forget" from the old cell state $C_{(t-1)}$.

$$f_t = \sigma\left(W_f * x_t + U_f * h_{t-1} + b_f\right) \quad \text{(II.44)}$$

- **The Input Gate (i_t):** Next, the network decides what new information to store. It uses the **input gate** to determine which values to update, and a tanh layerto form new candidate values $\tilde{C}_t$ through a tanh layer.

$$i_t = \sigma\left(W_i * x_t + U_i * h_{t-1} + b_i\right) \quad \text{(II.45)}$$

$$\tilde{C}_t = tanh\left(W_c * x_t + U_c * h_{t-1} + b_c\right) \quad \text{(II.46)}$$

- **Cell State Update:** The old cell state is then updated by forgetting the old information (multiplying by f_t) and adding the new information (multiplying by i_t):

$$C_t = C_{t-1} \odot f_t + \tilde{C}_t \odot i_t \quad \text{(II .47)}$$

- **The Output Gate (o_t):** Finally, the network computes its new hidden state h_t. It decides which parts of the cell state should be shared outside, and these are passed through a tanh function.

$$o_t = \sigma\left(W_o * x_t + U_o * h_{t-1} + b_o\right) \quad \text{(II.48)}$$

$$h_t = o_t \odot \tanh\left(C_t\right) \quad \text{(II.49)}$$

The symbol σ refers to the sigmoid function, The symbol $\odot$ represents element-wise multiplication. By training the parameters (W, U, b) of these gates, the LSTM builds a clever internal mechanism to remember and use information over long periods.

II.6.3 Gated Recurrent Unit (GRU): A Simpler Alternative

Following LSTM's breakthroughs, efforts were made to simplify the model architecture while preserving its ability to deal with long-term dependencies. This led to the launch of the Gated Recurrent Unit (GRU) by Cho et al. in 2014 **[79].**

The GRU is a clever simplification of the LSTM. As detailed in recent studies on human activity recognition, where GRUs have shown excellent performance **[80]**, its main architectural change is the merging of the cell state and the hidden state into a single, unified hidden state vector**,** $\mathbf{h_t}$. It also streamlines the gating mechanism, reducing the number of gates from three to two. This simpler design means fewer parameters, which often makes the GRU faster to train and less prone to overfitting on smaller datasets.

The internal workings of a GRU cell, as illustrated in Figure II.14, are controlled by two main gates:

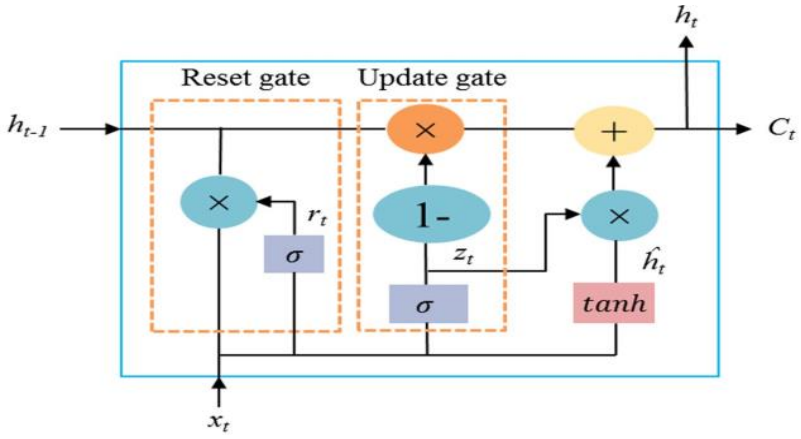

Figure II.14 : Architecture of a Gated Recurrent Unit (GRU) cell **[80]**.

The diagram shows the two gates (Reset and Update) that control the flow of information through the single hidden state.

- **The Reset Gate ($\mathbf{r_t}$):** This gate decides how much of the past information (from $h_{(t-1)}$) should be ignored when creating new candidate information. If the reset gate's value is close to 0, it allows the unit to effectively "reset" its memory and focus mainly on the current input, x_t. This is useful for capturing short-term dependencies or starting a new "thought."

$$r_t = \sigma\left(W_r * x_t + U_r * h_{t-1} + b_r\right) \quad \text{(II.50)}$$

- **The Update Gate (z_t):** This gate acts like a combination of the forget and input gates of an LSTM. It determines what fraction of the past information to keep and what fraction of the *new* candidate information to add.

$$z_t = \sigma\left(W_z * x_t + U_z * h_{t-1} + b_z\right) \quad \text{(II.51)}$$

These two gates are then used to compute the new candidate state $\tilde{\mathbf{h}}_t$ and the final hidden state $\mathbf{h_t}$. The reset gate $\mathbf{r_t}$ is applied to the previous hidden state before calculating the candidate:

$$\tilde{h}_t = tanh\left(W_h * x_t + U_h * r_t \odot h_{t-1} + b_h\right) \quad \text{(II.52)}$$

The final hidden state $\mathbf{h_t}$ is then a balanced mix—a linear interpolation—between the previous state $\mathbf{h_{(t-1)}}$ and the new candidate state $\tilde{\mathbf{h}}_t$, with the update gate $\mathbf{z_t}$ controlling the mixture:

$$h_t = \left(1 - z_t\right) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad \text{(II.53)}$$

II.6.4 Conceptual Distinction: GRU vs. LSTM

The main difference between the two is a trade-off between complexity and power. The LSTM, with its separate cell state, acts like a computer with dedicated RAM, which can be very powerful for precisely storing and retrieving information over very long periods. The GRU merges this memory directly into its "working" state, making it a simpler and faster model.

In real-world usage, the simpler GRU model often provides a speed benefit while maintaining comparable accuracy. Yet, for challenges requiring exact long-distance memory, LSTMs can be more suitable. The final selection is typically based on the dataset and computational limits.

II.6.5 Bidirectional LSTM (BiLSTM): Leveraging Past and Future Context

II.6.5.1 The Limitation of Unidirectional Processing

Standard LSTMs and GRUs, as we have described them, process information in a single temporal direction, from the beginning of a sequence to the end. This unidirectional approach is powerful, but it has a fundamental limitation: it bases its prediction at time **t** only on past and present information (x_1, ..., x_t).

However, in many real-world problems, the context of what happens after a certain point is just as important as what happened before. For example, when analyzing a sentence, the meaning of a word can be clarified by the words that follow it. Similarly, in time-series analysis, the nature of an anomaly might only become clear when you see how the signal behaves after the event. A standard LSTM is blind to this future context.

II.6.5.2 The Bidirectional Principle

The Bidirectional Recurrent Neural Network (BiRNN) was developed as a solution to this problem, with the Bidirectional LSTM (BiLSTM) being its most powerful and widely used implementation **[81]**.

The idea behind a BiLSTM is elegant and intuitive: instead of processing the sequence in one direction, it processes it in two opposite directions simultaneously using two independent LSTM layers. As illustrated in Figure II.15, the architecture is composed of:

1. A **Forward LSTM Layer**, which reads the input sequence from left to right (from t=1 to t=N) and computes a sequence of "forward" hidden states ($\mathbf{h}^{\rightarrow}_t$).

2. A **Backward LSTM Layer**, which reads the same input sequence from right to left (from t=N to t=1) and computes a sequence of "backward" hidden states ($\mathbf{h}^{\leftarrow}_t$).

By integrating these two independent flows, the BiLSTM effectively fuses information from the entire sequence at every single time step. Unlike the standard LSTM which only has access to the preceding history, the bidirectional structure generates a rich latent representation that captures both the 'cause' and the 'effect' of temporal variations. This dual-perspective is particularly advantageous for the objectives

of this thesis. For instance, in modeling photovoltaic I-V curves, the physical behavior near the short-circuit point is often better interpreted when the model has already 'seen' the trajectory toward the open-circuit voltage. Similarly, for fault classification, the BiLSTM can distinguish between transient noise and real degradation by analyzing the signal patterns both before and after a specific event, leading to a much higher diagnostic robustness in complex nonlinear environments.

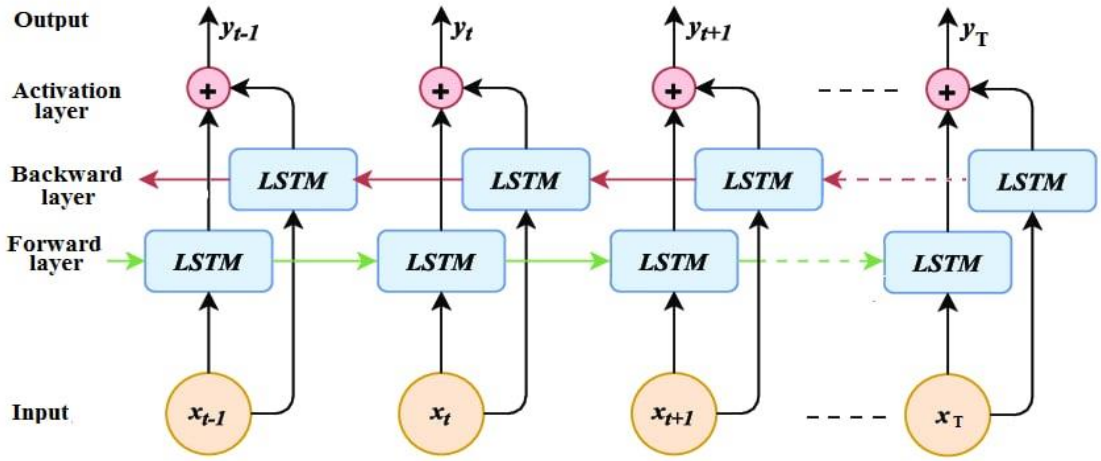

Figure II.15 : The architecture of a Bidirectional Long Short-Term Memory (BiLSTM) network **[82]**.

It consists of two parallel LSTM layers processing the input in opposite directions.

II.6.5.3 Mathematical Formulation

At each time step **t**, the BiLSTM computes its output based on the information from both directions.

The **forward hidden state** $\mathbf{h}^{\rightarrow}_{\mathbf{t}}$ is calculated based on the past context:

$$\vec{h}_t = LSTM\_forward\left(x_t, \vec{h}_{(t-1)}\right) \quad \text{(II.54)}$$

The backward hidden state $\mathbf{h}^{\leftarrow}_{\mathbf{t}}$ is calculated based on the future context:

$$\overleftarrow{h}_t = LSTM\_backward\left(x_t, \overleftarrow{h}_{(t+1)}\right) \quad \text{(II.55)}$$

The final output $\mathbf{y_t}$ at each time step is then a function of *both* the forward and backward hidden states. The most common way to combine them is by *concatenation*, though other operations like summation or averaging are also possible.

$$y_t = g\left(\left[\vec{h}_t; \overleftarrow{h}_t\right]\right) \tag{II.56}$$

Where $\mathbf{g}$ is typically a dense output layer with an appropriate activation function. By concatenating ($\mathbf{\vec{h}_t}$) and ($\mathbf{\overleftarrow{h}_t}$), the model at time $\mathbf{t}$ has access to a rich representation that summarizes information from the entire input sequence, both past and future.

II.6.5.4 Why BiLSTM is Crucial for This Thesis

The dual-directional context processing offered by BiLSTM renders it a crucial component for this thesis's focus. Its strengths are demonstrated in recent applications including complex biological data interpretation **[82],** environmental forecasting **[83]**, and machinery fault diagnosis **[84].**

- For the **State-Space Identification of a PV System (Chapter III)**, the task is to model the complete current-voltage (I-V) curve. The shape of the beginning of the curve is often better understood if the model has already seen the end of it. A BiLSTM can process the entire curve at once, allowing it to make more accurate point-by-point predictions by using global context.

- For the **Fault Detection and Classification in PV Systems (Chapter IV)**, the signature of a fault is not just an instantaneous event but a pattern that unfolds over a time window. A fault might be characterized by a gradual deviation before a peak and a specific recovery pattern *after* the peak. A standard LSTM would only see the "before," whereas a BiLSTM sees the entire event, allowing for a much more robust and accurate classification.

By using a BiLSTM, we are equipping our models with the ability to see the "whole story," not just the events that have already passed, which is a decisive advantage for the complex dynamic systems we aim to model and analyze.

II.7 Conclusion

In this chapter, we've put together the essential toolkit of technologies that we'll need for the rest of this thesis. We've taken a journey through the world of dynamic neural networks, following the story of how these models have evolved to better understand and learn from data that changes over time.

We began by establishing the fundamental challenge of modeling dynamic systems: the need for an internal memory. We first explored the family of Input-Output models, exemplified by the NARX architecture, which cleverly uses an external, fixed-size memory in the form of a regression vector. We then moved to Neural State-Space Models (NSSMs), a more structured approach that introduces an explicit, though often rigid, internal state.

This led us to the more flexible and unified paradigm of Recurrent Neural Networks (RNNs). After looking at the classic Elman and Jordan networks, we ran into their biggest weakness: they really struggle to learn from events that are far apart in time because of the vanishing gradient problem. This major challenge is what sparked the next big idea in our journey: the creation of gated architectures.

This is where we looked at the key tools that solve this memory problem. We dove into the inner workings of the Long Short-Term Memory (LSTM) and its simpler cousin, the GRU—both of which were invented to give these models a reliable way to manage information over long periods.

Our journey ended with a look at the Bidirectional LSTM (BiLSTM), a powerful architecture that takes things a step further by looking at both past and future context at the same time. This model has been identified as the most suitable tool for the specific tasks addressed in this thesis, as its ability to see the "whole story" is a decisive advantage for modeling complex curves and classifying fault patterns.

Now that we have this powerful set of architectural tools in hand and have explained why the BiLSTM is the right choice for our work, we're ready to put these models to the test. In the following chapters, we'll apply them to our two main contributions: building a state-space model for a photovoltaic system in Chapter III, and creating a robust framework for fault classification in Chapter IV.

CHAPTER III

Performance Analysis For Modeling Nonlinear PV Systems

III.1 Introduction

You can't navigate a new city without a good map, and it's the same in science and engineering. To get anything done—whether it's predicting the future, testing new ideas safely in a simulation, or spotting problems before they happen—you need a reliable map of your system. That's precisely what a good mathematical model is: a map that accurately shows you how a real-world system works.

But here's the real challenge: most systems in the real world have a 'memory'. What they do now is shaped by their entire history, not just the latest input. This makes them incredibly complex and nonlinear. This is a big problem for traditional 'white-box' models, which are built on known physical laws. Those models often struggle when things get this complicated, and that's precisely why researchers have turned to data-driven approaches instead **[12]**.

In response to these limitations, artificial neural networks have established themselves as a powerful "black-box" modeling paradigm **[14]**. By leveraging their universal approximation capabilities, Recurrent Neural Networks (RNNs) in particular are ideally suited for system identification in the state-space, a concept pioneered in early works **[2]** and which remains a highly active field of research today **[85, 86]**.

The field of PV modeling is buzzing with activity right now, thanks to the power of deep learning. As researchers move away from traditional models, the work has generally gone down one of two main roads.

The most common road has been to use input-output models. These are great for specific tasks like predicting how much power a plant will generate in the near future. Many recent studies have shown that hybrid models, like those combining CNNs and LSTMs, are very effective for this kind of forecasting **[87, 89]**. Other works have used neural networks or neuro-fuzzy systems to figure out key electrical parameters for tasks like MPPT optimization **[90-91]**. But the big drawback of these models is their narrow focus. They are specialists who only describe a single output or a static snapshot, so they can't capture the full internal story of how the PV system is behaving.

The other, less traveled road is the state-space framework. This approach is fundamentally more powerful because it tries to learn a model of how the system works on

the inside, instead of just memorizing input-output patterns **[2, 92]**. The potential of this approach has been known for a long time, but it has seen a significant revival recently with the rise of deep learning, as highlighted in comprehensive surveys on the topic **[85, 86]**.

Lately, even more advanced tools like Transformers **[93,95]** and newer architectures like Mamba **[96]** have entered the scene. Their great strength is their ability to uncover complex patterns over very long sequences, frequently outperforming traditional RNNs in this regard. Their main weakness, however, is that they are extremely data-hungry, often requiring massive datasets to learn properly. For our work on I-V curve modeling, where we have a well-structured but moderately sized dataset, a powerful recurrent model like the BiLSTM is a more practical and effective choice.

This is where we identified a critical gap in the research. Despite all this exciting work, very few studies have used a data-driven, state-space approach to model the *entire I-V curve* as a dynamic sequence. And, even more importantly, almost no one has checked if the internal state learned by the model actually means something physically. This is precisely where our work comes in.

But we need to look beyond just the numbers. A model that's incredibly accurate at making predictions can still be a failure if it's not robust. It can still be useless if it hasn't learned anything physically correct about the system. And it can be downright dangerous if we use it for critical tasks without knowing if it's truly reliable. This is particularly true for photovoltaic (PV) systems, our chosen case study, whose complex behavior under fluctuating environmental and health conditions makes them a significant modeling challenge **[97]**.

So, the real challenge for us in this chapter is not just about building another model. It's about figuring out a reliable process to create and test a recurrent neural network for a dynamic system. The model we build has to do three things well: it must be accurate, it must make physical sense, and it must be robust enough to handle the messiness of real-world operating conditions.

To answer this question, this chapter details our first primary contribution: a complete methodology for the state-space identification of a PV system's current-voltage (I-V)

characteristic. We frame this as a sequence-to-sequence task and employ a stacked Bidirectional

LSTM (BiLSTM) architecture to create a high-fidelity "digital twin" of the system. Our approach is comprehensive, covering the entire pipeline from data generation to a multi-faceted evaluation of the model's performance, with a particular focus on validating the physical relevance of its learned internal dynamics.

The following sections detail the organization of this chapter. Section III.2 details our complete methodology. Section III.3 presents the results and our in-depth analysis. Finally, Section III.4 concludes the chapter, summarizing our findings and setting the stage for our next contribution.

III.2 Methodology: From Data to a State-Space Model

Our journey from a real-world problem to a working state-space model involved several key stages**.** Starting in the lab with the physical configuration, we collected the required data. Then, we tailored the problem’s format to be suitable for a neural network’s learning process. This led us to the design of our specific BiLSTM architecture, which we then had to carefully train and fine-tune to get the best results.

III.2.1 Experimental Setup and Dataset Generation

There's a simple rule in machine learning: garbage in, garbage out. The quality of any black-box model depends entirely on the quality of the data it learns from. That's why our first and most important job—long before designing the network itself—was to build an excellent dataset that was both detailed and covered a wide variety of scenarios. To do this, we used a high-fidelity, real-time power electronics emulator of a photovoltaic array (PVA), a specialized platform developed and validated at the LEPCI laboratory at the University of Sétif 1 **[98]**.

The heart of this setup, as shown in Figure III.1, is a detailed electrical model of a PV array. This model was created in MATLAB/Simulink and deployed on a dSPACE DS1104 real-time control board. It simulates a PV array made up of six modules connected in series, with each module containing 36 solar cells that follow the well-established Bishop model

[99, 100]. This platform gives us a powerful and flexible environment to generate a huge range of I-V characteristics under perfectly controlled conditions.

Figure III.1. Photograph of the emulator used for data collection: 1) buck converters, 2) load 3) programmable power source, 4) scope, 5) ControlDesk, 6) DS1104 platform **.[98]**

The goal of the dataset was to capture the wide range of situations a PV system might face in the real world. To achieve this, the authors systematically changed the environmental conditions and introduced a wide variety of common faults. This process yieded data for one healthy state and five major fault categories. By breaking these down further by severity and pattern, a total of 22 distinct operational classes were defined.

To make sure our model could generalize well, approximately 130 unique I-V curves were generated for each of these 22 classes by varying the environmental settings. This resulted in a comprehensive database of nearly 2,860 samples. The specific conditions and fault scenarios that were simulated are detailed below:

- **Healthy Condition (NF):** Normal operation where irradiance (G) was varied from 100 to 1000 W/m² and temperature (T) from 0°C to 60°C.
- **Fault 1(F1): Partial Shading (PS)** : 9 different shading patterns were simulated, varying the number of shaded cells (25%, 50%, 75%) and the number of affected modules (one, two, or three).
- **Fault 2 (F2): Increased Series Resistance (ISR):** 5 levels of severity were created by increasing the series resistance of one module by 1 Ω, 5 Ω, 10 Ω, 15 Ω, and 20 Ω.
- **Fault 3 (F3): Bypass Diode Short-Circuit (BPD short):** 1 scenario where a single

bypass diode in the array is short-circuited.

- **Fault 4 (F4): Bypass Diode Impedance Fault (BPD imp) :**5 scenarios where a bypass diode is modeled as a faulty impedance with resistance values of 1 Ω, 5 Ω, 10 Ω, 15 Ω, and 20 Ω.
- **Fault ((F5): PV Module Short-Circuit (Mod short):** 1 scenario where an entire module is short-circuited.

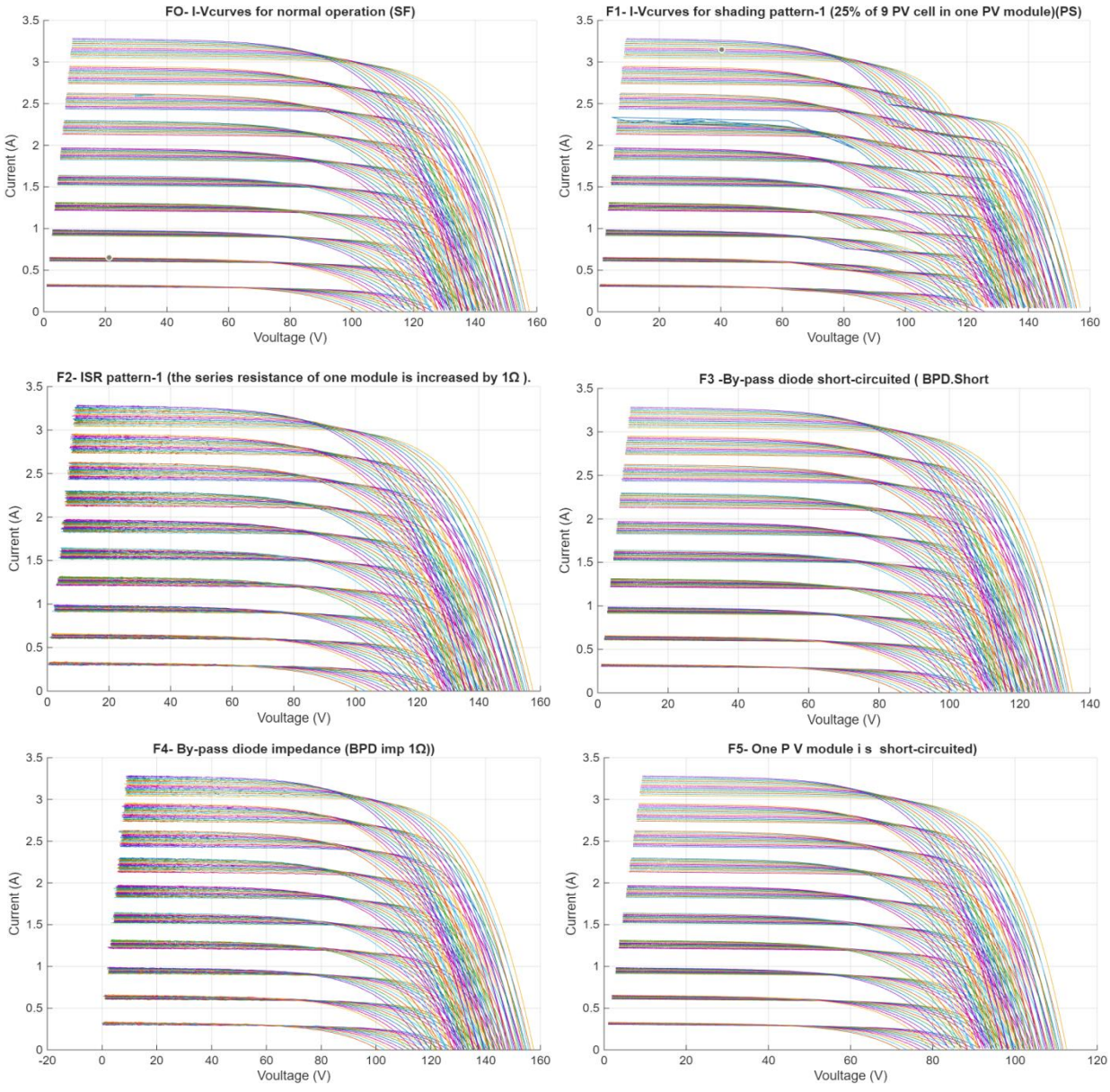

Figure III.2. Comparison of I-V curve families under different operational conditions.

The plots show all 130 curves for the healthy state (NF) and representative examples for each of the five fault classes (F1-F5), demonstrating the rich diversity of the generated dataset.

To visually illustrate the impact of these different operational conditions, Figure III.2 presents a sample set of I-V curves from our generated dataset. It shows the full range of curves produced under healthy conditions (NF) and compares them with representative examples from each of the five major fault categories.

As the plots clearly show, each fault type imposes a unique and complex signature on the I-V characteristic. For example, partial shading (F1) introduces multiple "knees" in the curve, while an increased series resistance (F2) alters its slope. It is this rich diversity of behaviors that provides our BiLSTM model with the necessary information to learn the underlying dynamics of the system.

III.2.2 Problem Formulation: A Unified State-Space Approach

Before focusing on the data particulars, we will first rigorously set up the problem within a state-space context. This will also allow us to clarify how our chosen method differs from the more traditional Neural State-Space Model (NSSM) architecture we discussed in Chapter II.

III.2.2.1 The General State-Space Framework

At its heart, a photovoltaic system is a dynamic system. Its behavior is governed by an unobservable internal state, x(k), which represents all the internal physical processes at a given moment. This internal state evolves and, along with the current external inputs u(k), produces the final measurable output, y(k).

As we established, this entire process can be described by a pair of general discrete-time equations:

The State Equation, which describes how the internal state changes:

$$x(k+1) = f\left(x(k), u(k)\right) \tag{III.1}$$

- The Observation Equation, which describes how the output is generated from the state:

$$y(k) = g\left(x(k), u(k)\right) \tag{III.2}$$

The goal of "black-box" system identification is to find a single model that can learn to

approximate both of these unknown nonlinear functions, $\boldsymbol{f}$ and $\boldsymbol{g}$, using only the input-output data we can observe.

III.2.2.2 Our Contribution: A Unified and Implicit Learning Approach

This is where our approach diverges from the classic NSSM architecture. While the traditional NSSM uses a decomposed structure with two separate sub-networks, our contribution is to use a more flexible and unified approach to capture the complex dynamics of a PV system.

Because the behavior of a PV system is highly nonlinear and depends heavily on changing environmental conditions (like irradiance and temperature) as well as its own health status, we frame the I-V curve modeling problem as a sequence-to-sequence task within a state-space context.

Here's how our "all-in-one" method works:

- We use a single Bidirectional LSTM (BiLSTM) network to learn the entire state-space model implicitly.
- The network is trained to predict the sequence of current values (I) based on an input sequence containing a voltage sweep (V) along with the contextual conditions (irradiance G, temperature T, and fault state D).
- During this process, the network's internal hidden state, h(k), learns to act as an estimator of the actual physical state, x(k).

In essence, by learning this sequence-to-sequence mapping, the BiLSTM's internal dynamics are forced to approximate the state transition function $\boldsymbol{f}$, while its output layers learn to approximate the observation function $\boldsymbol{g}$. The network is free to choose how it represents the state because it's not stuck with a preset two-part design. One of the main goals of our study is to verify that this learned state fits with physical reality.

III.2.3 The BiLSTM Architecture for State-Space Modeling

To learn the complex, nonlinear state-space functions f and g directly from our sequential data, we selected a stacked Bidirectional Long Short-Term Memory (BiLSTM) network as the core of our identification model.

III.2.3.1 Rationale for the Architectural Choice

This choice was motivated by several key factors that we established in Chapter II.

- ✓ First, the LSTM cell is specifically designed to overcome the vanishing gradient problem, making it capable of capturing long-term dependencies within the I-V curve sequence.
- ✓ Second, the bidirectional nature of the BiLSTM is a critical advantage for our task. This is what gives the BiLSTM its real power: it gets to see the whole picture. By processing the sequence from both the start and the end simultaneously, the model at any point k has a much richer understanding of its context. It knows what came before and what's coming next. This ability to see the full story is especially crucial for modeling the complex shapes caused by faults like partial shading.

III.2.3.2 Proposed Model Implementation

The specific architecture we implemented is a stacked BiLSTM network, designed for the sequence-to-sequence regression task. As visualized in Figure III.3, the model is composed of several layers, each playing a specific role in transforming the input sequence into the final predicted current sequence.

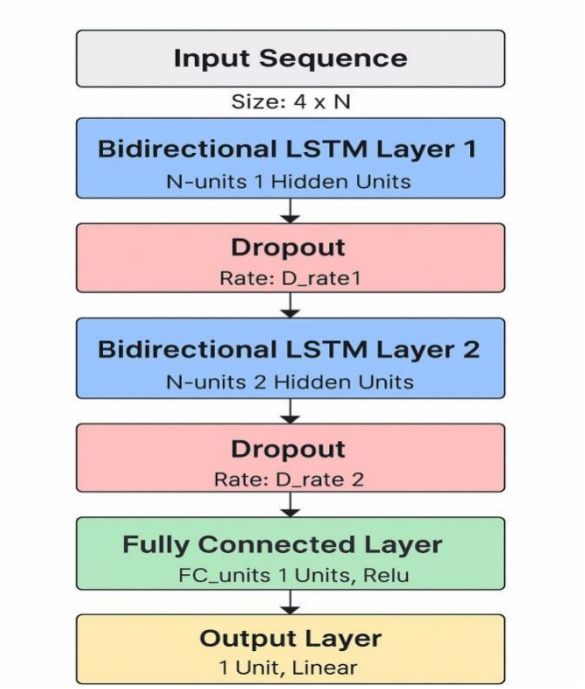

Figure III.3. The proposed stacked BiLSTM network architecture for I-V curve modeling.

The key components and hyperparameters of our final, optimized model are detailed in Table III.1.

Table III.1: Detailed description of the layers and key hyperparameters of the proposed BiLSTM model.

Layer type	Key Hyper-parameters	Purpose
Sequence Input Layer	Input size	Accepts the 4 x N input sequences.
Bilstm Layer	N_units_1 hidden units	Learns low-level temporal dependencies.
Dropout Layer	Rate: D_rate_1	Regularization to prevent overfitting
Bilstm Layer	N_units_2 hidden units .	Learns high-level temporal features
Dropout Layer	Rate: D_rate_2	Further regularization
Fully Connected Layer	FC_units units	Performs a final nonlinear transformation
Relu Layer		Introduces nonlinearity
Fully Connected Layer	unit, linear activation	Maps features to a single scalar output ($\hat{I}_k$) for each time step
Regression Layer		Computes the Mean Squared Error loss for training

- **Input Layer:** This layer accepts the input sequences, which have a dimension of 4 x N (four features for N time steps).

- **Stacked BiLSTM Layers:** At the heart of our model are two stacked BiLSTM layers, which work together to build up a deep understanding of the data. The first layer does the initial heavy lifting, looking at the raw input sequence to learn the basic, low-level patterns over time. It then passes its findings on to the second BiLSTM layer. The second layer uses this pre-digested information to look for bigger-picture, more abstract trends in the data. This stacking approach lets the network build a hierarchical view, where each layer builds on the insights of the one before it.

- **Dropout Layers:** To prevent overfitting, a dropout layer is applied after each BiLSTM layer. Dropout is a simple but powerful trick: at each step of the training, it randomly

ignores a fraction of the neurons. The effect is that the network can't put all its eggs in one basket, relying on just a few highly specialized neurons. By forcing the network to find multiple ways to represent the information, it learns features that are more robust and that generalize better to unseen examples.

- **Fully Connected and Output Layers:** Finally, the rich feature representation from the final BiLSTM layer enters its final processing stage. It first goes through a dense, fully connected layer that uses a ReLU activation function to perform one last nonlinear transformation. This is then mapped to a single scalar output (the predicted current $\hat{\mathbf{I}}_{\mathbf{k}}$) by a linear output layer.

- **Regression Layer:** The network's final layer is a regression layer. Its job is to measure how far off the model's predictions are from the actual current values by calculating the Mean Squared Error (MSE) loss between the predicted and actual current sequences, which is the objective function the model aims to minimize during training.

The specific values for the structural hyperparameters—such as the number of hidden units in the BiLSTM layers and the dropout rate—were not chosen arbitrarily. They were determined through the systematic Bayesian optimization process, which will be described in the next section.

III.2.4 Training and Optimization Protocol

To get a truly high-performing model, the architecture itself is only half the battle. The other half is finding the perfect set of 'tuning knobs'—the hyperparameters—that make it work best for our specific problem. Many researchers do this by hand or with inefficient grid searches, but we wanted a more rigorous process. That's why we built our training protocol around Bayesian optimization.

III.2.4.1 Hyperparameter Tuning with Bayesian Optimization

Bayesian optimization is a highly sample-efficient technique for finding the optimal values for parameters of "black-box" functions—in our case, the training process of our entire neural network **[101-103].** The way it works is by making intelligent guesses. Instead of trying options at random, it builds a 'map' of how the different hyperparameter settings

are likely to affect the model's performance. It then uses this map to intelligently pick the most promising new set of hyperparameters to try next.

For our study, we defined a search space for four critical hyperparameters:

- The number of hidden units in the BiLSTM layers.
- The dropout rate for regularization.
- The number of neurons in the fully connected layer**.**
- The initial learning rate for the optimizer.

The optimizer's goal was to find the combination of these parameters that minimized the validation loss. This systematic search allowed us to discover a high-performing and robust model configuration in a very efficient manner.

III.2.4.2 Final Model Training

Once the optimal hyperparameters were identified through Bayesian optimization, the final model was trained using this best configuration. For the actual training, we chose the Adam optimizer, a robust adaptive learning rate method that we discussed in Chapter II.

To keep the model from simply memorizing the training data, we also implemented an early stopping rule. This technique acts like a 'safety switch' during training. It constantly keeps an eye on the model's performance on a separate validation set. If the error on that validation data stops getting better for a certain number of epochs (a parameter known as 'patience'), the training process is halted automatically. This ensures that we save the model at the point where it generalizes best to new, unseen data, rather than continuing to train until it has simply memorized the training set.

This combination of a systematic hyperparameter search and a carefully regularized final training process is what allowed us to produce the robust and high-fidelity "digital twin" whose performance we will analyze in the next section.

III.2.5 Evaluation Metrics

Once the BiLSTM model was trained, the next step was to judge its performance objectively. To do this, we chose a set of evaluation metrics designed to answer two key questions about its predictions on the test data:

- On average, what is the magnitude of the prediction errors?
- Beyond just the average error, how well does the model reproduce the overall trends in the real data?

To answer these questions, we used two standard and complementary metrics for regression tasks.

III.2.5.1 Root Mean Square Error (RMSE)

To find out how large the prediction errors are on average, we used the **Root Mean Square Error (RMSE)**. This metric first calculates the square of the difference between each predicted current value ($\mathbf{I_{pred}}$) and the actual current value ($\mathbf{I_{real}}$). It then averages these squared errors over all the points in the test set and takes the square root to bring the final value back to the original unit (Amperes).

$$RMSE = sqrt\left((1/P) * \sum_{i=1}^{P} \sum \left(I_{real}(i) - I_{pred}(i) \right)^2 \right) \quad \text{(III.3)}$$

Where **P** is the total number of data points across all test sequences. A lower RMSE value indicates a better fit, meaning the model's predictions are, on average, very close to the true values.

III.2.5.2 Coefficient of Determination (R^2)

While RMSE tells us about the average error, it doesn't tell us how well the model explains the *variability* in the data. For that, we used the **Coefficient of Determination**, commonly known as **R-squared (R^2)**.

The R^2 value answers the question: "What percentage of the variation in the real data does our model successfully explain?". It compares the errors of our model to the errors of a very simple baseline model that just predicts the average value of the data **($\mathbf{I_{mean}}$).**

$$R^2 = 1 - \frac{\sum\left(I_{real} - I_{pred}\right)^2}{\sum\left(I_{real} - I_{mean}\right)^2} \quad \text{(III.4)}$$

The R^2 score ranges from $-\infty$ to 1.

- An **R^2 of 1** means a perfect fit; the model explains 100% of the variability.
- An **R^2 of 0** means the model is no better than simply guessing the average.
- A **negative R^2** means the model is actively worse than the simple average.

Together, a low RMSE and a high R^2 (close to 1) provide strong evidence that a model is not only accurate on a point-by-point basis but also captures the overall dynamic behavior of the system.

III.2.5.3 Physical Parameter Validation

Beyond these overall error metrics, we decided to go a step further to check if our model made physical sense. To do this, we also evaluated its ability to predict key physical parameters that are not directly trained but are derived from the I-V curves. We extracted parameters like Maximum Power (P_{max}), Voltage at Maximum Power (V_{mpp}), Short-Circuit Current (I_{sc}), and Open-Circuit Voltage (V_{oc}) from both the predicted and the actual I-V curves and compared them. This provides a crucial test of whether our data-driven model has learned a physically coherent representation of the PV system.

III.3 Results and Discussion

In this section, we present the results of our state-space identification carried out using the proposed BiLSTM model. Our analysis begins by showing the complexity of the identification challenge itself. A look at the I-V curves in Figure III.2, which shows a sample from our test set, immediately reveals how dramatically the system's behavior changes across its different operational and fault conditions.

The system's state has a profound impact on the shape of the I-V curve. For example, under healthy conditions, the curve is typically smooth and concave. A fault like partial shading, however, can drastically alter this shape, introducing significant distortions and

often creating multiple local power maxima. The true test for any system identification approach, therefore, is its ability to accurately model this full range of complex, nonlinear behaviors, rather than just fitting a single curve shape, a point emphasized in literature focused on high-fidelity dynamic models **[17].**

Following this, we will evaluate the model's predictive ability using global metrics, and then finish with a detailed study of its physical meaning and the internal dynamics it has learned.

III.3.1 Optimal Model Configuration and Training Convergence

Table III.2 shows the final hyperparameters for our BiLSTM model, chosen using the Bayesian optimization process described earlier. This careful search was important to find a good balance between model size and regularization, which led to the reliable results reported in the next section.

Table III.2: Final Hyperparameters of the Proposed BiLSTM Model

Parameter	Value	Description
Input Features	4	V, G, T, Fault State
BiLSTM Hidden Units	140 per layer	Number of hidden units in each of the two BiLSTM layers
BiLSTM Layers	2 (stacked)	Number of stacked BiLSTM layers.
Fully Connected Units	58	Number of units in the dense layer before the output
Dropout Rate	0.2225	Dropout rate applied after each BiLSTM layer
Optimizer	Adam	Adaptive moment estimation optimizer
Initial Learning Rate	0.0026	Initial learning rate for the Adam optimizer
Mini-batch Size	32	Number of samples per training iteration
Max Epochs	100	Maximum number of training epochs
Early Stopping Patience	10	Number of epochs to wait for improvement before stopping.

The training progress for the final model with these optimal settings is shown in Figure III.4. The plot tracks both the Root Mean Square Error (RMSE) and the overall loss function for the training and validation datasets over the course of the epochs.

As the figure shows, the training was successful and stable. In the initial epochs, both the training and validation errors dropped sharply as the model quickly learned the fundamental patterns in the data. The curves then began to stabilize as the model fine-tuned its parameters.

Crucially, the training was halted automatically after 23 epochs by the early stopping rule. This happened because the error on the validation set stopped improving, indicating that the model had reached its optimal point of generalization. This measure reduced the risk of overfitting during training, ensuring that its performance on new, unseen data would be reliable. The final validation RMSE at the point of convergence was **0.0475 A**, confirming a successful and efficient training process.

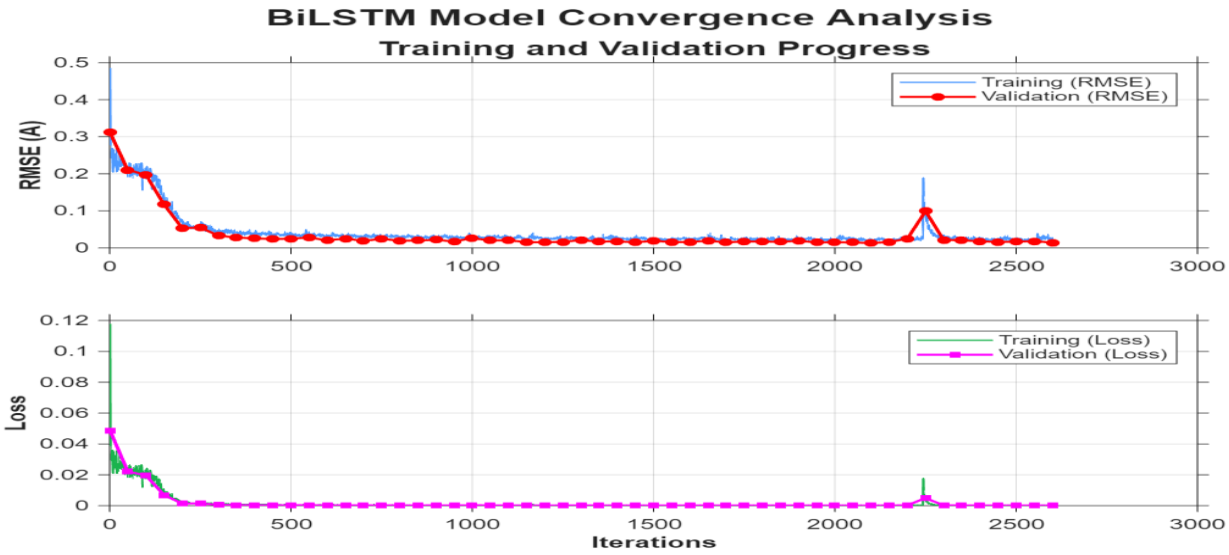

Figure III.4 : Training and validation Progress of Final BiLSTM Network

III.3.2 Overall Identification Performance

After training our model and seeing that it converged well, the next big question was how it would perform on brand-new data it had never seen before. To find out, we tested it on a separate set of 1630 I-V curves that were kept aside for this final evaluation.

Before evaluating the results, we converted the model's predictions into their original unit (Amperes). The model achieved very strong performance, indicating that it was able to generalize well. Below are the main test set metrics:

- **Root Mean Square Error (RMSE): 0.0475 A**
- **Coefficient of Determination (R^2): 0.9974**

These numbers tell a very clear story. The extremely low RMSE shows that, on average, the model's predictions are very close to the real current values. At the same time, the R^2 score of **0.9974** is nearly a perfect 1.0, which means our model was able to explain **99.74%** of the variability in the real-world data.

To get a more visual sense of this high level of accuracy, the correlation plot in Figure III.5 compares the predicted current values against the actual ones.

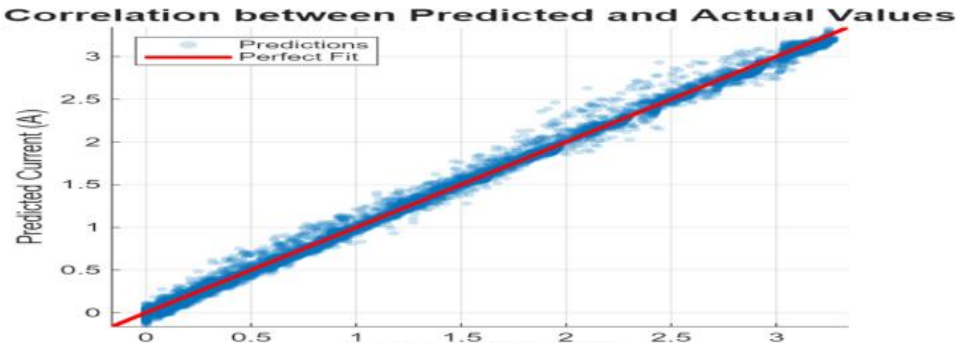

Figure III.5: Correlation between Predicted and Actual Current Values on the Test Set

As the figure shows, the data points form a very dense and narrow cluster right around the(y = x) line, which represents a perfect prediction. This tight alignment is a strong visual confirmation of the model's robustness and its ability to make highly accurate predictions, perfectly in line with the high R^2 score **[104]**.

III.3.3 Analysis of Prediction Errors

While the overall performance numbers are excellent, a deeper look at the prediction errors can give us more insight into the model's behavior. The error distribution histogram, shown in Figure III.6, does exactly that.

This chart tells a clear story about the reliability of our model. We can see that the distribution of errors is sharply peaked and centered almost perfectly at zero. This is a strong indication of two positive qualities:

1. **The model is unbiased:** It doesn't systematically overestimate or underestimate the current values. Its errors are balanced around zero.

2. **The errors are small:** The vast majority of the prediction errors fall within a very narrow range of ±0.1 Amperes.

This tight, zero-centered spread reinforces what the low RMSE already suggested: our model is not only accurate on average, but it is also consistently reliable across the entire test set.

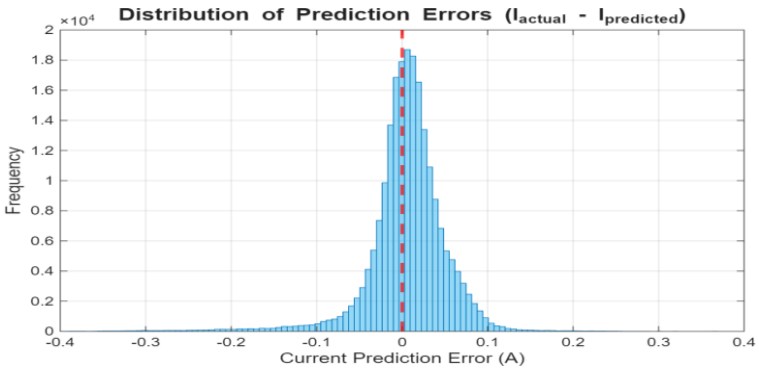

Figure III.6 . Distribution of Prediction Errors (I_{actual} - $I_{predicted}$) on the Test Set

III.3.4 Qualitative Performance: Visualizing the Predicted I-V Curves

The numbers confirm that the model is very accurate, but looking at its behavior on specific I-V curves makes the proof even clearer. In Figure III.7, we have overlaid the model's predicted I-V curves directly on top of the actual, measured curves for several randomly selected samples from the test set.

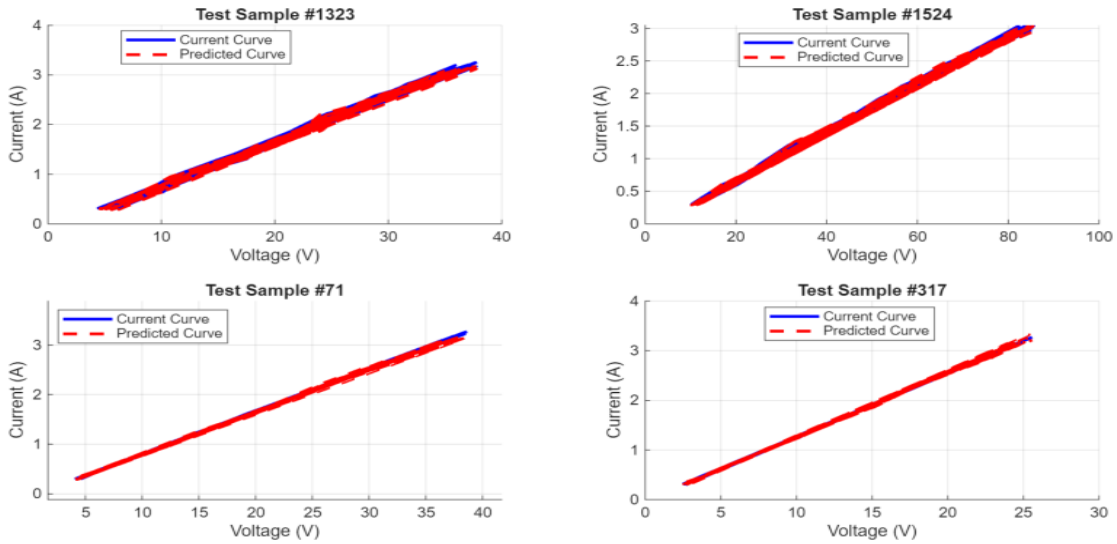

Figure III.7 . Comparison of actual and predicted I-V curves for randomly selected test samples.

This visual comparison confirms the model's excellent performance. We can see that for each sample, the two curves—predicted and actual—track each other almost perfectly. This holds true even as the curves cover very different voltage and current ranges, which is a strong visual testament to the model's robustness and generalization ability.

The tight agreement between the curves provides a qualitative validation of the low RMSE and high R^2 scores we reported earlier, showing that the model has successfully learned to capture the precise shape of the I-V characteristic under a variety of conditions.

III.3.5 Detailed Analysis and Validation of the Identified State-Space Model

So far, we have confirmed that our BiLSTM model can reproduce I-V curves with a high degree of statistical accuracy. But a central claim of our state-space approach is that the model should do more than just curve-fitting; it should learn a physically meaningful representation of the system. In this section, we conduct a series of detailed analyses to test this claim.

III.3.5.1 Dynamic Identification Performance

First, to visually assess how the model behaves on a dynamic trajectory, we selected a random sample from the test set and tracked its performance over the entire voltage sweep sequence. Figure III.8 illustrates this process.

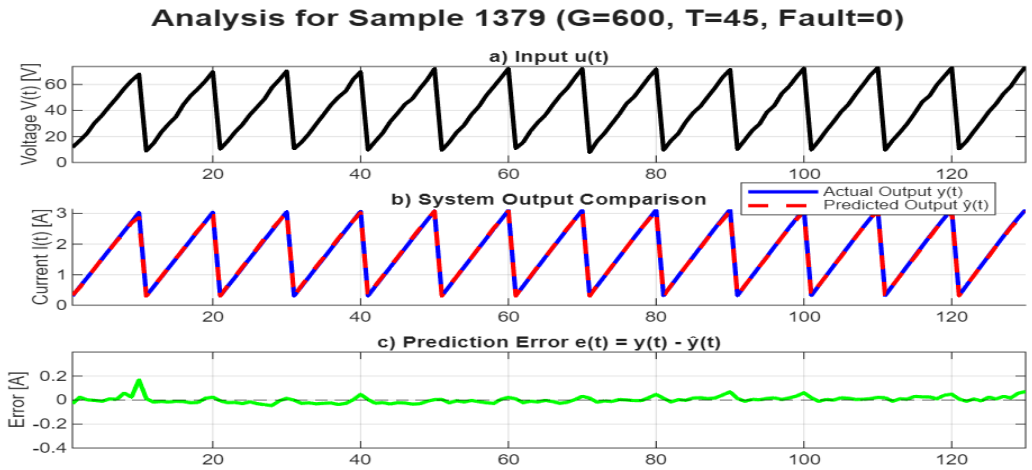

Figure III.8. Dynamic identification performance of the state-space model for a random test sample. Subplots show:
(**a**) input voltage u(t); (**b**) comparison between actual y(t) and predicted ŷ(t) current; (**c**) prediction error e(t).

The figure's subplots show that when the model is given a standard voltage sweep as input (a), its predicted output ŷ(t) almost perfectly tracks the actual system output y(t) (b). The strong tracking accuracy is confirmed by the prediction error e(t) (c), which stays small, stable, and close to zero over the entire sequence. This shows that the model provides consistent and unbiased predictions across the full dynamic range of the I-V curve.

III.3.5.2 Analysis of the Learned Internal State

The core idea of our state-space approach is that the hidden states of the BiLSTM should act as an estimator for the unobservable physical state of the system. To verify this, we first examined how these hidden states evolved for the same test sample. As shown in Figure III.9, the evolution of the first three components of the learned hidden state vector h(t) is not random.

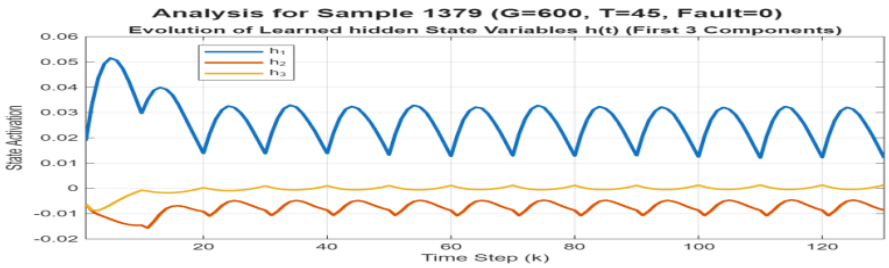

Figure III.9. Evolution of the first three components (h1, h2, h3) of the learned hidden state vector h(t) for the test sample

Instead, the hidden states follow a smooth and highly structured pattern that is clearly driven by the input voltage sweep. This tells us that the network has successfully learned a stable and coherent set of internal rules, a state-transition function h(k) = f(h(k-1), u(k)), which is precisely what a good state-space model should do.

To further validate that this learned state is physically meaningful, we visualized the final hidden state h(T) for all test samples using the **t-SNE technique**. The results, presented in Figure III.10, provide a direct look into the learned latent space.

The t-SNE visualizations confirm that the model has learned to organize its internal state representation based on a clear physical hierarchy. The primary organizing factor is **irradiance (a)**, followed by a secondary organization based on **temperature (b)** within each irradiance cluster, and a final, local organization based on **fault categories (c)**.

This automatically discovered structure is robust evidence that the final hidden state is a physically meaningful embedding, not just a random vector. It proves that our model has gone beyond simple curve-fitting to construct a true, interpretable internal state-space model of the PV system.

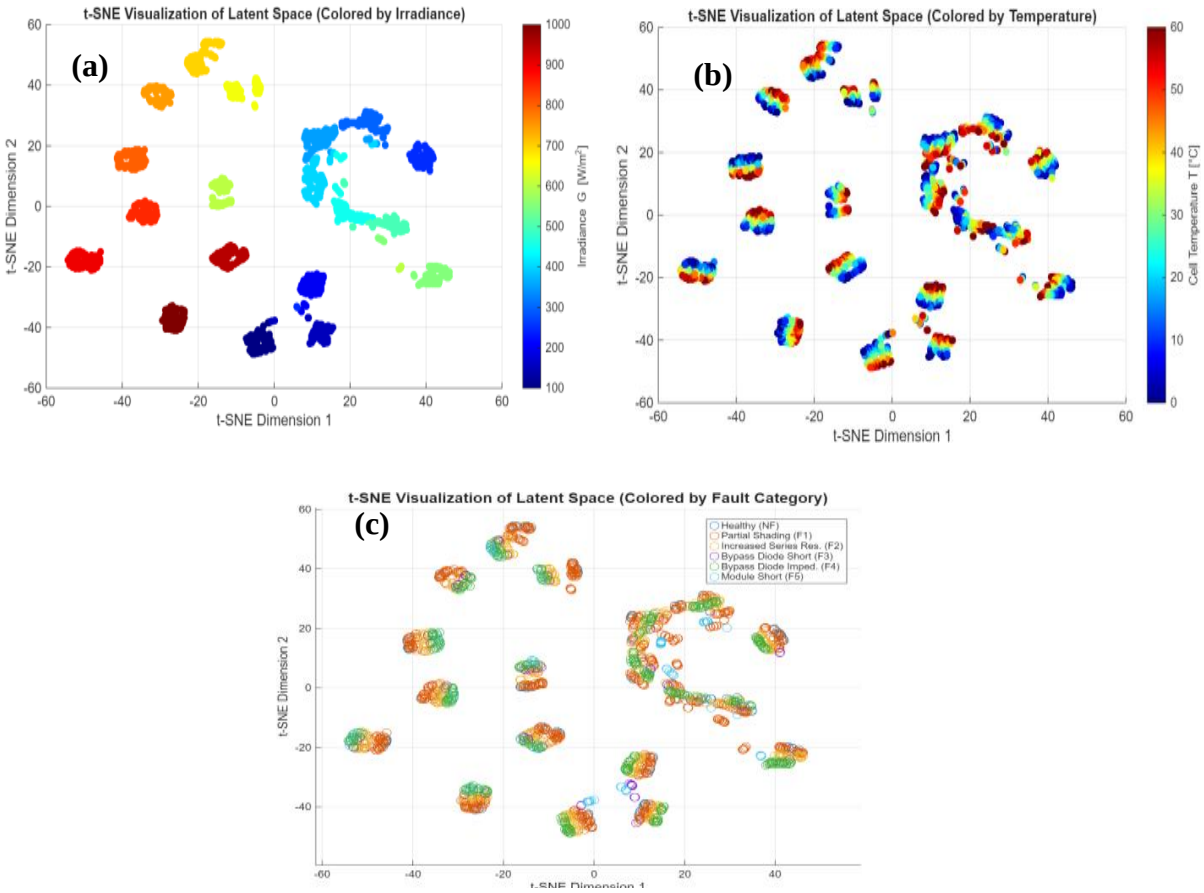

Figure III.10: t-SNE visualization of the learned latent space, colored by (a) Irradiance, (b) Temperature, and (c) Fault Category.

III.3.5 Physical Parameter Extraction

The last and most crucial test of physical meaning is to see if the model's predictions allow us to recover key electrical parameters.. We used the predicted I-V curves for all 1630 test samples to calculate five standard PV parameters: P_{max}, V_{mpp}, I_{mpp}, V_{oc}, and I_{sc}.

Table III.3 summarizes the results, comparing the mean of the actual and predicted values. The model predicts the mean maximum power (P_{max}), voltage (V_{mpp}), current (I_{mpp}), and open-circuit voltage (V_{oc}) with relative errors of less than 1%. This high level of accuracy

on derived physical quantities is strong evidence that the learned state-space representation is physically accurate.

Table III.3: Performance on Key Physical Parameter Extraction.

Parameter	Mean Actual	Mean Predicted	Mean Absolute Error	Relative Error (%)
P_{max} (W)	146.96	147.1	0.13498	0.091849
V_{mpp} (V)	88.54	88.032	0.50806	0.57382
I_{mpp} (A)	2.1939	2.1908	0.003102	0.14141
V_{oc} (V)	81.917	81.828	0.088648	0.10822
I_{sc} (A)	0.2626	0.30619	0.04357	16.59

A higher relative error (16.6%) was observed for the short-circuit current (Isc). However, as the error distribution analysis in Figure III.11 reveals, this is a statistical artifact, not a model weakness. The error distribution for Isc has a tall, narrow peak at zero, showing that the model has no systematic bias and that its absolute errors are almost always tiny. The large percentage in the table is simply a result of the math, as the relative error formula exaggerates even minor absolute errors when the actual I_{sc} value is very low (~0.26 A).

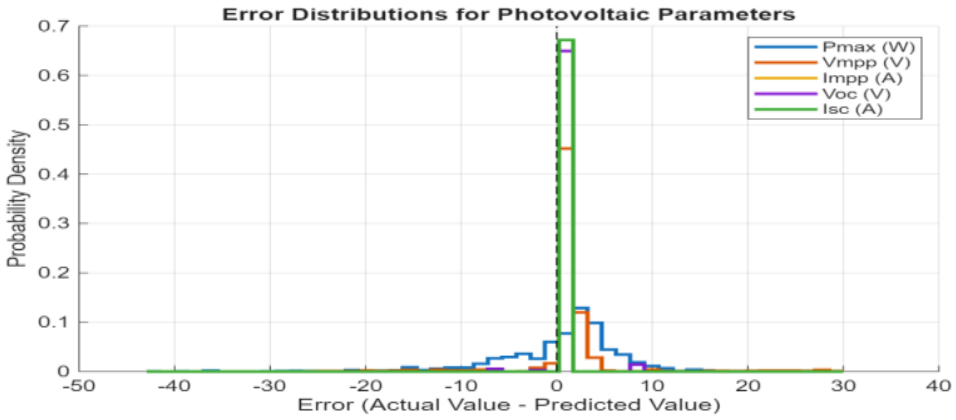

Figure III.11. Probability density functions of the prediction errors for each of the five extracted physical parameters.

This comprehensive analysis confirms that our identified model is not just internally consistent but also physically meaningful, making it a true "digital twin" of the PV system.

III.3.6 Validation Against Experimental Physical Trends

To confirm that our model is physically meaningful, we compared its results with experimental data from other studies. Although it was trained with simulations, the model clearly picked up the real physical trends that shape PV system behavior.

For example, the experimental work of Belaout et al. **[98]** presents the dynamic response of a real PV system's Maximum Power Point (MPP) to real-time changes in irradiance and temperature. By analyzing our model's predictions under similar changing conditions, we can check for qualitative agreement.

III.3.6.1 Comparison of Physical Trends

According to **[98]** and related studies, PV systems follow two basic physical rules. The first is the direct link between irradiance and current: when irradiance goes up, the maximum power point current (I_{mpp}) grows proportionally, and as a result the maximum power (P_{max}) increases too.

The second is the inverse relationship between temperature and voltage: as the cell temperature increases, the maximum power point voltage (V_{mpp}) falls. This effect slightly reduces the overall maximum power.

III .3.6.2 What Our Model Has Learned

Our analysis of the physical parameters extracted from the model's predictions (as shown in Table III.3 and the t-SNE plots in Figure III.10) confirms that our model has successfully learned these exact relationships.

- The t-SNE plot (Figure III.10a) clearly shows that irradiance is the primary factor organizing the latent space, directly reflecting its dominant impact on the current and power output.
- The precise predictions of V_{mpp}, I_{mpp}, and P_{max} under different temperatures and irradiance levels (Table III.3) show that the model has indeed captured these basic dependencies.

This comparison shows that, even though the BiLSTM was trained only as a "black-box" on static I-V curves, it still learned the key physical principles behind a PV system's dynamics. The agreement with experimental results confirms the model's physical consistency.

III.3.7 Comparison with a Static Input-Output Model

To fully appreciate the practical advantages of our state-space approach, we needed a point of comparison. For this, we built and trained a standard static input-output model, a Multi-Layer Perceptron (MLP), on the exact same dataset.

III.3.7.1 The Comparative Model

The MLP's role was to model the direct input-output relationship at each time step. It was trained to treat each (V_k, I_k) point as an independent sample, using the same input features (V_k, G, T, Fault State) to predict the current I_k.

To ensure a fair comparison, the MLP was designed with a comparable level of complexity (number of parameters) and was trained under similar conditions as our BiLSTM model. The structure and training settings of the MLP used for comparison are given in Table III.4.

Table III.4: Configuration of the Comparative MLP Model.

Parameter	Value
Input Features	4
Number of Hidden Layers	2
Neurons per Hidden Layer	128
Activation Function	ReLU
Output Layer	1 Neuron (Linear)
Optimizer	Adam
Initial Learning Rate	0.001
Mini-batch Size	32

III.3.7.2 Quantitative Performance Comparison

The quantitative performance of the two models on the test set is summarized in Table III.5.

Table III.5: Performance Comparison between the State-Space (BiLSTM) and Input-Output (MLP) Models.

Model Type	RMSE (A)	R-squared (R^2)
State-Space (BiLSTM)	0.0475	0.9974
Input-Output (MLP)	0.0815	0.9924

The data in the table clearly highlights the advantage of the state-space model. The BiLSTM achieves a significantly lower RMSE and a higher R^2 score. This model works better because it looks at the whole I-V curve as a sequence. The MLP treats every point separately, but the BiLSTM remembers what came before and what comes after. Thanks to this memory, it can give more accurate results.

III.3.7.3 Qualitative Performance Comparison

This difference in performance is even more striking when we look at representative test samples, as shown in Figure III.12.

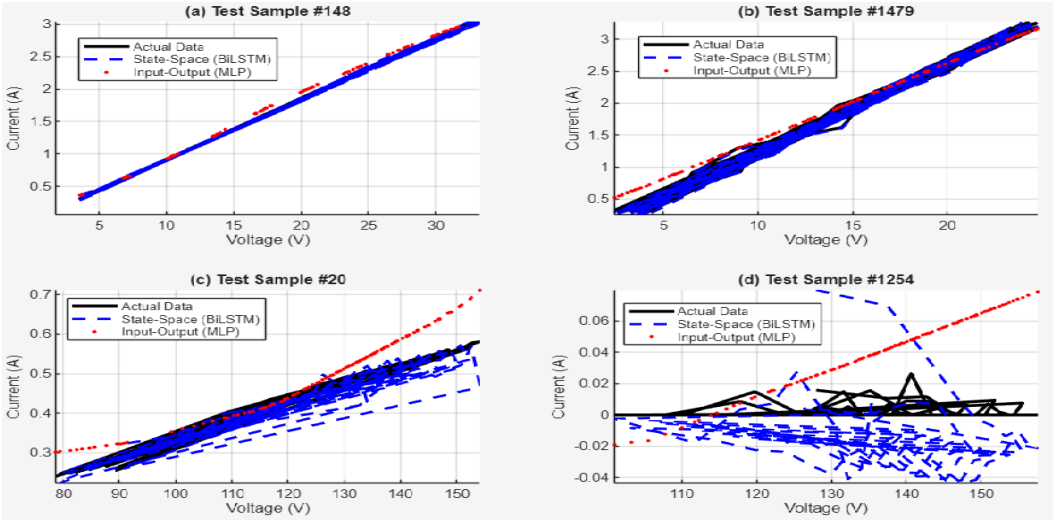

Figure III.12. Qualitative comparison of the BiLSTM and MLP models on four representative test samples.

The plots reveal the qualitative superiority of the sequential approach:

- On a clean, simple I-V curve (a), the BiLSTM's prediction is nearly perfect, while the MLP already shows slight deviations.

- On a complex, noisy curve (b), the BiLSTM successfully captures the subtle nonlinearities, whereas the MLP's prediction is overly linear and less accurate.

- On a highly distorted curve under a severe fault (c), the MLP's predictions diverge completely, failing to capture the shape of the data. The BiLSTM, though challenged, still manages to follow the overall trend.

- On a low-current, erratic curve (d), both models struggle, but the MLP produces a simple, incorrect linear trend, while the BiLSTM still attempts to model the complex, non-zero behavior of the data.

This comparison confirms that our state-space model (BiLSTM) is not only quantitatively more accurate but also qualitatively more robust. The main distinction is that the BiLSTM makes use of the sequence information, something that static input-output models are not built to handle. This shows the clear benefit of using a state-space approach when dealing with complex dynamic systems **[2, 92].**

III.4 Conclusion

In this chapter, we tackled our first major goal: to build a complete state-space model for the complex, nonlinear behavior of a photovoltaic (PV) system. To do this, we treated the I-V curve identification as a sequence-to-sequence problem. This allowed us to successfully train a stacked Bidirectional LSTM (BiLSTM) network that can act as a high-fidelity 'digital twin' of a PV array.

The results clearly show that this method is effective.. The proposed model achieved remarkable predictive accuracy, with a Coefficient of Determination (R^2) exceeding 0.997 and an RMSE value that is very low More importantly, we went beyond simple error metrics to validate the physical relevance of the learned model. Our analysis confirmed that the model's internal hidden states learn a structured, interpretable representation of the system's dynamics. The key electrical parameters (like P_{max}, V_{oc}, and V_{mpp}) can be accurately extracted from its predictions.

When it comes to modeling the complex dynamics of nonlinear faults, our results show a clear winner. The data-driven, state-space approach was able to produce a physically sound model that performed much better than the traditional static methods we tested it against. The fundamental ability of our model to capture the system's internal dynamics is what allows it to make both highly accurate I-V curve predictions and precise calculations of derived electrical parameters—a capability that static models inherently lack.

Functioning as a "virtual test bench," this high-fidelity surrogate model could be used to generate realistic I-V data for the development and validation of next-generation control algorithms, such as adaptive MPPT techniques.This aligns with a rich body of research that uses such state-space models as a foundation for robust control design and predictive control strategies **[105,106].**

This successful implementation highlights the key lesson of the End-to-End approach: This strategy, where the BiLSTM acts as its own feature extractor, is supremely effective for modeling and regression tasks when dealing with high-quality, information-rich data—such as the clean, high-resolution curves from our emulator. Under these perfect conditions, the stacked BiLSTM had enough power to learn the best internal representation straight from the raw sequence itself. This made using expert feature engineering unnecessary for reproducing the signal with high fidelity.

So, we've shown that our BiLSTM-based approach is great at modeling the system's behavior. This naturally leads to the next big question: can we use a similar method to actually make decisions about the system's health? In the next chapter, we will tackle this challenge by developing a robust framework for the detection and classification of faults, building upon the powerful tools we have established here.

CHAPTER IV

Fault Classification in a Nonlinear PV System: A DWT-BiLSTM Approach

IV.1 Introduction

While the previous chapter focused on modeling the continuous behavior of a photovoltaic system, this chapter tackles a different but complementary challenge: *classifying* its discrete state of health. Our goal is no longer to predict the exact shape of the I-V curve, but to make a decisive judgment: is the system healthy, or is it operating under a fault condition? This shift from modeling to classification is a critical step towards building truly intelligent monitoring systems, which is essential for ensuring the safety, reliability, and optimal performance of PV installations **[50, 51].**

The field of AI-based fault diagnosis for PV systems is evolving at a rapid pace. For years, the research landscape was dominated by classic machine learning algorithms like Support Vector Machines or Random Forests applied to a handful of electrical parameters **[107,108].** The significant shift came with deep learning. Researchers began using powerful Convolutional Neural Networks (CNNs) to analyze thermal images from drones, successfully spotting physical defects like hot spots and micro-cracks **[99,109].** For time-series data, Long Short-Term Memory (LSTM) networks became the tool of choice, proving effective at detecting everything from high-impedance faults in the grid **[110]** to the subtle signs of internal module degradation **[111].**

Even more recently, the trend has moved towards hybrid and multi-modal approaches. Looking at the current research, two key trends stand out. The whole field is now moving towards more sophisticated, multi-part models. For example, a lot of new research focuses on building hybrid networks by linking CNNs and LSTMs together, which gives a single model the ability to understand both spatial and temporal data at the same time **[108].** A second, parallel trend is the integration of traditional signal processing. To make their models tougher, researchers are now building techniques like the Wavelet Transform right into their deep learning pipelines **[112].** However, despite these impressive advances, a key challenge remains: building a model that is robust enough to distinguish between fault signatures that are often subtle, non-stationary, and masked by the noise of real-world operating conditions.

This is where our work makes its second primary contribution. While many studies focus on a single type of data or a single "end-to-end" model, we hypothesize that a truly robust hybrid framework will be more effective. Our methodology is built on a complete,

end-to-end process. First, we use the Discrete Wavelet Transform (DWT) to perform a multi-resolution analysis of the I-V curves, which allows us to extract a rich set of features that capture the subtle signatures of different faults. Next, to ensure our model focuses only on the most critical information, we use Neighborhood Component Analysis (NCA) to intelligently select the most discriminative features. Finally, these selected features are fed into a Bidirectional LSTM (BiLSTM) network. As we established, the BiLSTM is ideal for this task as it analyzes the feature sequence in both directions, and its architecture was rigorously fine-tuned using Bayesian optimization to maximize its robustness.

So, in the rest of this chapter, we're going to walk through how we built and tested this hybrid framework. Section 4.2 will describe our methodology in detail, from feature extraction to model optimization. In Section 4.3, we'll present the classification results, where we will take a close look at how well our model performed and see how it stacks up against some of the other leading methods out there. Finally, Section 4.4 will conclude the chapter, summarizing our findings and setting the stage for the final synthesis of our work.

IV.2 The Hybrid DWT-BILSTM Methodology

This section explains our hybrid fault diagnosis method in a straightforward way. The process starts by analyzing the raw I–V curves at different resolutions and then applies a BiLSTM network to carry out the final optimized classification.

It is important to highlight that this study relies on the same comprehensive dataset presented earlier in Section III.2.1. Using the same data ensures a consistent and straightforward comparison between the modeling and classification tasks.

IV.2.1 Data Preprocessing and Multi-Resolution Feature Extraction

The first and most important step in our hybrid method is to turn the raw one-dimensional I–V curves into a well-structured set of useful features. This step, called feature engineering, gives the BiLSTM model a richer view of the system, which helps it detect even very small fault patterns.

IV.2.1.1 The Rationale for a Feature-Based Approach

Although it is possible to train models directly on raw data, this usually comes at the cost of very large datasets and a strong sensitivity to noise. In real applications, diagnosing complex faults works better with a hybrid approach that relies on feature extraction. By first extracting key features from the I–V curves, the model is guided more effectively to:

- Focus on relevant information, ignoring noise and irrelevant variations.
- Capture both global and local characteristics of the curve's shape.
- Create a richer input representation that leads to better classification performance.

IV.2.1.2 Building a Comprehensive, Multi-Domain Feature Set

From each of the I-V curves in our database, we extracted a comprehensive set of 90 indicators. We selected these indicators with the aim of describing the curve’s behavior through three distinct perspectives:

- Multi-Resolution DWT Features: To capture the fine, local variations and non-stationary behaviors that are hallmarks of fault conditions, we used the Discrete Wavelet Transform (DWT). For each I-V curve, we applied a 5-level DWT decomposition using the 'coif5' wavelet to both the current (I) and voltage (V) signals. At every stage of the decomposition, we measured a few simple indicators—such as energy, entropy, skewness, and variance. Together, these values act like a unique signature of the curve’s shape at different levels of detail. Thanks to this ‘fingerprint,’ the method can detect even small distortions, for example those caused by partial shading.

- Global RAW Statistical Features: To provide a global overview of the curve's properties, we calculated a set of raw descriptive statistics. These include standard metrics like max, min, RMS, skewness, kurtosis, variance, and median absolute deviation (MAD) for both the I and V signals, as well as slope information.

- Physical I-V Curve Parameters: Finally, to give the model high-level, physically meaningful information, we extracted a wide range of standard and additional PV parameters. These include not only the key points (Isc, Voc, Pmax, Vmpp, Impp) but also the fill factor, various slopes at different points of the curve, the resistance at MPP, the number of power peaks (crucial for shading), and other derived metrics.

The full set of these 90 extracted features is essential for robust fault detection, especially for identifying faults with low amplitude or intermittent behaviors [100]. The combination of DWT, RAW, and I-V parameter features provides a much more complete view of the system's state than any single feature type alone.

IV.2.1.3 Target Label Re-Grouping

Following the assembly of these 90 features for each sample, the 22 original fault labels were grouped into six target superclasses (F0-F5) to create a more balanced and manageable classification problem. This re-categorization process led to a final set of 2210 samples. The dataset was then stratified and normalized following the procedure outlined in the previous chapter, which prepared it for the next essential stage: feature selection.

IV.2.2 Feature Selection with Neighborhood Component Analysis (NCA)

IV.2.2.1 The "Curse of Dimensionality": Why Feature Selection is Necessary

Once the complete set of 90 features had been extracted, we encountered a well-known challenge in machine learning: the curse of dimensionality. Having a high-dimensional feature space—in other words, having too many input features—can cause several problems:

- It can slow down the model's training process significantly.
- It can introduce noise and irrelevant information that confuses the model and harms its performance.
- It increases the risk of overfitting, where the model learns to memorize noise in the training data instead of the actual underlying patterns.

To address this, the next critical step in our methodology was to select only the most useful and informative features from this initial set.

IV.2.2.2 Our Chosen Method: Neighborhood Component Analysis (NCA)

For this feature selection task, we chose Neighborhood Component Analysis (NCA), a powerful, supervised, and non-linear method introduced initially by Goldberger et al. **[113].**

The principle behind NCA is both simple and powerful. Instead of just looking at statistical correlations, NCA directly learns a weight or "importance score" for each individual feature. It does this by finding the feature weights that would maximize the classification accuracy of a simple k-nearest neighbors (k-NN) classifier on the training data **[114].** In other words, NCA automatically gives a high score to the features that are most helpful in grouping samples from the same fault class together, while assigning a low score to those that are noisy or irrelevant.

By applying this method and setting a threshold on the resulting weights, we were able to create a lean and robust set of information, highly focused on the fault classification task. This cleanup step is a critical part of our framework for building a BiLSTM model that is both accurate and computationally efficient.

IV.2.3 The BiLSTM Classifier and its Optimization

IV.2.3.1 The BiLSTM Classifier Architecture

Once NCA refined the feature set, the final step was to train a powerful classifier to learn the mapping from the feature sequences to the fault labels. For this task, we chose the Bidirectional LSTM (BiLSTM) network, whose architecture and theoretical advantages we detailed in Chapter II.

The BiLSTM is the ideal choice for this problem for two key reasons:

- It is designed for sequential data. Although our input is a set of features, these features (especially the DWT indicators) have a natural sequence, representing different scales of analysis. The BiLSTM can learn patterns and dependencies *across* these different scales.
- It captures bidirectional context. By processing the feature sequence in both forward and backward directions, the BiLSTM can make a more robust classification decision, as the signature of a fault can be defined by the relationship between all the features, not just in one order.

Our specific implementation consists of an input layer that accepts the sequence of selected features, a BiLSTM layer to learn the temporal/sequential patterns, followed by several fully connected layers that perform the final classification

IV.2.3.2 Ensuring Robustness through Bayesian Optimization

A good architecture still needs the correct hyperparameters to work well. Rather than using default values or adjusting them by hand, we applied Bayesian optimization to systematically find the best setup for a robust and accurate model.

As described in Chapter III, this technique intelligently explores the hyperparameter space to find the combination that minimizes the classification error on a validation set. For this classification task, we optimized four key hyperparameters:

- The number of hidden units in the BiLSTM layer.
- The dropout rate for regularization.
- The size of the fully connected layer.
- The initial learning rate.

By systematically optimizing these parameters, the model's quality comes from a well-structured design, not from luck. Details of the final architecture and training parameters are presented in the Results section.

IV.3 Results And Analysis

In this section, we share the results of our hybrid DWT-BiLSTM model. We begin by looking at the raw data to see the main difficulties, then show how feature selection makes the data clearer, and finally present how well the optimized classifier performs.

IV.3.1 Initial Feature Space Visualization

Before we could even begin to select features, we first needed to understand the data we were working with. Our feature extraction process gave us a rich, 90-dimensional dataset, but is this data well-structured for a classification task?

To get a first look at the richness of our initial dataset, we can visualize some of the key DWT features. Figure IV.1 shows the DWT energy indicators for a single, representative sample from each of the six fault classes. Even with this simple view, we can see that the different fault types create very different energy profiles across the wavelet decomposition levels. This confirms that our feature extraction is capturing unique and potentially discriminative information.

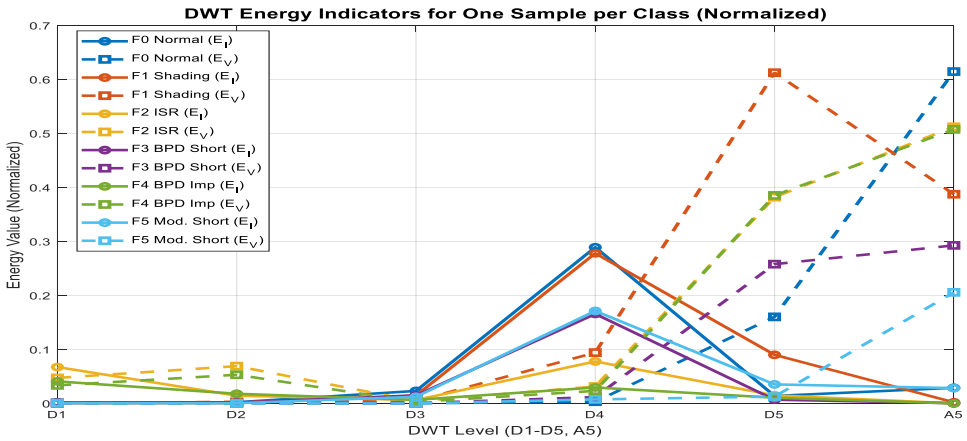

Figure IV.1: DWT Energy indicators for one sample from each class

To provide a complementary perspective, Figure IV.2 illustrates the DWT entropy indicators for a sample of each class.

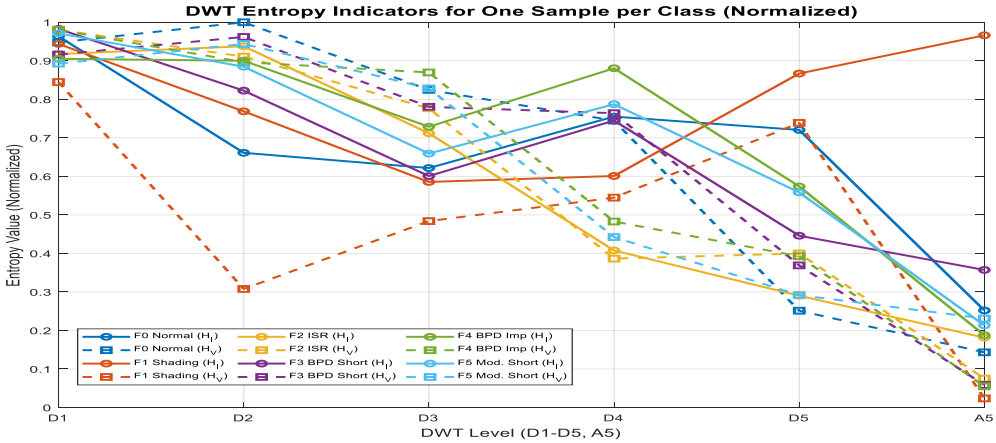

Figure IV.2: DWT Entropy Indicators for One Sample from each Class

These entropy profiles provide another layer of information about the fault signatures. While energy captures the magnitude of the signal's variations, entropy captures its complexity or disorder. The distinct shapes of these entropy curves for each fault type further confirm that our DWT-based features contain rich, discriminative information.

Finally, to get a broader picture of the entire feature set, Figure IV.3 illustrates the distribution of all three types of indicators (DWT, RAW, and I-V) for a representative sample of each class.

Here again, we see a notable variability between the different fault classes. The diversity in these distributions—from energy, entropy, and other statistical measures—shows that our initial feature set is rich and contains the necessary information to help the BiLSTM model

effectively distinguish the normal state from the various faulty states. However, it also highlights the complexity and high dimensionality of the problem we need to solve before moving on to classification.

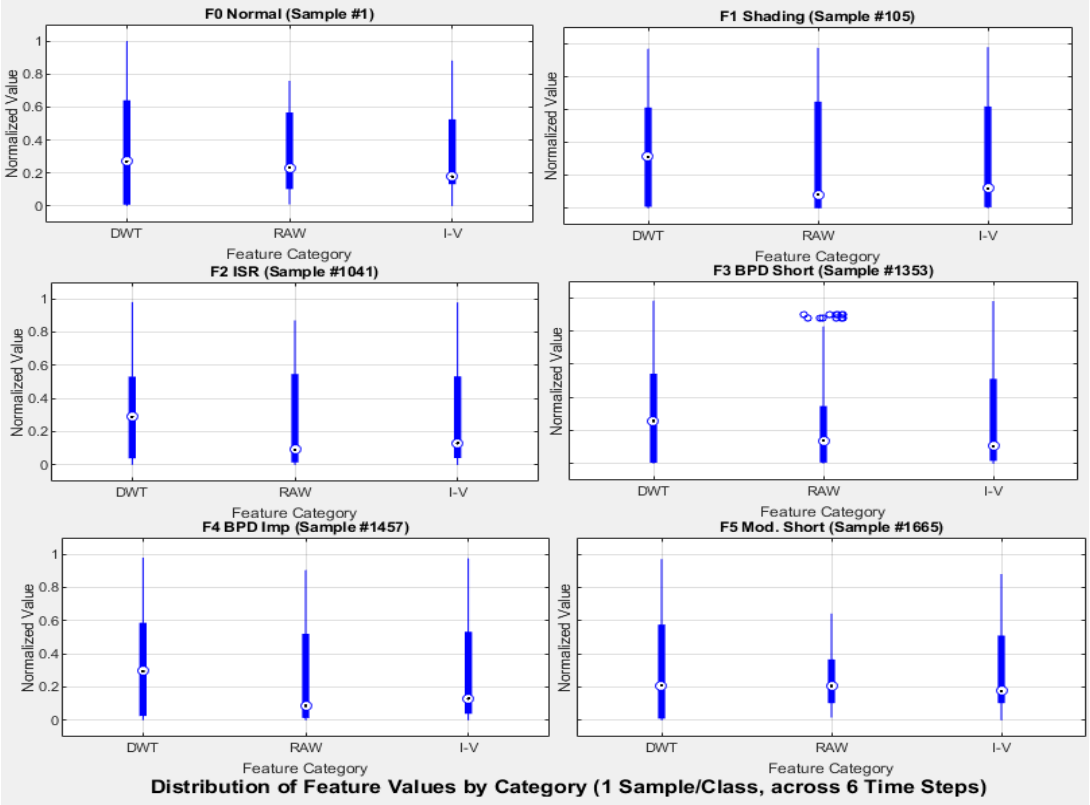

Figure IV.3: Distribution of Feature Values by Category for1 Sample per Class

IV.3.2 Feature Selection Using NCA and Its Impact on the Input Space

When we explored the dataset, we saw that the 90 features carry a lot of valuable details. Still, some of them are redundant or not very helpful, and this could slow down learning and affect performance. The next step was to use Neighborhood Component Analysis (NCA) to select only the most relevant features intelligently. To address this, we used Neighborhood Component Analysis (NCA) to perform a principled feature selection.

The core idea of NCA is to assign an "importance weight" to each feature based on its relevance for classification. The resulting weights for each of the 90 original features are shown in Figure IV.4.

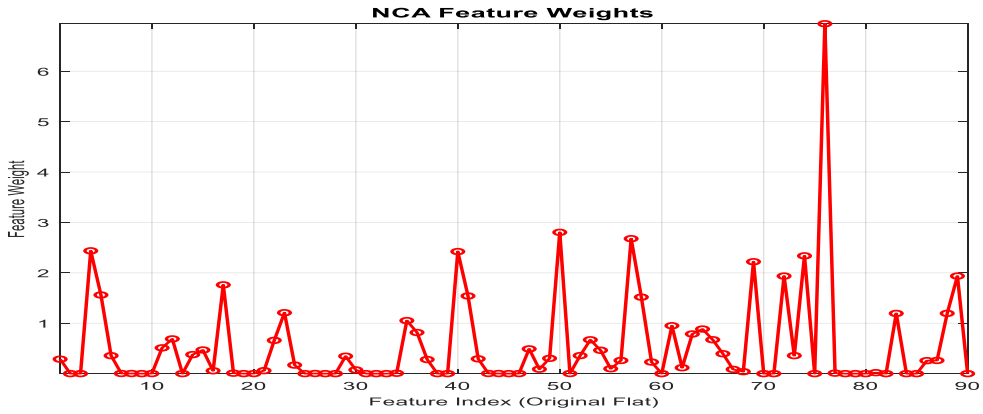

Figure IV.4: NCA Feature Weights for the initial 90 indicators

This graph provides a fascinating insight into the structure of our data. We can see a wide diversity of weights. Many indicators have a weight very close to zero, which confirms our hypothesis that they provide little to no useful information for distinguishing between the different fault classes.

On the other hand, the graph clearly shows that a select few features matter a lot more than the others. We can see several distinct peaks, with the feature located around index 75 having by far the highest score, making it the single most essential piece of information for our classifier. This tells us that a relatively small group of features is doing most of the "heavy lifting" in the classification task.

By applying a threshold of 0.0556 to these learned weights, the NCA process allowed us to systematically reduce our feature space from the original 90 features down to a more focused and robust set of 51 features.

A crucial aspect of this result is that the NCA process maintained a healthy balance between the three types of information. The final 51-feature set still contains significant contributions from each of our original categories:

- The DWT features, which capture the dynamic phenomena of the curve.
- The RAW statistics, which provide a global overview of deviations.
- The I-V parameters, which directly reflect the health status of the PV modules.

This confirms the relevance of our hybrid feature extraction approach. The final, reduced feature set is now a lean but powerful representation of the system's state. This reduction allows the BiLSTM to focus its training on the most essential information needed

to detect and classify the different types of PV defects, while eliminating the noise from less important characteristics.

IV.3.3 BiLSTM Classifier Performance and Detailed Analysis

After the preprocessing and NCA selection phases, our final dataset consisted of 2210 samples, which were split into a training set (1768 samples) and a test set (442 samples). The 51 selected features for each sample were then structured into sequences of six time steps, with each step represented by a vector of 35 indicators (eight dynamic DWT metrics combined with 27 static RAW and I-V indicators). This prepared data was then used to train and evaluate our proposed BiLSTM model.

IV.3.3.1 Optimization of Model Hyperparameters by Bayesian Approach

A key part of our methodology was to systematically optimize the model's hyperparameters to find the best possible configuration. To do this, we used Bayesian optimization (Bayesopt), which aims to find the hyperparameter combination that reduces the classification error measured on the validation set. In our case, we targeted four key hyperparameters: the number of BiLSTM units, the dropout rate, the number of units in the final fully connected layer, and the initial learning rate.

The optimization process was carried out over 20 iterations, which allowed it to explore different combinations of parameters efficiently. The optimal values found by the Bayesopt algorithm are summarized in Table IV.1.

Table IV.1: Optimal hyperparameters determined by Bayesian optimization**.**

Hyperparameter	Search Range	Optimal Value	Description
BiLSTM Units	[40-150]	106	Number of neurons in the BiLSTM layer.
Dropout Rate	[0.1-0.6]	0.1445	Dropout rate for regularization.
FC Units	[40-750]	500	Size of the dense (Fully Connected) layer.
Learning Rate	[1e-4, 2e-3]	2.48 e-4	Initial learning rate of the model.

We then took these optimal values and used them to put together our final model architecture. You can see the full configuration and all the training parameters we used in Table IV.2. It is worth noting that a number of additional training techniques also played a key role in reaching this level of performance. Class weights were applied during training to correct for the imbalance between the different fault classes (particularly the predominance of the F1 shading class). The training process used the Adam optimizer, an early stopping mechanism to prevent overfitting, and dropout for regularization.

Table IV.2: BiLSTM Model Architecture and Training Hyperparameters.

Parameter	Value	Explanation
Input Layer	35	Receives the input sequence at each time step
BiLSTM Layer	Optimized	Captures temporal dependencies in both directions
Dropout Rate	Optimized	Reduces overfitting by deactivating neurons
Fully Connected Layer 1 FC1	Optimized	First classification layer after the BiLSTM.
ReLU Layer	-	Introduces non-linearity after the FC1 layer
Fully Connected Layer 2 FC2	6 units	Final output layer with one neuron per class.
Softmax Layer	-	Converts the output scores into class probabilities.
Classification layer	-	Computes the loss and finalizes predictions
Learning Rate	Optimized	Initial learning rate for the Adam optimizer
Mini-batch Size	16	Number of samples processed in each iteration
Max Epochs	300	The maximum number of epochs for training
Early Stopping Patience	15	Number of epochs without improvement before stopping training

The decision to use Bayesian optimization instead of relying on default values was a critical factor in improving our model's performance. If we had just used the default settings for our model, the results would have been very different. Our systematic tuning process ended up discovering a much better configuration. It learned, for instance, that a dropout rate of 0.1445 —a third of the typical default—was far more effective. It also found that a slower learning rate of 2.48×10^{-4} was the key to a stable and accurate training process. These carefully chosen settings are the reason we were able to achieve the remarkable performance we will now describe. The effectiveness of this process is illustrated in the convergence plot in Figure IV.5.

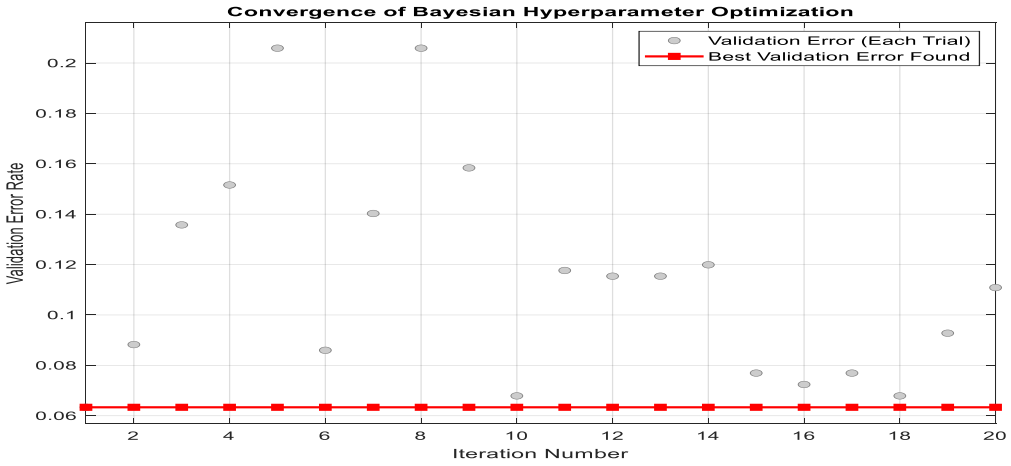

Figure IV.5: Convergence plot of the Bayesian optimization process. The gray circles represent the validation error for each of the 20 individual trials. The red line tracks the best validation error found up to each iteration,

IV.3.3.2 Quantitative and Per-Class Performance Analysis

To really understand how well our final model performs, we knew we had to go deeper than a single accuracy score. So, to get a complete, detailed picture of its capabilities, we used a whole set of more specific metrics. This includes looking at the confusion matrix, as well as the recall and F1-score for each individual fault class.

The initial evaluation revealed excellent overall performance. The model achieved an accuracy of 97.17% on the training set and 96.15% on the test set, corresponding to a low test error rate of just 3.85%. We also obtained a strong Kappa score of 0.9422, which indicates a "Very Good" match between our predictions and the real results. These metrics show that our classifier is trustworthy and generalizes well to new data. The Table IV.3 presents the performance of the BiLSTM model broken down by each fault class.

Table IV.3: Evaluation Results of the BiLSTM Model by Fault Class.

Class	Precision	Recall	F1-Score	Number Of sample
F0 Normal	96.2	0.962	0.962	26
F1 Shading	99.6	1.000	0.998	234
F2 ISR	100	0.808	0.894	78
F3 BPD Short	100	1.000	1.000	26
F4 BPD Imp	78.5	0.981	0.872	52
F5 Mod. Short	96.3	1.000	0.981	26

As the table shows, the model demonstrates an excellent ability to discriminate between the different classes. It shows exceptional reliability for several key fault types, achieving a perfect F1-score of 1.000 for F3 (BPD Short) and near-perfect scores of 0.998 and 0.981 for F1 (Shading) and F5 (Module Short), respectively. Both of these latter classes also benefit from a perfect recall of 1.000. The normal operating condition (F0) is also identified with high confidence, achieving a robust F1-score of 0.962.

However, the analysis also reveals a critical interaction between the F2 (ISR) and F4 (BPD Impedance) classes.

- For the F2 class, the model achieves perfect precision (100%), but its recall is lower at 0.808. This indicates that while its F2 predictions are always correct, it fails to identify approximately 19% of the actual ISR faults.

- Conversely, the F4 class exhibits an excellent recall of 0.981 but a lower precision of 78.5%. This suggests that while nearly all F4 faults are correctly found, the model sometimes incorrectly classifies other faults as F4.

This trade-off is further illustrated in the graph of per-class metrics (Figure IV.6) and the confusion matrix, which we will analyze in the next section.

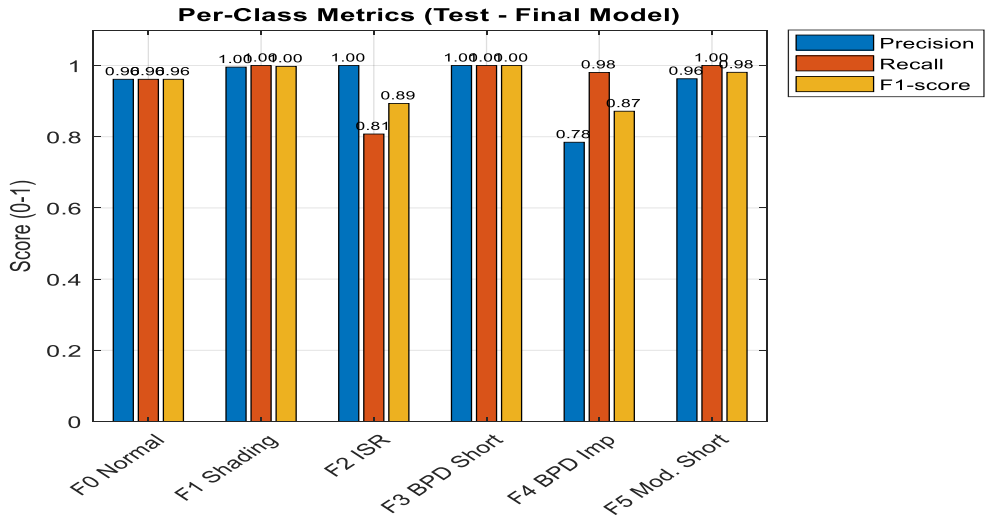

Figure IV.6: Graph of Per-Class Metrics (Precision, Recall and F1-score

IV.3.4 Visual Analysis of Classification Performance

While tables of metrics are precise, visual tools often give us a more intuitive understanding of a model's behavior. In this section, we will use the confusion matrix, a success rate plot, and a t-SNE visualization to take a deeper look at our classifier's strengths and weaknesses.

IV.3.4.1 The Confusion Matrix: A Detailed Look at the Model's Decisions

Let's take a closer look at the model's choices using the confusion matrix in Figure IV.7. It's a simple grid that gives us a visual report card of our model's performance on the test data. Each cell in the grid tells us how many samples from a real-world fault class ended up being classified correctly or incorrectly.

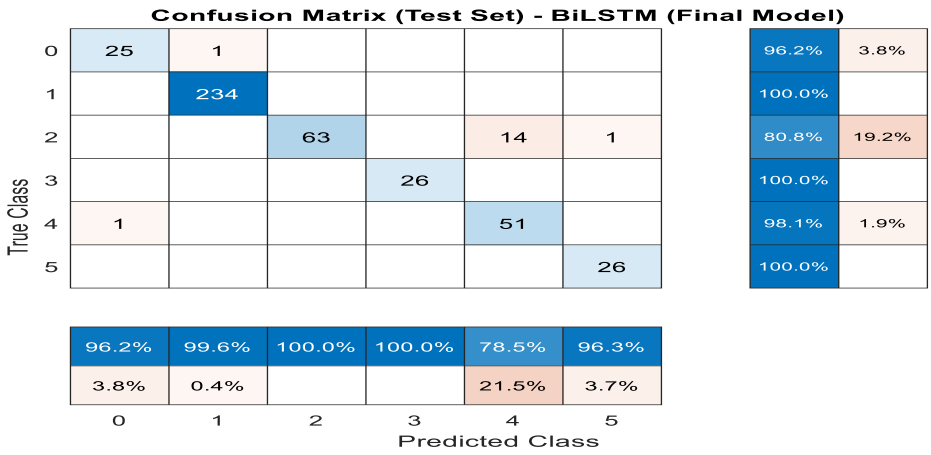

Figure IV.7: Confusion matrix for the BiLSTM model on the test set

The first thing we notice is the strong diagonal line, which represents correct classifications. This confirms the model's excellent overall performance, with particularly high accuracy for the Shading (F1), BPD Short (F3), and Module Short (F5) classes.

However, the matrix also clearly highlights the main challenge of this classification task: the significant confusion between the F2 (ISR) and F4 (BPD Impedance) classes. We can see that several real-world samples of the F2 class were misclassified as F4. This confirms a well-known issue in PV diagnostics: distinguishing between an increase in series resistance and a faulty bypass diode impedance is the primary fault classification challenge, as their electrical signatures can be very similar **[98, 99]**. This remains a significant area for

future work. From the confusion matrix, we can also directly calculate the model's overall precision on the test set:

$$\Pr ecision = \frac{Number\ of\ correct\ prediction}{Total\ number\ of\ test\ samples} \quad \text{(IV.1)}$$

$$\Pr ecision = \frac{424}{442} = 96{,}15\%$$

IV.3.4.2. Success Rate by Class

To visualize the recall for each class more directly, we can plot the success rate, as shown in Figure IV.8. This chart graphically illustrates the "hit rate," or the percentage of samples from each class that were correctly identified.

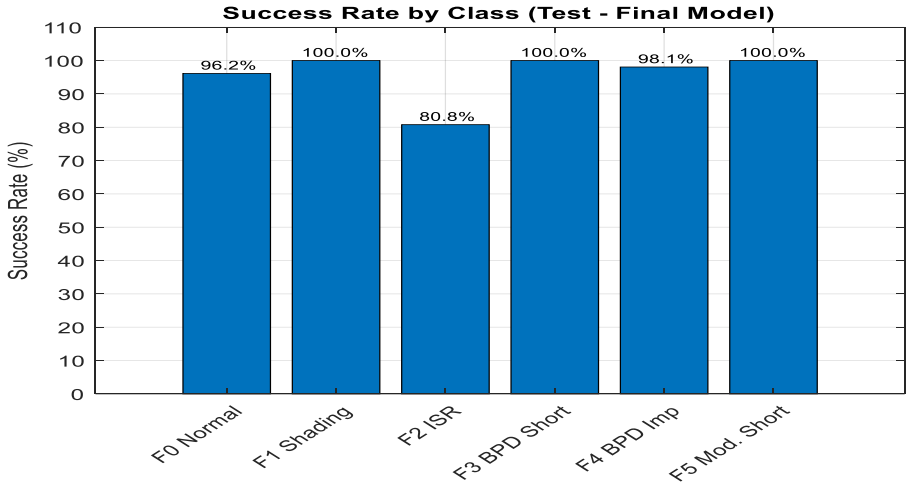

Figure IV.8: Success Rate (recall) by class for the BiLSTM model

This plot confirms the excellent recall values for most of the defect classes, especially F1, F3, and F5, which all reach 100%. It also visually highlights the lower, though still strong, recall for the F2 class (80.8%), providing another view of the confusion with the F4 class.

IV.3.4.3 Visualizing Defect Separation with t-SNE

Finally, we wanted to see how the network itself is "thinking" about the different fault classes. To do this, we used a t-SNE visualization, which takes the high-dimensional data representations learned by the network and projects them down into a 2D space that we can look at. The t-SNE plot, shown in Figure IV.9, reveals how the data representations learned by the network's final layers are clustered.

This visualization is very revealing. It shows that despite some overlap between the F2 and F4 classes, the model has learned to create largely distinct clusters for each fault type. This better separation is a direct result of our hybrid approach: the detailed features extracted by the DWT and the bidirectional processing of the BiLSTM have given the model the information it needs to tell these tricky defects apart.

The fact that our model achieves high F1-scores for both F2 (0.894) and F4 (0.872), even with their similarities, proves the effectiveness of our approach. It shows that the combination of rich preprocessing and an optimized BiLSTM architecture allows us to distinguish between defects much more effectively, even those that are difficult to separate.

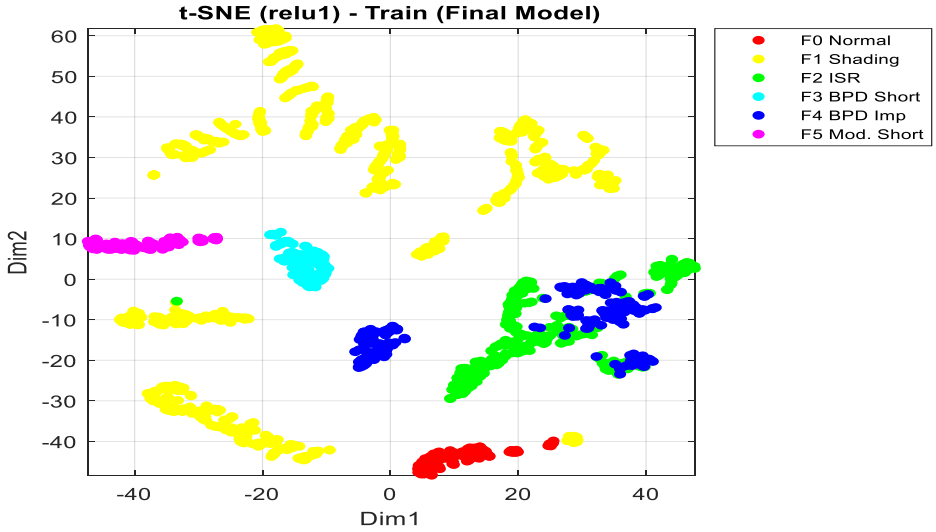

Figure IV.9: t-SNE visualization of the activations of the relu1 layer of the BiLSTM model

IV.3.5 Validating the Architectural Choices

Our hybrid framework is built on two big ideas. The first was to use a bidirectional architecture (the BiLSTM), and the second was to feed it with innovative features that we extracted using the DWT, rather than just raw data. In this final section of our analysis, we will present results that validate these choices by comparing our model's performance against two vital benchmarks: a unidirectional LSTM and a state-of-the-art model from the literature.

IV3.5.1 The Importance of Bidirectional Context: BiLSTM vs. Unidirectional LSTM

To prove that analyzing the feature sequence in both directions is a critical advantage, we trained a standard unidirectional LSTM model using the exact same data, features, and training protocol. We then compared its performance on a class-by-class basis against our final BiLSTM model.

The architecture of this comparative unidirectional LSTM was designed to be as similar as possible to our main BiLSTM model to ensure a fair benchmark. Its detailed configuration and training hyperparameters are presented in Table IV.4.

Table IV.4: Configuration of the Comparative Unidirectional LSTM Model.

Parameter	Value
Input Features	35 (from the 51 selected)
Number of LSTM Layers	1
Neurons per LSTM Layer	100
Dropout Rate	0.2
Number of FC Layers	2
Neurons in First FC Layer	100
Activation Function	ReLU
Output Layer	6 Neurons (Softmax)
Optimizer	Adam
Initial Learning Rate	0.001
Mini-batch Size	32
Regularization	Class Weighting, Early Stopping

The complete results of this comparison can be found in Table IV.5. and visualized in Figure IV.10.

The results show a clear and significant superiority for the BiLSTM model across almost all metrics. While the unidirectional LSTM performs reasonably well on the simpler fault classes, its performance drops dramatically on the more challenging ones.

Table IV.5: Comparative performance of the BiLSTM and unidirectional LSTM models.

	BiLSTM Precision	LSTM Precision	BiLSTM Recall	LSTM Recall	BiLSTM F1-Score	LSTM F1-Score
F0	**96.2**	**78.1**	**0.962**	**0.961**	**0.962**	**0.862**
F1	**99.6**	**99.1**	**1.000**	**1**	**0.998**	**0.995**
F2	**100**	**67.3**	**0.808**	**0.871**	**0.894**	**0.759**
F3	**100**	**86.3**	**1.000**	**0.730**	**1.000**	**0.791**
F4	**78.5**	**66.6**	**0.981**	**0.346**	**0.872**	**0.455**
F5	**96.3**	**95.8**	**1.000**	**0.884**	**0.981**	**0.920**

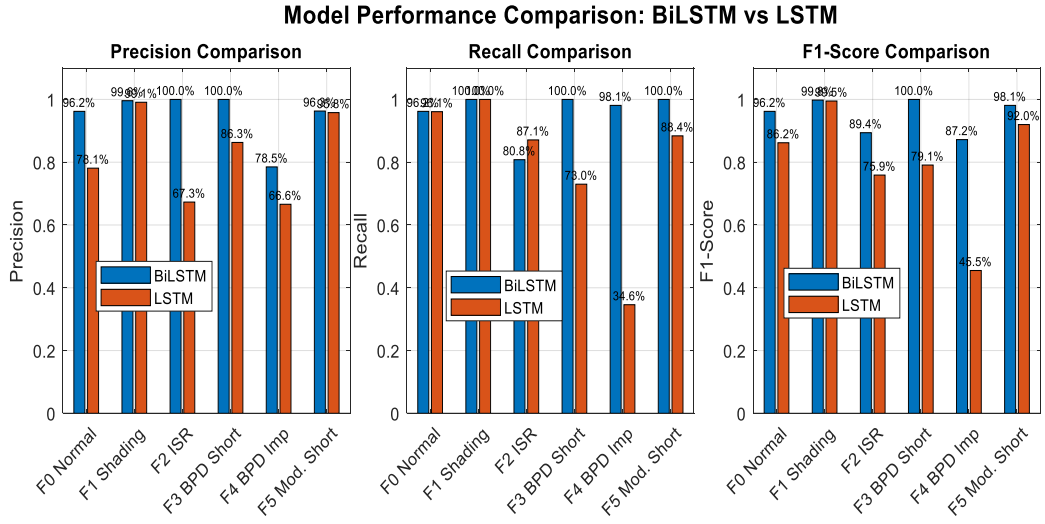

Figure IV.10: Visual comparison of Precision, Recall and F1-Score by Class, for BiLSTM vs. LSTM Models

The most telling difference is in the model's ability to distinguish between the difficult F2 (ISR) and F4 (BPD Impedance) classes.

- The BiLSTM achieves respectable F1-scores of 0.894 for F2 and 0.872 for F4.
- In stark contrast, the unidirectional LSTM completely fails on this task, with F1-scores dropping to 0.759 for F2 and a very poor 0.455 for F4.

This demonstrates that the "forward-only" view of the unidirectional LSTM is not sufficient to capture the subtle signatures of these faults. The ability of the BiLSTM to see the "whole

story"—using both past and future context within the feature sequence—is precisely what allows it to separate these tricky defect classes better. This confirms that our choice of a bidirectional architecture was not just a minor improvement, but a fundamental requirement for achieving robust performance on this problem.

IV.3.5.2 Benchmarking Against Previous Work: Comparison with MC-NFC

To further validate our model's performance, we benchmarked it against an existing state-of-the-art approach: the multiclass neuro-fuzzy classifier (MC-NFC) proposed for the same PV fault diagnosis problem in **[99]**.

a- Challenges of a Fair Comparison

Making a direct, apples-to-apples comparison between the two models presented some challenges. The reference study in **[99]** uses different metrics for its evaluation, focusing on per-class RMSE and R^2 without including standard classification indicators like precision or recall. Furthermore, their MC-NFC model uses a "one-against-all" binary classifier strategy, whereas our BiLSTM model is a true multiclass classifier.

This difference could affect the interpretation of a comparison based solely on RMSE. For this reason, we decided to use the per-class R^2 score as the most reliable driver for our comparison, as it measures how well each model can explain the variance for each specific fault type. To ensure the comparison was as fair as possible, the per-class R^2 for our BiLSTM model was calculated using the exact same formula as the one used in the reference study.

b- Analysis of the Comparative Results

The results of this comparison are presented in Table IV.6.

From these results, it is clear that our DWT-BiLSTM model demonstrates a performance that is either equivalent or significantly superior to the reference MC-NFC model across all analyzed defect classes (F1 to F5).

- For the Shading (F1), BPD Short (F3), and Module Short (F5) classes, both models reach or approach a perfect performance, with R^2 scores at or near 1.00. This shows that for these more distinct fault types, both methods are highly effective.

However, the real test comes with the more challenging and confusable fault types. Here, our model shows significant improvements:

- For the ISR (F2) fault, the R^2 of our BiLSTM is 0.87, compared to just 0.65 for the MC-NFC.
- For the BPD Impedance (F4) fault, the R^2 of our BiLSTM is an outstanding 0.98, compared to 0.74 for the MC-NFC.

Table IV.6: Comparative performance (R^2) by fault type: MC-NFC vs. our BiLSTM.

Class	R^2 MC-NFC of Article [99]	R^2 of our BiLSTM	Relative performance of BiLSTM
F0 Normal	N/A	0.00	- (non-comparable)
F1 Shading	0.95	1.00	Better
F2 ISR	0.65	0.87	Significantly Better
F3 BPD Short	1.00	1.00	Identical
F4 BPD Imp	0.74	0.98	Significantly Better
F5 Mod. Short	1.00	1.00	Identical

This dramatic outperformance, especially on the most difficult-to-distinguish fault types, provides strong proof of the effectiveness of our hybrid approach. The combination of rich, multi-resolution DWT features and the powerful bidirectional processing of the BiLSTM has made it possible to distinguish between complex fault signatures much more effectively than previous methods, even those that are difficult to separate.

IV.4 Conclusion

This chapter set out to tackle a different but complementary challenge to the previous one: instead of *modeling* a PV system, our goal was to classify its state of health. We started with the hypothesis that for a complex problem like fault diagnosis, a robust hybrid framework would be more effective than any single, monolithic approach.

The results we have presented strongly support this hypothesis. Our work has demonstrated the power of a complete and rigorous methodology that combines three key pillars: multi-resolution feature extraction using the DWT, intelligent feature selection with NCA, and a robust BiLSTM classifier whose architecture was systematically optimized through Bayesian optimization.

The results speak for themselves. With a final classification accuracy of 96.15% on the test set, our DWT-BiLSTM model proved to be highly effective. When we compared our model to a standard unidirectional LSTM and another leading method, a clear winner emerged. Our BiLSTM framework proved to be far more robust, especially on the most difficult part of the problem: correctly separating the challenging ISR and BPD Impedance faults.

This superior robustness confirms the necessity of the hybrid approach for high-level decision tasks. Unlike the modeling task (Chapter III), where the network reproduced a clean signal, the goal here was to make a robust decision based on subtle, often non-stationary signatures (e.g., those causing F2 and F4 confusion). In this context, expert feature engineering using the DWT was vital, allowing us to distil the raw signal into a smaller, highly informative set of indicators. This targeted input made the learning task easier for the BiLSTM, ensuring high accuracy and better generalisation even with a moderately sized dataset.

This high performance can be attributed to our model's ability to learn complex relationships directly from a rich, well-curated set of features.

However, our analysis also honestly acknowledged the remaining challenges. The residual confusion between the F2 and F4 fault classes highlights a fundamental difficulty in PV diagnostics and points to valuable directions for future research.

Ultimately, this chapter has delivered on its promise. It has presented a high-performing and robust solution for PV fault monitoring. But more importantly, it has validated a powerful methodological strategy—combining advanced signal processing with an optimized deep learning architecture—that can be extended to other complex classification problems.

Having now successfully demonstrated the power of the BiLSTM for both modeling (Chapter III) and classification (Chapter IV), we are fully equipped to take a step back. In the final chapter, we will conduct a transversal analysis, comparing the performance and characteristics of these two approaches to draw more general conclusions about the use of dynamic neural networks in engineering.

GENERAL CONCLUSION

General Conclusion

This doctoral work began with a fundamental question at the intersection of control theory and artificial intelligence: for the "black-box" modeling of nonlinear dynamic systems, does the state-space approach offer real advantages over traditional input-output models? To answer this question, we studied dynamic neural networks across two distinct yet complementary tasks: fine-grained modeling and robust classification, using the photovoltaic (PV) system as a challenging case study.

We first examined the basics of neural networks, observing how they evolved from simple, static models to more advanced ones, such as the BiLSTM, which can retain memory from both the past and the future. This groundwork laid the groundwork for our main contributions, with a strong focus on the specific learning methods required for these complex models, notably Backpropagation Through Time (BPTT) and newer optimisation techniques.

Our research takes place against the backdrop of a significant shift in global energy use. The rising demand for energy is prompting many countries to rely more heavily on renewable sources, particularly solar power. However, ensuring these systems are reliable remains a big challenge. Photovoltaic installations can face numerous problems—from partial shading to cell wear—that reduce their output, increase maintenance costs, and may even impact the safety of the electrical grid. Faced with this challenge, the development of fast, accurate, and robust modeling and diagnostic methods is no longer an option, but a necessity. It is precisely to meet this need that we turned to dynamic neural networks.

This work contributes to examining the benefits of black-box modeling of nonlinear dynamic systems using dynamic neural networks described in state-space, compared to input-output models. Our results clearly answered the initial question: the state-space approach does indeed offer significant advantages. Our work proved that the BiLSTM architecture is remarkably versatile. It performed exceptionally well on both fine-grained regression (like modeling, where it hit an outstanding R2 score over 0.997) and high-level classification (for fault diagnosis, achieving over 96% accuracy). The main reason behind this dual success is the BiLSTM's strong ability to grasp sequential context.

Our study revealed a critical methodological trade-off: The modeling task (Chapter 3) was a significant success using an end-to-end approach, which proved ideal because the input data was clean and the goal was to reproduce the signal accurately. By doing this, we validated the state-space paradigm. The network's hidden states were forced to learn internal dynamics that made physical sense, successfully changing the opaque "black box" into a meaningful "grey box." Conversely, for the high-level decision task of fault classification (Chapter 4), the hybrid approach (DWT for feature extraction + NCA for selection + BiLSTM) was proven superior, as guiding the model with expert feature engineering is a more robust strategy for systems where fault signatures are subtle and easily confused.

Overall, this work helps us understand and use dynamic neural networks more effectively. We have demonstrated that the state-space approach is not merely an alternative for modeling systems; It is the best choice, allowing us to build models that are both extremely precise and easy to interpret. The potential synergy identified—where the physically coherent internal state of the "digital twin" could be used as a "super-feature" for future classification networks—opens a fascinating path forward for making black-box models less opaque and more trustworthy.

While our results are auspicious, we must be honest about our limits. Our primary constraint is relying entirely on simulated data from a high-fidelity emulator. The actual test for robustness will be validating these models using noisy, unpredictable real-world experimental data. Furthermore, residual confusion persists between inherently confusable fault types (e.g., ISR/BPD Impedance), and our architectural focus suggests that exploring newer state-of-the-art architectures, such as Transformers or Mamba, is warranted for comparative studies. Based on these findings, this work opens up several exciting and logical directions for future work, including the essential validation on real-world data and the implementation of the modeling-classification synergy we proposed.

Finally, this research clearly shows how valuable dynamic neural networks are for managing complex systems. They offer significant benefits for predictive maintenance and more intelligent control of energy systems.

REFERENCES

[1] I. D. Mienye and T. G. Swart, "A Comprehensive Review of Deep Learning: Architectures, Recent Advances, and Applications," Information, vol. 15, no. 12, p. 755, 2024, doi: 10.3390/info15120755.

[2] I. Rivals and L. Personnaz, "Black-Box Modeling with State-Space Neural Networks," in Neural Adaptive Control Technology, R. Zbikowski and K. J. Hunt, Eds. World Scientific, 1996, pp. 237-264.

[3] S. H. Strogatz, Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering. Westview Press, 1994.

[4] K. Boudraa, M. A. Ouali, and M. Ladjal, "Hybrid FFT-ARMA-Burg Modeling and LSTM-Enhanced BBO Optimization for Fault Diagnosis in Induction Motors," ITEGAM-Journal of Engineering and Technology for Industrial Applications, vol. 11, no. 52, pp. 50-57, 2025, doi: 10.5935/jetia.v11i52.1510.

[5] S. Medjmadj, "Hybrid sensorless control of PMSM in full speed range using HFI and back-EMF," Engineering Review, vol. 44, no. 2, pp. 1-13, 2024, doi: 10.30765/er.2103.

[6] A. May, F. Krim, H. Feroura, and A. Belaout, "Power quality enhancement of grid-tied 7L-PUC inverter-based PV system using a novel DC-link controller," Arabian Journal for Science and Engineering, vol. 48, no. 11, pp. 15305–15319, Nov. 2023, doi: 10.1007/s13369-023-08074-3.

[7]P. Z. Csurcsia, J. Decuyper, B. Renczes, M. C. Runacres, et T. De Troyer, « Reducing black-box nonlinear state-space models: A real-life case study », Mechanical Systems and Signal Processing, vol. 211, p. 111230, avr. 2024, doi: 10.1016/j.ymssp.2024.111230.

[8] R. Hu et al., "Black-Box Modeling of Active Distribution Network Devices Based on Neural ODEs," J. Phys.: Conf. Ser., vol. 2826, p. 012029, 2024, doi: 10.1088/1742-6596/2826/1/012029.

[9] P. Z. Csurcsia et al., "Reducing black-box nonlinear state-space models: A real-life case study," Mechanical Systems and Signal Processing, vol. 211, p. 111230, 2024, doi: 10.1016/j.ymssp.2024.111230.

[10] G. Dreyfus et al., Réseaux de neurones, Méthodologies et Application. Paris, France: Eyrolles, 2002.

[11] D. Phan-Trong, H. Tran-The, and S. Gupta, "Neural-BO: A Black-box Optimization Algorithm using Deep Neural Networks," arXiv preprint arXiv:2303.01682, 2023.

[12] L. Ljung, System Identification: Theory for the User, 2nd ed. Prentice Hall, 1999.

[13] L. Bun, "Détection et Localisation de Défauts pour un Système PV," Thèse de doctorat, Université de Grenoble, 2011.

[14] K. S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," IEEE Trans. Neural Netw., vol. 1, no. 1, pp. 4–27, Mar. 1990, doi: 10.1109/72.53653.

[15] R. Zemouri, "Contribution à la surveillance des systèmes de production à l'aide des réseaux de neurones dynamiques : Application à la e-maintenance," Thèse de

doctorat, Univ. de Franche-Comté, Besançon, France, 2003.
[16] H. K. Khalil, Nonlinear Systems, 3rd ed. Prentice Hall, 2002.
[17] G. E. P. Box and G. M. Jenkins, Time Series Analysis: Forecasting and Control. Holden-Day, 1970.
[18] H. Tong, Nonlinear Time Series: A Dynamical System Approach. Oxford University Press, 1990.
[19] O. Nelles, Nonlinear System Identification. Springer, 2001.
[20] I. H. Sarker, "Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions," SN Computer Science, vol. 2, no. 6, pp. 1–20, 2021, doi: 10.1007/s42979-021-00815-1.
[21] A. Even-Zohar and D. Roth, "A Sequential Model for Multi-Class Classification," in Proceedings of the 39th Annual Meeting on Association for Computational Linguistics, Toulouse, France, 2001, pp. 209-216, doi: 10.3115/1073012.1073042.
[22] X. Xing, J. Pei, J. Wang, and P. Yu, "A Brief Survey on Sequence Classification," SIGKDD Explorations, vol. 12, no. 1, pp. 40-48, June 2010, doi: 10.1145/1809400.1809412.
[23] H. Chen et al., "Classification of multi-lead ECG based on multiple scales and hierarchical features with Lead Encoder Attention," Scientific Reports, vol. 14, no. 1, p. 9427, 2024.
[24] : T. P. C. da Costa et al., "Deep learning-based strategy for rotating machinery fault diagnosis," Sensors, 2021.]
[25] A. Graves, A. R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in 2013 IEEE international conference on acoustics, speech and signal processing, 2013.
[26] M. E. H. Benbouzid, “A review of induction motors signature analysis as a medium for faults detection,” IEEE Transactions on Industrial Electronics, vol. 47, no. 5, pp. 984-993, 2000.
[27] N. Mariani et al., “An Approach for Power System Condition Monitoring with Sensor Data Analytics,” IEEE Transactions on Power Systems, vol. 35, no. 4, pp. 2812-2821, 2020.
[28] : S. H. K. Narra et al., "An LSTM-based approach for analog circuit fault classification and localization," in 2021 IEEE 18th India Council International Conference (INDICON), 2021.]
[29] V. Veerasamy, A. Muhammad, M. A. Pasha, S. Aris, and K. S. Lim, "LSTM Recurrent Neural Network Classifier for High Impedance Fault Detection in Solar PV Integrated Power System," IEEE Access, vol. 9, pp. 32667–32686, 2021.
[30] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” Proceedings of the IEEE, vol. 77, no. 2, pp. 257-286, Feb. 1989.
[31] C. Cortes and V. Vapnik, “Support-vector networks,” Machine Learning, vol. 20, no. 3, pp. 273-297, Sep. 1995.

[32] H. T. Siegelmann, B. G. Horne, and C. L. Giles, “Computational Capabilities of Recurrent NARX Neural Networks,” IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, vol. 27, no. 2, pp. 208-215, Apr. 1997.

[33] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, “Phoneme recognition using time-delay neural networks,” IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, no. 3, pp. 328-339, Mar. 1989.

[34] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Networks, vol. 2, no. 5, pp. 359-366, 1989. doi: 10.1016/0893-6080(89)90020-8.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Adv. Neural Inf. Process. Syst., 2012, pp. 1097-1105.

[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436-444, 2015. doi: 10.1038/nature14539.

[37] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in Proc. Int. Conf. Mach. Learn., 2013, pp. 1310-1318. doi: 10.48550/arXiv.1211.5063.

[38] S. Haykin, Neural Networks and Learning Machines, 3rd ed. Upper Saddle River, NJ, USA: Pearson Education, Inc., 2009. (Note : Les éditions plus anciennes des manuels n'ont souvent pas de DOI assigné.)

[39] C. M. Bishop, Pattern Recognition and Machine Learning. New York, NY, USA: Springer, 2006.

[40] X. Zhu, "Semi-supervised learning literature survey," Computer Sciences Technical Report 1530, University of Wisconsin-Madison, 2005.

[41] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: The MIT Press, 2018.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, no. 6088, pp. 533–536, Oct. 1986, doi: 10.1038/323533a0.

[43] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: The MIT Press, 2016. (Note : Ce livre est disponible en ligne gratuitement, mais l'édition publiée n'a pas de DOI standard.)

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, Jun. 2014.

[45] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Networks, vol. 2, no. 5, pp. 359-366, 1989, doi: 10.1016/0893-6080(89)90020-8.

[46] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," IEEE Trans. Acoust., Speech, Signal Process., vol. 37, no. 3, pp. 328-339, Mar. 1989, doi: 10.1109/29.21701.

[47] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[48] A. Ermshaus, P. Schäfer, and U. Leser, "Window size selection in unsupervised time series analytics: A review and benchmark," in International Workshop on Advanced Analytics and Learning on Temporal Data, Springer, Cham, 2023, pp. 83-101, doi: 10.1007/978-3-031-77066-1_6.

[49] T. S. Babu, N. Rajasekar, and K. Sangeetha, "A review on challenges and issues of photovoltaic condition monitoring systems," Renewable and Sustainable Energy Reviews, vol. 47, pp. 719-737, Jul. 2015, doi: 10.1016/j.rser.2015.03.048.

[50] B. Taghezouit, F. Harrou, Y. Sun, and W. Merrouche, "Model-based fault detection in photovoltaic systems: A comprehensive review and avenues for enhancement," Results in Engineering, vol. 21, p. 101835, 2024.

[51] G. R. Venkatakrishnan, R. Rengaraj, S. Tamilselvi, et al., "Detection, location, and diagnosis of different faults in large solar PV system-a review," International Journal of Low-Carbon Technologies, vol. 18, pp. 659-674, 2023.

[52] Pourboghrat, F., Pongpairoj, H., Liu, Z., Farid, F., Pourboghrat, F., & Aazhang, B. (2003). Dynamic neural networks for modeling and control of nonlinear systems. Intelligent Automation and Soft Computing, 9(1), 1-12. https://doi.org/10.1080/10798587.2000.10642843

[53] Gündoğdu, S. (2020). Comparison of static MLP and dynamic NARX neural networks for forecasting of atmospheric PM10 and SO2 concentrations in an industrial site of Turkey. Environmental Monitoring and Assessment, 192(7), 1 16. https://doi.org/10.1080/15275922.2020.1771637

[54] Solovyeva, E. B. (2017). Types of recurrent neural networks for non-linear dynamic system modelling. Soft Computing Conference. https://doi.org/10.1109/SCM.2017.7970552

[55] Ogunmolu, O., Gu, X., Jiang, S. B., & Gans, N. (2016). Nonlinear Systems Identification Using Deep Dynamic Neural Networks. arXiv: Neural and Evolutionary copmuting. https://arxiv.org/pdf/1610.01439.pdf

[56] Han, Y., Huang, G., Song, S., Yang, L., Wang, H., & Wang, Y. (2022). Dynamic Neural Networks: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(11), 7436-7456. https://doi.org/10.1109/tpami.2021.3117837

[57] J. A. K. Suykens, J. Vandewalle, and B. L. R. De Moor, "NLq theory: a neural control framework for global modeling of nonlinear systems," IEEE Transactions on Neural Networks, vol. 7, no. 4, pp. 1021-1036, Jul. 1996

[58] José M. P. Menezes Jr. et Guilherme A. Barreto, “A New Look at Nonlinear Time Series Prediction with NARX Recurrent Neural Network,” Proceedings of the Ninth Brazilian Symposium on Neural Networks (SBRN 2006), octobre 2006. DOI : 10.1109/SBRN.2006.7

[59] M. I. P. Hidayat, P. S. M. M. Yusoff, and W. Berata, “Neural networks with NARX structure for material lifetime assessment application,” in Proc. 2011 IEEE Symposium on Computers and Informatics (ISCI 2011), Kuala Lumpur, Malaysia, Mar. 20–22 2011, pp. 273–278. doi: 10.1109/ISCI.2011.5958926.

[60] Kim, K. K., Patron, E. R., & Braatz, R. D. (2011). Universal approximation with error bounds for dynamic artificial neural network models: A tutorial and some new results. IEEE International Symposium on Computer Aided Control System design. https://doi.org/10.1109/CACSD.2011.6044542

[61] Amoura, K., Wira, P., & Djennoune, S., “A state-space neural network for modeling dynamical nonlinear systems,” in Proceedings of the International Conference on Neural Computation Theory and Applications (NCTA–IJCCI 2011), vol. 1, pp. 369–376, 2011, doi: 10.5220/0003680503690376.

[62] P. J. Werbos, "Backpropagation through time: what it does and how to do it," Proceedings of the IEEE, vol. 78, no. 10, pp. 1550-1560, Oct. 1990, doi: 10.1109/5.58337.

[63 I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016 , URL : https://www.deeplearningbook.org/

[64] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," Diploma thesis, Institut für Informatik, Technische Universität München, Munich, Germany, 1991. [in German].

[65]Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157-166, Mar. 1994, doi: 10.1109/72.279181.

[66]D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[67]M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," IEEE Transactions on Neural Networks, vol. 5, no. 6, pp. 989-993, Nov. 1994, doi: 10.1109/72.329697.

[68] D. J. C. MacKay, "Bayesian Methods for Adaptive Models," Ph.D. dissertation, Dept. Comput. Neural Syst., California Inst. Technol., Pasadena, CA, USA, 1992.

[69] J. L. Elman, "Finding structure in time," Cognitive Science, vol. 14, no. 2, pp. 179-211, Apr. 1990, doi: 10.1207/s15516709cog1402_1.

[70] M. I. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," in Proceedings of the Eighth Annual Conference of the Cognitive Science Society, Amherst, MA, 1986, pp. 531–546.

[71] G. Palm, "On representation and approximation of nonlinear systems, Part II: Discrete time," Biological Cybernetics, vol. 34, no. 1, pp. 49-52, 1979, doi: 10.1007/BF00337166.

[72] M. O. Franz and B. Schölkopf, "A unifying view of Wiener and Volterra theory and polynomial kernel regression," Neural Computation, vol. 18, no. 12, pp. 3097-3118, Dec. 2006, doi: 10.1162/neco.2006.18.12.3097.

[73] O. NERRAND, P. ROUSSEL-RAGOT, L. PERSONNAZ & G. DREYFUS « Neural Networks and Non-linear Adaptive Filtering: Unifying Concepts and New Algorithms. » Neural Computation, Vol. 5, pp 165-199, 1993

[74]I.RIVALS, L PERSONNAZ « Les réseaux de neurones formels pour la modélisation, la commande et la classification » CNRS Editions, Paris 2003

[75] P. Lara-Benítez, M. Carranza-García, and J. C. Riquelme, "An experimental review on deep learning for time series forecasting," Sensors, vol. 21, no. 11, p. 3905, 2021, doi: 10.3390/s21113905.

[76] Mienye, I.D.; Swart, T.G.; Obaido, G. Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications. Information 2024, 15, 517. https://doi.org/10.3390/info15090517

[77] S. Hochreiter et J. Schmidhuber, « Long short-term memory », Neural Computation, vol. 9, no. 8, pp. 1735–1780, nov. 1997, doi: 10.1162/neco.1997.9.8.1735. doi: 10.1109/CASE56687.2023.10260331.

[78] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," Neural Computation, vol. 12, no. 10, pp. 2451-2471, 2000, doi: 10.1162/089976600300015015.

[79] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, Oct. 2014, pp. 1724–1734, doi: 10.3115/v1/D14-1179.

[80] S. Mohsen, "Recognition of human activity using GRU deep learning algorithm," Multimedia Tools and Applications, vol. 82, pp. 47733-47749, May 2023, doi: 10.1007/s11042-023-15571-y.

[81] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM networks," Proceedings. 2005 IEEE International Joint Conference on Neural Networks, Montreal, QC, Canada, 2005, vol. 4, pp. 2047–2052, doi: 10.1109/IJCNN.2005.1556215

[82] I. K. Ihianle, A. N. Nwajana, S. H. Ebenuwa, R. I. Otuka, K. O. Owa, and M. O. Orisatoki, "A Deep Learning Approach for Human Activities Recognition From Multimodal Sensing Devices," *IEEE Access*, vol. 8, pp. 179028–179038, 2020, doi: 10.1109/ACCESS.2020.3027979.

[83] M. Aamir, M. A. Bhatti, S. U. Bazai, S. Marjan, A. M. Mirza, A. Wahid, A. Hasnain et U. A. Bhatti, "Predicting the Environmental Change of Carbon Emission Patterns in South Asia: A Deep Learning Approach Using BiLSTM," Atmosphere, vol. 13, no. 12, p. 2011, 2022, doi: 10.3390/atmos13122011

[84] A.Shenfield and M. Howarth, "A novel deep learning model for the detection and identification of rolling element-bearing faults," Sensors, vol. 20, no. 18, p. 5112, 2020, doi: 10.3390/s20185112

[85] D. Gedon, N. Wahlström, T. B. Schön, and L. Ljung, "Deep state space models for nonlinear system identification," IFAC-PapersOnLine, vol. 54, no. 7, pp. 481-486, 2021, doi: 10.1016/j.ifacol.2021.07.098.

[86] G. Pillonetto, A. Aravkin, D. Gedon, L. Ljung, A. H. Ribeiro, and T. B. Schön, "Deep networks for system identification: A survey," Automatica, vol. 144, p. 110349, 2024, doi: 10.1016/j.automatica.2023.110349.

[87] L. N. Boucetta, Y. Amrane, A. Chouder, S. Arezki, and S. Kichou, "Enhanced Forecasting Accuracy of a Grid Connected Photovoltaic Power Plant: A Novel

Approach Using Hybrid Variational Mode Decomposition and a CNN LSTM Model," Energies, vol. 17, no. 7, p. 1781, 2024.

[88] Y. Wang, Y. Zhou, C. Lv, et al., "Predicting photovoltaic power generation using double-layer bidirectional long short-term memory–convolutional network," International Journal of Energy and Environmental Engineering, vol. 13, p. 9339, 2022.

[89] B. Chen, P. Lin, Y. Lai, S. Cheng, Z. Chen, and L. Wu, "Very-short-term power prediction for PV power plants using a simple and effective RCC-LSTM model based on short term multivariate historical datasets," Electronics, vol. 9, no. 2, p. 289, 2020.

[90] A. F. Amiri, H. Oudira, A. Chouder, and A. F. Amiri, "Prediction Model of PV Module Based on Artificial Neural Networks for the Energy Production," in Proc. 5th Novel Intelligent and Leading Emerging Sciences Conference (NILES), Egypt, Oct. 2023.

[91] H. A. Mohammed, R. Mohd-Mokhtar, and H. I. Ali, "An optimal adaptive neuro-fuzzy inference system for photovoltaic power system optimization under partial shading conditions," Energy Systems, pp. 1-24, 2025.

[92] J. A. K. Suykens, B. L. R. De Moor, and J. Vandewalle, "Nonlinear system identification using neural state space models, applicable to robust control design," International Journal of Control, vol. 62, no. 1, pp. 129–152, 1995.

[93] S. Jiang and X. Li, "Approximation Rate of the Transformer Architecture for Sequence Modeling," arXiv preprint arXiv:2305.18475, 2023.

[94] S. Agrawal, K. Sharma, et al."Transformer Based Time Series Prediction of the Maximum Power Point of Photovoltaic Cells," arXiv preprint arXiv:2409.16342, 2024.

[95] J. Scott et al., "Transformers Applied to Short-term Solar PV Power Output Forecasting," arXiv preprint arXiv:2505.03188, 2025.

[96] J. Gu et al., "Mamba: Linear-Time Sequence Modeling with Selective State Spaces," arXiv preprint arXiv:2312.00752, 2023.

[97] E. I. Batzelis, G. Anagnostou, I. R. Cole, T. R. Betts, and B. C. Pal, "A state-space dynamic model for photovoltaic systems with full ancillary services support," IEEE Transactions on Sustainable Energy, vol. 10, no. 3, pp. 1399-1409, 2018.

[98] A. Belaout, F. Krim, B. Talbi, H. Feroura, A. Laib, S. Bouyahia, et A. Arabi, « Development of real time emulator for control and diagnosis purpose of Photovoltaic Generator », in 2017 6th International Conference on Systems and Control (ICSC), May 2017, pp. 139–144. doi: 10.1109/ICoSC.2017.7955940.

[99] A. Belaout, F. Krim, A. Mellit, B. Talbi, et A. Arabi, "Multiclass adaptive neuro-fuzzy classifier and feature selection techniques for photovoltaic array fault detection and classification," Renewable Energy, vol. 127, pp. 548–558, 2018, doi: 10.1016/j.renene.2018.05.083.

[100]A. Belaout, "Application of Artificial Intelligence Techniques for Faults Diagnosis in Photovoltaic Systems," Ph.D. dissertation, Dept. Electron., Ferhat

Abbas Sétif 1 University, Sétif, Algeria, 2018.

[101] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. Advances in Neural Information Processing Systems, 25, 2951-2959. https://doi.org/10.5555/2999134.2999301

[102] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & De Freitas, N. (2015). Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE, 104(1), 148-175. https://doi.org/10.1109/JPROC.2015.2494218

[103] Wu, J., Chen, X. Y., Zhang, H., Xiong, L. D., Lei, H., & Deng, S. H. (2019). Hyperparameter optimization for machine learning models based on Bayesian optimization. Journal of Electronic Science and Technology, 17(1), 26-40. https://doi.org/10.11989/JEST.1674-862X.707157

[104] M. Gheisari et al., "Deep learning: Applications, architectures, models, tools, and frameworks: A comprehensive survey," CAAI Trans. Intell. Technol., vol. 9, no. 5, pp. 835-862, Oct. 2024, doi: 10.1049/cit2.12180.

[105] M. Bocquel and J. Schoukens, “Nonlinear state-space modeling of systems with nonlinear feedback,” Automatica, vol. 48, no. 5, pp. 873–878, May 2012, doi: 10.1016/j.automatica.2012.02.015.

[106] S. Khan, M. H. F. Zarandy, and T. Bányász, “Computationally efficient predictive control based on ANN state-space models,” arXiv preprint arXiv:2303.17305, Mar. 2023. [Online]. Available: https://arxiv.org/abs/2303.17305

[107] Y. Gaaloul, O. Bel Hadj Brahim Kechiche, H. Oudira, et al., "Faults Detection and Diagnosis of a Large-Scale PV System by Analyzing Power Losses and Electric Indicators...," Energies, vol. 18, no. 10, p. 2482, 2025.

[108] A. F. Amiri, S. Kichou, H. Oudira, et al., "Fault detection and diagnosis of a photovoltaic system based on deep learning using the combination of a CNN and Bi-GRU," Sustainability, vol. 16, no. 3, p. 1012, 2024.

[109] M. Alrifaey, W. H. Lim, C. K. Ang, et al., "Hybrid deep learning model for fault detection and classification of grid-connected photovoltaic system," IEEE Access, vol. 10, pp. 13852-13869, 2022.

[110] V. Veerasamy, N. I. A. Wahab, M. L. Othman, et al., "LSTM recurrent neural network classifier for high impedance fault detection in solar PV integrated power system," IEEE Access, vol. 9, pp. 32672-32687, 2021.

[111] S. Sairam, S. Srinivasan, G. Marafioti, et al., "Explainable incipient fault detection systems for photovoltaic panels," arXiv preprint arXiv:2011.09843, 2020.

[112] B. Teta, D. Korich, N. Bakria, et al., "Fault detection and diagnosis of grid-connected photovoltaic systems using energy valley optimizer based lightweight CNN and wavelet transform," Scientific Reports, vol. 14, p. 18907, 2024.

[113] J. Goldberger, G. E. Hinton, S. Roweis, and R. R. Salakhutdinov, "Neighbourhood components analysis," in Proc. Advances in Neural Information

Processing Systems (NIPS), 2004, vol. 17.

[114] K. Song, J. Han, G. Cheng, J. Lu, and F. Nie, "Adaptive neighborhood metric learning," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 9, pp. 4591-4604, Sept. 2022.