Master Diploma Thesis

≪Recurrent Fuzzy Neural Networks for Time-Series Prediction≫

Supervisor: Mastorocostas Paris

Christodoulides Anastasios
(Registration Number: aivc22019)

Master of Science, University of West Attica in cooperation with University of Limoges

ATHENS, 2025

Μέλη Εξεταστικής Επιτροπής συμπεριλαμβανομένου του εισηγητή

Η διπλωματική εργασία εξετάστηκε επιτυχώς από την κάτωθι Εξεταστική Επιτροπή

Α/Α	ΟΝΟΜΑΤΕΠΩΝΥΜΟ	ΒΑΘΜΙΔΑ/ΙΔΙΟΤΗΤΑ	ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ
1	Νικόλαος Βασιλάς	Καθηγητής	NIKOLAOS VASILAS Digitally signed by NIKOLAOS VASILAS Date: 2025.03.05 16:09:54 +02'00'
2	Πάρις Μαστοροκώστας	Καθηγητής	PARIS MASTOROKOSTAS Digitally signed by PARIS MASTOROKOSTAS Date: 2025.03.05 12:50:48 +02'00'
3	Παναγιώτα Τσελέντη	ΕΔΙΠ	Panagiota Tselenti Digitally signed by Panagiota Tselenti Date: 2025.03.05 12:52:57 +02'00'

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ/ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

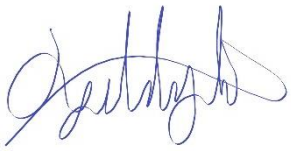
Ο κάτωθι υπογεγραμμένος Χριστοδουλίδης Αναστάσιος του Παντελάκη, με αριθμό μητρώου 22019 μεταπτυχιακός φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της πτυχιακής/διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών,

Χριστοδουλίδης Αναστάσιος



Abstract

Time series forecasting is a fundamental challenge in many scientific and industrial domains, especially when dealing with chaotic and highly nonlinear systems. Traditional forecasting models, including Artificial Neural Networks (ANNs) and statistical methods, often struggle to capture the complex temporal dependencies and uncertainty inherent in these datasets. To address these limitations, hybrid approaches that integrate fuzzy logic and Recurrent Neural Networks (RNNs) have emerged as promising alternatives. This thesis presents a comparative study of two hybrid models: the Multi-Functional Recurrent Fuzzy Neural Network (MFRFNN) and the Recurrent Neurofuzzy System ReNFuzz-LF. The effectiveness of these models is evaluated across multiple datasets, including Electric Loads, Lorenz chaotic system, Box–Jenkins Gas Furnace, Wind Speed Prediction, Google Stock Price Prediction and Air Quality Index (AQI).

Contents

List of Figures

List of Tables

1 Introduction

The ability to accurately forecast time series data is critical across various fields, including finance, energy, meteorology, and environmental science. Forecasting accuracy is a critical aspect in decision-making processes, impacting industries such as financial markets, energy grid management, and air quality monitoring. Improved prediction models allow businesses and policymakers to make informed decisions, minimize risks, and optimize resource allocation.

However, real-world time series often exhibit chaotic behaviour, nonlinearity, and noise, making traditional forecasting techniques inadequate. Classical models such as autoregressive integrated moving average (ARIMA) and standard artificial neural networks (ANNs) struggle when faced with nonlinear and multi-scale dependencies in time series data. RNNs, particularly LSTMs, improve sequential data modelling but still face challenges in complex systems. Overfitting presents a significant challenge for RNNs, as they may capture and retain noise in the data rather than learning meaningful patterns. Their sequential processing structure also leads to increased computational demands, making training both resource-intensive and time-consuming [1].

To address these challenges, integrating fuzzy logic with recurrent neural networks helps improve generalization by introducing rule-based representations that captures underlying trends more effectively. Such hybrid models have gained traction in recent years showing promising results. By combining the interpretability of fuzzy systems, the adaptive learning capabilities of neural networks and the memory mechanisms of recurrent architectures, these models enable for more accurate and robust forecasting.

This thesis explores two hybrid models designed for time series forecasting, MFRFNN and ReNFuzz-LF. MFRFNN[2] employs a dual-network structure where a fuzzy neural network (FNN) predicts the system's output, while a secondary FNN determines the system's state, allowing the system to learn multiple functions simultaneously. This makes it well-suited for datasets where the same input may lead to different outputs depending on the system state. On the other hand, ReNFuzz-LF[3] utilizes a rule-based fuzzy system with local RNN consequents, enabling it to adapt dynamically to short-term fluctuations without the complexity of a global feedback loop. Both models are assessed on diverse datasets, ranging from chaotic synthetic data (Lorenz system) to real-world applications such as stock price forecasting and AQI prediction.

1.1 Objectives

The primary objectives of this thesis are as follows:

1. Analysis of MFRFNN and ReNFuzz-LF models: A detailed examination of the structural differences between MFRFNN and ReNFuzz-LF, highlighting their respective mechanisms in handling nonlinear time series forecasting.
2. Implementation of ReNFuzz-LF in MATLAB: The development of ReNFuzz-LF in MATLAB.
3. Application to benchmark time series problems: Testing both models on well-established time series forecasting benchmarks, including the Lorenz system, Electric Load, Box–Jenkins Gas Furnace, Wind Speed Prediction, Google Stock Price Prediction, and Air Quality Index Prediction to evaluate their performance under different conditions.

2. Literature Review

2.1 Artificial Intelligence

According to Elaine Rich[4], "Artificial Intelligence (A.I.) is the study of how to make computers do things that people are better at". Many more definitions were given over the years but a common theme is that AI is expected to imitate human intelligence or carry out activities that previously necessitated human intervention.

AI is an umbrella term that encompasses a wide range of technologies and methodologies. These include areas such as machine learning, natural language processing, computer vision, robotics and expert systems. To build these intelligent systems, computational models, algorithms, and statistical techniques are employed that can analyse data, recognize patterns, learn from experiences, make informed decisions, comprehend natural language, and adapt to evolving conditions.

2.1.1 A Brief History of Artificial Intelligence

The roots of Artificial Intelligence stretch back to Aristotle (384-322 BCE), where his informal system of syllogisms was among the first recorded attempts to codify rational thought into systematic rules.

This early groundwork set the stage for centuries of exploration into reasoning and the mechanics of thought. In 1642, Blaise Pascal (1623-1662) built the Pascaline, a mechanical calculator and wrote that it “produces effects which appear nearer to thought than all the actions of animals”. Furthermore, Thomas Hobbes (1588-1679) suggested the idea of a thinking machine and that reasoning was like numerical computations.

The 20th century marked a shift from mechanical tools to computational theories. Alan Turing introduced the Turing Test, a criterion for determining a machine's capacity for intelligent behaviour. During World War II, Turing’s work on The Bombe, an electro-mechanical device used by British cryptologists to help decipher German Enigma-machine-encrypted secret messages during World War II, demonstrated the potential of machines to process complex information, a concept that underpins much of AI today.

AI was formally recognized as an academic discipline at the 1956 Dartmouth Conference[5], led by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. This pivotal event introduced the idea that machines could simulate human cognitive

functions. Early breakthroughs included programs like the Logic Theorist, capable of solving mathematical proofs, and the General Problem Solver, which demonstrated logical reasoning in simplified domains. However, these systems faced challenges when applied to real-world problems, leading to periods of reduced funding and research interest, referred to as "AI Winters".

The resurgence of AI in the late 20th century was driven by advancements in computational power, the emergence of neural networks, and the development of machine learning. Today, AI technologies are deeply integrated into our lives, influencing fields such as healthcare, finance, transportation, and entertainment.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a subfield of machine learning, inspired by the structure and function of the human brain and biological neural networks. The primary objective of neural networks is to identify patterns within data and enhance predictive performance through various optimization techniques. These networks are widely used for classification, pattern recognition, and sequential data processing.

2.2.1 Structure and Types of Artificial Neural Networks

Artificial Neural Networks consist of multiple layers of artificial neurons, weighted connections and activation functions such as sigmoid, tanh or ReLu are used to determine the degree of the neurons activation and also introduce non-linearity. Layers are structured into three types:

- input layer, which receives the input data
- hidden layers, which transform the data through the weighted connections of the network
- output layer, which produces the final result.

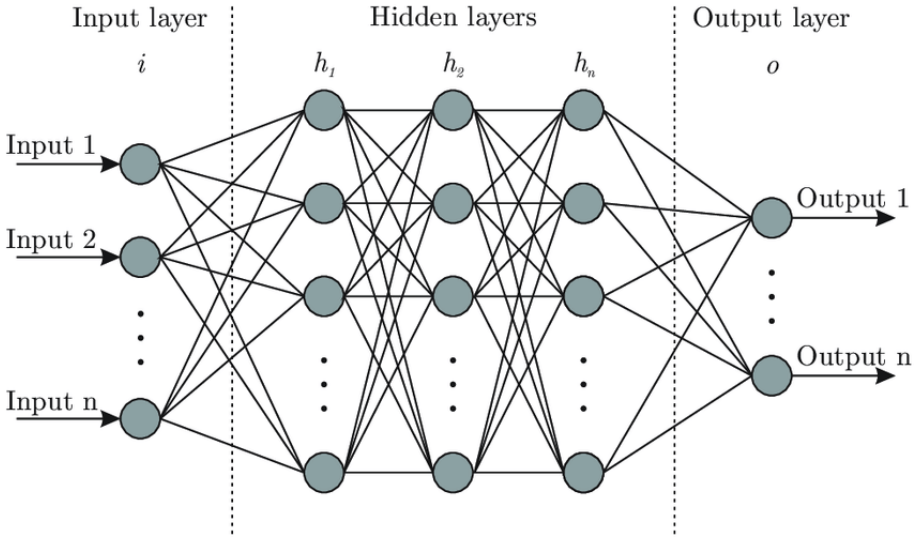


Figure 1: Artificial Neural Network Architecture

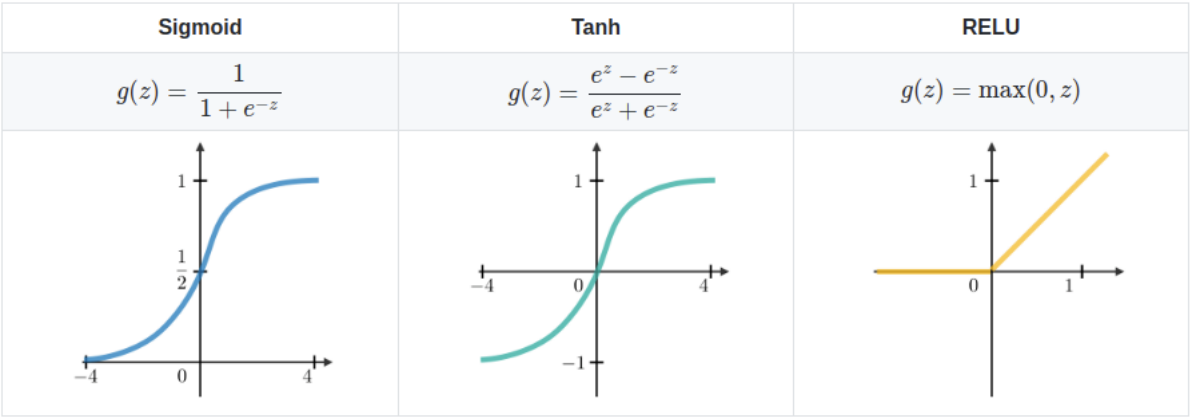


Figure 2: Activation Functions

There are different types of neural networks, each used for different tasks. Feedforward neural networks (FNNs) process data in one direction without loops, making them effective for tasks like classification. Convolutional neural networks (CNNs) are designed for spatial data, particularly in image processing, by using convolutional layers to extract spatial features and Recurrent neural networks (RNNs) are better suited for sequential data, allowing information from previous outputs to influence future predictions, making them ideal for tasks like speech recognition and time-series forecasting.

2.2.2 Training Process of Neural Networks

Training a neural network involves adjusting the weights of connections to minimize errors and improve performance. The process follows these main steps:

- Forward Propagation: Input data passes through the network layers, with each neuron computing a weighted sum of its inputs and applying an activation function.
- Loss Calculation: The difference between the predicted output and actual target value is measured using a loss function. Common loss functions include Mean Squared Error (MSE) for regression tasks or Cross-Entropy for classification tasks.
- Backpropagation: Backpropagation is a computational method that uses the chain rule to compute the gradients of the loss function with respect to each weight, propagating errors backward from the output layer to the input layer. This allows the network to compute how each weight should be adjusted.
- Gradient Descent Optimization: Gradient descent is an optimization algorithm used to minimize the loss function by updating the network's weights in the direction of steepest descent. It iteratively adjusts the parameters using a learning rate to determine the step size for weight updates.
- Iteration and Convergence: The process repeats for multiple epochs until the network converges, meaning the loss stops decreasing significantly. Regularization techniques such as dropout and L2 regularization help prevent overfitting by reducing model complexity.

2.2.3 Recurrent Neural Networks (RNNs) and Challenges

Recurrent Neural Networks (RNNs)[6] introduce a memory mechanism that allows them to retain information from previous inputs. Unlike feedforward networks, which treat each input independently, RNNs incorporate loops within their architecture, enabling them to maintain information over sequences.

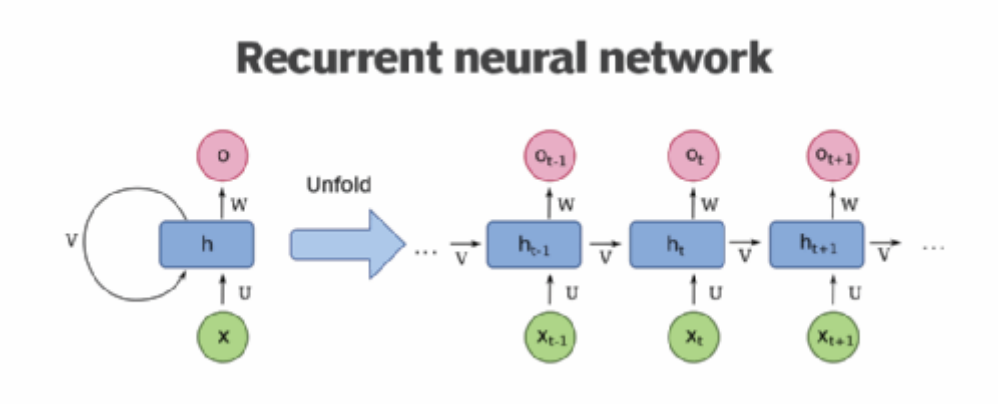


Figure 3: RNN Structure. The bottom is the input state; middle, the hidden state; top, the output state. U, V, W are the weights of the network.

Training RNNs involves a technique called Backpropagation Through Time (BPTT)[7], an extension of traditional backpropagation that adjusts weights by unrolling the network through past time steps. However, standard RNNs struggle with long-range dependencies due to the vanishing gradient problem, where gradients diminish exponentially as they are propagated backward, making learning difficult over long sequences[1].

To address the vanishing gradient issue, Long Short-Term Memory Networks (LSTMs)[8] were introduced. LSTMs use memory cells and gates (input, forget, and output) to regulate the information passing through the network. This structure allows LSTMs to retain or discard information as necessary. This makes them very effective for tasks requiring long-term dependencies such as speech recognition and machine translation. Furthermore, they are used in time-series prediction, where past data trends influence future outcomes.

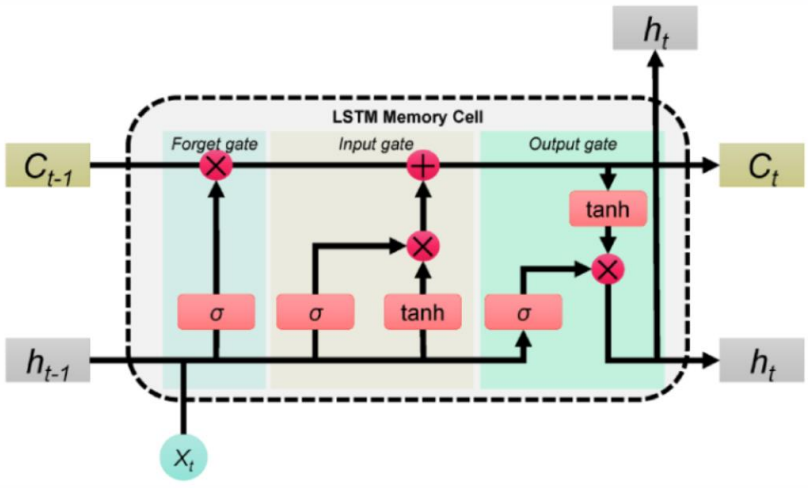


Figure 4: LSTM Structure

While RNNs, particularly LSTMs, improve sequential data modelling, they still face challenges in complex nonlinear systems. Overfitting is a major issue, as RNNs can memorize noise in data instead of generalizing patterns. Computational complexity also increases due to the sequential nature of RNNs, making them computationally expensive to train. Additionally, chaotic data poses difficulties, as RNNs may struggle to maintain stability and predictability when dealing with highly unpredictable sequences.

To address these challenges, particularly in complex nonlinear systems, hybrid fuzzy models offer a powerful solution by integrating the strengths of Fuzzy Logic and Recurrent Neural Networks (RNNs). These models enhance interpretability and robustness by leveraging fuzzy rules to handle uncertainty and imprecision, making them particularly effective in chaotic environments.

2.3 Fuzzy Systems

Fuzzy systems[9] are a subset of computational intelligence methodologies that aim to emulate human reasoning and decision-making processes using fuzzy set theory and fuzzy logic. They are particularly useful when faced with imprecise or uncertain information and are widely applied in various domains such as control systems, pattern recognition, decision-making, and data analysis.

2.3.1 Fuzzy Set Theory to Fuzzy Logic

In traditional mathematical logic, classical set theory categorizes elements in a binary manner, an element either belongs to a set or it does not. However, real-world data is often imprecise. To address this limitation, Lotfi A. Zadeh introduced Fuzzy Set Theory[10] in 1965, which extends classical set theory by allowing elements to have partial membership, meaning an element can belong to a set to a certain degree between 0 and 1. This advancement enables a more nuanced approach to modeling uncertain or linguistically described data, such as "tall people," "warm temperatures," or "high risk."

Fuzzy set theory provided the mathematical foundation for fuzzy logic by enabling reasoning under uncertainty. Fuzzy Logic[11] is a mathematical framework for reasoning under uncertainty, introduced by Lotfi A. Zadeh in 1973 as an extension of classical Boolean logic. Unlike traditional binary logic, which operates on strict true or false values, fuzzy logic allows for degrees of truth that range between 0 and 1. This makes it a multi-valued logic

system capable of handling vagueness and ambiguity, allowing for more flexible decision-making processes.

The motivation behind fuzzy logic arises from the fact that real-world concepts are often not sharply defined. Many everyday decisions are based on approximate reasoning rather than strict binary classifications. For instance, linguistic terms such as "hot," "cold," "tall," or "fast" do not have rigid boundaries but instead exhibit gradual transitions. Traditional logic fails to effectively model such imprecise concepts, whereas fuzzy logic provides a structured approach to quantify and process these uncertainties.

2.3.1.1 Crisp Sets vs. Fuzzy Sets

In classical set theory, a crisp set A is defined such that each element x in the universal set U either belongs to A ($x \in A$, membership value 1) or does not ($x \notin A$, membership value 0). Mathematically, this is expressed using an indicator function:

$$\mu_A(x) = \begin{cases} 1, x \in A \\ 0, x \notin A \end{cases}$$

However, in many real-world situations, classification is not absolute. For example, if we define a crisp set "Tall People" as those taller than 180 cm, a person who is 179 cm is not considered tall, even though the difference is negligible.

To address this limitation, fuzzy sets introduce partial membership, allowing elements to belong to a set to a certain degree between 0 and 1. This is achieved using a continuous membership function $\mu_A(x)$, such that:

$$0 \leq \mu_A(x) \leq 1$$

For example, in a fuzzy set "Tall People", someone who is 190 cm might have a membership of 0.9, while someone 170 cm might have a membership of 0.4. Instead of a strict boundary, the transition between not tall and tall is smooth.

2.3.1.2 Crisp Logic vs. Fuzzy Logic

The distinction between crisp sets and fuzzy sets extends naturally into logical reasoning. In crisp logic (Boolean logic), every statement must be either true (1) or false (0). This is based on the Principle of Bivalence, which states that there are only two possible truth values.

For example, consider a rule in classical logic:

- Crisp Logic Statement: *"If temperature is greater than 30°C, then it is hot."*
- Boolean evaluation:
    - If temperature = 31°C, the statement is true (1).
    - If temperature = 29°C, the statement is false (0).

This rigid classification ignores the fact that hotness is a gradual concept, and a temperature of 29.9°C is not significantly different from 30.1°C.

In contrast, fuzzy logic allows for partial truth values, enabling more nuanced decision-making. Instead of forcing a binary classification, a fuzzy rule would define "hot" as a continuous function of temperature, where 30°C is not an absolute cutoff but rather part of a smooth transition:

- Fuzzy Logic Statement: *"If temperature is around 30°C, then it is somewhat hot."*
- Fuzzy evaluation using a membership function:
    - 28°C → Membership 0.6 (*somewhat hot*)
    - 30°C → Membership 0.8 (*fairly hot*)
    - 35°C → Membership 1.0 (*definitely hot*)

Instead of a step function, fuzzy logic uses smooth functions to represent truth values, ensuring gradual transitions rather than abrupt jumps.

A graphical representation of this difference can be visualized through a membership function, where truth values change gradually rather than in discrete steps. Instead of a step-like function that jumps from 0 to 1 at a threshold point, a fuzzy membership function smoothly increases from cooler to hotter temperatures.

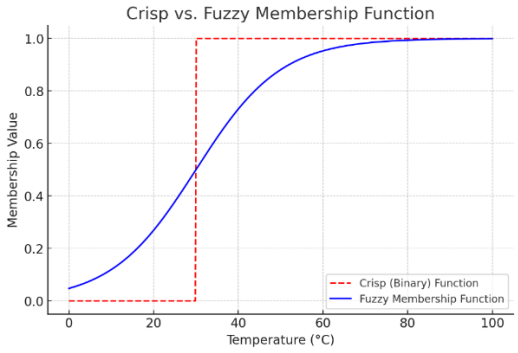


Figure 5: Crisp vs. Fuzzy Logic

2.3.1.3 Membership Functions

A membership function (MF) assigns a value between 0 and 1 to each element, representing its degree of belonging to a fuzzy set. The choice of membership function depends on the application and domain expertise. Below are some of the most commonly used membership functions, along with their mathematical formulas and explanations.

Triangular Membership Function (TriMF)

The triangular membership function is one of the simplest and most widely used fuzzy membership functions. It is defined by three parameters: a (left endpoint), b (peak point), and c (right endpoint), forming a triangle-shaped function.

$$\mu_{Tri}(x) = \begin{cases} 0, & x \leq a \text{ or } x \geq c \\ \dfrac{x-a}{b-a}, & a \leq x \leq b \\ \dfrac{c-x}{c-b}, & b \leq x \leq c \end{cases}$$

Explanation:

- The membership value is 0 outside the range $[a, c]$.
- The function increases linearly from a to b, reaching a maximum of 1 at b.
- The function decreases linearly from b to c, dropping back to 0 at c.

Example: Defining "moderate temperature" between 15°C and 30°C, with peak membership at 22°C. $(a = 15, b = 22, c = 30)$

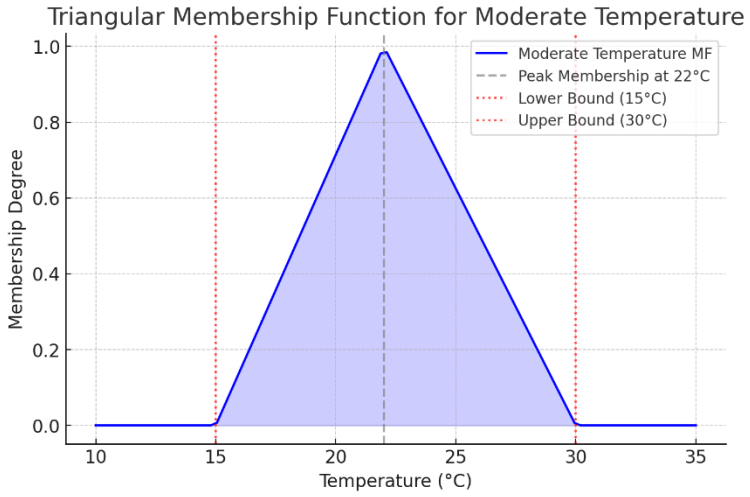


Figure 6: Triangular Membership Function

Trapezoidal Membership Function (TrapMF)

The trapezoidal membership function is an extension of the triangular function but allows a flat top instead of a single peak. It is defined by four parameters: a (left endpoint), b (start of plateau), c (end of plateau), and d (right endpoint).

$$\mu_{Trap}(x) = \begin{cases} 0, & x \leq a \text{ or } x \geq d \\ \dfrac{x-a}{b-a}, & a \leq x \leq b \\ 1, & b \leq x \leq c \\ \dfrac{d-x}{d-c}, & c \leq x \leq d \end{cases}$$

Explanation:

- The function is 0 outside the range $[a, d]$.
- The function increases linearly from a to b.
- It remains constant at 1 between b and c, forming a plateau.
- It decreases linearly from c to d.

Example**:** Defining "warm temperature" where 20°C to 25°C are fully considered warm, while values between 15°C and 30°C are partially warm. $(a = 15, b = 20, c = 25, d = 30)$

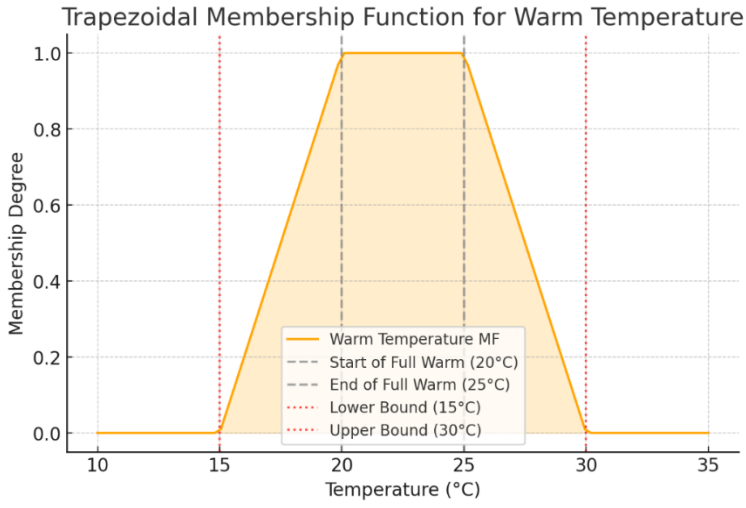


Figure 7: Trapezoidal Membership Function

Gaussian Membership Function (GaussMF)

The Gaussian membership function is a smooth, bell-shaped function defined by two parameters: c (mean or centre of the curve) and σ (standard deviation, controlling the width of the bell curve).

$$\mu_{Gauss}(x) = e^{-\frac{(x-c)^2}{2\sigma^2}}$$

Explanation:

- The function is always positive and symmetric around c.
- The peak (maximum membership) is 1 at c, and values decrease smoothly on both sides.
- The parameter σ controls the spread. A larger σ creates a wider function, while a smaller σ makes it narrower.

Example: Defining "medium speed" for a car, where the most confident medium speed is 50 km/h, but values between 40 km/h and 60 km/h also belong to the medium category with lower degrees of membership. $(c = 50, \sigma = 7)$

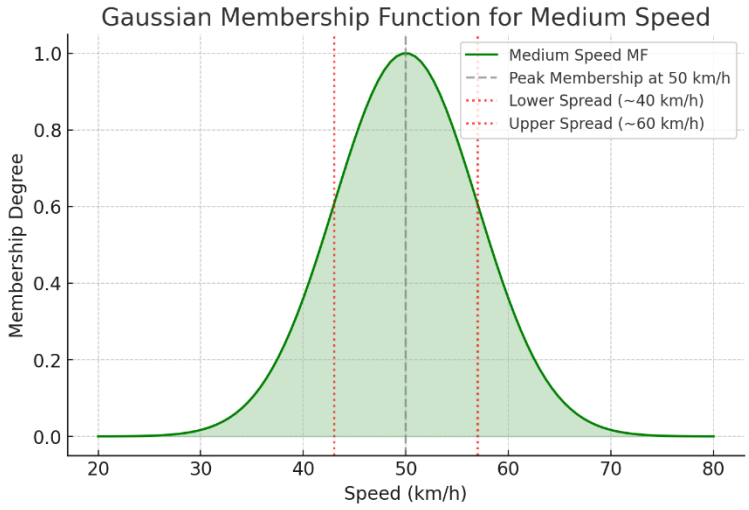


Figure 8: Gaussian Membership Function

Generalized Bell Membership Function (BellMF)

The generalized bell membership function provides more flexibility than the Gaussian MF by introducing an additional parameter. It is defined by three parameters: a (width), b (slope), and c (center or peak point).

$$\mu_{Bell}(x) = \frac{1}{1 + \left| \frac{x - c}{a} \right|^{2b}}$$

Explanation:

- The parameter a controls the width of the function.
- The parameter b determines the slope or sharpness of the function.
- The parameter c sets the center of the bell curve.

Example: Defining "moderate pressure" in an industrial system, where a soft transition is needed. $(a = 15,\ b = 3,\ c = 50)$

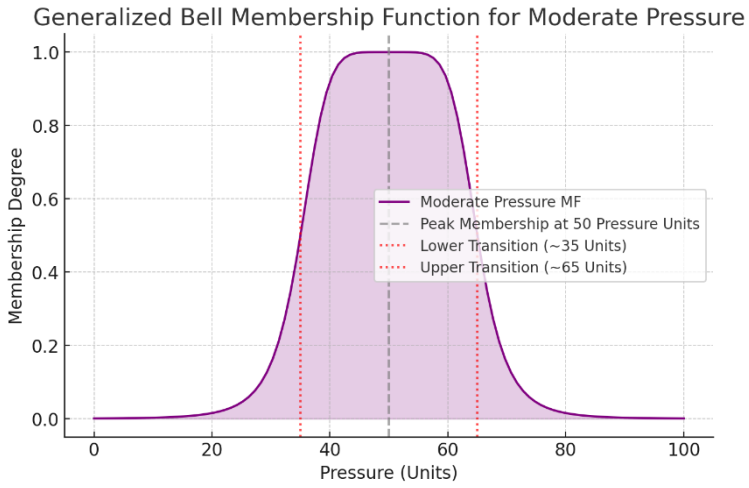


Figure 9: Generalized Bell Membership Function

2.3.1.4 Fuzzy Set Operations

Fuzzy sets generalize classical set operations using t-norms (AND), s-norms (OR), and negation (NOT):

- Union (OR operation): $\mu_{A \cup B}(x) = \max\big(\mu_A(x), \mu_B(x)\big)$.
- Intersection (AND operation): $\mu_{A \cap B}(x) = \min\big(\mu_A(x), \mu_B(x)\big)$.
- Complement (NOT operation): $\mu_{\neg A}(x) = 1 - \mu_A(x)$.

2.3.1.5 Advantages

One of the most significant advantages of fuzzy logic is its ability to handle gradual transitions rather than enforcing strict categorizations. Another major advantage is that fuzzy logic closely mimics human reasoning, which often relies on linguistic descriptions rather than numerical precision, translating qualitative descriptions into a quantitative framework, enabling machines to reason in a way similar to humans.

Fuzzy logic is also highly robust in uncertain environments, making it effective for dealing with noisy, incomplete, or imprecise data. Unlike traditional mathematical models that require exact equations, fuzzy logic systems can work with approximate rules and still produce meaningful outputs. Instead of relying on complex equations, fuzzy logic systems can be built using simple IF-THEN rules that encode expert knowledge. For example, in an automatic braking system, a rule might state: "IF distance to obstacle is small AND speed is high, THEN apply brakes strongly." Such rules are easy to understand and modify, making fuzzy logic an accessible and interpretable approach to system design. Additionally, its flexibility and ease of implementation have led to widespread adoption across multiple disciplines. As a result, fuzzy logic has cemented its role in the development of intelligent systems that can process and respond to real-world uncertainty in a sophisticated manner.

2.3.2 Fuzzy Inference Systems (FIS)

A Fuzzy Inference System (FIS) is a computational framework that applies fuzzy logic to map inputs to outputs using a set of fuzzy rules. It serves as the core reasoning mechanism in fuzzy logic-based decision-making and control systems

2.3.2.1 Core Principles of Fuzzy Systems

At the heart of fuzzy systems lie fuzzy sets, linguistic variables, and membership functions.

A fuzzy set is a collection of elements with varying degrees of membership, defined through a membership function. These sets can be discrete or continuous, depending on the nature of the problem. For instance, a fuzzy set describing "closeness to Vienna" for different cities, can be characterized using discrete membership values. Vienna itself has a membership value of 1, representing absolute closeness, while cities like Bratislava, Budapest, Berlin, and

Moscow gradually decrease in membership as the distance increases. Canberra, on the other hand, may have a membership value of zero, as it is extremely far away. In contrast, continuous fuzzy sets describe gradual transitions between categories. An example of this can be seen in temperature levels: a temperature of 5°C might fully belong to the set "Cold," while at 20°C it transitions toward "Warm" and further evolves to "Hot" as it approaches 35°C.

Linguistic variables add another layer of abstraction to fuzzy logic. Instead of crisp numerical values, they use qualitative terms, such as "hot," "cold," "young," or "old," to represent concepts in a way that aligns with human intuition. For example, temperature can be described qualitatively rather than numerically, enhancing the system's ability to interpret ambiguous or subjective information.

Membership functions, which lie at the core of fuzzy logic systems, define the relationship between input values and their corresponding degrees of membership in fuzzy sets. For any input value within a given universe of discourse, the membership function assigns a degree of membership, ranging from 0 (no membership) to 1 (full membership). As mentioned above these functions enable fuzzy systems to handle complex, nonlinear, and vague scenarios effectively.

Finally, the strength of fuzzy systems lies in their ability to apply fuzzy rules, expressed as intuitive "IF-THEN" statements. These rules provide the mechanism through which inputs are transformed into outputs, and by combining multiple of them, fuzzy systems can model intricate relationships and support complex decision-making processes.

2.3.2.2 Fuzzy Inference Process

The process of implementing a fuzzy system typically involves fuzzification, rule evaluation, aggregation, and defuzzification.

In the fuzzification stage, crisp numerical inputs are converted into fuzzy values using membership functions. This transformation allows the system to handle degrees of truth rather than binary classifications.

Once fuzzification is complete, the rule evaluation stage processes the fuzzy inputs using a predefined set of IF-THEN rules to establish relationships between input and output variables. This involves calculating the firing strength of each rule, which determines how

strongly the rule is activated based on the input membership values. The firing strength is computed using fuzzy logic operators, such as AND, OR, and NOT, and is used to adjust the degree of membership of the rule's output, ensuring that the fuzzy inference process accurately reflects the influence of multiple overlapping rules.

Following rule evaluation, the aggregation stage merges the outputs of all activated fuzzy rules into a unified fuzzy output set. Since multiple rules may contribute overlapping values to the same output variable, this step ensures that every relevant rule influences the final result based on its firing strength. The system combines these outputs using mathematical aggregation techniques, such as maximum or sum operations, to construct a single fuzzy set that accurately represents the collective decision-making process before defuzzification.

Finally, in the defuzzification stage, the fuzzy output is converted back into a crisp numerical value suitable for real-world applications. Since fuzzy logic operates on linguistic and continuous truth values, a final crisp decision is required for actions such as adjusting fan speed or making a classification decision. Several defuzzification methods exist, each with its strengths and applications:

- The Centroid of Area (COA) method is one of the most widely used, as it computes the centre of gravity of the fuzzy output distribution and provides smooth, stable, and intuitive defuzzification.

  $$x^* = \frac{\int x \cdot \mu(x)\, dx}{\int \mu(x)\ dx}$$

  Where x^* is the crisp defuzzified output, x represents the variable (e.g., temperature, speed, pressure, etc.) and $\mu(x)$ is the membership function of the fuzzy output.

- The Center of Sums (COS), which unlike COA who integrates over the entire fuzzy area, it only considers the peak values of the output membership functions.

  $$x^* = \frac{\sum_{i=1}^{n} c_i \cdot \max\left(\mu_i(x)\right)}{\sum_{i=1}^{n} \max\left(\mu_i(x)\right)}$$

Where x^* is the crisp defuzzified output, c_i is the center (typically mean) of the i^{th} fuzzy output set, $\max(\mu_i(x))$ is the peak value (maximum membership degree) of the i^{th} fuzzy set and n is the number of active fuzzy rules.

- The Center-Averaged Defuzzifier only considers the centers of the fuzzy rules. This method is primarily used in TSK fuzzy inference systems, where the output of each rule is a constant or linear function, making it much more efficient than the Centroid of Area (COA) approach.

$$x^* = \frac{\sum_{i=1}^{n} w_i \, c_i}{\sum_{i=1}^{n} w_i}$$

Where x^* is the crisp defuzzified output, c_i is the center (typically mean) of the i^{th} fuzzy set, w_i is the weight (firing strength) of the i^{th} fuzzy rule and n is the number of active fuzzy rules.

2.3.2.3 Types of Fuzzy Inference Systems

Fuzzy inference systems are broadly classified into two major types: the Mamdani Fuzzy Inference System (Mamdani FIS) and the Takagi-Sugeno-Kang (TSK) Fuzzy Inference System. Each type has distinct characteristics, advantages, and application domains, making them suitable for different tasks.

The Mamdani FIS[12], introduced by Ebrahim Mamdani in 1974, was the first fuzzy inference model applied in control systems. It uses fuzzy sets for both input and output variables and applies rule-based reasoning. This system is particularly well-suited for control applications, such as HVAC systems, washing machines, and industrial automation, where expert-defined rules govern system behaviour. Its linguistic rule structure makes it highly interpretable, allowing human experts to design rules in an intuitive, human-like manner.

In contrast, the Takagi-Sugeno-Kang (TSK) FIS[13], developed in 1985 by Takagi and Sugeno, employs mathematical functions instead of fuzzy sets to represent outputs. Unlike Mamdani FIS, which produces a fuzzy output requiring defuzzification, TSK models use a weighted average of all rule outputs to generate crisp numerical values directly. Additionally, TSK models can incorporate linear and nonlinear functions, allowing them to model complex systems with higher accuracy than Mamdani FIS.

2.3.4 Modern Advancements: Neuro-Fuzzy Systems

Neuro-Fuzzy Systems represent a hybrid approach that integrates Neural Networks with Fuzzy Logic, combining the learning capabilities of neural networks with the interpretability of fuzzy inference systems. Unlike traditional fuzzy logic systems, which rely on manually defined rules, neuro-fuzzy systems can learn and optimize fuzzy rules automatically from data.

One of the most significant advancements in neuro-fuzzy modeling was the introduction of the Adaptive Neuro-Fuzzy Inference System (ANFIS)[14] by Jang in 1993. ANFIS was designed as an extension of the TSK model, incorporating artificial neural networks (ANNs) to enable the system to learn from data rather than relying solely on manually defined rules. By integrating machine learning techniques such as gradient descent and least squares estimation, ANFIS can optimize rule parameters and membership functions in a data-driven manner, making it highly adaptable to changing environments.

The introduction of ANFIS marked a significant milestone in the evolution of fuzzy systems, as it bridged the gap between interpretable fuzzy reasoning and adaptive machine learning models. Its ability to learn from numerical data while maintaining the structure of a fuzzy inference system made it particularly useful in complex, nonlinear problems such as time-series forecasting.

2.4 Time Series

A time series[15] is a sequence of observations recorded at regular time intervals, usually equally spaced. The data is collected over time to analyse patterns, trends, and relationships between observations to make predictions or gain insights into the behaviour of a process. Time series can have either a single variable, known as a univariate time series, or multiple variables, referred to as multivariate time series. Typical examples of time series data can be, the daily closing prices of a stock or daily temperatures. Time series are ubiquitous in many fields, such as economics, engineering, natural sciences, and finance.

The structure of a time series is usually influenced by four fundamental components (Trend, Cyclical, Seasonal, Irregular):

- The trend reflects the long-term progression of the series, indicating whether the values increase, decrease, or remain constant over time. For example, the growth of a population often has upward or downward trends.

- Seasonality, involves periodic fluctuations within a specific time frame, such as monthly or yearly. Seasonal changes are observed in retail sales, where demand increases during holidays, or temperature records, which rise during summer and fall during winter.
- Time series data may display cyclical behaviour, characterized by oscillations over extended periods due to broader factors such as economic cycles. For instance, a business cycle often goes through phases like prosperity, decline, depression, and recovery, which repeat over multiple years.
- Irregular variations or randomness arise from unpredictable influences such as wars or natural disasters, which do not follow any pattern.

2.4.1 Time-Series Analysis

The applications of time series analysis span across numerous fields[15]. In business and economics, time series are used to forecast sales and analyse stock prices. In finance, exchange rates, interest rates, and stock market behaviour are often modelled using time series techniques. Similarly, in scientific and engineering domains, time series play a vital role in climate studies, where temperature and rainfall trends are analysed over decades. In healthcare, time series analysis is applied to monitor patient vitals over time, detect disease outbreaks, or track the progression of epidemics[16]. Environmental studies, too, rely heavily on time series to observe pollution levels[17], atmospheric patterns, and natural resource consumption over time. Another application of time series analysis is in electric load forecasting, where it is used to predict electricity demand over different time horizons, ranging from short-term (hourly or daily) to long-term (monthly or yearly).

The process of time series analysis involves identifying patterns within the data to build suitable models that can describe the underlying dynamics of the series. This model is then used to predict future events based on the observed historical patterns.

Time series forecasting is particularly useful when the statistical relationships among successive observations are unclear. For example, historical trends in airline passenger counts can be used to anticipate demand in upcoming months. Two famous examples of time series include the weekly BP/USD exchange rate series, which illustrates how currency exchange rates fluctuated over a 13-year period, and the monthly international airline passenger dataset, which shows both seasonal variations and an overall upward trend in air travel from 1949 to

1960. Graphical representations of these datasets often reveal trends, cycles, and seasonal patterns that are essential for effective forecasting.

Figure 10: BP/USD exchange rate series

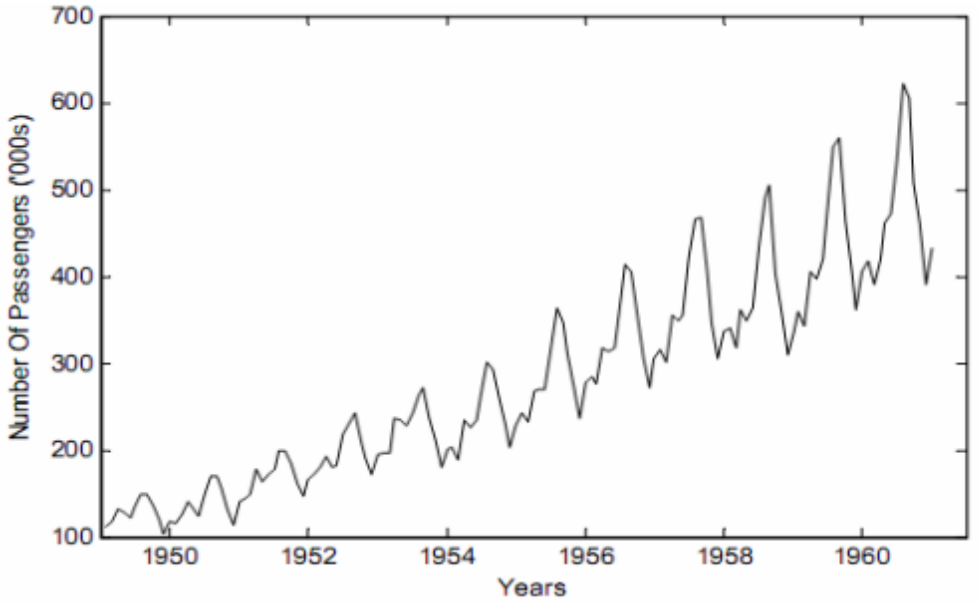


Figure 11: Monthly international airline passenger series

While many time series exhibit regular trends, cycles, and seasonality, some systems exhibit chaotic behaviour, where small variations in initial conditions lead to significantly different outcomes over time. These chaotic time series appear random but are governed by deterministic rules.

2.4.2 Chaos Theory

Chaos theory[18] is a branch of mathematics and science that studies systems that are highly sensitive to initial conditions, often described as the butterfly effect. The field originated in the mid-20th century with the work of researchers like Edward Lorenz, whose study of weather systems highlighted how minor changes in input data could produce significantly different forecasts. Chaos theory applies to various disciplines, including meteorology, physics, biology, economics, and engineering, where it helps explain the behaviour of complex, dynamic systems.

2.4.2.1 Chaotic Time Series and Nonlinear Dynamics

Unlike traditional time series, which often follow linear or periodic patterns, chaotic time series[19] emerge in highly dynamic and nonlinear systems. In these systems, small differences in the starting state can lead to vastly different outcomes, making long-term prediction extremely difficult or even impossible. Despite this apparent randomness, chaotic time series can reveal underlying patterns and structures.

One well-known example of a chaotic time series is the Mackey-Glass Chaotic Time Series, which arises from a delayed differential equation originally used to model the variation in the relative quantity of mature cells in the blood. The Mackey-Glass system demonstrates complex, nonlinear, and chaotic behaviour depending on its delay parameter.

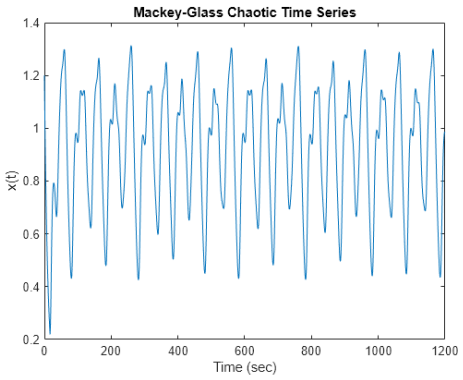


Figure 12: Mackey-Glass Chaotic Time Series

Traditional forecasting models often fail in chaotic systems because they assume stable and repeatable patterns in historical data. This limitation is particularly evident in various real-world applications, where nonlinear dependencies and unpredictable fluctuations challenge conventional prediction techniques. For example, predicting electricity consumption is difficult

because demand varies based on time of day, seasonality, and human activity patterns. Similarly, sales forecasting in retail is influenced by multiple factors such as holidays, and consumer behaviour, making it hard to model accurately. In temperature prediction, daily variations depend on local conditions, historical trends, and external influences, requiring models that can adapt to changing patterns. These complexities demonstrate the limitations of traditional forecasting techniques and highlight the need for more advanced, AI-driven approaches that can adapt to chaotic and dynamic environments more effectively. Hybrid models incorporating fuzzy logic and neural networks have shown promise in capturing the complexity of these datasets, improving forecasting accuracy.

2.5 Hybrid Algorithms

The prediction of chaotic time series, which exhibit highly non-linear and dynamic behaviour, remains a challenging task in system modeling. Existing models, such as Artificial Neural Networks (ANNs), Fuzzy Neural Networks (FNNs), and Recurrent Neural Networks (RNNs), have shown considerable promise but exhibit limitations when dealing with chaotic systems where multiple outcomes may depend on the system's state.

Several alternative methods have demonstrated promising performance in time series prediction, among which Recurrent Fuzzy Neural Networks (FNNs) stand out. RFNNs are hybrid models that integrate the learning capabilities of Recurrent Neural Networks (ANNs) with the interpretability and semantic transparency of fuzzy systems. Their ability to provide local representation and align with human reasoning makes them particularly effective in handling non-stochastic uncertainties. By capturing the underlying relationships within data, RFNNs have achieved significant success in time series prediction.

2.5.1 MFRFNN: Multi-functional Recurrent Fuzzy Neural Network

To leverage the temporal learning ability of Recurrent Neural Networks (RNNs) alongside FNNs' capacity to process fuzzy information, various Recurrent Fuzzy Neural Networks (RFNN) have been introduced but most of them are designed to learn only a single function. As a result, they generate a specific output based on current and previous inputs at each time step. However, when dealing with chaotic time series, where strong nonlinearity is present, a single-function approach is often insufficient.

For example, consider the return map of the Mackey-Glass chaotic time series. It shows that for a given value of x, two possible outputs (a and b) can emerge depending on the system's state. In such cases, an effective algorithm must simultaneously learn multiple functions (F1 and F2) and use system states to determine the appropriate output. If an algorithm only learns a single function, it will be unable to distinguish between possible outcomes at x, leading to reduced accuracy in time series prediction. Therefore, a network is required that can identify system states and learn a separate function for each state. This means the system should be capable of learning multiple functions concurrently.

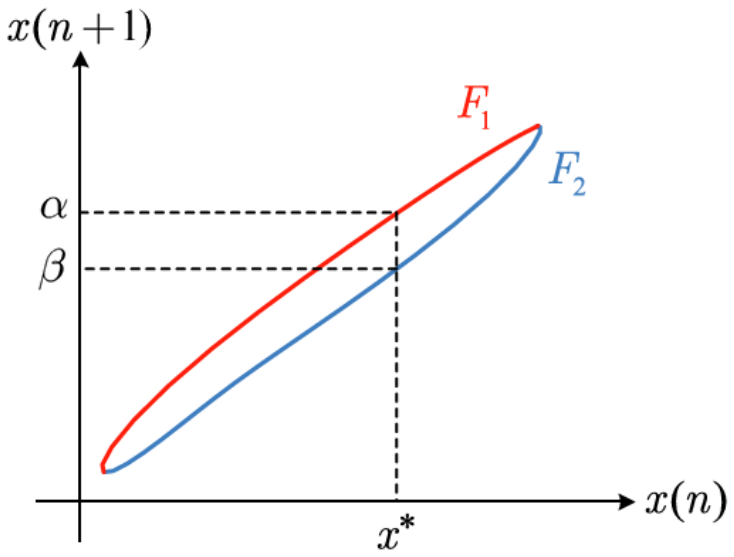


Figure 13: Return map of Mackey-Glass chaotic time series

Another key challenge in predicting chaotic time series is their high sensitivity to initial conditions, which results in long-term unpredictability. A network designed for long-term predictions must be able to learn system states dynamically to accurately capture the evolving behaviour of chaotic time series. Additionally, it must incorporate a feedback loop to retain historical information and make informed predictions. To overcome these limitations, the Multi-Functional Recurrent Fuzzy Neural Network (MFRFNN) can be used.

The Multi-Functional Recurrent Fuzzy Neural Network (MFRFNN)[2] is a hybrid model designed to handle complex time series forecasting problems, particularly those involving chaotic systems. It consists of two interconnected FNNs. One network predicts future values of the time series and the other determines the system's state. By maintaining a memory

of past states and using them through the feedback mechanism in the model, it helps overcome long-term dependency issues that many traditional models face. The most important part of MFRFNN, is its ability to learn multiple functions simultaneously by employing the system states, which is crucial for handling highly nonlinear and chaotic data

2.5.2 ReNFuzz-LF: A Recurrent Neurofuzzy System

ReNFuzz-LF[3] is a Recurrent Neurofuzzy System that was designed for short-term electric load forecasting and operates with a single input. Unlike traditional static fuzzy models, it features dynamic consequent parts that incorporate small-scale recurrent neural networks (RNNs). These networks possess local output feedback, which allows the system to learn the temporal dependencies of time-series data. Additionally, the training method used for ReNFuzz-LF, is Simulated Annealing Dynamic Resilient Propagation (SA-DRPROP)[20], which helps alleviate the disadvantages of standard gradient-based methods.

Based on the initial application of this system, its structure enabled the model to capture complex time dependencies in electricity demand, reduce the number of required inputs, simplifying the forecasting process and it improved prediction accuracy through the hybridization of fuzzy logic and neural networks. This system will be evaluated on various non-linear datasets in the next chapters to assess its adaptability and performance.

3 Methodology

3.1 Architecture of MFRFNN

The Multi-Functional Recurrent Fuzzy Neural Network (MFRFNN)[2] uses a state-based mechanism that switch between different states and allows the system to learn multiple functions simultaneously. This makes the system capable of modeling complex time-series data where an input may generate multiple different outputs based on the state of the network.

The MFRFNN consists of two fuzzy neural networks (FNNs) employing Takagi-Sugeno-Kang (TSK) fuzzy rules. One network generates the system's output, while the other determines the system's state. The system state network uses a feedback loop that enables it to retain historical information from the past states of the network. Additionally, the state signals are fed into the output network using a delay unit to calculate the final system output.

- N: number of states
- K_1 and K_2: number of fuzzy rules in output and state network
- $\mathbf{x} = \{x_1, x_2, \dots, x_d\}^T$: input vector
- $\hat{y}(t)$: system output at time step t
- $A_{i,j}$ and $B_{i,j}$: MFs for the j-th input variable x_j in the i-th rule of the output and state networks
- $\mu_{A_{i,j}}(x_j)$: membership value of the j-th input variable x_j on $A_{i,j}$
- $r_i(x)$ and $q_i(x)$: overall rule activation strength for the i-th rule in the output and state networks
- $\bar{r}_i(x)$ and $\bar{q}_i(x)$: normalized the firing strengths of $r_i(x)$ and $q_i(x)$
- F_j and G_j: approximated functions for the output and state network for the j-th state of the system
- w_{ij} and v_{ij}: link weights associated with the i-th rule in the output and state network, for the j-th state of the system
- W and V: link weight matrices for output and state networks
- s_j: state signal for the j-th state of the system
- $F(t)$: vector of approximated functions of the output network at time step t
- $S(t)$: vector of state signals at time step t
- $G(t)$: vector of approximated functions of the state network at time step t

- $R(t)$: vector of approximated functions of the output network at time step t
- $Q(t)$: vector of normalized firing strengths of the fuzzy rules at time t.
- $o(t)$: intermediate output of state's network output layer
- $\bar{o}(t)$: normalized $o(t)$ within the range $[1, N]$
- E_i: i-th membership function in the output layer of the state network
- $\mu_{E_i}(o(t))$: membership function value for $o(t)$ on the i-th state

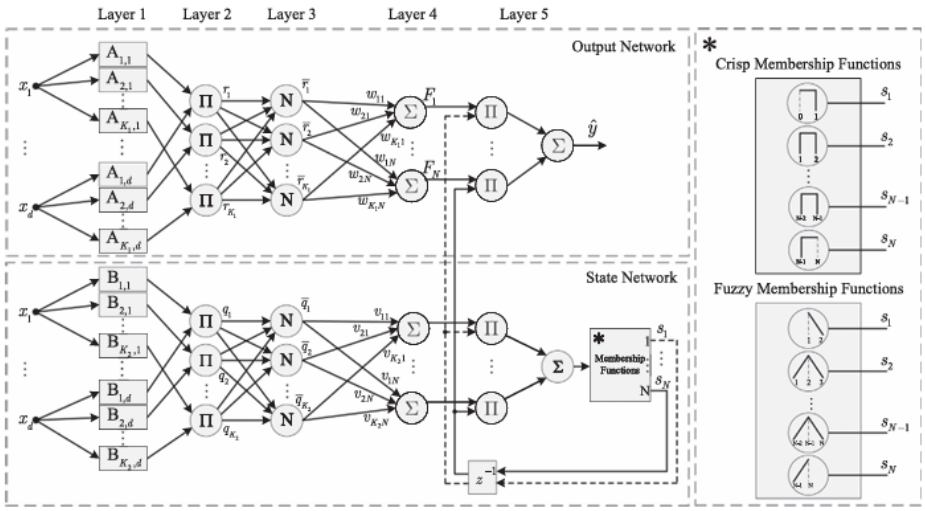


Figure 14: MFRFNN architecture

The output network performs N function approximations, using K_1 fuzzy rules. The system's output comprises N segments, where each segment represents a function approximation for a state, and the final output is obtained by summing these functions. The state network performs N function approximations using K_2 fuzzy rules to determine the system's next state. The system consists of five layers, each serving a distinct purpose as explained below.

3.1.1 MFRFNN Layers

3.1.1.1 Input Layer

The input layer receives the input data and transforms it using fuzzy membership functions. The output of each neuron of the input layer is the membership value of x_j on $A_{i,j}$, i.e., $\mu_{A_{i,j}}(x_j)$. Clearly, there are $K_1\ x\ d$ neurons in the output network and $K_2\ x\ d$ neurons in the state network in this layer.

3.1.1.2 Fuzzy Rule Layer

In the fuzzy rule layer, the firing strengths of the rules are calculated for both output and state networks. To calculate the firing strengths, the algebraic product is used as a T-norm operator ensuring that the overall activation strength reflects the combined influence of all input variables for a given rule.

$$r_i(x) = \prod_{j=1}^{d} \mu_{A_{i,j}}(x_j) \qquad q_i(x) = \prod_{j=1}^{d} \mu_{B_{i,j}}(x_j)$$

3.1.1.3 Normalized Fuzzy Rules Layer

The normalized rule layer, normalizes all rule activations making their sum equal to 1 and effectively representing the relative contribution of each rule.

$$\bar{r}_i(x) = \frac{r_i(x)}{\sum_{j=1}^{K_1} r_j(x)} \qquad \bar{q}_i(x) = \frac{q_i(x)}{\sum_{j=1}^{K_2} q_j(x)}$$

$$R(t) = [\bar{r}_1, \bar{r}_2, \ldots, \bar{r}_{K_1}]^T \qquad Q(t) = [\bar{q}_1, \bar{q}_2, \ldots, \bar{q}_{K_2}]^T$$

3.1.1.4 Extended Fuzzy Rule layer

The extended fuzzy rule layer computes the weighted sums of the activation rules, using the normalized firing strengths from the previous layer. The output of the layer represents the output of the approximated functions F_j and G_j for the output network and state network, respectively.

$$F_j = \sum_{i=1}^{K_1} \bar{r}_i \, w_{ij} \qquad G_j = \sum_{i=1}^{K_2} \bar{q}_i \, v_{ij}$$

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ & & \ddots & \\ w_{K_1 1} & w_{K_1 2} & \cdots & w_{K_1 N} \end{bmatrix} \qquad V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1N} \\ v_{21} & v_{22} & \cdots & v_{2N} \\ & & \ddots & \\ v_{K_2 1} & v_{K_2 2} & \cdots & v_{K_2 N} \end{bmatrix}$$

$$F(t) = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_N \end{bmatrix} = W^T R(t) \qquad G(t) = \begin{bmatrix} G_1 \\ G_2 \\ \vdots \\ G_N \end{bmatrix} = V^T Q(t)$$

3.1.1.5 Output Layer

Finally, the output layer of the output network provides the final prediction $\hat{y}$. The outputs of the approximate functions F_j are multiplied by the state signals s_j which activates only the

functions that correspond to the current state. The final output of the system is the sum of these functions.

$$\hat{y}(t) = S(t)^T F(t) = \sum_{j=1}^{N} s_j F_j \,, \qquad S(t) = \begin{bmatrix} s_1 \\ s_2 \\ \\ s_N \end{bmatrix}$$

To generate the state signals s_j, firstly the output of Layer 4 $G(t)$, is multiplied by the current state signals $S(t)$ and the results are summed giving us the intermediate output of the state's network output layer $o(t)$.

$$o(t) = G(t)^T S(t)$$

Then, $o(t)$ is normalized in the range $[1, N]$, $\bar{o}(t)$, to be used as input to membership functions in the output layer of the state network.

$$\bar{o}(t) = [o(t)(N-1) + 1]$$

The output of the MFs, $S(t+1)$, are the state signals (state network output) that determine the next state of the network. (Note: The network can use crisp or fuzzy MFs. In the case of crisp MFs the network has discrete states. On the other hand, when using fuzzy MFs the states become continuous and the final output is a weighted sum of the N approximated functions.)

$$S(t+1) = \begin{bmatrix} \mu_{E_1}(\bar{o}(t)) \\ \mu_{E_2}(\bar{o}(t)) \\ \vdots \\ \mu_{E_N}(\bar{o}(t)) \end{bmatrix}$$

3.1.2 Training Algorithm

MFRFNN employs a hybrid learning approach that combines two main techniques: the Least Squares Method and Particle Swarm Optimization (PSO). MFRFNN's total number of trainable parameters is (K_1 + K_2) * N which can be broken down to K_1 * N for the link weight matrix W of the output network and K_2 * N for the link weight matrix V of the state network.

3.1.2.1 Output Network's weight matrix (Least Squares Method)

To construct the output network's weight matrix, we need to estimate the optimal weight vector θ^* using the least squares method. To achieve this, the algorithm iterates over the training dataset, computing the rule normalized firing strengths, for both networks. Additionally, for each data point the approximate functions, normalized intermediate output and state signals of the state network are calculated. Once all data points have been processed, the optimal weight matrix is obtained using the Moore–Penrose pseudoinverse. Finally, the final cost value is calculated as the root mean squared error between predicted and actual outputs.

$$\text{RMSE} = \sqrt{\frac{1}{p}\sum_{i=1}^{p}(y^{[i]} - \hat{y}_i)^2}$$

- $x[t]$ and $y[t]$: input and output at timestep t
- p: $number\ of\ training\ samples$
- $D = \{(x[t], y[t])\}_{t=1}^{p}$: Training Dataset
- A: contribution of each fuzzy rule to each state for all training samples
- $\bar{r}_1^{[1]}$: normalized firing strength of *i*-th fuzzy rule for *j*-th training sample
- $y^{[i]}$: actual output of *i*-th training sample
- θ: vectorized representation of the weight matrix W
- $\theta *$: optimal weight vector that minimizes the square error $\|A\theta - y\|^2$

Aside from equation $\hat{y}(t) = F(t)^T S(t)$, the final output of the system can be also computed using $\hat{y} = \text{tr}(R(t)S^T(t)W)$. This can be derived by expressing $F(t)$ in terms of the firing strength vector $R(t)$ and the weight matrix W. Since $F_j = \sum_{i=1}^{K_1} \overline{r_i}\, w_{ij}$, we rewrite it in matrix form as $F(t) = W^T R(t)$. Substituting this into $\hat{y}(t)$ we get the following:

$$\hat{y}(t) = \left(W^T R(t)\right)^T S(t) = R(t)^T W S(t)$$

Applying the trace identity $a^T Bc = \text{tr}(ca^T B)$, we obtain:

$$\hat{y} = \text{tr}(R(t)S^T(t)W)$$

By inputting the training data into $\hat{y} = \mathrm{tr}(R(t)S^T(t)W)$ we get the matrix equation:

$$A\theta = y\,, \qquad y = \begin{bmatrix} y^{[1]} \\ y^{[2]} \\ \vdots \\ y^{[p]} \end{bmatrix}$$

$$A = \begin{bmatrix} \bar{r}_1^{[1]}s_1 & \bar{r}_1^{[1]}s_2 & \cdots & \bar{r}_2^{[1]}s_1 & \cdots & \bar{r}_{K_1}^{[1]}s_N \\ \bar{r}_1^{[2]}s_1 & \bar{r}_1^{[2]}s_2 & \cdots & \bar{r}_2^{[2]}s_1 & \cdots & \bar{r}_{K_1}^{[2]}s_N \\ \cdots & \cdots & \ddots & \vdots & \ddots & \vdots \\ \bar{r}_1^{[p]}s_1 & \bar{r}_1^{[p]}s_2 & \cdots & \bar{r}_2^{[p]}s_1 & \cdots & \bar{r}_{K_1}^{[p]}s_N \end{bmatrix}, \quad \theta = [w_{11} \quad w_{12} \quad w_{21} \quad \ldots \quad w_{K_1N}]^T$$

To estimate the optimal weight vector $\theta*$ we need to find a θ that minimizes the square error $\|A\theta - y\|^2$. Since A has more equations than unknowns we seek the least-squared solution which finds $\theta*$ that minimizes the residual error. This optimal solution is given by the Moore-Penrose pseudoinverse, which provides a closed-form solution:

$$\theta *= (A^T A)^{-1} A^T y$$

The Moore-Penrose pseudoinverse is particularly useful because it provides a direct, non-iterative solution and guarantees the optimal θ that minimizes the squared error.

Training of output network weight matrix (Least Squared Errors)

Input:

$K_1, K_2, N, V,$ Training dataset: $D = \{x^{[t]},\ y^{[t]}\}_{t=1}^{p}$

Output:

Optimal weight matrix θ^*, Cost value C

S ← 0

S_1 ← 1 // Initialize with starting state

A ← [ ] // Initialize an empty matrix

for t ← 1 to p do

  for i ← 1 to K_1 do

    Compute $r_i(x^{[t]})$

  end

  for i ← 1 to K_1 do

    Compute $\bar{r}_i(x^{[t]})$

  end

  Add new row $\left[\bar{r}_1^{[t]}s_1 \quad \bar{r}_1^{[t]}s_2 \quad \cdots \quad \bar{r}_2^{[t]}s_1 \quad \cdots \quad \bar{r}_{K_1}^{[t]}s_N\right]$ to matrix A

  for i ← 1 to K_2 do

    Compute $q_i(x^{[t]})$

  end

  for i ← 1 to K_2 do

    Compute $\bar{q}_i(x^{[t]})$

  end

  Compute $G(t)$

  Compute $o(t)$

  Compute $\bar{o}(t)$

  Compute S using// Determine the next state

end

Compute θ^*

$\hat{y} \leftarrow A\theta^*$

$C \leftarrow \sqrt{\frac{1}{p}\sum_{i=1}^{p}(y^{[i]} - \hat{y}_i)^2}$

3.1.2.2 State Network's weight matrix (Particle Swarm Optimization)

Since the state network's relationship to the output is nonlinear, to get the optimal weights of the state network V we cannot use the linear least-squares method. Instead, Particle Swarm Optimization (PSO)[21] is used.

Particle Swarm Optimization (PSO) is a population-based stochastic optimization algorithm inspired by the social behavior of flocking birds, originally proposed by Kennedy and Eberhart (1995).

- x_i^k: weight matrix of state network for the *i*-th particle at the *k* -th iteration
- v_i^k: velocity of the i-th particle at the *k* -th iteration
- p_i: personal best position of the i-th particle
- p_{gbest}: global best position among all particles in the swarm
- w_i: output network's weight matrix for the *i*-th particle
- c_1 and c_2: cognitive and social acceleration coefficients, determining the influence of p_i^k and p_{gbest}^k
- r_1 and r_2: random vectors uniformly distributed within [0,1]

In PSO, each potential solution to the optimization problem is called a particle, and a collection of these particles forms a swarm. Every particle is associated with a position (position= weight matrix of state network) and velocity, which are iteratively updated to explore the solution space effectively. The position x_i^k and velocity v_i^k of the i-th particle at iteration k are updated using the following equations:

$$v_i^{k+1} = w_i v_i^k + c_1 r_1 (p_i - x_i^k) + c_2 r_2 (p_{gbest} - x_i^k)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1}$$

3.1.2.3 Training

Together, these two algorithms enable MFRFNN to learn both the output and state network parameters. Firstly, PSO randomly initializes $x_i, v_i \; and \; w_i$ for each particle of the swarm. As mentioned above $x_i \; and \; w_i$ represent the weight matrix of state and output networks for the *i*-th particle, respectively. The weights are then optimized for each particle using the LSE algorithm which also calculates the cost value using the Root Mean Squared Error.

Additionally, now knowing the initial cost value of each particle, the global best position is initialized using the particle with the minimum error.

Having the initialized values of x_i, v_i, w_i and p_{gbest}, their training begins and goes on until a stopping criterion is met (such as a convergence threshold or a maximum number of iterations). During the PSO algorithm, for each iteration and each particle, x_i and v_i are update using the equations mentioned previously. Additionally, the weight matrix w of each particle is updated and its cost value is calculated. Taking into account if the cost value is lower than the previous ones, the particle's best position is updated and in the event of an overall best position of the swarm, the global best position is updated as well. The global best position of the swarm at the end of training is equal to the state network's weight matrix. Finally, having the optimized weight matrix for the state network, the optimized weight matrix of the output network can be created using the LSE algorithm.

3.1.3 Testing

To test the network, $\hat{y} = A\theta^*$ must be calculated, where θ^* are the trained weights of the output network and A can be constructed during the testing algorithm based on the input. Similarly, as the Output Network weights algorithm, the algorithm iterates over the testing dataset, computing the rule normalized firing strengths, for both networks. Additionally, for each data point the approximate functions, normalized intermediate output and state signals of the state network are calculated. At each iteration a new row $\left[\bar{r}_1^{[t]}s_1 \quad \bar{r}_1^{[t]}s_2 \quad \cdots \quad \bar{r}_2^{[t]}s_1 \quad \cdots \quad \bar{r}_{K_1}^{[t]}s_N\right]$ is added to matrix *A*. At the end of the algorithm, we have the complete matrix *A* and the already known θ^*, which are multiplied together to get the predictions for the testing dataset.

MFRFNN Training (PSO)

Input:

$K_1, K_2, N, V,$ Training dataset: $D = \{x^{[t]}, \ y^{[t]}\}_{t=1}^{p}$

Output:

The global best particle (gbest)

for each particle i do

Initialize $x_i \epsilon \mathbb{R}^{(K_2 xN)x1}$ and $v_i \epsilon \mathbb{R}^{(K_2 xN)x1}$ to random vectors

Initialize $w_i \epsilon \mathbb{R}^{(K_2 xN)x1}$ to random vectors

$p_i \leftarrow x_i$

Update w_i and calculate the cost value $f(p_i)$ using LSE algoritm

end

$gbest \leftarrow argmin_i f(p_i)$

while stopping criterion is not met do

for each particle i do

Update v_i

Update x_i

Update w_i and calculate $f(x_i)$ using Algorithm 1

If $f(x_i) < f(p_i)$ then

$p_i \leftarrow x_i$

If $f(p_i) < f(p_{gbest})$ then

$gbest \leftarrow i$

end

end

end

end

Testing Algorithm

Input:

K_1, K_2, N, V, θ^* Training dataset: $D = \{x^{[t]},\ y^{[t]}\}_{t=1}^{p}$

Output:

Predicted Output ($\hat{y}$)

S ← 0

S₁ ← 1 // Initialize with starting state

A ← [ ] // Initialize an empty matrix

for t ← 1 to p do

  for i ← 1 to K_1 do

    Compute $r_i(x^{[t]})$

  end

  for i ← 1 to K_1 do

    Compute $\bar{r}_i(x^{[t]})$

  end

  Add new row $\left[\bar{r}_1^{[t]}s_1 \quad \bar{r}_1^{[t]}s_2 \quad \cdots \quad \bar{r}_2^{[t]}s_1 \quad \cdots \quad \bar{r}_{K_1}^{[t]}s_N\right]$ to matrix A

  for i ← 1 to K_2 do

    Compute $q_i(x^{[t]})$

  end

  for i ← 1 to K_2 do

    Compute $\bar{q}_i(x^{[t]})$

  end

  Compute $G(t)$

  Compute $o(t)$

  Compute $\bar{o}(t)$

  Compute S // Determine the next state

end

$\hat{y} \leftarrow A\theta^*$

3.2 Architecture of ReNFuzz-LF

The rule base of traditional Takagi–Sugeno–Kang (TSK)[3] models consist of fuzzy sets in the premise part and linear functions of the inputs as the consequent part but in general these functions can be any continuous and derivable nonlinear function. ReNFuzz-LF follows a hybrid fuzzy-neural architecture where each fuzzy rule contains a small-scale recurrent neural network with local feedback as its consequent part.

$$\text{IF } x_1(k) \text{ is } A_1 \text{ AND } \dots \text{ AND } x_m(k) \text{ is } A_m \text{ THEN } g\big(x(k)\big)$$

The model doesn't incorporate external output feedback, enabling it to maintain local learning capabilities of the classical TSK model while benefiting from the structured uncertainty handling of fuzzy logic. The premise part of the fuzzy rules is static, whereas the consequent parts dynamically adjust to capture time-dependent variations in the time series data. These dynamic consequent parts, connect with each other during the defuzzification process to calculate the output of the model.

- N: number of samples
- $x(k) = [x_1, \cdots, x_m]^T$: *k*-th sample vector
- $x_i(k)$: *i*-th input of *k*-th sample vector
- A_i^l: fuzzy set of the *i*-th input of a *l*-th rule
- R: number of rules
- $\mu_l(k)$: firing strength of *l*-th rule for *k*-th sample
- $\mu_{A_i^l}(x_i(k))$: membership degree of $x_i(k)$ in A_i^l
- $m_l = [m_{l1}, \cdots, m_{lm}]^T$: mean values of all Gaussian membership functions of *l*-th rule
- $\sigma_l = [\sigma_{l1}, \cdots, \sigma_{lm}]^T$: standard deviations of all Gaussian membership functions of *l*-th rule
- $w_{lhi}^{(1)}, w_{lh}^{(2)}$: synaptic weights at the hidden layer of the consequent parts. ($w_{lh}^{(1)}$ is a vector of all the weights connecting each input from the input layer to the hidden neurons)
- $w_{lh}^{(3)}$: bias terms of the hidden layer of the consequent parts
- $w_{lh}^{(4)}, w_l^{(5)}$: synaptic weight at the output layer of the consequent parts
- $w_l^{(5)}$: bias terms of the output layer of the consequent parts
- $f(k, l)$: output of activation function of the *l*-th fuzzy rule of the *k*-th sample
- $s_{lh}(k)$: output of the *h*-th hidden neuron of the *l*-th fuzzy rule of the *k*-th sample
- g_l: output of the *l*-th fuzzy rule

3.2.1 Fuzzy Rules and Defuzzification

3.2.1.1 Premise Part

The premise part of each rule of the fuzzy rule base, is composed of single-dimension Gaussian membership functions.

$$\mu_{A_i^l}(x_i(k)) = exp\left\{-\frac{(x_i(k) - m_{li})^2}{2\sigma_{li}^2}\right\}$$

The firing strength of each rule is calculated as the algebraic product of the Gaussian membership functions

$$\mu_l(k) = f_\mu(x(k);\, m_l, \sigma_l) = \prod_{i=1}^{m} \mu_{A_i^l}(x_i(k)), \qquad l = 1, \ldots, R$$

3.2.1.2 Consequent Part

Each fuzzy rule's consequent part is a three-layer RNN with local output feedback, which enables the system to retain historical information. The RNN input layer receives *m* inputs, the hidden layer has *H* hidden neurons and the output layer calculates the output of the RNN. The activation function $f(\cdot)$ used in the network is the hyperbolic tangent.

$$f(k, l) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

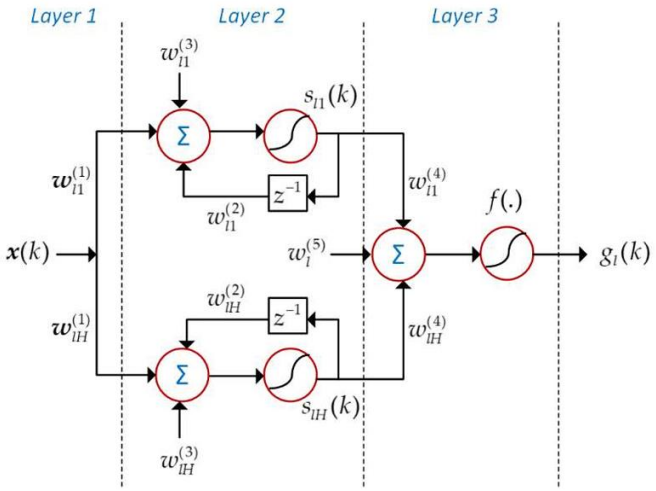


Figure 15: ReNFuzz-LF consequents part RNN configuration

The output of the *h*-th hidden neuron of the *l*-th rule is calculated as follows:

$$s_{lh}(k) = f\left((w_{lh}^{(1)})^T x(k) + w_{lh}^{(2)} s_{lh}(k) + w_{lh}^{(3)}\right) = f\left(\sum_{i=1}^{m}\left[w_{lhi}^{(1)} x_i(k)\right] + w_{lh}^{(2)} s_{lh}(k) + w_{lh}^{(3)}\right)$$

and the output of the *l*-th fuzzy rule:

$$g_l(k) = f\left(\sum_{h=1}^{H}\left[w_{lh}^{(4)} s_{lh}(k)\right] + w_l^{(5)}\right)$$

3.2.1.3 Defuzzification Part

After calculating all the rule activations, the final output of ReNFuzz-LF is calculated using the weighted average method.

$$y(k) = \frac{\sum_{l=1}^{R} \mu_l(k) g_l(k)}{\sum_{l=1}^{R} \mu_l(k)}$$

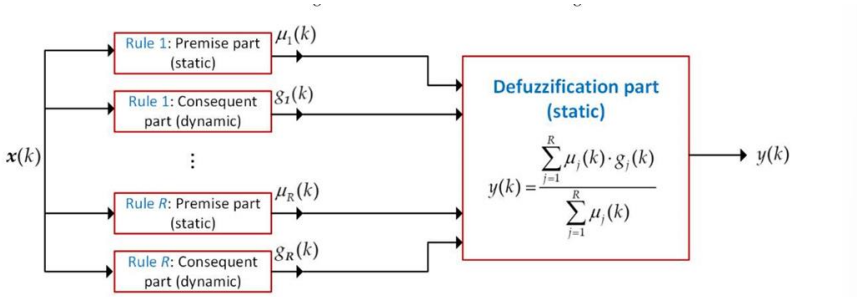


Figure 16: ReNFuzz-LF diagram

3.2.2 Training Algorithm

ReNFuzz-LF is trained using a hybrid optimization method using the Fuzzy C-means clustering to calculate the premise's part parameters and an iterative algorithm based on Simulated Annealing Dynamic Resilient Propagation (SA-DRPROP) to optimize the synaptic weights of the consequent's part RNN.

The total number of trainable parameters is *R*(2m+(m+3)H+1)* which can be broken down to *2mR* for the premise part and *((m+3)H+1)R* for the consequent part, where m is equal to the number of inputs of each sample vector.

3.2.2.1 Premise Part Parameters (FCM)

- m_{li}: mean value of the Gaussian MF of the *i*-th input of the *l*-th rule (centres of FCM clusters)
- σ_{li}: standard deviation of the Gaussian MF of the *i*-th input of the *l*-th rule
- $u_{li}(k) = [u_{li}(k), \dots, u_{lm}(k)]^T$: membership degree that *x(k)* belongs to the *l*-th cluster
- c: scale parameter within [0, 1]

The trainable parameters of the premise part are static. This means that they are calculated once and remain unchanged during the training of the consequent's part parameters. The algorithm employed to calculate the m and σ parameters of the premise part and consequently the partition of the input space, is the fuzzy C-means (FCM).

FCM is a generalization of the K-means algorithm. It's an unsupervised clustering algorithm that incorporates fuzzy set theory. Rather than assigning each data point to a specific cluster it allows for varying degrees of membership to multiple.

The mean values *m* of the Gaussian membership functions are equal to the cluster centers and the standard deviations *σ* are calculated as shown below.

$$m_{li} = \frac{\sum_{k=1}^{N} u_{li}(k) x_i(k)}{\sum_{k=1}^{N} u_{li}(k)}$$

$$u_{li}(k) = \left[\sum_{k=1}^{R} \frac{\sum_{i=1}^{m} (m_{li} - x_i(k))^2}{\sum_{i=1}^{m} (m_{li})^2} \right]^{1-\frac{1}{c}}$$

$$\sigma_{li} = \frac{c \sum_{k=1}^{N} u_{li}(k) (m_{li} - x_i(k))^2}{\sum_{k=1}^{N} u_{li}(k)}$$

To evaluate the best partition of the input data the Davies-Bouldin Index (DBI) is calculated. DBI measures the compactness and separation of clusters to determine how well-defined they are. Partitions with the lower DBI, result in better clustering results. This might not always create the best model though because a high number of clusters may result in overfitting. That's why multiple models with different partitions might be train to evaluate which is the best.

3.2.2.2 Consequent Part (SA-DRPROP)

The use of an iterative approach based on SARPROP for the training of the RNN, helps address the problem of local minima trapping, and makes a broader search across the weight space to find the optimal weights.

- w_i: consequent synaptic weight i
- $\frac{\partial^+ E(t)}{\partial w_i}$ and $\frac{\partial^+ E(t-1)}{\partial w_i}$: partial derivatives of error function E with respect to the adaptive weight w_i for present *t* and previous *t-1* iteration of SA-DRPROP
- $f'(k,l) = 1 - f(k,l)^2$: derivative of $g_l(k)$ with respect to its arguments
- $\hat{y}(k)$: actual output value
- w_l: $one\ of\ weights\ w_1, w_2\ or\ w_3\ of\ l-th\ fuzzy\ rule$
- $\lambda_{lh}(N)$: boundary condition of the h-th neuron of l-th rule(N-final sample)
- $\lambda_{lh}(k)$: Lagrange multiplier of the h-th neuron of l-th rule for the k-th sample
- $Temp = 1.2$: Temperature
- $SA = 2^{-t \cdot Temp}$: Simulated Annealing term at t-th iteration
- $\alpha_1 = 0.01$
- $\alpha_2 = 0.4$
- $\Delta_0 = 0.01$: initialized step size
- $\Delta_{min} = 0.0001$: minimum step size
- $\Delta_{max} = 0.5$: maximum step size
- $\eta^- = 0.5$: step size decrease
- $\eta^+ = 1.05$: step size increase
- r: random noise value within [0,1]

Before initiating and after each iteration of the training algorithm the derivatives of the error function E with respect to all weights must be calculated (Note: the synaptic weights are initialized randomly). The error function used for the extraction of the error gradients is the Root Mean Squared Erro (RMSE).

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=1}^{N}[y(k) - \hat{y}(k)]^2}$$

For weights $w_4\ and\ w_5$ the derivatives are calculated using the classic chain rule:

$$\frac{\partial E}{\partial w_{lh}^{(4)}} = \frac{2}{N}\left\{\sum_{k=1}^{N}[\,y(k) - \hat{y}(k)]\frac{\mu_l(k)f'(k,l)s_{lh}(k)}{\sum_{i=1}^{R}\mu_i(k)}\right\}$$

$$\frac{\partial E}{\partial w_{l}^{(5)}} = \frac{2}{N}\left\{\sum_{k=1}^{N}[\,y(k) - \hat{y}(k)]\frac{\mu_l(k)f'(k,l)}{\sum_{i=1}^{R}\mu_i(k)}\right\}$$

For weights $w_1,\ w_2\ and\ w_3$ ordered derivatives $\frac{\partial^+ E}{\partial w_l}$ are necessary to unfold in time the neuron's operation.

$$\frac{\partial^+ E}{\partial w_l} = \sum_{k=1}^{N}\frac{\partial E}{\partial y(k)}\frac{\partial^+ y(k)}{\partial w_l}$$

$$= \sum_{k=1}^{N}\frac{\partial E}{\partial y(k)}\frac{\partial y(k)}{\partial g_l(k)}\frac{\partial^+ g_l(k)}{\partial w_l} = \frac{2}{N}\sum_{k=1}^{N}\left\{[y(k) - \hat{y}(k)]\frac{\mu_l(k)}{\sum_{i=1}^{R}\mu_i(k)}\frac{\partial^+ g_l(k)}{\partial w_l}\right\}$$

Additionally, to facilitate the calculation of $\frac{\partial^+ g_l(k)}{\partial w_l}$ Lagrange multipliers are incorporated. Firstly, the boundary conditions $\lambda_{lh}(N)$ are calculated and then the Lagrange multipliers are calculated backwards from $\lambda_{lh}(N-1)$ to $\lambda_{lh}(1)$.

$$\lambda_{lh}(N) = \frac{2}{N}\sum_{k=1}^{N}\left\{[y(N) - \hat{y}(N)]\frac{\mu_l(N)f'(N,l)w_{lh}^{(4)}}{\sum_{i=1}^{R}\mu_i(N)}\right\}$$

$$\lambda_{lh}(k) = \lambda_{lh}(k+1)f'(k+1,l,h)w_{lh}^{(2)} + \frac{2}{N}\sum_{k=1}^{N}\left\{[y(k) - \hat{y}(k)]\frac{\mu_l(k)f'(k,l)w_{lh}^{(4)}}{\sum_{i=1}^{R}\mu_i(k)}\right\}$$

By applying the Lagrange multipliers to $\frac{\partial^+ E}{\partial w_l}$ the ordered derivatives of E with respect to $w_1,\ w_2\ and\ w_3$ are:

$$\frac{\partial^+ E}{\partial w_{lhi}^{(1)}} = \sum_{k=1}^{N} \lambda_{lh}(k) f'(k,l,h) x_i(k)$$

$$\frac{\partial^+ E}{\partial w_{lh}^{(2)}} = \sum_{k=1}^{N} \lambda_{lh}(k) f'(k,l,h) s_{lh}(k-1)$$

$$\frac{\partial^+ E}{\partial w_{lh}^{(3)}} = \sum_{k=1}^{N} \lambda_{lh}(k) f'(k,l,h)$$

Having calculated all error gradients, the new SA-DRPROP error gradients are calculated by adding a weight decay term to the error gradients. (Note: SA-DRPROP error gradients are initialized randomly)

$$SA - DRPROP\ error\ gradient\ (itereation\ t) = \frac{\partial^+ E(t)}{\partial w_i} - \alpha_1 \cdot SA \cdot \frac{w_i}{1 + w_i^2}$$

To update each weight after each iteration, their respective step sizes must be updated. The step sizes of each weight are initialized to a small value Δ_0 and during training they are adjusted according to the sign of the SA-DRPROP error gradient at the current and previous iteration of the respective weight. Obviously, there are 3 different cases:

1. If the error gradients have the same sign $\left\{\frac{\partial^+ E(t)}{\partial w_l} \cdot \frac{\partial^+ E(t-1)}{\partial w_l} > 0\right\}$, this leads to a step size increase.

$$\Delta_i^{(t)} = min\left\{\eta^+ \Delta_i^{(t-1)}, \Delta_{max}\right\}$$

2. If the error gradients have opposite signs $\left\{\frac{\partial^+ E(t)}{\partial w_l} \cdot \frac{\partial^+ E(t-1)}{\partial w_l} < 0\right\}$, this leads to a step decrease. When the step size is lower than the threshold $\alpha_2 \cdot SA^2$ indicates a possibility of falling into a local minimum, so noise r is added to the step size to help the weight overcome it,

$$\Delta_i^{(t)} = max\left\{\eta^- \cdot \Delta_i^{(t-1)} \cdot r \cdot SA^2, \Delta_{mix}\right\}$$

otherwise,

$$\Delta_i^{(t)} = max\left\{\eta^- \Delta_i^{(t-1)}, \Delta_{mix}\right\}$$

3. In the case where $\left\{\frac{\partial^+ E(t)}{\partial w_l} \cdot \frac{\partial^+ E(t-1)}{\partial w_l} = 0\right\}$ the step size remains the same

Finally, the weight updates are calculated as follows:

$$w_i(t) = w_i(t-1) - sign\left(\frac{\partial^+ E(t)}{\partial w_i}\right)\Delta_i^{(t)}$$

SA-DRPROP Training

Input:

$SA\ error\ gradients(SAEG_i)\ and\ weights(w_i)\ of\ previous\ iteration,$

$\frac{\partial^+ E(t)}{\partial w_i}, \Delta_i^{(t-1)}, \alpha_1, \alpha_2, r, SA, \eta^-, \eta^+, \Delta_{mix}, \Delta_{max}$

Output:

Updated weights

For each weight w_i do

$\text{SAEG}_i(\text{t}) = \frac{\partial^+ E(t)}{\partial w_i} - \alpha_1 \cdot SA \cdot \frac{w_i(t-1)}{1+w_i(t-1)^2}$

If $\frac{\partial^+ E(t)}{\partial w_i} \cdot \frac{\partial^+ E(t-1)}{\partial w_i} > 0$

$\Delta_i^{(t)} = min\left\{\eta^+ \Delta_i^{(t-1)}, \Delta_{max}\right\}$

Else, if $\frac{\partial^+ E(t)}{\partial w_i} \cdot \frac{\partial^+ E(t-1)}{\partial w_i} < 0$

if $(\Delta_i^{(t)} < \alpha_2 \cdot SA^2)$

$\Delta_i^{(t)} = max\left\{\eta^- \cdot \Delta_i^{(t-1)} \cdot r \cdot SA^2, \Delta_{min}\right\}$

else

$\Delta_i^{(t)} = max\left\{\eta^- \cdot \Delta_i^{(t-1)}, \Delta_{min}\right\}$

Else, $\Delta_i^{(t)} = \Delta_i^{(t-1)}$

end

end

Update w_i: $w_i(t) = w_i(t-1) - sign\left(\frac{\partial^+ E(t)}{\partial w_i}\right)\Delta_i^{(t)}$

3.3 Datasets

3.3.1 Wind Speed Prediction Problem

This dataset was sourced from the Iowa Department of Transport's website and consist of 647 samples of wind direction and speed from February 2011. This dataset represents a non-linear, dynamic, and volatile time series problem, where the objective is to predict the future wind speed based on current wind speed and wind direction. Due to the chaotic nature of wind patterns, predicting wind speed is particularly challenging, requiring robust forecasting models that can effectively capture temporal dependencies and stochastic variations. The dataset was split into 500 samples for training and 147 samples for testing.

3.3.2 Box-Jenkins Gas Furnace

The Box-Jenkins Gas Furnace problem is a time-series forecasting benchmark, where the objective is to predict the CO_2 concentration rate based on the input oxygen flow rate. The dataset was split into 200 samples for training and 96 samples for testing. Unlike the Wind Speed Prediction dataset, where external environmental factors influence predictions, the Box-Jenkins problem is a controlled process with a clearer input-output dependency. However, its non-linearity make it a strong benchmark for evaluating adaptive learning models.

3.3.3 Google Stock Price

Stock price prediction is inherently a non-linear, chaotic, and volatile problem due to the influence of market fluctuations, investor sentiment, and external economic factors. In the Google Stock Price dataset, the objective is to predict the future stock price based on the previous day's closing price. The dataset consists of 1532 daily google stock prices, from which the 1200 are used for training and the 332 for testing.

3.3.4 Lorenz System

The Lorenz system is a system. originally developed by Edward Lorenz. It is modeled by three differential equations which when $\sigma=10$, $\beta=8/3$ and $\rho=28$, it has chaotic solutions:

$$\frac{dx}{dt} = \sigma(y - x), \quad \frac{dy}{dt} = x(\rho - z) - y, \quad \frac{dz}{dt} = xy - \beta z$$

The dataset consists of 20,000 time-series samples, each containing three state variables x(t), y(t), and z(t), which evolve over time. The data was generated using the fourth-order Runge-Kutta method and the objective is to do one-step ahead predictions. The first 11250 samples were used for training and the last 5000 for testing. The rest were used as a validation set.

3.3.5 Air Quality Index (AQI)

The Air Quality Index (AQI) prediction dataset is a real-world time-series dataset used for evaluating models in multi-step-ahead forecasting of air pollution levels. This dataset was collected from 12 air quality monitoring stations around Beijing, China, covering the period from 2013 to 2017. The data contains frequent and drastic fluctuations, making it a challenging benchmark for predictive modeling. The objective of this dataset is to predict future AQI levels based on past pollution measurements. The dataset consists of 35,064 samples, recorded hourly over a period of 1,461 days (4 years). Each sample contains six major air pollution components that significantly impact AQI values:

1. PM2.5 (Fine Particulate Matter, μg/m³) – Tiny particles that pose significant health risks.
2. PM10 (Respirable Particulate Matter, μg/m³) – Coarser particles that affect respiratory health.
3. SO_2 (Sulfur Dioxide, ppm) – A gas that contributes to acid rain and respiratory issues.
4. NO_2 (Nitrogen Dioxide, ppm) – A pollutant associated with vehicle emissions and industrial activity.
5. CO (Carbon Monoxide, ppm) – A toxic gas affecting human health, often from combustion sources.
6. O_3 (Ozone, ppm) – A key component of smog, which can cause breathing problems.

The dataset is split into 22800 samples (950 days) for training, 1200 samples (50 days) for validation and the rest for testing. Additionally, a sliding window approach is used, where four past time steps are utilized as input sequences to predict future AQI values.

3.3.6 Electric Load

The electric load dataset for derived from the Greek Power Transmission Operator. The dataset consists of 35064 samples at one-hour intervals of the electric load consumption. The training dataset, comprises of 26280 samples, representing historical data from three consecutive years (2013–2015) and the testing set, contains 8784 samples of electric load values for the year 2016.

3.4 Evaluation Metrics

The **Root Mean Squared Error (RMSE)** measures the average magnitude of prediction errors while penalizing larger errors more significantly.

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

The **Mean Squared Error (MSE)** is similar to RMSE but does not apply the square root transformation.

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

The **Mean Absolute Error (MAE)** averages the absolute differences between predictions and actual values making it a more straightforward metric. Unlike RMSE and MSE, MAE treats all errors equally without disproportionately penalizing larger errors.

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

The **Symmetric Mean Absolute Percentage Error (sMAPE)** provides an error measure in percentage form, making it useful when comparing datasets with different scales.

$$\text{sMAPE} = \frac{100\%}{N}\sum_{i=1}^{N}\frac{| y_i - \hat{y}_i |}{\frac{| y_i | + | \hat{y}_i |}{2}}$$

4 Results and Analysis

In this chapter a performance analysis is made for MFRFNN and ReNFuzz-LF on the Lorenz chaotic system and five real-world datasets, including Box–Jenkins Gas Furnace, Wind Speed Prediction, Google Stock Price Prediction, Air Quality Index Prediction and Electric Load Dataset. To evaluate their performance the Mean Square Error (MSE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Symmetric Mean Absolute Percentage Error (sMAPE) were used.

For MFRFNN all datasets were normalised to the range [0, 1] and for ReNFuzz-LF to [-0.8, 0.8]. Since having this difference in normalisation, the results in Tables 1-7 are calculated with the denormalized predictions and outputs of each model.

The hyperparameters of MFRFNN, including the number of fuzzy rules for the output and state networks, the number of states, and the maximum fitness evaluations for the PSO algorithm. These hyperparameters are portrayed in Table 10 for each benchmark. For the input layer of both of MFRFNN's networks, uniformly distributed triangular membership functions were used as well as for the continuous MFs in the output layer of the state network.

The hyperparameters of ReNFuzz, include the number of hidden neurons, number of fuzzy rules, MFs and the learning parameters of the SA-DRPROP algorithm which are shown in Tables 8-9. In order to create the MFs, partition by means of FCM (Fuzzy C-Means) was performed to obtain the mean and standard deviation parameters of the Gaussian MFs. Additionally, the Davies-Bouldin index was examined to choose the best partition of the data for each dataset. Once the fuzzy sets were created, the parameters of the MFs remained unchanged during training of the model. Finally, the number of hidden neurons in the rule consequents were chosen by trial and error by examining [2, 3, 4, 5, 10] number of neurons.

4.1 Results

Performance metrics across several datasets underscore the advantages of each approach in different contexts.

4.1.1 Google Stock Price (1-step ahead prediction)

In the one-step-ahead prediction of the Google Stock Price dataset, which is highly non-stationary and exhibits significant short-term fluctuations, MFRFNN outperforms ReNFuzz-LF, achieving lower RMSE (0.696 vs. 0.782) and MAE (0.289 vs. 0.782). ReNFuzz-LF's

higher absolute error suggests that its localized memory and rule base finds it harder to adapt to the high volatility and rapid fluctuations in stock prices, in comparison to MFRFNN. This indicates that MFRFNN's feedback mechanism and dual network architecture, appears to be better suited for capturing short-term dependencies of stock prices, resulting in higher accuracy and lower prediction errors.

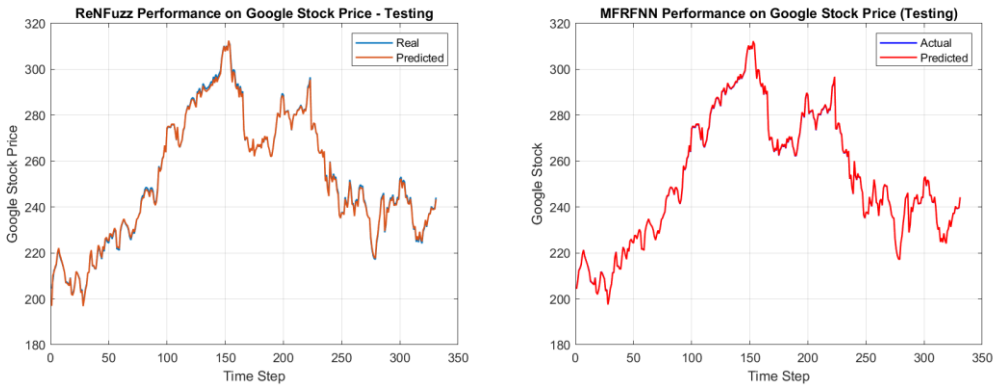


Figure 17: Performance on Google Stock Price Dataset

Table 1: Google Stock Price problem errors

One step ahead prediction error on Google Stock Price problem				
Method	**RMSE**	**MSE**	**MAE**	**sMAPE**
MFRFNN	0.696	0.484	0.289	0.204
ReNFuzz-LF	0.782	0.612	0.576	0.232

4.1.2 Box-Jenkins Gas Furnace (Two-Input)

The Box–Jenkins Gas Furnace dataset, is a two-input forecasting problem. Here, ReNFuzz-LF and MFRFNN perform comparably. For one-step ahead predictions, ReNFuzz-LF achieves a slightly lower RMSE (0.606 vs. 0.649), indicating a good fit to the data. However, MFRFNN achieves lower MAE (0.424 vs 0.493) which suggests it might have some large errors that inflate RMSE. Additionally, as it can be seen on figure 19 both models struggled to predict local min and max values, suggesting that additional improvements may be needed.

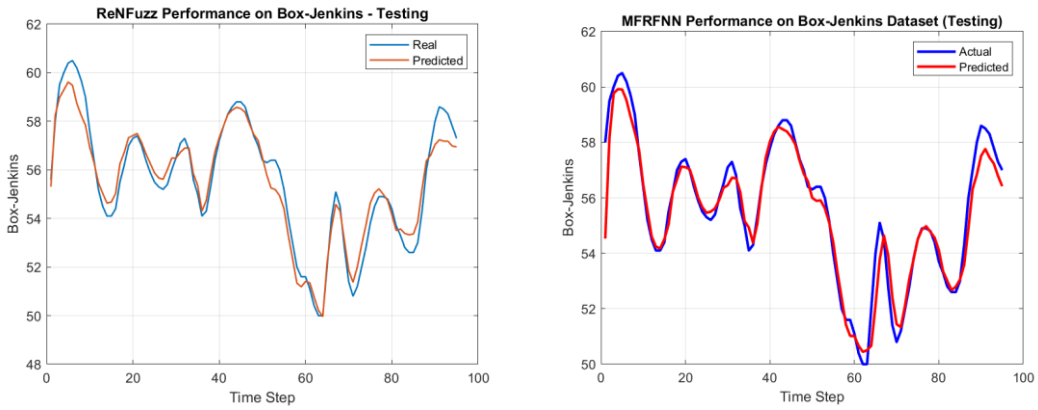


Figure 18: Performance on Box-Jenkins Dataset

Table 2: Box-Jenkins Gas Furnace problem errors

One step ahead prediction error on Box-Jenkins gas furnace problem				
Method	**RMSE**	**MSE**	**MAE**	**sMAPE**
MFRFNN	0.649	0.421	0.424	0.765
ReNFuzz-LF	0.606	0.367	0.493	0.887

4.1.3 Wind Speed Forecasting (Two-Input)

The Wind Speed dataset, is also a two-input problem. For this task, MFRFNN outperforms ReNFuzz-LF across all metrics, achieving a lower RMSE (0.553 vs. 1.009), lower sMAPE (9.829 vs. 16.897), and better MAE (0.435 vs. 0.779). These results indicate that ReNFuzz-LF struggled to learn the wind speed dynamics. As can be seen on figure 20, the ReNFuzz model is unable to predict the low wind values. The feedback mechanism of MFRFNN appears to provide better adaptability, making it the superior model for short-term wind speed forecasting in this case.

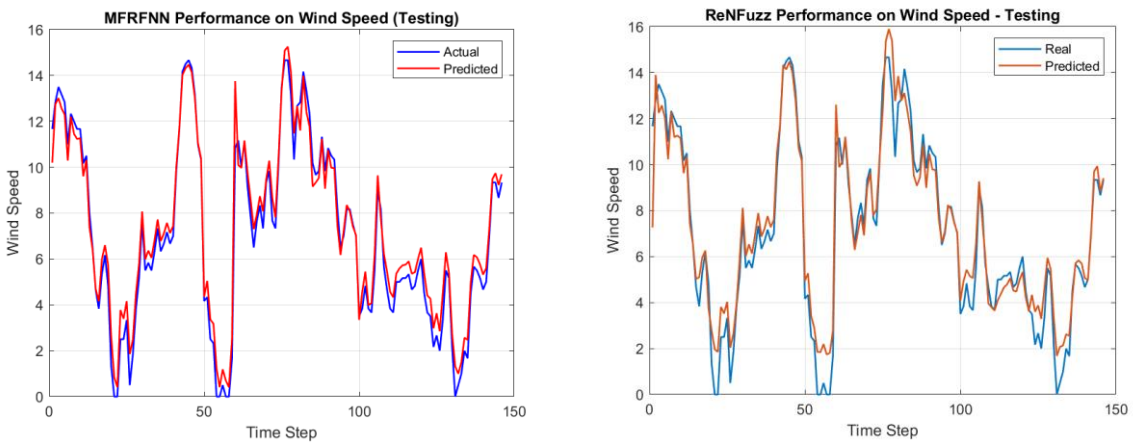


Figure 19: Performace on Wind Speed Dataset

Table 3: Wind Speed problem errors

One step ahead prediction error on Wind Speed problem				
Method	**RMSE**	**MSE**	**MAE**	**sMAPE**
MFRFNN	0.553	0.306	0.435	9.829
ReNFuzz-LF	0.930	0.865	0.707	22.944

4.1.4 Lorenz System (Chaotic System)

For the chaotic time series generated by the Lorenz system, ReNFuzz-LF's design, with its localized RNN consequents, captures the short-term chaotic dynamics better than MFRFNN, allowing the model to achieve very low error measures. The model's ability to retain short-term memory seems to be well matched to the sensitive nature of chaotic systems, making it a better forecaster for this problem. This is probably due to MFRFNN was overestimating some values as it can be seen on Figures 21. Despite this, MFRFNN ability to learn multiple functions simultaneously show very promising results as it was able to capture the pattern of the data.

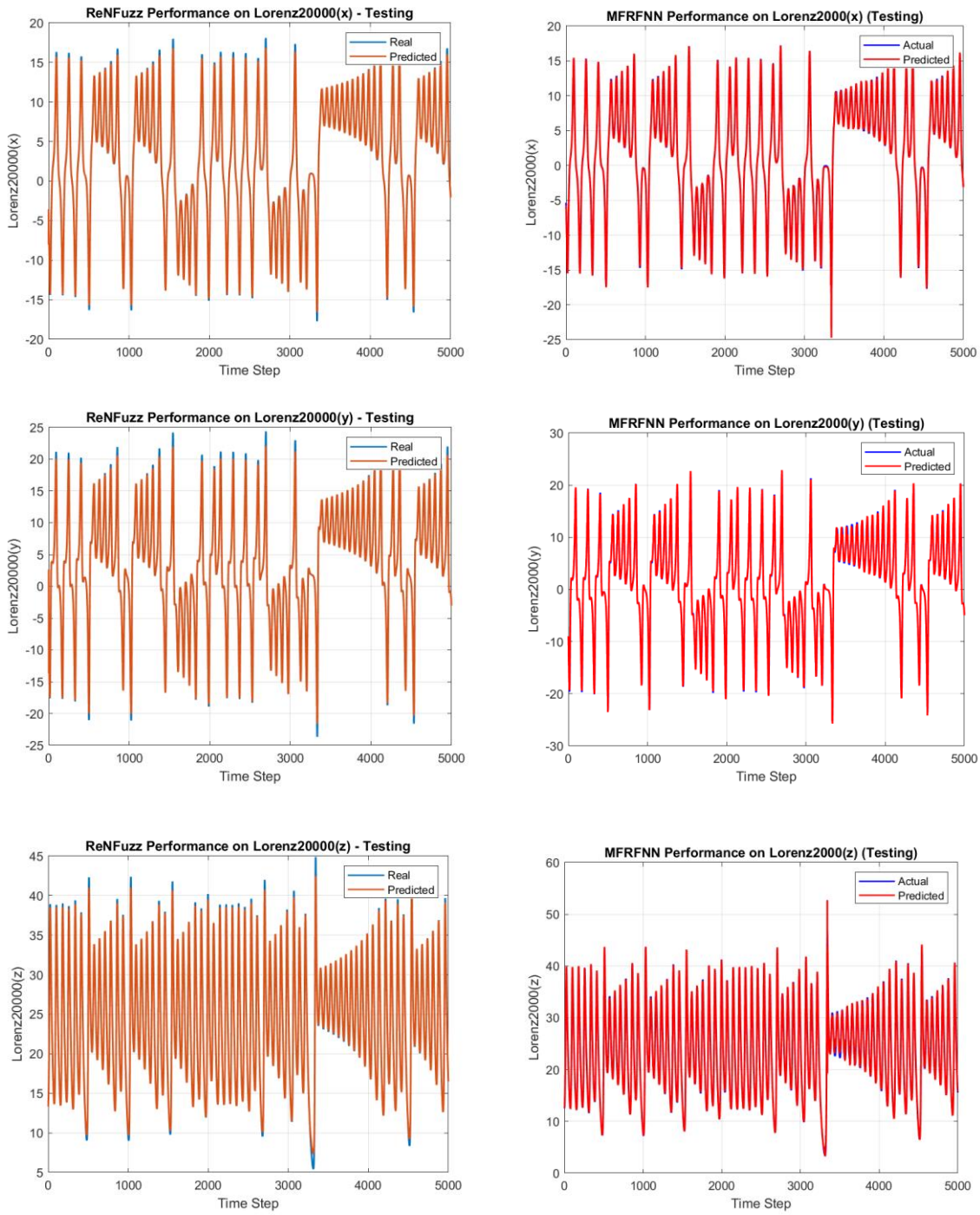


Figure 20: Performance on Lorenz system

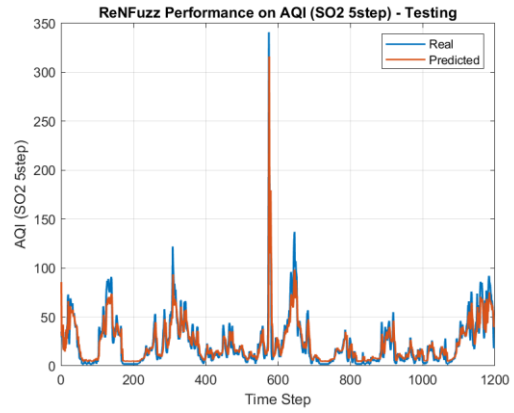
Table 4: Lorenz System problem errors

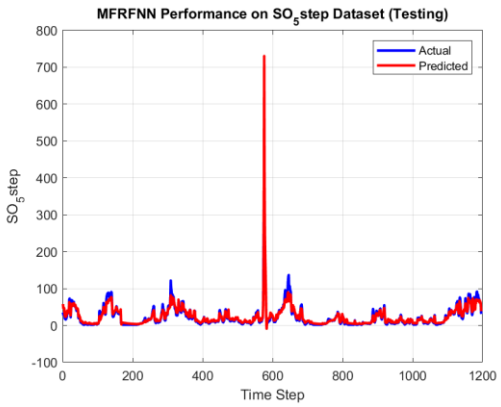
One step ahead prediction error on Lorenz System problem

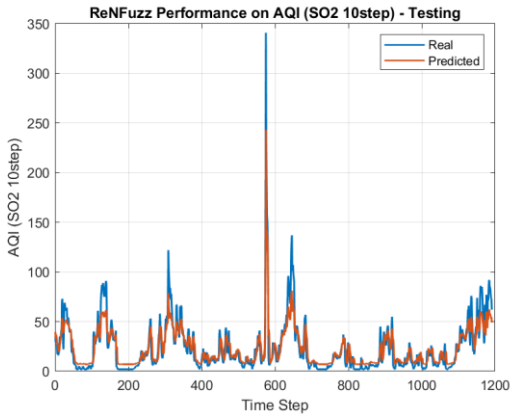
Method	**RMSE**			**MSE**			**MAE**			**sMAPE**		
	x	***y***	***z***	***x***	***y***	***z***	***x***	***y***	***z***	***x***	***y***	***z***
MFRFNN	0.344	0.505	0.843	0.118	0.256	0.71	0.225	0.312	0.479	7.452	7.858	1.983
ReNFuzz-LF	0.198	0.342	0.299	0.039	0.117	0.089	0.138	0.182	0.198	6.003	4.679	1.006

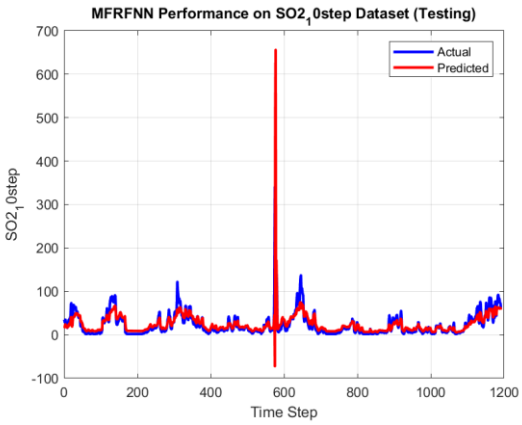
4.1.5 AQI (5 & 10 step predictions)

Multi-step forecasts, such as those for various AQI datasets (e.g., 5-step and 10-step ahead predictions), introduce additional challenges due to error accumulation. To train these models, a rolling window of 4 inputs through the data was used at each timestep to try to capture the underline dynamics of the time-series. The results indicate that ReNFuzz-LF outperforms MFRFNN in forecasting PM2.5, PM10, SO2, and O3, particularly in both five-step and ten-step ahead predictions based on the RMSE, and MFRFNN exhibits better performance for NO2 and CO. However, both models experience increased error metrics in comparison with the previous datasets. ReNFuzz-LF, which relies on short-term memory in each fuzzy rule, seems to struggle maintaining information over extended horizons, leading to higher relative errors. MFRFNN, while designed to capture multiple state transitions through its feedback loop, also faces difficulties with cumulative uncertainty. From figures 21 is evident that the models struggle to predict very low and high values while capturing the overall outline of the data and having better predictions on medium values. Additionally, MFRFNN seems to overestimate some outliers in the data adding to its worse performance metrics. These challenges underscore the inherent complexity of long-horizon forecasting and suggest that both approaches may require further refinement.









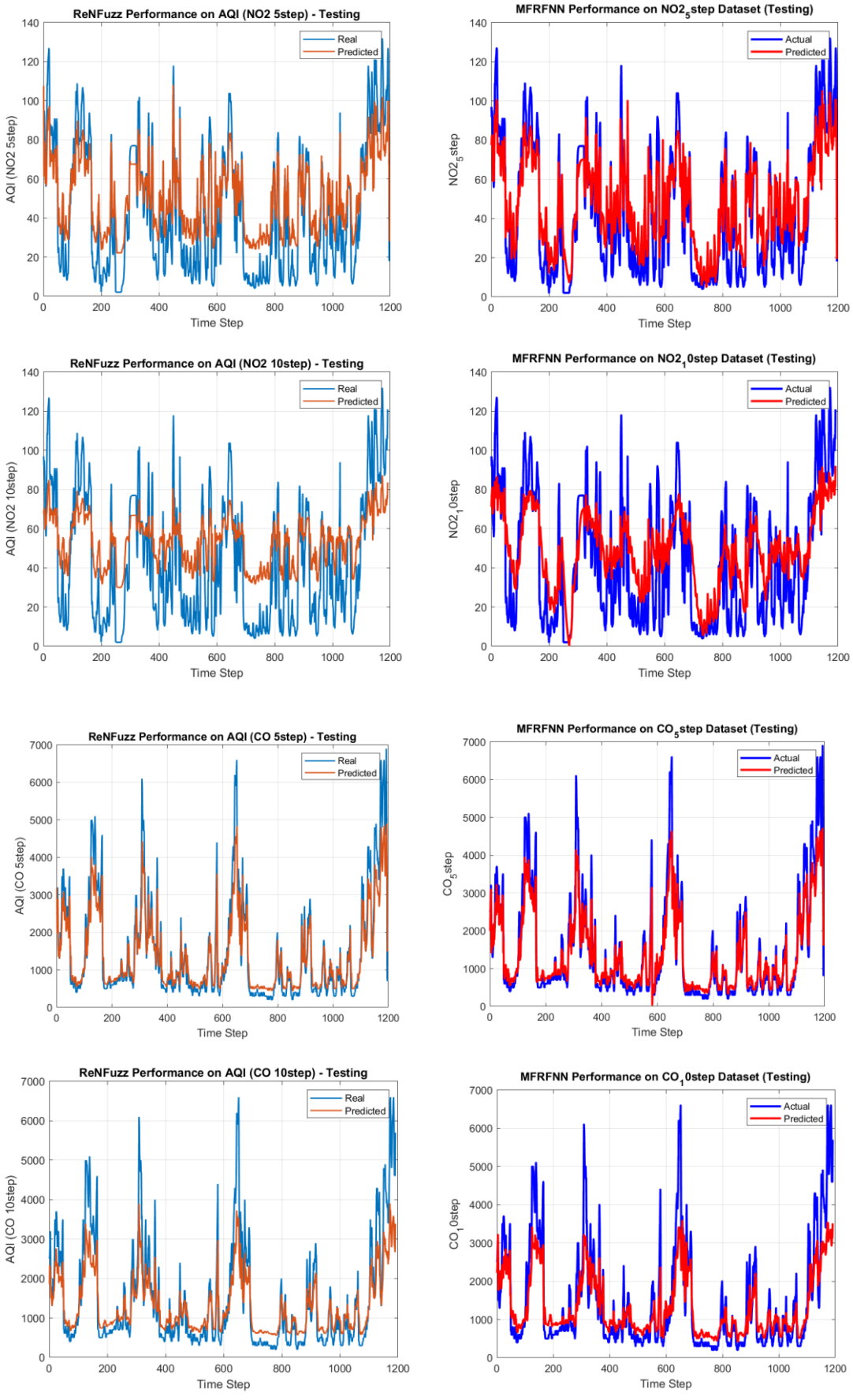


Figure 21: Performance on AQI dataset

Table 5: 5-step ahead AQI errors

Five step ahead prediction error on AQI Data								
Method	**PM2.5**				**PM10**			
	RMSE	***MSE***	***MAE***	***sMAPE***	***RMSE***	***MSE***	***MAE***	***sMAPE***
MFRFNN	50.04	2521.11	14.46	36.76	47.35	2247.80	21.42	38.26
ReNFuzz-LF	32.18	1035.51	18.02	47.87	40.41	1632.99	26.04	44.00
	NO2				**CO**			
	RMSE	***MSE***	***MAE***	***sMAPE***	***RMSE***	***MSE***	***MAE***	***sMAPE***
MFRFNN	13.45	180.87	11.63	40.82	446.99	199803.17	282.77	22.07
ReNFuzz-LF	16.96	287.62	14.91	49.98	503.87	253881.33	332.91	25.70
	SO2				**O3**			
	RMSE	***MSE***	***MAE***	***sMAPE***	***RMSE***	***MSE***	***MAE***	***sMAPE***
MFRFNN	15.15	234.00	4.91	29.67	34.33	1318.15	19.48	51.06
ReNFuzz-LF	11.23	126.16	6.32	36.73	16.46	270.97	13.29	44.98

Table 6: 10-step ahead AQI errors

Ten step ahead prediction error on AQI Data								
Method	**PM2.5**				**PM10**			
	RMSE	***MSE***	***MAE***	***sMAPE***	***RMSE***	***MSE***	***MAE***	***sMAPE***
MFRFNN	50.93	2593.36	22.55	54.11	59.45	3534.54	31.41	49.73
ReNFuzz-LF	38.00	1444.00	24.67	60.66	53.00	2809.30	34.01	54.00
	NO2				**CO**			
	RMSE	***MSE***	***MAE***	***sMAPE***	***RMSE***	***MSE***	***MAE***	***sMAPE***
MFRFNN	20.84	434.10	17.23	51.03	683.44	467093.06	447.23	34.45
ReNFuzz-LF	23.27	541.48	20.55	60.28	670.82	450003.77	452.81	34.64
	SO2				**O3**			
	RMSE	***MSE***	***MAE***	***sMAPE***	***RMSE***	***MSE***	***MAE***	***sMAPE***
MFRFNN	22.44	503.47	8.16	46.56	57.49	3305.58	31.16	64.52
ReNFuzz-LF	10.62	112.85	6.68	41.46	25.43	646.49	20.88	57.89

4.1.6 Electric Load (24 step ahead prediction)

On the other hand, on the Electric Load dataset, ReNFuzz-LF, demonstrates stronger performance, maintaining stable error measures. The improved results on this dataset are likely attribute to the clearer trends in load data compared to environmental data, despite the longer forecasting horizon. The results demonstrate that ReNFuzz-LF significantly outperforms MFRFNN in 24-step-ahead (1-day) electric load forecasting. ReNFuzz-LF achieves an RMSE equal to 124.6 in contrast to the 181.4 of the MFRFNN model. This indicates a better overall prediction accuracy and lower error variance. Additionally, the lower MAE (86.4 vs 117) and sMAPE (1.8 vs. 2.56) highlight its superior precision. It is important to note that while both models perform better on this dataset than the AQI dataset, they continue to struggle with accurately predicting high and low values compared to the mid-range values of the time series.

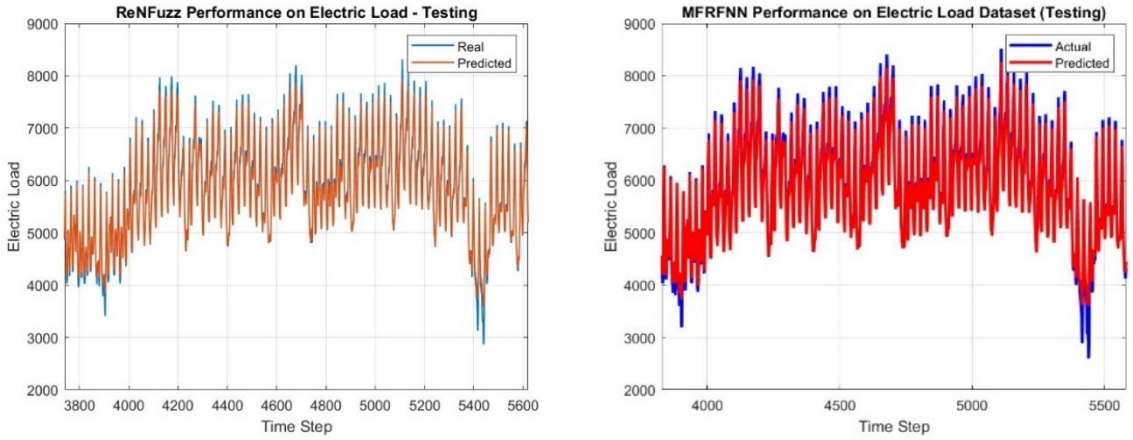


Figure 22: Performance on Electric Load dataset

Table 7: Electric Load problem

24 step ahead prediction error on Electric Load problem			
Method	**RMSE**	**MAE**	**sMAPE**
MFRFNN	186.32	116.98	2.56
ReNFuzz-LF	124.64	86.40	1.80

4.2 Parameters

A recurring theme in the experimental results is the importance of balancing model complexity with the risk of overfitting. In ReNFuzz-LF, the number of hidden neurons in the local RNN consequents must be carefully chosen. For instance, experiments with the Lorenz y(t) prediction indicate that configurations with 4 neurons can lead to oscillatory behaviour (Figure 24), while 3 neurons yield smoother, more stable forecasts. This illustrates that a lean architecture is often preferable when the forecasting task does not demand extensive memory.

Another aspect worth noting is the role of training epochs. For instance, in the Lorenz y(t) and z(t) predictions, both models showed that training beyond a certain number of epochs resulted in diminishing returns, likely due to overfitting or the model reaching an optimal learning plateau. In such cases, the decision to stop training at 250 epochs was justified, as extending training did not yield improved performance and risked degrading the model's ability to generalize. This is a critical consideration for both models, emphasizing the importance of early stopping and validation-based training strategies.

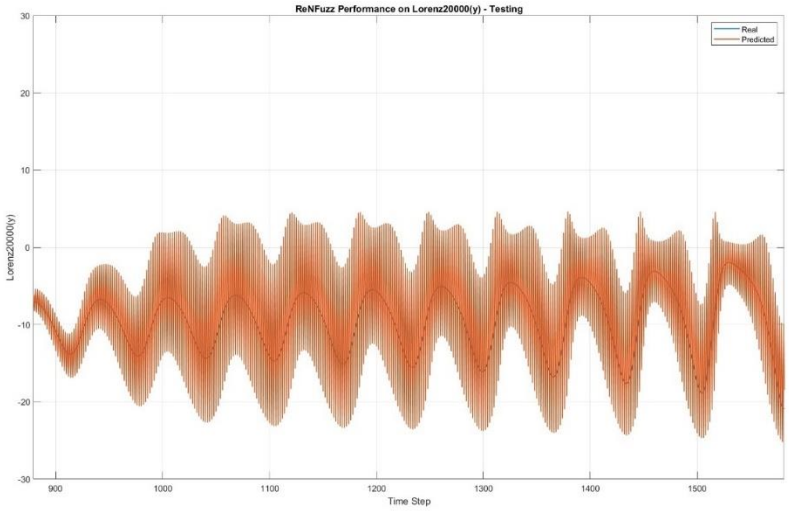


Figure 23: Oscillations (ReNFuzz - Lorenz system)

Table 8: ReNFuzz-LF Parameters

ReNFuzz-LF Parameters			
Benchmark	**Rules**	**Hidden Neurons**	**Input Dimensions**
Google Stock Price	6	2	1
Electric Load	3	2	1
Lorenz x(t)	3	10	1
Lorenz y(t)	3	3	1
Lorenz z(t)	2	2	1
Wind Speed	4	2	2
Box-Jenkins	2	3	2
AQI PM2.5 5-step	4	4	4
AQI PM10 5-step	6	4	4
AQI SO2 5-step	4	10	4
AQI NO2 5-step	6	4	4
AQI CO 5-step	5	2	4
AQI O3 5-step	3	4	4
AQI PM2.5 10-step	6	5	4
AQI PM10 10-step	6	5	4
AQI SO2 10-step	4	5	4
AQI NO2 10-step	6	4	4
AQI CO 10-step	5	3	4
AQI O3 10-step	3	3	4

Table 9: SA-DRPROP Learning Parameters for ReNFuzz-LF

Learning parameters of SA-DRPROP (ReNFuzz-LF)							
Temp	**n+**	**n-**	**Δmin**	**Δmax**	**Δ0**	**α1**	**α2**
1.2	1.05	0.5	0.0001	0.5	0.01	0.01	0.4

MFRFNN, performance is sensitive on the number of fuzzy rules and states. Careful tuning of the rule base and the number of states is essential to fully leverage its ability to learn multiple functions. When the system's state space is adequately represented, MFRFNN can capture the nuances of complex, nonlinear data more effectively. However, an overly complex rule base or state structure may lead to overfitting.

Table 10: MFRFNN Parameters

MFRFNN Parameters					
Benchmark	**K1**	**K2**	**N**	**Maximum Number of FES (PSO Algorithm)**	**Number of input steps**
Air Quality Index	16	16	2	250	4
Lorenz System	27	27	3	500	1
Box-Jenkin Gas Furnace	9	4	2	4000	1
Wind Speed	4	4	2	4000	1
Google Stock Price	3	3	2	4000	1
Electric Load	3	3	2	500	1

Table 11: Normalized Errors

	Normalized Metrics MFRFNN				Normalized Metrics ReNFuzz-LF			
Dataset	***RMSE***	***MSE***	***MAE***	***sMAPE***	***RMSE***	***MSE***	***MAE***	***sMAPE***
Wind Speed	0.0205	0.0004	0.0161	9.8291	0.0551	0.0030	0.0419	23.1692
Box-Jenkins	0.0436	0.0019	0.0285	4.5896	0.0153	0.0002	0.0125	1.8542
Google Stock Price	0.0022	0.0000	0.0009	1.1354	0.0039	0.0000	0.0029	3.7481
Electric Load	0.0258	0.0007	0.0162	4.9526	0.0276	0.0008	0.0192	16.9020
Lorenz x(t)	0.009	0.0001	0.006	1.89	0.0089	0.0001	0.0062	5.9673
Lorenz y(t)	0.01	0.00009	0.006	1.5346	0.0114	0.0001	0.0061	4.6131
Lorenz z(t)	0.0195	0.0004	0.011	2.6	0.0121	0.0001	0.0080	7.5657
AQI PM2.5 5-step	0.0559	0.0031	0.0162	45.8437	0.0575	0.0033	0.0322	6.5446
AQI PM10 5-step	0.0482	0.0023	0.0218	40.6609	0.0658	0.0043	0.0424	7.9938
AQI SO2 5-step	0.0445	0.0020	0.0144	31.1177	0.0527	0.0028	0.0297	5.4423
AQI NO2 5-step	0.0467	0.0022	0.0404	45.9219	0.0942	0.0089	0.0828	16.1827
AQI CO 5-step	0.0452	0.0020	0.0286	26.1955	0.0814	0.0066	0.0538	18.1831
AQI O3 5-step	0.0812	0.0074	0.0461	51.4735	0.0623	0.0039	0.0503	7.9950
AQI PM2.5 10-step	0.0569	0.0032	0.0252	63.6104	0.0679	0.0046	0.0441	8.6228
AQI PM10 10-step	0.0605	0.0037	0.0320	52.2109	0.0864	0.0075	0.0554	9.7293
AQI SO2 10-step	0.0659	0.0043	0.0239	48.2281	0.0499	0.0025	0.0314	5.6383
AQI NO2 10-step	0.0723	0.0052	0.0598	56.5346	0.1293	0.0167	0.1142	21.8999
AQI CO 10-step	0.0690	0.0048	0.0452	39.9908	0.1084	0.0118	0.0732	20.8772
AQI O3 10-step	0.1360	0.0185	0.0737	64.9483	0.0962	0.0093	0.0790	12.3967

5 Conclusion and Future Directions

In conclusion, our findings show that ReNFuzz-LF and MFRFNN each have distinct strengths depending on the forecasting task. ReNFuzz-LF offers a streamlined, efficient approach without excessive computational demands. With it's RNN consequents, it excels in electric load forecasting as well as in capturing the short-term dynamics of chaotic systems like the Lorenz system. In contrast, MFRFNN, is designed to address complex, multi-state problems by determining the state of then network and making predictions with its dual-network feedback architecture. With this architecture, it was able to capture the unpredictability of wind speed and volatility of stock prices. Overall, both models were able to capture the patterns of all datasets, however, both face challenges in long-term forecasting on environmental data, most likely due to error accumulation over multiple prediction steps. Additionally, in many cases MFRFNN tend to overestimate outliers, and both models had issues with capturing low and high values in data while making great predictions on mid-range values. Although extended forecasting remains difficult, the insights into parameter tuning, training strategies, and architectural design provide clear avenues for further refinement. These findings underscore that selecting the appropriate model depends critically on the dataset's structure and the forecasting horizon, and they offer a roadmap for future enhancements in neurofuzzy time series prediction.

Future research may focus on integrating advanced techniques such as attention mechanisms[22], or adaptive normalization[23] strategies to further enhance long-term predictive performance. Additionally, when a model consistently underestimates high values or overestimates low values, this may indicate that the fuzzy membership functions do not adequately cover the tails of the data distribution, or that the loss function does not sufficiently penalize errors at these extremes. One potential refinement is to modify the membership functions by incorporating additional, specialized fuzzy sets that focus specifically on the high and low ends of the data range. By assigning extra fuzzy sets to the tail regions, the model may gain a more granular representation of rare events, ensuring that these extremes are better captured during inference. These refinements, whether through enhanced membership function design or loss function reweighting, provide targeted mechanisms for addressing the common challenge of accurately predicting high and low extremes in time series data.

6. References

[1] Y. Bengio, P. Simard, and P. Frasconi, "Learning Long-Term Dependencies with Gradient Descent is Difficult," *IEEE Trans Neural Netw*, vol. 5, no. 2, pp. 157–166, Mar. 1994, doi: https://doi.org/10.1109/72.279181.

[2] H. Nasiri and M. M. Ebadzadeh, "MFRFNN: Multi-Functional Recurrent Fuzzy Neural Network for Chaotic Time Series Prediction," *Neurocomputing*, vol. 507, pp. 292–310, Oct. 2022, doi: https://doi.org/10.1016/j.neucom.2022.08.032.

[3] G. Kandilogiannakis, P. Mastorocostas, and A. Voulodimos, "ReNFuzz-LF: A Recurrent Neurofuzzy System for Short-Term Load Forecasting," *Energies (Basel)*, vol. 15, no. 10, p. 3637, May 2022, doi: https://doi.org/10.3390/en15103637.

[4] E. Rich, "Artificial Intelligence and the Humanities," *Comput Hum*, vol. 19, no. 2, pp. 117–122, 1985, [Online]. Available: http://www.jstor.org/stable/30204398

[5] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955," *AI Mag*, vol. 27, no. 4, pp. 12–14, 2006, doi: https://doi.org/10.1609/aimag.v27i4.1904.

[6] R. M. Schmidt, "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview," Nov. 2019, [Online]. Available: http://arxiv.org/abs/1912.05911

[7] 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: https://doi.org/10.1109/5.58337.

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: https://doi.org/10.1162/neco.1997.9.8.1735.

[9] P. Mastorocostas, *Fuzzy and Neurofuzzy Systems-Fuzzy Deep Learning[Postgraduate textbook]*. Kallipos, Open Academic Editions, 2022. doi: http://dx.doi.org/10.57713/kallipos-146.

[10] L. A. Zadeh, "Fuzzy Sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, Jun. 1965, doi: https://doi.org/10.1016/S0019-9958(65)90241-X.

[11] L. A. Zadeh, "Fuzzy Logic," *Computer (Long Beach Calif)*, vol. 21, no. 4, pp. 83–93, Apr. 1988, doi: https://doi.org/10.1109/2.53.

[12] E. H. Mamdani, "Application of fuzzy algorithms for control of simple dynamic plant," *Proceedings of the Institution of Electrical Engineers*, vol. 121, no. 12, pp. 1585–1588, Dec. 1974, doi: https://doi.org/10.1049/piee.1974.0328.

[13] T. Takagi and M. Sugeno, “Fuzzy Identification of Systems and Its Applications to Modeling and Control,” *IEEE Trans Syst Man Cybern*, vol. SMC-15, no. 1, pp. 116–132, Jan. 1985, doi: https://doi.org/10.1109/TSMC.1985.6313399.

[14] J.-S. R. Jang, “ANFIS: Adaptive-Network-Based Fuzzy Inference System,” *IEEE Trans Syst Man Cybern*, vol. 23, no. 3, pp. 665–685, May 1993, doi: https://doi.org/10.1109/21.256541.

[15] R. Adhikari and R. K. Agrawal, *An Introductory Study on Time Series Modeling and Forecasting*. 2013. doi: https://doi.org/10.48550/arXiv.1302.6613.

[16] A. N. Desai *et al.*, “Real-time Epidemic Forecasting: Challenges and Opportunities,” *Health Secur*, vol. 17, no. 4, pp. 268–275, Aug. 2019, doi: https://doi.org/10.1089/hs.2019.0022.

[17] R. L. R. Salcedo, M. C. M. A. Ferraz, C. A. Alves, and F. G. Martins, “Time-series analysis of air pollution data,” *Atmos Environ*, vol. 33, no. 15, pp. 2361–2372, Jul. 1999.

[18] C. Oestreicher, “A History of Chaos Theory,” *Dialogues Clin Neurosci*, vol. 9, no. 3, pp. 279–289, Sep. 2007, doi: https://doi.org/10.31887/DCNS.2007.9.3/coestreicher.

[19] Z. Liu, “Chaotic Time Series Analysis,” *Math Probl Eng*, vol. 2010, no. 1, p. 720190, 2010, doi: https://doi.org/10.1155/2010/720190.

[20] N. K. Treadgold and T. D. Gedeon, “Simulated Annealing and Weight Decay in Adaptive Learning: The SARPROP Algorithm,” 1998.

[21] J. Kennedy and R. Eberhart, “Particle Swarm Optimization,” *Proceedings of ICNN’95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948, Nov. 1995, doi: https://doi.org/10.1109/ICNN.1995.488968.

[22] G. Brauwers and F. Frasincar, “A General Survey on Attention Mechanisms in Deep Learning,” *IEEE Transactions on Knowledge and Data Engineering* , vol. 35, no. 4, pp. 3279–3298, Nov. 2021, doi: https://doi.org/10.1109/TKDE.2021.3126456.

[23] E. Ogasawara, L. C. Martinez, D. De Oliveira, G. Zimbrao, G. L. Pappa, and M. Mattoso, “Adaptive Normalization: A Novel Data Normalization Approach for Non-Stationary Time Series,” *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, Jul. 2010, doi: https://doi.org/10.1109/IJCNN.2010.5596746.