Persian Gulf University

DEPARTMENT OF FACULTY OF INTELLIGENT SYSTEMS ENGINEERING AND DATA SCIENCE

Prediction of Chaotic Time Series with Echo State Network

Supervisors:

Dr. Ebrahim Sahafizade

Dr. Habib Rostami

Advisor:

Dr. Ahmad Shirzadi

Author:

Leila Gonbadi

Computer Science, Artificial Intelligence

A thesis submitted in fulfilment of the requirements for the degree of Master, April 2025

Certificate

It is certified that the work contained in this thesis entitled "Prediction of Chaotic Time Series with Echo State Network" by "Leila Gonbadi" has been carried out under my supervision and that it has not been submitted elsewhere for a degree.

Dr. Ebrahim Sahafizade	Assistant Professor	Computer Engineering	Supervisor
Dr. Habib Rostami	Associate Professor	Computer Engineering	Supervisor
Dr. Ahmad Shirzadi	Associate Professor	Mathematics	Advisor

April 2025

Abstract

Echo State Networks (ESNs) have emerged as powerful tools for time series prediction, yet their performance heavily depends on reservoir initialization, which traditionally relies on random weights independent of input data characteristics. This paper presents a novel theoretical framework and optimization approach for ESN reservoir design, demonstrating that reservoir weights should be adapted based on input data properties, and that both topology and weights significantly influence prediction accuracy. Building on theoretical insights about input-dependent reservoir behavior, we propose two complementary methods: a supervised approach that directly optimizes reservoir weights through gradient descent, and a semi-supervised technique that combines small-world and scale-free network properties with hyperparameter optimization. Our extensive experiments across multiple datasets, including Mackey-Glass and NARMA time series, demonstrate that the proposed methods consistently outperform traditional ESNs by achieving substantially lower prediction errors. Most notably, our analysis reveals that edge connectivity parameters play a crucial role, second only to reservoir size in determining network performance. These findings provide important practical guidelines for ESN design and open new directions for automated reservoir optimization based on input data characteristics.

Keywords: Echo State Networks, Reservoir Computing, Network Topology, Time Series Prediction, Optimization, Chaotic

Acknowledgements

I would like to express my heartfelt gratitude to my dear sister, Dr. Maryam Gonbadi, whose unwavering encouragement and support have been a source of inspiration throughout my life. Her belief in my abilities has motivated me to strive for excellence in my academic journey.

I am also deeply thankful to my cherished classmates, who became like a family to me during this program. Their companionship, collaboration, and kindness created an environment of mutual support and made this journey both meaningful and memorable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Time series prediction plays a critical role in many scientific and industrial fields, serving as the backbone for decision-making processes and forecasting in diverse areas such as finance, meteorology, healthcare, and energy management[25],[20],[22],[31]. Among these, chaotic time series, characterized by their nonlinearity and sensitivity to initial conditions, present unique challenges[21]. These datasets often appear in real-world applications, including weather forecasting, biological systems, and engineering dynamics, where traditional linear models fail to capture the complex underlying patterns accurately[29]. The increasing demand for precise forecasting of such intricate data highlights the need for advanced methodologies capable of addressing these challenges[25].

Echo State Networks (ESNs), with their reservoir-based architecture, have shown considerable potential in modeling and predicting chaotic time series[12]. Unlike conventional models, ESNs leverage their dynamic reservoir to capture temporal dependencies and nonlinearities effectively[18]. However, their reliance on randomly assigned reservoir weights has often led to suboptimal performance[4]. Consequently, improving ESN architectures and optimization strategies has become a focal point for researchers aiming to enhance time series prediction accuracy, particularly in chaotic environments[19].

1.2 Research Problem

Despite the proven capability of ESNs to model chaotic time series[12], several limitations persist. Random initialization of reservoir weights often results in inconsistent performance[4], and the exclusive focus on optimizing the output layer neglects the potential of internal reservoir dynamics[8]. These challenges are further exacerbated when scaling ESNs to handle high-dimensional, chaotic datasets with intricate dependencies[17]. Existing optimization approaches, while fast and straightforward, struggle to adapt to the complexities of chaotic systems, leading to high error rates and limited generalizability[19]. Thus, there is a pressing need for robust optimization frameworks and innovative architectural designs to unlock the full potential of ESNs for chaotic time series prediction[15].

1.3 Significance and Necessity

The ability to accurately predict chaotic time series has profound implications for both scientific inquiry and practical applications[21]. In meteorology, reliable weather forecasting can save lives and resources by enabling better disaster preparedness. In healthcare, precise modeling of physiological signals such as heart rate dynamics can lead to improved diagnostics and treatments[21]. Similarly, in engineering, understanding turbulent fluid dynamics can optimize industrial processes. The development of advanced ESN models and optimization techniques can bridge the gap between the complexity of chaotic datasets and the need for accurate predictions[29], offering transformative solutions across various domains. Furthermore, by addressing the limitations of current ESN approaches, this research can contribute to advancing machine learning methodologies and expanding their applications to other complex systems[14].

1.4 Research Objectives

The goal of this research is to improve the performance of Echo State Networks (ESNs) in predicting chaotic time series by exploring methods for optimizing the network's randomly assigned reservoir weights[4]. By examining and applying different optimization techniques, this study aims to develop more accurate and reliable models capable of handling the complexity of chaotic data[19]. Ultimately, the research seeks to advance machine learning approaches for time series prediction and demonstrate their practical applications in various fields[21].

Chapter 2

Preliminaries and Literature Review of ESN

2.1 Preliminaries of Echo State Networks (ESNs)

Echo State Networks (ESNs) are a type of recurrent neural network with a unique architectural approach. The network consists of three main components: an input layer, a large randomly connected hidden layer called the "reservoir," and an output layer. The key innovation is that only the output connections are trained, while the input and reservoir connections remain fixed with their random initial weights. The reservoir acts as a dynamic memory, creating complex transformations of the input signal through its recurrent connections. This approach drastically simplifies training while maintaining powerful computational capabilities, as the reservoir naturally creates rich temporal patterns that can be combined linearly at the output[12][11].

The simplicity and efficiency of ESNs arise from the *echo state property*, which ensures that the reservoir's state depends only on the input history and not on initial conditions. The reservoir essentially "echoes" the input through its internal dynamics, enabling the network to represent temporal dependencies effectively. The state of the reservoir, denoted as $x(n)$, is updated according to the equation:

$$x(n+1) = F(W_{\text{in}} \cdot u(n) + W_{\text{res}} \cdot x(n) + W_{\text{back}} \cdot y(n)) \tag{2.1}$$

$$y(n+1) = W_{\text{out}} \cdot \begin{bmatrix} u(n) \\ x(n+1) \end{bmatrix} \tag{2.2}$$

Equation 2.1 governs the reservoir state evolution. The next state x(n+1) integrates three components:

- Input weight: W_{in}
- Input influence: $u(n)$
- reservoir weight: W_{res}
- Internal dynamics: $x(n+1)$
- feedback weight: W_{back}
- Last Output : $y(n)$

These components undergo transformation through a nonlinear activation function F (typically tanh or ReLU), enabling the modeling of complex patterns. The reservoir state effectively serves as a memory mechanism, capturing both input influences and internal dynamic patterns.

Equation 2.2 describes the output generation process, where W_{out} operates on the concatenated vector of current input and reservoir state.

- Output: $y(n+1)$
- Output weight: W_{in}

The design of ESNs combines simplicity with effectiveness, enabling them to excel in tasks requiring temporal pattern recognition and modeling. They are particularly useful in chaotic time series prediction, system control, and real-time signal processing. By leveraging the reservoir's ability to generate rich temporal dynamics, ESNs provide an efficient solution for many nonlinear modeling challenges [12].

This introduction provides a foundation for understanding the principles and mechanisms of ESNs, setting the stage for deeper exploration of their theoretical underpinnings and

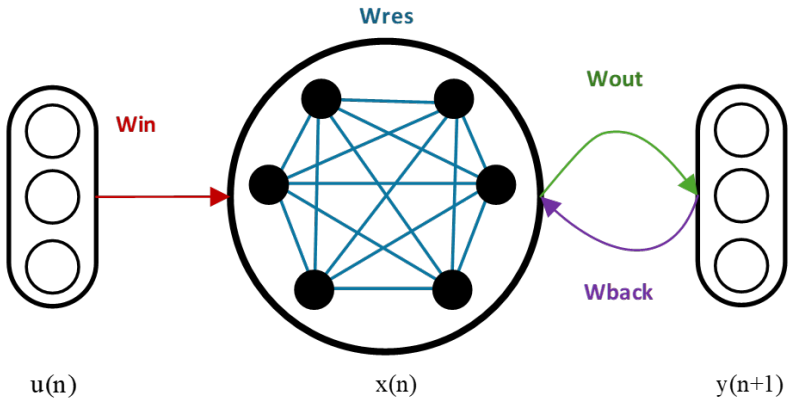

FIGURE 2.1: A schematic representation of an Echo State Network showing the flow of data from the input layer, through the reservoir, to the output layer. This visual aligns with the state update equation $x(n+1)$ and the output calculation equation $y(n+1)$.

practical applications. To provide a systematic overview of the research background, we categorize previous studies into four main sections as follows:

The first section focuses on implementing ESNs in advanced applications across diverse domains. These implementations have demonstrated the networks' versatility in addressing complex challenges and their adaptability to various real-world problems.

The second section encompasses research dedicated to structural modifications of ESNs. These studies have introduced innovative architectural changes to enhance network performance and capabilities, leading to significant improvements in prediction accuracy and computational efficiency.

Research in the third section has concentrated on developing and optimizing data feeding methods for ESNs. These studies have explored various techniques for data preprocessing and input optimization, leading to improved network performance across different applications.

The fourth section involves investigations into mathematical concepts and parameter impacts on ESN outputs. These studies have provided deeper insights into the theoretical foundations of ESNs and how various parameters influence their performance.

2.2 Using Echo State Networks for Advanced Applications

Echo State Networks (ESNs) have demonstrated remarkable versatility in addressing diverse and complex applications[12]. Researchers have explored their potential in various fields, utilizing their ability to model and predict nonlinear and chaotic systems effectively[21].

In one study, ESNs were applied to forecast chaotic time series, emphasizing their utility in predicting dynamic systems with nonstationary and stochastic components. By separating time series data into trend, seasonality, and residual elements, the study enhanced the network's capability to handle unpredictability and long-term dependencies[24].

Another research effort focused on multivariate time series prediction, employing ESNs to address challenges in datasets with high-dimensional and noisy features[29]. This work highlighted ESNs' robustness in chaotic and multivariate environments, making them particularly suitable for applications involving complex interdependencies[10].

ESNs were also used for seasonal streamflow prediction in hydrological systems, where they excelled at modeling nonlinear patterns and capturing temporal dynamics. This application is crucial for water resource management, where accurate forecasting supports planning and decision-making. Comparative evaluations with other methods demonstrated ESNs' superior adaptability in such critical contexts[15].

Lastly, ESNs were evaluated for chaotic time series forecasting alongside other neural network models[25]. The study explored their effectiveness in predicting the behavior of dynamic systems with limited data availability. This research demonstrated ESNs' capability in achieving reliable short-term forecasts, particularly for systems where data is sparse or chaotic[19].

2.3 Structural Modifications of Echo State Networks (ESNs)

2.3.1 Single reservoir

The evolution of reservoir computing structures has witnessed significant advancements through innovative architectural designs. Researchers [6] introduced a scale-free highly-clustered echo state network (SHESN) that integrates the properties of small-world and scale-free networks. This architecture comprises three layers: an input layer, a naturally

evolving state reservoir, and an output layer. The reservoir is designed with thousands of sparsely interconnected neurons organized hierarchically into domains, each containing a backbone neuron and several local neurons. This hierarchical and distributed structure represents a key milestone in early reservoir computing designs.

Building on these foundational concepts, researchers [15] proposed the Robust Echo State Network (RESN), which preserves the traditional three-layer ESN architecture but incorporates a Bayesian framework with a Laplace likelihood function. This enhancement emphasizes robustness against outliers, introducing a novel training mechanism specifically tailored to handle noisy or contaminated data while maintaining sparse reservoir connectivity.

Continuing the trend of architectural refinement, researchers [18] advanced clustered reservoir designs by developing three distinct models: SCESN, SPESN, and SWESN. Each model leverages a specific clustering algorithm:

SCESN: Utilizes K-Means (C-Centroid) clustering. SPESN: Implements Partitioning Around Medoids (PAM). SWESN: Adopts the Ward hierarchical clustering algorithm. These structures integrate principles from complex network theory, such as power-law distributions and high clustering coefficients, to enhance network performance. Between 2016 and 2018, reservoir computing architectures saw further diversification with innovations aimed at improving robustness and scalability. For instance, the Adaptive Elastic Echo State Network (AEESN) introduced in 2016 combines a linear readout layer with a randomly generated reservoir structure [28]. By utilizing the adaptive elastic net algorithm, this architecture effectively addresses collinearity issues through quadratic regularization and adaptive lasso shrinkage.

The following year marked the emergence of the Deep Belief Echo State Network (DBEN), which integrated deep learning principles with reservoir computing for the first time [24]. DBEN employs a deep belief network (DBN) for unsupervised feature learning and incorporates an innovative regression layer that replaces traditional backpropagation with an echo-state learning mechanism.

In 2018, two notable architectures were introduced. The Laplacian Echo State Network (LAESN) employed manifold learning using Laplacian eigenmaps to better capture reservoir dynamics [10], while the Polynomial Echo State Network (PESN) extended ESNs by incorporating polynomial functions of input variables into output weights [30]. Both innovations aimed to handle high-dimensional data more efficiently while retaining computational simplicity.

In 2020, clustered reservoir designs gained prominence with architectures featuring high clustering coefficients and deep network structures [1][13]. These designs incorporated backbone neurons surrounded by local nodes, enhancing information flow and network stability. The development of Deep ESNs further extended these principles by stacking multiple reservoir layers, enabling better management of complex temporal dependencies [13].

The year 2021 saw a focus on optimizing input weight matrices using selective opposition-based strategies [4]. This innovation restructured input connections while preserving the echo state property, leading to improved prediction accuracy for chaotic time series.

The most recent advancements in reservoir computing emphasize hierarchical architectures and hardware efficiency. The foundational Echo State Network (ESN) structure comprising an input layer, a reservoir layer with randomly fixed hidden neurons, and a linear output laye has been enhanced through hierarchical designs and optimized computation strategies [14].

One significant innovation is the Hierarchical Echo State Network with Sparse Learning (HESN-SL), which connects multiple reservoirs in sequence for layer-by-layer processing. Sparse learning in the output layer eliminates redundant components from random projections, allowing for efficient hardware implementations while maintaining accuracy [17]. In addition, the network limits reservoir neurons to four inputs, further simplifying hardware requirements.

Complex architectures like Long Short-Term Memory (LSTM) networks and LSTM Encoder-Decoder (LSTM ED) frameworks have also been explored. These architectures leverage gating mechanisms such as input, forgetting, and output gates to enhance long-range dependency handling [25]. The encoder-decoder structure in LSTM ED further refines input sequence representation into fixed-length internal states.

Finally, significant progress has been made in hardware implementations. Recent work highlights the successful use of FPGA-based architectures with optimized hyperbolic tangent approximations and fixed-point arithmetic [5], achieving a balance between computational efficiency and prediction accuracy.

2.3.2 Multi reservoir

In 2017, researchers introduced a double-reservoir architecture for echo state networks (ESNs) to handle multi-regime time series prediction. This structure utilized two separate reservoirs - one dedicated to processing sensor measurements and another for regime parameters. The dual reservoir approach allowed the network to simultaneously model different aspects of the input data, with both reservoirs contributing to the final prediction through an aggregated output layer. This modification proved particularly effective for systems operating under dynamic conditions [32].

By 2020, the focus shifted to multi-reservoir architectures capable of handling high variability in physical implementations. This approach moved beyond single-reservoir computing to incorporate multiple interconnected reservoirs, creating deeper architectures that could better process complex temporal patterns. The key innovation was the development of training methods that could effectively coordinate multiple reservoirs without requiring direct error backpropagation [8].

Most recently (2024), research has examined the impact of modifying internal network structures through time-history terms. This work explored how structural elements like leak rates and decay factors within reservoir neurons affect the network's temporal processing capabilities. The introduction of these structural modifications helped maintain both diversity and stability in reservoir dynamics while enhancing memory capacity [7].

2.4 Data Feeding Methods in Echo State Networks

The first approach introduced robust optimization and validation techniques for ESNs learning chaotic dynamics [19]. This work established several validation strategies including Single Shot Validation, Walk Forward Validation, and K-Fold Cross Validation, along with a novel Recycle Validation approach. Their network utilized both model-free and model-informed architectures, with data being fed through a random input matrix to a large dynamic reservoir.

Building upon these foundations, a subsequent study presented a feature-based approach to ESNs that prioritized interpretability and efficiency [9]. This newer method introduced a systematic way of processing input data by decomposing it into features using a feature matrix. Instead of feeding data into a single large reservoir, the system employs multiple smaller parallel reservoirs, each processing different input features. The architecture

proved particularly effective for handling partial observations through delay-embedded inputs.

The evolution between these papers demonstrates a shift from focusing on validation methods to developing more interpretable and efficient data processing architectures. While the first study emphasized robust validation strategies for improving prediction accuracy [19], the later research showed how feature-based data feeding could achieve similar or better results with significantly reduced computational complexity [9]. Both approaches maintain the core ESN principle of reservoir computing but differ in how they process and validate input data.

2.5 Mathematical Concepts and Parameter Impacts on ESN Outputs

Echo State Networks (ESNs) process time series data through three interconnected layers: input, reservoir, and output. The mathematical foundation centers on a hidden state formula:

$$h(k) = \tanh(W_{\text{in}} x(k) + W_r s(k-1)) \tag{2.3}$$

which combines input weights (W_{in}) and reservoir weights (W_r) to process temporal information [26].

The reservoir size significantly influences network performance, though larger reservoirs do not necessarily guarantee better results. Research has demonstrated performance improvements ranging from 32% to 100% when optimizing reservoir size for specific datasets. The density of reservoir weights also proves crucial, with sparse matrices often outperforming dense ones, leading to performance improvements between 9% and 90% [26].

The spectral radius, which controls the echo state property, shows performance gains of up to 45%. The study challenges the traditional constraint of keeping the spectral radius below 1, finding that some datasets benefit from larger values. The leakage rate, managing information flow between states, demonstrates substantial impact with performance gains between 21% and 90%, with optimal rates varying between 41% and 88% depending on the dataset characteristics [26].

Regularization through a coefficient β proves beneficial even in small amounts, with performance improvements of 1% to 10%. The research consistently shows that weak regularization (approximately 10^{-7}) enhances model generalization across different types of data.

When combined, proper tuning of these parameters leads to remarkable improvements in ESN performance, ranging from 85.1% to 99.8% compared to default configurations [26].

The systematic evaluation reveals how each parameter influences ESN behavior and output quality. The findings emphasize that parameter optimization significantly reduces prediction variability across different random initializations, making ESN performance more reliable and consistent. This comprehensive understanding of parameter relationships proves essential for achieving optimal performance in time series prediction tasks [26].

Chapter 3

Materials and Methods

3.1 Introduction

This chapter provides a comprehensive explanation of the materials and methods used in this research. It aims to ensure the reproducibility of the study by detailing the datasets, preprocessing techniques, and methodologies employed for modeling and analysis. By adhering to systematic and well-documented procedures, this study seeks to enhance the reliability of the findings. The chapter is organized into two main sections: About Data, which describes the datasets and their characteristics, and About Method, which outlines the techniques and algorithms used to achieve the research objectives.

3.2 About Data

In this section, we explain the datasets used in this study, including their characteristics and preprocessing steps.

3.2.1 Mackey-Glass Time Series

The dataset used in this study consists of time series generated using the Mackey-Glass equation, a nonlinear time delay differential equation known for producing chaotic behavior. Two distinct time series were generated using different time delay parameters (τ) to analyze the system's behavior under varying conditions.

3.2.1.1 Mackey-Glass Equation

The Mackey-Glass equation is defined as:

$$\frac{dx}{dt} = \beta \frac{x(t-\tau)}{1+x(t-\tau)^n} - \gamma x(t), \tag{3.1}$$

Where $x(t)$ is the value of the time series at time t, τ is the time delay parameter, β and γ are system parameters, and n is the nonlinearity parameter.

3.2.1.2 Dataset Characteristics

Two distinct time series were generated with the following parameters: 1. First series: $\tau = 29$ (strongly chaotic regime) 2. Second series: $\tau = 17$ (moderately chaotic regime)

Common parameters for both series: sequence length: 20,000 points, initial washout period: 10% of sequence length, sampling rate: 10 increments per time unit, and initial conditions: $x(t) = 1.2 + 0.2\xi$ for $t \leq 0$, where $\xi \sim U(-0.5, 0.5)$.

3.2.1.3 Time Series Analysis

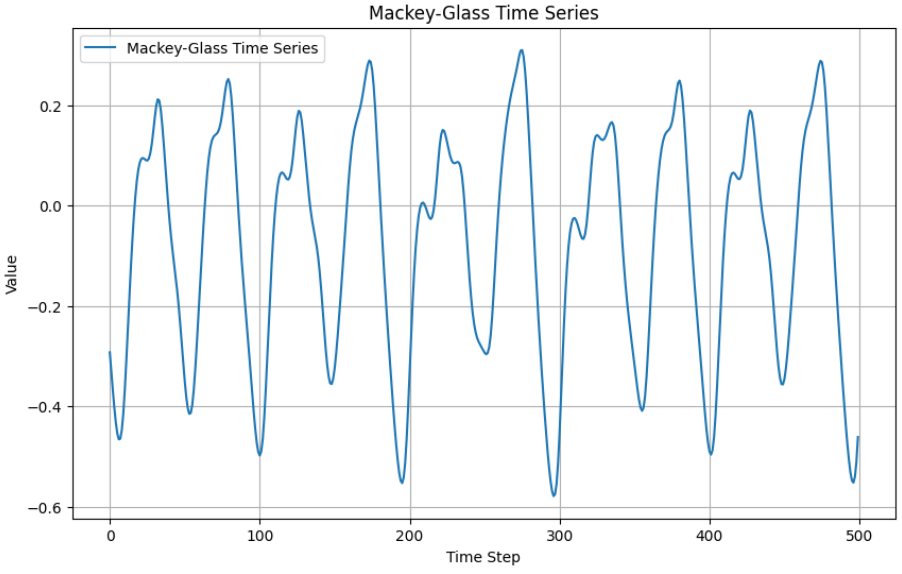

FIGURE 3.1: Mackey-Glass Time Series with $\tau = 17$

Figure 3.1 shows the time series generated with $\tau = 17$. This moderately chaotic system exhibits:

- Regular oscillatory patterns with predictable short-term behavior
- Lower dimensional attractor structure with stable periodic components
- Moderate complexity suitable for initial testing of prediction algorithms
- Clear signal structure with relatively consistent amplitude variations
- More pronounced periodicity compared to higher τ values

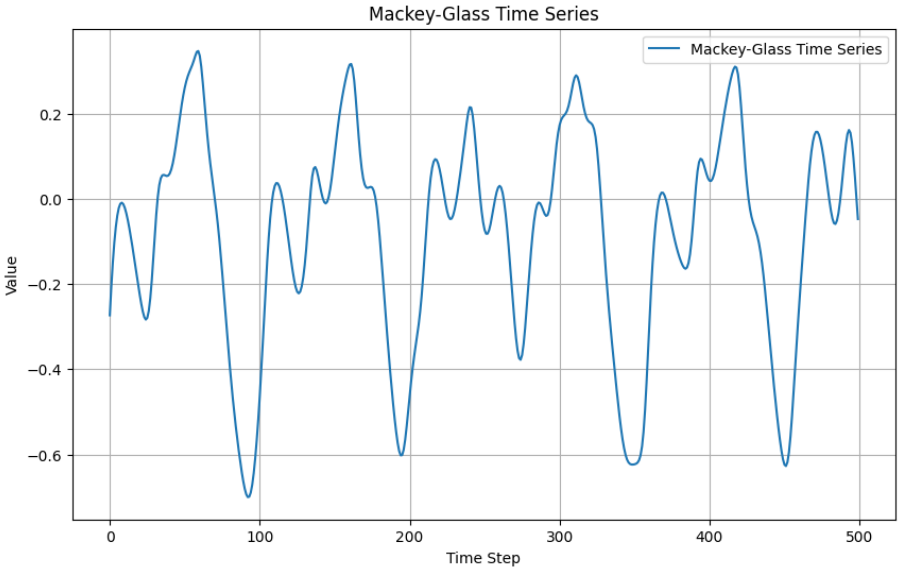

FIGURE 3.2: Mackey-Glass Time Series with $\tau = 29$

Figure 3.2 presents the time series generated with $\tau = 29$. This configuration demonstrates:

- Highly chaotic behavior with irregular oscillations and complex patterns
- Higher dimensional attractor structure with sophisticated nonlinear dynamics
- Limited predictability over longer time horizons due to chaos
- Greater amplitude variations and more frequent extreme values
- Rich nonlinear interactions making it challenging for prediction tasks

3.2.1.4 Pseudocode

The generated datasets provide two distinct scenarios for testing prediction algorithms:

Algorithm 1 Mackey-Glass Time Series Generation

procedure GENERATEMGTIMESERIES($sequenceLength, \tau, dimensions$)
  Initialize history buffer of length $\tau \times 10$
  Set initial conditions to 1.2 with small random perturbations
  $washoutLength \leftarrow sequenceLength/10$
  for $n \leftarrow 1$ to $sequenceLength + washoutLength$ **do**
    for $i \leftarrow 1$ to 10 **do** ▷ 10 increments per time unit
      $tauValue \leftarrow$ history[$step$ mod $historyLength$]
      $newValue \leftarrow oldValue + \frac{0.2 \times tauValue}{1+tauValue^{10}} - 0.1 \times oldValue$
      Update history buffer
    end for
    if $n > washoutLength$ **then**
      Store $newValue$ in output sequence
    end if
  end for
  return input and output sequences
end procedure

- The $\tau = 17$ series offers a moderately complex case suitable for initial algorithm validation and baseline performance assessment
- The $\tau = 29$ series presents a more challenging scenario for evaluating algorithm robustness and performance under highly chaotic conditions

3.2.2 NARMA Time Series

This study utilizes the Nonlinear AutoRegressive Moving Average (NARMA) time series, a benchmark problem in time series prediction and system identification. The NARMA model is particularly valuable for testing the memory capability and nonlinear computation power of prediction algorithms.

3.2.2.1 NARMA System Description

The NARMA system is defined as a discrete-time temporal task where the next value in the sequence depends on both previous system outputs and external inputs. The system is described by:

$$y(t+1) = 0.7u(t-n) + 0.1 + (1 - y(t))y(t), \tag{3.2}$$

Where $y(t)$ is the system output at time t, $u(t)$ is the system input at time t, and n is the memory length (time delay).

3.2.2.2 Dataset Characteristics

The NARMA time series was generated with the following parameters: sequence length: 1,000 points, memory length: 10 time steps, input range: random values uniformly distributed in $[0, 1]$, and initial conditions: $y(0) = 0.1$.

Key features of the system: - Nonlinear input-output relationships - Memory-dependent behavior - Complex temporal dependencies - Inherent difficulty in long-term prediction

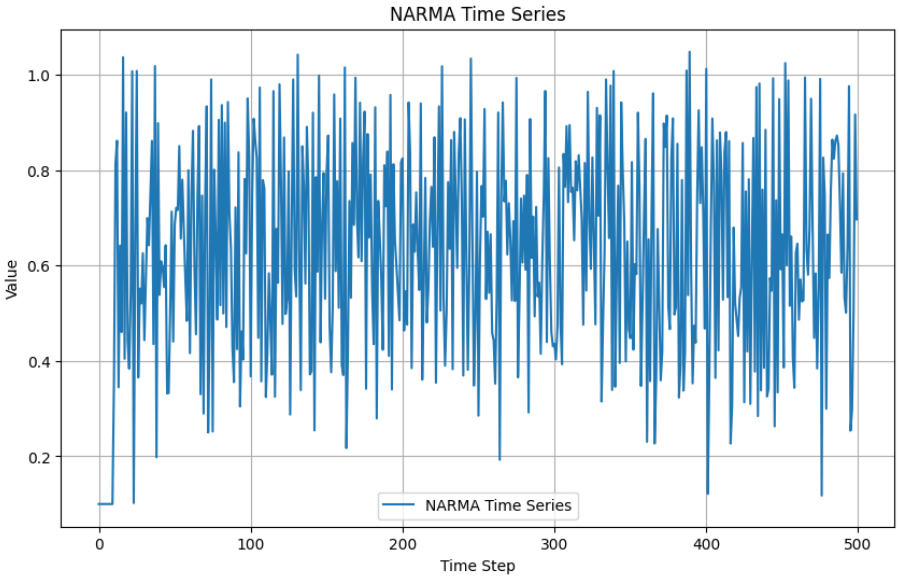

FIGURE 3.3: NARMA Time Series with Memory Length = 10

Figure 3.3 demonstrates the characteristics of the NARMA time series:

- Non-uniform oscillations driven by random input sequence
- Complex temporal dependencies due to the memory term
- Nonlinear amplification effects visible in output variations
- Bounded output behavior due to the system's structure
- Clear demonstration of the system's memory effects

Algorithm 2 NARMA Time Series Generation

procedure GENERATENARMASEQUENCE($sequenceLength, memoryLength$)
    Initialize input sequence with random values $\in [0, 1]$
    Initialize output sequence with 0.1
    for $t \leftarrow memoryLength$ to $sequenceLength$ **do**
        $u \leftarrow \text{input}[t - memoryLength]$
        $y \leftarrow \text{output}[t - 1]$
        $\text{output}[t] \leftarrow 0.7 \times u + 0.1 + (1 - y) \times y$
    end for
    return input and output sequences
end procedure

3.2.2.3 Pseudocode

3.2.2.4 System Properties

The NARMA system presents several challenges for prediction tasks:

- The memory requirement makes it difficult for systems with limited temporal processing capability
- The nonlinear interactions between current state and input create complex dynamics
- The random input sequence ensures unique patterns in each generation
- The system exhibits both short-term dependencies through the nonlinear term and long-term dependencies through the memory term

The NARMA dataset serves as an excellent benchmark for:

- Evaluating a model's ability to capture temporal dependencies
- Testing nonlinear processing capabilities
- Assessing memory retention in prediction systems
- Comparing different architectural approaches to temporal sequence processing

3.3 About Theoretical Methodology

The Lipschitz condition is a mathematical concept that describes how rapidly a function can change. In simple terms, if a function satisfies the Lipschitz condition, it means that

the rate of change of the function is bounded and cannot suddenly become very large or very small.

3.3.1 Mathematical Definition

A function f satisfies the Lipschitz condition if there exists a constant L (called the Lipschitz constant) such that for any two points x_1 and x_2, the following relationship holds:

$$|f(x_1) - f(x_2)| \leq L \cdot |x_1 - x_2|,$$

The components of the Lipschitz condition can be interpreted as follows:

- $|f(x_1) - f(x_2)|$: The difference between the function outputs for two inputs x_1 and x_2
- $|x_1 - x_2|$: The difference between the two input points x_1 and x_2
- L: A constant number that represents the maximum rate of change of the function

3.3.2 Practical Implications

The value of the Lipschitz constant L determines the behavior of the function:

- If L is small, the function cannot change rapidly
- If L is large, the function can have more rapid changes

This condition ensures that the function changes smoothly and continuously, without any sudden jumps or discontinuities.

3.4 About Proposed heuristic approach

This section provides a comprehensive overview of the mathematical functions and methodological approaches employed in our model. The mathematical framework encompasses

various functions that serve as fundamental building blocks for the proposed methodology. These functions were carefully selected based on their ability to capture the underlying patterns and relationships within the data, while maintaining computational efficiency and mathematical robustness. The following subsections detail the specific functions utilized, their mathematical properties, and their roles in the overall model architecture. Each function was chosen to address particular aspects of the problem domain, ensuring that the model can effectively handle both linear and non-linear relationships present in the data. Understanding these functions is crucial for comprehending the model's behavior and its theoretical foundations.

3.4.1 Backpropagation

Backpropagation, short for "backward propagation of errors," is a supervised learning algorithm widely used for training artificial neural networks. It is an optimization technique based on gradient descent and plays a critical role in minimizing the error of a neural network by adjusting its weights systematically.

Backpropagation operates in two main phases: the forward pass and the backward pass. During the forward pass, the input data propagates through the network, layer by layer, and produces an output. This output is then compared to the actual target values using a loss function, such as Mean Squared Error (MSE) or Cross-Entropy Loss, to calculate the error. The backward pass begins with this error and uses the chain rule of calculus to compute the gradient of the error with respect to the network's weights.

The algorithm iteratively updates the weights of the network by moving in the opposite direction of the gradient, ensuring the error decreases with each iteration. The learning rate, a hyperparameter, controls the step size of these updates and significantly affects the convergence speed and stability of the training process.

One of the notable advantages of backpropagation is its ability to efficiently train multi-layer networks, often referred to as deep neural networks. It allows these models to learn complex patterns and relationships in data, making them suitable for tasks such as image recognition, time series prediction, and natural language processing.

In this study, the backpropagation algorithm was utilized to optimize the parameters of the proposed neural network structure. By iteratively reducing the error during training, the model achieved improved performance on the target problem. The implementation of

backpropagation in this work was carried out using the PyTorch library, which provides robust tools for automatic differentiation and gradient computation.

3.4.2 Small-World and Scale-Free Networks

Small-world and scale-free networks are two prominent models used to describe the topological structure of complex systems. These models have significant implications in the study of neural networks, including Echo State Networks (ESNs), where the connectivity of the reservoir greatly impacts performance.

3.4.2.1 Small-World Network

The small-world network method generates a weight matrix based on the Watts-Strogatz model [27] using the **networkx** library. A small-world network is a type of network where most nodes are not directly connected but can be reached through a small number of steps. This model creates a network that combines high clustering and short average path lengths, resembling real-world networks such as social networks or neural networks. A simple example of a small-world network structure is shown in the figure below.

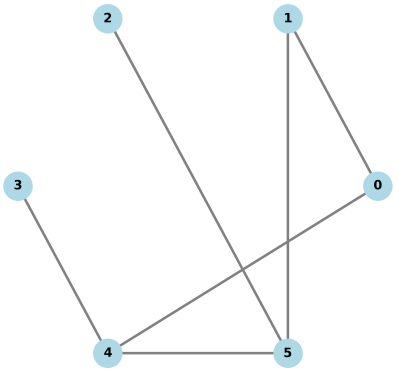

FIGURE 3.4: Example of a Small-World Network topology

3.4.2.1.1 Properties of Small-World Networks

Small-world networks exhibit several key characteristics:

- **Clustering Coefficient**: Has a high clustering coefficient, indicating nodes tend to form tightly connected local groups. This property is significantly higher than in random networks.

- **Short Path Length**: Despite the clustered structure, the average path length between any two nodes remains remarkably short, typically scaling logarithmically with network size.
- **Connectivity**: Features strong local connectivity within clusters while maintaining efficient global connections through random shortcuts.
- **Sparsity**: Maintains a sparse connection matrix where most possible connections are absent, yet the network remains highly functional through strategic linking.

3.4.2.1.2 Small-World Pseudo Code

Algorithm 3 Create Small World Weight Matrix

procedure CreateSmallWorldMatrix($size, k, p, weightScale, seed$)
  Input:
    $size$: number of nodes
    k: mean degree of nodes
    p: rewiring probability
    $weightScale$: scaling factor for weights
    $seed$: random seed for reproducibility
  Initialize:
    Set random seed to $seed$
  Create Base Graph:
    Create regular ring lattice with $size$ nodes
    Connect each node to k nearest neighbors
  Rewire Connections:
  for each edge (u, v) in graph **do**
    if random number $< p$ **then**
      Remove edge (u, v)
      Add new random edge from u to another node
    end if
  end for
  Generate Weight Matrix:
    $W \leftarrow$ Convert graph to adjacency matrix
    Generate random matrix R with values in $[0, 1]$
    $W \leftarrow W \times (R - weightScale)$
  Return: Weight matrix W
end procedure

3.4.2.1.3 Function Parameters

Key features of this function:

- It uses the Watts-Strogatz model to create the network structure.
- The `size` parameter determines the number of nodes in the network.
- `k` specifies the number of nearest neighbors each node is initially connected to.
- `p` is the probability of rewiring each edge, introducing randomness into the network.
- The resulting connections are converted into a weight matrix.
- Weights are adjusted using random values scaled by the `weight_scale` parameter.

3.4.2.1.4 Weight Scaling Operation

The weight scaling operation introduces weighted connections to the network structure. This transformation converts the binary adjacency matrix into a weighted network where connections have varying strengths. The operation centers these weights around zero through the `weight_scale` parameter, enabling both positive and negative connection strengths. This approach maintains the network's topology while introducing weighted relationships between connected nodes, which is essential for modeling complex systems where connection strengths vary and can have different influences on connected nodes.

3.4.2.2 Scale-Free Network

The scale-free network method uses the Barabási-Albert model [3]from the **networkx** library to generate a weight matrix. A scale-free network is a type of network in which the degree distribution follows a power law, meaning that a few nodes (called hubs) have a very high degree, while most nodes have a low degree. This model emphasizes highly connected nodes, creating a network where the degree distribution follows a power law, resembling real-world networks such as the internet or social networks. A simple example of a scale-free network structure is shown in the figure below.

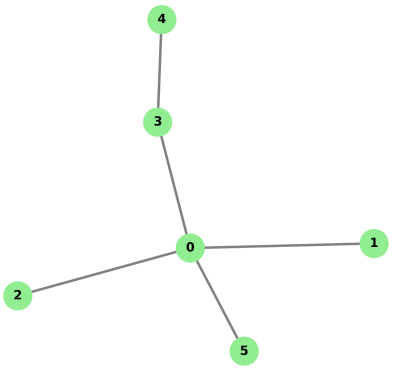

FIGURE 3.5: Example of a Scale-Free Network topology

3.4.2.2.1 Properties of Scale-Free Networks

Scale-free networks exhibit several important properties:

- **Power Law**: The node degree distribution follows a power law $P(k) \sim k^{-\gamma}$, where γ is typically between 2 and 3, meaning a few nodes have many connections while most have few.
- **Matrix Sum Property**: The sum of absolute values of the weight matrix elements tends to be higher than other network types due to the presence of highly connected hub nodes.

3.4.2.2.2 Scale-Free Pseudo Code

Algorithm 4 Create Scale Free Weight Matrix

procedure CreateScaleFreeMatrix($size, m, weightScale, seed$)
    Input:
        $size$: number of nodes
        m: number of edges for each new node
        $weightScale$: scaling factor for weights
        $seed$: random seed for reproducibility
    Initialize:
        Set random seed to $seed$
    Create Base Graph:
        Initialize graph with m nodes
        Create fully connected graph among initial nodes
    Grow Network:
    for node i from $m + 1$ to $size$ **do**
        Add new node to graph
        Connect to m existing nodes using preferential attachment ▷ Probability proportional to node degree
    end for
    Generate Weight Matrix:
        $W \leftarrow$ Convert graph to adjacency matrix
        Generate random matrix R with values in $[0, 1]$
        $W \leftarrow W \times (R - weightScale)$
    Return: Weight matrix W
end procedure

3.4.2.2.3 Function Parameters

Key features of this function:

- It uses the Barabási-Albert model to create the network structure.
- The `size` parameter determines the number of nodes in the network.
- `m` specifies the number of edges to attach from a new node to existing nodes during network growth.
- The resulting connections are converted into a weight matrix.
- Weights are adjusted using random values scaled by the `weight_scale` parameter.

Similar to the small-world network implementation, the weight scaling operation serves the same purpose of introducing weighted connections with varying strengths to the network structure.

3.4.3 Graph Metrics and Network Characteristics

3.4.3.1 Clustering Coefficient

The clustering coefficient is a measure that describes how well the neighbors of a node in a network are connected to each other. It provides insight into the local connectivity and structure of the network. A high clustering coefficient indicates that the neighbors of a node form a tightly connected group, while a low clustering coefficient suggests sparse interconnections among neighbors.

The clustering coefficient (C) is mathematically defined as:

$$C = \frac{1}{N} \sum_{i=1}^{N} \frac{2e_i}{k_i(k_i - 1)}, \tag{3.3}$$

where:

- e_i: Number of edges between the neighbors of node i.
- k_i: Degree of node i (the number of its connections).

Simple Example: Clustering Coefficient Calculation

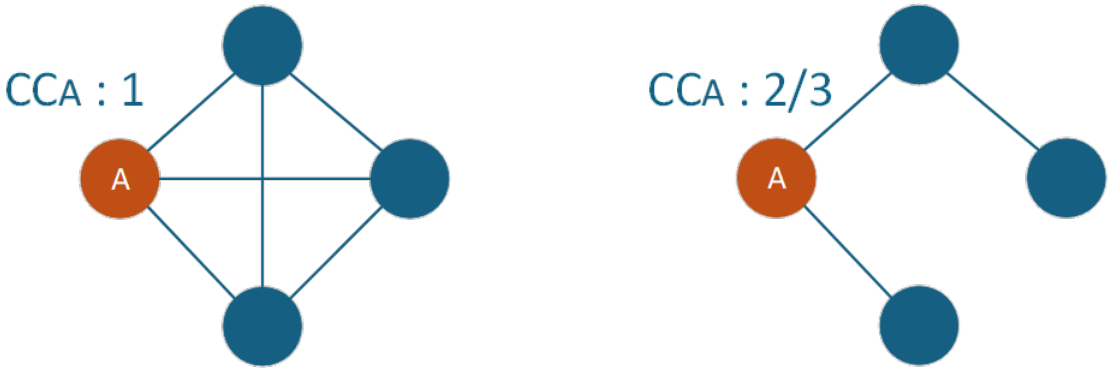

FIGURE 3.6: An example illustrating the clustering coefficient.

Consider two scenarios for a node A and its connections in a network:

1. **Fully Connected Neighborhood:** In the first graph (left side), node A is connected to all its neighbors (nodes B, C, and D). The clustering coefficient (C) for node A is calculated as 1, indicating a fully connected neighborhood.

2. **Partially Connected Neighborhood:** In the second graph (right side), node A is connected to nodes B and C, but there is no connection between B and C. As a result, the clustering coefficient (C) for node A is 2/3, indicating that the neighborhood is not fully interconnected.

3.4.3.2 Path Length

The average path length is a measure that quantifies the efficiency of information transfer within a network. It represents the mean of the shortest path lengths between all pairs of nodes. A smaller path length indicates a more interconnected network where information can travel quickly between nodes.

The average path length (L) is defined mathematically as:

$$L = \frac{1}{N(N-1)} \sum_{i \neq j} d_{ij}, \tag{3.4}$$

where:

- d_{ij}: The shortest distance between nodes i and j.
- N: The total number of nodes in the network.

Simple Example: Path Length Calculation

Consider the following graph where we calculate the path length between node A and node E:

- The path length between node A and node E is 2, as it requires two steps to move from A to E. The possible path is:

- $A \rightarrow B \rightarrow E$

This example demonstrates that the shortest path between nodes A and E is of length 2, highlighting the concept of path length in network analysis.

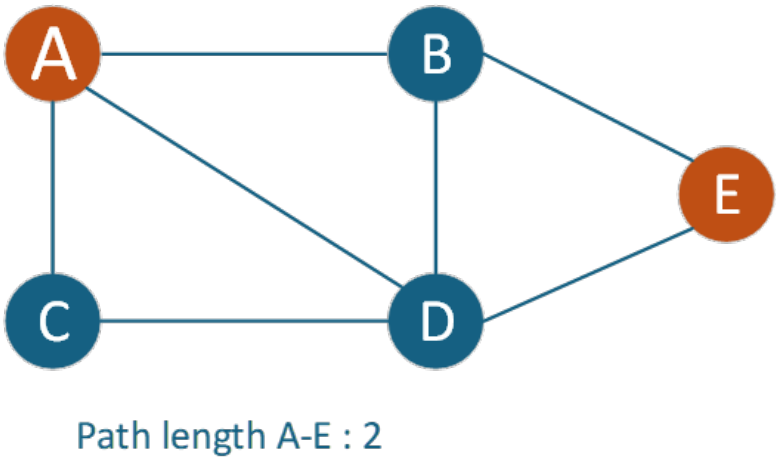

FIGURE 3.7: An example illustrating the calculation of path length.

3.4.3.3 Connectivity

Connectivity describes how nodes in a network are linked and determines whether the network is fully connected. A graph is fully connected if there is a path between every pair of nodes, either directly or through intermediate nodes. This ensures that all nodes can communicate with each other, making the network robust and efficient. Nodes with more connections, known as hubs, play a key role in maintaining this connectivity.

3.4.3.4 Sparsity

Sparsity measures the density of connections in a network and provides insight into how interconnected the nodes are relative to a fully connected network. It is particularly useful for understanding the structure and efficiency of networks, especially in cases where connections are sparse.

The sparsity S of a network is mathematically defined as:

$$S = 1 - \frac{E}{N(N-1)/2}, \tag{3.5}$$

where:

- E: The total number of edges in the network.
- N: The total number of nodes in the network.

Simple Example: Sparsity Calculation

Consider a network with 5 nodes and 6 edges. The sparsity S can be calculated as follows:

1. Calculate the total possible edges in a fully connected graph with N nodes:

$$\frac{N(N-1)}{2} = \frac{5(5-1)}{2} = 10$$

2. Substitute the values of E and N into the sparsity formula:

$$S = 1 - \frac{E}{N(N-1)/2} = 1 - \frac{6}{10} = 1 - 0.6 = 0.4$$

3. Thus, the sparsity of the network is:

$$S = 0.4$$

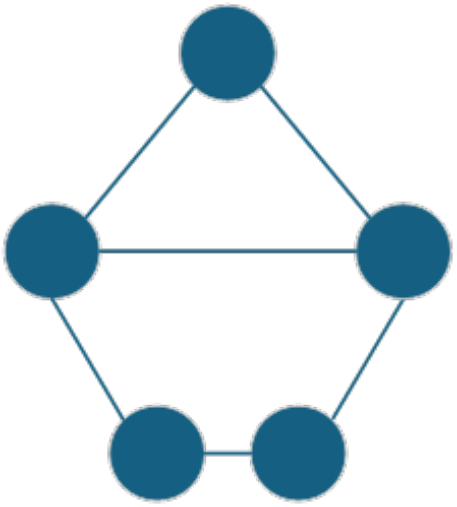

FIGURE 3.8: An example illustrating sparsity calculation in a network.

This example demonstrates that the network is relatively sparse, with only 40% of the possible connections absent. Sparsity provides a simple yet powerful metric for understanding the structure and connectivity of complex networks.

3.4.3.5 Average of Absolute Matrix

The average of the absolute values of all elements in a weight matrix W is defined as:

$$\text{Average}(W) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} |w_{ij}|, \tag{3.6}$$

where:

- w_{ij}: The weight of the connection between nodes i and j.
- N: The total number of nodes in the matrix.

Simple Example: Average of Absolute Matrix

Consider a weight matrix W for a network with 4 nodes:

$$W = \begin{bmatrix} 0.2 & -0.5 & 0.3 & -0.1 \\ 0.6 & 0.1 & -0.2 & 0.4 \\ -0.3 & 0.7 & 0.5 & -0.4 \\ 0.1 & -0.4 & 0.2 & 0.3 \end{bmatrix}$$

The sum of the absolute values for W is calculated as:

$$\begin{aligned} \text{Sum}(W) = &|0.2| + |-0.5| + |0.3| + |-0.1| + |0.6| + |0.1| + |-0.2| + |0.4| \\ &+ |-0.3| + |0.7| + |0.5| + |-0.4| + |0.1| + |-0.4| + |0.2| + |0.3| \end{aligned}$$

$$\text{Sum}(W) = 2.5 \tag{3.7}$$

Now, calculate the average using the formula:

$$\text{Average}(W) = \frac{\text{Sum}(W)}{N^2} = \frac{2.5}{4^2} = \frac{2.5}{16} = 0.15625$$

This calculation reflects the average magnitude of weights in the matrix, providing insight into the overall connection strengths in the network.

Chapter 4

Proposed Method

4.1 Introductoin

In this chapter, we present our methodology which consists of two main parts: theoretical foundations and practical implementations. In the theoretical section, we establish the significance of reservoir topology in Echo State Networks (ESNs) and demonstrate how the structural properties of the reservoir directly influence the network's computational capabilities. This theoretical framework provides the foundation for our subsequent practical implementations. Building upon these theoretical insights, we then present two novel approaches for optimizing ESN reservoirs through weight adjustment using supervised learning and semi-supervised learning learning paradigms. Throughout these implementations, we demonstrate how our theoretical understanding of reservoir topology can be practically applied to enhance ESN performance across different learning scenarios. This comprehensive methodology bridges the gap between theoretical insights and practical applications, offering flexible solutions for various real-world contexts where different amounts of training data may be available.

4.2 Theoretical baseis of the methodology

In this section, we study the behavior of ESN reservoir and provide some insights on it. Also, we provide an upper bound on error prediction of the network.

As it is mentioned before, the weights of readout connections are computed based on state matrix collection (M) and teacher matrix T. Therefore, deviation of states of unseen data from training states (rows of matrix M) affects the prediction accuracy.

To develop a foundation to estimate prediction error and have more insights on ESN mechanism, we present theorem 4.1

Theorem 4.1. *If* $S_1 =< (u(k+1), y(k+1)), \ldots, (u(k+t), y(k+t)) >$ *and* $S_2 =< (u(l+1), y(l+1)), \ldots, (u(l+t), y(l+t)) >$ *are two input/output of a time series* ($||S_1|| = ||S_2|| = t$) *and* f^{res} *satisfies Lipschitz condition (e.g., tanh and identity functions) and* $X(k) \overset{S_1}{\rightsquigarrow} X(k+t)$ *and* $X(l) \overset{S_2}{\rightsquigarrow} X(l+t)$ [1]*, then*[2]

$$||X(k+t) - X(l+t)|| \leq ||W^{res}||^t ||X(k) - X(l)|| +$$
$$\sum_{m=0}^{t-1} ||W^{res}||^m ||W^{in}|| ||u(k+t-m) - u(l+t-m)|| +$$
$$\sum_{m=0}^{t-1} ||W^{res}||^m ||W^{back}|| ||y(k+t-m-1) - y(l+t-m-1)||$$

Proof. The considered update equation is $X(n) = f^{res}(W^{res}X(n-1) + W^{in}u(n) + W^{back}y(n-1))$ which is the most general case in term of connections. So, we have:

$$||X(k+t) - X(l+t)|| = ||f^{res}(W^{res}X(k+t-1) + W^{in}u(k+t) + W^{back}y(k+t-1)) -$$
$$f^{res}(W^{res}X(l+t-1) + W^{in}u(l+t) + W^{back}y(l+t-1))||$$
$$\overset{Lipschits\ Condition}{\Longrightarrow}$$
$$||X(k+t) - X(l+t)|| \leq$$
$$||W^{res}(X(k+t-1) - X(l+t-1)) + W^{in}(u(k+t) - u(l+t)) + W^{back}(y(k+t-1) - y(k+t-1))||$$
$$= ||W^{res}(f^{res}(W^{res}X(k+t-2) + W^{in}u(k+t-1) + W^{back}y(k+t-2)) - f^{res}(W^{res}X(l+t-2) +$$
$$W^{in}u(l+t-1) + W^{back}y(l+t-2))) + W^{in}(u(k+t) - u(l+t)) + W^{back}(y(k+t-1) - y(k+t-1))||$$
$$\leq ||||W^{res}|| ||W^{res}(X(k+t-2) - X(l+t-2)) + W^{in}(u(k+t-1) - u(l+t-1)) + W^{back}(y(k+$$
$$t-2) - y(k+t-2))|| + W^{in}(u(k+t) - u(l+t)) + W^{back}(y(k+t-1) - y(k+t-1))||$$
$$\leq ||W^{res}||^2 ||X(k+t-2) - X(l+t-2)|| + ||W^{res}|| ||W^{in}|| ||u(k+t-1) - u(l+t-1)|| +$$
$$||W^{res}|| ||W^{back}|| ||y(k+t-2) - y(k+t-2)|| + ||W^{in}|| ||u(k+t) - u(l+t)|| +$$
$$||W^{back}|| ||y(k+t-1) - y(k+t-1)||$$
$$\leq ||W^{res}||^i ||X(k+t-i) - X(l+t-i)|| + \sum_{m=0}^{i-1} ||W^{res}||^m ||W^{in}|| ||u(k+t-m) - u(l+t-m)|| +$$
$$\sum_{m=0}^{i-1} ||W^{res}||^m ||W^{back}|| ||y(k+t-m-1) - y(l+t-m-1)||$$
$$\overset{i=t}{\Longrightarrow}$$

[1] $X(p) \overset{S}{\rightsquigarrow} X(q)$ means that if the state of reservoir is $X(p)$ and the sequence S is fed to the network, the resulted state will be $x(Q)$.

[2] In this paper, $||W||$ of a matrix (vector) W is a matrix (vector) of absolute values of entries of W. In addition, when we write $A \leq B$, it means that each element of A is less than or equal to its corresponding element in B.

$$||X(k+t) - X(l+t)|| \leq$$
$$||W^{res}||^t||X(k+t-t) - X(l+t-t)|| + \sum_{m=0}^{t-1} ||W^{res}||^m||W^{in}||||u(k+t-m) - u(l+t-m)|| +$$
$$\sum_{m=0}^{t-1} ||W^{res}||^m||W^{back}||||y(k+t-m-1) - y(l+t-m-1)||$$

Theorem 4.2. *Under condition of theorem 4.1 and with assumption that f^{out} (activation function of output layer) is a function with Lipschitz property and with output computing $y(n) = f^{out}(W^{out}(X(n); u(n); y(n-1)))$, $W^{out} = [W^{res-out}, W^{in-out}, Wloop]$ we have:*

$$||y(k+t) - y(l+t)|| \leq ||W^{res-out}||||X(k+t) - X(l+t)|| +$$
$$||W^{in-out}||||u(k+t) - u(l+t)|| + ||W^{loop}||||y(k+t-1) - y(l+t-1)||$$

Proof. $||y(k+t) - y(l+t)|| = ||f^{out}(W^{out}(X(k+t); u(k+t); y(k+t-1))) - f^{out}(W^{out}(X(l+t); u(l+t); y(l+t-1)))||$

$\leq ||W^{out}(X(k+t); u(k+t); y(k+t-1)) - W^{out}(X(l+t); u(l+t); y(l+t-1))||$

$= ||W^{res-out}X(k+t) + Win - outu(k+t) + W^{loop}y(k+t-1) - (W^{res-out}X(l+t) + Win - outu(l+t) + W^{loop}y(l+t-1))||$

$||W^{res-out}(X(k+t) - X(l+t)) + Win - out(u(k+t) - u(l+t)) + W^{loop}(y(k+t-1) - y(l+t-1))||$

$\Longrightarrow$

$||y(k+t) - y(l+t)|| \leq ||W^{res-out}||||X(k+t) - X(l+t)|| +$

$||W^{in-out}||||u(k+t) - u(l+t)|| + ||W^{loop}||||y(k+t-1) - y(l+t-1)||$

Theorems 4.1 and 4.2 give some insights on behavior of ESN discussed in the rest of this section. Assume the input sequence S_1 is a part of training sequence, after stabilization

of the reservoir, so $(X(l), u(l), y(l-1))$ and $(X(l+t), u(l+t), y(l+t-1))$ are in state collection matrix M. In addition, $|y(l+t) - d(l+t)| \leq \sigma$, where σ is training error.

If S_2 is in test sequence (unseen data) and $y(n)$ is scalar, we have the followings: For repeated patterns with length t (e.g., $S_1 = S_2$), the maximum deviation between training and test states is bounded by $|W^{res}|^t|X(k)–X(l)|$ and if spectral radius of W^{res} is less than one (sufficient condition for echo state property) and t is big enough, the deviation converges to zero.

An important point is that the best case of W^{res} is depends on length of the patterns and all the patterns have not the same length. If spectral radius of $W^{res}(\|W^{res}\|)$ is very small, we have immature convergence and only a small segment of the pattern puts the reservoir on a local optimum.

In the other words, the input sequence (both training and test) are divided to many small patterns each of which with a small error that leads to significant error over the whole input.

On the other side, big values of $\|W^{res}\|$ causes that distinct concatenated patterns are considered to be one pattern. It means that for example S_1S_2 and $S_1'S_2'$ as two patterns with the same length and if $S_1 = S_1'$ and $S_2! = S_2'$, then some errors are introduced in the result.

Theorems 4.1 and 4.2 show that both weights and topology of reservoir affect prediction accuracy, generating W^{res} randomly as it is done in ESN may not be optimal.

Theorems 4.1 and 4.2 show that the optimal weights and topology of the reservoir depend on the input. Therefore, an input-dependent design of the reservoir may not always be optimal, and different reservoir designs reported in previous work may not be generalizable to all input time series. Hence, we should think beyond a fixed reservoir topology in Echo State Networks. We should pay some computational cost to build an input-driven reservoir, but when prediction accuracy is critical, this trade-off is necessary. In the next subsection, we propose two approaches for building an input-driven reservoir.

4.3 Proposed heuristic approach

4.3.1 Introduction

In this section, we aim to find optimal reservoir matrices through different optimization methods. We present two distinct approaches for optimizing Echo State Networks (ESNs): supervised, semi-supervised methods. The supervised model focuses on the reservoir matrix optimization through an iterative process, utilizing an optimization loop to enhance the reservoir weights based on loss. The semi-supervised model begins by providing hyperparameters to the network, essentially performing Hyperparameter Optimization (HPO). In this phase, we systematically sample the network's behavior by introducing different parameter values and calculating the corresponding error rates. This sampling process allows us to understand how various hyperparameter combinations affect the network's performance. Through this systematic exploration and error calculation, we build a comprehensive dataset that captures the relationship between hyperparameter settings and network behavior. After collecting this dataset, we employ two distinct approaches to find hyperparameters that yield minimal error rates. Each approach offers a unique methodology for processing the sampled data and identifying optimal parameter configurations. Through these dual training strategies, we aim to determine the most effective hyperparameter settings that minimize the network's error and maximize its performance.

4.3.2 Superviesd Model

The supervised optimization framework operates on the principle of iterative refinement of the reservoir matrix while maintaining the essential properties of Echo State Networks. Our approach distinguishes itself from traditional ESN optimization methods by directly incorporating error gradients into the reservoir weight optimization process, allowing for more precise control over the network's dynamic properties.

4.3.2.1 Detailed Methodology

The optimization process begins with the initialization of the reservoir matrix W, where initial parameters are carefully chosen to ensure the echo state property is maintained. During this phase, the network's basic structure and hyperparameters are established to provide a stable foundation for the optimization process.

The complete architecture of our supervised optimization approach is illustrated in Figure 4.1, which provides a visual representation of the optimization process. The figure demonstrates the iterative nature of our methodology, showing how the W matrix is progressively refined through the calculation block. As shown in the figure, the process begins with initialization of W (top) and proceeds through a cycle of computation and optimization (middle) before producing the final optimized W matrix (bottom). The calculation block, highlighted in blue, encompasses the core components of the ESN operation, including the processing of time series data and parameter management. This visual representation helps clarify the interconnected nature of the various components and the flow of information through the system.

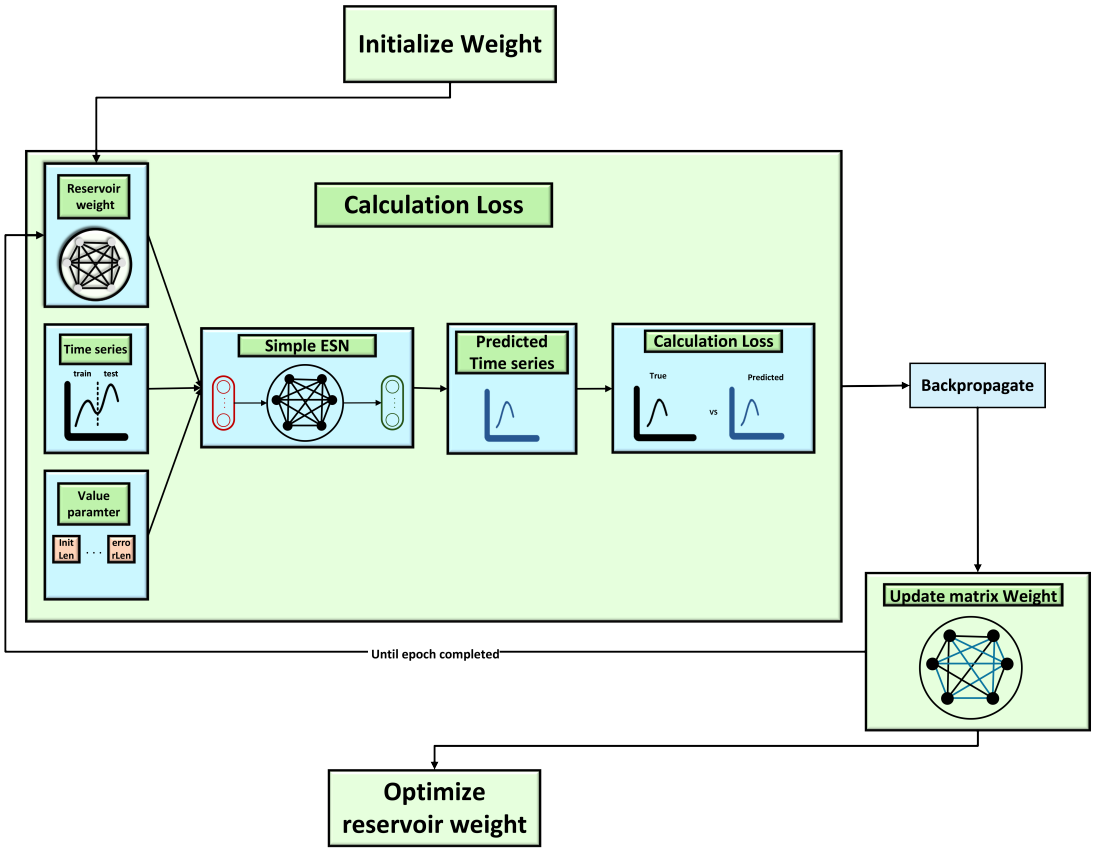

FIGURE 4.1: Supervised optimization architecture for Echo State Networks

At the core of our methodology lies the calculation block, which serves as the central component for processing and optimization. This block takes two primary inputs: the current state of the W matrix, the time series data under analysis, and the fundamental ESN parameters. Within this block, a simple ESN is constructed using the current configuration, which then generates predicted time series values. The accuracy of these predictions is

assessed through the calculation of the Normalized Root Mean Square Error (NRMSE) between the predicted and actual values.

The iterative optimization process involves a sophisticated feedback loop where the computed NRMSE is utilized to generate error gradients. These gradients guide the systematic update of the W matrix, with this refinement process continuing until the specified number of epochs is completed. The mathematical framework underlying this process involves precise calculations for both the NRMSE and the weight update rules, ensuring consistent and reliable optimization.

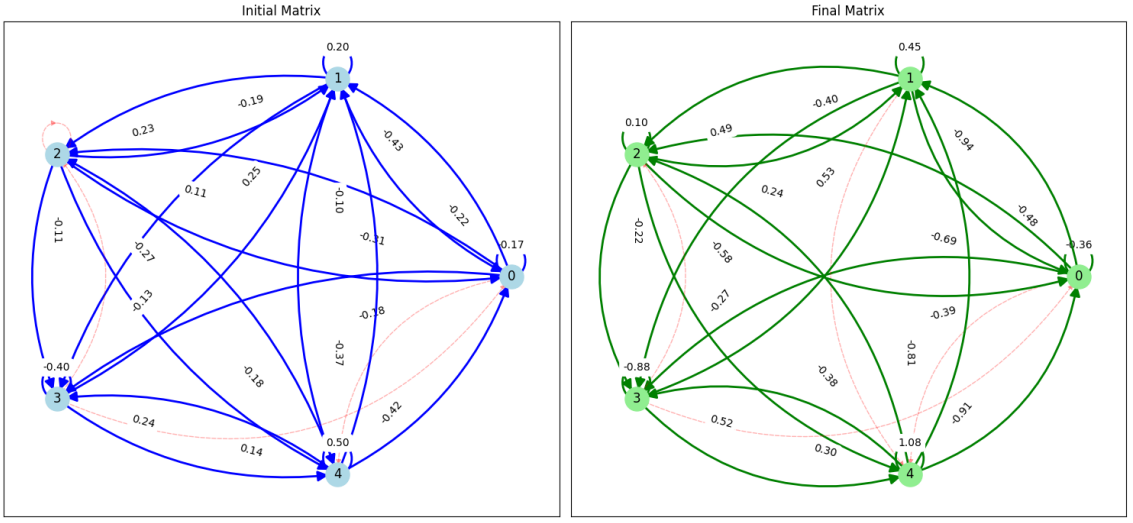

FIGURE 4.2: Comparison of Initial (left) and Final (right) reservoir matrices after optimization, showing eliminated connections as red dotted lines and changes in edge weights.

As mentioned earlier, connection weights can either increase or decrease during the training process, and we have provided an example to demonstrate this behavior. Figure 4.2 presents two reservoir matrices showing both Initial Matrix (left) and Final Matrix (right) states of the reservoir. In this example, two edges (from node 4 to node 0 and from node 3 to node 0) were eliminated when their weights reached zero during the training process, shown as red dotted lines in the Final Matrix. While no new edges emerged during this optimization, we observe significant changes in existing edge weights, such as the self-loop weights at nodes 1, 0, 3, and 4, which demonstrate how the optimization process adjusts the network's connectivity strengths to improve performance. For clarity and better visualization purposes, this example uses a simplified reservoir with a small number of nodes, though real-world implementations typically involve larger networks with more complex connectivity patterns.

4.3.2.2 Pseudocode

The following algorithm demonstrates a supervised method for Echo State Network (ESN) optimization using gradient descent with sparsification techniques. The method processes Mackey-Glass time series data to optimize reservoir weights.

Algorithm 5 Supervised Method

Input:
    Time series data
    Learning rate
    Number of epochs
    Reservoir size
    Initial washout length
    Error length
    Training/Testing split ratio
Output:
    Optimized reservoir weights
    NRMSE performance metrics
function Loss Function(W)
    // Initialize ESN with current weights
    Initialize ESN parameters (input weights)
    // Forward Pass through ESN
    Run ESN predictions for training data
    // Calculate loss
    Calculate loss between predictions and actual values
    return loss
end function
function Optimization
    Initialize W_{old} randomly
    Initialize optimizer
    for epoch = 1 to num_epochs **do**
        // Calculate loss and gradients
        loss = Loss Function(W_{old})
        Calculate gradients of loss with respect to W_{old}
        // Update weights using gradient descent
        $W_{new} \leftarrow W_{old} - lr * gradients$
        // Store new weights and loss
    end for
    return W_{new}
end function

4.3.3 Semi-supervised Model

Our semi-supervised approach introduces a novel method for optimizing Echo State Network (ESN) parameters through a two-phase process: data generation and parameter prediction. This approach combines the strengths of neural networks with established network generation models to create an efficient and robust optimization framework.

4.3.3.1 Detailed Methodology

The foundation of our method rests upon two well-established network models: the Watts-Strogatz model for small-world networks and the Barabási-Albert model for scale-free networks. These models serve as the basis for generating the reservoir connectivity patterns in our ESN architecture.

The Watts-Strogatz model requires three key parameters:

- n: number of nodes
- k: number of edges connected to each node
- p: probability of rewiring edges to random nodes

The Barabási-Albert model requires two parameters:

- n: number of nodes
- m: number of edges for each new node added to the network

Each model generates its respective adjacency matrix according to the following equations:

$$F_1(k, p) = \text{WattsStrogatz}(n, k, p) \tag{4.1}$$

$$F_2(m) = \text{BarabasiAlbert}(n, m) \tag{4.2}$$

After generating these matrices, which initially contain only binary values (0 or 1), we transform them using random weight scaling. This process involves multiplying each matrix by a random matrix with values adjusted by weight-scale parameters:

$$W_1 = F_1(k, p) \cdot (R_1 \cdot \text{scale\_weight}_1) \tag{4.3}$$

where R_1 represents a random matrix with values between 0 and 1, and:

$$W_2 = F_2(m) \cdot (R_2 \cdot \text{scale\_weight}_2) \tag{4.4}$$

where R_2 represents a random matrix with values between 0 and 1.

The final hybrid matrix is constructed through a weighted combination using the mixing parameter α:

$$W_{\text{hybrid}} = \alpha W_1 + (1 - \alpha) W_2 \tag{4.5}$$

This hybrid approach allows us to leverage the advantages of both network types while maintaining flexibility through the mixing parameter α.

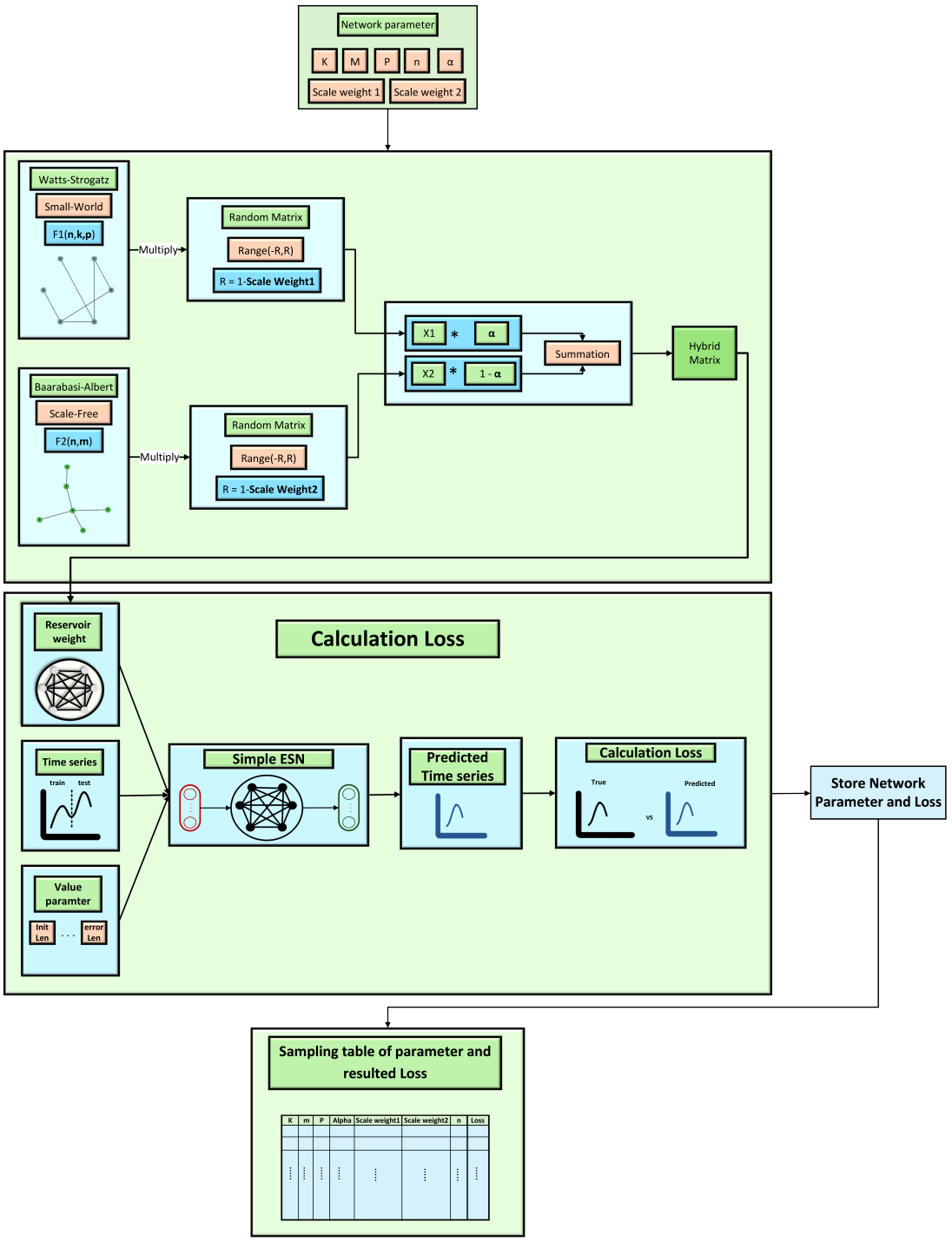

FIGURE 4.3: Architecture of the semi-supervised hybrid matrix generation and evaluation process.

The complete process of our hybrid matrix generation and parameter sampling approach

is illustrated in Figure 4.3. The figure shows the systematic flow from network parameter initialization through matrix generation and combination, to the final evaluation and storage of results. As depicted in the upper portion of the figure, network parameters (k, m, p, alpha, scale weights) are used to generate and combine the small-world and scale-free characteristics. The lower portion demonstrates the evaluation process, where the resulting hybrid matrices are tested using a simple ESN to calculate NRMSE values. To build our training dataset, we systematically sample the parameter space by generating matrices with different combinations of parameters (m, k, p, alpha, weight-scale-1, weight-scale-2) and evaluate their performance using NRMSE as our metric. This comprehensive evaluation generates a dataset that maps network parameters to performance metrics, establishing the foundation for our supervised learning approach. The sampling process captures the relationship between structural parameters and network performance, enabling the development of predictive models for optimal ESN configuration. The sampling table generated through this process provides valuable insights into the relationship between network parameters and performance, serving as the foundation for the subsequent supervised learning phase. This systematic approach to data generation ensures comprehensive coverage of the parameter space while maintaining the computational feasibility of the optimization process.

4.3.3.2 Pseudocode

The following algorithm outlines the sampling process for our semi-supervised model, which explores different combinations of hyperparameters to optimize the ESN performance through loss calculation.

Algorithm 6 Semi-Supervised Model - sampling

Input:
    List of network hyperparameter ranges
Output:
    Sampling for different combinations of hyperparameters
for each parameter combination **do**
    // Create hybrid matrix
    W = hybrid matrix(hyperparameters)
    // loss calculation
    Loss = Loss_function(W)
    Store hyperparameters for calculated loss
end for
function Loss Function(W)
    // Initialize ESN with current weights
    Initialize ESN parameters (input weights, reservoir size)
    // Forward Pass through ESN
    Run ESN predictions for training data
    // Calculate loss
    Calculate loss between predictions and actual values
    return loss
end function

4.3.3.3 Direct Parameter Prediction Approach (DPP)

4.3.3.3.1 Method Details

In the first approach, we develop a neural network model that maps a single NRMSE value to seven ESN parameters. The network's architecture progressively transforms the data dimensions through several stages. Starting with a single input neuron for NRMSE, the first block expands the dimension to 256 features ($1 \rightarrow 256$). The second block further expands to 512 features ($256 \rightarrow 512$), while the third block contracts back to 256 features ($512 \rightarrow 256$). Finally, a linear layer reduces the dimension to seven outputs ($256 \rightarrow 7$), corresponding to the ESN parameters. Each processing block, except the final layer, combines a linear transformation with layer normalization, ReLU activation, and dropout for regularization.

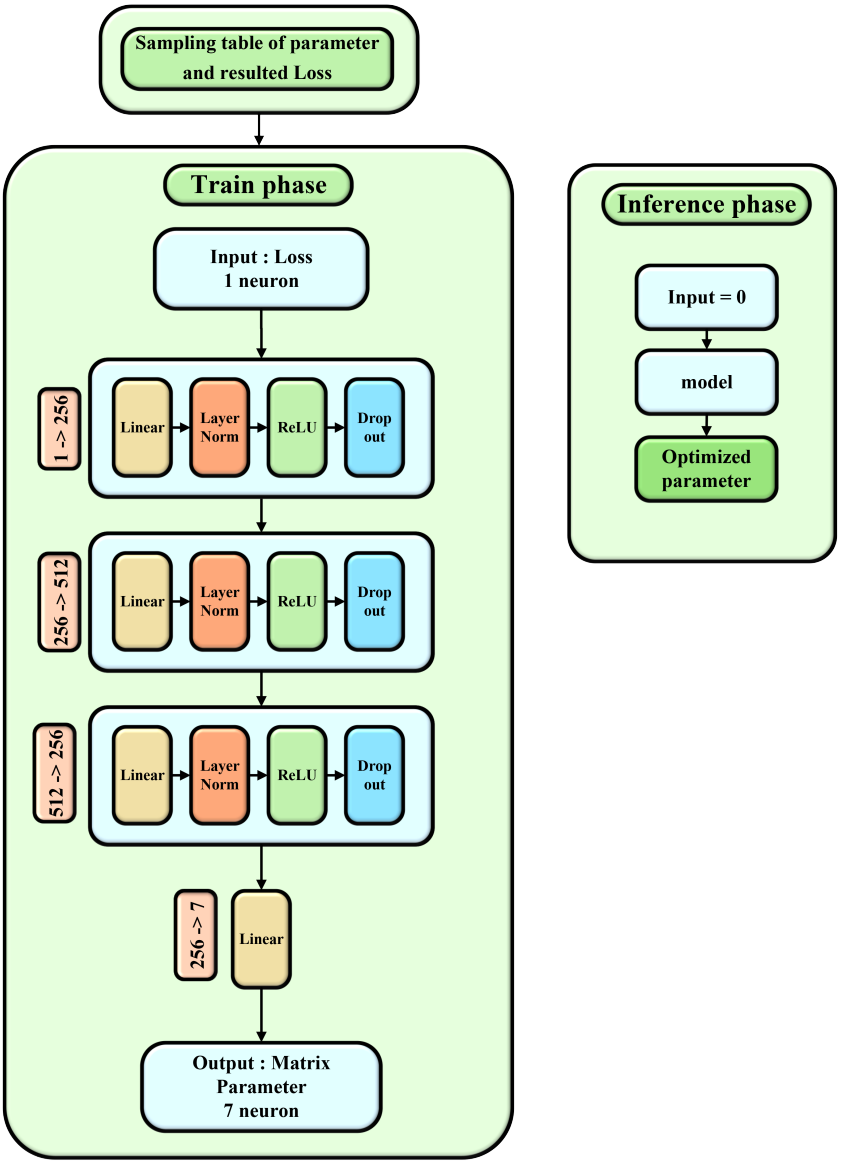

FIGURE 4.4: Neural network architecture for direct parameter prediction. The network consists of multiple processing blocks that transform the input NRMSE through various dimensional spaces, ultimately producing optimized ESN parameters. The left side shows the training phase with its progressive dimensional transformations, while the right side illustrates the inference phase where the trained model predicts optimal parameters.

As shown in Figure 4.4, our architecture consists of two distinct phases: training and inference. During the training phase, the network processes NRMSE values through multiple transformation blocks, each incorporating sophisticated regularization techniques to

ensure robust learning. The inference phase demonstrates the network's ability to predict optimal parameters when presented with a target NRMSE value. To ensure the predicted parameters remain physically meaningful, we implement custom constraint layers in our architecture. These constraints handle different parameter types appropriately: k and m are constrained to be integers within a specific range, p and α are transformed to values between 0 and 1, and weight scales are ensured to be positive values. This ensures all predictions fall within valid ranges while maintaining differentiability during training. After training the model, we evaluate its effectiveness by using it to predict the optimal parameters that would theoretically achieve an NRMSE of 0. This means we input a target NRMSE of 0 into the trained network and obtain the corresponding set of seven parameters that the model predicts would achieve this perfect performance. This method significantly reduces the computational burden of manual parameter tuning while maintaining high performance standards, offering a practical tool for reservoir computing systems.

4.3.3.3.2 Pseudocode

The following algorithm presents DPP Approach of our semi-supervised model, which employs a neural network architecture to determine optimal hyperparameters through distinct training and testing phases.

Algorithm 7 Semi-Supervised Model - Direct Parameter Prediction

Input:
    List of network hyperparameter ranges
Output:
    Optimal hyperparameters
Train phase:
function NN_Model
    // Initialize network with layers
    Input layer (loss) // 1 node
    Hidden layers // 256, 512, 256 nodes
    Output layer (hyperparameter) // 7 node
end function
Test phase:
Best hyperparameter = NN_Model(0)

4.3.3.4 Inverse Parameter Prediction Approach(APP)

4.3.3.4.1 Method Details

In the second approach, we implement an inverse optimization strategy by first developing a neural network with seven inputs and one output, designed to map ESN parameters to NRMSE values. The network's architecture progressively transforms the data through several stages: the input layer accepts seven parameters, followed by two processing blocks. The first block expands the dimension to 256 features ($7 \rightarrow 256$), the second block further expands to 512 features ($256 \rightarrow 512$), while the third block contracts back to 256 features ($512 \rightarrow 256$). Finally, a linear layer reduces the dimension to a single output ($256 \rightarrow 1$), representing the predicted NRMSE. Each processing block, except the final layer, combines a linear transformation with layer normalization, ReLU activation, and dropout for regularization.

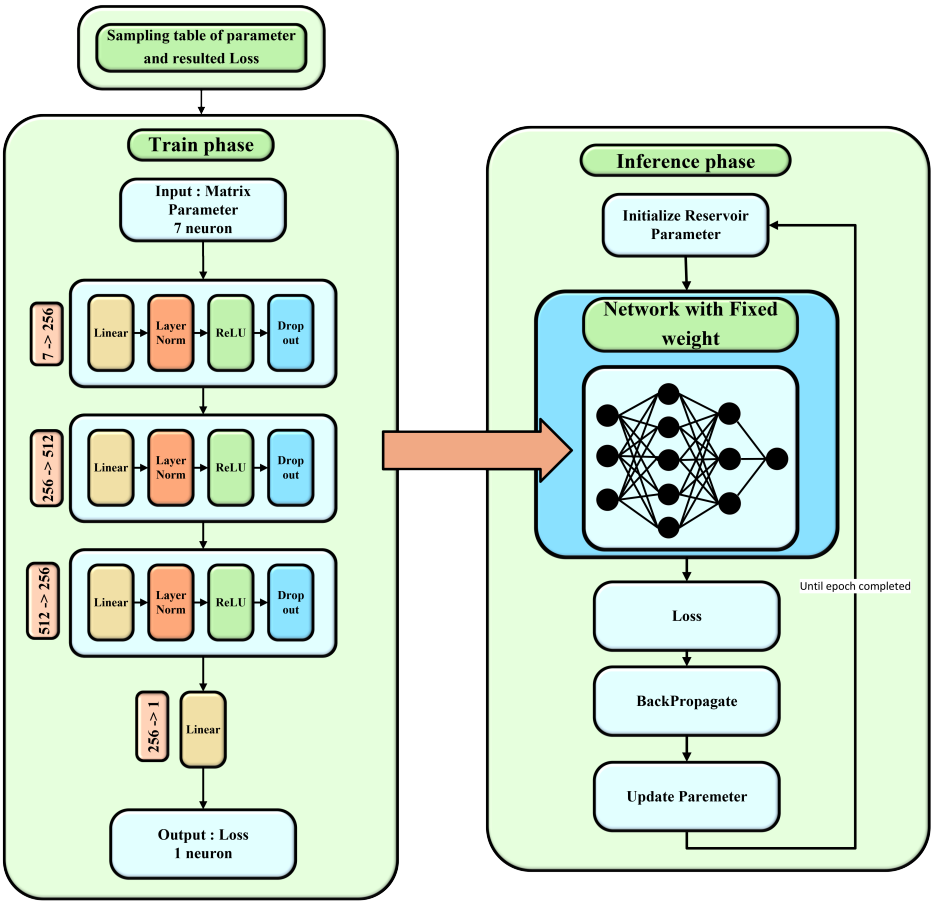

FIGURE 4.5: Neural network architecture for inverse parameter optimization. The network consists of two phases: training phase (left) which learns the mapping from parameters to NRMSE, and inference phase (right) which optimizes parameters through the frozen network to minimize NRMSE.

As shown in Figure 4.5, our architecture implements both training and inference phases. The training phase features a deep neural network that learns to predict NRMSE from network parameters, while the inference phase uses this trained network to optimize parameters through gradient descent.

After training and obtaining the network weights, we freeze the network and treat the input parameters as trainable variables. The loss function is defined as the NRMSE output of the network, which can be expressed as:

$$\text{Loss} = |f(\text{inputs}) - 0| \tag{4.6}$$

Or simply:

$$\text{Loss} = f(\text{inputs}) \tag{4.7}$$

where $f(\text{inputs})$ represents the neural network's output for a given set of seven input parameters. This creates an optimization problem where we seek to find input parameter values that minimize this loss function.

The optimization process uses gradient-based methods to iteratively adjust the input parameters while maintaining their physical constraints. Starting with initialized parameters within predefined bounds, these values are passed through the frozen trained network to compute the NRMSE. The gradients are then calculated with respect to the input parameters, allowing for their systematic adjustment. This process continues until convergence or for a specified number of epochs, effectively searching for optimal input parameters that yield the minimum possible NRMSE value. Throughout this process, all parameters remain within their valid ranges: k and m are constrained between defined bounds, p and α are maintained within valid probability ranges, and weight scales are similarly constrained to ensure physically meaningful values.

4.3.3.4.2 Pseudocode

This approach extends our semi-supervised model by incorporating a parameter optimization function that iteratively refines hyperparameters using gradient descent to minimize the loss.

Algorithm 8 Semi-Supervised Model - Inverse Parameter Prediction

Input:
    List of network hyperparameter ranges
Output:
    Optimal hyperparameters
Train phase:
function NN_MODEL
    // Initialize network with layers
    Input layer (hyperparameter) // 7 node
    Hidden layers // 256, 512, 256 nodes
    Output layer (loss) // 1 node
end function
function FIND_BEST_HYPERPARAMETER
    $param_{old}$ = Initialize hyperparameter value
    freeze weight of NN Model
    Loss = NN_Model($param_{old}$)
    $param_{new} = param_{old} - lr * gradient$
end function
Test phase:
Best hyperparameter = find_best_hyperparameter()

Chapter 5

Experimental Result

5.1 Setup

Our total dataset consisted of 20,000 data points. We divided this data into two parts: 14,000 points for training and 6,000 points for testing.

This setup forms the foundation for all Echo State Network (ESN) models implemented in our research, including:

Parameter	Value
Input Size / Output Size	1
Spectral Radius	1.25
Leaking Rate (α)	0.3
Training / Testing Split	20 / 80
Initial Length	100 time step
Error Length	500 time step
Regularization	4.8e-07
Initial Input Weights value range	[-0.5,0.5]
Initial Reservoir Weights value range	[-0.5,0.5]

TABLE 5.1: Network Parameters and Configuration Values for the Echo State Network Implementation

5.1.1 Setup of Supervised Model

We utilized the Adam optimizer and set a learning rate range between $1e^{-06}$ and $1e^{-03}$. The training process was conducted over a span of 50 to 200 epochs, with the data split allocated as 20% for training and 80% for testing.

Through extensive experimentation with different combinations of reservoir sizes, learning rates, and epoch numbers, we observed interesting patterns in model behavior. Notably, we discovered that each reservoir size required its own specific learning rate - reservoirs with fewer neurons needed higher learning rates to prevent overfitting, while larger reservoirs were stable with lower learning rates within our specified range ($1e^{-06}$ to $1e^{-03}$). This adaptive approach to learning rate selection based on reservoir size helped us maintain model stability and prevent overfitting across different configurations.

5.1.2 Setup of Semi-Supervised Model

For network parameter configuration, we implemented varying ranges based on reservoir size. For small networks (5 nodes), k and m values range from 2 to 4. Medium networks (20 nodes) use ranges of 2 to 7, while larger networks (50 nodes) extend this to 2 to 9. Across all network sizes, we maintained consistent values for rewiring probability [0.1, 0.3, 0.9], network mixing ratio α [0.3, 0.9], and weight scales [0.3, 0.9] for both small-world and scale-free components. Additional configurations for very large networks (500 and 1000 nodes) are prepared but currently disabled.

5.1.2.1 First Approach

For training, we use Mean Squared Error (MSE) as our loss function, paired with the AdamW optimizer. The learning rate starts at 0.0001, with a weight decay of 0.01 for regularization. The model trains for up to 1000 epochs, but includes an early stopping mechanism that triggers after 100 epochs without improvement. To prevent gradient explosions, we clip gradients at 1.0, and we've set a minimum learning rate of $1e^{-6}$ to ensure stable training.

5.1.2.2 Second Approach

Our model used a reservoir size of 10-400, with K and M parameters ranging from 2-12. P parameter was constrained to 0.1-0.5, while α and weight scales operated within 0.1-0.9. Training utilized AdamW optimizer (learning rate: 0.0001, weight decay: 0.01) with MSE loss function. We implemented early stopping (patience: 100 epochs), gradient clipping (1.0), and minimum learning rate ($1e^{-6}$) with an 80/20 train-test split.

Data was normalized to range [-5, 5], removing infinite/NaN values. The optimization used SGD (momentum: 0.9) with gradient clamping over 1000 epochs, and ReduceLROnPlateau scheduler (patience: 50 epochs). Random seed 42 ensured reproducibility.

5.2 Dataset

In this study, we utilized two distinct functions to generate our datasets: the Mackey-Glass (MG) chaotic time series and the Nonlinear AutoRegressive Moving Average (NARMA) model [2]. The Mackey-Glass system [16] is particularly notable for its parameter τ (tau), which determines the degree of nonlinearity in the generated time series. We explored three different values of τ to create varying levels of complexity in our time series data. For our experimental analysis, we constructed a total of four datasets using these generation methods.The coding framework, are documented in Table 5.2.

Name dataset	**Special parameter**	**Code**
Mackey Glass	Tau = 29	A
	Tau = 17	B
	Tau = 10	C
NRAME	-	D
Parameter	Resize = 5, 20	1
	Resize = 5, 20, 50	2

TABLE 5.2: Dataset Parameters and Codes

5.3 Metric

To evaluate the performance of our proposed methods, we employed the Normalized Root Mean Square Error (NRMSE) as our primary evaluation metric. NRMSE was specifically chosen as it provides a standardized measure of prediction accuracy while accounting for the scale of the target data. This metric effectively quantifies the average magnitude of prediction errors between the model's output and actual values, normalized by the range of observed values. The normalization aspect of NRMSE makes it particularly suitable for our analysis as it enables meaningful comparisons across different scales and units of measurement. The NRMSE is calculated as:

$$\text{NRMSE} = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}}{y_{\max} - y_{\min}} \tag{5.1}$$

In this formula, predicted values are denoted by $\hat{y}$ and actual values by y, while n represents the total number of samples. The denominator consists of the range of actual data, calculated as the difference between the maximum ($y_{\max}$) and minimum ($y_{\min}$) observed values.

5.4 Result

5.4.1 Supervised Model

Table 5.3 presents a comprehensive comparison between our proposed Model 1 and a baseline Simple ESN implementation across various network sizes (N = 5, 50, 500, and 1000) for four distinct datasets (A, B, C, and D). The results demonstrate several notable patterns and insights regarding model performance and scalability.

For Dataset A, Model 1 exhibits superior performance at smaller (N = 5) and larger (N = 500, 1000) network sizes, achieving NRMSE values as low as 0.02001 at N = 1000. However, it's worth noting that the model encounters stability issues at N = 50, indicated by the infinite error value. Conversely, the Simple ESN shows more stable but generally higher error rates across all network sizes.

Dataset B reveals a consistent pattern of superior performance by Model 1 across all network configurations. Particularly noteworthy is the dramatic improvement in accuracy as the network size increases, with the NRMSE decreasing from 0.2407 (N = 5) to an impressive 4.415e-06 (N = 1000). This suggests that Model 1 effectively leverages larger network capacities for this particular dataset.

The results for Dataset C demonstrate Model 1's exceptional performance across all network sizes, with remarkably low NRMSE values ranging from 0.005 to 6.33e-06. This represents a significant improvement over the Simple ESN, which shows considerably higher error rates, particularly at smaller network sizes.

Dataset D presents an interesting case where both models achieve comparable performance, with Model 1 maintaining a slight edge across all network sizes. The relatively consistent NRMSE values (approximately 0.22) across different network sizes suggest that this dataset

might present a fundamental complexity threshold that neither model can substantially improve upon through increased network capacity.

These results collectively indicate that Model 1 generally outperforms the Simple ESN architecture, particularly in scenarios involving larger network sizes. The superior performance is most pronounced in datasets B and C, where Model 1 achieves error rates several orders of magnitude lower than the baseline model. However, the performance characteristics vary significantly across different datasets, highlighting the importance of considering dataset-specific properties when selecting optimal network configurations.

Data set	Model name	N = 5	N = 50	N = 500	N = 1000
A	Model 1	**0.22**	Inf	**0.028**	**0.02001**
	Simple ESN	Inf	**0.293**	0.236	0.251
B	Model 1	**0.2407**	**0.0023**	**6.47e-05**	**4.415e-06**
	Simple ESN	0.3337	0.0751	0.0088	0.0194
C	Model 1	**0.005**	**5.09e-5**	**6.84e-05**	**6.33e-06**
	Simple ESN	0.32	0.001	0.005	0.002
D	Model 1	**0.2206**	**0.2198**	**0.2252**	**0.2234**
	Simple ESN	0.2215	0.2209	0.2279	0.2270

TABLE 5.3: Comparison of Supervised Model and Simple ESN across different N values

5.4.2 Unsupervised Model

Table 5.4 presents a comparative analysis of different model configurations, examining their performance using various evaluation metrics. The results are organized across multiple datasets (labeled A1 through D2) and show three main model variants: Loss-feature, Line, and Simple ESN. Looking at the loss measurements, we can observe that the Loss-feature configuration generally demonstrates better performance across most datasets, with values ranging from approximately 0.1000 to 0.2998. The Line configuration shows comparable but slightly higher loss values in many cases. The Simple ESN configuration, while competitive in some instances, typically shows higher loss values compared to the Loss-feature variant. Notably, in datasets C1 and C2, some configurations show 'inf' (infinite) values, suggesting potential convergence issues or limitations in those specific scenarios.

The consistency of these patterns across multiple datasets suggests that the Loss-feature approach offers more robust and reliable performance in this experimental context. The relatively lower loss values achieved by this configuration indicate its superior ability to

minimize prediction errors compared to the alternative approaches. This pattern is particularly evident in datasets A1 and A2, where the Loss-feature configuration maintains a clear advantage over other model variants.

Dataset	Model	Input	Output	Loss avg (Repeat=5)	Loss avg (Repeat=6)
A1	Loss-feature	1	7	**0.1000**	**0.1210**
	feature-Loss	7	1	0.1869	0.1740
	Simple ESN	-	-	0.3080	0.3080
A2	Loss-feature	1	7	0.1573	**0.1472**
	feature-Loss	7	1	**0.1547**	0.1658
	Simple ESN	-	-	0.3011	0.3011
B1	Loss-feature	1	7	0.1652	0.1647
	feature-Loss	7	1	**0.0583**	**0.0504**
	Simple ESN	-	-	0.2463	0.2463
B2	Loss-feature	1	7	0.1684	0.1753
	feature-Loss	7	1	**0.1068**	**0.0407**
	Simple ESN	-	-	0.2473	0.2473
C1	Loss-feature	1	7	0.2998	0.2846
	feature-Loss	7	1	**0.1420**	**0.1210**
	Simple ESN	-	-	Inf	Inf
C2	Loss-feature	1	7	0.304	0.302
	feature-Loss	7	1	**0.185**	**0.198**
	Simple ESN	-	-	Inf	Inf
D1	Loss-feature	1	7	0.1239	0.1293
	feature-Loss	7	1	**0.0601**	**0.0544**
	Simple ESN	-	-	0.2372	0.2372
D2	Loss-feature	1	7	0.1509	0.1552
	feature-Loss	7	1	**0.1092**	**0.1130**
	Simple ESN	-	-	0.2371	0.2371

TABLE 5.4: Comparison of different Unsupervised Model and Simple ESN across different N values

5.5 Discussion

In our approach to generating reservoirs, we initially created graphs with distinct small-world and scale-free properties, then combined them using weighted combinations. Our analysis demonstrates that after weighted combination of these graphs, the resulting hybrid network successfully maintains both small-world and scale-free characteristics. As shown in Figure 5.1, a comprehensive examination of four key metrics reveals how effectively our hybrid matrix integrates these network properties. The small-world index of our hybrid network achieves a value of 2.55, significantly exceeding the critical threshold of 1.0. While this is lower than the pure small-world network's impressive index of 8.74, it convincingly demonstrates that the hybrid network preserves essential small-world characteristics while incorporating scale-free properties. This balanced integration is further supported by the clustering coefficient analysis, where our hybrid network exhibits a coefficient of 0.184. This value strategically positions it between the small-world network (0.316) and scale-free network (0.135), indicating successful preservation of local connectivity patterns while accommodating the broader connectivity typical of scale-free structures. 26 Particularly noteworthy is the hybrid network's performance in average path length, achieving a remarkably efficient 2.32. This value not only improves upon both the small-world (3.53) and scale-free (2.69) networks but approaches the efficiency of random networks (2.11). Such short path lengths are crucial for rapid information transfer across the network, a key characteristic we aimed to preserve from small-world structures. The final metric presents a combined evaluation of small-world and scale-free properties, where our hybrid network achieves a score of 0.42. While the scale-free network shows a higher score of 0.80 due to its pronounced hub structure, our hybrid network's score represents a substantial and balanced integration of both network characteristics.

To further validate the preservation of these essential network properties, Figure 5.1 presents a detailed structural analysis of the hybrid network's connection patterns. The weight matrix reveals a well-balanced distribution of positive and negative connections, demonstrating how our parametric combination approach successfully merges the local connectivity patterns typical of small-world networks with the long-range connections characteristic of scale-free architectures. This integration is particularly evident in the adjacency matrix, which displays a prominent diagonal band indicating strong local clustering (a key small-world property), while the scattered off-diagonal connections represent the long-range links that facilitate hub formation (a defining scale-free characteristic). The weight distribution histogram further confirms this successful integration, showing a balanced, approximately normal distribution centered around zero, with appropriate

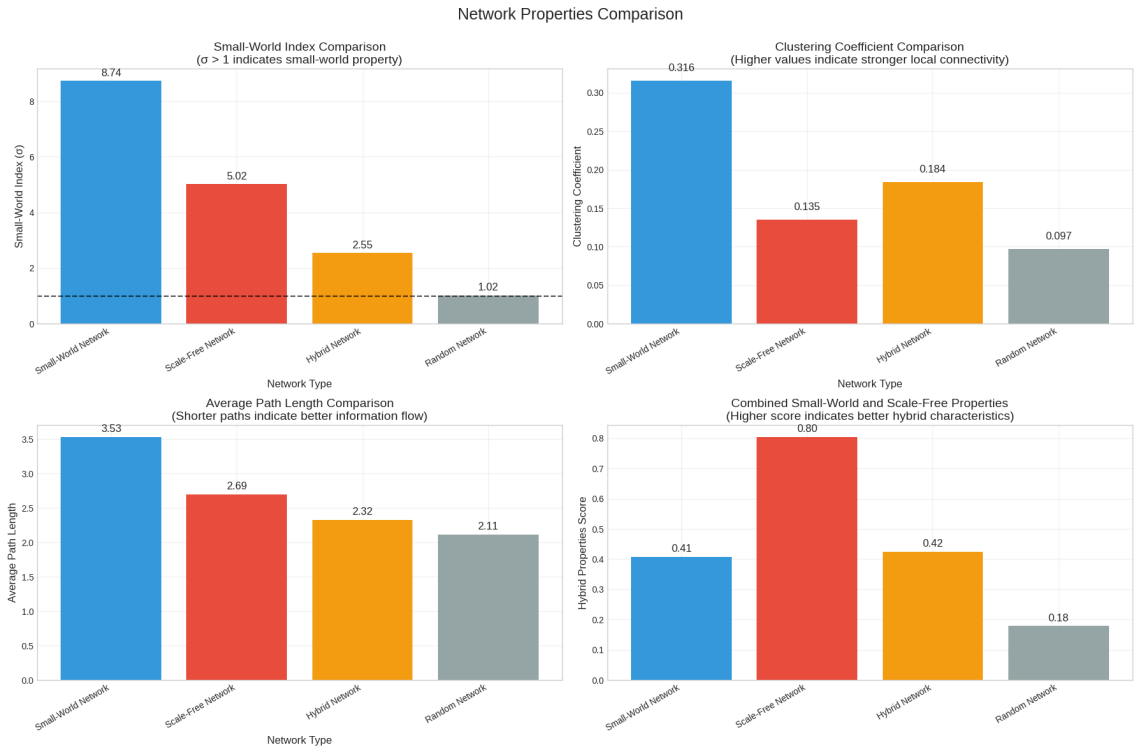

FIGURE 5.1: Comparative analysis of key network metrics across different network architectures

tails extending in both directions. The network visualization provides a final confirmation of our hybrid approach's success, clearly displaying both the clustered structures typical of small-world networks and the high-connectivity nodes characteristic of scale-free networks. Together, these structural analyses reinforce our earlier metric-based findings, demonstrating that our hybrid network effectively preserves and balances both small-world and scale-free properties through the weighted combination process.

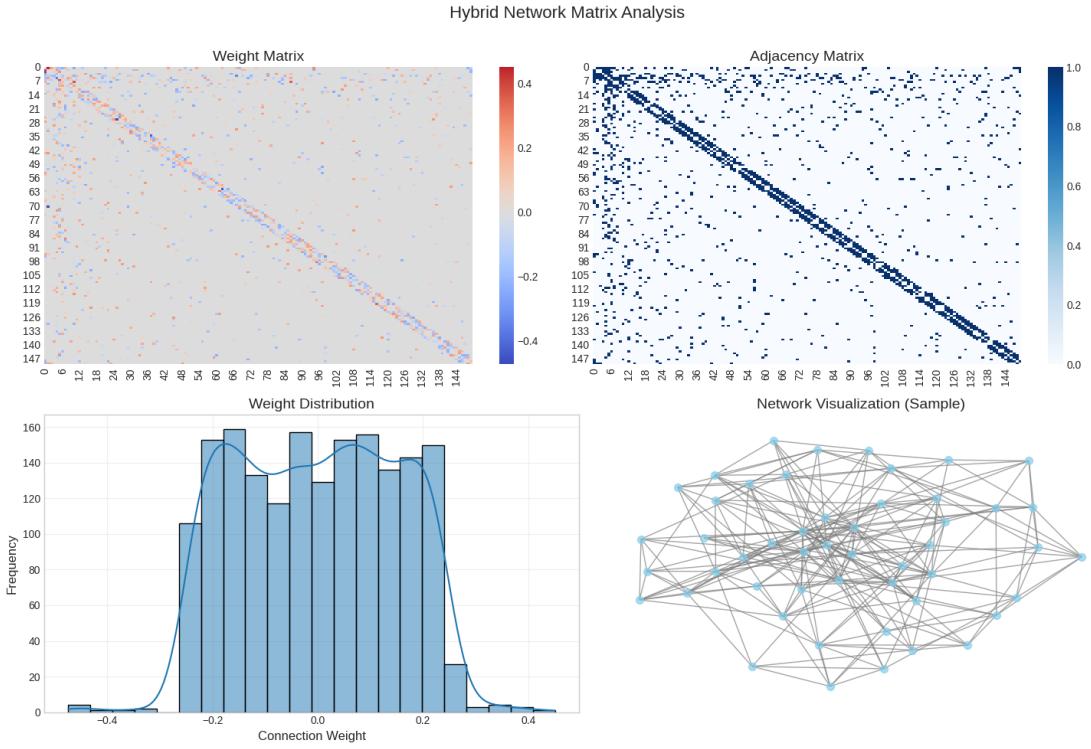

FIGURE 5.2: Structural analysis of the hybrid network matrix

Having established the successful preservation of both small-world and scale-free properties in our hybrid network, we next validated the effectiveness of our hyperparameter selection strategy. The performance of complex networks like ours is highly dependent on the choice of hyperparameters, and selecting optimal values is crucial for achieving desired performance characteristics.

Figure 5.2 provides compelling evidence for the effectiveness of our hyperparameter selection methodology. The visualization shows NRMSE values (≤ 1.15) across different hyperparameter combinations, demonstrating the significant influence these parameters have on network performance. The blue line tracks loss variations across different parameter combinations, while the gray band represents variance bounds around the mean.

Our NRMSE analysis reveals substantial fluctuations across different hyperparameter combinations, providing clear evidence of their influence on network performance. As observed in Figure 5.2, the majority of NRMSE values oscillate around the mean of 0.3009, with several notable spikes exceeding 0.8. These variations validate our parameter selection methodology and confirm hyperparameters' role as effective control mechanisms for network optimization. The consistent pattern of fluctuations within the variance bounds ($\pm\sigma$) further supports the statistical significance of our findings.

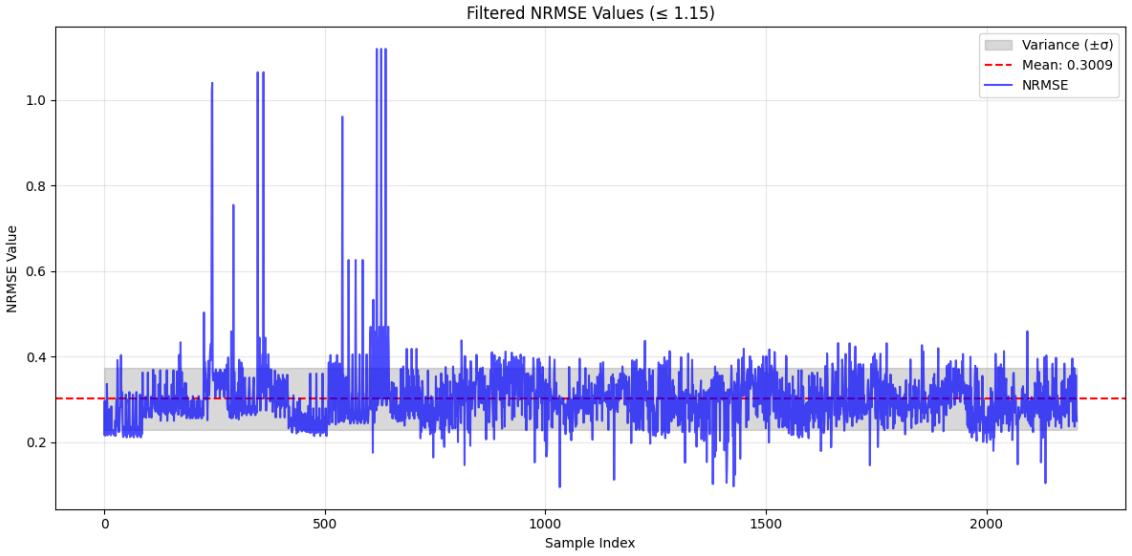

FIGURE 5.3: NRMSE values ($\leq$ 1.15) across different hyperparameter combinations, demonstrating parameter influence on network performance. The blue line shows loss variations, while the gray band represents variance bounds around the mean

To gain deeper insight into how individual hyperparameters affect network performance, we conducted a comprehensive correlation analysis between each parameter and the NRMSE values. Figure 5.3 presents this analysis from two complementary perspectives: the left panel shows parameter correlations across different datasets (A1-D2), while the right panel displays the average effect of each parameter across all datasets combined. Our correlation analysis reveals distinctive patterns in how different hyperparameters influence network performance. As shown in Figure 5.3, the node connectivity parameters 'm' and 'k' demonstrate substantial impact, reaching approximately half the influence of the reservoir size parameter. This observation is particularly significant as it validates our hybrid network design approach, showing that edge connectivity patterns play a crucial role in determining network performance. The consistency of these correlations across different datasets (A1-D2) provides strong evidence for the robustness of our parameter selection methodology. Moreover, the averaged correlations (right panel) offer a clear picture of each parameter's overall influence, helping us identify which parameters are most critical for optimizing network performance. This systematic analysis not only validates our hyperparameter choices but also provides valuable insights for future network optimization strategies.

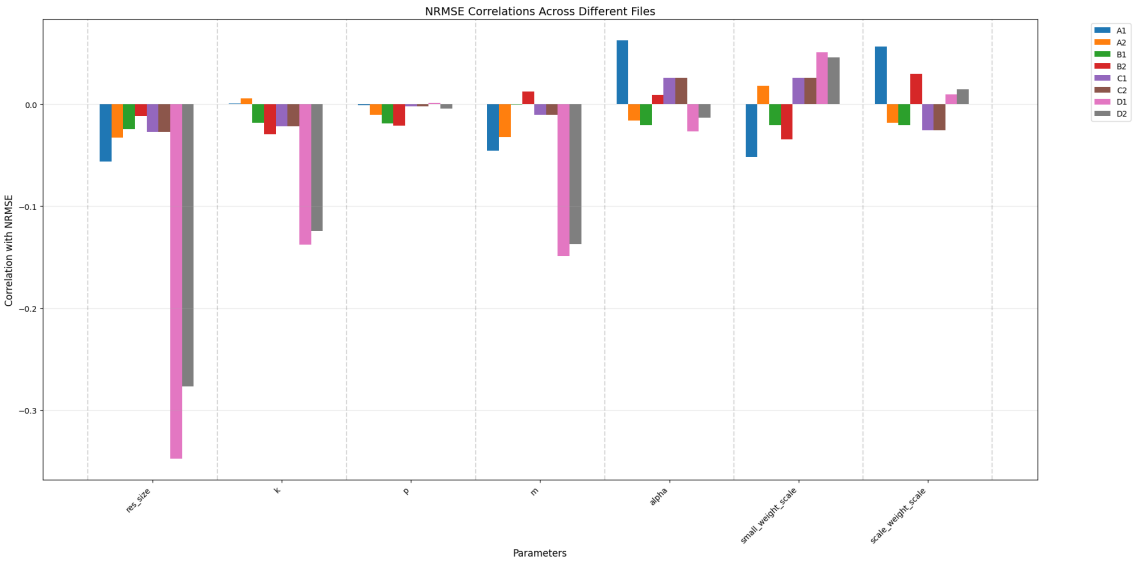

FIGURE 5.4: The figure shows parameter correlations with NRMSE in two parts. The plot displays a detailed comparison of parameters across eight different cases (A1 to D2), separated by vertical lines for better readability.

Chapter 6

Conclusion and Future work

6.1 Introduction

The optimization of Echo State Networks for time series prediction remains a significant challenge, particularly in determining optimal reservoir configurations. This paper has explored both theoretical foundations and practical implementations for enhancing ESN performance through reservoir optimization. Our findings reveal important insights about the relationship between network topology and prediction accuracy, while suggesting promising directions for future development.

6.2 Detailed Research Plan

In this work, we first established theoretical foundations through two key theorems that demonstrate the relationship between reservoir properties and network performance. Building on these theoretical insights, we proposed two approaches: a supervised method that directly optimizes reservoir weights, and a semi-supervised approach that combines small-world and scale-free network characteristics. Through systematic analysis of the semi-supervised approach, our comprehensive study across multiple datasets revealed significant insights into the relative importance of hyperparameters in hybrid network architectures. Through systematic correlation analysis, we demonstrated that reservoir size consistently emerges as the primary determinant of network performance. Notably, the edge connectivity parameters (k in Watts-Strogatz and m in Barabási-Albert models) show substantial influence at approximately half the impact of reservoir size. This finding is particularly

significant as both parameters fundamentally control the edge density in their respective network models - k determining the initial number of edges per node in the Watts-Strogatz model, and m governing both the initial complete graph size (m+1) and the number of edges added with each new node in the Barabási-Albert model. The consistent importance of these edge-related parameters across different network formation mechanisms suggests that edge density plays a crucial role in reservoir computing performance, second only to the reservoir size itself. This parallel influence of k and m, despite their implementation in different network growth models, provides strong evidence for the fundamental importance of edge connectivity in hybrid reservoir architectures. Looking ahead, we aim to leverage these insights by conducting a comprehensive analysis across various benchmark datasets. Our goal is to establish clear relationships between different types of time series data and optimal edge density configurations, along with other network parameters. This systematic approach will enable us to develop robust guidelines for unsupervised reservoir generation, potentially automating the network design process based on input data characteristics. Through this future work, we anticipate developing a more nuanced understanding of how different data types interact with network connectivity patterns, leading to more efficient and automated approaches to reservoir computing network design. The strong correlation between edge-related parameters and network performance suggests that edge density optimization could be a key factor in developing these automated design strategies.

Bibliography

[1] Akrami, A., Rostami, H., and Khosravi, M. R. (2020). Design of a reservoir for cloud-enabled echo state network with high clustering coefficient. *Eurasip Journal on Wireless Communications and Networking*, 2020.

[2] Atiya, A. and Parlos, A. (2000). New results on recurrent network training: unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11(3):697–709.

[3] Barabási, A.-L. and Albert, R. (1999). arxiv:cond-mat/9910332v1 [cond-mat.dis-nn] 21 oct 1999.

[4] Chen, H. C. and Wei, D. Q. (2021). Chaotic time series prediction using echo state network based on selective opposition grey wolf optimizer. *Nonlinear Dynamics*, 104:3925–3935.

[5] de la Fraga, L. G., Ovilla-Martínez, B., and Tlelo-Cuautle, E. (2023). Echo state network implementation for chaotic time series prediction. *Microprocessors and Microsystems*, 103.

[6] Deng, Z. and Zhang, Y. (2006). Collective behavior of a small-world recurrent neural system with scale-free distribution.

[7] Ebato, Y., Nobukawa, S., Sakemi, Y., Nishimura, H., Kanamaru, T., Sviridova, N., and Aihara, K. (2024). Impact of time-history terms on reservoir dynamics and prediction accuracy in echo state networks. *Scientific Reports*, 14.

[8] Freiberger, M., Bienstman, P., and Dambre, J. (2020). A training algorithm for networks of high-variability reservoirs. *Scientific Reports*, 10.

[9] Goswami, D. (2024). Feature-based echo-state networks: A step towards interpretability and minimalism in reservoir computer.

[10] Han, M. and Xu, M. (2018). Laplacian echo state network for multivariate time series prediction. *IEEE Transactions on Neural Networks and Learning Systems*, 29:238–244.

[11] Jaeger, H. (2010). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note 1.

[12] Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304:78–80.

[13] Kim, T. and King, B. R. (2020). Time series prediction using deep echo state networks. *Neural Computing and Applications*, 32:17769–17787.

[14] Köster, F., Patel, D., Wikner, A., Jaurigue, L., and Lüdge, K. (2023). Data-informed reservoir computing for efficient time-series prediction. *Chaos*, 33.

[15] Li, D., Han, M., and Wang, J. (2012). Chaotic time series prediction based on a novel robust echo state network. *IEEE Trans. Neural Networks and Learning Systems*, 23:787–799.

[16] Mackey, M. C. and Glass, L. (1977). Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289.

[17] Na, X., Ren, W., Liu, M., and Han, M. (2023). Hierarchical echo state network with sparse learning: A method for multidimensional chaotic time series prediction. *IEEE Transactions on Neural Networks and Learning Systems*, 34:9302–9313.

[18] Najibi, E. and Rostami, H. (2015). Scesn, spesn, swesn: Three recurrent neural echo state networks with clustered reservoirs for prediction of nonlinear and chaotic time series. *Applied Intelligence*, 43:460–472.

[19] Racca, A. and Magri, L. (2021). Robust optimization and validation of echo state networks for learning chaotic dynamics. *Neural Networks*, 142:252–268.

[20] Seghier, M. E. A. B., Truong, T. T., Feiler, C., and Höche, D. (2025). A hybrid deep learning model for predicting atmospheric corrosion in steel energy structures under maritime conditions based on time-series data. *Results in Engineering*, page 104417.

[21] Shahi, S., Fenton, F. H., and Cherry, E. M. (2022). Prediction of chaotic time series using recurrent neural networks and reservoir computing techniques: A comparative study. *Machine Learning with Applications*, 8:100300.

[22] Shterev, V. A., Metchkarski, N. S., and Koparanov, K. A. (2022). Time series prediction with neural networks: a review. In *2022 57th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*, pages 1–4.

[23] Siqueira, H., Boccato, L., Attux, R., and Filho, C. L. (2012). Lncs 7664 - echo state networks and extreme learning machines: A comparative study on seasonal streamflow series prediction.

[24] Sun, X., Li, T., Li, Q., Huang, Y., and Li, Y. (2017). Deep belief echo-state network and its application to time series prediction. *Knowledge-Based Systems*, 130:17–29.

[25] Sun, Y., Zhang, L., and Yao, M. (2023). Chaotic time series prediction of nonlinear systems based on various neural network models. *Chaos, Solitons and Fractals*, 175.

[26] Viehweg, J., Worthmann, K., and Mäder, P. (2023). Parameterizing echo state networks for multi-step time series prediction.

[27] Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393:440–442.

[28] Xu, M. and Han, M. (2016). Adaptive elastic echo state network for multivariate time series prediction. *IEEE Transactions on Cybernetics*, 46:2173–2183.

[29] Xu, M., Han, M., Qiu, T., and Lin, H. (2019). Hybrid regularized echo state network for multivariate chaotic time series prediction. *IEEE Transactions on Cybernetics*, 49:2305–2315.

[30] Yang, C., Qiao, J., Han, H., and Wang, L. (2018). Design of polynomial echo state networks for time series prediction. *Neurocomputing*, 290:148–160.

[31] Zhang, L., Wang, R., Li, Z., Li, J., Ge, Y., Wa, S., Huang, S., and Lv, C. (2023). Time-series neural network: A high-accuracy time-series forecasting method based on kernel filter and time attention. *Information*, 14(9).

[32] Zhong, S., Xie, X., Lin, L., and Wang, F. (2017). Genetic algorithm optimized double-reservoir echo state network for multi-regime time series prediction. *Neurocomputing*, 238:191–204.