Research internship report

Master 1 - Electrical Energy, Electronics, Automation

Bifurcation Embedding in One-Dimensional Chaotic Dynamics Using Reservoir Computing

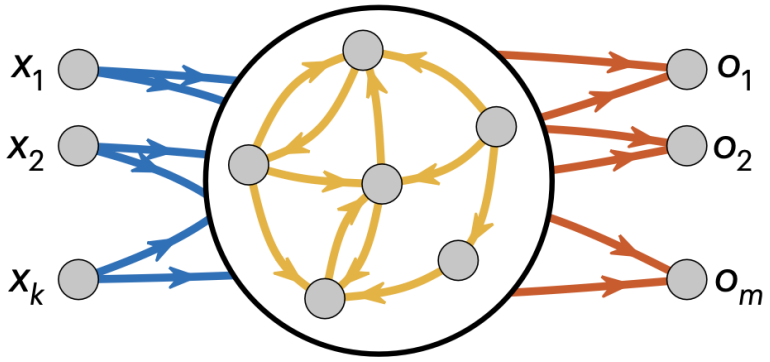

Figure 1: Scheme of a Recurrent Neural Network (RNN) in open-loop

Student : Rémi Al Ajroudi

Supervisor : Professor Kohei Nakajima

Laboratory : Intelligent Systems and Informatics Laboratory

Duration of the internship : from May 2024 to August 2024, 3 months

Contents

1 Introduction

The Master's 1 program in Electronics, Electrical Energy, and Automation at the University of Paris-Saclay and the École Normale Supérieure Paris-Saclay concludes with a research internship of at least 10 weeks in a laboratory or company. In my case, I completed a 3-month internship from 18th May 2024 to 16th August 2024 at one of the University of Tokyo's laboratories, called the "Intelligent Systems and Informatics Laboratory". This internship focused on machine learning and the study of nonlinear dynamics. The University of Tokyo is renowned as a world-class institution and a leader in Asia. Every August, final-year high school students take a highly selective entrance exam for a chance to enroll. The university has three campuses in Tokyo. The main campus, where my office was located, is in the Hongo district. The second campus is in Komaba, and the third is in the Kashiwa area. Beyond my research work, this experience allowed me to immerse myself in the Japanese society. The university offers various extracurricular activities, and I had the opportunity to join the university's boxing club located on the Komaba campus. This engagement provided me with a perspective on the student life in Todai (name given in Japan to the University of Tokyo) and complemented my academic experience. Through this internship, I aimed to deepen my understanding of advanced machine learning techniques and contribute to ongoing research projects.

Figure 2: Tokyo University, Komaba Campus

Figure 3: Tokyo University, Komaba Campus

Figure 4: Tokyo University, Hongo Campus

Figure 5: Tokyo University, Hongo Campus

1.1 Presentation of the laboratory

My internship took place at the University of Tokyo, in the Intelligent Systems and Informatics Laboratory (ISI Lab.) under the supervision of Professor Kohei Nakajima, from May 18th 2024 to August 17th 2024. The Intelligent Systems and Informatics Laboratory (ISI Lab at the University of Tokyo, part of the Graduate School of Information Science and Technology, is dedicated to advancing research in intelligent systems, robotics, and cognitive computing. The lab focuses on human-machine interaction, intelligent information processing, and brain-inspired computational models. Their work are related to biomedical, robotics, user interfaces. Below you can see the workspace and my personal desk.

Figure 6: My personal desk

Figure 7: Workspace

Within this lab, Professor Kohei Nakajima studies the areas of soft robotics and reservoir computing. His research explores how soft, flexible materials can be used in robotics to create machines that take profit of their physical properties to interact more naturally with their environments. He mainly focuses on reservoir computing, a branch of machine learning which is used to develop new computational models capable of real-time processing and adaptation.

1.2 Subject of the internship

During my time in the ISI lab, the subject of my research internship was around non-linear, attractor, chaotic dynamic replication, intrapolation and extrapolation using reservoir computing. Many recent papers, mentioned on the bibliography, were used as a conduct line to follow. The goal was to replicate the experiments they have conducted. The researchers conducted their experiments on a certain type of predictable yet chaotic dynamics such as the Lorentz attractor, and the goal of my internship was to conduct the same kind of experiment on a different set of dynamic, while fine-tuning the parameters used for their experiments so that it fits this experiment.

This internship is part of the growing field of soft robotics. Soft robots offer significant advantages over more conventional robots, which have more predictable movements. Due to their nature, soft robots can navigate through confined spaces, which is highly valuable in disaster scenarios such as earthquakes. Another application is biomimicry, such as mimicking the swimming of a fish in water. This is the focus of some researchers at the ISI laboratory (cf. Fig. 8), who are replicating a life-sized fish using materials like rubber, whose physical properties can be used for computation. This topic is discussed in more detail in the section on Physical Reservoir Computing.

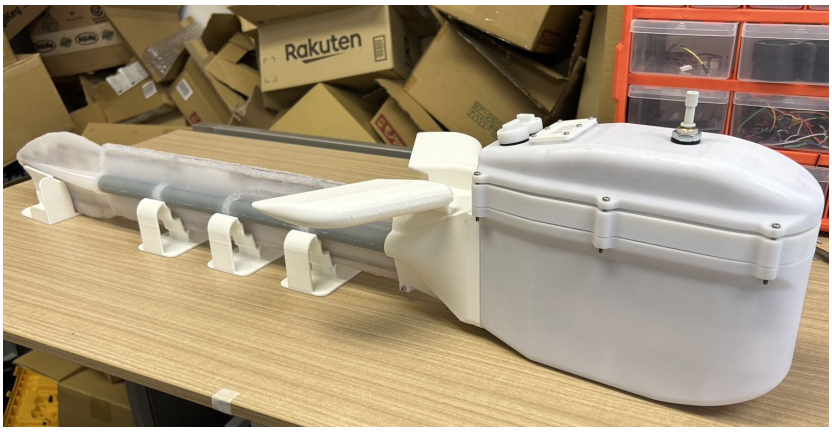

Figure 8: The fish soft-robot that the ISI lab is making

1.3 Training on the Subject

To ensure the success of my internship, I attended a 2-hour weekly course on reservoir computing, followed by practical exercises lasting between 4 to 8 hours, from April 19, 2024, to June 29, 2024. In addition to this, I participated in a seminar on reservoir computing, where each week, for about 3 hours, three speakers (students at the University of Tokyo, PhD candidates, post-doctoral researchers, and professors) presented an assigned paper related to the topic of reservoir computing. These sessions were very enriching, and I presented the paper titled "Teaching recurrent neural networks to infer global temporal structure from local examples." You can find the seminar recording on YouTube (cf. ref [1]).

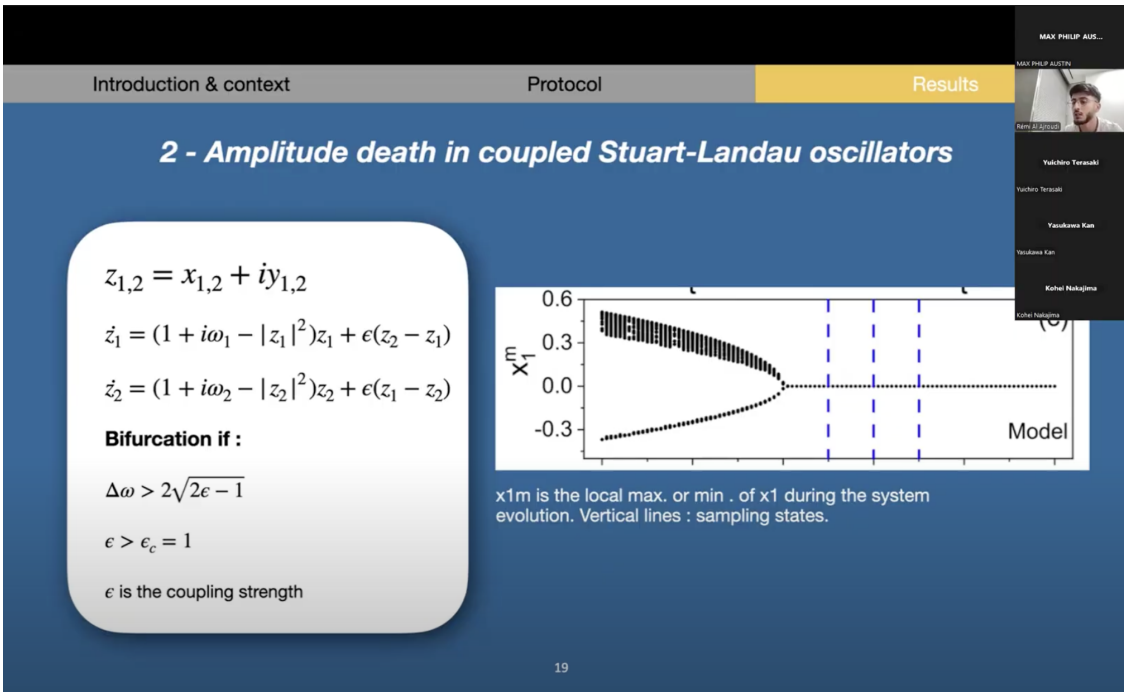

Figure 9: My presentation in the RC seminar

2 Technical Notions

In this section, technical concepts related to the internship are presented, such as RNNs, reservoir computing, dynamical systems, bifurcations, etc.

2.1 RNN

A Recurrent Neural Network (RNN) is a type of neural network designed to process sequences of data. Unlike traditional neural networks, RNNs have recurrent connections that allow them to maintain a memory of the network's past state, making them particularly suitable for analyzing time series and sequential data. This architecture enables RNNs to capture long-term dependencies in data, which is essential for applications such as speech recognition, machine translation, and language modeling.

2.2 Reservoir Computing

Reservoir computing is a computational framework for training recurrent neural networks (RNNs), which are well-suited for processing temporal data due to their inherent ability to maintain a state that can represent past inputs. Unlike traditional RNN training methods, reservoir computing simplifies the learning process by using a fixed, randomly initialized recurrent network, known as the reservoir, and only training the output layer. The reservoir is a high-dimensional, dynamic system that projects the input data into a rich feature space, allowing complex patterns and dependencies in the data to be more easily captured. This approach reduces the computational cost and difficulty associated with training RNNs, making it particularly effective for tasks involving time-series prediction, classification, and signal processing. By adjusting the parameters of the reservoir, such as the spectral radius and sparsity, one can control the network's dynamics and memory capacity, fine-tuning it to different applications.

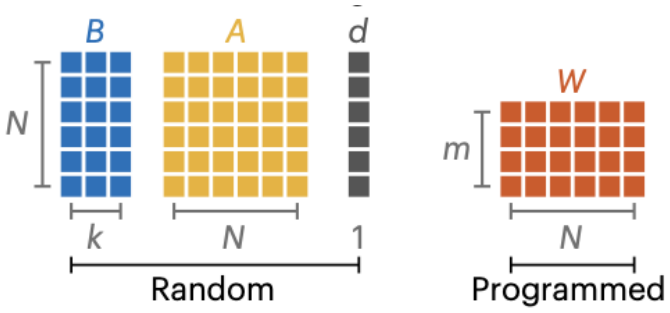

Figure 10: The readout layer is the only one that is programmed, the other are set randomly (2)

As showed on the figure 10 the input matrix B, the reservoir A and the bias vector d are all set randomly using a uniform law or a gaussian law while the readout layer W is programmed using the ridge regression or a linear regression in order for the output $o_{t+1} = Wr_{t+1}$ with $r_{t+1} = tanh(Ar_t + Bx_t + d\beta)$ to match the desired output. This part is detailed later.

2.3 Echo State Network (ESN)

ESN is a type of recurrent neural network (RNN) proposed by H. Jaeger (2001). Unlike feedforward networks commonly used in deep learning, RNNs are characterized by their recursive connections (Figure 1). Additionally, unlike standard RNNs (such as Long Short-Term Memory (LSTM) networks), the recurrent connections in an ESN remain fixed, and only the external parameters, known as the

readout layer, are adjusted. An RNN configured in this manner is referred to as a reservoir, making an ESN a type of reservoir computing.

The defining feature of an echo state network is its "echo state property." This property ensures that two states with different initial conditions will eventually converge to the same output response over time.

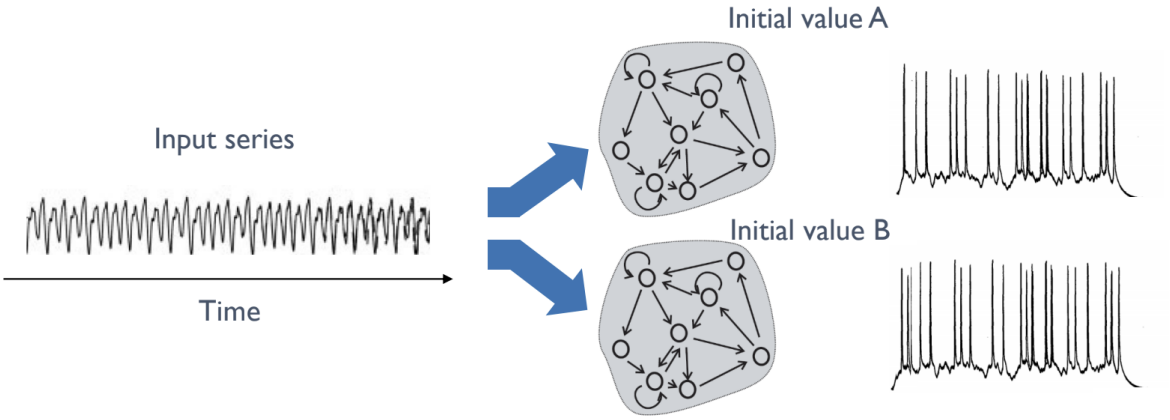

Figure 11: Effect of the initial conditions of the reservoir on the predicted time series

As illustrated in Figure 11, the initial conditions do not affect the output time series after a certain period. To verify if an ESN exhibits the echo state property, we calculate the distance between the states of two ESNs at time step t, given by $d_t = \left\| x_t^{(1)} - x_t^{(2)} \right\|_2$. The system possesses the echo state property if this distance converges to zero over time. Additionally, to evaluate long-term convergence, we can compute the distance averaged over the last S steps: $\bar{d} = \frac{1}{S} \sum_{t=T-S+1}^{T} d_t$.

- **Typical ESN formulation**: The reservoir state $\mathbf{x}_{n+1}$ is updated using the activation function f, input weights $\mathbf{W}_{\text{in}}$, and internal weights $\mathbf{W}$, as follows:

$$\mathbf{x}_{n+1} = f(\mathbf{W}\mathbf{x}_n + \mathbf{W}_{\text{in}}\mathbf{u}_{n+1}) \tag{1}$$

  The output $\hat{\mathbf{y}}_n$ is obtained through a simple linear regression on the reservoir state:

$$\hat{\mathbf{y}}_n = \mathbf{W}_{\text{out}}\mathbf{x}_n \tag{2}$$

- **Training phases**: The learning process of an ESN is divided into three phases:
  1. **Washout phase**: Elimination of initial transients to make the data stationary.
  2. **Training phase**: Adjustment of output weights $\mathbf{W}_{\text{out}}$ through linear regression.
  3. **Testing phase**: Evaluation of the model's generalization on unknown data.

ESNs often use Mean Square regression or Ridge Regression to adjust the output weights, minimizing the error between the predictions and the target values. Below are the hyperparameters that are used to configure an ESN:

- **Reservoir size** : The number of neurons in the reservoir. As you can see on figure 12 the NMRSE between output of an ESN and desired output decreases as the number of nodes in the reservoir increases. However increasing the number of nodes will also increase a lot the time needed for the computer to complete the computation.

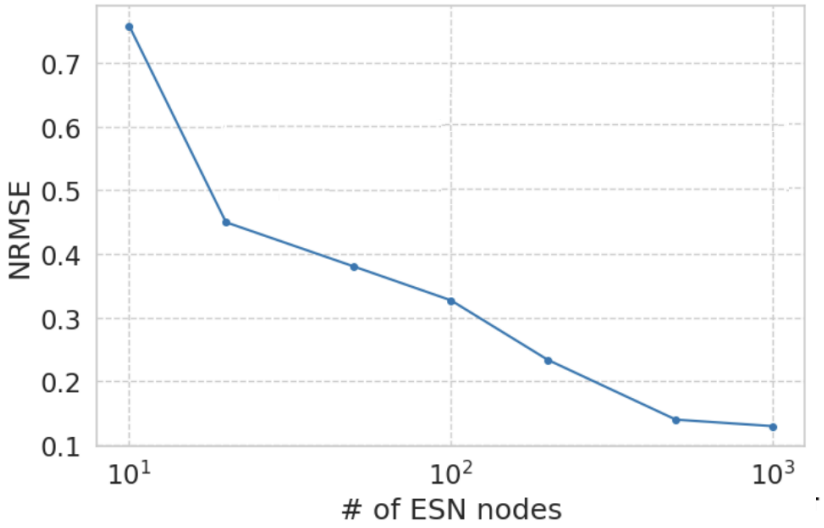

Figure 12: Evolution of NMRSE between output of an ESN and desired output, when the number of nodes of the ESN grow (1)

- **Spectral radius** ρ: The maximum absolute value of the eigenvalues of $\mathbf{W}$. $\rho = \max_i(|\lambda_i|)$. This is a crucial parameter for the stability of the reservoir state. A ρ less than 1 ensures that the state remains stable over time.
- **Activation function** f : The activation function f is a nonlinear transformation applied to the inputs of the reservoir. It plays a key role in the network's ability to capture and represent complex dynamics of the input data. Commonly used activation functions in ESNs include the hyperbolic tangent function (tanh) and the sigmoid function. The nonlinearity introduced by these functions allows the reservoir to project input data into a higher-dimensional space, facilitating the learning of nonlinear patterns by the output layer.

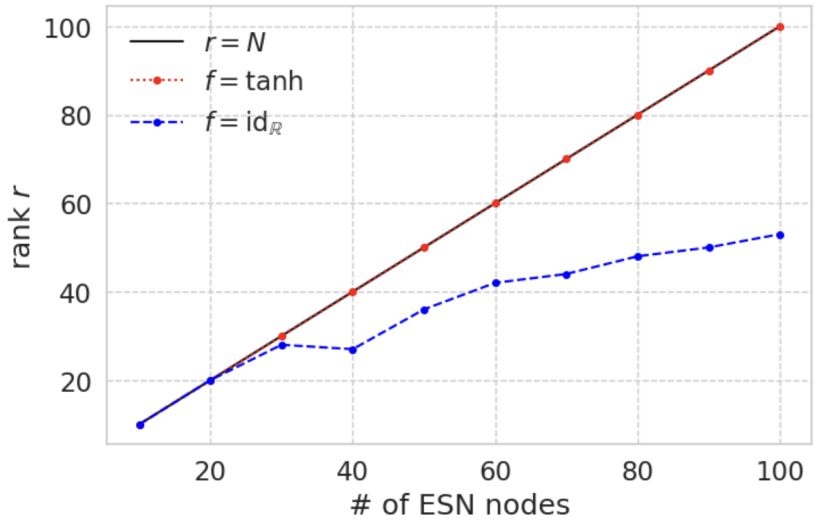

Figure 13: Effect of activation functions on the reservoir rank.

The figure 13 shows how different activation functions affect the rank of the reservoir matrix in Echo State Networks. Nonlinear activation functions, like $f = \tanh$, increase the computation capabilities of the reservoir by enhancing its rank. In contrast, linear functions, such as $f = \mathrm{id}_R$, result in lower ranks, indicating a reduced capacity to model complex dynamics.

- **Sparsity** : Sparsity refers to the proportion of non-zero connections in the reservoir weight matrix $\mathbf{W}$. A high level of sparsity means that most neurons in the reservoir are weakly connected to each other. Each neuron reacts differently to the inputs so sparsity can help prevent overfitting and maintain rich and varied dynamics, essential for learning complex tasks.

- **Intensity/Input scaling** σ : The scaling factor for the input weights $\mathbf{W}_{\text{in}}$. A larger value of σ increases the range of input signals considered by the reservoir, thereby influencing the network's dynamics. Increasing the intensity σ leads to a higher rank of the reservoir, allowing it to model more complex patterns and interactions within the input data. Additionally, as σ increases, the degree of input increases, meaning that the network can utilize a broader range of input values, which further enhances its capacity to explore both linear and nonlinear regions of the activation function, leading to richer temporal representations.

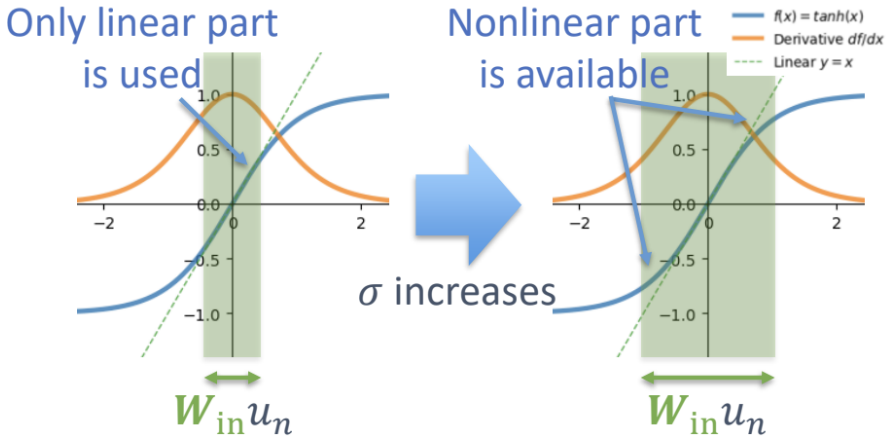

Figure 14: The effect of input scaling σ on the utilization of the activation function's linear and nonlinear regions in Echo State Networks (ESNs).

As shown in Figure 14, when σ increases, the network transitions from primarily using the linear region (left) of the activation function to utilizing the nonlinear region (right). This shift enhances the reservoir's computational capabilities by enabling more complex and varied dynamics, which are essential for capturing intricate temporal patterns in the input data.

- **Spectral Radius** ρ : The spectral radius is defined as the largest absolute eigenvalue of the reservoir's weight matrix $\mathbf{W}$.

$$\rho = \max_i \left(|\lambda_i| \right)$$

It controls the dynamics of the reservoir states by influencing how much past states contribute to the current state. A larger ρ allows the reservoir to maintain longer memory of past inputs, which can be beneficial for tasks requiring long-term dependencies.

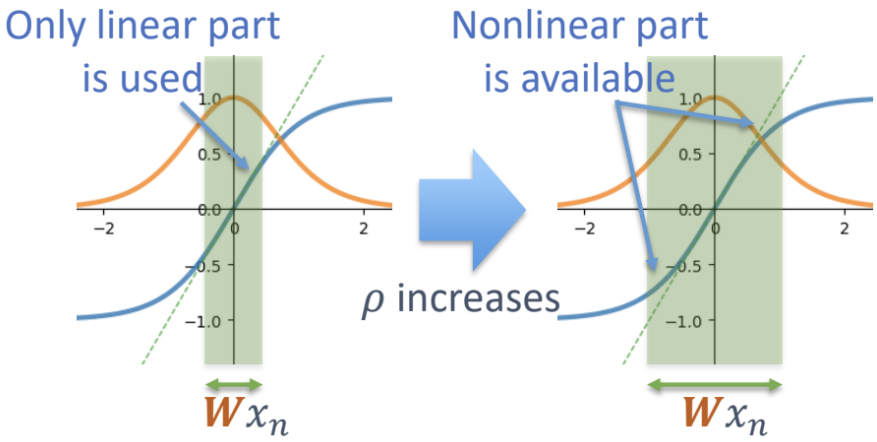

Figure 15: The effect of the spectral radius ρ on the performance and rank of the reservoir in Echo State Networks (ESNs). As ρ increases, the network transitions from utilizing the linear region to accessing the nonlinear region of the activation function, enhancing the reservoir's expressiveness and ability to model complex dynamics.

As shown in Figure 15, increasing ρ has several effects:

1. Rank Increase: A higher spectral radius typically increases the rank of the reservoir matrix, allowing for more diverse and complex internal states.
2. Order of Input Increase: With a larger ρ, the reservoir incorporates a wider range of past input states, effectively increasing the order of the input's influence over time.
3. Maximum Delay of Input Increase: The maximum delay or memory of past inputs that the network can handle also increases with ρ, enabling the ESN to model longer temporal dependencies in the input data.

These effects make adjusting the spectral radius a crucial parameter for optimizing the ESN's performance on different tasks, particularly those that require capturing complex temporal patterns.

- **Leakage rate** α: The parameter controlling the decay rate of the reservoir state. It ranges from 0 to 1, where $\alpha = 0$ corresponds to no leakage (full memory), and $\alpha = 1$ represents complete update at each time step. The state update equation incorporating the leakage rate is:

$$\mathbf{x}_{n+1} = (1 - \alpha)\mathbf{x}_n + \alpha\mathbf{x}_{\text{next}} \tag{3}$$

where $\mathbf{x}_n$ is the current state, $\mathbf{x}_{\text{next}}$ is the next state before leakage, and α is the leakage rate.

Reservoir Computing (RC) is a framework that utilizes reservoirs to address various time series tasks and is classified as supervised learning for time series data. In a typical RC setup, a time series is fed into the internal state of the reservoir as input at each time step, and the internal state evolves according to recursive combinations and nonlinear functions. RC can realize many nonlinear input/output relationships by tuning only the external readout layer, without adjusting the internal recursive connections of the reservoir.

Key features of RC include the high dimensionality and nonlinearity of the reservoir. The high-dimensional space allows for storage of a larger amount of input information as history, while the nonlinear activation function produces various nonlinear transformations of input, which are stored and exploited as computational resources.

Other important features of RC include:

- Use of activation functions for nonlinearity (similar to RNNs)
- Randomly initialized and fixed recurrent connection weights
- Simple training of the readout layer, resulting in lower training costs compared to backpropagation
- Capability for multitasking (multiplexing; using different readouts for different tasks, particularly useful for resource conservation)
- Parameter-aware RC (utilizing a parameter characteristic of the desired dynamics during both training and prediction phases)

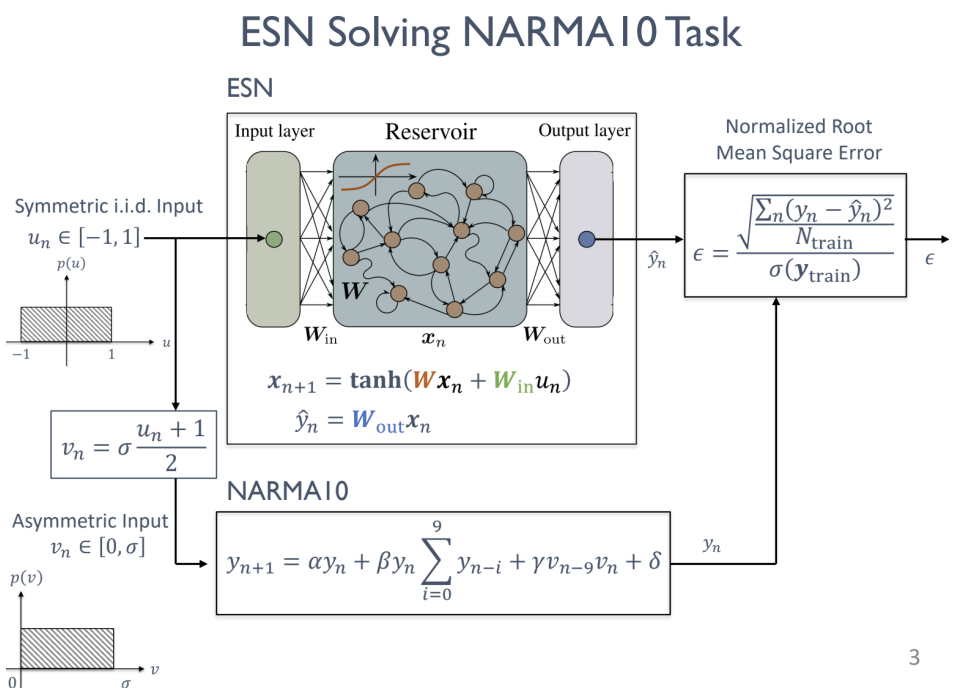

Figure 16: ESN Solving NARMA10 Task

Figure 16 illustrates how an Echo State Network is configured to solve the NARMA10 task, a benchmark problem in time series prediction. NARMA10 (Nonlinear AutoRegressive Moving Average of order 10) is a system identification task that requires the network to model a complex, nonlinear dynamical system with long-term dependencies. The task involves predicting the output of a 10th-order nonlinear autoregressive moving average equation, given a random input sequence. This task is particularly challenging due to its high nonlinearity and the need to consider a history of 10 time steps, making it an excellent benchmark for assessing the memory capacity and nonlinear modeling capabilities of RC systems.

A parameter-aware RC has the specificity to also have a parameter that is a characteristic of the desired dynamic, used during training, and that is also fed to the reservoir during the prediction phase. This allows the reservoir to adapt its dynamics to the desired dynamic in closed-loop.

As you can see on the figures 23 & 17, there are two configurations for the parameter-aware reservoir computer, the training phase and the predicting phase. During the training phase, there are two inputs. The first one is the vector corresponding to the desired input and the second one is the

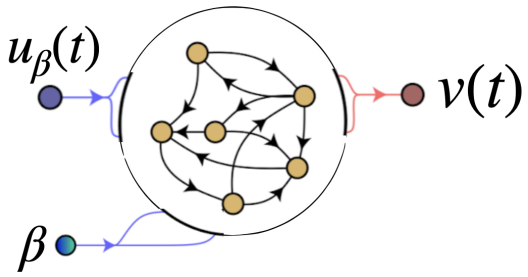

Figure 17: Parameter-aware RC, training phase

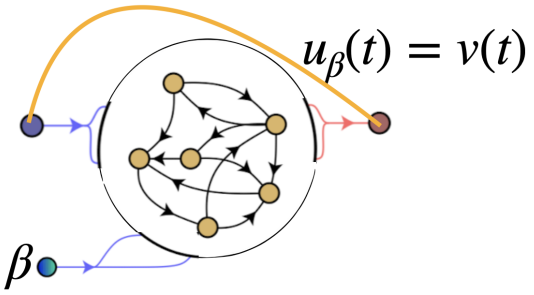

Figure 18: Parameter-aware RC, predicting phase

parameter associated to the studied system. In the predicting phase, we set the parameter-aware RC in close loop, which means that the input correspond to the output of the RC and the second input is still the parameter is used to specify a characteristic of the dynamic that is wanted to be reproduced.

The difference between a parameter-aware reservoir computer and a reservoir computer resides in the state equation, we saw previously that for an ordinary RC it was $x_{n+1} = f(Wx_n + W_{\text{in}}u_{n+1})$, however it slightly differs for a parameter-aware RC because there is also a term related to the parameter, $x_{n+1} = f(Wx_n + W_{\text{in}}v(t) + \beta b)$.

To add more nonlinear transformation and thus more computational capabilities to the RC we create new reservoir states. The difference with these new reservoir states is that even nodes are squared and odd nodes are left as they are.

$$\tilde{R} = \begin{pmatrix} \tilde{r_1}[n] & \tilde{r_1}[n+1] & \cdots & \tilde{r_1}[n+L] \\ \tilde{r_2}[n] & \tilde{r_2}[n+1] & \cdots & \tilde{r_2}[n+L] \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{r_k}[n] & \tilde{r_k}[n+1] & \cdots & \tilde{r_k}[n+L] \end{pmatrix}$$

With $\tilde{r}_i = \begin{cases} r_i & \text{if } i \text{ is odd} \\ r_i^2 & \text{if } i \text{ is even} \end{cases}$

Then the output is calculated as this $v = W_{\text{out}}\tilde{R}$

2.4 Deep Reservoir Computing

Deep Reservoir Computing (DRC) extends the traditional Reservoir Computing (RC) framework by employing multiple layers of reservoirs stacked in a hierarchical manner. Unlike standard RC, which uses a single reservoir to map input data into high-dimensional space, DRC utilizes a series of reservoirs to progressively transform and enrich the temporal features of the data. This multi-layer architecture enhances the model's ability to capture complex temporal patterns and improve prediction performance. By stacking reservoirs, DRC leverages the depth of the network to extract more abstract and hierarchical representations of the input data.

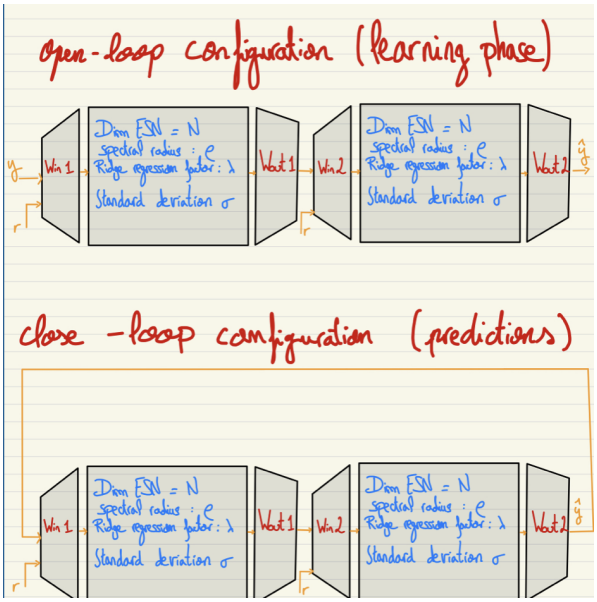

Figure 19: Illustration of a deep reservoir computing framework

2.5 Physical Reservoir Computing

Physical Reservoir Computing is a method that uses the physical properties of a system as a reservoir to perform computation. Thus, Physical Reservoir Computing are used to compute complex problems using less computational resources, for example controlling the movement of a soft robot. The way it works is that many measurements are taken from the physical system. Then, these measurements are gathered into a vector. This vector is then used to train a readout layer (cf. Training the output layer), which is a linear regression model that maps the vector to the desired output (cf. figure 20).

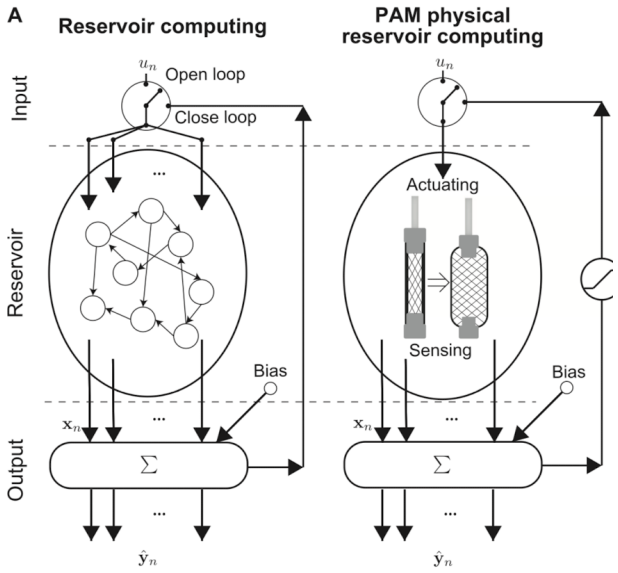

Figure 20: Difference between a reservoir computing and a physical reservoir computing

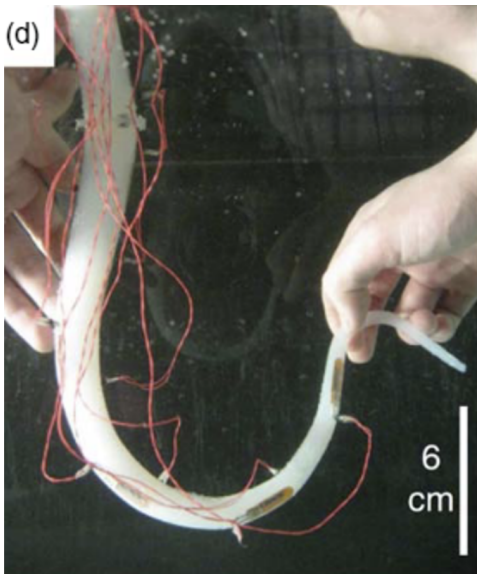

Figure 21: Example of PRC, soft-robot octopus arm, with measurements taken on it

After training, the reservoir can be used in closed-loop, with its output being fed back as input to the system, allowing the system to be controlled.

In Physical Reservoir Computing, the physical system itself acts as the reservoir. The physical system is used to perform computation by exploiting its natural dynamics. The physical system is driven by an input signal, and its response is measured. These measurements are then used as input to a readout layer, which is a linear regression model that maps the input signal to the measured response. The readout layer is trained using the input-output pairs of the system. Once the readout layer is trained, it can be used to predict the response of the system given new input signals. This allows the physical system to be used as a computational resource for solving complex problems. The reservoir, which in this case is the physical system itself, perform computations thanks to its non linear behaviour due to its material properties.

Since it uses a random neural network that does not involve adjustment of internal connections, in principle, any nonlinear system with large degrees of freedom is a candidate for the physical implementation of the reservoir. For example, complicated dynamics obtained by sensing the movement of an octopus' arms on the water's surface can solve various nonlinear tasks by exploiting the intrinsic computational capability.

3 Visit to Bridgestone Laboratories in Tokyo

During my internship in Tokyo, I had the opportunity to visit Bridgestone laboratories located in the city of Kodaira in Tokyo prefecture. These laboratories host numerous experiments, including the development of Pneumatic Artificial Muscles (PAMs), particularly for soft robotics applications. One such application is a hand made of PAMs, as shown in Figure 22.

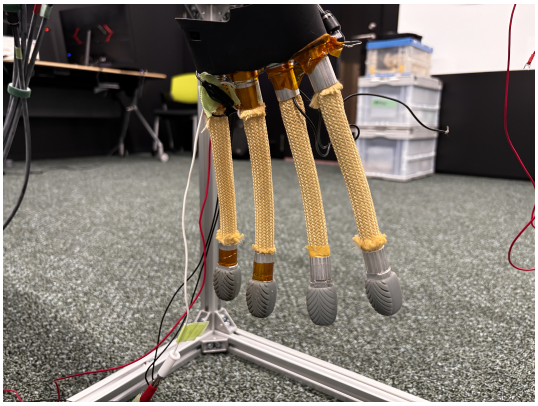

Figure 22: Four pneumatic artificial muscles assembled to form a hand

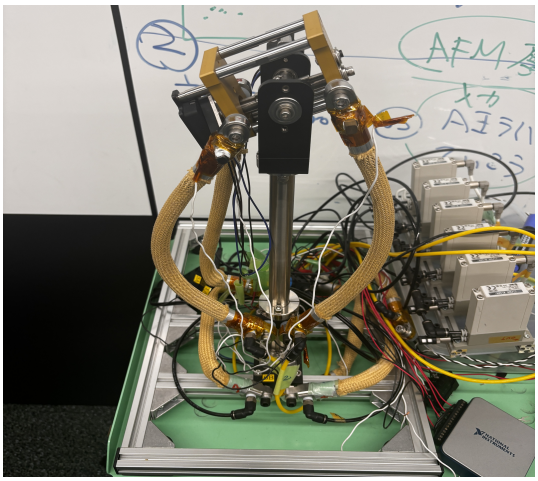

Figure 23: PAMs arranged to enable complex movements

As illustrated in Figure 24, these pneumatic artificial muscles are constructed using a rubber tube covered with braided cord. Carbon black is applied to this structure to enhance the electrical conductivity of the PAM.

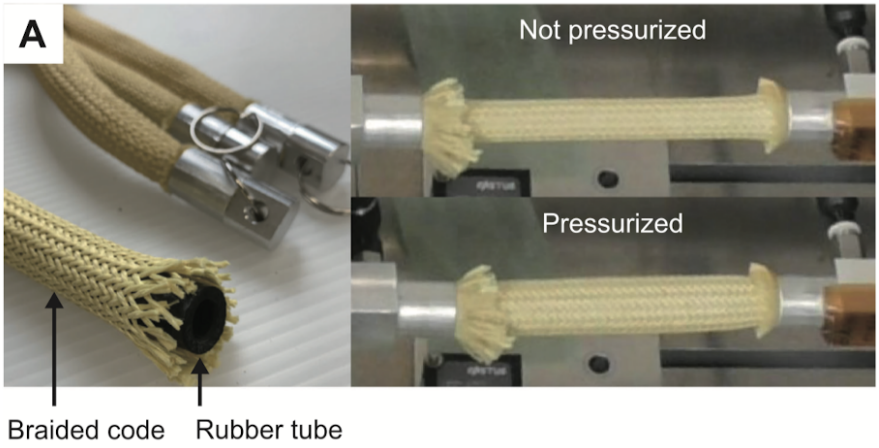

Figure 24: PAM construction and behavior during experimentation

The experiment conducted there involves cyclically injecting and releasing pressurized air into and out of the PAM while performing various measurements. One such measurement is the electrical resistance induced by the PAM materials. This resistance varies as the PAM's length and thickness change: becoming shorter and thicker as pressurized air enters, then longer and thinner as the air is released. As previously mentioned, these measurements, sampled over L time steps, generate the reservoir output states. These states are subsequently used to train a readout layer using ridge regression, enabling the computation of a target output.

4 Reproducing a dynamic with Reservoir Computing

In the paper "Teaching recurrent neural networks to infer global temporal structure from local examples" co-authored by Jason Z. Kim et al., researchers have been able to interpolate and extrapolate a chaotic yet predictable dynamics (if the initial conditions are known), the Lorenz attractor (cf. ref. [2], fig. 25).

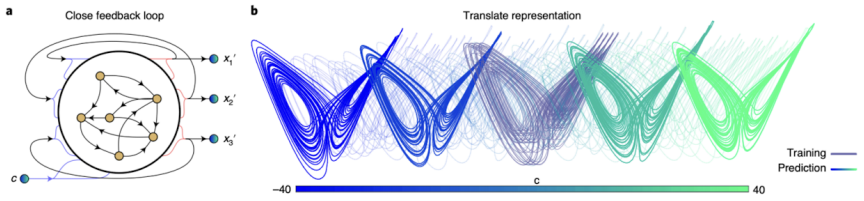

Figure 25: Scheme of the ESN framework and the Lorenz prediction obtained

In their case, the control parameter is used to perform a linear translation of the Lorenz attractor in the state space. And the ESN predicts 3 output vectors as the Lorenz attractor has 3 dimensions.

In this work, we use a one-dimensional model, the logistic map, and we use an Echo State Network (ESN) to try to interpolate and extrapolate the dynamics. The ESN is trained to predict the output of the logistic map. The ESN is trained using the ridge regression method. The ESN is trained for different values of the hyperparameters lambda, rho, matrix sparsity, dimension of the ESN, and sigma. The trained ESN is evaluated and the results are plotted. The results show that the ESN is able to predict the output of the logistic map with high accuracy. The bifurcation diagram of the logistic map is also generated and compared with the ESN prediction. The results show that the ESN is able to capture the dynamics of the logistic map. The results demonstrate the potential of using physical reservoir computing for performing computation.

4.1 Chaotic dynamics, bifurcations, period-doubling bifurcations

Chaotic dynamics describe systems where small changes in initial conditions can lead to vastly different outcomes, showcasing a sensitive dependence on initial conditions. According to Dr. Jason Kim from the University of Pennsylvania, dynamical systems define changes in state over time. By examining these changes, we can understand how systems evolve and respond to varying conditions.

A key concept in the study of dynamical systems is bifurcation, which occurs when a small change in a system parameter leads to a qualitative change in its behavior. One common route to chaos is through period-doubling bifurcations. This type of bifurcation involves the system's behavior doubling in periodicity—shifting from one stable cycle to two, then four, and so on, eventually leading to chaotic dynamics.

In this study, we aimed to interpolate and extrapolate the dynamics of such chaotic systems. The logistic map is a prime example of a simple mathematical model that can display complex behavior, including period-doubling bifurcations leading to chaos. Figure 21 illustrates the logistic map plotted on a bifurcation diagram, demonstrating how changes in the control parameter can lead to transitions from stable states to chaotic behavior.

5 Working Environment

The experiments were carried out on an Apple laptop equipped with an Apple silicon M1 processor and 8 GB RAM. The computer runs the MacOS Sonoma operating system. The code was written in Python, using the following libraries: math, numpy, matplotlib, and scipy.

The time needed to train the readout layer depends mainly on the number of nodes inside the reservoir. For a reservoir of 100 nodes, it takes on average around 2 ms (cf. Fig. 27).

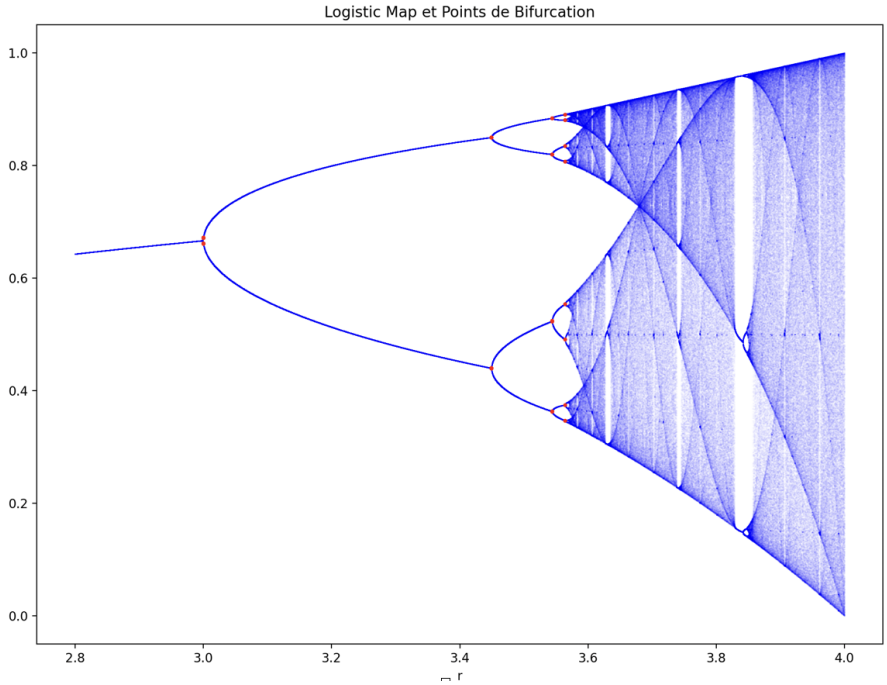

Figure 26: Logistic map model plotted on the bifurcation diagram

```
Training time:  2.624034881591797  ms
```

Figure 27: Time taken by the MacBook Air M1 to train a readout layer when the reservoir has 100 nodes

5.1 Code Structure

The code follows an object-oriented programming (OOP) approach and is divided into several classes, each representing a different component of the reservoir computing system. The complete code is available on my personal GitHub page at https://github.com/TheRemi120/Tokyo_2024_Internship.

The main classes are:

- **Module**: Represents a generic module in the reservoir computing system.
- **Linear**: Represents a linear transformation module in the reservoir computing system.
- **ESN**: Represents an Echo State Network (ESN) module in the reservoir computing system.
- **LinearReadout**: Represents a linear regression readout module in the reservoir computing system.
- **RidgeReadout**: Represents a ridge regression readout module in the reservoir computing system.

5.2 ESN Class

The main class is the ESN class. It has the following attributes:

- `dim`: The dimension of the reservoir.

- `f`: The activation function of the reservoir, the hyperbolic tangent has been chosen. This function should be non-linear in order to add computational ability.
- `a`: The leakage rate of the reservoir.
- `p`: The sparsity of the reservoir.
- `bound`: The bound of the reservoir weights.
- `x_init`: The initial state of the reservoir.
- `x`: The current state of the reservoir.
- `w_net`: The internal weights of the reservoir. They are initialized using a gaussian law or a uniform law.

The ESN class has the following methods:

- `reset_state()`: Resets the state of the reservoir to initial random states.
- `transform_reservoir_state()`: Transforms the state of the reservoir to the new reservoir states in order, as seen previous to enhance the reservoir capabilities.
- `step()`: Updates the state of the reservoir.

The ESN class is used to create an ESN module in the reservoir computing system. This module processes the input data and produces the output data. The output layer is trained using the ridge regression method, then the ESN is evaluated, and the results are plotted.

6 Training the output layer

In echo state networks, the only layer that is programmed is the output/readout layer. This approach significantly reduces the training time compared to traditional recurrent neural networks where all layers are trained.

6.1 Ridge regression class

The Ridge regression class, called "RidgeReadout" in the code, has the following attributes:

- `lmbd`: The regularization parameter of the ridge regression.

The RidgeReadout class has the following method:

- `train()`: Trains the readout layer using the ridge regression method.

The RidgeReadout class creates a ridge regression readout module in the reservoir computing system. This module maps the output of the reservoir to the desired output. It is trained using the ridge regression method, evaluated, and the results are plotted.

6.2 Comparison with the linear regression

The class that performs a linear regression to train the output layer of the ESN is also implemented in the code but not used. The difference between the linear regression and the ridge regression lies in the cost function. The cost function of the linear regression doesn't have a regularization term η, Indeed the cost function of the linear regression is :

$$W_{out} = \arg\min_{W_{out}}(\|U - W_{out}V\|^2) \tag{4}$$

While the cost function of the ridge regression is :

$$W_{out} = \arg\min_{W_{out}}(\|U - W_{out}V\|^2 + \eta\|W_{out}\|^2) \tag{5}$$

6.3 Purpose of the ridge regression:

This method improves the ordinary least squares (linear regression) by adding a regularization term η
Note: Matrix V contains the new reservoir states for each sampling time, with L being the time series length. New reservoir states have been used to increase computational capabilities of the reservoir as mentioned previously.

6.4 Mathematical derivation of ridge regression:

To obtain the optimal solution for ridge regression, we minimize the following objective function and derive the analytical expression for W_{out}:

$$W_{out} = \arg\min_{W_{out}}(\|U - W_{out}V\|^2 + \eta\|W_{out}\|^2) \tag{6}$$

The following $|U - W_{out}V\|^2 + \eta\|W_{out}\|^2$ is a quadratic function, thus it admits a minimum if the first derivative with respect to W_{out} is zero (and the second derivative is negative).

$$\Rightarrow \frac{\partial}{\partial W_{out}}((U - W_{out}V)^T(U - W_{out}V) + \eta W_{out}^T W_{out}) = 0 \tag{7}$$

$$\Leftrightarrow \frac{\partial}{\partial W_{out}}(U^TU - U^TW_{out}V - V^TW_{out}^TU + V^TW_{out}^TW_{out}V + \eta W_{out}^TW_{out}) = 0 \tag{8}$$

$$\Leftrightarrow -UV^T - UV^T + 2W_{out}VV^T + 2\eta W_{out} = 0 \tag{9}$$

$$\Leftrightarrow -2UV^T + 2W_{out}VV^T + 2\eta W_{out} = 0 \tag{10}$$

$$\Leftrightarrow UV^T = W_{out}(VV^T + \eta I) \tag{11}$$

$$\Leftrightarrow W_{out} = UV^T(VV^T + \eta I)^{-1} \tag{12}$$

6.5 Why is it effective?:

The matrix XX^T is a positive semi-definite matrix (all eigenvalues are non-negative), so $XX^T + \lambda I$ is a positive definite matrix and always has an inverse. Therefore, ridge regression is numerically more stable than ordinary linear regression. The parameter η offers a trade-off between fitting training data and model generalization: When η is **small** (e.g., 10^{-9}):

- The term $\eta|W_{out}|^2$ has minimal impact
- Optimization focuses primarily on minimizing $|U - W_{out}V|^2$

- $\Rightarrow$ Better fit to training data
- Risk of overfitting if it's too small

Meanwhile, a bigger η allows the weights to generalize

7 Results

The aim of the research project was to interpolate and extrapolate the logistic map dynamics and predict its bifurcation points. The logistic map equation is: $x_{n+1} = rx_n(1-x_n)$, where the bifurcation parameter is 'r'. Even slight changes to this parameter can lead to chaos in the dynamics. Figure 26 illustrates the logistic map on a bifurcation diagram. The bifurcation diagram plots the dynamics with the x-axis representing the bifurcation parameter 'r', and the y-axis showing the last 50 computed values of the logistic map equation.

This figure was generated by computing thousands of iterations for 'r' ranging from 2.8 to 4.0 with a step of 0.01, retaining the last 50 values of each series for plotting. The next step was to create the dataset. Following the approach of most articles in the bibliography, we created the training dataset using only four values of r. We conducted two separate trainings of two distinct ESNs to observe the effect of the training dataset on subsequent ESN predictions. The first training dataset contained 4 values of 'r', all in period-doubling stable states, while the second training dataset used 4 values of 'r' at bifurcation points (marked in red in Figure 26) No significant difference were found between the two trainings.

Including a washout period did not significantly affect the results, so for each 'r', the time series contained 2000 data points, including the transient. This transient is crucial for the ESN to learn the dynamics, as noted in the article "Learning the dynamics of coupled oscillators from transients" by Huawei Fan et al.

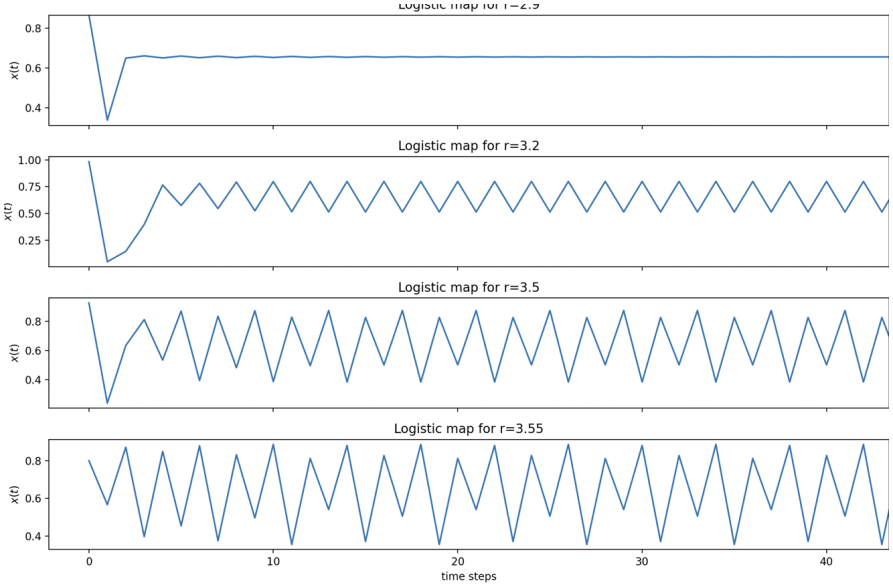

Figure 28: Training dataset time series

Figure 28 shows that the reservoir has been driven with four time series, each corresponding to a different value of the bifurcation parameter r: r = 2.9, r = 3.2, r = 3.5, and r = 3.55.

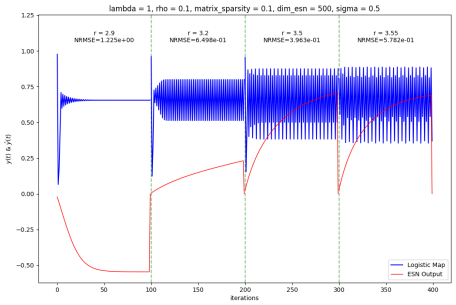

Figure 29: Poor prediction by the ESN (red), compared to the training dataset (blue)

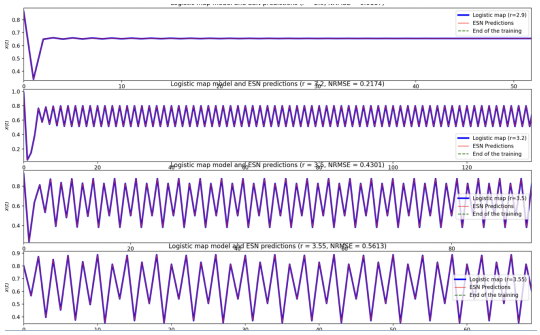

Figure 30: Accurate prediction by the ESN (red), compared to the training dataset (blue)

Figure 29 illustrates the ESN's prediction on the training dataset, allowing us to assess the success of the training. The training's success depends on numerous factors, as previously discussed. These include the chosen hyperparameters for the ESN, the values and number of bifurcation parameter r selected for the training dataset, the length of the training dataset, the probability distribution used to initialize the weights of the input layer and the reservoir, and the cost function used to train the readout layer (in our case, ridge regression). In this dataset, the r parameters are chosen from the stable regimes of the logistic map (between two bifurcation points). As mentioned earlier, another dataset was created to train a second ESN using values of the parameter r associated with bifurcation points on the logistic map. The red curve, representing the ESN's prediction, and the blue curve, showing the target time series of the logistic map for a given value of r, do not match, indicating that the ESN was not successfully trained. However, in Figure 30, we can observe that the ESN's prediction (red curve) and the logistic map model (blue curve) match exactly, demonstrating a successful training of the readout layer.

The hyperparameters define the ESN's behavior, making their selection a crucial part of this work. To optimize these parameters, we iterate through different loops, with each loop associated with a specific hyperparameter, as shown in Figure 31.

```
for param_lambda in lambda_liste:
    for rho in rho_liste:
        for matrix_sparsity in sparsity_liste:
            for dim_esn in dim_esn_liste:
                for sigma in sigma_liste:
```

Figure 31: Iterations over the ESN's hyperparameters

Initially, these lists contain different orders of magnitude for the hyperparameter values. Then, after observing how the ESN behaves when a hyperparameter is around a certain value by analyzing the ESN's predictions on the logistic map, we can fine-tune the hyperparameter values until satisfactory results are achieved.

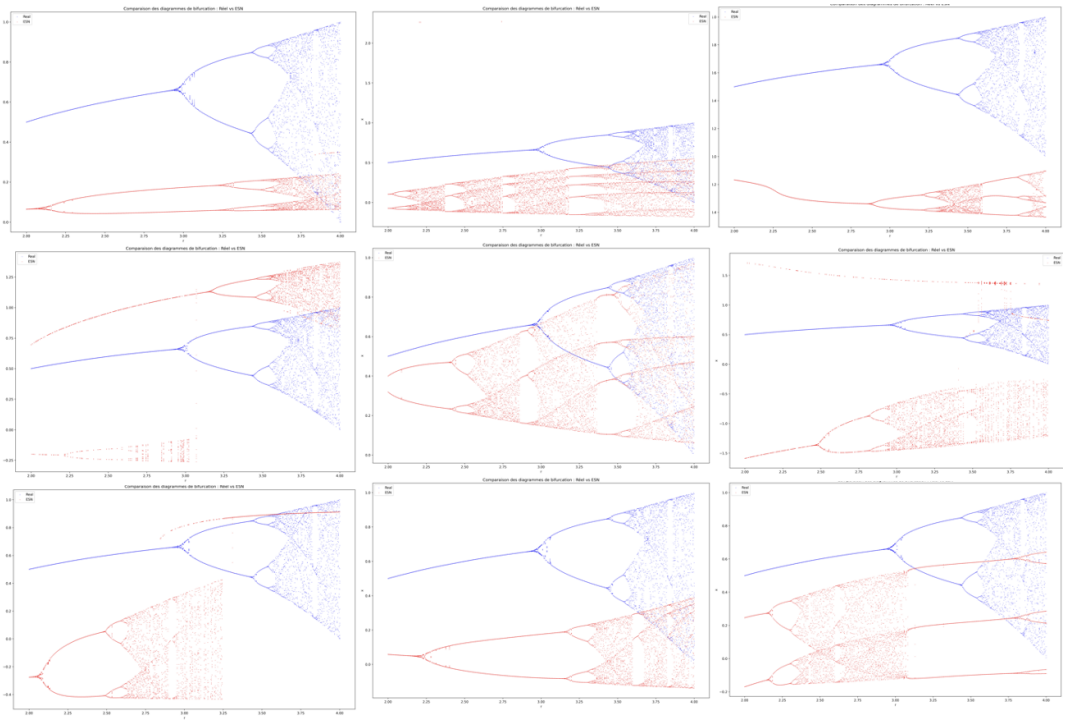

Figure 32: Logistic map model (blue) and ESN prediction (red)

In Figure 32, we can see a few of the thousands of predictions obtained for different values of hyperparameters used to shape the ESN.

For each graph, the following procedure was implemented. First, we set the ESN in open-loop (cf. Fig. 17). We drive the ESN with two inputs. In the papers mentioned in the bibliography, the first input is typically the time series of the target dynamics, in our case the logistic map, for a given value of the bifurcation parameter, 'r'. However, better predictions were obtained using this vector: $input = [u_\beta, r, r * u_\beta, r + r * u_\beta]$ where u_β is the logistic map model. The second input (called β in Fig. 17), known as the control parameter, is the bifurcation parameter itself, which in our case is r. Then during the prediction phase, we set the ESN in close-loop, that means that the first input becomes : $input = [y_{out}, r, r * y_{out}, r + r * y_{out}]$ where y_{out} is the output of the ESN. The second input is still the bifurcation parameter r.

As we can observe in Figure 32, the predictions of the ESN do not exactly match the model of the logistic map. However, the shape and behavior are globally well represented, which is already a satisfactory result. Papers in the bibliography also reported predictions that did not exactly match their target dynamics but were slightly transformed, for example with a linear translation between the model and the prediction.

The ESN was able to reproduce both the stable states and the chaos with bifurcation points and period-doubling bifurcations, while being trained with only 4 values of the r bifurcation parameter, all in the stable states. This ability of the reservoir to reproduce all the dynamics behavior is still not well explained and remains an active field of investigation for researchers.

8 Conclusion & Perspectives

This three-month research internship at the Intelligent Systems and Informatics Laboratory at the University of Tokyo allowed me to discover a new field that I was not previously familiar with: reservoir computing. It was incredibly valuable to have the support of the laboratory, which provided me with the opportunity to attend their reservoir computing course from April to July, as well as participate in and give a talk at their reservoir computing seminar.

For the internship, I had to start from scratch, using Python with object-oriented programming for a better understanding of the code. This approach was necessary as all the codes written by the researchers, whose articles guided this internship, were originally written in MATLAB. I was able to achieve key results such as the interpolation and extrapolation of the dynamics of a chaotic yet predictable (given known initial conditions) system: the logistic map. The experiment could have been extended by testing how the ESN could interpolate and extrapolate the dynamics of a real system, specifically the Pneumatic Artificial Muscle (PAM) from which we obtained data at the Bridgestone laboratories. This would have meant using the PAM as the reservoir, effectively implementing physical reservoir computing. Unfortunately, the data from the PAM were not available during my time in Tokyo. I consider this as a potential area for future improvement.

A significant part of this research-oriented internship was dedicated to reading articles, assimilating their content, reproducing the experiments described, and presenting them to other lab members. While I had experience with this process from my time at École Normale Supérieure Paris-Saclay, it was much more frequent during this internship. I observed a real improvement in the efficiency of my reading, which was initially very time-consuming, as well as in my comprehension of the scientific English vocabulary used in research. An article could be written after more work on this project, especially after embedding the bifurcation of the PAM that I was not able to do during my time in Tokyo because of time constraint.

I enjoyed this internship and learned a great deal from it, both in terms of living 8000km away from my home country alone and in the field of research. I am grateful to my professors in France for preparing me for such an experience, and to the laboratory team in Tokyo, especially Professor Kohei Nakajima, for their guidance and support.

References

[1] My presentation at the RC seminar. Available at : https://www.youtube.com/watch?v=hfJn5xS9bFE

[2] Teaching recurrent neural networks to infer global temporal structure from local examples, Jason Z. Kim, Zhixin Lu, Erfan Nozari, George J. Pappas and Danielle S. Bassett. DOI : https://doi.org/10.1038/s42256-021-00321-2

[3] Embedding Bifurcations into Pneumatic Artificial Muscle, Nozomi Akashi, Yasuo Kuniyoshi, Taketomo Jo, Mitsuhiro Nishida, Ryo Sakurai, Yasumichi Wakao, and Kohei Nakajima. DOI : https://doi.org/10.1002/advs.202304402

[4] Learning the dynamics of coupled oscillators from transients, Huawei Fan, Liang Wang, Yao Du, Yafeng Wang, Jinghua Xiao, and Xingang Wang. DOI : https://doi.org/10.1103/PhysRevResearch.4.013137

[5] Bifurcation Embedding in One-Dimensional Chaotic Dynamics Using Reservoir Computing, Jason Z. Kim, Dani S. Bassett. DOI : https://doi.org/10.1038/s42256-023-00668-8

[6] Emergence of transient chaos and intermittency in machine learning, Ling-Wei Kong, Huawei Fan, Celso Grebogi and Ying-Cheng Lai. DOI : https://doi.org/10.1088/2632-072X/ac0b00

[7] Machine learning prediction of critical transition and system collapse, Ling-Wei Kong, Hua-Wei Fan, Celso Grebogi, and Ying-Cheng Lai. DOI : https://doi.org/10.1103/PhysRevResearch.3.013090

[8] Physics-Informed Recurrent Neural Networks for Soft Pneumatic Actuators, Wentao Sun , Nozomi Akashi, Yasuo Kuniyoshi and Kohei Nakajima. DOI : https://doi.org/10.1089/soro.2020.0024

[9] Reservoir computing tutorial from ISI Lab. - The University of Tokyo

[10] Logistic map, Wikipedia. https://en.wikipedia.org/wiki/Logistic_map

List of Figures