Danmarks Tekniske Universitet

DTU

BACHELOR PROJECT

Using AI to study bifurcations in dynamical systems

Authors: Đurđija Milinković (s204724)
Felix Oscar Ærtebjerg (s204797)

Supervisor: Poul Georg Hjorth

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

May 8, 2023

PREFACE

The following thesis was prepared at the Department of Applied Mathematics and Computer Science (DTU Compute) at the Technical University of Denmark as a fulfillment of requirements for acquiring a BSc in General Engineering. The thesis has been done and written in the period from February to May 2023. The project has a weight of 20 ECTS-points.

Đurđija Milinković (s204724)

Signature: Đurđija Milinković

Date, Location: May 8, 2023, DTU Lyngby

Felix Oscar Ærtebjerg (s204797)

Signature: Ærtebjerg

Date, Location: May 8, 2023, DTU Lyngby

ACKNOWLEDGEMENTS

This thesis was made in cooperation with Poul G. Hjorth, associate professor at DTU Compute. We would, therefore, like to thank Poul G. Hjorth for making this project possible, for introducing us to the topics, and for guiding and supporting us throughout the whole project.

We would also like to thank Ling-Wei Kong, Hua-Wei Fan, Celso Grebogi and Ying-Cheng Lai for their outstanding articles "Machine learning prediction of critical transition and system collapse" [1] and "Emergence of transient chaos and intermittency in machine learning" [2], both of which represent the fundamental resources on which our work was built.

Finally, we want to thank Steven H. Strogatz for his book "Nonlinear Dynamics and Chaos" [3] which conveyed the fundamentals of Dynamical Systems in an entertaining and clear manner.

Contents

1 Introduction

Bifurcations, often called 'critical transitions' and found in the field of Dynamical Systems, are an important concept in many systems in science, including applications in physics, biology, chemistry, engineering, and medicine. Dynamical systems change under the influx of one or many parameters and usually rather unexpectedly so, grabbing the attention of scientists seeking to understand them for many decades now. Someone who is not familiar with dynamical systems may wonder why it is so important to grasp these concepts and what their purpose might be. To them, we present one of the most critical dynamical systems of all: the Earth's climate.

Climate can be seen as a system that always stood out due to its continuous changes over time. Hence, it is normal to expect more changes in the future. Due to a lack of currently available information, it is impossible to determine when the next change will happen, how fast the transition will last, and how drastic the shift and the system's stability will be since this system has many degrees of complexity. For example, the Greenland Ice Sheet is close to a melting point when the system, our global climate, and its behavior will irreversibly change bringing unwanted outcomes. Such is the rise in the global sea level of 7 meters [4]. Crossing this critical point can happen very quickly after which nothing could be done to stop it, highlighting the importance of real-life bifurcations.

The above-stated facts also emphasize the importance of understanding the bifurcations in multiple systems and the possibility of predicting the system's behavior and when tipping points are reached to potentially prevent collapses. That represents the main purpose of this project. However, predicting bifurcations with high accuracy has proven to be a challenge.

It largely depends on the complexity and non-linearity of the studied systems. To ease some of these problems we propose the usage of deep learning, or artificial intelligence (AI), as many have before us [1, 2, 5, 6, 7]. We are witnessing rapid growth in the field of machine learning due to major improvements in technology and they seem to be really suitable for this kind of usage cases. Therefore, we are going to take advantage of one form of AI: the dynamical recurrent neural network (RNN) and its subset reservoir computing.

We were highly dependent and based our work on the code implementation provided by the authors of the article "Machine learning prediction of critical transition and system collapse" [1] and "Emergence of transient chaos and intermittency in machine learning" [2]. The code was made for the three-dimensional, complex chaotic systems, thus an accent had to be put on picking the right systems for testing. It required a lot of research and testing of the performance until the final systems had been chosen.

The work on the project included reviewing and studying literature about dynamical systems and bifurcations, prediction of critical transition and system collapse, recurrent neural networks (RNN), and reservoir computing. The milestones of the project are understanding why and when the bifurcation occurs, system and model analysis testing, as well as having the final model and its successful implementation. Finally, evaluating the results and examining the obtained plots.

2 Dynamical Systems and Bifurcations

A wide range of systems are changing over time and capturing the interest of scientists seeking to understand them. In the field of mathematics, those systems are defined as dynamical systems. Like introduced before, an example we can give is a system everyone is aware of: climate, which has changed in the past, it is currently changing and we expect it will also change in the future. Thus, this shows the dynamical behavior of the climate. Although climate change is influenced by numerous factors and presents various degrees of complexity with little information yet known, it is good to emphasize that not all dynamical systems are as intricate. In fact, one of the most elementary examples of dynamical systems is the swinging pendulum. Its dynamical behavior can be represented by mathematical equations and as the pendulum swings back and forth we can monitor its motion over time.

2.1 Introduction to Bifurcations

The motion of the swinging pendulum is always stable and periodic. But again, not all systems are like that and one question arises: what does happen when we have a system whose behavior suddenly changes in a dramatic way? A bifurcation emerges. As per Strogatz's explanation, bifurcations refer to the qualitative changes in a system's dynamics [3]. Bifurcations impose a significant role in helping us to understand and predict the system's behavior. We can closely monitor the system's behavior over a period of time and even manipulate it by adjusting the so-called control parameter values until a transition or instability is triggered at a specific moment. Before we start observing and detecting bifurcations in more complex systems, it is crucial to get a solid and thorough understanding of the simplest systems and their bifurcations in one dimension.

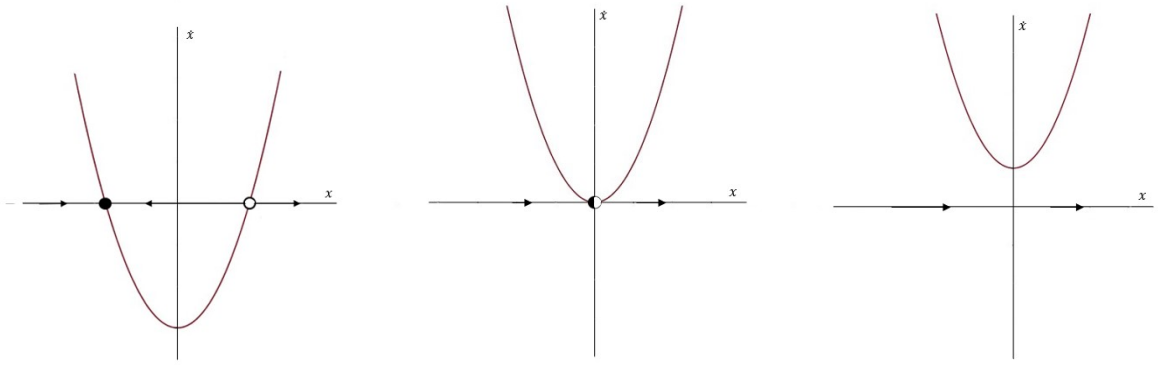
2.2 Saddle-Node Bifurcation

Since the language used to describe different terms in the field of bifurcations has not settled down yet and come to a mutual agreement, different people use different terms to describe the same occurrence. As a result, saddle-node bifurcation is often called a fold bifurcation, a turning-point bifurcation, or even a blue sky bifurcation. Nevertheless, we have chosen to maintain consistency with our wordings and identify it exclusively as the saddle-node bifurcation.

The saddle-node bifurcation is one of the easiest examples to show how two fixed points are firstly getting closer, then colliding, and finally annihilating each other. The above-written process happens when a certain parameter is varied. It is worth mentioning that the presence and behavior of fixed points in dynamical systems are tightly linked to their equilibrium properties. As fixed points do not vary with time, their movement and disappearance will signal changes in the observed system's equilibrium state. We observe that in the well-known example, given by the first-order system, for the saddle-node bifurcation [3]:

$$\dot{x} = a + x^2 \tag{1}$$

Here a represents a bifurcation parameter that can be negative, zero, or positive, while x is the state variable, representing the main quantity of interest that changes over a period of time. Detecting fixed points for different parameter values can tell us a lot about the system's behavior.



(a) $a < 0$ **(b)** $a = 0$ **(c)** $a > 0$

Figure 1: Fixed points, vector fields and bifurcations for a saddle-node bifurcation for different parameter values inserted to the Equation 1. In (a) there is a stable fixed point to the left and an unstable fixed point to the right of zero, (b) shows the semi-stable fixed point at 0, while in (c) the system does not have any fixed points anymore.

It can be seen in Figure [1a], when $a < 0$, we have two fixed points: one stable and the other unstable. As we vary the bifurcation parameter, those two fixed points are moving towards each other, until finally merging into a half-stable point when $\dot{x} = 0$, see Figure [1b]. This fixed point is also known by the name semi-stable and occurs when one side of the phase space goes towards, while the other side moves away from the fixed point [3]. Finally, when the varying parameter is greater than zero, i.e. $a > 0$, [1c], the half-stable fixed point disappears, and the system does not have an equilibrium point anymore.

In this case, the system underwent the bifurcation at $x = 0$ which we can confirm by the bifurcation diagram:

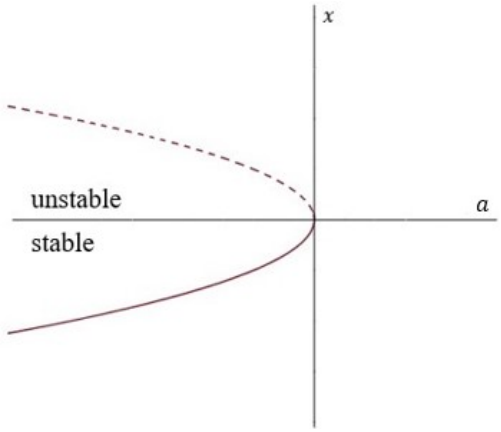


Figure 2: The bifurcation diagram for the saddle-node bifurcation occurring when both x and a are equal to zero. We see unstable and stable fixed points for negative a values approach each other as a gets closer to 0. At 0 they collide and annihilate each other and no stable point exists beyond $a = 0$.

In bifurcation diagrams, it is common to plot the parameter a on the horizontal $x-$axis. This allows us to visually monitor how changes in a affect the equilibrium points and behavior of the system. Hence, when a shift in a parameter leads to a change in the observed system's behavior, we can analyze the bifurcation diagram to understand the transition. In this case, we notice that bifurcation occurs at the point when both x and a are equal to zero, confirming our earlier assertion.

2.3 Pitchfork Bifurcation

Moving on, we arrive at the following type of bifurcation, known as the pitchfork bifurcation. It is specific in a way that its fixed points are moving in the same fashion, i.e. appearing and disappearing together. Hence, this type of bifurcation is usually found in symmetric systems. In 1D, the system transforms from one to three fixed points. The original fixed point usually undergoes changes in stability, while the other two are born and displayed in the symmetric matter [8]. The name "pitchfork" perfectly fits and is derived from the shape this bifurcation obtains, which will be shown below. There are two different types of pitchfork bifurcation: supercritical and subcritical. We now present only one example for the supercritical to delve into it to get a deeper understanding and insight, as well as to illustrate the behavior of the system when the bifurcation parameter is varied. For more examples or further explanation for the subcritical pitchfork bifurcation, please have a look at Strogatz's book [3].

The standard form of an equation representing a supercritical pitchfork bifurcation [3]:

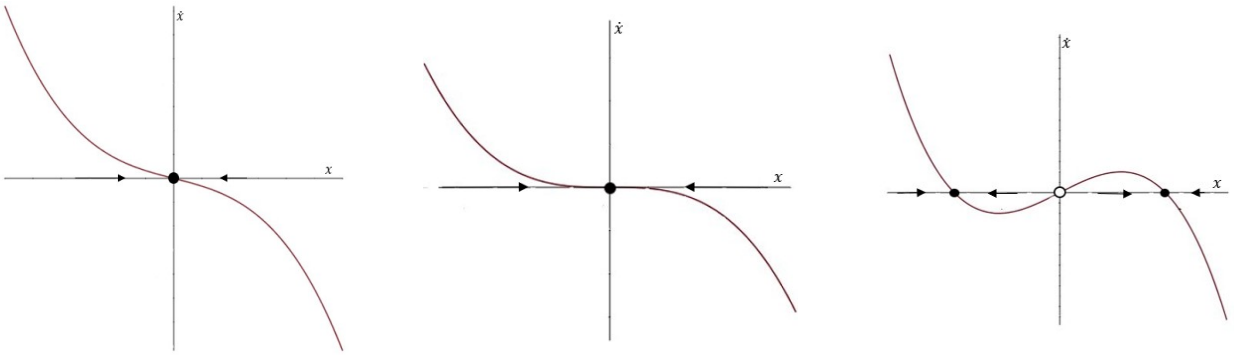
$$\dot{x} = ax - x^3 \tag{2}$$

It is good mentioning that this equation is invariant [3] when the variables are changed. It means that if we change the sign of the x variable, from positive to negative ($x \to -x$),

the solution would be the same, as we would still have the original Equation 2. Thus, the system's invariance property proves its close relationship with symmetry.

$$-\dot{x} = -ax + x^3 \quad \Longleftrightarrow \quad \dot{x} = ax - x^3$$

We preserve the same notation as for the saddle-node bifurcation, and hence, a still represents a bifurcation parameter, while x is the state variable. The system's behavior can be observed when varying the bifurcation parameter as is done below:



(a) $a < 0$ **(b)** $a = 0$ **(c)** $a > 0$

Figure 3: Fixed points, vector fields, and bifurcations for a supercritical pitchfork bifurcation for different parameter values inserted to the Equation 2. In (a) there is one stable fixed point at zero, which remains unchanged in (b), while in (c) the stable fixed point changes its state and becomes unstable, and two new stable fixed points are born.

It can be seen in Figure [3a], when $a < 0$, we have one stable fixed point at the origin. Varying the bifurcation parameter, we notice that nothing has changed and that at $a = 0$, [3b], we still have the same stable fixed point, but weaker [3]. Lastly, when the bifurcation parameter gets greater than zero, i.e. $a > 0$, as seen in Figure [3c], the stable fixed point at the origin changes its stability and becomes unstable, while the two new stable points are born. Those two new stable fixed points appear symmetrically on both sides from the origin.

Considering this particular case, we observe that the system underwent the bifurcation at $x = 0$ which is confirmed by the following bifurcation diagram:

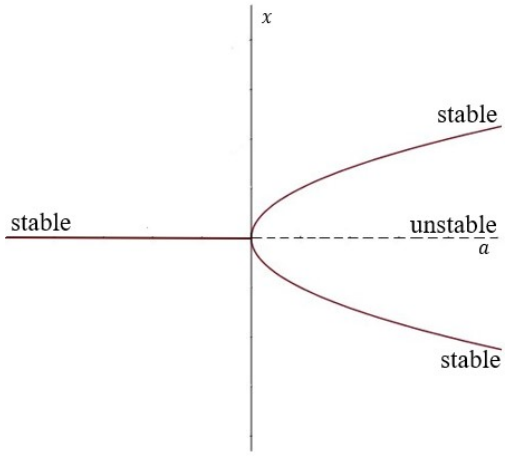


Figure 4: The bifurcation diagram for the supercritical pitchfork bifurcation. One stable point at 0 approaches the origin and splits into two stable points branching out in opposite ways, as well as one unstable point continuing on $x = 0$.

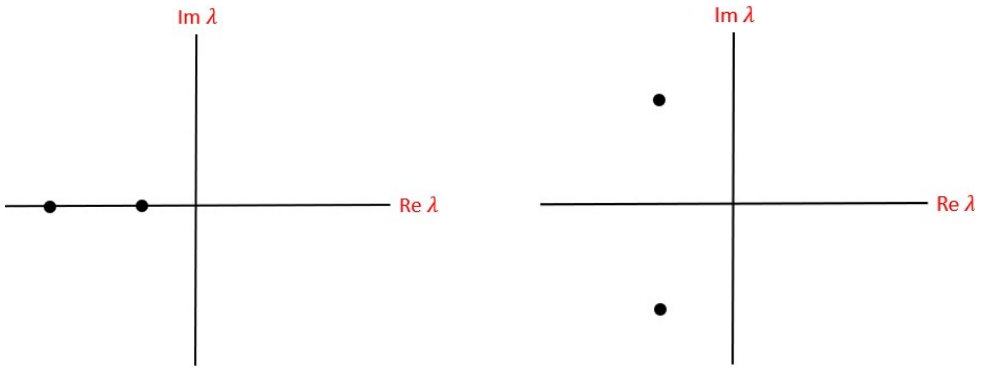
As mentioned above and after the graphical representation, it is completely clear why this bifurcation has the name that it has, since the curves on the graph look like they are forming a literal pitchfork.

2.4 Hopf Bifurcation

Now, we turn to a more complex type of bifurcations: bifurcations in two dimensions. Above discussed bifurcations in one dimension refer to the change in stability as the parameter is varied. The good news is that they remain the same when more dimensions are added. All the activity and changes still happen in one dimension. The main difference between bifurcation in one dimension and two dimensions is the presence of closed orbits (periodic behavior where the point repeats itself) and oscillations which make the system more complicated [3].

Even though this type of bifurcation exists in all dimensions greater or equal to 2, we explain it and see how it looks in an example of the bifurcation named *Hopf bifurcation* in two dimensions. Similar to pitchfork bifurcations, Hopf bifurcations also come in two types (supercritical and subcritical) of which subcritical is usually more dangerous and complex, and we will not cover it in this report.

Assume a two-dimensional stable system is given, i.e., it has a stable fixed point. We want to see what and in which way influences the behavior of the system so it changes its state, i.e. destabilizes. Thus, as the bifurcation parameter a is varied, we need to take a closer look at the eigenvalues of the Jacobian.



(a) Real eigenvalues **(b)** Complex eigenvalues

Figure 5: Example Eigenvalues of a stable system. (a) shows two negative real eigenvalues and (b) two complex eigenvalues with negative real parts.

If the system has a stable fixed point then the real part of both eigenvalues has to be less than zero. Since eigenvalues are calculated from the quadratic equation with real coefficients, we have two possible scenarios as seen in Figure [5]: either both λ_1 and λ_2 are real numbers less than zero [5a] or both λ_1 and λ_1 are complex conjugates whose real parts are again, less than zero [5b]. Therefore, in order to make the system unstable, we will vary the parameter a until at least one of the eigenvalues has crossed to the right half-plane.

At this moment it is nothing new when a real eigenvalue goes through zero, $\lambda = 0$, since we will have saddle-node, transcritical, or pitchfork bifurcations. So, now, we turn to more interesting scenarios.

Let us now consider the system in which both complex eigenvalues pass over the imaginary axis, Im λ, to the right half-plane at the same time. Suppose we have a system that has damped oscillations that are reducing over time until they fully vanish when the system reaches an equilibrium state. Let the decay rate be the rate at which those oscillations decrease over time until finally reaching the equilibrium. If the decay rate changes when the control parameter a shifts, then we can expect a change in the stability of the system when the decay rate becomes slower and slower until it starts oscillating around some other value. This system's change in stability means nothing but that it has undergone a bifurcation known as a supercritical Hopf bifurcation.

(a) The stable system before the bifurcation represented by dumped oscillations that are reducing over time until reaching the equilibrium state.

(b) The system after the supercritical Hopf bifurcation which is now represented by an unstable spiral. Motion is a limit cycle oscillation.

Figure 6: The system before and after the supercritical Hopf bifurcation

One of the main purposes of studying bifurcations is to be able to predict the outcome and point in time when it will occur, which becomes a challenging task to accomplish, especially in systems undergoing subcritical bifurcation. It can be dangerous for many real-life systems where stability is of significant importance. A small shift or parameter drift can influence very drastic and unpredictable behavior of the system.

A concrete example would again be the longstanding concern of climate change. It is a complex system in which interaction between its variables could cause the climate system to undergo a subcritical bifurcation and transition to a new stable or unstable, different climate regime [9]. It could potentially have irreversible consequences with catastrophic outcomes for human societies and future life, natural ecosystems, water sources, infrastructure, and many others.

3 Neural Networks

3.1 Problem and Solution

With that introduction to the mathematical background of our problem, let us move on to how we actually plan on detecting said bifurcations.

In nature, many of the interesting systems are chaotic, meaning they are very hard to predict, if not impossible. Nevertheless, we still want to know if these systems are stable or have already passed their critical point and are now facing imminent collapse. The fundamental issue here is that we will almost certainly always face two problems: the chaotic nature prevents us from predicting the system's future with conventional tools and even with momentary data we do not always know whether a system is still stable or if it is heading towards a catastrophe.

This is where machine learning becomes interesting. Machine learning, or rather deep learning, has the ability to perceive and infer data structures in the data beyond what we can comprehend. Deep learning is a subject within supervised machine learning that makes use of stacked shallow learning algorithms and it is supervised in the sense that we have data on the desired outcome beforehand. The most commonly used deep learning tool is artificial neural networks, sometimes referred to as 'AI'. In recent years, neural networks have seen explosive growth in popularity thanks to user-friendly implementations and improving processing power that makes it possible to circumvent the above-discussed issues to some degree.

The main idea of this paper is to use AI or neural networks to learn from time series data, predict dynamical system evolution, and detect bifurcations. To do this we will use reservoir computing, a form of the dynamical recurrent neural network, but let us first take a step back and introduce neural networks generally.

3.2 History of Neural Networks

Artificial neural networks, originally conceptually proposed by the neuroscientist McCulloch and the mathematician Pitts in 1943 [10], were inspired by how our brain does

computations. They stipulated that the nature of neurons in our brains is "all-or-none", meaning that a neuron will either be fully active or not at all. This led the two to develop networks that emulated neurons with simple logic functions of binary nature to do information processing. Later works showed that neurons are not in fact binary, but rather take a range of values and are strengthened by usage [11], the foundation of the way we learn.

Already in 1975 the scientist Fukushima developed a stepwise trained multilayered neural network. It was based on previous works that used machines with computations in accordance with biological principles and learning capability [12]. Fukushima used the neural network to interpret handwritten characters [12].

After another 40 years or so of exploding interest and rapid developments, the field has been expanded immensely and the applications of neural networks reach into many facets of innovation. Most of these improvements would not be feasible without the vastly improved computers, see Figure [7] [13], but at the same time, neural networks are making computers even more powerful as well. A good example is the microprocessor True North by IBM, which supposedly can simulate the work of 256 million synapses [12] only using a chip the size of a postage stamp.

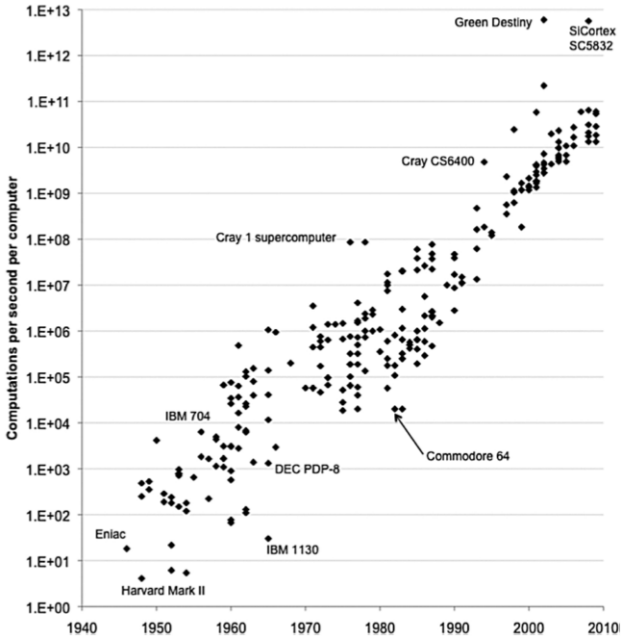


Figure 7: The increase of computational power from 1940 to 2010 that made our technological revolution possible, including the advancements in neural networks. Some of these advancements were due to improved silicon chip architecture, others were made in optimization and by deepening our understanding of computations. The recent upward push came with the invention of quantum computers.

Today, neural networks can be split into many different factions, but the essential three groups can be categorized as: artificial neural networks (ANN) also called feed-forward neural networks, convolution neural networks (CNN) and recurrent neural networks (RNN). The use cases of the three sections can be generalized as ANN's using tabular data, CNN's image

and audio data, and RNN's time series data. There exist many more specialized offshoots of the three of course, and the choice of neural network heavily depends on the problem and data types anyhow. The neural network that we will be using, a reservoir computer, is an adaptation of the RNN and because both that and ANNs share the same basic structure, we will introduce the elemental mathematical model of a feed-forward neural network below.

3.3 Mathematical Introduction to Neural Networks

Most of this section will be based on Chapter 15 from the course book "Introduction to Machine Learning and Data Mining" by Tue Herlau, Mikkel N. Schmidt, and Morten Moerup.

The central idea of a neural network is to emulate how the brain uses its neurons and synapses to do computing, which is by way of the brain receiving some information, exciting some neurons with the input that in turn may excite even more neurons through their synapse connections.

Like the brain, neural networks are also made up of units called 'artificial neurons' or sometimes 'perceptrons' that have weighted connections to other neurons or perceptrons. In the case of an ANN or feed-forward neural network, the information is handed sequentially from one neuron layer to the next, executing a mathematical operation on each step.

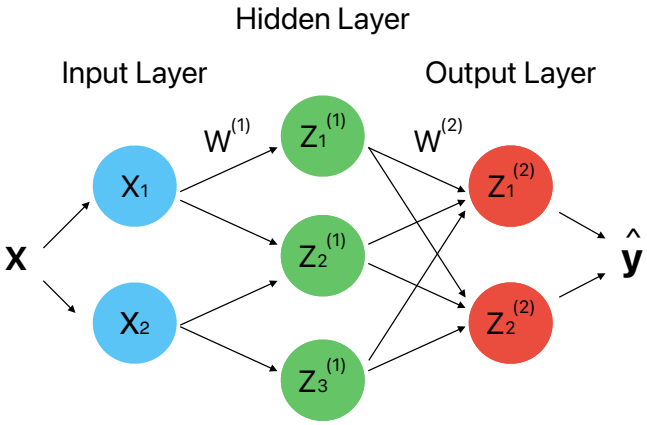


Figure 8: The configuration of a simple ANN, given two input and two output streams. It consists of two neurons in the input layer, a single hidden layer with three neurons, and an output layer with another two neurons. Data is input into the initial layer and transformed into values only intelligible by a computer, after which it can go through numerous hidden layers and neurons before it reaches the output layer. There it is transformed back into a measure that we can grasp. The description feed-forward of the ANN stems from the exclusively sequential data processing.

In Figure [8], we depicted the structure of a simple feed-forward neural network that takes two inputs and gives two outputs. This could be anything from tabular data or a time series going in and classifications or another time series found through regression coming out.

While we, for clarity's sake, chose only one hidden layer in the depiction, we must remember that the feed-forward network is not constricted to a specific number of hidden layers. We could add arbitrarily many sequential hidden layers, although the upper limit depends on the processing power of the computer that has to train and use the network.

Essentially, the way that a neural network operates is by applying a stacked linear regression and logistic regression to every input data and outputting a value that determines the significance of the neuron response. In the output layer, it then evaluates the machine's response using an error function that constitutes the learning part of machine learning.

Considering our example neural network, we shall now go through the first part of the feed-forward neural network algorithm: To begin with, each input layer neuron i is initialized to have the value, or using the proper neural network term, the activity of x_i. The input layer will now excite the first hidden layer and each neuron j is activated using the following term [14]:

$$\mathbf{a}_j^{(1)} = \tilde{\mathbf{x}}^T \mathbf{w}_j^{(1)}, \tag{3}$$

where $\tilde{\mathbf{x}}^T$ is network input vector and $\mathbf{a}_j^{(1)}$ is the numeric activity given to neuron j in layer 1 determined by the weights. The $\mathbf{w}_j^{(1)}$ term denotes the weight given to the neurons in the hidden layer 1 through training. These weights are the fundamental piece that gives neural networks the ability to learn and predict. For instance, if a neuron's response values one characteristic of our data too much, the weight for that characteristic will be lowered in the subsequent training round. The neural network will thus have 'learned' to not consider that part of our data as important.

After the activation of the first hidden layer neurons, they are transformed using a nonlinear activation function *h()* [14]:

$$\mathbf{z}_j^{(1)} = \mathrm{h}(\mathbf{a}_j^{(1)}), \tag{4}$$

where $\mathbf{z}_j^{(1)}$ is the output of neuron j in layer 1 produced by the activation function. The reasoning behind the activation functions is literally to determine how much the neuron should be *activated*, thus the amount of value that should be given to the neuron in the larger computation. Typically, the activation function will be the hyperbolic tangent that maps any input between -1 and 1 or the logistic sigmoid that maps to [0,1]. Say, if the output of the tangential function is 0, then the neuron will naturally not have any significant influence on the next step in the neural network.

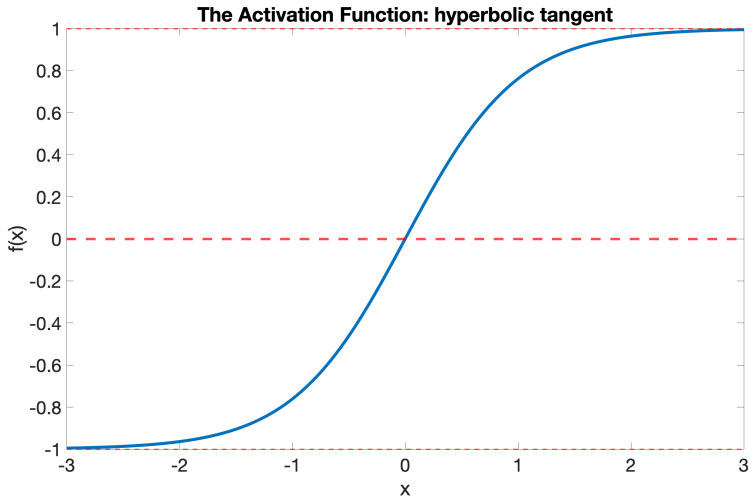


Figure 9: Simple plot of the hyperbolic tangent and non-linear activation function $\mathrm{h}(x) = \tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$. We can see that the function takes a real input and maps in between -1 and 1. This is used to control the activation of neurons, 1 and -1 being significant activation while 0 indicates that the neurons' information input is not important.

Going by Figure [9], each neuron k in the output layer is given an activation $\tilde{\mathbf{z}}^{(1)}$ based on the output from the previous layer and transformed with another activation function. Combined, the two steps look like this [14]:

$$\mathbf{z}_k^{(2)} = \mathrm{h}^{(2)}\big((\tilde{\mathbf{z}}^{(1)})^T \mathbf{w}_k^{(2)}\big) \tag{5}$$

$\mathbf{z}_k^{(2)}$ stands for the output from neuron k in layer 2, $\tilde{\mathbf{z}}^{(1)}$ refers to the output from the previous layer and $\mathbf{w}_k^{(2)}$ to the weights for layer 2. For our network, the final output would be [14]:

$$\mathbf{f}(\mathbf{x}, [\mathbf{w}^{(1)}, \mathbf{w}^{(2)}]) = [\mathbf{z}_1^{(2)} \mathbf{z}_2^{(2)}] \tag{6}$$

It is simply the output of the two neurons in the output layer. The process above can be generalized to feed-forward neural networks with any number of hidden layers and layer sizes. All that changes is that there will be more weights to determine through training, but the accuracy might improve with it.

3.4 Training process of neural networks

For any size neural network the above algorithm gives us a parametric function $\mathbf{f}(\mathbf{x}, \mathbf{w})$ that maps $\mathbf{x}$ to $\mathbf{y}$ given a weights matrix $\mathbf{w} = [\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, ...]$ [14]. At the beginning of the training process the weights matrix is set an initial guess. Now, let us say we run the algorithm once and are left with the parametric function $\mathbf{f}(\mathbf{x}, \mathbf{w})$. But we see that the training error, or the error between real and predicted values, of the neural network is quite large and does not allow for immediate use. Our goal is now to change the weight values in $\mathbf{w}$ so that the mapping is the one we desire. In our problem, for example, we have a set of

observations $\mathbf{x}$ consisting of the time series before the bifurcation and a target $\mathbf{y}$ of the time series after the bifurcation.

This training problem is common in the field of supervised machine learning that has an equally common solution approach. As mentioned, the neural network predictions will likely not be $\mathbf{y}$ exactly, but we will assume that $\mathbf{y}$ is normally distributed around the neural network's predictions [14].

We have a set of observations $\mathbf{x}$ and targets $\mathbf{y}$, which in our problem is the time series of the bifurcation system before a bifurcation. As mentioned, the neural network predictions will likely not be $\mathbf{y}$ exactly, but we will assume that $\mathbf{y}$ is normally distributed around the neural network's predictions [14].

We then apply some probability theory that involves maximum likelihood [14] to find that in order to maximize the likelihood of the weights $\mathbf{w}$ being as close to the true values as possible, we need to minimize the cost function E [14]:

$$\mathbf{w}^* = \arg\min E_\lambda(\mathbf{w}) \tag{7}$$

$$E_\lambda(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{f}(\mathbf{x},\mathbf{w}) - \mathbf{y}\| + \lambda\mathbf{w}^T\mathbf{w} \tag{8}$$

We can read from Equation [7] that we find the adjusted or trained weights $\mathbf{w}^*$ from $\mathbf{w}$ by selecting or finding the term with the lowest error from the cost function E. And the cost function [8] is essentially just the total error between all predicted targets and the known targets while adding some regularization using λ. In case regularization is unfamiliar to the reader, explained in one sentence we would say that increasing λ will lead to smaller weights and predictions with lower variance and higher bias while decreasing it does the opposite [14].

So, with the cost function in mind, we should theoretically be able to solve for the optimal weights. However, it turns out that it is actually analytically impossible due to the nonlinear nature of the activation functions. Instead, scientists have found iterative algorithms like Gradient Descent [15] and Backpropagation that find the minimum of the cost function by taking steps toward the valley of the function. More sophisticated versions like AdaBoost exist that have optimized the process. As our math model does not make use of any of these algorithms, we will not go into more detail. We now move on to the practical part of our paper, reservoir computing and its applications.

4 Reservoir computing and Math Model

4.1 Reservoir computing background

Now that we have a good understanding of the most basic neural network, let us return to the issue of using a neural network to detect bifurcations in dynamical systems. The feed-forward neural network discussed could theoretically be used to analyze temporal data but by adding recurrent connections inside of the hidden layers, we transform the system into

a dynamical one [16]. Recurrence in this case refers to connection cycles where the output of one node can affect the subsequent input into itself. The new system is able to represent the dynamical nature of time-sensitive systems much better. Networks like these are called 'Recurrent Neural Networks' (RNN) and are powerful tools for investigating sequential data [17]. The drawback of these new networks, however, is that they are computationally very expensive to train and converge slowly [16], preventing those without access to a lot of resources from working with them.

In an attempt to circumvent this dilemma, Jaeger [18] built the Echo State Network (ESN) framework. An ESN is a recurrent neural network with randomly set internal weights that is able to mirror the dynamical nature of the RNN without the need for training. Instead, only the output layer is trained to ensure that the output data follows the solution data. For a sketch of how the network looks refer to Figure [10]. The echo-state network is now considered to be part of the reservoir computing field.

So, after covering the basic feed-forward neural network and transitioning through the RNN we have reached reservoir computing. The remaining pages of this paper will only be concerned with the latter and the experiments that we conducted.

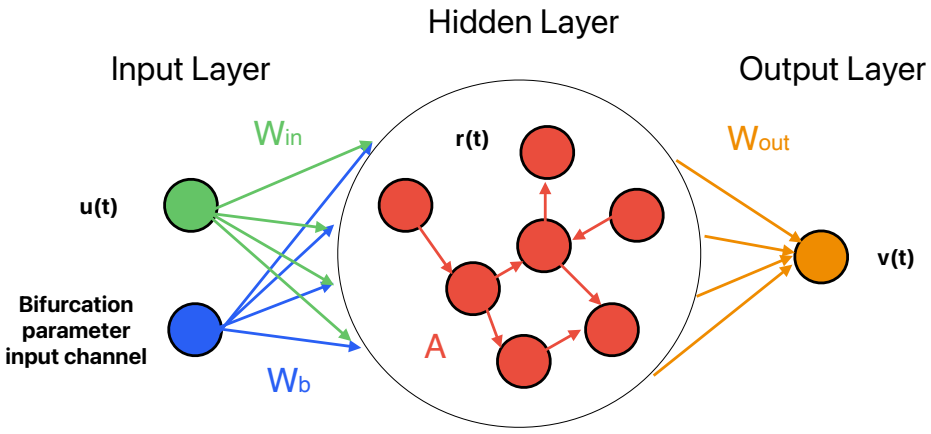


Figure 10: The configuration of the reservoir computer that we are using. We have two input channels, a time-series channel $\mathbf{u}(t)$ and one for a bifurcation parameter. Both channels get fed into the hidden layer that uses random weights and recurrent neural connections. This means that the data might get fed through the same hidden layer multiple times. The processed data is then mapped back into a time series using the output layer and given as the vector $\mathbf{v}(t)$. In the case of predictions, the input bifurcation parameter is the parameter we wish to predict the time series of and the output layer is the one we trained with known time series data.

4.2 Math Model of the Reservoir Computer

Generally, the reservoir computer shares many similarities with the previously discussed ANN. Both consist of neurons, an input, hidden and output layer. Figure [10] displays the computational features of the machine, the most important parts being the random

connections inside the hidden layer and the additional input channel. The following section will give more insight into how it works.

4.2.1 Terminology and Classification

Let us begin by explaining the terminology and all terms involved in reservoir computing and specifically our model. Using Figure [10] as a guide, here is a list of the relevant terms:

1. **Input and output**:
   - $\mathbf{u}(t)$ - the input time-series
   - $\mathbf{r}(t)$ - the hidden layer result
   - $\mathbf{v}(t)$ - the output time-series
2. **Bifurcation parameters**: They belong to the system we want to predict and they can change the stability of the system. To train the reservoir neural machine to be able to make accurate predictions of system chaos and collapse, we insert different values of the bifurcation parameter into the input channel.
   - $\mathbf{b}$ - the bifurcation parameter
3. **Fixed parameters of Target Systems:** These are all predetermined and fixed. They belong to the systems we try to predict, in those cases all parameters but the bifurcation one.
4. **Elements of the input matrices and reservoir adjacency matrix [1]**: Chosen randomly before training and fixed during the prediction process.
   - $\mathcal{W}_{in}$ - a weight matrix which is used to map the vector $\mathbf{u}(t)$ to the state vector $\mathbf{r}(t)$ of the hidden layer.
   - $\mathcal{W}_b$ - a weight matrix that helps to connect an additional input parameter channel to every node of the reservoir network.
   - $\mathcal{A}$ - the adjacency matrix of the reservoir network that transfers information from the hidden state $\mathbf{r}(t)$ to $\mathbf{r}(t + \Delta)$.
5. **Hyperparameters of the reservoir machine**: The purpose of this set of parameters is to control the training process. They are determined through optimization, preassigned before training and they define the machine's characteristics.
   - n - number of nodes in the hidden layer
   - α - the leakage parameter
   - β - the regularization parameter
   - k_b and b_0 - scaling factors for $\mathcal{W}_{in}$ and $\mathcal{W}_b$

- d - average degree of $\mathcal{A}$
- ρ - spectral radius of $\mathcal{A}$
- b_0 - bifurcation parameter bias

6. **Training parameters**: they belong to the reservoir machine, values are not fixed but determined through the process of training and depend on the training data sets.

   - $\mathcal{W}_{out}$ - a weight matrix that is used to map the state vector $r(t)$ to the output vector $v(t)$

4.2.2 Training

The training procedure follows the same regiment as the ANN, the only significant differences are the pre-set weights and the additional bifurcation parameter input channel. As we already discussed the ideas behind the training of a neural network in Chapter 4, we will keep the formulas brief and focus on the new aspects.

To train the reservoir machine we give it a real time-series input and let it calculate the next step. The dynamical evolution of each time step in the reservoir computing machine [1] is then explained in the following equation:

$$\boldsymbol{r}(t+\Delta t) = (1-\alpha)\boldsymbol{r}(t) + \alpha \ \tanh(\mathcal{A}\cdot\boldsymbol{r}(t) + \mathcal{W}_{in}\cdot\boldsymbol{u}(t) + k_b\mathcal{W}_b(b+b_0)) \tag{9}$$

The equation basically describes a few things at once. The hidden layer is initialized with $\mathcal{W}_{in}\cdot\boldsymbol{u}(t) + k_b\mathcal{W}_b(b+b_0)$, which are the time-series and bifurcation parameter channel inputs. To that the hidden layer computation is added with $\mathcal{A}\cdot\boldsymbol{r}(t)$. All three are fed into the activation function $tanh()$ and follow the previous time-step $\boldsymbol{r}(t)$. The output is then determined [1]:

$$\boldsymbol{v}(t) = \mathcal{W}_{out}\cdot\boldsymbol{r}(t) \tag{10}$$

Like for an ANN we now seek to compare the machine prediction to the real one. To do that we calculate the difference between the real time-series and prediction using the following loss function with l_2 regularization [1]:

$$\mathcal{L} = \sum_t ||\boldsymbol{u}_{all}(t) - \mathcal{W}_{out}\cdot\boldsymbol{r}'_{all}(t)||^2 + \beta||\mathcal{W}_{out}||^2, \tag{11}$$

where $\boldsymbol{u}_{all}(t)$ is the real time-series for all trained bifurcation parameters. $\mathcal{W}_{out}\cdot\boldsymbol{r}'_{all}$ is the machine's time-series prediction for all parameters, the hidden layer output $\boldsymbol{r}'_{all}$ is adapted for chaotic systems [1]. Notice how the reservoir actually minimizes the error between all trained bifurcation parameters all at once. It is thus learning information from the dynamical system as a whole, which is an important feature of the machine if it is to predict bifurcations. The regularization used here is to help prevent the reservoir machine from overfitting the data, therefore acting as a complexity control [14]. Like in the ANN, the loss function is used to minimize the information loss or error between the reservoir prediction and the input data.

But there are two significant differences: the reservoir only trains the output layer weight matrix $\mathcal{W}_{out}$ and it uses ridge regression instead of Gradient Descent to solve for the minimum of the loss function. Both differences stem from the original idea of reservoir computing: the randomly set input and hidden layer weights to simulate dynamical behaviour without the need to train every layer like in RNNs. The only thing we minimize for is $\mathcal{W}_{out}$ to reduce the difference between the predicted output time-series and the real one. And because we only consider one layer for training (unlike the ANN that trains for every layer) we are able to use ridge regression and do not have to deal with Gradient Descent. Ridge regression basically describes a linear regression with regularization and standardized data [14]. It is commonly used in cases like ours when there is high collinearity between multiple linear regressions applied at once [19].

A critical aspect of the training for our model is that the reservoir machine is only trained with time series under **pre-bifurcation** circumstances with different parameter values. The additional parameter input channel (as seen in Figure [10]) helps the machine learn and start "acknowledging" the changes in the time-series data when fed with different parameter values. It learns to 'detect' an impending bifurcation. In that way, the machine is being trained to learn patterns in the original system to predict the behavior of the future system using the input and dynamical terms from the past.

4.2.3 Prediction

After the machine is done training, the following change is made to the time-step Formula [9] above:

$$\boldsymbol{r}(t+\Delta t) = (1-\alpha)\boldsymbol{r}(t) + \alpha \ \tanh(\mathcal{A}\cdot\boldsymbol{r}(t) + \mathcal{W}_{in}\cdot\boldsymbol{v}(t) + k_b\mathcal{W}_b(b+b_0)) \tag{12}$$

Notice that instead of $\boldsymbol{u}(t)$ as the input we now use the previous prediction solution $\boldsymbol{v}(t)$. The next output time step becomes:

$$\boldsymbol{v}(t+\Delta t) = \mathcal{W}_{out}\cdot\boldsymbol{r}(t+\Delta t) \tag{13}$$

The machine now uses its own previous step as the starting point for the next predicted time-series step. It is a closed loop and self-evolving from an initial condition.

5 Implementation Code and Testing Setup

To conduct our experiments we used the code provided with the papers that formed the foundation of our thesis, namely the work of Kong et al. in their 2021 studies [1, 2]. We will now briefly discuss the main aspects of the code and how it is used:

Training data: As we do not have any real system data to analyze, we will instead use predetermined and known dynamical systems. The time series of the differential equation will be numerically estimated using the Runge-Kutta Order 4 Method for multiple bifurcation parameter values below the critical point. These will then form the training data needed.

Functions: The code consists of two primary functions, one that trains the reservoir computer using our training data and one that predicts the evolution of a time series given a specified bifurcation parameter and initial warm-up. The parameter itself can be below or above the critical point.

Hyperparameter Optimization: Before we can start to train and use the reservoir, we must first optimize the hyperparameters. This is a crucial step that allows the system to learn and predict the most accurately. The code [2] uses a Bayesian Optimization method in Matlab that works by iteratively training a reservoir, finding its training error, and using probability to guess a better set of hyperparameters. Very simple pseudocode for the hyperparameter optimization [20] could look like this:

Algorithm 1 Bayesian Optimization

function OPTIMIZE(hyperparameter space Θ, Target score function $H(\boldsymbol{\theta})$, max evaluations n_{max})
    Select initial hyperparameter configuration $\boldsymbol{\theta}_0$
    Evaluate loss function to find initial score $y_0 = H(\boldsymbol{\theta})$
    Set $\boldsymbol{\theta}* = \boldsymbol{\theta}_0$ *and* $y* = H(\boldsymbol{\theta})$
    for $n = 1, ..., n_{max}$ **do**
        Select a new hyperparameter configuration $\boldsymbol{\theta}_n \epsilon \Theta$ *by optimizing an acquisition function*
        Evaluate $y_n = H(\boldsymbol{\theta}_n)$ *to obtain a numeric score*
        if $y_n <$ threshold **then**
            $\boldsymbol{\theta}* = \boldsymbol{\theta}_n$ and $y* = y_n$
        end if
    end for
    return $\boldsymbol{\theta}*$ and $y*$
end function

Training: The experiments are conducted by initially setting the optimized hyperparameters and the non-bifurcation parameters based on the system that will be analyzed. The reservoir then trains on a few bifurcation parameter values below the critical value until the training error converges below an error threshold. The procedure for this can be summarized with the following:

Algorithm 2 Training procedure

```
procedure RESERVOIR MACHINE TRAINING
    P ← machine parameters found through training
    r_min ← minimum training error/ loss function score
    Set machine hyperparameters
    Set machine time-steps and time-frame
    Select bifurcation parameters to train on
    for Number of Training Attempts do
        for Number of Bifurcation Parameters do
            Train reservoir machine with current bifurcation parameter, call training function
            P_temp and r_temp ← Training results
            if r_temp < r_min then
                P ← P_temp
                r_min ← r_temp
            end if
        end for
    end for
    return P and r_min
end procedure
```

Prediction: The prediction itself is executed separately. Here, the system is first 'warmed-up' using known data with a bifurcation parameter value below the threshold. After a specified warm-up period, the bifurcation value is changed to the test value and the prediction function [1] is called. It will then output the time series that the reservoir predicts for a given time interval.

6 Results

With all our tools in hand, we now proceed to the experiments we conducted. Our goal was to demonstrate that neural networks and specifically reservoir computing can not only be used to detect chaos attractors [1, 2] but also be used to detect bifurcations in simpler dynamical systems. To test this thesis we used two exemplary dynamical systems found in Strogatz's book [3] and tried to predict their known bifurcations using our adapted code. The former will be the classic Supercritical Pitchfork Bifurcation and the latter a famous chemical oscillation reaction that involves a Hopf bifurcation.

Like mentioned in the setup above, we did not work with real data but rather simulated training data by calculating the solutions to the dynamical systems numerically. We restricted the training data to solutions with bifurcation parameter values *below the critical point*. The reservoirs thus still have to learn and predict the bifurcation point using the bifurcation input channel.

6.1 1D Example: Supercritical Pitchfork Bifurcation

As discussed in Section [2.3], the standard and most simple form of a dynamical system with a supercritical pitchfork bifurcation has the form:

$$\dot{x} = ax - x^3 \tag{14}$$

with a known bifurcation at $\mathbf{a_c} = \mathbf{0}$. The stable point for $r < 0$ is 0 and splits for $r > 0$ into two up and down diverging branches, see Figure [4].

We train the reservoir computer for r values of $r = -0.15$, -0.1 and -0.05, all below the critical point and set a constant initial condition for all training instances. After training, we warm up the machine with a known bifurcation parameter below the critical point and the same initial condition. Then we use the trained reservoir computer to predict the evolution of the time series for different a values, see an example in Figure [11]. To check the prediction accuracy we trained 200 random reservoir computers and tested when they would predict bifurcations over a range of a-values, see Figure [12]. We found that the empirical critical point was $a_c = 0 \pm 0.05$. The uncertainty is potentially due to the sub-optimal decision criteria or training values for a.

The hyperparameter values we used to obtain these results were $n = 400$, $d = 152$, $\rho =$, $k_{in} = 1.31$, $k_b = 1.47$, $b_0 = 0.13$, $\alpha = 0.5$ and $\beta = 10^{-10}$. The time-step of the reservoir is $\Delta t = 1$, the training length is $t_{train} = 25$ and the validation length is $t_{validation} = 10$. The root-mean-square validation error is found by averaging over 100 random reservoir machine training instances, comparing the real and predicted time series, and is calculated as $E_{validation} = 6.6 \cdot 10^{-6}$.

The machine generally demonstrates the capability of detecting that there is a bifurcation taking place. That the predicted time series is different after bifurcation than the real system is something we expected [1]. It is because the reservoir machine was not trained on a-values after the bifurcation and therefore has no information to work with. That the predicted time-series chose a different bifurcation branch is something we cannot explain, but could again be due to the choice of training values.

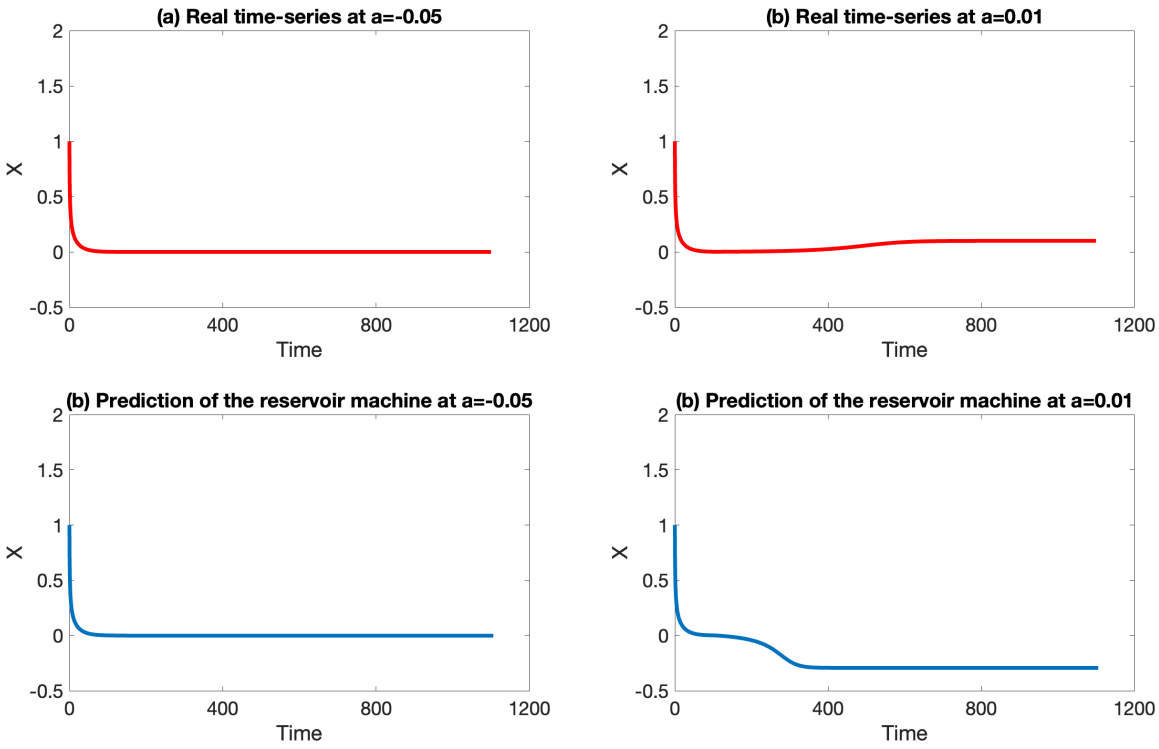


Figure 11: Four graphs, divided into a group with bifurcation parameter values below the critical point and a group with values above. **(a)** The real and machine predicted time-series for $a = -0.05 < a_c$. After a short transient both settle into the stable point 0, exactly as the analytical solution in Section [2.3] said it would. **(b)** The real and machine predicted time-series for $a = 0.01 > a_c$. After they both have a warm-up period of 100 steps, we see that both the real and predicted time series undergo some bifurcation and reach a stable point different from 0. In the case of the real system, the stable point is slightly above 0 while the stable point for the prediction is more noticeably below 0. This is a curious result, as it seems that the real system and prediction each took on a different branch of the bifurcation discussed in Section [2.3]. We do not know a reason why the reservoir machine chose the bottom branch over the top one. Nevertheless, the machine was able to detect that a bifurcation in the system took place. That the time series after bifurcation was not accurate is an issue Kong et al. [1] ran into as well and can partly be explained that the machine was not trained on any bifurcation parameter values *after* the bifurcation.

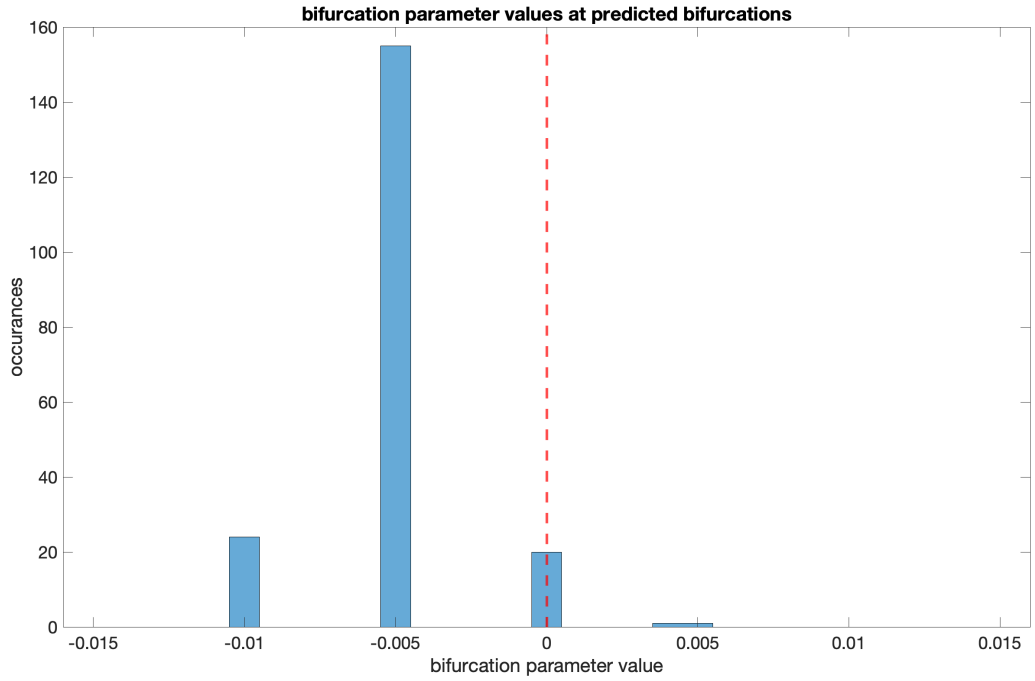


Figure 12: This histogram depicts the result of training 200 random instances of the reservoir machine and determining for which parameters of a a bifurcation occurred. Our decision threshold for whether or not a bifurcation occurred was if the time-series deviated by more than 1% of 0. We see that the vast majority of the bifurcation seemed to have occurred at -0.005 even though the critical point is 0. Reasons for that could be internal error margins of the machine, sub-optimal choices for the bifurcation parameters used for training or bad decision criteria. The machines prediction is nevertheless very close to the true value.

6.2 2D Example: Oscillating Chemical Reaction

Our second example to test the reservoir machines capability is the oscillating chlorine dioxide-iodine-malonic acid reaction found by Lengyel et al. in 1900 [21]. These types of reactions are based on the work of Russian biochemist Boris Belousov who discovered the first oscillating chemical reaction in 1951 as he was trying to recreate the Krebs cycle in a test tube [22].

Lengyel et al. analyzed the chlorine dioxide-iodine-malonic acid reaction and found that some of the reactants changed very slowly and could be considered constants, producing a two-dimensional system [3, 21]:

$$\dot{x} = a - x - \frac{4xy}{1+x^2} \tag{15}$$

$$\dot{y} = bx(1 - \frac{y}{1+x^2}) \tag{16}$$

x and y are the dimensionless concentrations of iodine and chlorine dioxide respectively. The parameters $a, b > 0$ represent the empirical rate constant and concentration of the before mentioned slow reactants [3]. In the following, we will assume a to be constant at $a = 10$ and b to be the bifurcation parameter.

This system undergoes a supercritical Hopf bifurcation at the critical point $\boldsymbol{b_c = 3.5}$ [3] at which the system stops oscillating (going around limit cycles [3]) and converges onto one stable point. For the system, this means that the iodine and chlorine dioxide concentrations both approach a stable end concentration. Like before, we train the reservoir machine with optimized hyperparameters at multiple bifurcation parameters below the critical point, namely $b = 3.46$, 3.47, 3.48 and 3.49. The initial condition for all training warm-up instances will be randomly chosen in the close vicinity of $x = 1$ and $y = 0$. Like in the 1D example we warm up the machine with a known bifurcation parameter value below the critical point. An example result of a trained reservoir machine for predictions for $b < b_c$ and $b > b_c$ can be seen in Figure [13]. To test the bifurcation detection accuracy we train 200 random reservoir machines and plot their predicted critical point b_c in a histogram [14]. The histogram indicates that the machine is able to predict the bifurcation occurring with an accuracy of 3.5 ± 0.002.

The hyperparameter values used for this example system were $n = 400$, $d = 236$, $\rho = 1.79$, $k_{in} = 0.61$, $k_b = 0.76$, $b_0 = 0.92$, $\alpha = 0.09$ and $\beta = 5 \cdot 10^{-4}$. The time-step of the reservoir is $\Delta t = 0.015$, the training length is $t_{train} = 1000$ and the validation length is $t_{validation} = 8$. Like in example (1) we train 100 random instances of the reservoir machine and calculate their mean validation error, the error type again being the root-mean-square error. We find that $E_{validation} = 2 \cdot 10^{-4}$.

With this, the reservoir computer again successfully demonstrates the ability to detect bifurcations before they occur. It bears repeating that the badly predicted time series for $b > b_c$ is due to the system only having been trained on b-values below b_c. This example system also displays a use-case scenario. If an industrial process relies on oscillating chemical reactions staying in limit cycles then it is crucial to know when they could collapse and what kind of change could cause it.

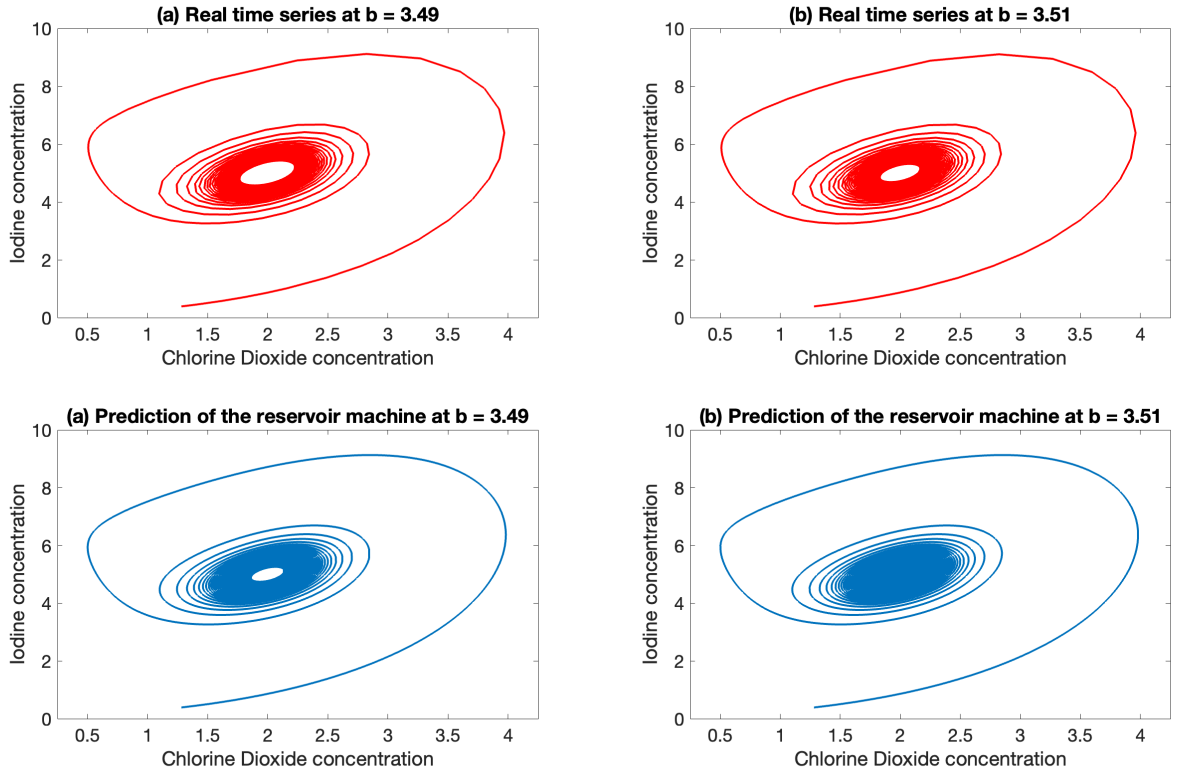


Figure 13: Four graphs, again divided into a group with bifurcation parameter values below the critical point and a group with values above. **(a)** The real and machine predicted time-series for $b = 3.49 < b_c$. We see that both approach a limit cycle, fast at the beginning and slower with each step. We notice that the accuracy seems to go down the closer the prediction gets to the limit cycle. The limit cycle then is significantly smaller for the prediction compared to the real time series. This is because the prediction accuracy of the reservoir machine decreases slightly with each step and at the end the cumulative error is noticeable. **(b)** The real and machine predicted time-series for $b = 3.51 > b_c$. Unlike before, here only the real time series approaches a limit cycle. The predicted time series actually goes toward a stable point. This demonstrates that even if the actual systems has not yet changed dramatically after the critical value, the reservoir machine will still be able to detect the bifurcation itself. But the prediction is inaccurate because the machine was not trained for any values beyond the bifurcation.

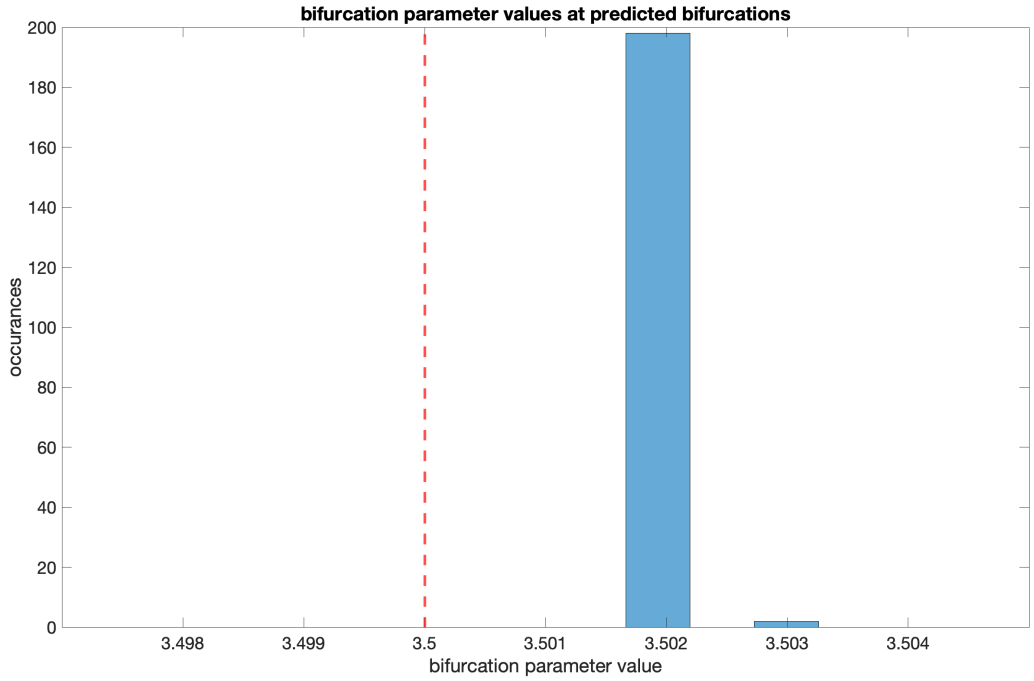


Figure 14: A histogram showing the predicted critical points of the reservoir machine. The red dotted line is the true critical point $b_c = 3.5$. The results were produced by training 200 random reservoir machines and using criteria to determine the predicted critical point. The criteria that determined if a prediction had undergone a bifurcation was if the variance of the final 100 time steps was smaller than 0.001. This would indicate that the time series was converging onto one point. We see in the graph that the overwhelming majority of predicted critical points were at $b = 3.502$, differing from the actual value only by 0.2%. The offset is likely due to our rudimentary decision criteria.

7 Discussion

The main finding of our project, as shown in the previous section, was that we were indeed able to predict the bifurcation point of two different dynamical systems with adequate accuracy. Even the bad time-series predictions after the critical points can be explained by, as mentioned before, the lack of training for parameters after bifurcation. The only significant issue that we did not expect is the skewed statistical predictions of the critical point. It is because they are heavily influenced by the choice of decision criteria on when a bifurcation occurred. We believe we were not able to accurately determine decision criteria that separated a system before and after bifurcation, but with more data analysis experience we might have been able to.

Another point to make is regarding our choice of example systems. We chose two super-critical systems because of their converging solutions close to or at 0. Subcritical systems or even the simple saddle-node differential Equation [1] showed to go towards infinity extremely

quickly in their solutions. We were not able to surmise if the reservoir machine was capable of detecting bifurcations in systems that have these explosive growths towards infinity. Because we did not have a clear understanding of how to separate the numerical solutions before and after bifurcations in every case, we limited ourselves to these two example systems that allowed us to do so to some degree.

Moreover, we ran into some issues with the computational power required to run the reservoir machine. When keeping the training length of the example systems somewhat short, we could execute the training and prediction code on our laptops in a few minutes at most. But if we added more training data like more bifurcation parameters or lengthening the training length, we quickly stopped seeing progress in the execution. It was especially bad for the 1D example (1) because the time series converged onto 0 quite fast. If we were to do a more in-depth analysis of the reservoir results and their accuracy, then we would likely need more computing resources.

Finally, we briefly want to discuss the choice of analysis. Indeed, the reservoir computer works. But it was originally built to analyze chaotic dynamical systems and we think that the use of it or generally AI might not be necessary for many one-dimensional systems. Instead, simpler machine learning tools like the Principal Component Analysis could have potentially been applied. Two-dimensional or higher dimensional systems, however, do deserve the usage of AI as demonstrated by oscillating chemical reaction example.

8 Conclusion

This work presents the theoretical foundation for dynamical systems, bifurcations, neural networks, and reservoir computing. A comprehensive description of our goal, mathematical model, and a simplified code implementation and testing setup has also been provided. To teach the machine how to make accurate predictions, hyperparameters were optimized using Bayesian Optimization.

Picking the appropriate systems for testing posed a considerable difficulty since the code used was originally built for three-dimensional, complex chaotic systems. Regardless, we wanted and still could confirm the theory with the plots obtained for two different, simpler, supercritical systems: one-dimensional and the other two-dimensional. The produced graphs showed wanted results, successful bifurcation detection and prediction using AI on two different testing systems.

Despite the challenges we faced throughout the project, it brings the opportunity to several aspects that could be considered for further work on the project. One of them would include going into more detail about reservoir computing and getting a deeper understanding of why it is so good for dynamical systems. Moreover, more system testing could be done, especially regarding the more complex and chaotic systems.

References

[1] Ling-Wei Kong et al. "*Machine learning prediction of critical transition and system collapse*". In: Phys. Rev. Res. 3 (1 2021), p. 013090. DOI: https://doi.org/10.1103/PhysRevResearch.3.013090.

[2] Ling-Wei Kong et al. "*Emergence of transient chaos and intermittency in machine learning*". In: Journal of Physics: Complexity 2.3 (2021), p. 035014. DOI: https://doi.org/10.1088/2632-072X/ac0b00.

[3] Steven H. Strogatz. *Nonlinear Dynamics and Chaos. with Applications to Physics, Biology, Chemistry, and Engineering*. CRC Press, Taylor & Francis Group, 2018. ISBN: 978-0-8133-4910-7.

[4] Reinhard Calov Volker Klemann Meike Bagge Dennis Honing Matteo Willeit and Andrey Ganopolski. "*Multistability and Transient Response of the Greenland Ice Sheet to Anthropogenic CO2 Emissions*". In: 3 (1 2023). DOI: https://doi.org/10.1029/2022GL101827.

[5] M Basso, R Genesio, and A Tesi. "*A frequency method for predicting limit cycle bifurcations*". In: Nonlinear Dynamics 13 (1997), pp. 339–360. DOI: https://doi.org/10.1023/A:1008298205786.

[6] Christian Kuehn. "*A mathematical framework for critical transitions: Bifurcations, fast–slow systems and stochastic dynamics*". In: Physica D: Nonlinear Phenomena 240.12 (2011), pp. 1020–1035. DOI: https://doi.org/10.1016/j.physd.2011.02.012.

[7] Joosup Lim and Bogdan I Epureanu. "*Forecasting a class of bifurcations: Theory and experiment*". In: Physical Review E 83.1 (2011), p. 016203. DOI: https://doi.org/10.1103/PhysRevE.83.016203.

[8] Lawrence Perko. *Differential Systems and Dynamical Systems*. Texts in Applied Mathematics 7, Third Edition. y Springer-Verlag, New York, Inc., 2001. ISBN: 9788771251470.

[9] Timothy M. Lenton. "*Early warning of climate tipping points*". In: (2011). DOI: https://doi.org/10.1038/NCLIMATE1143.

[10] Warren S McCulloch and Walter Pitts. "*A logical calculus of the ideas immanent in nervous activity*". In: The bulletin of mathematical biophysics 5 (1943), pp. 115–133. DOI: https://doi.org/10.1007/BF02478259.

[11] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005. ISBN: 0-8058-4300-0.

[12] Bohdan Macukow. "*Neural networks–state of art, brief history, basic models and architecture*". In: Computer Information Systems and Industrial Management: 15th IFIP TC8 International Conference, CISIM 2016, Vilnius, Lithuania, September 14-16, 2016, Proceedings 15. Springer. 2016, pp. 3–14.

[13] Max Roser, Hannah Ritchie, and Edouard Mathieu. "*Technological Change*". In: Our World in Data (2013). https://ourworldindata.org/technological-change.

[14] Mikkel N. Schmidt Tue Herlau and Morten Moerup. *Introduction to Machine Learning and Data Mining.* Lecture notes, Spring 2022, version 1.0. Technical University of Denmark, 2022. ISBN: 9788771251470.

[15] Sebastian Ruder. "*An overview of gradient descent optimization algorithms*". In: CoRR abs/1609.04747 (2016). DOI: https://doi.org/10.48550/arXiv.1609.04747.

[16] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. "*An overview of reservoir computing: theory, applications and implementations*". In: Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007. 2007, pp. 471–482.

[17] Zachary C Lipton, John Berkowitz, and Charles Elkan. "*A critical review of recurrent neural networks for sequence learning*". In: arXiv preprint arXiv:1506.00019 (2015). DOI: https://doi.org/10.48550/arXiv.1506.00019.

[18] H. Jaeger. *The "echo state" approach to analysing and training recurrent neural networks.* GMD Report 148. GMD - German National Research Institute for Computer Science, 2001.

[19] Gary C. McDonald. "*Ridge regression*". In: WIREs Computational Statistics 1.1 (2009), pp. 93–100. DOI: https://doi.org/10.1002/wics.14.

[20] Bruno Galuzzi et al. "*Hyperparameter optimization for recommenderx systems through Bayesian optimization*". In: Computational Management Science 17 (Dec. 2020). DOI: https://doi.org/10.1007/s10287-020-00376-3.

[21] Istvan Lengyel, Gyula Rabai, and Irving R Epstein. "*Experimental and modeling study of oscillations in the chlorine dioxide-iodine-malonic acid reaction*". In: Journal of the American Chemical Society 112.25 (1990), pp. 9104–9110. DOI: https://doi.org/10.1021/ja00181a011.

[22] Richard J Field and Maria Burger. "*Oscillations and traveling waves in chemical systems*". In: (1985).

Appendices

A Reservoir machine training function

```
%% Source: Kong et al, 2021

function [validation_performance,W_in,W_r,W_out,t_validate,
   x_real,x_validate] = ...
    func_STP_train(udata,tp_train_set,flag,W_in_type,W_r_type,
       validation_type)

% W_in: The input matrix.
% W_r: The recurrent matrix in the hidden layer, i.e. the
   reservoir network, or the adjacency matrix of the hidden
   layer.  (We called it matrix 'A' in the paper)
% W_out: The output matrix.
% udata: The training data set.
% tp_train_set: The set of training control parameters. (
   hyperparameters)

% use multiple trials of training time series to train one
   W_out
% Tp is affecting globally. Each node receives the same all
   control parameter
% W_in_type
%           1 : Each node in the hidden layer receives all
   dimensions of the input, i.e. the matrix W_in is dense.
%           2 : (relevant to us) Each node in the hidden layer
   receives only one dimension of the
%               input time series. But the input control
   parameter is projected on all the hidden nodes.
% W_r_type
%           1 : (relevant to us) symmeric, normally distributed
   , with mean 0 and variance 1
%           2 : asymmeric, uniformly distributed between 0 and
   1
% validation type
%           1 : using the largest RMSE among the tp_length
   numbers of trials of training time series
%           2 : prediction horizon
%           3 : product of all the tp_length numbers of RMSE
%           4 : average of all the tp_length numbers of RMSE

```

```
27 % size of udata: ( tp_length, temporal_steps, dim + tp_dim )
28
29 fprintf('in train %f\n',rand)
30 % flag_r_train = [n k eig_rho W_in_a a beta...
31 %                  0 train_r_step_length validate_r_step_length
       reservoir_tstep dim
32 %                  success_threshold];
33 n = flag(1); % number of nodes in the hidden layer
34 k = flag(2); % mean degree of W_r
35 eig_rho = flag(3); % spectral radius of W_r
36 W_in_a = flag(4); % scaling of W_in
37 a = flag(5); % leakage parameter
38 beta = flag(6); % regularization parameter of the l-2 linear
       regression
39
40 train_length = flag(8);
41 validate_length = flag(9);
42
43 tstep = flag(10); % length of each time step in the training
       data
44 dim = flag(11); % dimensionality of the training time series (
       excluding the control parameter)
45
46 if validation_type == 2
47     success_threshold = flag(12); % tolerence threshold of
           error when determining prediction horizon
48 else
49     success_threshold = 0;
50 end
51
52 tp_length = length(tp_train_set); % number of trials of
       training time series, i.e. the number of different control
       parameters (bifurcation parameters) in the training data
53 tp_dim = size(udata,3) - dim; % dimensionality of the control
       parameter (bifurcation parameter)
54
55 validate_start = train_length+2;
56
57 % define W_in
58 if W_in_type == 1
59     W_in = W_in_a*(2*rand(n,dim+tp_dim)-1);
60 elseif W_in_type == 2
61     % each node is inputed with with one dimenson of real data
62     % and all the tuning parameters
```

```
63      W_in=zeros(n,dim+tp_dim);
64      n_win = n-mod(n,dim);
65      index=randperm(n_win); index=reshape(index,n_win/dim,dim);
66      for d_i=1:dim
67          W_in(index(:,d_i),d_i)=W_in_a*(2*rand(n_win/dim,1)-1);
68      end
69      W_in(:,dim+1:dim+tp_dim) = W_in_a*(2*rand(n,tp_dim)-1);
70  elseif W_in_type == 3
71      dim_sep = dim+tp_dim-1;
72      W_in = zeros(n,dim_sep+1);
73      n_win = n - mod(n,dim_sep);
74      index = randperm(n_win); index=reshape(index,n_win/dim_sep,
            dim_sep);
75      for d_i=1:dim_sep
76          W_in(index(:,d_i),d_i) = W_in_a*(2*rand(n_win/dim_sep
                ,1)-1);
77      end
78      W_in(:,end) = W_in_a*(2*rand(n,1)-1);
79  else
80      fprintf('W_in type error\n');
81      return
82  end
83
84  % define W_r
85  if W_r_type == 1
86      W_r=sprandsym(n,k/n); % symmeric, normally distributed,
            with mean 0 and variance 1.
87  elseif W_r_type == 2
88      k = round(k);
89      index1=repmat(1:n,1,k)'; % asymmeric, uniformly distributed
            between 0 and 1
90      index2=randperm(n*k)';
91      index2(:,2)=repmat(1:n,1,k)';
92      index2=sortrows(index2,1);
93      index1(:,2)=index2(:,2);
94      W_r=sparse(index1(:,1),index1(:,2),rand(size(index1,1),1),n
            ,n);
95  else
96      fprintf('res_net type error\n');
97      return
98  end
99  % W_r, adjacency matrix of the hidden layer, i.e. the reservoir
        network
100 % rescale eig
```

```
101 eig_D=eigs(W_r,1); %only use the biggest one. Warning about the
        others is harmless
102 W_r=(eig_rho/(abs(eig_D))).*W_r;
103 W_r=full(W_r);
104
105 % training
106 %disp('  training...')
107 len_washout = 10; % number of intial training steps to be
        dropped during regression,
108 % to lower the possible effects of transients in the hidden
        states.
109
110 r_reg = zeros(n,tp_length*(train_length-len_washout));
111 y_reg = zeros(dim,tp_length*(train_length-len_washout));
112 r_end = zeros(n,tp_length);
113 for tp_i = 1:tp_length % for all the different control
        parameters (bifurcation parameters)
114     train_x = zeros(train_length,dim+tp_dim);
115     train_y = zeros(train_length,dim+tp_dim);
116     train_x(:,:) = udata(tp_i,1:train_length,:);
117     train_y(:,:) = udata(tp_i,2:train_length+1,:); % the
            desired output is the prediction of the next step
118     train_x = train_x';
119     train_y = train_y';
120
121     r_all = [];
122     r_all(:,1) = zeros(n,1);%2*rand(n,1)-1;%
123     for ti = 1:train_length
124         r_all(:,ti+1) = (1-a)*r_all(:,ti) + a*tanh(W_r*r_all(:,
                ti)+W_in*train_x(:,ti));
125     end
126     r_out = r_all(:,len_washout+2:end);
127     r_out(2:2:end,:) = r_out(2:2:end,:).^2; % squaring the even
            rows
128     r_end(:,tp_i) = r_all(:,end);
129
130     r_reg(:, (tp_i-1)*(train_length-len_washout) +1 : tp_i*(
            train_length-len_washout) ) = r_out; % the hidden state
            during training
131     y_reg(:, (tp_i-1)*(train_length-len_washout) +1 : tp_i*(
            train_length-len_washout) ) = train_y(1:dim,len_washout
            +1:end); % the training target
132 end
```

```
133 W_out = y_reg *r_reg'*(r_reg*r_reg'+beta*eye(n))^(-1); % the
    ridge regression between y_reg and r_reg
134 % Now we got the readout layer (output layer) W_out
135
136 % validate
137 %disp('validating...')
138 rmse_set = zeros(1,tp_length);
139 success_length_set = zeros(1,tp_length);
140
141 validate_predict_y_set = zeros(tp_length,validate_length,dim);
142 validate_real_y_set = zeros(tp_length,validate_length,dim);
143 for tp_i = 1:tp_length
144     validate_real_y_set(tp_i,:,:) = udata(tp_i,validate_start:(
        validate_start+validate_length-1),1:dim);
145
146     r = r_end(:,tp_i);
147     u = zeros(dim+tp_dim,1); % input
148     u(1:dim) = udata(tp_i,train_length+1,1:dim);
149     %u(dim+1:end) = udata(tp_i,train_length+1,dim+1:end);
150     for t_i = 1:validate_length
151         u(dim+1:end) = udata(tp_i,train_length+t_i,dim+1:end);
152         r = (1-a) * r + a * tanh(W_r*r+W_in*u);
153         r_out = r;
154         r_out(2:2:end,1) = r_out(2:2:end,1).^2; % squaring even
            rows
155         predict_y = W_out * r_out; % output
156         validate_predict_y_set(tp_i,t_i,:) = predict_y;
157         u(1:dim) = predict_y; % update the input. The output of
            the current state is the input of the next step.
158     end
159
160
161     error = zeros(validate_length,dim);
162     error(:,:) = validate_predict_y_set(tp_i,:,:) -
        validate_real_y_set(tp_i,:,:);
163     %rmse_ts = sqrt( mean( abs(error).^2 ,2) );
164     se_ts = sum( error.^2 ,2);
165
166     success_length_set(tp_i) = validate_length * tstep;
167     for t_i = 1:validate_length
168         if se_ts(t_i) > success_threshold
169             success_length_set(tp_i) = t_i * tstep;
170             break;
171         end
```

```
    end
    % if there is NaN in prediction, success length = 0 and
       rmse = 10
    if sum(isnan(validate_predict_y_set(:))) > 0
        success_length_set(tp_i) = 0;
        se_ts = 10;
    end

    rmse_set(tp_i) = sqrt(mean(se_ts));
end

if validation_type == 1
    validation_performance =  max(rmse_set);
elseif validation_type == 2
    success_length = min(success_length_set);
    %fprintf('attempt success_length = %f \n',success_length);
    validation_performance = success_length;
elseif validation_type == 3
    for tp_i = 1:tp_length
        rmse_set(tp_i) = max(rmse_set(tp_i),10^-3);
    end
    validation_performance = prod(rmse_set);
elseif validation_type == 4
    validation_performance =  mean(rmse_set);
else
    fprintf('validation type error');
    return
end

t_validate = tstep:tstep:tstep*validate_length;
x_validate = validate_predict_y_set;
x_real = validate_real_y_set;

end
```

B Prediction function

```
%% Source: Kong et al, 2021

function predict = func_STP_predict(x_warmup,tp,W_in,W_r,W_out,
   flag)
% included warmup
```

```
5  % flag_r = [n dim a warmup_r_step_length predict_r_step_cut
       predict_r_step_length];
6  n = flag(1); % number of nodes in W_r
7  dim = flag(2); % dimensionality of the target system
8  a = flag(3);
9  warmup_length = flag(4);
10 predict_cut = flag(5); % drop the (possible) transient
11 predict_length = flag(6);
12
13 dim_tp = length(tp); % the dimensionality of the bifurcation
       parameter (tp).
14 % In this work, we focus on the simple case where length(tp) =
       1.
15
16 r = zeros(n,1); % hidden state
17 u = zeros(dim+dim_tp,1); % input
18 u(dim+1:end) = tp; % bifurcation parameter
19 %% warm up begin
20 % the purpose of warm up is to prepare the hidden state r (so
       that the
21 %  hidden sate already has gone through the time series for
       some steps
22 %  before it begins predicting, probably leads to better
       accuracy)
23 if warmup_length ~= 0
24     x_warmup = x_warmup(1:warmup_length,:);
25     for t_i = 1:(warmup_length-1)
26         u(1:dim) = x_warmup(t_i,:);
27         r = (1-a) * r + a * tanh(W_r*r+W_in*u);
28     end
29 else
30     x_warmup = zeros(dim,1);
31 end
32 %% warm up end
33
34 %% predicting
35 % disp('  predicting...')
36 predict = zeros(predict_cut + predict_length,dim);
37 u(1:dim) = x_warmup(end,:); % prepare the starting input for
       prediction
38 % the starting hidden state r for prediction is prepared during
       the warm up
39
40 for t_i=1:predict_cut + predict_length
```

```
    r = (1-a) * r + a * tanh( W_r*r+W_in*u );

    r_out = r;
    r_out(2:2:end) = r_out(2:2:end).^2; % even number -> squared
    predict(t_i,:) = W_out*r_out; % output

    u(1:dim) = predict(t_i,:); % update the input. The output of the current step becomes the input of the next step.
end

predict = predict(predict_cut+1 : end,:);

end
```

C RK4 function

```
%% Source: Kong et al, 2021

function [t,x] = ode4(f,t,x0)
% 4th order Runge-Kutta solver with constant step length
% x0 is the initial state

x = zeros(length(x0),length(t));
x(:,1) = x0;
h = t(2) - t(1); % step length
for step_i = 1: length(t)-1
    k1 = f( t(step_i), x(:,step_i) );
    k2 = f( t(step_i) + h/2, x(:,step_i) + h/2 * k1 );
    k3 = f( t(step_i) + h/2, x(:,step_i) + h/2 * k2 );
    k4 = f( t(step_i) + h, x(:,step_i) + h * k3 );
    x(:,step_i + 1) = x(:,step_i) + h/6 * (k1 + 2*k2 + 2*k3 + k4);
end
x = x';
end
```

D ODE4 function (Example 2)

```
function dxdt = eq_Chem(~,x,flag)
% flag = [a b];
a = flag(1);
```

```
b = flag(2);

dxdt = zeros(2,1);
dxdt(1) = a-x(1)-4*x(1)*x(2)/(1+x(1)^2);
dxdt(2) = b*x(1)*(1-x(2)/(1+x(1)^2));
end
```

E Training and Prediction function (Example 2)

```
%% Training
% Parameters of the system
dim = 2;
a_chem = 10; % bifurcation parameter

n = 400;
k = 236;
eig_rho = 1.79;
W_in_a = 0.61;
tp_W = 0.76;
tp_bias = 0.92;
a = 0.09;
beta = 0.0005;

reservoir_tstep = 0.015;
ratio_tstep = 5;
% reservoir_tstep/ratio_tstep = length of time step for RK4

train_r_step_cut = round( 10 / reservoir_tstep ); % drop the
   transient in data
train_r_step_length = round( 1000 /reservoir_tstep );
validate_r_step_length = round( 8 /reservoir_tstep );


bo = 5;  % best of

para_train_set = [3.46 3.47 3.48 3.49];
tp_train_set = para_train_set;

%% main

tmax_timeseries_train = (train_r_step_cut + train_r_step_length
    + validate_r_step_length + 20) * reservoir_tstep; % time,
   for timeseries
```

```
32 rng('shuffle');
33 tic;
34
35 rmse_min = 10000;
36 for bo_i = 1:bo
37     %%% preparing training data
38     fprintf('preparing training data...\n');
39
40     train_data_length = train_r_step_length +
          validate_r_step_length + 10;
41     train_data = zeros(length(tp_train_set), train_data_length,
          dim+1); % data that goes into reservior_training
42     for tp_i = 1:length(tp_train_set)
43         tp = tp_train_set(tp_i);
44         b_chem = para_train_set(tp_i);  %% system sensitive
45
46         flag_Chem = [a_chem b_chem];
47         ts_train = ones(200,dim);
48         while var(ts_train(end-100:end,1)) < 1e-5
49             x0 = [1+rand,0+rand];
50             [t,ts_train] = ode4(@(t,x) eq_Chem(t,x,flag_Chem)
                  ,0:reservoir_tstep/ratio_tstep:
                  tmax_timeseries_train,x0);
51         end
52         t = t(1:ratio_tstep:end);
53         ts_train = ts_train(1:ratio_tstep:end,:);
54         ts_train = ts_train(train_r_step_cut+1:end,:); % cut
55
56         train_data(tp_i,:,1:dim) = ts_train(1:train_data_length
              ,:);
57         train_data(tp_i,:,dim+1) = tp_W * (tp + tp_bias) * ones
              (train_data_length,1);    %% system sensitive
58     end
59
60     %%% train
61     fprintf('training...\n');
62     flag_r_train = [n k eig_rho W_in_a a beta train_r_step_cut
          train_r_step_length validate_r_step_length...
63         reservoir_tstep dim];
64     [rmse,W_in_temp,res_net_temp,P_temp,t_validate_temp,
          x_real_temp,x_validate_temp] = ...
65         func_STP_train(train_data,tp_train_set,flag_r_train
              ,1,1,1);
66     fprintf('attempt rmse = %f\n',rmse)
```

```
67
68      if rmse < rmse_min
69          W_in = W_in_temp;
70          res_net = res_net_temp;
71          P = P_temp;
72          t_validate = t_validate_temp;
73          x_real = x_real_temp;
74          x_validate = x_validate_temp;
75          rmse_min = rmse;
76      end
77
78      fprintf('%f is done\n',bo_i/bo)
79      toc;
80  end
81
82  %% Prediction
83
84  warmup_r_step_cut = round( 10 /reservoir_tstep ); % drop the
        transient in warming up data
85  warmup_r_step_length = round( 200 / reservoir_tstep );
86
87  predict_r_step_cut = round( 0 /reservoir_tstep );
88  predict_r_step_length = round( endi / reservoir_tstep );
89
90  tp_a = 3.51; % prediction bifurcation parameter
91  b_chem_warmup = min(para_train_set); % warm-up bifurcation
        parameter
92
93  tmax_timeseries_warmup = (warmup_r_step_cut +
        warmup_r_step_length + 5 ) * reservoir_tstep;
94
95  rng('shuffle');
96  tic;
97
98  %% main
99  flag_Chem = [a_chem b_chem_warmup];
100 ts_warmup = NaN;
101 while isnan(ts_warmup(end,1))
102     x0 = [1+rand,0+rand];
103     [t,ts_warmup] = ode4(@(t,x) eq_Chem(t,x,flag_Chem),0:
          reservoir_tstep/ratio_tstep:tmax_timeseries_warmup,x0);
104 end
105
106 t = t(1:ratio_tstep:end);
```

```
107 ts_warmup = ts_warmup(1:ratio_tstep:end,:);
108 ts_warmup = ts_warmup( warmup_r_step_cut+1 : warmup_r_step_cut+
      warmup_r_step_length, :);
109 flag_r = [n dim a warmup_r_step_length predict_r_step_cut
      predict_r_step_length];
110
111 predict_r = func_STP_predict(ts_warmup,tp_W * ( tp_a + tp_bias)
      ,W_in,res_net,P,flag_r);
112 toc;
```