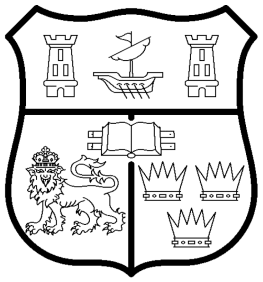


University College Cork

MSc

Mathematical Modelling and Machine Learning

Predicting R-tipping using a machine learning approach based on early warning signals for B-tipping

Barry O'Donnell

September, 2022

Acknowledgement

I would like to offer thanks to my supervisor Dr. Kieran Mulchrone. His support and teachings for the duration of this thesis were invaluable.

Thank you to my family, especially my parents, Bryan and Petrina, who have offered an endless amount of support throughout my studies.

I would like to thank friends, old and new, for their good company and passing wisdom throughout the years.

Abstract

Tipping points are dramatic and typically undesirable sudden changes in the state of a system. The ability to predict tipping points in this modern data-driven society could prove useful in the areas of climate studies, finance, and medicine. In this report we show the possibility of teaching a neural network the ability to predict tipping points in a system. The network predicts using the early warning signals (EWS) that occur before tipping points. We will show that it is possible to predict rate-induced (R) tipping points from learning the EWS from bifurcation-driven (B) and noise-induced (N) tipping, and thus support the idea that EWS are universal indicators for tipping points.

Contents

Chapter 1

Introduction

It is not uncommon to hear the phrase that "[we] are approaching a tipping point," in modern day discourse when discussing the world around us. A tipping point is typically interpreted as a point at which a radical change has occurred, where a cascade of events has been unleashed and cannot be reversed. Tipping points are ubiquitous across all aspects of society, including climate change [1], finance [2] and medicine [3, 4]. Tipping points can cause unpredictable changes to dynamical systems or models that can leave the system in an unwanted or a less preferable state. It is beneficial for everybody to be aware of the possibility of a tipping point in order to prepare for these unwanted changes. If given enough time ahead of the tipping point, an undesirable outcome may be avoided. While tipping points can be analytically examined in well-defined systems, it is unrealistic to assume that current models can sufficiently and accurately imitate real life conditions. It is more likely that the systems are subject to stochastic effects, or are typically overly complex and non-trivial. Although the design of these models involves compromises, they still allow us to predict and analyse outcomes close to the system's true intentions. "All models are wrong, but some are useful" - this quote from statistician George Box is the mantra for model design [5].

Predicting an unpredictable outcome may seem a Herculean task, but systems

can produce early warning signals (EWS) that indicate an oncoming tipping point. These EWS consist of a system exhibiting correlation with a lagged version of itself (aptly called autocorrelation), increase in variance in the system, and critical slowing in the system as it transitions from one stability to another. These EWS do not occur solely in well-defined mathematical systems, but in real life systems too [6].

The goal of this thesis is to test replicate the results produced by Bury et al. [7], who used a deep neural network to predict the possibility of a tipping point occurring. We will expand to predicting a more recently discussed form of tipping called rate-induced (R) tipping. Ritchie et al. have proven that EWS, similar to those found in bifurcation-driven (B) and noise-induced (N) tipping points, can be found in systems which exhibit R-tipping [8]. Using a network trained on the EWS found in B-/N-tipping systems, we will show it is possible to predict R-tipping points.

Chapter 2

Dynamical Systems

The concept of a dynamical system originates from a paper by H.J. Poincaré on the three body problem [9]. Poincaré's approach to the three body problem made use of a set of interconnected, non-linear differential equations defining the motion and energy of each body. Poincaré would establish over the course of three memoirs the basic concepts and ideas which would form what we now refer to as *dynamical systems*. Bifurcation analysis, non-linear dynamics, and perturbation theory were all developed from this starting point. This allowed mathematicians and physicists to understand sudden changes in systems, complicated behaviours in systems, and to produce and analyse chaos in systems.

To instill an intuition of what a dynamical system can achieve, we will consider the scenario of a flu outbreak in a small population. As the town's mathematicians, we are tasked with predicting the amount of people who will be infected over the course of the next month to help the local hospital prepare. We begin by categorising the groups of people within the population. The first group is comprised of people who are susceptible to the flu. The second group are people who are currently infected and are infectious. The third group consists of people who were infected, but are now recovered and immune to the flu. Intuition would tell us that the amount of infections would depend on some sort of interaction between the susceptible and the infectious populations. As the

susceptible population decreases, so will the rate at which people will become infected.

Let the population of susceptible people be S, the infected be I, and recovered be R. The total population of the town is then $N = S + I + R$. We assume no incoming or leaving population implying N is constant. Initially, there is a small group of infected individuals. Over time, this infectious group will infect the susceptible people, transferring them into the infected group. Therefore, the change of the amount of susceptible people over time decreases with respect to the current amount of infected people, or

$$\frac{dS}{dt} \propto -I. \tag{2.1}$$

Similarly, the change of the amount of susceptible people over time will depend on the amount of susceptible people at that time, or

$$\frac{dS}{dt} \propto S. \tag{2.2}$$

Combining these two statements, we obtain a formula for the change of the amount of susceptible people with respect to time,

$$\frac{dS}{dt} = -\beta SI, \tag{2.3}$$

where β is a scaling factor which represents the amount of interaction between the susceptible and the infected. A small β value would indicate a low amount of mixing between the populations. It could also represent the flu's infection rate.

Consider the change in the amount of infected people. This quantity will change with regards to two separate values. Firstly, the incoming newly infected people, βSI and secondly, the people who have recovered from the infection. Constructing it in a similar way to (2.3) we obtain,

$$\frac{dI}{dt} = \beta SI - \gamma I, \tag{2.4}$$

where γ is another scaling factor. It can be interpreted as the severity of the flu, or how long the infected are infectious. Since the recovered become immune after infection, a person must go from $S \to I \to R$. The recovered cohort cannot reenter susceptibility once they are infected. Therefore the change in the amount of recovered is the same as the amount of infected leaving the infected group, or

$$\frac{dR}{dt} = \gamma I. \tag{2.5}$$

Pooling (2.3 – 2.5) together, we have a system of equations which describe the change of each group's population with respect to time.

$$\begin{aligned} \frac{dS}{dt} &= -\beta SI, \\ \frac{dI}{dt} &= \beta SI - \gamma I, \\ \frac{dR}{dt} &= \gamma I, \end{aligned}$$

This collection of interconnected equations is called a dynamical system. A sense of direction can be found in this system, as mentioned earlier where a person must go from susceptible (S) to infected (I) to recovered (R). This specific set of equations is commonly referred to as the SIR model for disease spread. It is the most basic model which mimics the outbreak of disease amongst a population. More sophisticated models make use of more realistic factors, such as birth and death rates (N no longer constant), vaccinations (a fourth variable), reinfection (susceptible population increasing and decreasing) and in more modern papers, the prominence of waves of reinfections [10]. This system is non-linear, which opens up the possibility of there being no analytic solution, requiring a numerical solution instead (this specific version of the SIR model can be solved analytically using a change of variables, but is left as an exercise for the reader). Numerical solutions are an iterative method of solving non-linear models. There are many numerical solutions used today, ranging from simple methods such as Euler's

method, to more sophisticated methods such as the Runge-Kutta 4th order. The numerical solution we will use for this system will be Euler's method due to its simplicity. This involves solving the system one step at a time, calculating S, I and R and repeating it up until a sufficient amount of steps ahead in time is reached. For a human this would be computationally arduous, but it is trivial for a computer. We assume some values for β and γ, either from studying the population before hand or by guessing. Putting it together, we set the initial population to be split into $0.9 : 0.1 : 0$; susceptible, infected and recovered respectively, and choose the time step $\Delta t = 1$ hour, up to 30 days ($24 * 30 = 720$ steps in total).

Fig. 2.1 depicts the relationship between the three groups explicitly. Note the initial sharp decrease in the susceptible population as the group becomes widely infected. With a lack of susceptible people to infect, the infected population growth slows and begins to decrease. Meanwhile the recovered population slowly climbs, eventually becoming the most populated group by the end of the month.

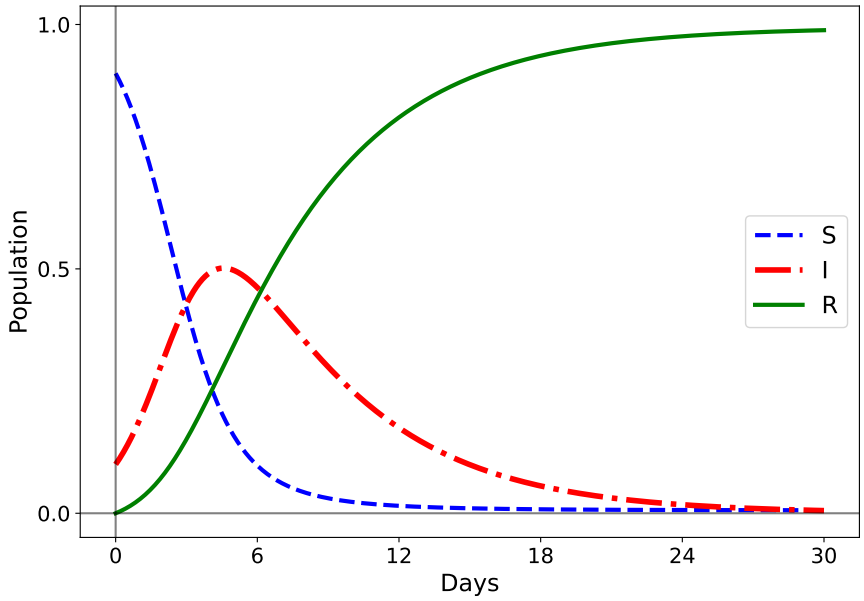


Figure 2.1: Plot of SIR Model. It models a flu outbreak in a population. S refers to the susceptible population, I refers to the infected population, and R refers to the recovered population. $\beta = 1, \gamma = 0.2$.

Modelling the outbreak of a virus is just one application of dynamical systems.

Dynamical systems are not exclusive to virology, or even to the natural sciences; economists use dynamical systems to analyse economic growth and business cycle theories [11].

The paper we will be focusing on and replicating is authored by Bury et al. [7]. The scope of this project will not encompass the entirety of the paper, but rather the underlying principle of using deep learning to predict tipping points in dynamic systems. Let us start by defining clearly what a dynamical system is.

Consider a generic 1-d ordinary differential equation (ODE),

$$\frac{dx}{dt} = \dot{x} = f(x), \tag{2.6}$$

where $f(x)$ is a polynomial of any degree. A system of equations can contain a multitude of ODEs which are connected. This in turn will increase the dimensionality of the system. We can expand the dimensionality of the system by increasing the number of variables, stating that an n-d dynamic system is constructed with,

$$\frac{dx_i}{dt} = \dot{x}_i = f_i(x_1, x_2, \ldots, x_n), \quad i = 1, 2, \ldots, n. \tag{2.7}$$

It is usually displayed as row of ODEs like so:

$$\begin{aligned}
\dot{x}_1 &= f_1(x_1, x_2, \ldots, x_n) \\
\dot{x}_2 &= f_2(x_1, x_2, \ldots, x_n) \\
&\vdots \\
\dot{x}_n &= f_n(x_1, x_2, \ldots, x_n)
\end{aligned}$$

Recall (2.3 – 2.5) which described the SIR model. This is a three dimensional problem as we have three connected ODEs. To understand the most crucial

aspects of this project, we will be using a 1-d model.

Consider the 1-d ODE,

$$\dot{x} = x^2 - 1. \tag{2.8}$$

Letting $f(x) = x^2 - 1$, we can graph this function and do further analysis on the system. We can garner good information from Fig. 2.2, as it is the plot which represents how $\dot{x}$ will grow depending on x itself. Let x^* be an equilibrium or fixed point, i.e. $f(x^*) = 0$. These are points at which $\dot{x}$ is zero, unchanging, which means that x no longer changes over time. At these points, the system is in equilibrium, and will remain at this point. We can solve and find these points in this system, and find that $x^* = \{-1, 1\}$. We now need to determine if the system will ever reach these points. This can be achieved by observing the 'flow' of the system. The flow of the system indicates how the solution of the system will change over time. We notice that if $|x| > 1$, $f(x) > 0$, and thus x will increase in the next time step. We indicate this in Fig. 2.2 with an arrow pointing in the positive direction. Similarly, if $|x| < 1$, $f(x) < 0$, and x will decrease in the next time step. This is indicated with an arrow pointing in the negative direction.

Combining the plot, the equilibrium points, and the arrows indicating flow of the system, we obtain a fully realised depiction of the ODE (2.8). We see two arrows pointing towards one equilibrium point ($x^* = -1$) and two arrows pointing away from the other ($x^* = 1$). We classify the point towards which the system flows as a stable point, and the point from which the system retreats as an unstable point.

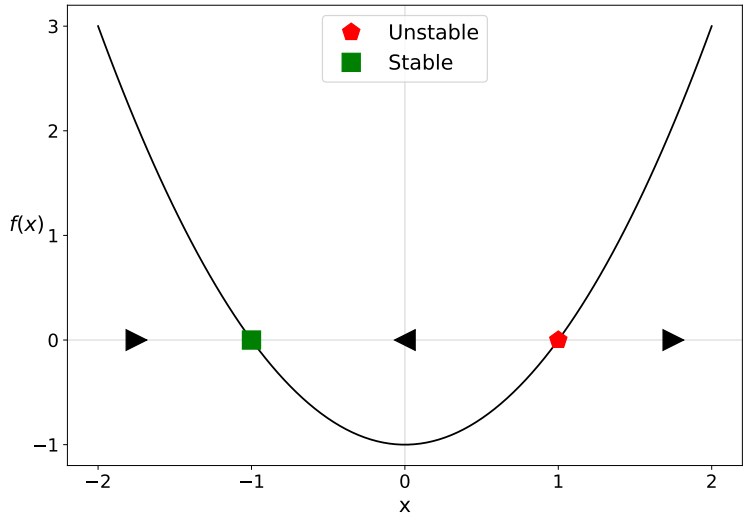


Figure 2.2: Plot of (2.8). The arrows along the x-axis indicate the 'flow' direction, or sign of $\dot{x}$. We see that when the function is negative, the flow of the system is pointing towards the left or negative direction. Conversely, when the function is positive, the flow of the system is in the positive direction.

Expanding to 2-d creates a more complex problem, with the points of equilibrium moving off the x-axis and into 2-d space. Many of these systems are in turn linearised to obtain solutions. The mathematical intricacies of higher dimensional dynamical systems are fascinating and are worth studying, but the complexity of such problems will be omitted from this discussion. Analysis of higher dimension dynamics is discussed by Strogatz at length within *Nonlinear Dynamics And Chaos* [12].

2.1 Bifurcations

In dynamical systems, a bifurcation is the change in the topological structure of a system due to a small change in the system's parameters. We will be interacting with it as a change in stability of a fixed point.

2.1.1 Zero degree system

To build a rigid understanding of bifurcations we will begin with the simplest ODE; the zero degree, 1-d ODE

$$\dot{x} = r, \tag{2.9}$$

where $r \in \mathbb{R}$.

Consider the flow for this system. If $r < 0$, $\dot{x}$ is always negative. There are no fixed points as the sign of $\dot{x}$ never changes. This holds true for the reverse case too. There is an interesting scenario when $r = 0$, $\dot{x} = 0$ and thus there is no flow, and every value along the x-axis is fixed point.

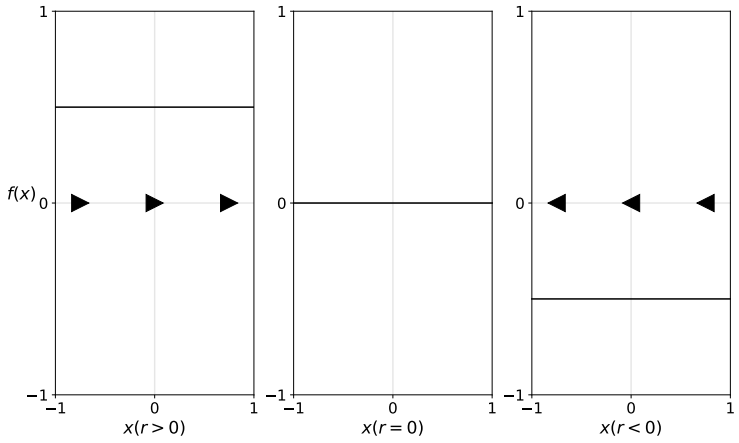


Figure 2.3: Graphical representation of (2.9) as r is varied.

2.1.2 One degree system

Consider the one degree ODE,

$$\dot{x} = cx + r, \tag{2.10}$$

where $c \neq 0$, $\{c, r\} \in \mathbb{R}^2$.

If we set $c = 0$, we return to (2.9). Unlike (2.9), this ODE will always have a fixed point. The fixed point is found at $x^* = -r/c$, but whether it is a stable

or unstable point depends on the sign of c. For $c > 0$, we have a positive sloped line. The flow for $x < x^*$ will be negative, and positive for $x > x^*$, meaning the fixed point at x^* will be unstable. For $c < 0$, the fixed point is stable. It is important to note that r has no effect of the stability of the fixed point, only on the position.

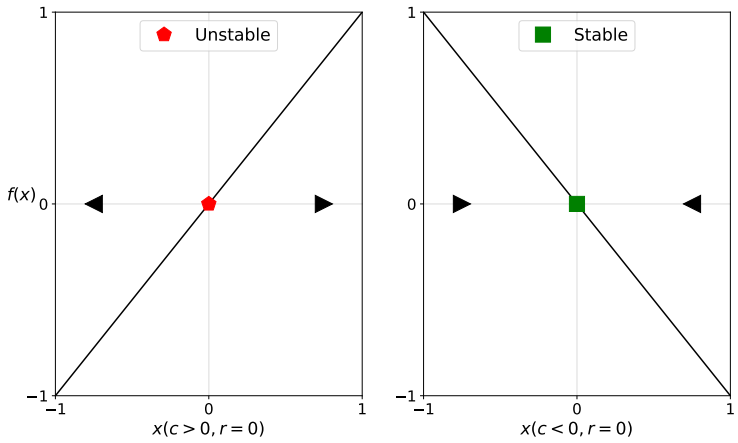


Figure 2.4: Graphical representation of the different states of (2.10).

2.1.3 Two degree system

Consider the two degree ODE,

$$\dot{x} = bx^2 + cx + r \tag{2.11}$$

where $b \neq 0$, $\{b, c, r\} \in \mathbb{R}^3$.

For the sake of brevity, we set $c = 0$. To define all outcomes, one would utilise each parameter. Consider the scenario where $b > 0$ and $r < 0$. This scenario mimics the case discussed in (2.8), with two fixed points, one stable and one unstable. We discovered earlier that the fixed points could be determined from letting $f(x^*) = 0$. When solved we find that fixed points are found at $x^* = \pm\sqrt{-r/b}$. While previously the fixed point was dependent on r, it was always real. As $r \to 0$, the fixed points approach each other. We reach $r = 0$, and the two fixed points become one. The curve $\dot{x}$ is tangential to the x-axis,

and touches it at $x = 0$. For $x < 0$ the flow is in the positive direction, and it is positive for $x > 0$ too. We mark this point as half-stable. If we start at some position $x < 0$ we will reach $x = 0$ after an infinite amount of time. Starting from $x > 0$, $x \to \infty$ rapidly. To increase r any further will transfer the fixed points into the imaginary domain, leaving the x-axis. We observe this transition in Fig. 2.5.

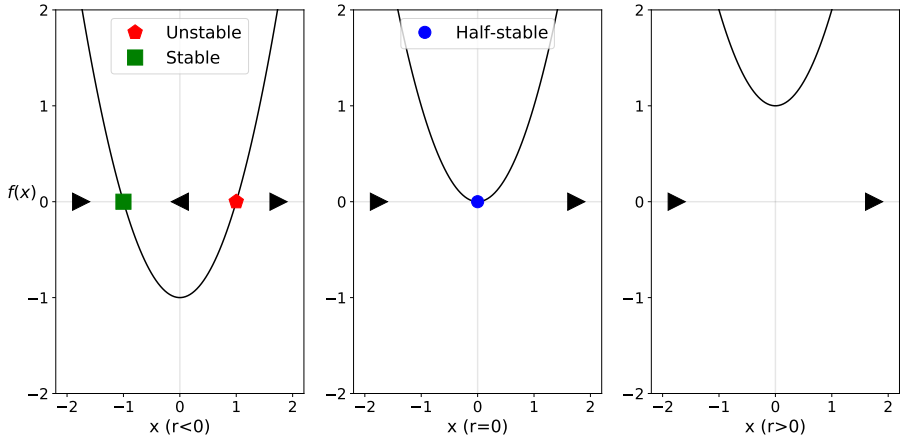


Figure 2.5: Plot of (2.11). As r passes from below zero to above, we see the two equilibrium points become one, then annihilate.

This transition is a bifurcation. The topological structure of the system has changed due to the creation, transition, or vanishing of fixed points in the system. To gather a better understanding for this, the position of the fixed points with respect to the parameter r is graphed. This will produce what what is referred to as a bifurcation diagram as seen in Fig. 2.6. This specific type of bifurcation is called a saddle-node bifurcation. We call the parameter which we vary to incur a bifurcation, the bifurcation parameter.

Were we to flip the sign of b, the reverse would happen. The stability of the fixed points would be swapped, but the bifurcation would occur at $r = 0$ as before.

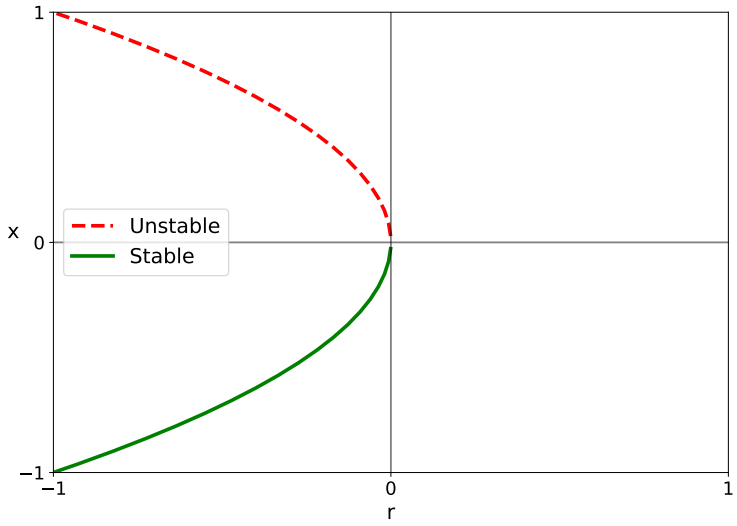


Figure 2.6: Bifurcation diagram of (2.11), $b = 1, c = 0$.

Understanding bifurcations is essential to analyse dynamical systems. They indicate a change in the system's state, and can incur tipping points in systems.

2.1.4 Three degree system

Consider the three degree ODE,

$$\dot{x} = ax^3 + bx^2 + cx + r, \tag{2.12}$$

where $a \neq 0$, $\{a, b, c, r\} \in \mathbb{R}^4$. As before, we will choose specific values to achieve the outcome we are most interested in. We choose $b = 0, c > 0$, and set $r = 0$ for now. This provides us with the typical N-shaped cubic curve. Depending on the sign of a, this curve is either N-shaped ($a > 0$), or backwards N-shaped ($a < 0$). We choose $a < 0$. Initially, we have three fixed points, which can be solved trivially for $x^* = \{0, \pm\sqrt{-c/a}\}$. These fixed points will be stable, unstable, and stable respectively. Reintroducing r, we can see it acts as a sort of 'slider' which we can use to adjust the function vertically. Initially fixed points will be displaced, but the stability will remain the same. Now consider the scenario wherein we increase r towards some critical value r^*. At r^*, a local minimum

will touch the x-axis. Three fixed points have become one stable fixed point and one half stable. Increasing r beyond r^* further reduces this to one stable fixed point only, where it will remain for all values $r > r^*$. This will hold true should we instead reduce r towards, and beyond, $-r^*$ due to symmetry. We now have two points of bifurcation at $r = \pm r^*$. We see this series of transitions in Figs. 2.7.

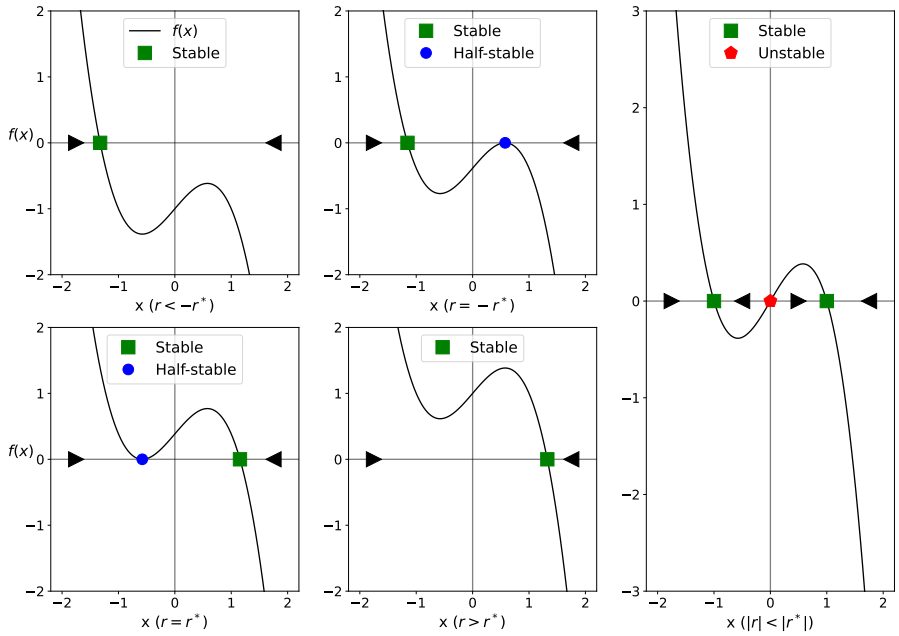
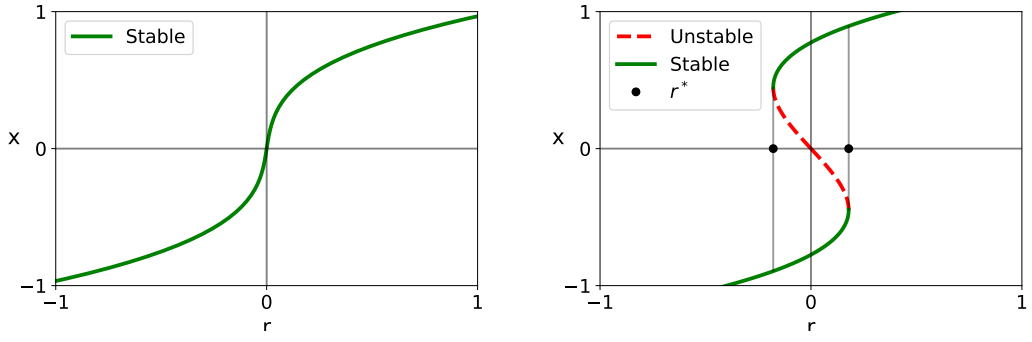


Figure 2.7: Cubic ODE with bifurcation parameter being varied.

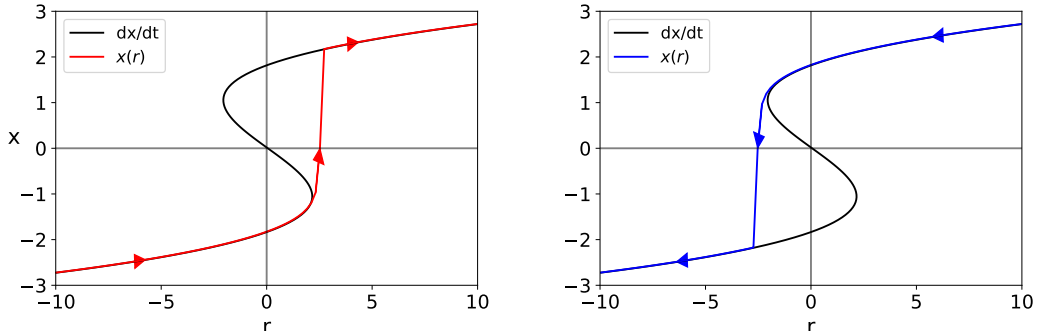
This is seemingly inconsequential, but consider the scenario where we begin at $r < -r^*$ and slowly increase the value. We will always tend towards to the fixed point at $x < 0$. Suddenly, as r passes through $-r^*$, two more fixed points will arise. Since the unstable point is between us and the other stable point, we will remain at our current fixed point. As r continues towards r^* and eventually passes through, our current stable point has vanished and we now rapidly transition towards the other stable point. We remain at this point as r increases. If we graph the bifurcation diagram for such a system we receive Fig. 2.8b. This specific type of bifurcation is a saddle-node bifurcation.



(a) (b)

Figure 2.8: Bifurcation diagrams for (a) $a < 0, b = c = 0$, (b) $a < 0, b = 0, c > 0$.

This bifurcation diagram depicts a 'fold,' where we have multiple values for a given range of r. As we increase r towards r^*, we appear to 'fall' or 'tip' from one stable solution towards the other stable solution. The reverse will hold true too. If we were to decrease r once we have tipped to the other solution, we would decrease along the new solution, not back along the previous solution. As we decrease to $-r^*$, we will again tip back to the original stable solution, as in Fig. 2.9b.



(a) (b)

Figure 2.9: The irreversible nature of a system containing a tipping point.

Such irreversible qualities do have real life counterparts: electrical hysteresis curves, irreversible chemical reactions, and in the same vein many thermodynamic processes.

The scope of this project will require only well defined tipping points from three degree ODEs. Next, we will need a method of generating these systems in

which tipping will always occur.

2.2 Tipping Points

Tipping points of dynamical systems are points in which a system undergoes a radical, irreversible change over a short period of time. A real life parallel for this phenomenon can be seen in many of our climate's dynamical systems [13], competition models [14], and social structures [15]. The study of tipping points is essential for predicting sudden changes in dynamical systems, and mitigating the possible impact of such a change.

Tipping points typically occur through two common means called bifurcation-driven tipping (B-tipping), and noise-induced tipping (N-tipping). Recently, climate studies have been analysing rate-induced tipping (R-tipping), wherein a tipping point occurs due to a sudden increase in the rate at which the bifurcation parameter is changing.

We will begin by discussing B-tipping. Saddle-node bifurcations first begin to appear in two degree ODEs. Tipping of this nature is not physically compatible with the real world as solutions from these tipping points tend to infinity quickly. Instead we will begin with saddle-node bifurcations in three degree ODEs. Consider the three degree ODE,

$$\dot{x} = ax^3 + bx^2 + cx + r = f(x), \tag{2.13}$$

where $\{a, b, c, r\} \in \mathbb{R}^4$. We will apply the same analysis to this ODE as previously done in Section 2.1.

The specific form of ODE we are seeking must comply with the following conditions: have unique local minima and a local maxima, and have a negative third derivative. The first condition can be met by obtaining an expression for the location of the local peaks. Begin by taking the first derivative of $f(x)$

$$\frac{df}{dx} = f'(x) = 3ax^2 + 2bx + c. \tag{2.14}$$

Local maxima and minima are found by solving $f'(x_{\min/\max}) = 0$, obtaining,

$$x_{\min/\max} = -\frac{b}{3a} \pm \frac{1}{3a}\sqrt{b^2 - 3ac}. \tag{2.15}$$

To ensure $x_{\min/\max}$ are real and unique, (2.15) must satisfy the inequality $b^2 - 3ac > 0$, or,

$$b^2 > 3ac. \tag{2.16}$$

The second condition will be required to generate a system which contains a tipping point. Recall from Section 2.1.4 that the system had fixed points with the order of stability stable, unstable, and stable. If we were to reverse this condition, the stabilities would be swapped. If we further recall the bifurcation diagram Fig. 2.8b for this system, with the stabilities swapped, the tipping experienced would be tipping from a stable solution to infinity. This condition will thus require the third derivative of (2.13) to be less than zero,

$$f'''(x) = 6a < 0, \tag{2.17}$$

or,

$$a < 0. \tag{2.18}$$

Provided we adhere to these conditions, we can generate ODEs which contain a saddle-node bifurcation on demand. In Fig. 2.10 we depict a few examples, along with a graph of b – c parameter space. An interesting results of these conditions is that provided $c > 0$, b can be any real value; the closer b^2 is to $3ac$, the flatter the point of inflection becomes.

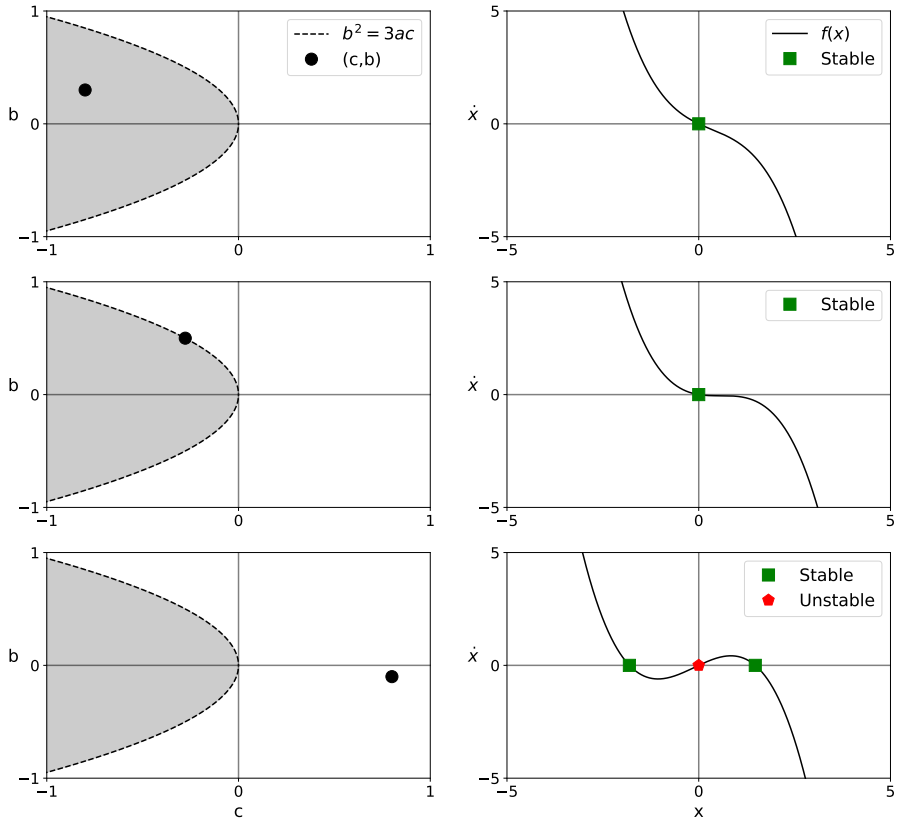


Figure 2.10: Here we are depicting three degree ODEs along with the corresponding $b - c$ parameter space. For simplicity, $a = -0.3$ and $d = 0$. Top: b, c are chosen such that the condition is not met, resulting in a cubic ODE with only one real stable root. Centre: b, c chosen such that $b^2 = 3ac$, causing the cubic ODE to become flat about the origin. Bottom: b, c meet the conditions, producing a cubic ODE with suitable conditions for a saddle-node bifurcation.

Recall earlier that we needed r to pass through some critical value r^* to incur a bifurcation in the system. As r acts similarly to a vertical slider on the function, it is apparent that r^* is the point when either a local maxima or minima is brought to the x-axis. Therefore, $r^*_{\min/\max} = f(x_{\min/\max})$. We again see that should $x_{\min/\max}$ be complex, then r^* leaves the real plane, further indicating that a bifurcation does not occur for real values of r.

For the process of training the neural network, we will be generating solutions from tipping points induced by saddle-node bifurcation. Generating systems

which contain saddle-node bifurcations is useful, but we have only discussed B-Tipping. In reality, it is extremely unlikely that a physical phenomena will follow such a well defined system and tip naturally by bifurcation. It is more meaningful to instead analyse N-tipping or R-tipping.

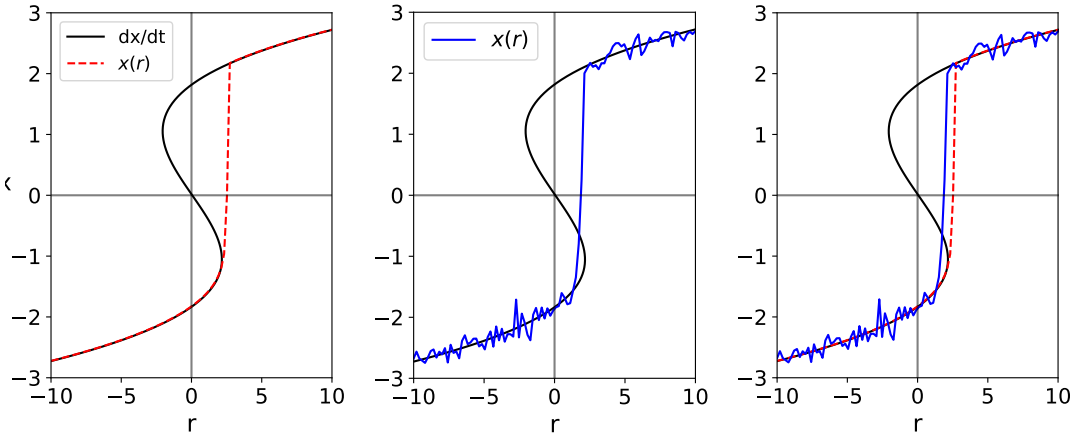
2.2.1 N-tipping

Dynamical systems created to predict physical scenarios are generally fitted with some noise factor. To adapt the well defined three degree ODE to this, we add a stochastic noise term, obtaining,

$$\begin{aligned} dx &= f(x,t)dt + \sigma dW_t, \\ &= \left(ax^3 + bx^2 + cx + r\right)dt + \sigma dW_t, \end{aligned}$$

where σ is a noise scaling factor, and W_t is a standard Wiener process. This expression is a stochastic differential equation (SDE) and can be numerically approximated using the Euler-Maruyama method, a modified version of Euler's method.

Suppose $|r - r^*| = \epsilon$. Suppose the system was subject to an amount of noise greater than ϵ momentarily, causing the bifurcation parameter to jump beyond the bifurcation threshold. A sudden unexpected bifurcation would occur, caused by the noise. Similarly, a bifurcation can be prolonged or delayed should the noise sufficiently decrease the bifurcation parameter below the threshold. This is called N-tipping. In Fig. 2.11, we see an example of N-tipping occurring before B-tipping.



(a) (b) (c)

Figure 2.11: An example of noise causing tipping to happen before bifurcation was reached.

A physical analogy to this form of tipping would be quantum tunnelling, where an electron has a non-zero chance to cross a short potential boundary. Once the electron has tunneled through the barrier, it will shoot off unlikely to cross the boundary again.

2.2.2 R-tipping

We have so far varied the bifurcation parameter linearly over time. Just as with N-tipping, it is unlikely that a parameter will be as well behaved in reality. We will be analysing a simple example system from [13]. Consider the 2-d ODE

$$\dot{x} = (x + \lambda)^2 - \mu \tag{2.19}$$

$$\dot{\lambda} = r, \tag{2.20}$$

with parameter $\lambda(t)$, r is the drift, and $\mu > 0$. Drift refers to how the parameter λ changes over time.

We can solve this system to find the equilibrium points as before in Section 2.1. Solving $\dot{x} = 0$ we obtain two solutions; $\tilde{x}_s := \lambda = \sqrt{\mu} - x$, and $\tilde{x}_a := -\sqrt{\mu} - x$.

These are not singular points but instead a line along which $\dot{x} = 0$ as λ is not a fixed value. These lines are quasi-stable equilibriums (QSE). Flow through these points is purely in the λ direction. Further dynamical analysis will show that these QSE are a stable node and a saddle node respectively.

We are interested in finding a curve along which all solutions are the same called the invariants. This is analogous to the fixed points discussed in Section 2.1, where if we start on those points we will not deviate from them. It can be imagined as the 2-d form of a single point. We assume the form of this curve or line to be $\lambda = mx + c$. Let

$$V(x(t), \lambda(t)) = mx + c - \lambda. \tag{2.21}$$

where V is the solution set of fixed points. We need all solutions to this equation to be constant for each value $(x, \lambda) \in \mathbb{R}^2$, i.e., $\frac{dV}{dt} = 0$. Solve

$$\frac{dV}{dt} = \frac{dV}{dx}\frac{dx}{dt} + \frac{dV}{d\lambda}\frac{d\lambda}{dt} \tag{2.22}$$

$$= m \cdot \Big((x + \lambda)^2 - \mu\Big) + -1 \cdot r = 0. \tag{2.23}$$

We are seeking solutions for m, c. Reorganising the expression, we obtain

$$(m^3 + 2m^2 + m)x^2 + (2m^2c + 2mc)x + (mc^2 - m\mu - r) = 0. \tag{2.24}$$

To obtain values for m, c, we let the coefficients equal to zero.

$$m^3 + 2m^2 + m = 0 \;\; \rightarrow \;\; m = 0,$$

$$m^2 + 2m + 1 = (m + 1)^2 = 0 \;\; \rightarrow \;\; m = -1.$$

We disregard $m = 0$ as it does not have a corresponding c solution.

$$(-1)c^2 - (-1)\mu - r = 0, \tag{2.25}$$

$$\rightarrow \; c = \pm\sqrt{\mu - r}. \tag{2.26}$$

Therefore we have two invariants for this system,

$$A(\mu, r) := \lambda = -\sqrt{\mu - r} - x, \tag{2.27}$$

$$B(\mu, r) := \lambda = \sqrt{\mu - r} - x. \tag{2.28}$$

If $0 < r < \mu$, the system will remain stable if initial $\lambda < \sqrt{\mu - r} - x$. Should the inequality not hold, then $x \to \infty$ as $t \to \infty$. If $r = \mu$ then invariants collapse into the same expression. If $r > \mu$, the invariants enter the complex plane. Every solution regardless of starting position tends towards infinity as $t \to \infty$. Fig. 2.12 depicts the QSE and invariants, and it also shows how the invariants collide and vanish as r approaches μ. The solid lines with arrows indicate flows in the system. Notice that these flows are purely vertical along the QSE.

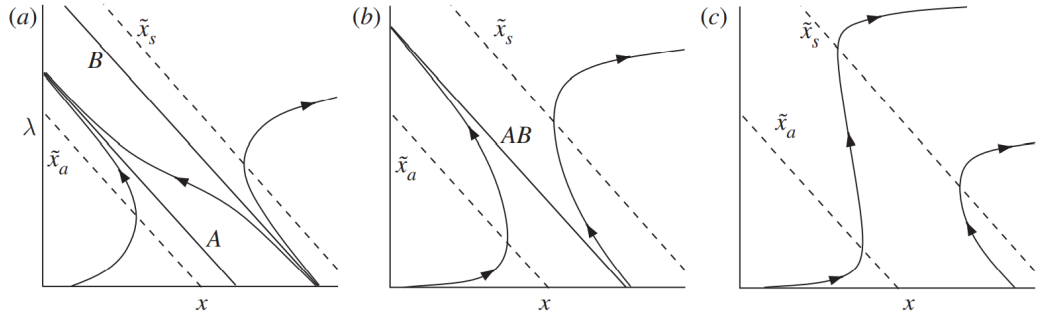


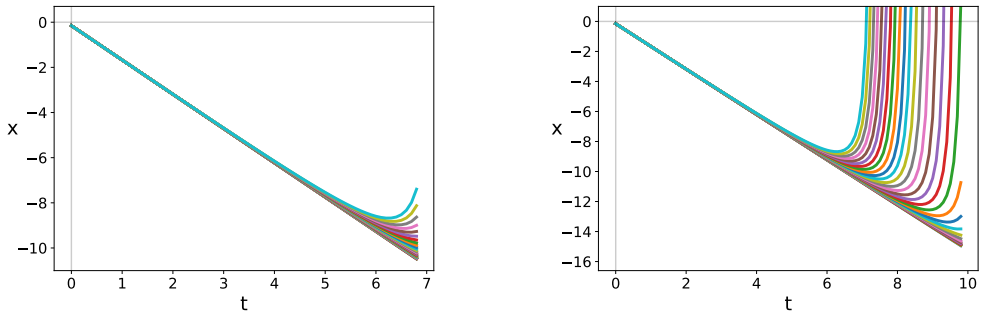
Figure 2.12: Figures depicting the flow near the invariants. (a) $0 < r < \mu$, (b) $r = \mu$, and (c) $r > \mu$. Shown also are the QSE. [13]

In Fig. 2.13, we see the numerical solution to the toy model. Here, each line has the same parameters except for r. It is important to note that each model has an r value greater than μ, so each model should tip. This is because this system is dependent on the period of time it is being analysed over. To dispel

confusion, we will be referring to time steps Δt when discussing the individual time iterations when solving numerically. When referring to the amount of time being analysed, we will use time units. For example, if we are going up 100 time units in 1,000 time steps, $\Delta t = 100/10000 = 0.1$ time units in length.

We use a generic term here for units of time as the systems are scalable. As before in the SIR model, we were discussing days and hours, but dynamic systems are not bound to certain scales of time. They can be as short or as long as needed.

Returning to the R-tipping example, notice that after seven time units, some models have tipped, while others remain stable. If we include a further three time units, a significant portion of the initial models have tipped, but not all of them. This implies that the process for generating data will be different to the process for generating B-tipping samples.



(a) (b)

Figure 2.13: Numerical solutions to (2.19). $x_0 = 0, \lambda_0 = 0, \mu = 1.5, \Delta t = 0.1, r = \{1.51, \ldots, 1.55\}$. In both figures, each colour represents a different value r. The first to tip in each figure is the highest value of r, which is 1.55 in this scenario. We obtain a numerical solution up to (a) $t = 7$, (b) $t = 10$.

In Fig. 2.14, we see the percentage of models which tipped with respect to t time units. When we run the numerical simulation up to 30 time units, we see every model tips within this time period.

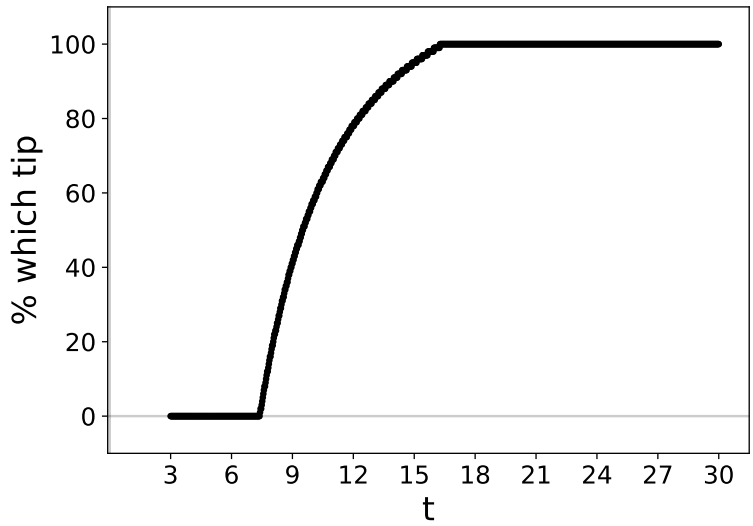


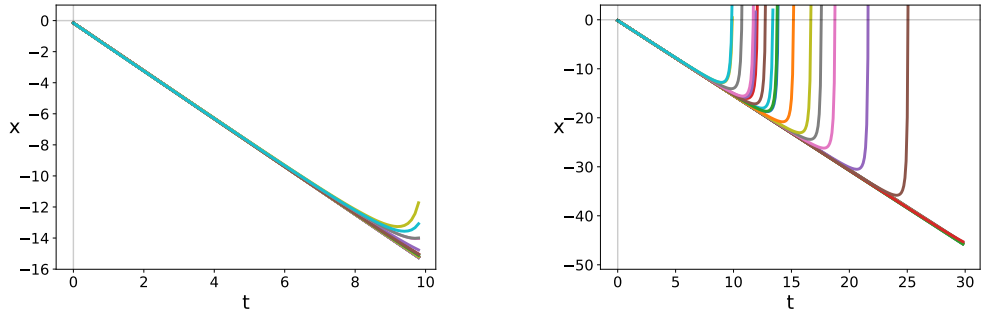
Figure 2.14: Figure depicting the amount of tipped models within t amount of time units. The parameters for this system are $x_0 = 0, \lambda_0 = 0, \mu = 1.5, r = \{1.51, \ldots, 1.55\}, \Delta t = 0.1$. The parameter r is a set of $n = 100$ linearly spaced values between 1.51 and 1.55

Similar to before, we can modify the system and introduce a Wiener process to simulate noise in the system. This system now has the form

$$dx = \Big((x + \lambda)^2 - \mu\Big)\, dt + \sigma dW_t. \tag{2.29}$$

We do not add noise to the parameter $\lambda(t)$ as we are only observing x over time, and a sufficient amount of noise is introduced to the system.

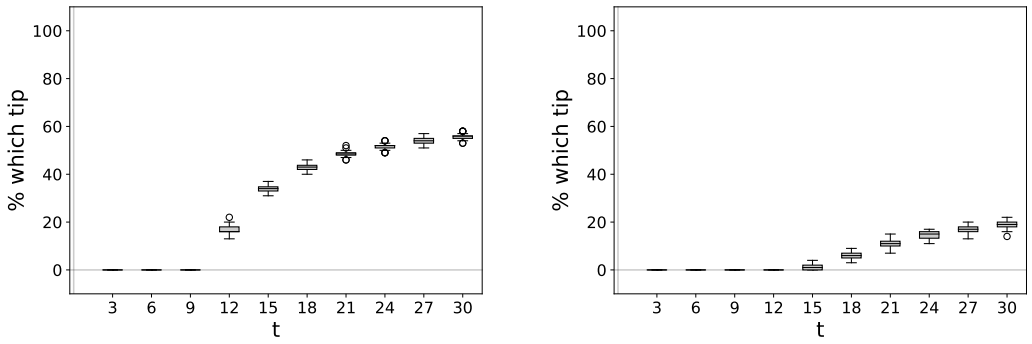
In Fig. 2.15, we see the numerical solutions for the toy model with a small amount of noise introduced. Despite requiring 10 time units for a significant portion of the models to tip previously, it is not sufficient to fully tip a single model exposed to noise. Increasing the duration of the simulation to 100 time units, we begin to see multiple tipping models again.



(a) (b)

Figure 2.15: Numerical solutions to (2.29). The parameters are: $x_0 = 0, \lambda_0 = 0, \mu = 1.5, r = \{1.51, \ldots, 1.55\}, \sigma = 0.005, \Delta t = 0.1$. The parameter r is a set of $n = 30$ linearly spaced values between 1.51 and 1.55, represented in the figure by different line colours.

Further numerical analysis indicates that we expect tipping under noise to reliably begin around 9 – 12 time units. This holds true for this specific set of parameters, but changing the parameters will affect the frequency of R-tipping. Doubling the noise delays the the reliability of the beginning of tipping.



(a) (b)

Figure 2.16: Figure depicting the amount of tipped models within t amount of time units. The parameters are: $x_0 = 0, \lambda_0 = 0, \mu = 1.5, r = \{1.51, \ldots, 1.55\}, \Delta t = 0.1$. The parameter r is a set of $n = 100$ linearly spaced values between 1.51 and 1.55. (a) $\sigma = 0.005$, (b) $\sigma = 0.01$.

This specific system which exhibits R-tipping has no real world analogy. Ashwin et al. applied a system with R-tipping to a climate model, with promising results [13].

2.3 Early Warning Signals

As a system begins to approach a tipping point, early warning signals (EWS) begin to develop. Early warning signals are essential for predicting the sudden changes in dynamical systems. These are split into three categories.

2.3.1 Variance

Variance is the measure of how dispersed the values of a distribution are. It is calculated using,

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{2.30}$$

where n is the number of values being examined, x_i is the i-th value, and $\bar{x}$ is the mean of all n-values. Fig. 2.17 is a visual depiction of variance. Variance increases as we approach a tipping point as the stability of the initial state begins to waver, and ultimately vanish. The sudden jump in values would be apparent in the variance of the system.

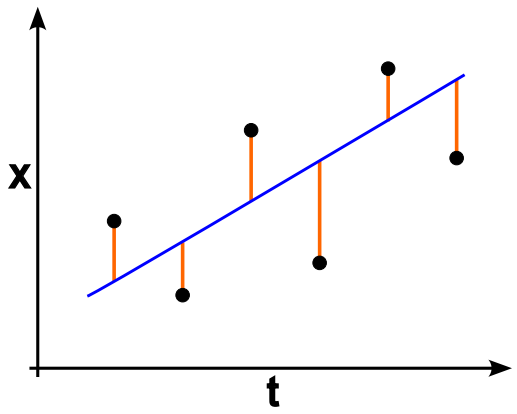


Figure 2.17: Variance is the average vertical displacement from the line of best fit through data.

When calculating the variance for the system as the bifurcation parameter is varied, we will need to use a 'moving window.' We can imagine a moving window being passed over the system only looking at the variance contained within the

window as in Fig. 2.18.

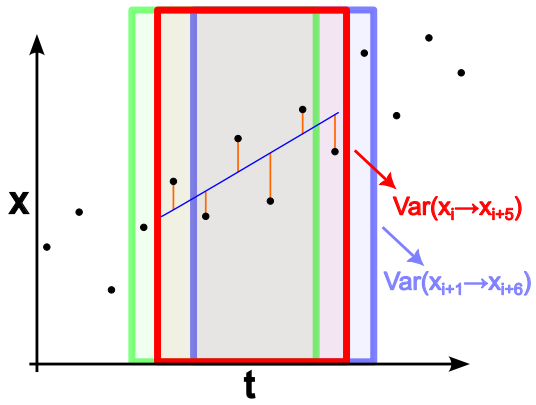


Figure 2.18: Visualisation of the moving window method used to calculate variance in the system.

2.3.2 Autocorrelation

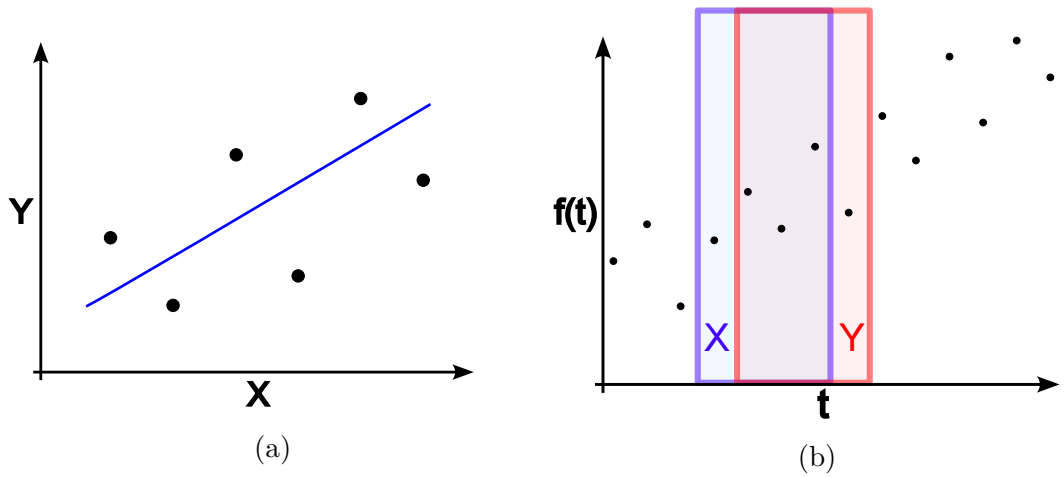
Correlation is a measure of how much effect one set of data (X) has on another (Y). Datasets that have positive correlation indicate that Y increases with X as in Fig. 2.19a, and the opposite holds true for negative correlation. Autocorrelation (AC) is the similarity of a time series and its delayed copy. It is useful for finding repeating patterns in systems. An increase in AC is an indication that a sudden transition will occur.

We will be using the Pearson correlation coefficient r when measuring autocorrelation. Consider two datasets $X = \{x_1, x_2, \ldots, x_n\}, Y = \{y_1, y_2, \ldots, y_n\}$. The correlation between the datasets is given by,

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}} \tag{2.31}$$

where each sum is from $i = 1, \ldots, n$. $\bar{x}, \bar{y}$ are the means of each dataset.

Autocorrelation will make use of a moving window similar to variance. We calculate the correlation between the subsets captured in the windows. For example, we can imagine each window as a subset X and Y as in Fig. 2.19b. We then graph similarly to Fig. 2.19a by joining up the first value of each subset, then the second, and so on. This specific form of AC is called lag-1 AC. The



(a) (b)

Figure 2.19: (a) depicts the line of best fit. Correlation refers to how closely correlated X and Y are. (b) shows how we obtain X and Y from the system using a moving window.

subsets overlap by all but one value in each subset.

2.3.3 Slowing

Critical slowing is an indication that the system is approaching a tipping point. Consider a small perturbation is added to a sufficiently stable system. The system would quickly return to its stable state. As the system approaches the tipping point, the small perturbation will take longer to recover from. This is called critical slowing.

There is no calculation for slowing, but its effects are present in both variance and autocorrelation. We can visualise slowing by using the 'ball in a valley' analogy. Consider Fig. 2.20. The first figure depicts a solution far away from a tipping point. A small perturbation to the ball will still leave it inside the valley, but it may sway around the valley with little change to its initial position. Variance in the ball's position would be low here. The second figure depicts the system approaching a tipping point. A small perturbation to this ball may cause the ball to roll down the hill to its left and to a different solution. This would be analogous to N-tipping if that were to occur. Consider the scenario where it reaches close to the local peak to the left of its starting position. The ball would

slowly return to the starting position. AC would be high, and the variance would increase as the distance from the starting point would be larger.

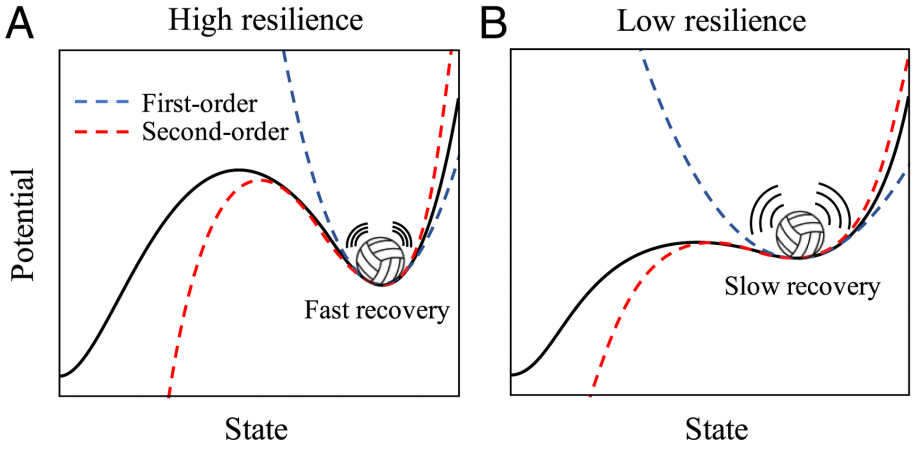


Figure 2.20: Ball in valley example. The axes here represent the first integral of $f(x)$ on the y-axis, and x on the x-axis [7].

Chapter 3

Deep Learning

The goal of this thesis is to train a network that can recognise EWS and predict tipping points in a dynamical system.

3.1 Machine Learning

The first known reference to machine learning methods stems from tech pioneers IBM in 1959 [16, 17]. This research investigated if a computer "can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program." This process utilised the 'tree method'. This method is most analogous to human thought process, and involves looking forward in time trying to perceive each possible move, then choosing the one that benefits you the most. Most humans have an intuitive understanding of the goals of a game of checkers/draughts; you want more pieces than the opponent, and you want the opponent to have no pieces. The machine must be taught this, and so we apply a reward system to inform the machine which moves are 'good' and 'bad.' Fig. 3.1 is a scenario where if a machine wasn't aware of the possible blunder it would make, it would cause a catastrophic loss of pieces. As the machine continues to play the game, it remembers how previous moves affected its current position and score. Depending on whether it is in advantageous or disadvantageous position, the 'desire' to use that same move again

will increase or decrease respectively. Next, we train the machine by making it play hundreds, thousands, maybe even millions of games until it is exposed to as many moves as possible. This is mostly computationally inexpensive for modern computers, but in 1959 this took the engineers over 8 hours of learning to reach a stage where the machine performed better than its creator at a game of draughts.

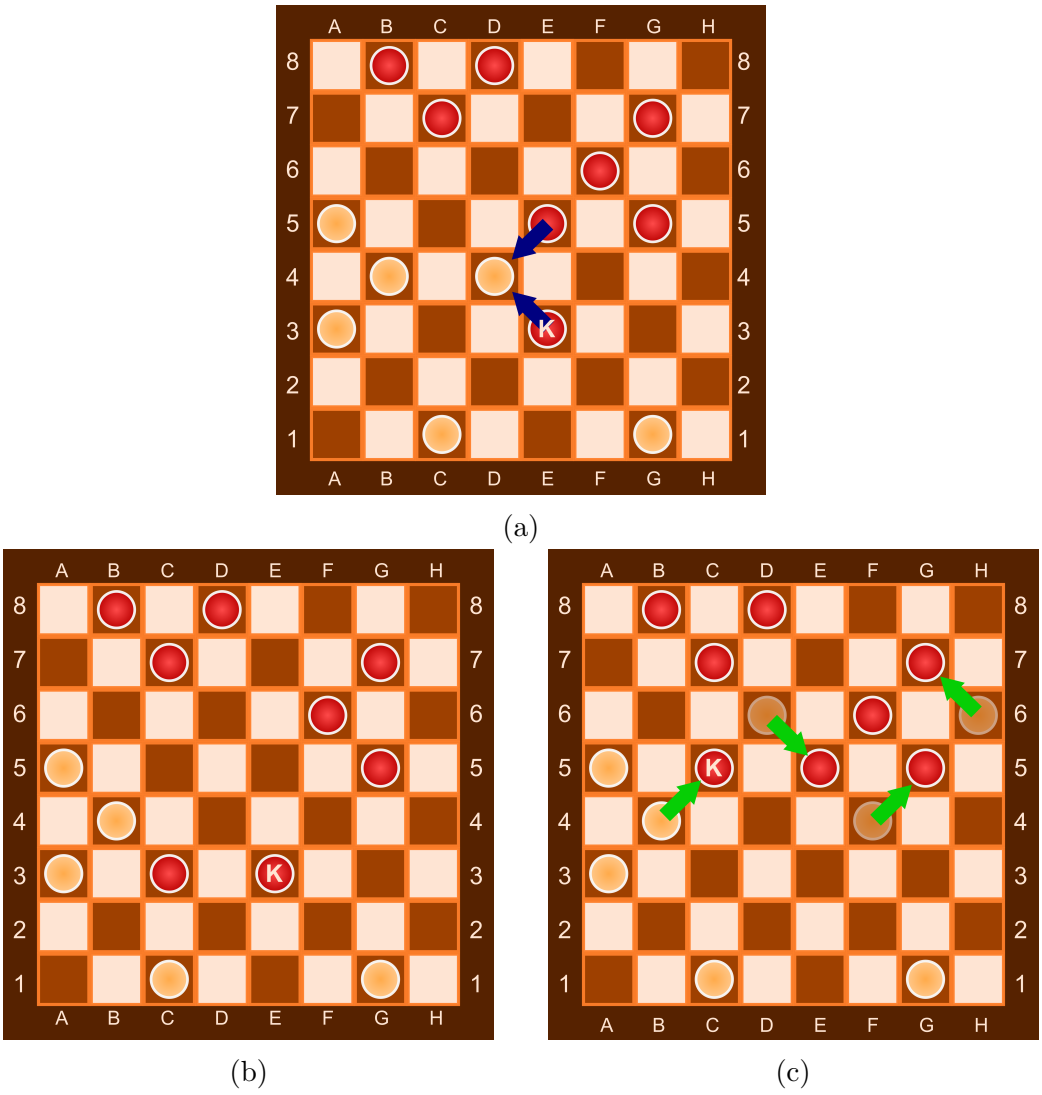


Figure 3.1: Draughts board example. (a) Red's move. They must choose between pawn initiating the capture (b),or king initiating (c). Both moves have the same immediate gain, but the outcome has varying losses. Machine learning would inform the machine to reject the king capture move as it results in catastrophic loss.

While the method used in this first paper is quite primitive, it is noted that it is chosen because of its efficiency with single task optimisations. They note

that one method that could be used instead is the 'Neural-Net Approach,' but the underlying principle remains the same: the machine is asked to make moves, then reward the machine depending on how good the move is.

The Neural-Net Approach, now more commonly referred to as artificial neural networks (ANN), can be traced back to 1943 [18]. The thought process behind the method was to replicate the brain's own processes. This involves many neurons influencing more neurons, some having more effect on one than others.

Consider a neuron receiving a signal from three inputs a, b, c. The neuron will be 'activated' if the neuron receives a certain amount of signal from the inputs. For example, say we are trying to determine if an image contains a dog. Let a represent whether the image contains a tail, b says the image contains four legs, and c states that the image contains two ears. The neuron will activate if the image does have a dog, and remain inactive if there are no dogs. This is perfect! We now have a network that can detect dogs in images – or perhaps only detects if an image contains a tail, four legs and two ears. A cat or mouse would hit all these qualifications, as well as many other animals, so perhaps we need more inputs such as fur length, coat colour, maybe even size.

In reality, ANNs use many clever optimisation techniques to train neurons so that they can detect animals such as dogs detection in images. Understanding and outlining the inner processes of an ANN is outside the scope of this thesis, but further sources are indicated in the bibliography [19, 20, 21].

3.1.1 Architecture of an ANN

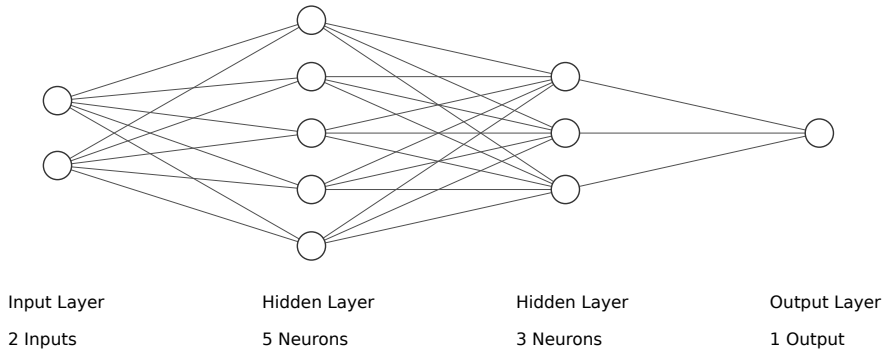


Figure 3.2: Graphical representation of an ANN [22]

ANNs consist of three main components: the inputs, the hidden layers, and the outputs. The input layer defines the shape of the data being processed by the network. This can be an image, 1-d data arrays such as variance as we will make use of later, or even higher dimension dataframes that contain columns of different data points. Once an ANN has a defined input shape, it can only take said input. An ANN which predicts weather using columns of air pressure and wind speed data will be unable to take an image as an input.

The hidden layers connect the input layer to the output. They are 'hidden' as the user is not shown the mathematical operations applied within the layers. Hidden layers can be any size. 'Deep learning' refers to networks which consist of more than one hidden layer. As discussed in Section 3.1, hidden layers can be a collection of neurons. Each neuron is connected by weighted paths and biases to the previous layer. This is commonly referred to as a dense layer.

The output layer is the result of the network's processes on the input. In the case of the dog identifier mentioned above, the output would be true or false (one or zero). Now consider the case of weather prediction. A day can be sunny or rainy, hot or cold, windy or calm. The output here would be three 'true or

false' responses. Output layers are not limited to binary outputs. They can have continuous outputs which can represent scale or probability.

3.1.2 Training Networks

The process of training a network requires a significant amount of data. In the original experiment from IBM, the network was trained on multiple hours of gameplay before an adequate performance was reached [16]. The process of training makes use of an optimisation technique called 'backpropagation.' In short, this method corrects the weights and biases of a networks neurons by checking how incorrect the neuron's influence was on the final answer. Thorough training requires a significant amount of data to obtain optimum weights and biases for networks. Recalling the example of dog image detection, we would need a large amount of images of not just dogs, but also other animals. It is important that the network is exposed to and trained on as much relevant data as possible.

Nowadays, data is generated at unprecedented levels, with rough estimates stating that 2.5 billion gigabytes of data are generated each day [23]. The notion of 'big data' is founded on the principle that each sector of industry produces a significant amount of data waiting to be manipulated in such a way to improve profit, or to optimise a way of living. As machine learning becomes more common and applied to every day life, the ethics of such large amounts of data generation and processing is an area of ongoing debate [24, 25, 26].

3.2 Convolutional Neural Networks

Convolutional neural networks (CNN) are a form of ANNs that use convolution operations within the hidden layers of the network. Convolution is a mathematical operation that passes a function through a 'filter' defined by another function. The convolution between two functions returns how the shape of one

function is affected by the other.

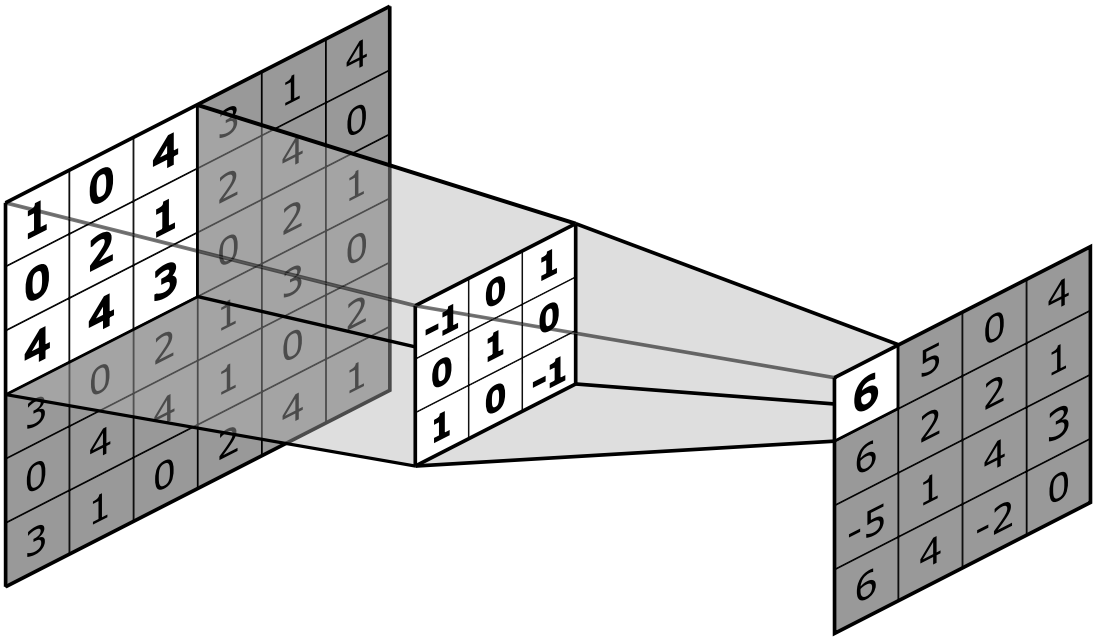


Figure 3.3: A visualisation of the convolutional method. A 3×3 kernel is passed over a 6×6 array. The kernel produces a 4×4 array. The top left corner is calculated from $(1 \times -1) + (0 \times 0) + (4 \times 1) + (0 \times 0) + (2 \times 1) + (1 \times 0) + (4 \times 1) + (4 \times 0) + (3 \times -1) = 6$

In the context of machine learning, the function which acts as a filter is referred to as the 'kernel.' The function which passes through the filter is the input. Convolutional layers in neural networks allow the network to cleverly filter out non-essential aspects of the input, while retaining the useful information needed for classification.

We will be using a CNN to classify whether a tipping point will occur from looking at EWS.

3.3 Recurrent Neural Networks

Recurrent neural networks (RNN) are another form of ANNs that can retain memory of previous data passed through the network. Artificial neural networks are generally built so that data enters the network, passes through the network, and a resulting output is calculated. This network would be referred to as a 'feedforward' network, as the data only passes through each layer once. Neural

networks that allow for the data to be reprocessed once it has entered the neural network are referred to as 'feedback' or recurrent neural networks.

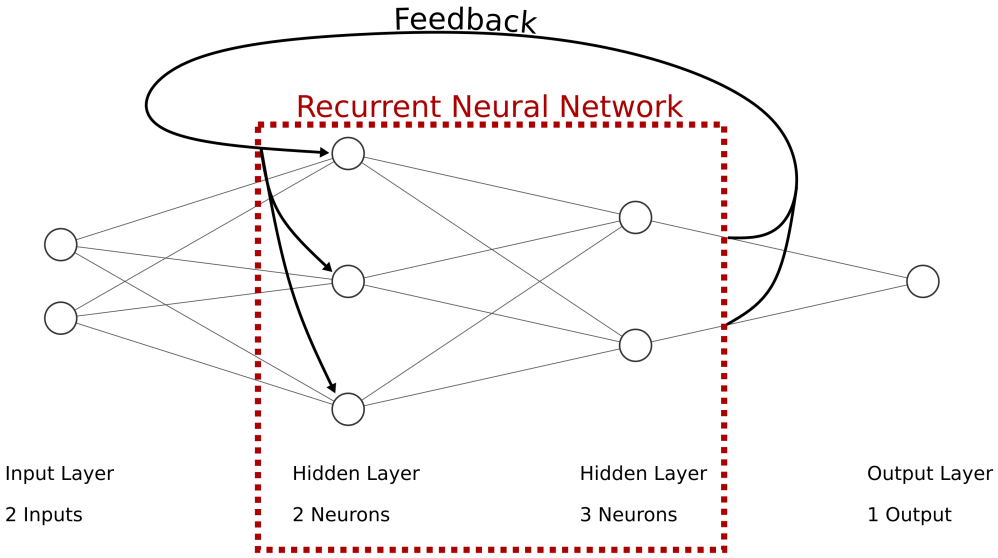


Figure 3.4: Visualisation of the recurrent property of an RNN [22]

Recurrent neural networks pass data backwards into the network. This allows the network to pick up 'context.' Consider the scenario wherein we try to detect an object moving in a video. We would begin by splitting the video into its individual frames or images, and then feeding it into the network. Before, an ANN would read in a frame, and produce a guess. It would then read in the next frame, unaware of the contents of the previous frame, and produce a guess. The ANN would not be able to interpret the difference in each frame, as it would see the frames as two, unique and distinct images with no relation. An RNN will read in the first frame and produce a guess similar to the ANN. It would then pass this guess back to the start of the network. The next frame is passed into the network and paired with the guess from the first frame. The frame and previous guess pass through the network and produce a guess. Now that the network has context of the previous frame, it can guess that something in the frame has moved. It could then pass this guess back to the start of the network again for the next frame.

We can see that RNNs can interpret a specific sequence of data called a time-

series. ANNs or CNNs are good at processing a single moment of data, while RNNs can interpret data over time. This specific quality of RNNs will be useful as we will require previous context when trying to predict tipping points.

RNNs are becoming more popular in modern day applications. It is actively being used today to assist in financial or weather forecasting [27, 28] and in natural language processing (NLP) [29], although the latter requires a slight modification to RNNs called long-short term memory (LSTM).

3.3.1 Long-Short Term Memory

Long-short term memory networks are a variation of RNNs that use LSTM layers within the network. An RNN contains only an immediate context, storing or using data from a single previous moment of time. LSTMs offer the ability to store data or have a 'memory' of data previously passed through the network beyond the previous step. This implies that past inputs will alter the outcome of far future outputs, a necessary step in forecasting.

We will be using LSTMs for forecasting variance and autocorrelation. From this forecasting, we will use a CNN to predict whether a tipping point is imminent or not.

Chapter 4

Methods

For the generation of data and machine learning, all code was written in Python programming language, version 3.9.7 [30]. Tensorflow 2.10.0 was chosen for its machine learning library [31].

4.1 Generating Tipping Points

We began by first attempting to replicate the results of Bury et al. [7] by following the methods dictated in the article. Initial results proved promising, but complications arose when generating 2-d ODEs with stable end solutions. Using randomly generated models, we would obtain on average 37 stable models per 1,000 generated models (distribution seen in Fig. A.1). Once a stable model was found, it needed to be examined for any bifurcations (such as saddle-node or Hopf) using AUTO-07P. AUTO-07P is a bifurcation classifying software widely used in the field of bifurcation theory and mathematical modelling. It proved arduous to set up, and after multiple attempts, it was decided to avoid using the software and the scope of the project was diverted to instead focus on saddle-node bifurcations in 1-d ODEs.

Initially, we began with randomly generating the models. Similar in process to generating 2-d ODEs, we obtained around 17 stable models per 1,000 generated models. Since the system is significantly simpler than a 2-d ODE, an analytical

solution can be soundly achieved. As discussed in Section 2.2, we can generate ODEs which will contain a tipping point, and ODEs which do not. We can then randomly generate parameters which fit the desired outcome at will.

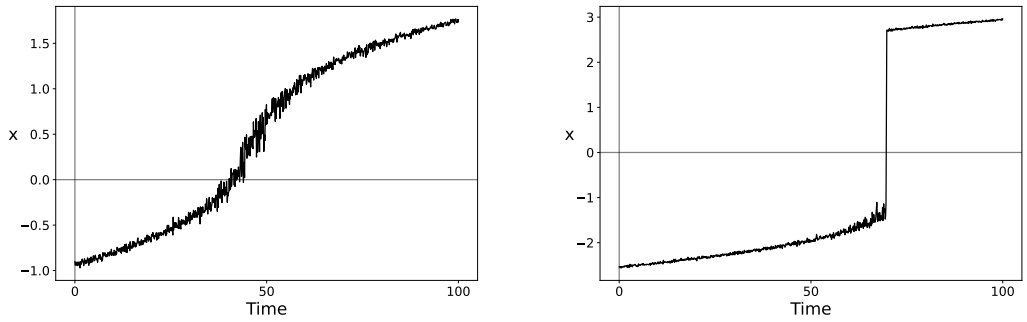
The process of generating data was done by randomly generating parameters a, c where $a < 0$ as per (2.18). a is chosen from $\mathbb{U}(-1, 0)$. We choose c either from $\mathbb{U}(-1, 1)$ if we are seeking a fold, or $\mathbb{U}(-1, 0)$ if we are not. If we are seeking a fold, and $c > 0$, b is chosen from $\mathbb{U}(-1, 1)$. If $c < 0$, we calculate (2.16), randomly add noise from $\mathbb{U}(1, 2)$, and then randomly decide whether it is positive or negative. If we are not seeking a fold, b is chosen from $\mathbb{U}(-\sqrt{3ac}, \sqrt{3ac})$. The parameters are scaled by a factor of three to encourage a wide variety of systems. We set $r = -7, \ldots, 10$ with $M = 1,000$ increments. This range was chosen after multiple test runs, proving to be effective at containing a saddle node bifurcation or tipping point well within its range for a majority of systems.

Once a, b, c, r are chosen, we check if the system will undergo a saddle-node bifurcation for all values of r. This is achieved by checking if r^*_{min} is within than the range of r values. If it is, then the local minima will be lifted above the x-axis at some point, indicating a saddle-node bifurcation or tipping point may occur (could be prolonged by noise, causing it to occur outside the range of values).

Once the tipping point has been checked for, we set $x_0 = x_{i-1} = 0$ and we begin to solve the ODE at each increment of r. We solve the ODE using the Euler-Maruyama method, sampling from $\mathbb{U}(-1, 1)$ each iteration as the Wiener process. The random walks are scaled using σ. We have chosen $\sigma = 0.06$ as we believe this adds a sufficient amount of noise before systems become overly incoherent. We solve over $N = 200$ time steps for 1 time unit ($\Delta t = 0.005$) and take the average of the last 10 time steps as x_i. This will form the solution of the ODE. If the ODE fails or has an integer overflow before we reach 10 steps, we take the average of the previous steps and begin again. We let $x_{i-1} = x_i$ for the next r increment, and the loop begins anew.

After this loop has iterated through each value r, we have one full solution

to a randomly generated ODE. We select another batch of randomly generated parameters and begin again. This loop is repeated $P = 5,000$ times, meaning we have collected the solutions of 5,000 randomly generated ODEs each 1,001 values in length, with the first value of each solution indicating whether a tipping point has occurred with a 0 or 1. We can choose parameters such that we always have a tipping point (within reason, as we may want the tipping point to happen well within the range of r values). Doing this, we generate 2,317 ODE solutions with tipping points, and 2,683 without. This process is repeated for $\sigma = 0.03$ to obtain a wider a variety of systems. This gives us a total of 10,000 solutions with 4,621 tipping solutions, and 5,379 non-tipping solutions. Fig. 4.1 depicts a sample non-tipping solution and a sample tipping solution.



(a) (b)

Figure 4.1: Sample solution of (a) a non-tipping model, (b) a tipping model

4.2 Machine Learning

The next step is processing the data for machine learning. We obtain the variance and autocorrelation of each ODE using the moving window method discussed in Section 2.3 with a window size of 100 points for variance, 99 for AC. AC is one less than variance because with lag-1 AC the window overlap will be 98 points, with one point unobserved by the other's window resulting in the same 100 values being manipulated. The result is two sequences from each ODE, seen in Fig. 4.2.

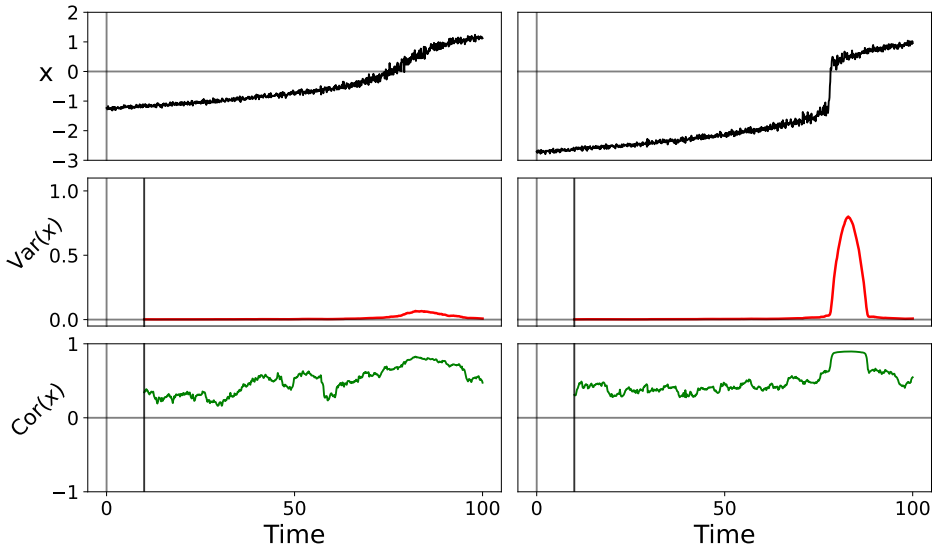


Figure 4.2: Sample break down of (L) non-tipping solution, (R) tipping solution.

For the task of predicting tipping points, it is unproductive to have the entire sequence of variance and AC. In the field, we will have only the data leading up to the tipping point, so we must trim the dataset. This is done using a harsh cut off point. We take the position of the highest value in variance, and take 135 points before it. If there isn't 135 points, the solution is discarded. We then remove 35 values before the peak, leaving us with a string of 100 continuous values. The result of this process is seen in Fig. 4.3. For now, we are taking the entire trimmed set despite the clear sharp rise and clear indication of a tipping point beginning to occur. Later we will use the sharp rise as the training goal for the LSTM.

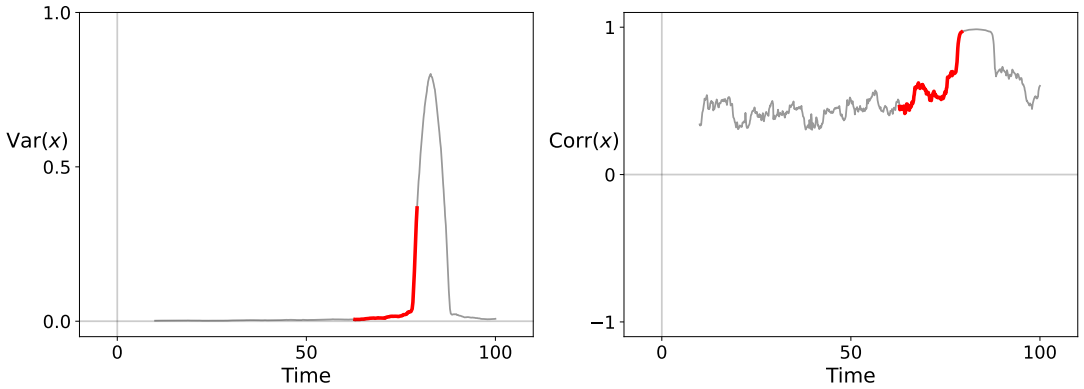


Figure 4.3: Example of taking only the most necessary components on the variance and autocorrelation.

4.2.1 Feature Scaling

During training, large outlier values in the data may cause the network to underfit or over-weigh some of the weights. The dataset currently consists of two inputs: variance and AC. AC has a range of values between $[-1, 1]$, while variance is any positive real number. The distribution of the max values of each sequence of variance is seen in Fig. 4.4. Here it is scaled by dividing the entire sequence of max values by the largest value in that sequence (divided by around $\sim$700 in this case). We can see that the distribution is not normal, as extreme outliers are skewing the data heavily. Ideally, the dataset would be normally distributed, with little to no outliers within the dataset. We can achieve this by feature scaling the dataset.

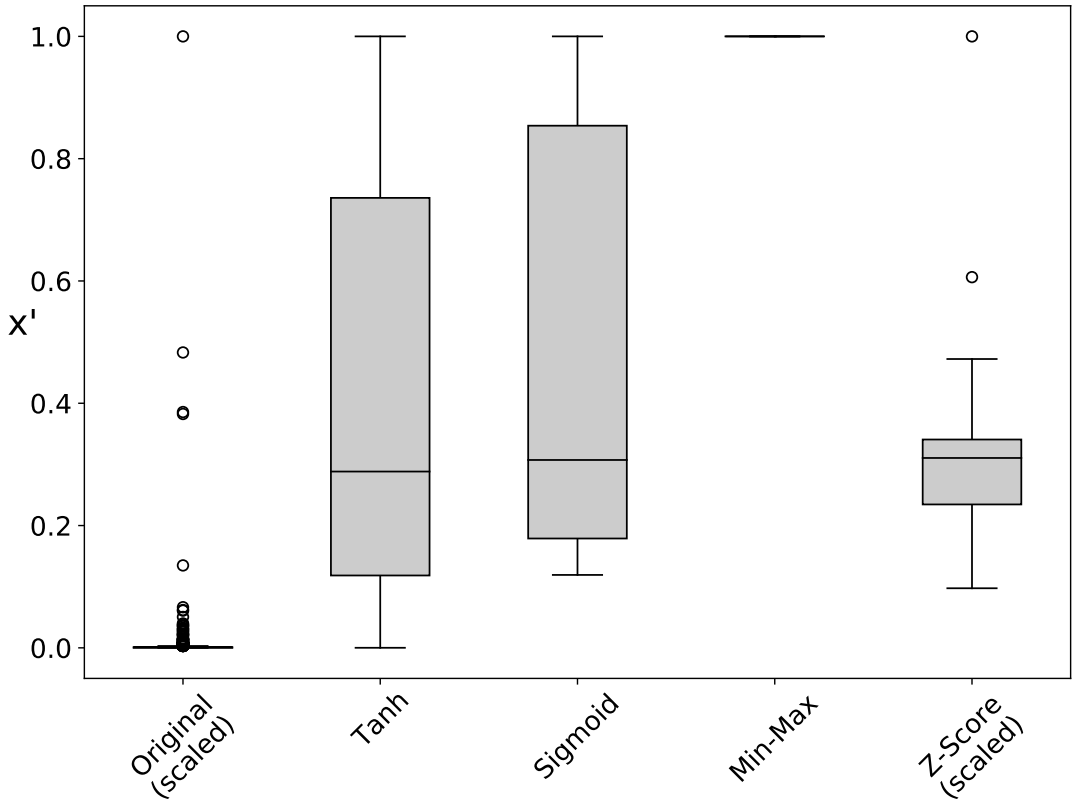


Figure 4.4: Distributions of max values in variance. From left to right; non-normalised data scaled by max value (by a factor of ∼700), tanh mapping, logistic (sigmoid) mapping, min-max scaling, and Z-score normalisation (also scaled by a factor of ∼12). Full distributions shown in A.2.

We will be using normalisation to scale the data. This can be achieved using many different methods and formulae depending on what task is being analysed. For this task, we wish to correct the distribution and reduce the value range as much as possible. We will consider x' to be the normalised vector of values x. The first method we will use is a tanh mapping method. The tanh function is given by

$$x' = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{4.1}$$

When we pass the variance through a tanh function, it will result in some value between $(0, 1)$, as variance is always positive. It will have a skew towards lower values. This provides a better distribution, but many of the larger values are now localised around 1. A similar situation occurs with a logistic mapping. The

logistic function is given by

$$x' = \frac{1}{1 + e^{-k(x-x_0)}}. \tag{4.2}$$

where k is a scaling factor and x_0 is the position of the halfway point between $(0, 1)$. The output of the logistic function is a value in the range of $(0, 1)$. The distribution is tighter, but again many values are mapped close to $+1$. The last two methods we will try are the min-max and Z-score methods. Min-max method is defined by

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{4.3}$$

where $\min(x)$ and $\max(x)$ are the minimum and maximum values of the vector x. Min-max scales the input between $[0, 1]$. Max values will be mapped to 1 and minimum values will be mapped to 0. The result of this is that large peak values will cause lower values in the dataset to diminish quite closely to 0. The magnitude of the peak would be conserved within the vector. The last method we will use is the Z-score method of normalisation. The Z-score method is defined by

$$x' = \frac{x - \bar{x}}{\sigma}, \tag{4.4}$$

where $\bar{x}$ is the mean of the vector, and σ is the standard deviation of the vector. The Z-score's main goal is to reduce the mean of a dataset to 0, and to have unit variance. If a value in the vector now has a value $+1$, that value is greater than 68% of the values in the vector, i.e. it is one standard deviation greater than the rest of the values in the vector.

The main downfall of the previous two methods is the irreversible nature of each method. If given just the result of the Z-score method, we cannot return to the original vector without knowing the previous mean and standard deviation. Z-score normalisation offers the best distribution. It is scaled to fit within the

figure (by a value of 12).

After running tests to check performance of each normalisation, we find that each method of normalisation offers roughly the same performance, with slight better performance from Z-score normalisation. We choose Z-score normalisation for feature scaling.

4.2.2 Network Construction

We first design a simple ANN that can determine whether a tipping point has occurred or not from variance and autocorrelation. We split the dataset into 70 : 20 : 10; training, validation and testing respectively. We also construct a CNN to test its ability. These networks have high accuracy when asked to predict whether a tipping point occurred or not, indicating that tipping points can be picked up by these networks.

We now want to create a network that can forecast future values. We will achieve this using an LSTM. When constructing an LSTM, the data input must be read in as a time-series. We split the dataset into 'before' tipping and 'during' tipping. The network will be trained on the 'before' data and asked to predict the 'during' tipping values. The network's predictions will be tested on the true 'during' tipping values.

The breakdown of the data is as follows: We have taken 200 points of continuous data. On the 200th value, we reach the peak in variance. We have trimmed it down such that we have from the 65th value to the 165th value. For the LSTM, we are providing it the 65th to the 150th value, and asking it to forecast the 151st to the 165th value. A visualisation using a sample variance is shown in Fig. 4.5.

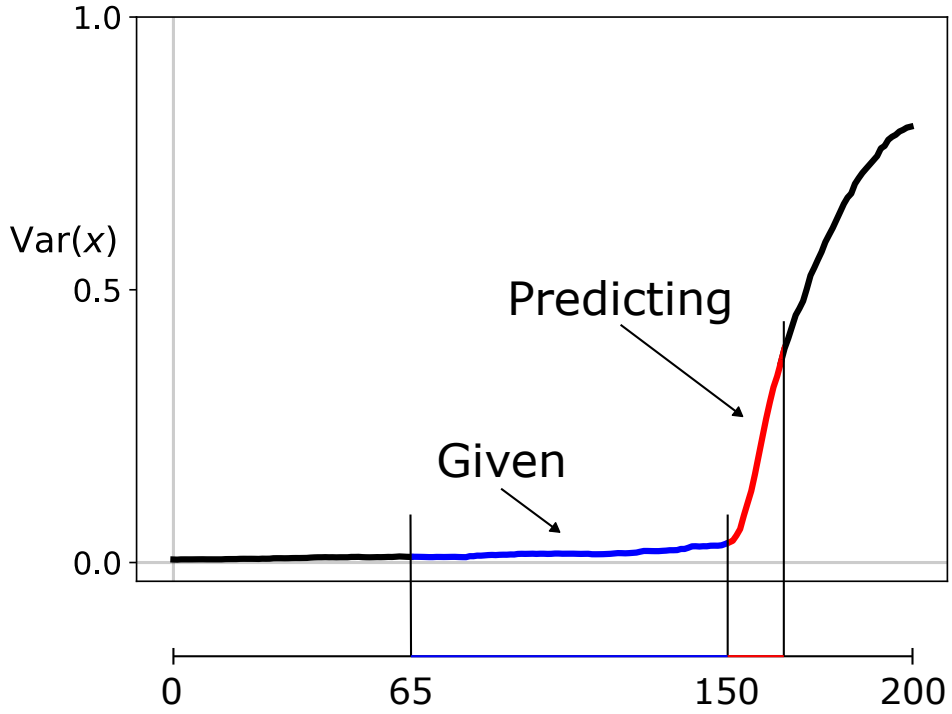


Figure 4.5: Visual breakdown of a time series for a sample variance.

Next we create a CNN that interprets the predictions of the LSTM. It will predict the possibility of a tipping point occurring in the future (15 time steps in this scenario). The architecture of the LSTM and CNN is detailed in Figs. A.6a, A.6b respectively. Both were trained separately over 100 epochs.

4.3 Generating R-tipping

We start by generating systems of the form discussed in Section 2.2.2. The parameters we choose are $x_0 = 0, \lambda_0 = 0, \mu = 1.5, r = \{1.51, \ldots, 1.55\}, \sigma = 0.005$ up to 100 time units. We use the Euler-Maruyama method to solve over $M = 1,000$ time steps.

We generate $P = 1,000$ systems, of which 447 tip. We decide that a system tips if x becomes greater than zero at any point in the numerical solution.

We need to repeat the process in which we manipulated the data before we

trained the networks earlier, to ensure the data is in the standard form that the network can interpret. We start by using the window method to calculate the variance and AC of the systems, trimming the variance and AC, and normalising the variance using the normalisation method discussed earlier in Section 4.2.1. After this process is completed, we have 649 eligible systems for analysis, 96 of which have a tipping point. We convert each system into a time series and pass them through the LSTM-CNN.

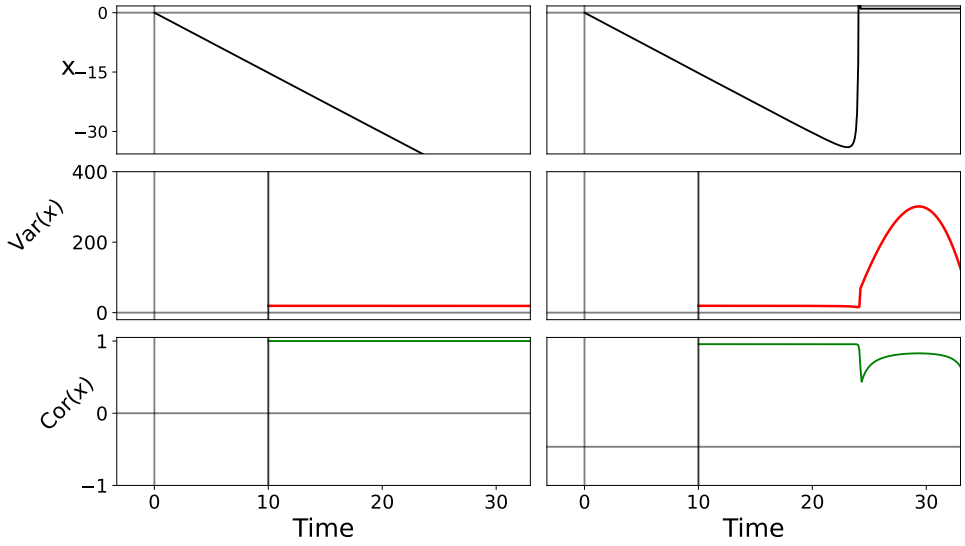


Figure 4.6: Sample break down of (L) non-tipping solution, (R) tipping solution.

4.4 Thermoacoustic Data

Finally, we will be testing the network on real life data obtained from Hennekam et al. [32]. This is an X-ray spectroscopy of a sample of earth taken from the surface of the Mediterranean Ocean. The specific data set we are looking at is '64PE406-E1_calibratedXRF.tab'. We will be analysing the column 'Mo [mg/kg]' which measures the parts per million (ppm) molybdenum detected within the sample, seen in Fig. 4.7. We start by calculating a moving average on the data to reduce noise (window size of 100). We will be parsing this data by taking peak values obtained using Scipy. From each peak we take 200 previous

values and split it up as done before. The data is transformed into a time series, and passed into the LSTM-CNN.

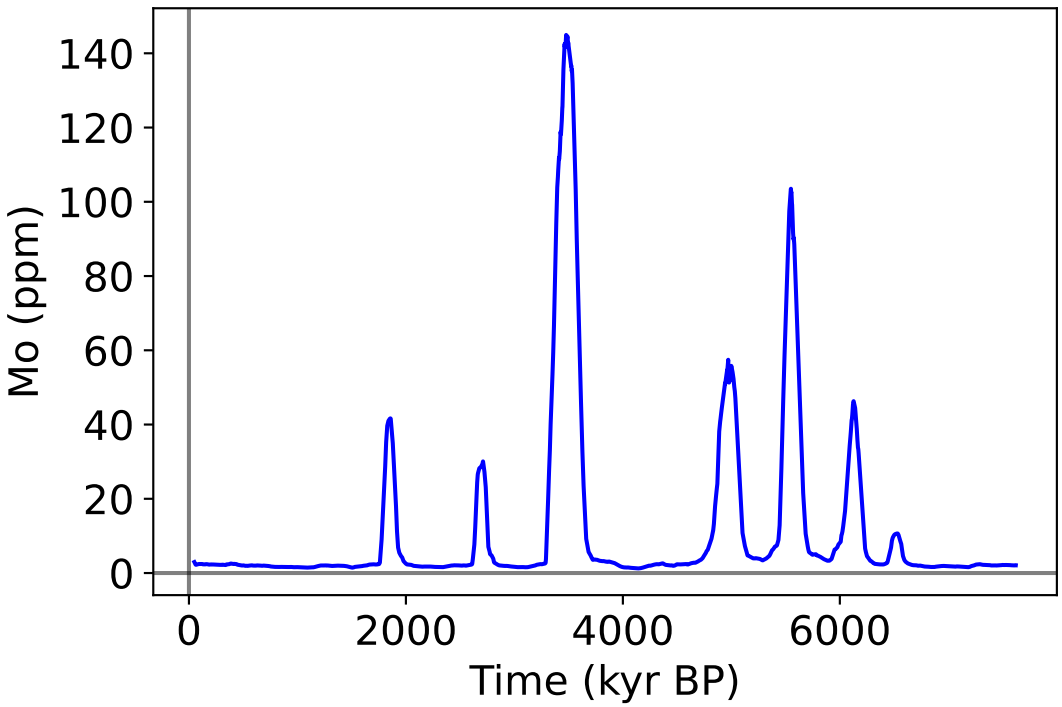


Figure 4.7: Readout from 'Mo [mg/kg]' column from Hennekam et al. [32]. A moving window of size 100 was passed over the data to remedy the amount of noise prevalent in the data.

Chapter 5

Results

From Fig. 5.1 we see the receiver operating characteristic (ROC) curves for the LSTM-CNN on the test set for B-/N-tipping. ROC curves are a method of visualising the performance of a classifier. We measure the performance of a classifier by the area under the ROC curve (AUC). A perfect classifier has AUC = 1, while a classifier which has AUC ≤ 0.5 is just as good as randomly guessing (the AUC for the dashed line is 0.5). It appears that the LSTM-CNN is a sufficient classifier for B-/N-tipping scenarios. The AUC for the test set is >65% when predicting values up to 70 steps before tipping, a satisfactorily high performance. The AUC increases as we provide the LSTM-CNN with values closer to the actual tipping point. We see that given values up to 50 steps before the tipping point, AUC > 0.9, whereas when provided with values up to 30 steps before the tipping point, AUC drops dramatically. This could be a result from how the network was trained. It was only trained using data leading up to the tipping point, not data during the tipping point. Specifically, it was trained on data leading up to 50 points before tipping. We would expect this category of prediction to be the strongest.

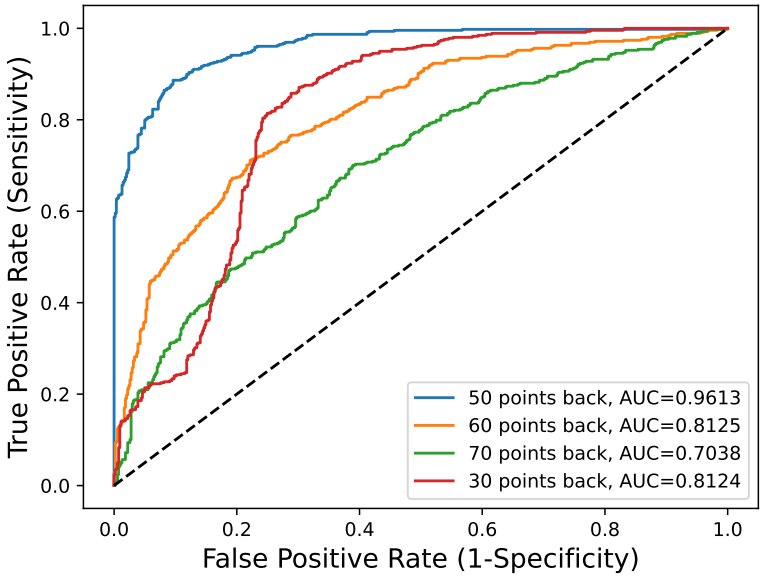


Figure 5.1: ROC curve of LSTM-CNN on B-/N-tipping test samples. 'k points back' refer to how close the points the system is exposed to is from the peak in variance.

The ROC curves for the LSTM-CNN on R-tipping are shown in Fig. 5.2. The network has not seen any data from R-tipping before this test set was shown to it. We would expect it to struggle to classify, and this is clearly seen when given data 70 steps before tipping point. Since AUC is significantly lower than random guesses, this may signify that the network has difficulty classifying one of the groups. Recalling Fig. 2.13, taking up to 70 steps before tipping may provide a variance that is roughly constant, and a stable AC. These conditions would indicate no tipping will occur in the system. This would cause the network to consistently and incorrectly classify the system as a non-tipping one, resulting in worse-than-guessing scores.

When giving the network values up to 60 steps and 50 steps back, we see a considerable increase in AUC for the network. Again, since the network was trained on data up to 50 steps before tipping, we expect the greatest performance to be here. Most important is the performance of the network on this dataset. Despite never been exposed to the variance or AC of a system which undergoes R-tipping, the network has an AUC > 0.9 for 50 steps back. This would support

the claim that R-tipping contains similar EWS to B- and N-tipping during saddle-node bifurcations [8].

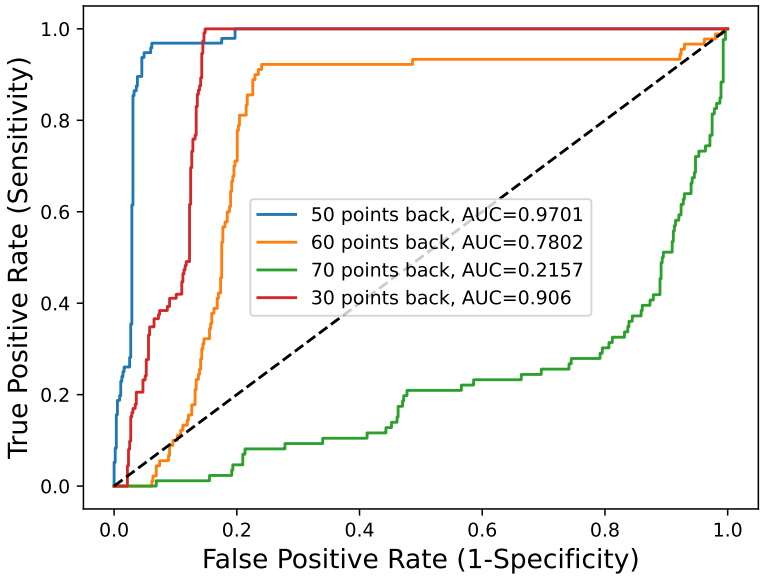
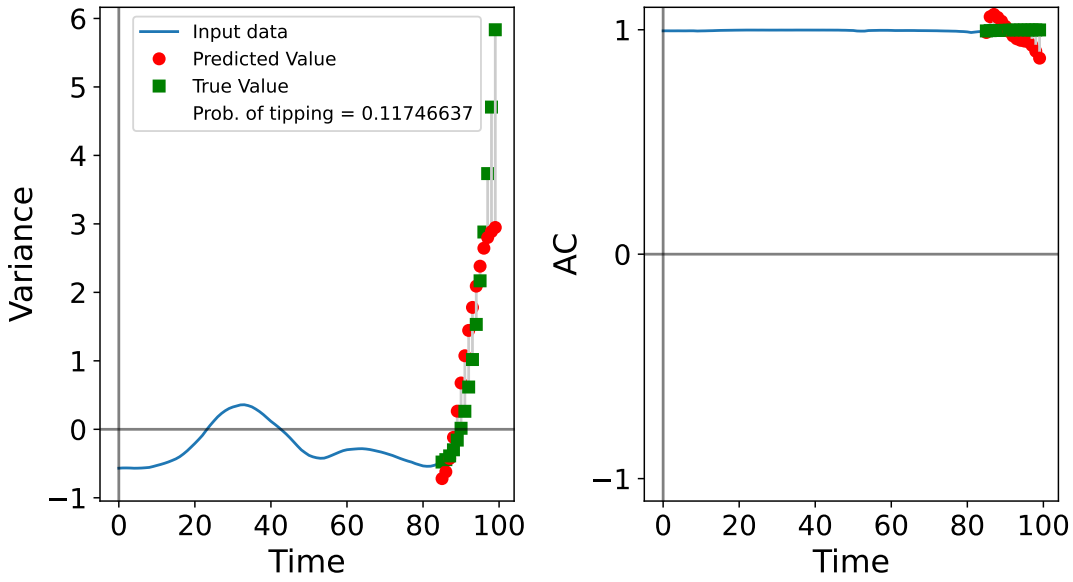


Figure 5.2: ROC curve of LSTM-CNN on R-tipping test samples.

The ability to forecast tipping has proven difficult for the LSTM. The network can forecast reasonably well in some cases as seen in Appendix Figs. A.3a, A.3b, but can struggle to forecast particularly noisy systems as seen in Fig. A.3c. Interestingly, the same discrepancies can be found for the R-tipping systems, as seen in Fig. A.4. The forecasting from the LSTM is not quite dependable. We measure the fit of the forecasting by the mean squared error (MSE) of the predictions on the test set. The error is the displacement of the prediction from the true value. The MSE from forecasting on the test set for B-/N-tipping is 0.0742, a low value as it was trained on data similar to the test set. This MSE is significantly larger for R-tipping at 0.516. We would expect forecasting to be of poor quality on R-tipping systems as the network has not been exposed to tipping points of this nature. The predictions from the CNN trained on those outputs seem reliable despite the MSE from the LSTM forecasting.

Finally, the MSE on the molybdenum data was 0.443. This was achieved from splitting up the data into a set of peaks, and asking the network to predict

on the peaks. It appears to forecast relatively well (see Fig. 5.3) compared to R-tipping, indicating that the tipping experienced within this dataset more closely resembles that produced from B-tipping. Prediction on the single peak example is poor, indicating a failure in the network. Continuous prediction output on this data does not perform so well. The network predicts many tipping points despite relatively flat readouts.



(a) (b)

Figure 5.3: Forecasting on a sample peak from 'Mo [mg/kg]'

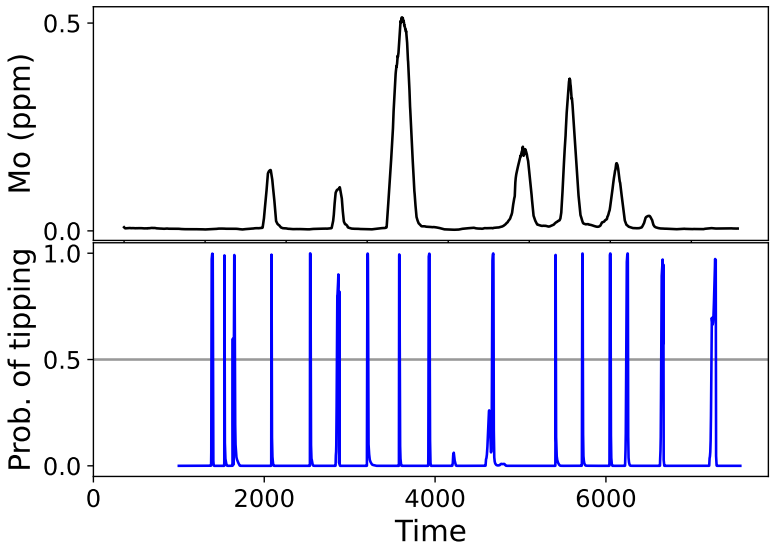


Figure 5.4: Example of real time output of (bottom) LSTM-CNN predictions on (top) 'Mo [mg/kg].'

5.1 Discussion and Conclusion

It is apparent that the same early warning signals which appear from B-tipping and N-tipping seem to appear for R-tipping. This aligns with the analysis achieved by Ritchie et al. [8]. A network which can detect the probability of tipping points in the future could prove useful, as it would pick up on all forms of tipping, not just the form of tipping it was trained on. This would limit the chance of an unexpected form of tipping to occur should the network be used in practice.

The scope of this project was scaled down to consider only models which tipped by saddle-node bifurcation. Future analysis into other forms of bifurcations such as Hopf or transcritical bifurcations would be necessary to further support the idea of some form of universal tipping point recognition network.

Further work could be done to supply the network with a more sufficient dataset. We provided the network with data going in one direction (increasing r), but systems can propagate in any direction. One could argue that as variance is always positive, that it would remain unaffected by this change, and that we

would need only to obtain the absolute value of AC to account for both scenarios.

Similarly, the design of the network is a topic which could use further scrutiny. The network we designed was relatively shallow, only using a minimal amount of layers each with a small amount of nodes. Layer optimization could be found using grid search methods, but neural network design is a topic infamously known for not having one standard recipe [33]. It is worth noting that the CNN was trained on only the output of the LSTM, not the concatenated input and output. Higher accuracy and better training may be achieved in such a scenario. The CNN was also trained on the output of the trained LSTM. One could argue that training the network all at once (LSTM in tandem with CNN) could produce better accuracy and less loss, but the prominence of reservoir computing may indicate that performance would not be hindered by the individual training of networks.

Finally, for a network to be beneficial to in-field work, it would be necessary to be functional on real-life data. An ideal network could produce real time outputs quicker than the system can produce an updated output. This process is inherently computationally inexpensive, but for deeper networks it may become problematic.

References

[1] M. Scheffer *et al.*, "Early-warning signals for critical transitions," *Nature*, vol. 461, no. 7260, pp. 53–59, Sep. 2009, ISSN: 1476-4687. DOI: 10.1038/nature08227.

[2] W. Yan, R. Woodard, and D. Sornette, "Diagnosis and prediction of tipping points in financial markets: Crashes and rebounds," *Physics Procedia*, vol. 3, no. 5, pp. 1641–1657, Aug. 2010, ISSN: 1875-3892. DOI: 10.1016/j.phpro.2010.07.004.

[3] M. I. Maturana *et al.*, "Critical slowing down as a biomarker for seizure susceptibility," *Nat. Commun.*, vol. 11, 2020. DOI: 10.1038/s41467-020-15908-3.

[4] V. de Leemput Ingrid A. *et al.*, "Critical slowing down as early warning for the onset and termination of depression," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 111, no. 1, pp. 87–92, Jan. 2014, ISSN: 0027-8424. DOI: 10.1073/pnas.1312114110.

[5] G. E. P. Box, "Robustness in the Strategy of Scientific Model Building," in *Robustness in Statistics*, Cambridge, MA, USA: Academic Press, Jan. 1979, pp. 201–236, ISBN: 978-0-12-438150-6. DOI: 10.1016/B978-0-12-438150-6.50018-2.

[6] V. Dakos, M. Scheffer, E. H. van Nes, V. Brovkin, V. Petoukhov, and H. Held, "Slowing down as an early warning signal for abrupt climate change," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 105, no. 38, pp. 14 308–14 312, Sep. 2008, ISSN: 0027-8424. DOI: 10.1073/pnas.0802430105.

[7] M. Bury Thomas *et al.*, "Deep learning for early warning signals of tipping points," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 118, no. 39, e2106140118, Sep. 2021, ISSN: 0027-8424. DOI: 10.1073/pnas.2106140118.

[8] P. Ritchie and J. Sieber, "Early-warning indicators for rate-induced tipping," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 26, no. 9, p. 093 116, Sep. 2016, ISSN: 1054-1500. DOI: 10.1063/1.4963012.

[9] P. Holmes, "Poincaré, celestial mechanics, dynamical-systems theory and "chaos"," *Phys. Rep.*, vol. 193, no. 3, pp. 137–163, Sep. 1990, ISSN: 0370-1573. DOI: 10.1016/0370-1573(90)90012-Q.

[10] A. Dutta, "COVID-19 waves: variant dynamics and control," *Sci. Rep.*, vol. 12, no. 9332, pp. 1–9, Jun. 2022, ISSN: 2045-2322. DOI: 10.1038/s41598-022-13371-2.

[11] P. N. V. Tu, *Dynamical Systems*. Berlin, Germany: Springer, ISBN: 978-3-642-78793-5. [Online]. Available: https://link.springer.com/book/10.1007/978-3-642-78793-5.

[12] S. H. Strogatz, *Nonlinear Dynamics And Chaos: With Applications To Physics, Biology, Chemistry, And Engineering*. Boca Raton, FL, USA: CRC Press, Dec. 2000, ISBN: 978-0-73820453-6. [Online]. Available: https://www.amazon.co.uk/Nonlinear-Dynamics-Chaos-Applications-Nonlinearity/dp/0738204536.

[13] P. Ashwin, S. Wieczorek, R. Vitolo, and P. Cox, "Tipping points in open systems: bifurcation, noise-induced and rate-dependent examples in the climate system," *Philos. Trans. Royal Soc. A*, vol. 370, no. 1962, pp. 1166–1184, Mar. 2012, ISSN: 1471-2962. DOI: 10.1098/rsta.2011.0306.

[14] J. Jiang *et al.*, "Predicting tipping points in mutualistic networks through dimension reduction," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 115, no. 4, E639–E647, Jan. 2018, ISSN: 0027-8424. DOI: 10.1073/pnas.1714958115.

[15] R. A. Bentley *et al.*, "Social tipping points and Earth systems dynamics," *Front. Environ. Sci.*, vol. 0, 2014, ISSN: 2296-665X. DOI: 10.3389/fenvs.2014.00035.

[16] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 210–229, Jul. 1959, ISSN: 0018-8646. DOI: 10.1147/rd.33.0210.

[17] A. L. Fradkov, "Early History of Machine Learning," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 1385–1390, Jan. 2020, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2020.12.1888.

[18] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943, ISSN: 1522-9602. DOI: 10.1007/BF02478259.

[19] M. A. Nielsen, "Neural Networks and Deep Learning," *Determination Press*, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com.

[20] S. Lague, *How to Create a Neural Network (and Train it to Identify Doodles)*, [Online; accessed 29. Aug. 2022], Aug. 2022. [Online]. Available: https://www.youtube.com/watch?v=hfMk-kjRv4c.

[21] 3Blue1Brown, *But what is a neural network? | Chapter 1, Deep learning*, [Online; accessed 29. Aug. 2022], Oct. 2017. [Online]. Available: https://www.youtube.com/watch?v=aircAruvnKk.

[22] *NN SVG*, [Online; accessed 6. Sep. 2022], Feb. 2022. [Online]. Available: http://alexlenail.me/NN-SVG.

[23] J. Wise, "How Much Data Is Created Every Day in [year]? [NEW Stats]," *EarthWeb*, Jul. 2022. [Online]. Available: https://earthweb.com/how-much-data-is-created-every-day/#Key_Data_Creation_Statistics_2022.

[24] S. Miller, "Machine Learning, Ethics and Law," *1.*, vol. 23, May 2019, ISSN: 1449-8618. DOI: 10.3127/ajis.v23i0.1893.

[25] N. M. Barbosa and M. Chen, "Rehumanized Crowdsourcing: A Labeling Framework Addressing Bias and Ethics in Machine Learning," in *CHI '19: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA: Association for Computing Machinery, May 2019, pp. 1–12, ISBN: 978-1-45035970-2. DOI: 10.1145/3290605.3300773.

[26] T. Basu, S. Engel-Wolf, and O. Menzer, "The Ethics of Machine Learning in Medical Sciences: Where Do We Stand Today?" *Indian J. Dermatol.*, vol. 65, no. 5, p. 358, Sep. 2020. DOI: 10.4103/ijd.IJD_419_20.

[27] A. K. Rout, P. K. Dash, R. Dash, and R. Bisoi, "Forecasting financial time series using a low complexity recurrent neural network and evolutionary learning approach," *Journal of King Saud University - Computer and Information Sciences*, vol. 29, no. 4, pp. 536–552, Oct. 2017, ISSN: 1319-1578. DOI: 10.1016/j.jksuci.2015.06.002.

[28] A. G. Salman, B. Kanigoro, and Y. Heryadi, "Weather forecasting using deep learning techniques," in *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, IEEE, Oct. 2015, pp. 281–285. DOI: 10.1109/ICACSIS.2015.7415154.

[29] X. Zhang, M. H. Chen, and Y. Qin, "NLP-QA Framework Based on LSTM-RNN," in *2018 2nd International Conference on Data Science and Business Analytics (ICDSBA)*, IEEE, Sep. 2018, pp. 307–311. DOI: 10.1109/ICDSBA.2018.00065.

[30] G. Van Rossum and F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[31] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[32] R. Hennekam, B. Van der Bolt, E. H. Van Nes, G. J. de Lange, M. Scheffer, and G.-J. Reichart, *Calibrated XRF-scanning data (mm resolution) and calibration data (ICP-OES and ICP-MS) for elements Al, Ba, Mo, Ti, and U in Mediterranean cores MS21, MS66, and 64PE406E1*, data set, 2020. DOI: 10.1594/PANGAEA.923197. [Online]. Available: https://doi.org/10.1594/PANGAEA.923197.

[33] P. V. Heusser and Matt, "5 Guidelines for Building a Neural Network Architecture," *InfoWorld*, Jan. 2017. [Online]. Available: https://www.infoworld.com/article/3155052/5-guidelines-for-building-a-neural-network-architecture.html.

Appendix A

Additional Figures

A.1 Convergence distribution

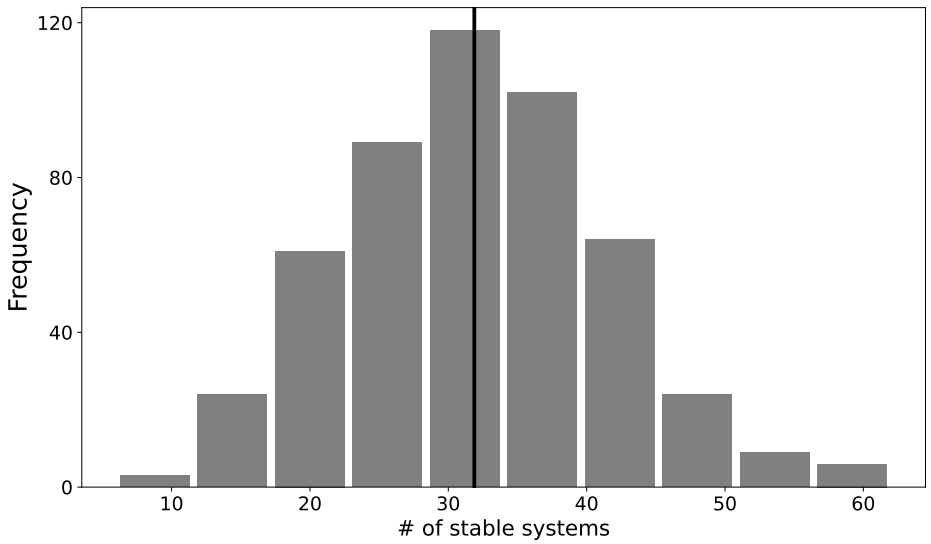


Figure A.1: Distribution of frequency of 2-d ODEs which converged.

A.2 Variance distributions

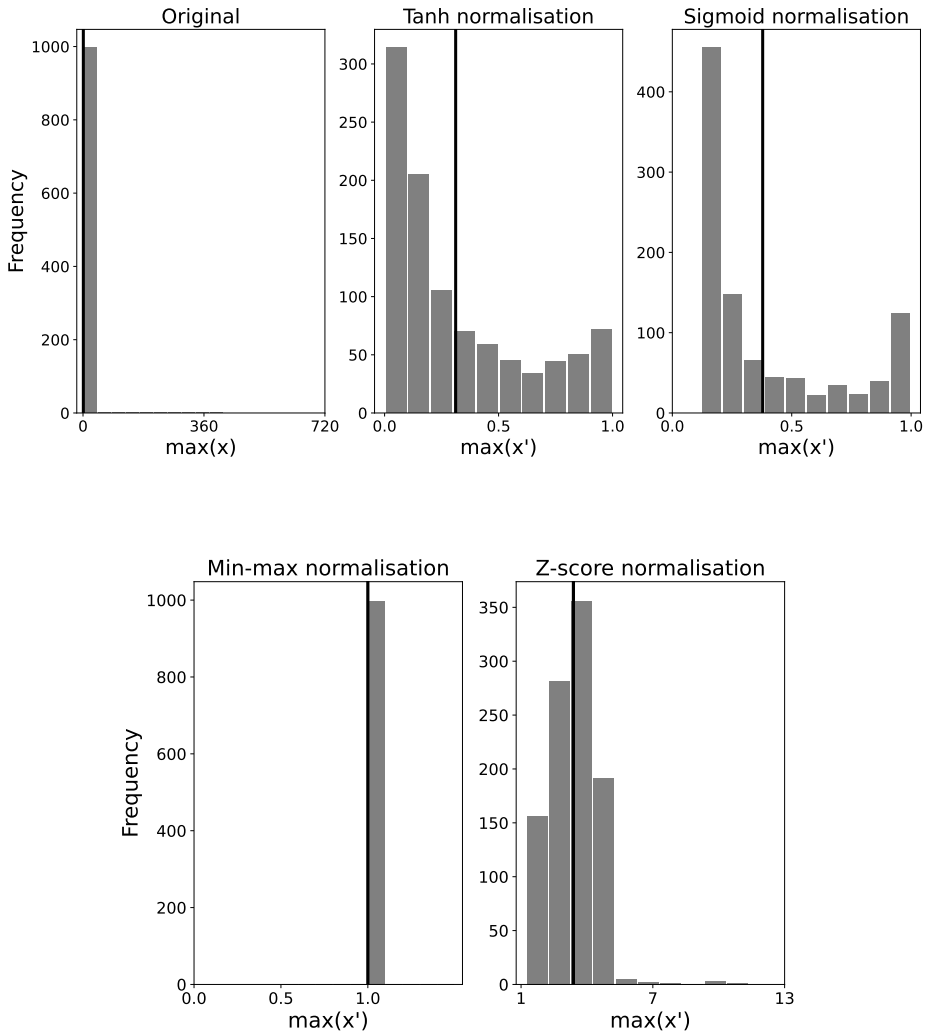
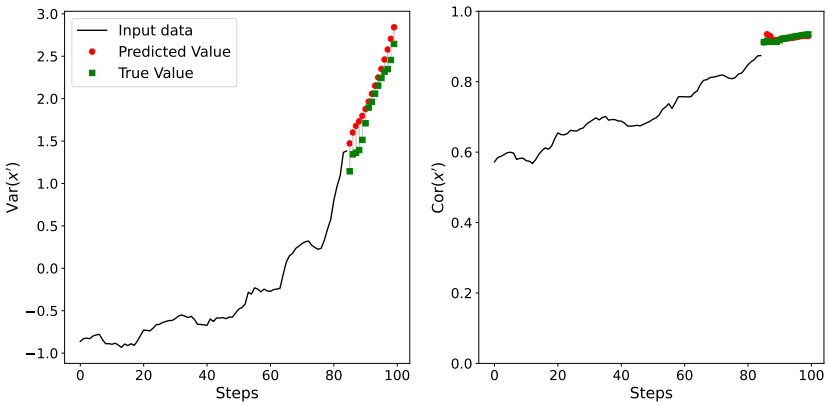
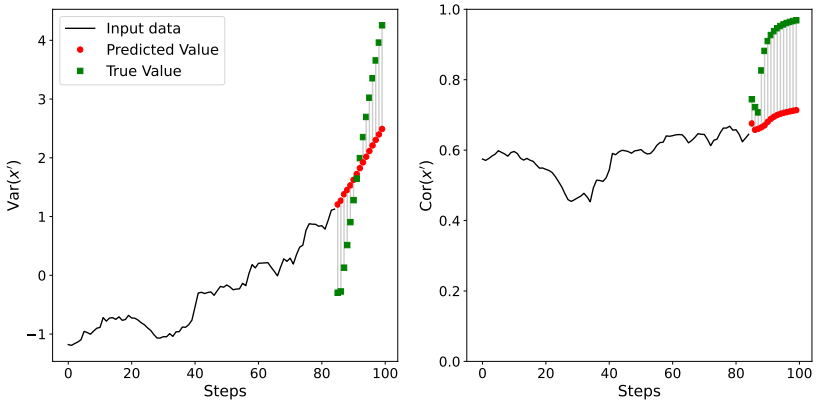


Figure A.2: Distributions of max variance values using different methods of normalisation.

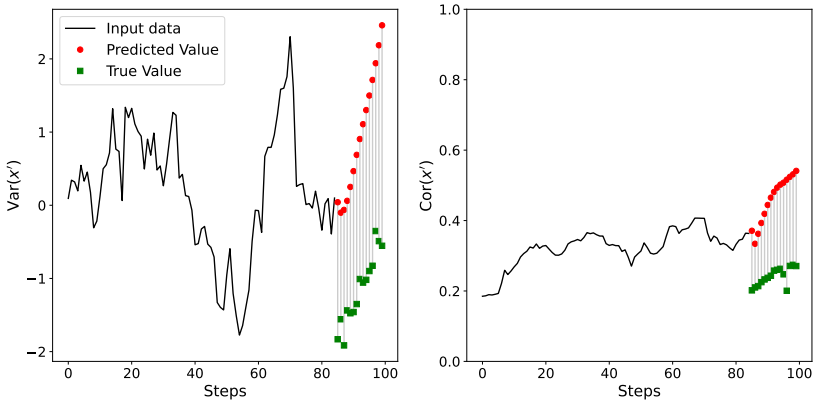
A.3 LSTM Predictions



(a)

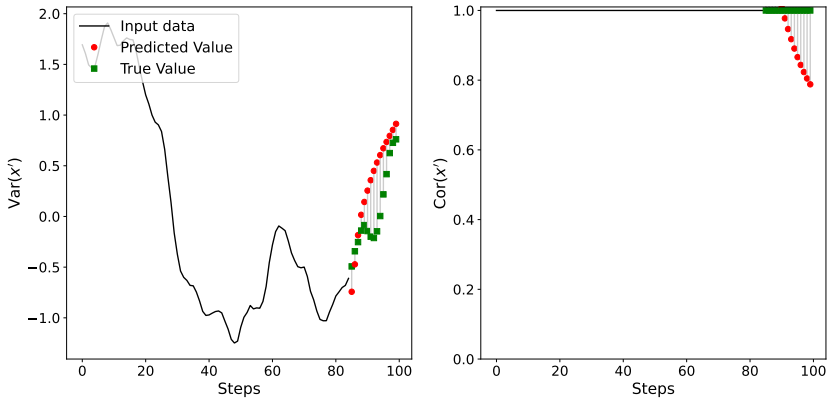


(b)

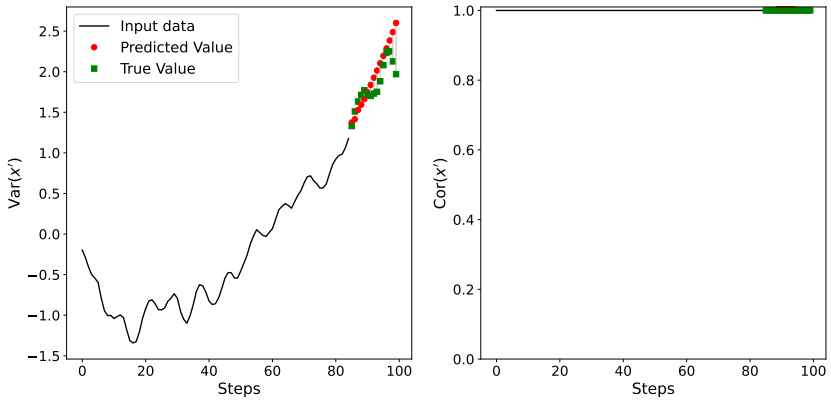


(c)

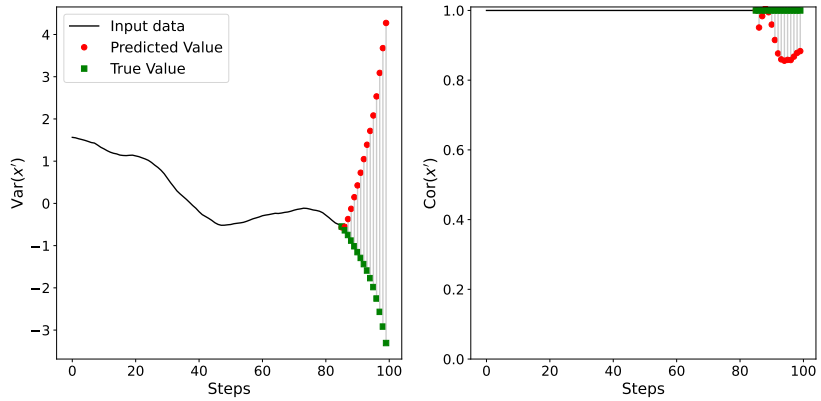
Figure A.3: LSTM forecasting on B-/N-tipping solutions.



(a)



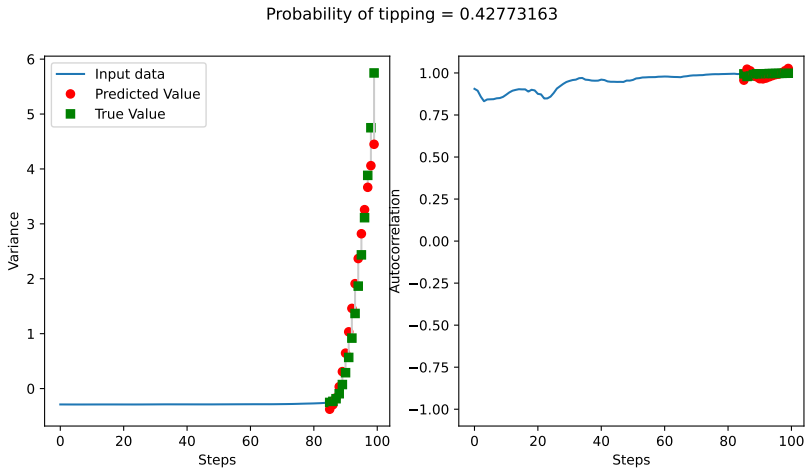
(b)



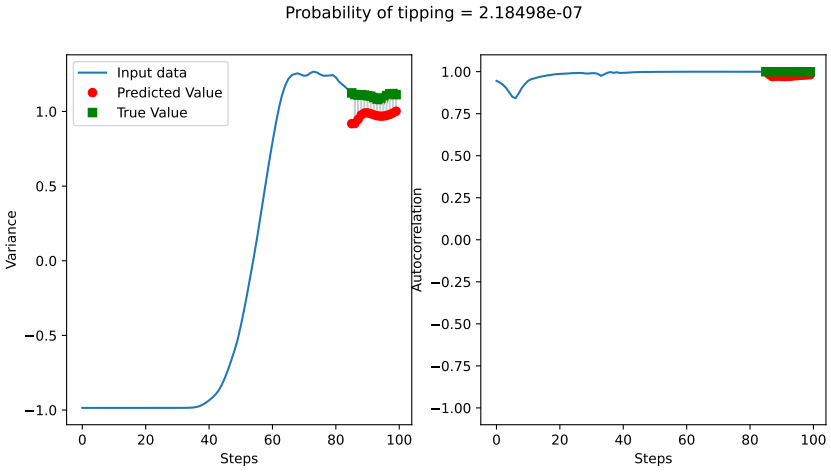
(c)

Figure A.4: LSTM forecasting on R-tipping solutions.

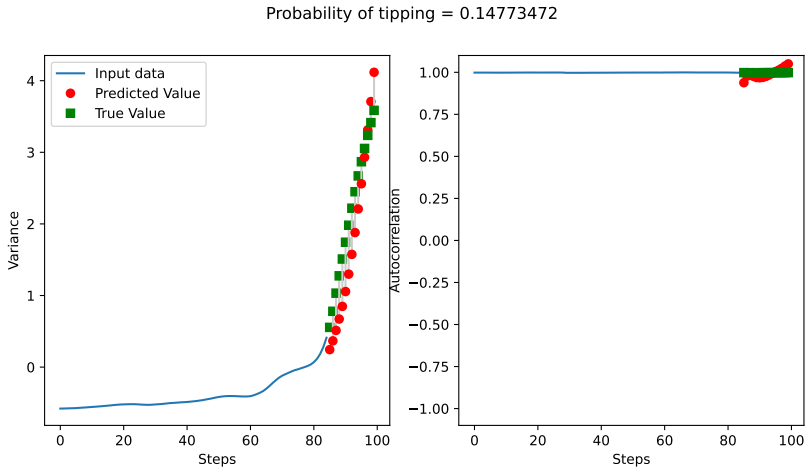
A.4 Molybdenum predictions



(a)



(b)



(c)

Figure A.5: LSTM forecasting on 'Mo (ppm).'

A.5 Model architectures

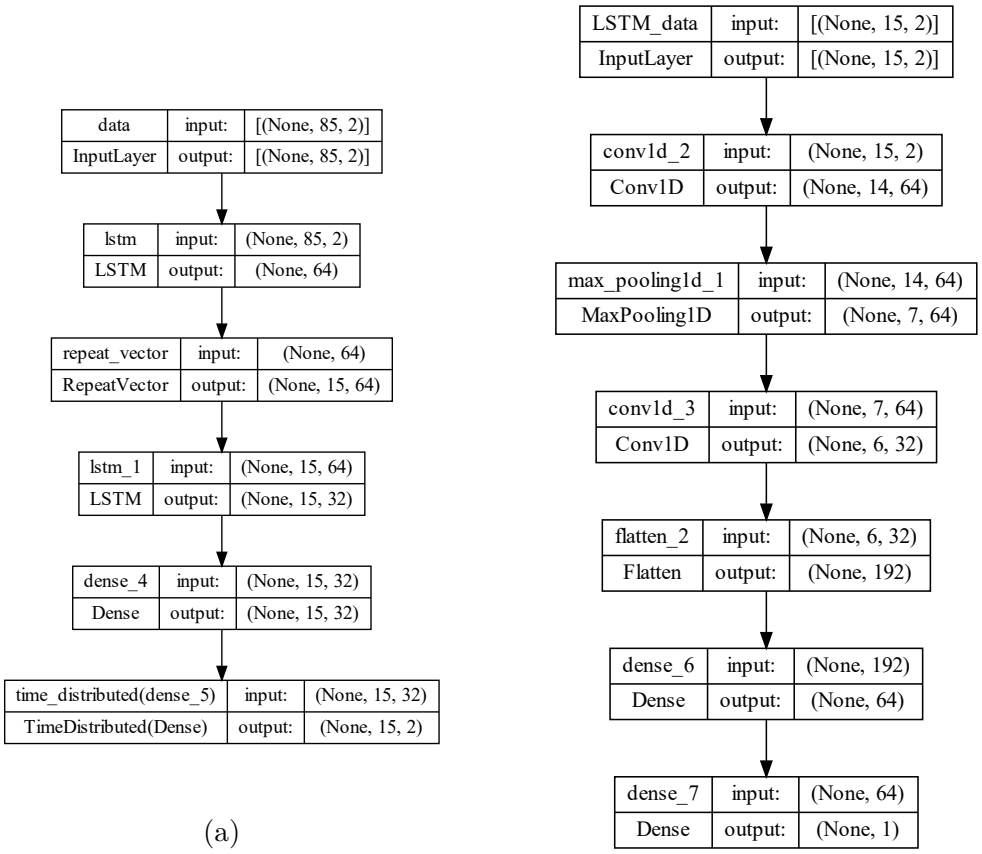


Figure A.6: Architecture of the (a) LSTM, (b) CNN.

Appendix B

Supplementary Code

B.1 GitHub

All code used for this project can be accessed at:

https://github.com/BarryOD4/Thesis