2021

Simulations of hybrid systems using neural networks

MCOMP (HONS) COMPUTER SCIENCE,
SCHOOL OF COMPUTING,
NEWCASTLE UNIVERSITY

ADANNA OBIBUAKU 180251870 | PROJECT SUPERVISOR: SERGIY BOGOMOLOV | WORD COUNT: 16,876

Declaration

"I declare that this dissertation represents my own work except where otherwise stated."

Acknowledgements

A huge thank you to Dr. Sergiy Bogomolov for his guidance which allowed me to achieve this project. Throughout this project he has helped me and provided guidance that made it possible to achieve the aims.

Abstract

Processes that occur in engineering fields and processes that occur in nature can be modelled using a dynamical system. A dynamical system is a system with a function based on an ordinary differential equation (ODE). An ODE describes the evolution (changes) of a variable through time. Numerical simulations are used to approximate the evolution of a variable through time. This project proves that given the required training data; neural networks can be used as an alternative method to approximate the evolution of a variable through time. The training data consists of input-output pairs of samples of state spaces. Each state space is derived from a different initial value. The implemented neural network takes in an initial value, along with a time sequence to predict the corresponding state space. Here, we investigate the neural network's ability to predict a state space that goes beyond the samples in the training data.

Table of Contents

List of Abbreviations

CPS Cyber physical Systems

ODE Ordinary differential equations

Glossary

Cyber Physical Systems	A computer system whose mechanism is controlled by a computer algorithm.
Dynamic systems	A dynamical system is a function which describes the evolution of a variable with respect to time
Execution	A single state in a state space
Hybrid automata	The mathematical notation for a hybrid system
Hybrid Systems	A dynamical system is a system which exhibits both continuous and discrete behaviour
Initial condition/initial state variables	This state is at the beginning of a dynamic system.
Learning parameters	The parameters in the neural network
Neural network	A neural network is a deep learning algorithm composed of neurons.
Numerical Simulation/Simulation	A procedure which approximates the evolution of the variables with a sequence of points in discretized time.
Ordinary differential equations	A differential equation containing one/more functions of one independent variable, including the derivatives of those functions
Run	The evolution of the system states over time.
State	The state of the system in terms of the value of the variable, current time and discrete location.
State space / Sample simulation	A set of possible states of the dynamic system derived from the same initial condition
Test data	Data used to evaluate neural network performance
Training data	Data is used to train a neural network.
Validation data	Data that provides an unbiased evaluation of the neural network while tuning the neural network

1. Introduction

1.1 Background

Neural networks are deep learning algorithms inspired by biological neural networks, whereby they mimic the operations of a human brain to be able to recognise the relationship in a large amount of data. It is used in a variety of different applications such as customer research, speech/image recognition and forecasting. Each application is suited to a certain neural network architecture. In this context, a feed-forward neural network is used to approximate an ordinary differential equation (ODE). An ODE is an equation that relates to the function of an independent variable and the derivative. The derivate represents the rate of change and the differential equation is used to describe the relationship. (Wikipedia, 2021). The specific ODE we are solving is used for dynamical systems. A dynamical system is a system which has a function based on an ODE. The ODE has a time independent variable that relates to a certain point. This allows you to capture specific points corresponding to a specific time. Such examples include the flow of water, pendulum and speed. This project divides dynamical systems into two types:

1. Continuous dynamic equations
2. Hybrid systems.

Continuous dynamic systems only exhibit continuous behaviour represented by ODEs. Hybrid Systems exhibits continuous behaviour and introduces discrete events. In recent years, there has been a lot of intense research in hybrid systems. The is because it is commonly used for engineering systems such as Cyber physical systems (CPS). CPS is a computer system that controls a mechanism using algorithms. Examples of CPS include industrial control systems, autonomous systems and medical monitoring systems. There are two fundamental components, physical components and computer software components. The hybrid system uses its properties to model the CPS. The continuous behaviour is used to describe the physical components while the discrete event models the computational components. For example, in a CPS Collision system, the functionality is designed to detect collisions of objects. The physical component is the movements of objects. In a hybrid system, this property is modelled by the continuous behaviour to describe the distance of objects. Additionally, the computational components detect whether a collision has occurred. In a hybrid system, this property is modelled with discrete behaviour. This can be formalised using a mathematical representation shown below:

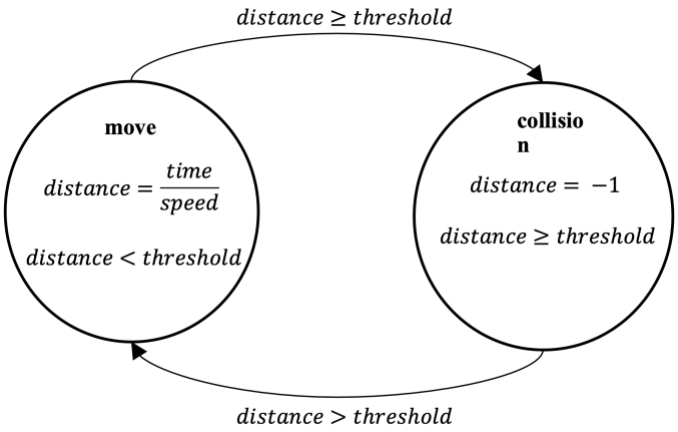


Figure 1: A hybrid automaton representing a collision systems

Hybrid automation is a mathematical representation of hybrid systems. An example is shown in figure 1. Hybrid automation combines mathematical language and a finite state automation. A precise formal definition is shown in figure 2.

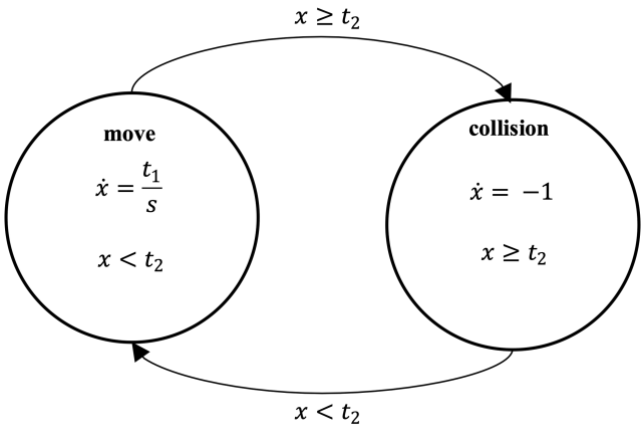


Figure 2: A formal definition of a hybrid automaton representing a collision system

The nodes represent the states: movement and collision. Each state is associated with an ODE. The discrete event, also known as transitions, is denoted with arrows. The OSE describes the rate of movement of an object. A transition occurs when the distance is more or equal to a threshold. Figure 2 demonstrates how a hybrid system represents a collision system CPS. This can be extended to more complex CPS in engineering fields. The intention here is to demonstrate how the properties of a hybrid system are useful for modelling CPS. The importance of a hybrid system allows us to have an abstract model useful for the verification and safety analysis of a CPS. Principles that must be adhered to in the engineering fields. This especially holds true for safety-critical systems.

1.2 Solving Numerical Simulations

At any given time, the system has a state (i.e., distance at a specific time, t). A numerical solution approximates the evolution of a dynamic system. The evolution of the dynamic system is a future set of states derived from the previous state. Using the collision detection example, the state of the system could be an object with a distance of 15cm at a time 0. At time 1, the distance would have changed to 8cm. These numbers belong in a state space denoted as $state\ space\ 1$. A state space is a simulation which produces a list of values derived from the same initial state at time 0. Additionally, the state of the collision system could be 20cm at time 0, at time 1, the distance is 5cm. This produces a list of values in a different state space, $state\ space\ 2$.

There are many approaches to solve simulations. Such examples include the Euler method. A numerical simulation of a dynamic system produces an evolution of variables within the system corresponding to a sequence of points in discretized time (Frehse, 2015).

1.3 Aims

Numerical simulation is used to solve the evolution of dynamic systems. This project aims to introduce an alternative way to simulate the evolution of dynamic systems using a neural network. The aims of the project are as follows:

- Explore hybrid systems by defining a precise hybrid automata model

This project uses 2 additional hybrid automata models to represent hybrid systems. This goes into detail about the semantics used in hybrid automaton. A discussion will be made about the limitations of hybrid systems and the software used to represent them.

- An understanding of the structure of a neural network

The architecture of the neural network used is a feed-forward neural network. The fundamental components within these deep learning algorithms are discussed. The parameters of the algorithms and the effect it has on the neural network are explored.

- Simulating hybrid systems/dynamic systems using neural networks

Numerical simulation of dynamics systems is discussed here, while addressing the software that can be used to implement them. An alternative approach to using neural networks is discussed and explored, while the outputs are examined.

- Evaluation of the accuracy of hybrid systems/continuous dynamic systems

An evaluation is made on the results of the neural network. A discussion was made about the ability to simulate different state spaces.

1.4 Changes

My initial proposal is detailed on focusing on hybrid systems only. However, at the beginning of my project, I decided to add purely continuous dynamic systems. The structure used to tackle the problem remains the same.

1.5 Overview

- Chapter 1: Introduction

An introduction of the problem is introduced in chapter 1. It details the main components of this project: dynamic systems, hybrid systems and neural networks. Hopefully, this provides context for the problem being solved.

- Chapter 2: Theoretical framework

This goes into further detail of the main components of this project, while addressing further concepts related to the components. This further explains why neural networks can be used to simulate dynamic systems.

- Chapter 3: Methodology

This discusses the approach used to solve the problem and why. The training data is explored. Further concepts within the theoretical framework are introduced and implemented. This project will explore the challenges faced with implementation. Additionally, this section provides details of the software used for implementing hybrid systems and the limitations of certain languages.

- Chapter 4: Results

This section goes into detail about the predictions made by the neural network. It details the ability of the neural network to produce accurate simulations and compares the outputs of the numerical simulations within the training data.

- Chapter 5: Discussions

This justifies the reason for the result while linking back to the theoretical framework.

- Chapter 6: Conclusions

This is a reflection on the overall project.

2. Theoretical framework

2.1 Dynamical systems

A dynamic system is a system that has function (The ODE) which describes the system. The notation of an ODE describes the evolution of a certain variable against time. This is the rate of change in variable v with respect to the time. For any given time, the system has a state (point). A state space consists of future states which are derived from the current state.

For example, Newton's law of cooling. This measures the rate of change in temperature, t_2 (output value) with respect to the time t_1 (input value). The ODE for Newtons cooling law is represented below:

Equation 1 : Newtons law of cooling

$$N(t_1) = t_2$$

$$\frac{dt_1}{dt_2} = \alpha A(t_{2_s} - t_0)$$

$A\ is\ surface\ area\ of\ object, t_0\ is\ initial\ temperature\ of\ object, t_{2_s}\ is\ temperature\ of\ surroungings, t_2\ is\ time, t_1\ is\ temperture$

The evolution of the variables corresponds to a sequence of time. ODE can be solved using integration. However, where an ODE is too complex to solve, numerical methods such as the Euler method are applied. This method will be further explained later.

2.1.1 Pure Dynamic System

A pure dynamic system exhibits continuous behaviour. The continuous behaviour is described using an ODE. Such examples include the equation above demonstrating the Newton law of cooling.

2.1.2 Hybrid Systems

A hybrid system exhibits both continuous behaviours and discrete behaviours. In addition, a hybrid system demonstrates the interaction between continuous and discrete behaviours. The continuous variables are influenced by the continuous dynamics and are affected by discrete models.

2.1.3 Hybrid Automata

Hybrid automation is the mathematical notation used for modelling hybrid systems. This covers the discrete states and the continuous evolution of real-value variables. A change in a discrete state can change the continuous real time variables and modify the ODE that describes how variables evolve with time.

To understand the semantics of hybrid automata the following example is used:

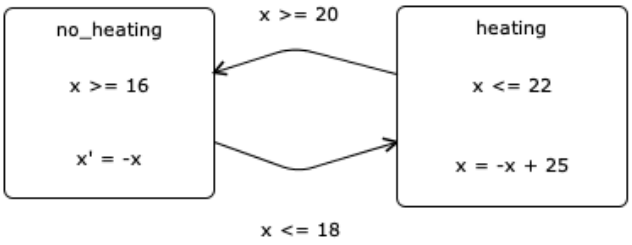


Figure 3: A hybrid automaton for maintaining the remperture of a room.

Consider a mechanism for maintaining the temperature in a room at a desired level, $19°C$. An abstract model is presented, by only considering the temperature of the room. It does not take into account of how the thermostat function behaves. The temperature of the room can be in two states, denoted as *no heating* and *heating*. Within each state is a different ODE. The ODE

is the dynamics of the systems. A discrete change from *no heating* to *heating* will change the dynamics being applied to the variable x (temperature). There is heating between the temperatures $16°C$ and $18°C$. Therefore, the dynamics of the temperature increases. There is no heating between the temperatures $20°C$ and $22°C$, here the dynamics of the temperature begin to decrease. The 2 main components of a discrete events are shown below:

(1) invariants/domains – "$x >= 16$" and "$x <= 22$" this is a predicate that states the property of x to be valid within a state.
(2) The guards "$x >= 20$" and "$x <= 18$". This determines when a discrete transition is allowed to occur.

The interpretation is as follows: The continuous variable x (temperature) will continue to decrease/increase corresponding to a differential equation given that the x holds true for an invariant within a state.

There are different "hidden" properties many systems carry that can be captured by a hybrid automaton. In particular, the property of ambiguity is presented in the thermostat model above. This can be demonstrated when the state is $(q, x) = (no\ heating, 17)$. A transition may take place and become on state $(q, x) = (heating, 17)$. This is because it satisfies the guard condition "$x \leq 18$". However it could continue to stay on the current state $(q, x) = (heating, 17)$. This is because it satisfies the invariant condition "$x >= 16$". This prominent characteristic is defined as a non-deterministic hybrid automation. A non-deterministic hybrid automaton may occur when:

(1) A subset of the condition, which holds true for an invariant in a state, also occurs within a guard transition.
(2) 2 locations share the same or a subset of invariants.
(3) Different continuous x values may leave from the same state.

We refer to (Tiziano Villa, 2015) for further information.

The notation of a non-deterministic hybrid automata simply means multiple simulations applied with the same initial condition would lead to different state spaces. This may present a problem when simulating predictions of hybrid systems from a neural network. The reason is that neural networks depend on recognising relationships and patterns within data. Fortunately, there are many software tools (further discussed in the methodology), that take care of that. The simulation algorithm of the software decides what action to take when there is a choice

between a transition or a choice to remain in the same state. A common approach is to use take the transition as soon as they are satisfied (Karl Henrick Johansson, 2009). This argument is applied to simulation algorithms and produces a precise deterministic simulation. This can then be used within a neural network to accurately find patterns within the data. However, the modelling language used for some software may have constraints, unsuitable to capture the behaviours of a hybrid system. This will be further discussed in the methodology.

Aside from non-deterministic hybrid automata, there are also other issues to consider when modelling hybrid automata. These are explained in the section below.

2.1.4 Modelling issues with Hybrid Automata

There are many modelling issues that need to be addressed in a hybrid automaton such as:

2.1.4.1 Existence of Executions

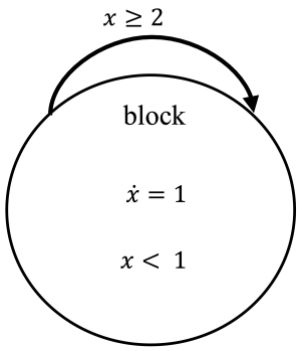


Figure 4: A blocking hybrid automaton

In hybrid automata the evolution of the state (the continuous variable) over time is called a run of the system. An execution is an individual state within the run. An execution relates to a specific variable corresponding to a specific time and node (discrete location).

Equation 2 : Definition of an execution

$$\chi = (t, q, x)$$
$$\chi = execution, t = time, q = node\ , x = continous\ variable$$

A run can therefore be defined as multiple executions. There are hybrid automata which provide no executions for a certain initial state. This is called a blocking hybrid system. Figure 4 shows an example of a blocking hybrid automaton. The state of the system found itself at $(q, x) = (block, 1)$. The invariant would force the state to leave the discrete location (node). However, the guard condition will not be enabled since $x = 1$. This is what is called a blocking hybrid automaton – where the number of executions is finite. If the initial state was $x = 1$. There will be no executions at all. This is seen as an undesirable property, as it suggests the mathematical model defined does not reflect the systems physical reality accurately (Karl Henrick Johansson, 2009).

To allow the neural network to make predictions, it will need training data. A hybrid system which exhibits a blocking state will lead to lack of/ or no executions. This would limit the neural network's ability to recognise patterns. The approach is to use hybrid automated systems that do not exhibit blocking properties.

2.1.4.2 Zeno Executions

A Zeno execution is defined as many discrete transitions as possible in a finite amount of time. Consider the bouncing ball hybrid automata, the ball bounces an infinite amount of time within a finite time. This is a complication, from hybrid systems. There is something inaccurate about the model. This is because in reality the ball will eventually stop bouncing. However, within the hybrid automata it will continue to multiple times even if the $time \longrightarrow \infty$.

Although this is not a good reflection on how the ball bounces in reality, this would not affect the neural network's ability to predict outputs. The neural network will be able to recognise the relationships and patterns within the data.

2.2 Numerical Simulation

The run of a dynamic system is called a numerical simulation. This computes how continuous variables evolve through time. The simulation begins with an initial state variable which continuously changes through time depending on the dynamics and the discrete events. A simulation can be run multiple times -each with a different initial state. The state space is denoted as a numerical simulation (a set of states) derived from the same initial state variable.

Numerical simulations can be used for formal verification. For example, it can be used for testing, where a large number of simulations are done to sample as many state spaces as possible.

This is in fact what we are doing to simulate our training data. There will be a multiple run of simulations to represent as many states as possible.

2.2.1 Numerical Simulation: Continuous Dynamic systems

The steps for computing a numerical simulation for a continuous dynamic system and hybrid automata differ. This is due to the properties they hold. A continuous dynamic system only has continuous dynamics. The steps for a numerical simulation are as follows.

1. Choose a single state location to begin with and initial values for the initial state variable.
2. Continuous step: compute the behaviour by solving the ODE of the system. A stop time is defined to end the simulations when it reaches the stop time. When the simulation reaches the stop time the simulation ends.

2.2.2 Numerical Simulations: Hybrid automata

Hybrid automation has discrete and continuous behaviours. The steps used to run a numerical simulation for a hybrid automaton differ from a continuous dynamic system. The steps for a numerical simulation are as follows:

1. Choose a single state location to begin with and initial values for the initial state variables.
2. Continuous step: Compute the behaviour by solving the ODE of the location. When an invariant condition has failed or a stop time has been reached, the simulation is stopped.
3. Discrete step: Detect the guards/transitions that are satisfied as they evolve through the continuous steps.
4. When the stop time has been reached, the simulation stops. Otherwise, continue from step 2.

2.2.3 Solving ODEs.

The continuous steps mention solving an ODE. This can be solved by using integration. However, where the ODE is complex, a numerical approach would be easier. This involves computing a sequence state values: $x_0, x_1 \dots x_n$ for a given state variable that correspond to time $t_0, t_1 \dots t_n$. The choice of the time plots could either be fixed where:

$$t_{i+1} = t_i + \Delta$$

$$\text{where } \Delta \text{ is denoted as the time step}$$

or the time step Δ is adapted (variable time step) to achieve minimal errors for the approximations. The approximation error is stated to generally reduce when $\Delta \longrightarrow 0$. This means a smaller time step leads to an accurate simulation.

Euler method: A numerical procedure that allows you to solve ODEs. The procedure can be shown below:

Equation 3: Eulers method

$$x_{i+1} = x_i + f(x_i) \times \Delta$$

where $x_0 \in \mathbb{R}$ denotes the initial value of x.

For a better understanding look at the example demonstrated below. Euler method is being applied to the ODE; $\frac{dy}{dx} = y$. It uses the exact same numerical procedure shown on equation 3 to produce a sequence of approximations.

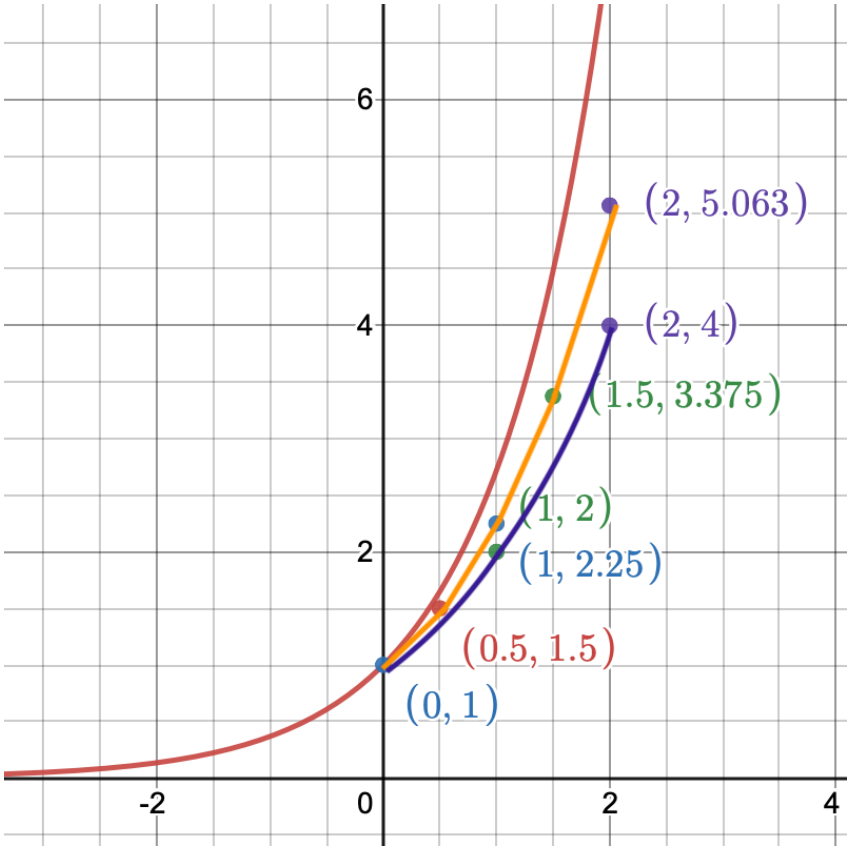


Figure 5 : An example of Eulers method

The purple line represents the time step $\Delta = 1$. The orange line represents the time step $\Delta = 0.5$. The red line is the actual function we want. This provides an approximation of what the particular solution is for a differential equation. Within each time step there is a local error from the approximation. This produces a global error - the overall error of the Euler method. The Figure above demonstrates that a small-time step would lead to a better approximation.

2.2.4 Stiff ODES

A stiff ODE occurs when a numerical simulation for solving the equations is numerically unstable. A numerical unstable ODE is when the growth of the global error grows with each timestep causing a large deviation from the actual answer (function). Nonetheless, solvers are forced to take extremely small-time steps to reduce the approximation error in each time step. (Frehse, 2015)

2.3 Neural Network

There are many neural networks architectures. The one used for this project is a feed forward neural network. Therefore, the concepts and architecture explained are mostly specific to a feed forward neural network.

2.3.1 Analogy of a Neural Network

The architecture of a neural network is inspired by biological neuron networks. These neurons are found within the visual cortex in our brain, allowing us to process information relayed to the retinas. This provides the ability to recognise patterns. Artificial neural networks are inspired and use the same analogy to extract useful information from raw data (training data) to learn how to predict outputs.

2.3.2 The neuron

A neuron is associated with a data value. This would make more sense further on. A layer in a neural network has a set of neurons and 1 bias. The bias is represented in red. The network can consist of a number of layers This is visualised below:

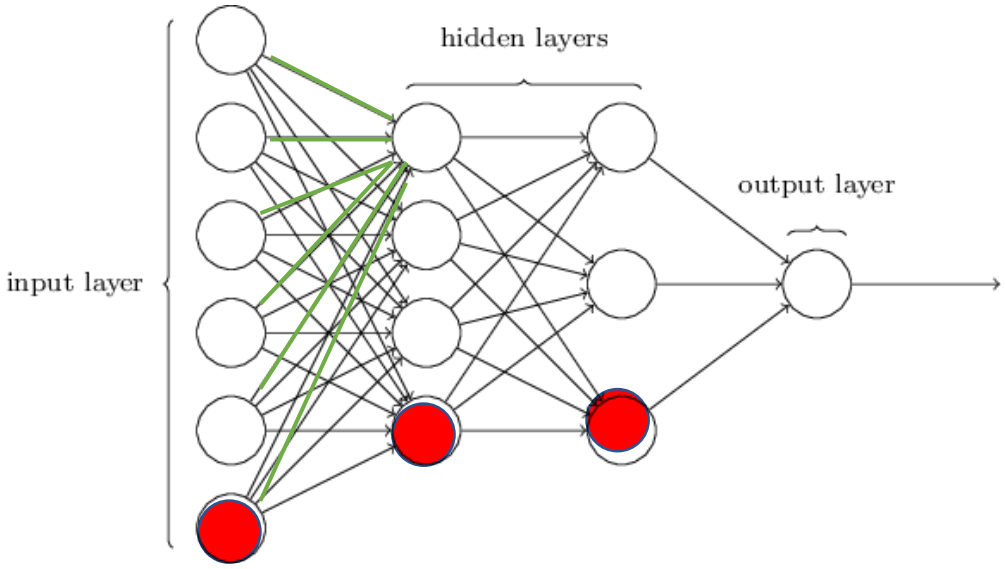
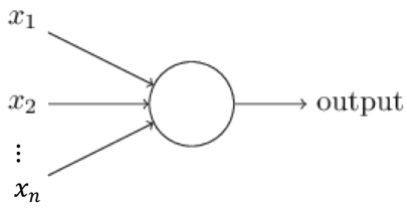


Figure 6: A neural network

To understand how the neural network works, we take a look at the first neuron in the second layer.



$n = 6$

Figure 7: A single neuron

Each neuron in the first layer is connected to this neuron as shown by the green edges (lines). This connection is associated with a weight. Therefore, this neuron takes in a set of weights, data values (the neurons in the first layer) and a bias denoted in red. These inputs are put into a function called the linear function as defined below:

$$g(w, x, b) = b + \sum_{i=0}^{m} x_i \times w_i$$

Equation 4: Linear function

Where w is a set is weights x is a set of data values (neurons) and b is the bias (denoted in red). The linear function is then put into an activation function to make $g(w, x, b)$ value between 0 and 1. A value which is close to 1 implies the neuron is activate while the value that is close to 0 means the neuron is not as activate. The activation function is denoted below:

$$\sigma(g(w, x, b))$$

Equation 5: Activation function

There are many activation functions, this will be further discussed later. A linear function and an activation function are applied to each neuron within the neural network. The input layers are the first layer in the neural network. The data value associated with the neurons in the first layer is the input given to the neural network. For example, for a neural network which classifies an image. The images will be encoded into an array of pixels. A 28×28 image will mean that the input layer would need to consist of 784 pixels. The neural network used in this project is simulating dynamic systems. This essentially means that it is predicting an output for a function. Therefore, our input is the number of inputs corresponding to the dynamic systems which are being modelled

2.3.3 Linear function

This focuses on the linear function itself. The linear function can be seen as a quantifier to measure how close a value which is the bias. The bias can be thought of as a threshold. The linear function can be represented as an inequality as shown below:

$$\sum_{i=0}^{m} x_i \times w_i > b$$

$$\sum_{i=0}^{m} x_i \times w_i \leq b$$

Equation 6 : Inequality of the linear function

In return, this bias affects the activation function. In terms of how to activate the neuron, it is provided by the activation function. For example, numbers close to 0 are inactive while numbers close to 1 are considered active.

2.3.4 Activation function

The activation function described above is the sigmoid activation function. This is defined below:

$$\sigma\big(g(w,x,b)\big) = \frac{1}{1+e^{g(w,x,b)}}$$

Equation 7: Sigmoid function

The activation function transforms a number into a value between 0 and 1. This function is usually made use of neural networks designed for classification problems, as it involves probabilities. Probabilities are defined between 0 and 1. The problem we are solving is a linear regression problem. This means our neural network uses a linear approach to model the relationship between a scalar response and one or more dependant variables. In context to this project, dynamical systems are described by ODEs with a time dependence. The neural network will model how variables (scalar response) change with respect to time (dependant variable). Therefore, this problem is more suited to the RELU activation function defined below:

$$\sigma\big(g(w,x,b)\big) = \max\,(0, g(w,x,b))$$

Equation 8: Relu function

2.4 Learning

To model the relationship, the neural network will adjust the weights and bias to capture the relationship/patterns between a function. Within every dynamic system there is a relationship to describe how variables evolve with time (ODE). Once there is a coherent relationship, the

neural network can adjust its weights and biases to be directly proportional to the function. Initially, these weights and biases are random then adjusted to correspond to a function. This is done through a process called gradient descent.

2.4.1 Gradient descent

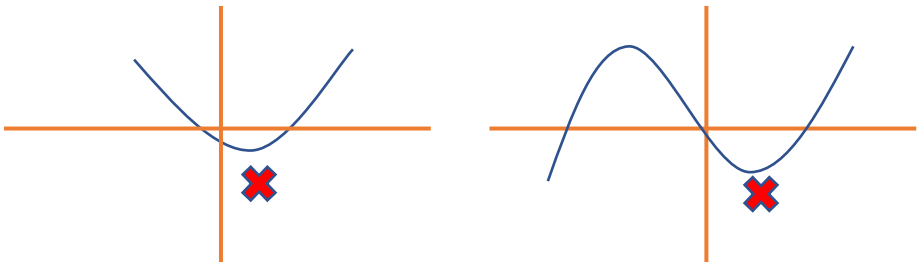


Figure 8: Functions

Gradient descent allows us to find the local minimum of a function. The local minimum has a slope (gradient) of 0. For example, the figures above show the local minimum denoted by the red cross. To get a better understanding of how gradient descent works in neural networks, we apply gradient descent to a simple model. Let's use the simple function $y = (x + 5)^2$. Gradient descent allows us to locate the minimum point (x, y). The steps are as follows:

1. Find the derivative of the function. In this case it is $y' = 2(x + 5)$.
2. Assign the dependent variables to a random value. In this case, we allow $x = 3$.
3. Iteration step: Computer the minimum point on the graph by using the procedure of gradient descent iteratively. The procedure is defined below:

$$x_n = x_{n-1} - learning\ rate \times\ gradient(x_{n-1})$$

Equation 9: Gradient descent formula

The $gradient(x)$ provides the gradient (e.t direction/slope) at point x. The learning rate is used to define how much we want to move in a certain direction provided by $gradient(x)$. This calculation is then subtracted from x_{n-1}, to allow us to move in the negative direction.

This procedure can be demonstrated in the figure 9. The steps are defined in the table below

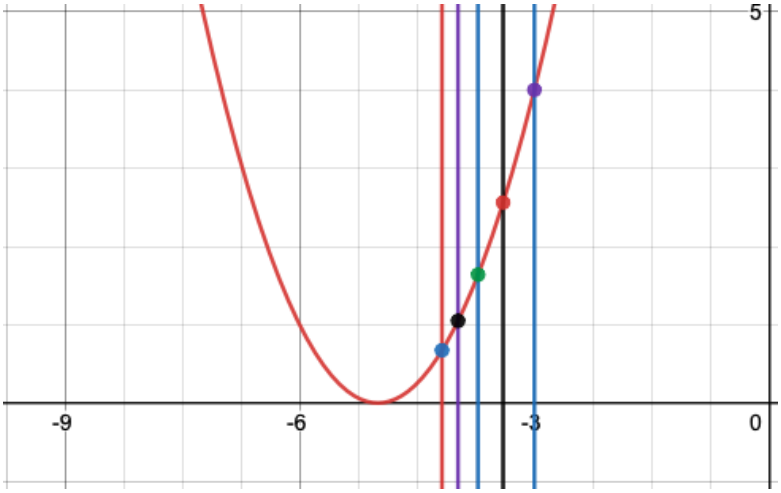


Figure 9: Gradient descent

x_n	$x_{n-1} - (learning\ rate) \times\ gradient(x_{n-1})$	Answer
x_1	$-3\ -\ (0.1)\ \times\ (2(-3)\ +\ 5)$	-3.4
x_2	$-3.4\ -\ (0.1)\ \times\ (2(-3)\ +\ 5)$	-3.72
x_3	$-3.72\ -\ (0.1)\ \times\ (2(-3)\ +\ 5)$	-3.976
x_4	$-3.962\ -\ (0.1) \times\ (2(-3)\ +\ 5)$	-4.1808

The same technique is extended to a linear regression model. The linear regression model finds the equation of a straight line which models the relationship between an output and the dependant variables. The equation of a straight line is shown below:

$$y = ax + c$$

$$Where\ a\ =\ slop\ of\ line\ and\ c\ =\ intercept\ of\ line$$

Let's say we are given a set of points that needs to be modelled using a straight line. We need to find values a and c in the line equation which will give the best approximate for given set of points. These points can be shown below.

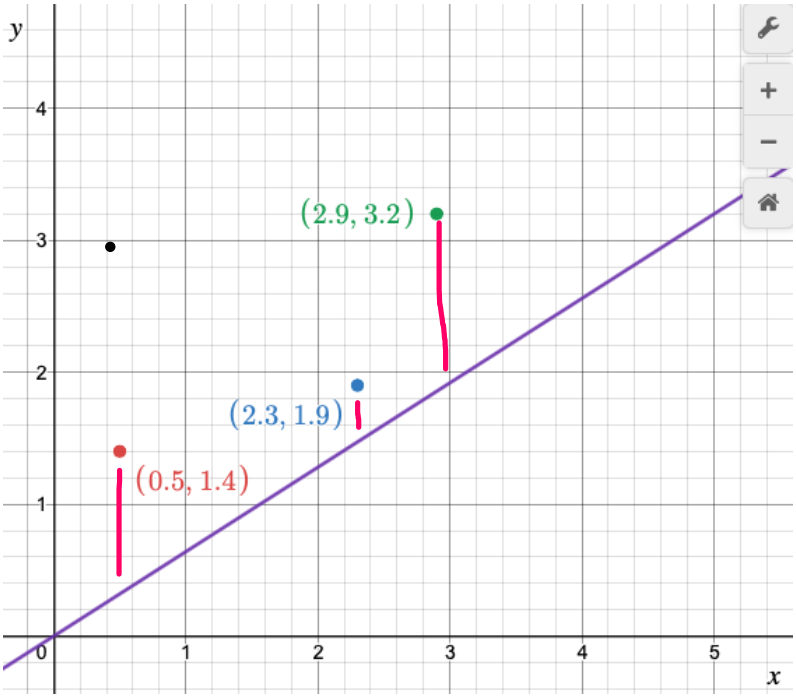


Figure 10 : y = ax + c

For simplicity, the value of a is provided, $a = 0.64$. Therefore, we are only finding the value of c that will provide the best approximation. To start the procedure of gradient descent we first assign c with a random value $c = 0$.

The red line denotes the distance. The distance of points is fairly high therefore an intercept using c = 0 does not provide the best approximation. To quantify how good the approximation is we use a formulate called (mean squared error) MSE.

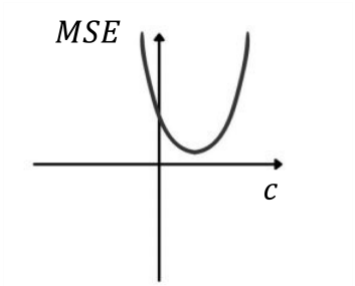
$$MSE = \frac{1}{n}\sum_{i=0}^{n}(y_i - \hat{y}_i)^2$$

Equation 10: MSE formula

y = actual result, $\hat{y}$ = predicted result. For our model $\hat{y}$ = c + 0.64*x. The MSE for this specific for our problem is solved below:

$$MSE(c_n) = (y_0 - \hat{y}_0)^2 + (y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2$$
$$MSE(c_0) = (1.4 - 0.32)^2 + (1.9 - 1.47)^2 + (3.2 - 1.86)^2 = 3.1$$

To find the best solution, we want $MSE \approx 0$. This is done the process of gradient descent. Imagine a graph of MSE plotted against the dependant variable c. This will produce a graph similar to the one shown below:



We want to find a value of c where MSE is close to 0. The same steps defined for gradient descent are applied to MSE. The complicated part is defining the gradient for the MSE. This is calculated using the chain rule. The gradient of MSE for this specific problem is shown below:

$$\frac{dMSE}{dc} = \sum_{i=0}^{n} \frac{d}{dc_n}$$

$$where \frac{d}{dc_n} = -2(y_n - \hat{y})$$

The gradient when $c = 0$ is defined below

$$\frac{dMSE}{dc} = -2(y_0 - \hat{y}_0) + -2(2y_1 - \hat{y}_1) + -2((y_2 - \hat{y}_2)$$

$$= -2(1.4 - 0.32) + -2(1.9 - 1.47) + -2(3.2 - 1.86) = -5.7$$

The gradient descent for the MSE steps is shown below:

c_n	$c_{n-1} - (learning\ rate) \times gradient(c_{n-1})$	Answer
c_1	$0 - (0.1) \times -5.7$	0.57
c_2	$0.57 - (0.1) \times -2.3$	0.8
c_3	$0.8 - (0.1) \times -0.9$	0.89
c_4	$0.89 - (0.1) \times -0.37$	0.92

Let's compare $c_4 = 0.92$ to our initial guess $c_0 = 0$

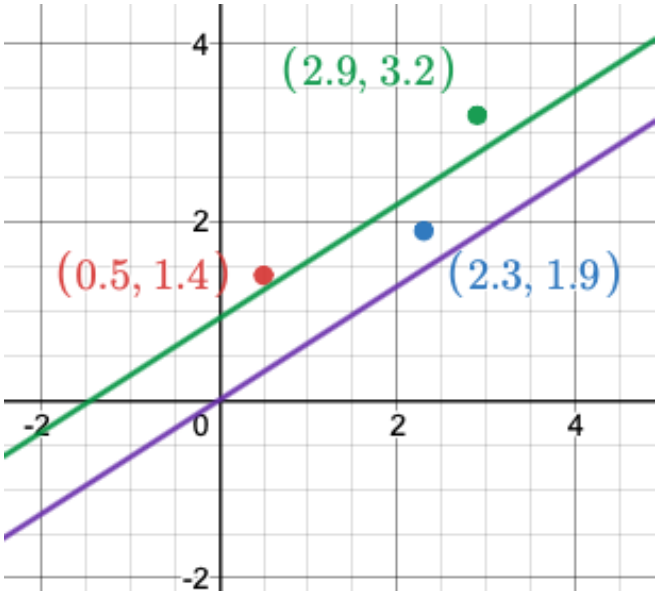


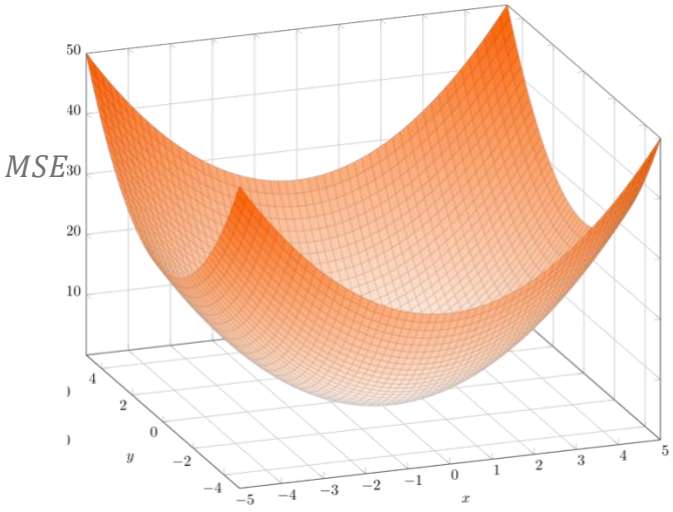
Figure 11: Comparisons of intercepts of c

As you can see c_4 (denoted in green) models the set of plots a lot better than c_0 (denoted in purple). The gradient can go into more steps until our gradient (c_4) is fairly close to 0 meaning we found the minimum point or are close enough to the minimum point.

In summary:

- The MSE is used to measure how well the approximation is.
- Gradient descent is used to find the optimal approximation.

This simple problem can be extended to further problems. For example, if the parameter of a was not provided in the line equations. The MSE function would look something like this:



(Wikimedia commons, 2020)

Figure 12: 3D function

The MSE used to evaluate the performance of the neural network is denoted as $MSE(w, x, b)$. This is usually called the cost function.

2.5 Backpropagation

Backpropagation takes the concept of gradient descent and extends it further. Whereby a single neuron uses gradient descent to adjust inputs of weights w, data values x and bias b. For example, the neuron shown in figure 7 uses the inputs from the previous layers and calculates $MSE(w, x, b)$ to evaluate how well the input parameter approximates to the actual value.

To summarise, the concept of backpropagation begins from the last $layer_n$ of the neural network then adjust the values predicted using gradient descent. $layer_{n-1}$ uses these adjustments to further adjust its own parameters. This is essentially how the overall neural network learns.

3. Methodology

The methodology addresses the approach used to achieve the aims of the project. This is done by introducing the dynamic systems that will be used in the project, while linking back to the important topics introduced in the *theoretical framework* which are applied in the *methodology*. In addition, describe and justify the process done to simulate output of dynamic systems and the experiments used to evaluate the performance.

3.1 Explore hybrid systems and continuous dynamic systems, including defining a precise definition of the hybrid automata model.

This project has a set of continuous dynamic and hybrid systems that will be numerically simulated. These numerical simulations will be used as training data for the neural network. The predictions of the neural network will then be evaluated. Before we start investigating the predictions, it's important to understand:

1. The structure of the training data and testing data
2. Software used to produce these dynamic systems
3. Numerical simulation

3.1.1 Training and Testing data

Training data is provided for the neural network model to learn the function of a dynamic system. Training data and test data are provided by numerical simulations. The modelling issues discussed in the theoretical framework will directly affect the neural network's ability to learn. This is because the neural networks training data relies on a numerical simulation. Dynamic systems that hold properties of either a blocking, non-deterministic hybrid automata or stiff ODE will affect the neural network's ability to learn. This is because the numerical simulation produced will be incoherent and cannot be used for training the neural network.

To produce training data and test data, a large number of simulation runs will be computed to collect samples of state spaces. The initial condition specifies what the initial state variable is. A state space will consist of consecutive state variables derived from the same initial state variable.

3.1.2 Software

The process of gathering training data/test data for each dynamic system is as follows: First, gather N number of samples. The simulation of each sample is solved numerically and saved within a CSV file. Each line of a CSV denotes an execution, were $execution = initial_n, time_n, x_n$. That is the initial condition of the state, specific time of the state and the corresponding state variable x respectively. Note that x, can be thought of as a list of continuous variables that belong in a dynamic system. For continuous dynamic systems, to solve the ODE the LSODA solver is used. This is obtained by using the function odeint from python's SciPy library. For constructing hybrid systems, Simulink's StateFlow chart is used. This is found in MATLAB.

Stateflow simplifies the design of a hybrid system that contains complex logic. This is done by defining state diagrams and flowcharts. Figure 6 demonstrates a *spiking neurons* hybrid automaton implemented using Stateflow. This was easy to construct by using state diagrams and specifying the continuous dynamics.

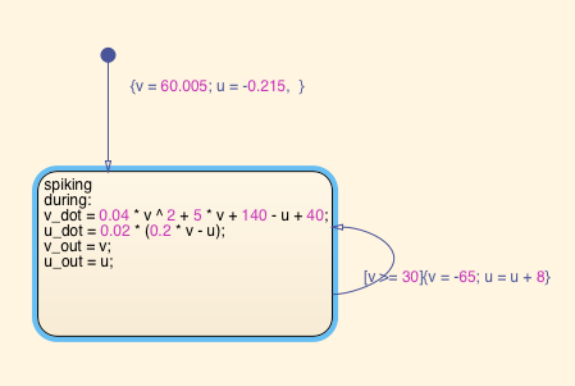


Figure 13 : A hybrid automaton model of spiking neuron constructed in stateflow.

Despite the simplification Stateflow offers, the models lack formal mathematical semantics of a hybrid automata. In particular, the language does not provide semantics for an invariant in a discrete location. This is a problem, as it prevented some hybrid automata being correctly simulated as it does not reflect the actual system. This project will construct a hybrid automaton that does not rely on invariants.

3.1.3 Numerical simulation

The specific solver selected in Simulink was “ode1”. This used the Euler method which was explained in the *theoretical framework*. This method was used to simulate hybrid systems. The LSODA solver found in the odeint function within python is used for solving continuous dynamic systems. This solver is automatically selected between the Adams and BDF methods. The method selected for solving the ODE depends on the ODE itself; for a non-stiff ODE it uses Adams, for a stiff ODE is used BDF. (Hindmarsh & Petzold, 2005).

3.2 Hybrid and continuous dynamic systems

Below defines a set of continuous dynamic and hybrid systems. These will be used to experiment how well a neural network model simulates a dynamic system. In the following sections, they are classified according to their dynamics.

3.2.1 Continuous dynamic systems

This section introduces the 4 dynamical systems used within the experiments.

3.2.1.1 Newton's Cooling Law

Equation 11 : Newtons cooling law

$$dT/dt = -0.015(T - 22)$$

$$T(0) = T_0$$

$$T = Temperature\ (any\ real\ number); t = time; T_0\ is\ the\ inital\ temperature\ value$$

This equation is a dynamic system that adheres to the newtons law of cooling. The variable T denotes the object temperature. The number 22 is the ambient temperature (environment temperature). A real-life analogy could be the cooling of a water in a cup placed in a room. The water temperature's rate of change in the cup depends on the object temperature (that is the temperature of the water itself) and the ambient temperature. When the ambient temperature is greater than the object temperature, the temperature of water increases. This means the temperature's rate of change would be positive. When the ambient temperature is less than the object temperature, the temperature of water decreases. This means rate of change would be negative.

The dynamics of the newton cooling law are quite simple. Therefore, it makes sense to test the neural network model on this simple problem first.

The analogy of the temperature of water in a cup can be applied to the model. The neural network for this system would take two inputs, as shown in equation 12.

Equation 12 : A neural network modeled for simulating Newton's cooling law.

$$Neural_1(T_0, t) = \mathrm{T}(\mathrm{t}, \mathrm{T}_0)$$

t is a sequence of time steps, T_0 is the initial state variable (The temperature at $t = 0$). $T(t, T_0)$ is the state space for variables corresponding to a sequence of time t, derived from T_0.

The neural network model returns a state space corresponding to T_0

To get a better understanding of the meaning of state space. The simplest model, which is Newton cooling law, is used as an example to explain the importance of a state space. The current knowledge is that a state space consists of state variables derived from an initial state variable. For the dynamics of Newtons Cooling Law, how the variables evolve (decrease/increase) through time is dependent on the initial temperature. For example, the initial temperature for one state space is 22, the consecutive temperatures for a specific time would be different, than of that for an initial temperature beginning with 16.

3.2.1.2 Van der Pol Oscillator

Equation 13 : Van der Pol Oscillator

$$d^2y/dt^2 - \mu(1 - y)\, dy/dt - x = 0$$

Van der Pol Oscillator is an oscillator with non-linear damping. The ODE is a second-order differential equation. This can be further broken down into a system of equations. As shown below:

Equation 14 : System of equations of van der Pol Oscillator

$$\frac{dy}{dt} = \mu(1 - x^2)y - x$$

$$\frac{dx}{dt} = y$$

This equation is used for describing the behaviour of "self-sustaining oscillations in which energy is fed into small oscillations and removed from large oscillations" (MathWorld Team, 2021). In the experiments, we let $\mu = 0.5$.

The neural network model for this input will have three inputs: the 2 initial state variables (x, y) , and time t.

Equation 15: A neural network modeled for simulating the van der Pol Oscillator

$$Neural_2(x_0, y_0, t) = \begin{bmatrix} x(x_0, y_0, t) \\ y(x_0, y_0, t) \end{bmatrix}$$

Where $x(x_0, y_0, t)$ denotes the evolution of variable x corresponding to a sequence of time steps, t. $y(x_0, y_0, t)$ denotes the evolutions of variable y corresponding to sequence of time steps, t. x_0, y_0 are the initial variables. The neural network returns a state space consisting of x and y variables derived from the initial conditions of x_0 and y_0 with the corresponding time sequence t.

3.2.1.3 Laub Loomis

This dynamic system is from a paper by Laub and Loomis. The is a set of non-linear differential equations (shown in equation 16). This dynamic system model changes in the enzymatic activities with respect to time t. (Michael T. Laub, 2017)

Equation 16: Systems of Equation for Laub Loomis

$$\frac{dx_1}{dt} = 1.4x_3 - 0.9x_1$$

$$\frac{dx_2}{dt} = 2.5x_5 - 1.5x_2$$

$$\frac{dx_3}{dt} = 0.6x_7 - 0.8x_2x_3$$

$$\frac{dx_4}{dt} = 2 - 1.3x_3x_4$$

$$\frac{dx_5}{dt} = 0.7x_1 - x_4x_5$$

$$\frac{dx_6}{dt} = 0.3x_1 - 3.1x_6$$

$$\frac{dx_7}{dt} = 1.8x_6 - 1.5x_2x_7$$

The corresponding neural network model would have 8 inputs: 7 initial state variables and time t.

Equation 17: A neural network modeled for simulating Laub Loomis

$$Neural_3(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, t) = \begin{bmatrix} x_1(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, t) \\ x_2(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, t) \\ x_3(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, t) \\ x_4(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, t) \\ x_5(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, t) \\ x_6(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, t) \\ x_7(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, t) \end{bmatrix}$$

Where $x_1(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, t) \dots x_7(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, t)$ denotes the evolution of variables $x_1, \dots x_7$ corresponding to a sequence of time t. $x_{1_0} \dots x_{7_0}$ are the initial variables. The neural network returns a state space consisting of variables of $x_1 \dots y_7$ corresponding to a time sequence t derived from the initial variables $x_{1_0} \dots x_{7_0}$.

3.2.1.4 Biological model

This dynamic system is constructed from 9 non-linear differential equation (Lehrach, 2005). Similar to Laub Loomis, this ODE model is a biological system within the 9-equation shown below:

Equation 18: Systems of equations for Biological model

$$\frac{dx_1}{dt} = 3x_3 - x_1x_6$$

$$\frac{dx_2}{dt} = x_4 - x_2x_6$$

$$\frac{dx_3}{dt} = x_1x_6 - 3x_3$$

$$\frac{dx_4}{dt} = x_2x_6 - x_4$$

$$\frac{dx_5}{dt} = 3x_3 + 5x_3 + x_4 - x_6(x_1 + x_2 + 2x_8 + 1)$$

$$\frac{dx_6}{dt} = 5x_4 + x_2 - 0.5x_7$$

$$\frac{dx_7}{dt} = 5x_4 + x_2 - 0.5x_7$$

$$\frac{dx_8}{dt} = 5x_7 - 2x_6x_8 + x_9 - 0.2x_8$$

$$\frac{dx_9}{dt} = 2x_6x_8 - x_9$$

The corresponding neural network model will have 10 inputs: 9 initial state variables and time t.

Equation 12: A neural network modeled for simulating the biological model

$$Neural_4(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, x_{8_0}, x_{9_0}\, t) = \begin{bmatrix} x_1(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, x_{8_0}, x_{9_0}t) \\ x_2(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, x_{8_0}, x_{9_0}t) \\ x_3(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, x_{8_0}, x_{9_0}t) \\ x_4(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, x_{8_0}, x_{9_0}t) \\ x_5(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, x_{8_0}, x_{9_0}t) \\ x_6(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, x_{8_0}, x_{9_0}t) \\ x_7(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, x_{8_0}, x_{9_0}t) \end{bmatrix}$$

$x_1(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{6_0}, x_{7_0}, x_{8_0}, x_{9_0}t) \dots x_9(x_{1_0}, x_{2_0}, x_{3_0}, x_{4_0}x_{5_0}, x_{8_0}, x_{9_0}, t)$ denotes the evolution of variables $x_1 \dots x_7$ in Laub Loomis corresponding to a time sequence t. $x_{1_0} \dots x_{9_0}$ are the initial variables. The neural network returns a state space consisting of $x_1 \dots y_9$ derived from the initial variables and corresponds to the time sequence t

3.3 Hybrid Automata

This section explains the 2 hybrids automata models used within the experiments.

3.3.1.1 Bouncing ball

The bouncing ball hybrid automaton was made using Simulink's Stateflow chart on MATLAB.

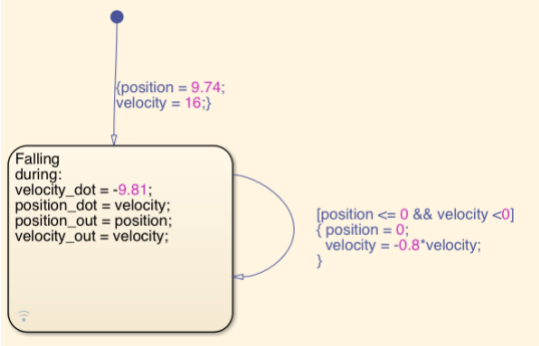


Figure 14: A hybrid automation of a bouncing ball constructed using Stateflow.

Hybrid automation represents a scenario where a ball drops from a predefined height, p. It hits the ground then bounces up again. There are 2 differential equation which describes the behaviour of this model.

Equation 13: A neural network modeled for simulating the bouncing ball

$$\frac{dv}{dt} = -9.81$$

$$\frac{dp}{dt} = v$$

$$let\ v = velocity\ , p = position\ (height)$$

For this model it's assumed the mass of the object is 1kg. The guard ensures that the position is less or equal to 0 (when the ball hits the ground) and the velocity is less than 0 (to ensure the ball is falling). This model introduces reset condition semantics. This means that when it satisfies a guard, it resets the value of the continuous variable. In this case velocity is set to $v = -0.8 * velocity$.

The neural network used to model this hybrid automaton takes in 3 variables. This includes 3 initial state variables and time t.

$$Neural_5(p_0, v_0, t) = \begin{bmatrix} p(p_0, v_0, t) \\ v(p_0, v_0, t) \end{bmatrix}$$

Where $p(p_0, v_0, t)$ denotes the variables of p corresponding to a sequence of time t. $v(p_0, v_0, t)$ denotes the variables of v corresponding to sequence of time t. p_0, v_0 are the initial variables. The neural network returns a state space consisting of p and v variables derived from the initial conditions of (p_0 , v_0) and the corresponding time sequence.

The bouncing ball hybrid automaton was briefly discussed in the *theoretical framework*, as it exhibits the Zeno property.

Figure 15 : Shows how the variables p, v evolves through the time.

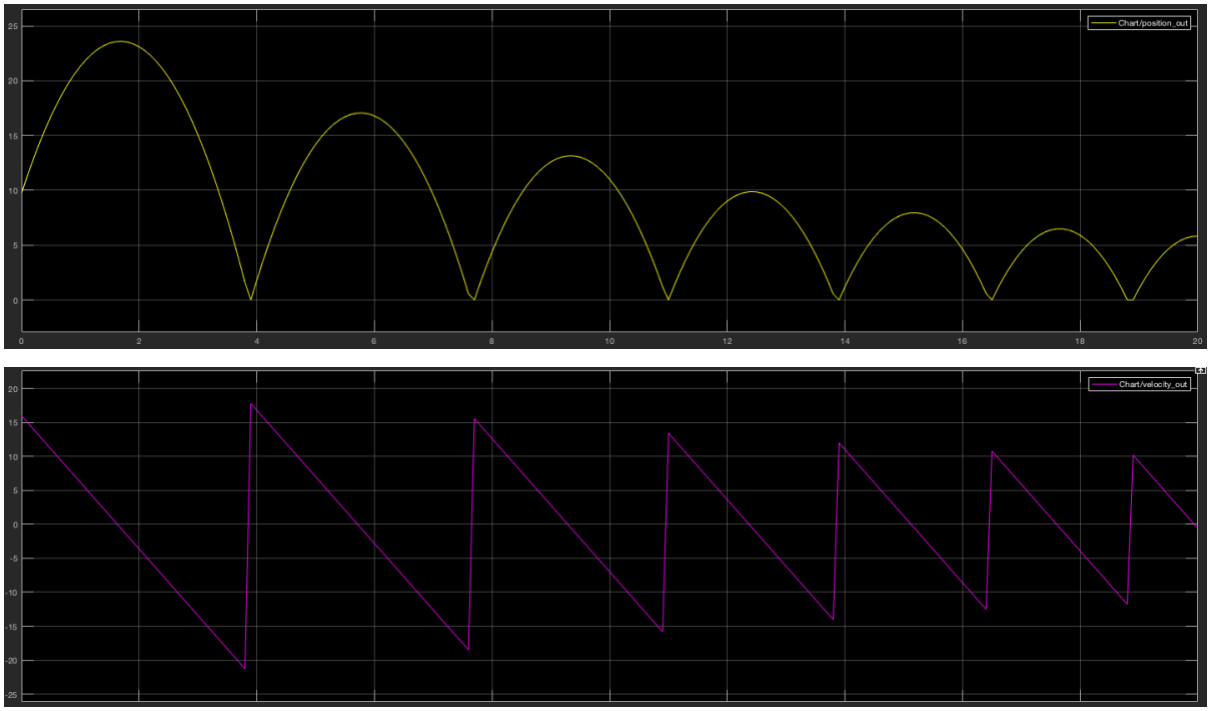


Figure 15 shows the position/velocity changing over time. The top graph shows the position. The bottom graph shows the velocity. Time t is shown on the x-axis, along with the corresponding variable shown on the y axis.

3.3.1.2 Spiking neurons

This is a hybrid automaton made using Stateflow on MATLAB. This dynamic system is used for modelling the voltage of a neurons cell.

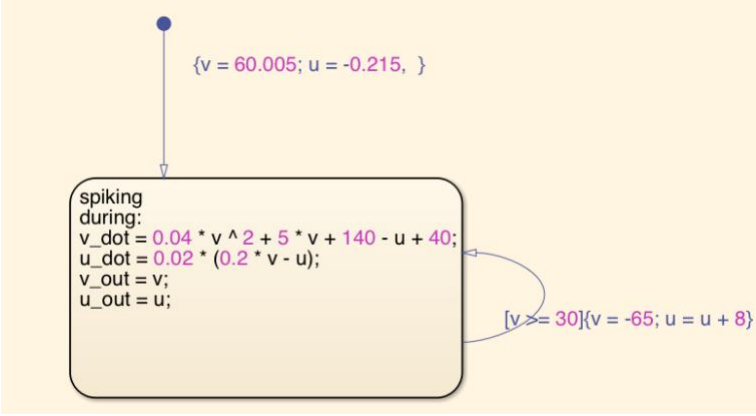


Figure 16: This shows the hybrid automation of spiking neurons constructed using Stateflow.

This is modelled with 2 ODEs:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I$$

$$\frac{du}{dt} = a(bv - u)$$

$$let\ a = 0.02, b = 0.2, c = -65, d = 8\ , I = 40$$

The neural network model for this dynamic system has 2 initial variables and a time input t.

$$Neural_5(u_0, v_0, t) = \begin{bmatrix} v(u_0, v_0, t) \\ v(u_0, v_0, t) \end{bmatrix}$$

Where $u(u_0, v, t)$ denotes the evolution of variable u corresponding to a sequence of time steps, t. $v(u_0, v_0, t)$ denotes the evolutions of variable v corresponding to sequence of time steps, t. u_0, v_0 are the initial variables. The neural network returns a state space consisting of u and v variables derived from the initial conditions of x_0 and y_0 with the corresponding time sequence t.

3.3.2 Dynamic Systems: Experiential Settings (1)

This section details the experiment that takes place in the project. This experiment compares the predictions and the simulations (training data/ test data) of 6 dynamic systems. The process for this experiment is explained below:

First gather $N_{training}$ amount of training data of each system by using a numerical simulation. Keep in mind that the training data consist of $S_{training}$ state space samples. The test data consist of simulations of state space that does not exist in the training data. This is because the performance of the neural network is done by evaluating the test data. The performance of the neural network is measured by the prediction of accuracy of data which is not within the training data. The performance of the neural network is explored further by looking at the input values within the test data. This mean it takes a further look at the accuracy of prediction of state spaces that are close to the training data, compared to those further away from the state spaces within the training data. This would be further discussed later in the evaluation. The initial state variables for the training data for each system is represented as the vector $I_{training}$. Each sample space consists of a finite number of $E_{training}$ executions, which is dependent on

the stop time t_s of a numerical simulation. For each simulation there would be a time step Δ. The time step, Δ is chosen such that the state variables simulated is sufficient enough to represent the dynamic system.

The testing data is simulated similar to the training data with 2 distinct differences:

1. The sample space for testing will include samples of state spaces that are not within the training data.
2. The state space is chosen at random.

The reason for (1) is to explore states spaces that are further away from the state space within the training data. This will allow us to evaluate how the accuracy changes for certain input values. The initial state vector for testing is represented as $I_{testing}$.

To gain a better understanding, an example of an experiment using Newton's cooling law is explained. Let's say $I_{training} = [2,3,4, \ldots 20]$. This implies we have of 19 simulations ($S_{training} = 19$) samples. Each sample corresponds to one initial variable within the $I_{training}$. The structure of the training data is as follows: the first simulation the temperature evolves from initial state variable temperature 2, for the second simulation the temperatures evolve from initial state variable 3 and so on. Let's say $I_{testing}$ = [0,0.2,0.3, … 25]. The testing data would consist of $S_{testing}$ random samples from the initial variable condition $I_{testing}$. For example, if $S_{testing}$ = 5. This means 5 random initial condition will be selected from $I_{testing}$.

The parameter values of the dynamic systems are shown in Table 1. Once the predictions of the neural network are obtained, we evaluate the predictions of the neural networks by using various statistical methods. Each individual prediction is shown as input-output vectors

$$[x(t_0), t_n, x_n]$$

x *denotes a list of continuous variables that belongs to the system*

$x(t_0)$*provides the initial variables corresponding to* x_n

T_n *is the specific time step*

x_n, is the output provided by the neural network and $x(t_0), t_n$ is the inputs to the neural network.

Table 1: Descriptions and numerical values for some of the parameters

Parameter	Description	Newtons Cooling law	Van der Pol	Laub Loomis	Biological model II
$N_{training}$	Number of training data	12000	321600	640000	256000
$N_{testing}$	Number of testing data	6000	321600	640000	256000
$I_{training}$	The initial state variable for the training data	[1,2...60]	[1,4] for x, y	[1,2] for $x_1, x_2, x_3, x_4, x_5, x_6, x_7$	[0.99, 1.01] for $x_1, x_2, x_3, x_4, x_5, x_6, x_7 x_8 x_9$
$I_{testing}$	The initial state variables for the testing data	[0,0.1,0.3, … 65]	[0,0.001, 0.002, 0.003, …5]	[Any real number between 0 and 4]	[Any real number between 1.02 and 1.04]
$S_{training}$	Number of state space samples within the training data	60	16	49	81
$S_{testing}$	Number of state space samples within the testing data	30	16	64	512
$E_{training}$	Number of executions for a simulation sample within the training data.	200	40200	5000	8500
$E_{testing}$	Number of executions for simulation samples within the testing data	200	40200	5000	8500
t_s	The stop time of a simulation	199	20	499	4.99
Δ	Time step	1	0.001	0.1	0.001

⋮

Parameter	Bouncing ball	Spiking neurons
$N_{training}$	20002	6007
$N_{testing}$	30002	7006
$I_{training}$	[10,11] for position [16] for velocity	[-0.2] for u, [60, 61, 62, 63, 64] for v

$I_{testing}$	[10.65, 11.45, 9.74] for position [16] for velocity	[-0.15, -0.21 -0.215, -0.245, -0.265, -0.345, -0.356] for u [59.001, 59.001, 60.005, 65.361, 63.895, 65.452, 64.691] for v
$S_{training}$	2	6
$S_{testing}$	3	7
$E_{training}$	10001	1001
$E_{testing}$	10001	1001
t_s	10000	100
Δ	0.1	0.1

The choice of parameters (number of training data etc) is completely arbitrary. The choice of a parameter was decided when the dynamic system was implemented. However, the amount of training data and test data were hindered due to the software tools being used. The simulation of continuous system was easier to produce, as it was more of an autonomous process. For example, in order to produce a numerical simulation of the van der pol model, the function f shown on figure 17 would be passed through the odeint function along with initial state variables and a time sequence. This would return a simulation state space which is saved in a csv file.

```
MU = 0.5
def f(state, t):
    x, y = state
    dxdt = y
    dydt = MU*(1 - x * x) * y - x
    return dxdt, dydt
```

Figure 17: The function of van der Pol is implemented in python

Further samples state space will be simulated using a for loop to produce the overall training data/test data.

For hybrid systems, gathering samples of state spaces was tedious. This is because when running a sample simulation on Stateflow it will produce an output consisting of a time sequence and the corresponding sequence of variables. This output would then need to be copied to excel. This is the main reason the hybrid system has less training data than the continuous dynamic systems.

Neural networks can overfit. This means that the neural network performs better on the training data than it does on predicting new data. This is further explained in the *overfitting* section. Therefore, it is important to use testing data to evaluate the performance of the neural network. In industry standards there are different ways to determine how the training and testing data are split. A neural network dealing with classification will perform a random split of a large dataset. This will produce training and testing data. A neural network dealing with time-series, usually splits the data into consecutive blocks (using the first 80% of the timeseries for training and the 20% for testing etc). For the models within our project, multiple simulations are computed to sample many state spaces. The training-test split is structured in such a way that the samples in the test data are not contained with the training data. However, the number of executions for each sample of simulations within the test data and training data remains the same. This essentially means that for each sample simulation, the stop time remains the same.

This project has 6 dynamical systems. This includes 4 continuous dynamic systems and 2 hybrid automated models. These dynamical systems have been defined using the appropriate semantics and implemented on specified software tools. Once the training data for each dynamic system has been collected, it can now be carried forward and used for training the neural network.

3.4 Simulating the output of a hybrid system with a neural network implementation.

Section *Dynamic Systems: Experimental settings (1*) details the method that is used to collect results. This section focuses on comparing the predictions made by the neural network and numerical simulations (training data/ testing data).

3.4.1 Dynamic Systems: Experimental settings (2)

This experiment follows on from *Dynamic Systems: Experimental settings (1)*. This experiment focuses on evaluating the performance of the neural network using test data with appropriate statistical methods. A comparison will be made between the predictions and numerical simulation (test data). The format of the experiment is explained below:

1. The **training data** and predictions of sample simulations will be plotted on a graph. This is to visually compare the outputs of a dynamic system produced by the predictions and simulations.
2. Similar to (1), the **test data** and predictions made by the neural network of each simulation sample would be plotted on a graph.
3. To evaluate the performance of the neural network, the loss of each sample simulation in the test data would be compared. This is to make a conclusion about the input variable that goes into the neural network and how that affects the loss
4. In addition, the output (loss) for each simulation would be shown on a boxplot. This allows us to view the distribution of the data.
5. The loss of each time step for each simulation sample is collected. The average of a specific time step for all the simulations is then calculated and plotted on graph. This technique is used to evaluate the performance of the neural network. This is because it provides conclusions, about the input of a certain time step and the effect of the loss.

3.5 An understanding of the overall structure of a neural network.

The structure of the neural network we have is visualised below:

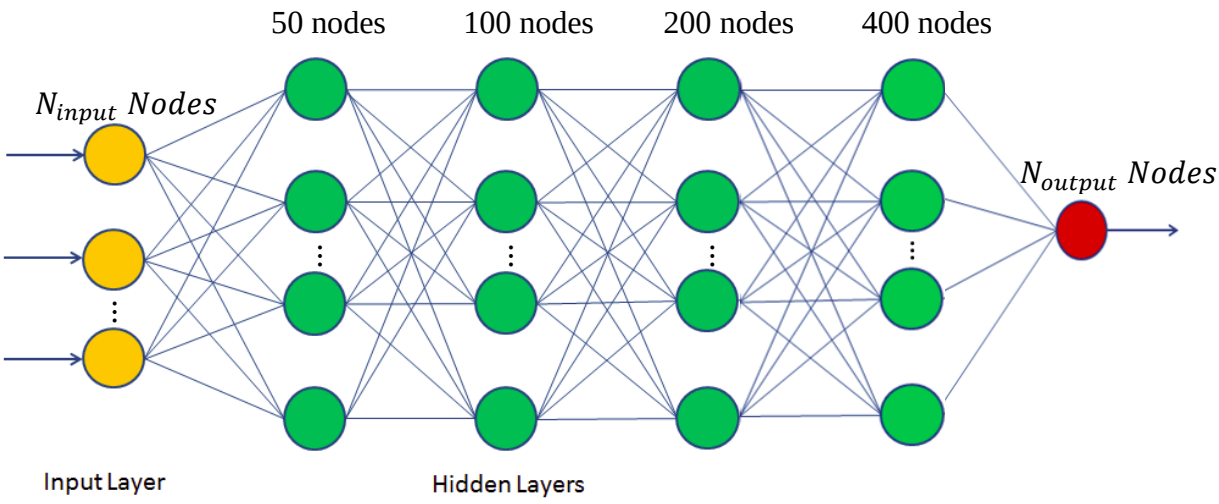


Figure 18: The structure of the neural network

This can be seen as a vector where the position of a variable denotes the layer.

$$[N_{input}, 50, 100, 200, 400, N_{output}]$$

The first layer is the input layer. The last layer is the output layer. These layers will vary depending on the dynamic system. The architecture of the neural network is a feed-forward network. This means each neuron is densely connected to the next layer of neurons. This

architecture allows relationships and patterns within the state space to be captured. The number of parameters (weight and bias) within the hidden layer is 105,004 This essentially means that 105,004 weights and bias within the hidden layers of the neural network will have to be shifted to satisfy a function. This is more than enough parameters to represent each dynamic system. However, this presents the issue of overfitting which will be discussed later.

3.5.1 Neural Network: Experimental settings (1)

This experiment focuses on finding the best parameter for the neural network. This includes experimenting with different learning parameters (parameters of the neural network) and analysing the effects on the loss. The learning parameters of a neural network include:

- Learning rate
- Number of epochs
- Batch size
- Number of layers
- Number of nodes within a specific layer.

Various values will be applied to these learning parameters to see the effect of loss. The parameter of a neural network also depends on the type of data the neural network is training. This means that a learning parameter that works well for a sequence of data may not work for another sequence of data. This is why the training data of these 3 dynamical systems is used in this experiment. The 3 dynamical system include:

- Newtons Cooling law
- Van der Pol Oscillator
- Laub Loomis

The training data of these dynamic systems are used to explore how the parameter affects the loss. The approach is as follows: For each dynamic system, we explore the best parameters by collecting 30 loss items. These 30 loss items each correspond to a certain value of a learning parameter. The 30 data items are illustrated in the boxplot. This is because a boxplot will allow us to see the distribution of data and quickly identify the statistical averages.

To collect 30 loss items for a specific parameter, the neural network will run 30 times using the same value for a specific parameter. For example, when exploring the effect of the learning rate, we start with the learning rate of 0.001. Afterwards, the neural network runs for 30 times with a learning rate of 0.001. The 30 items will be collected. A learning rate of 0.00001 is then chosen next. The same process is repeated. Statistical averages of 30 data items are then calculated. This will show the effect of a particular parameter on the performance of the neural network. The reason for collecting 30 data items is because it provides a reliable measurement of the performance of the neural network model for a specific learning parameter. However, there is an issue with gathering 30 data loss items. The issue is that it will take a long time. Therefore, the use of parallel programming is applied to perform the repeated number of loops simultaneously. This provides 2-3x time performance with the 4-core machine.

This was done using the python multiprocessing library. This is demonstrated with the code shown below:

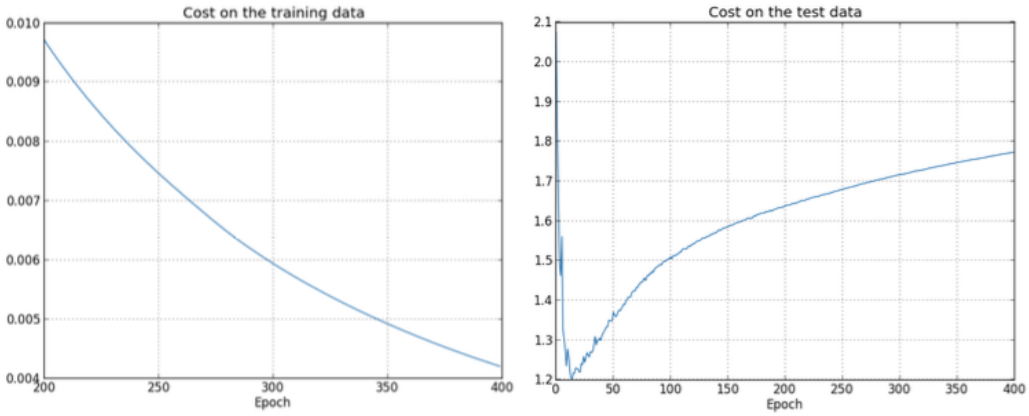
```
Parallel(n_jobs=multiprocessing.cpu_count())(delayed(predictions)(number_inputs,
number_classes, learning_rate, batch_size_1, num_epoches_1, inputs, targets, True,
None, None) for _ in range(30))
```

Despite the parallel functionality, the resources on the computer were not enough to gather 30 pieces of data in a reasonable time. The computer had 4 virtual core processes. This means only 4 parallel processes took place. Therefore, Amazon web services (AWS) instances were used. The instances act as a virtual machine. The virtual machine has many cores. This helped reduce the time it took to gather data. To transfer code between the home computer and the virtual machine created, docker images and containers were used. This made the transfer of code between different machines a lot easier, without the need to install any required libraries.

1.1.1 Overfitting

The hidden layer within the neural network is approximately 105,004 parameters. This is quite a lot of parameters. Therefore, there is a chance that overfitting can occur. Overfitting occurs when a neural network provides better predictions for the training set than it does for the testing data. The neural network has many free parameters. That includes weight and bias. These free parameters are adjusted to fit a function/model. For example, we have a free parameter A that needs to be adjusted to be directly proportional to variables x, y, and z in a given function.

However, parameter A is adjusted only proportional to x. This does not conclude that it is proportional to y, z. Thus, the free parameter fails to correctly generalise and act accordingly proportional the variables y, z. The same analogy can be applied in the neural network model. The given training data is used to train the model. The model performs well with the training data given. However, fails to predict data it has not seen before. This is defined as overfitting.



(Nielsen, 2009)

Figure 19: Overfitting

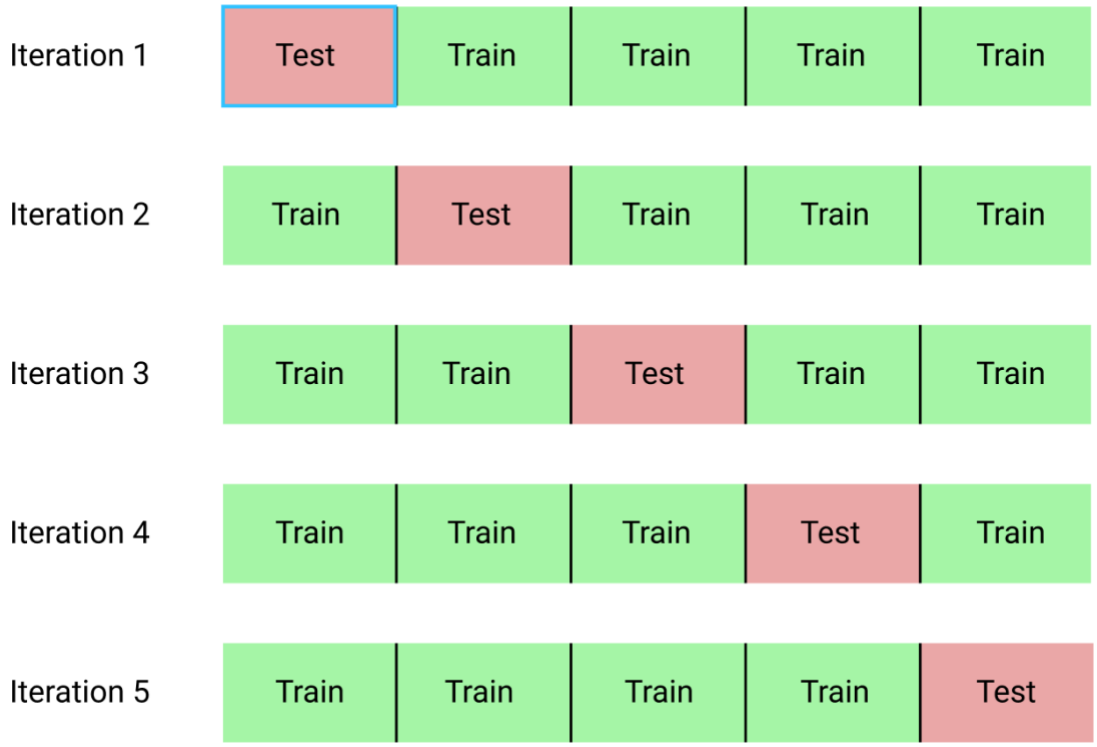
The two graphs shown in figure 19 demonstrate the concept of overfitting, where the loss of the training data is reducing, and the loss of the testing data is increasing. This indicates that the weights and biases in the algorithm are directly adjusted and correspond to the training data used however not the general function.

There are many techniques to prevent the problem of overfitting in a neural network. The standard technique used for neural networks is to use validation data. This does not necessarily get rid of overfitting, rather gives a general approximation of the performance of a neural network. By industry standards, when producing predictions for a neural network, it will often have 3 data sets: validation data, test data and training data. The validation data is used to simply tune the network's parameters. The problem with using only two sets of data: training data and test data is that the training data may have a lower loss (overfitting) than the test data. This is because learning parameters (learning rate, epochs) and parameters (weights, bias) are tuned to for the training data. This implies that the performance of the training data is not representative of the general function. The validation data introduced simply provides an unbiased evaluation of the neural network's performance while tuning the hyperparameters.

The definition of training data, test data and validation data are further explained below:

- Training data: “A set of examples used for learning, that is to fit the parameters of the classifier”
- Validation data: “A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network.”
- Test data: “A set of examples used only to assess the performance of a fully-specified classifier” (Ripley, 1996)

However, the validation data may still provide a biased performance of the neural network. Cross-validation is another technique for tuning parameters while measuring the performance of neural networks. The procedure is as follows:



(Shaikh, 2018)

Figure 20: illustrates the procedure for cross validation

The procedure has a single parameter k. k denotes the number of groups that the training data would be split into. This procedure is called k-fold cross validation, where the value for k is chosen it and used in reference for the name of the procedure. For example, $k = 5$ would become 5-fold cross validation. Following on from the example of $k = 5$. The training data would be divided into 5 equal parts and for each 5 iterations it will.

1. Shuffle the training data
2. Within the 5 groups, make use of one group as a test data set.
3. Combine the remaining groups as part of the training data set.
4. Use that training data for the neural network and then get loss of the test data.
5. Add the test loss to the list. This list is denoted as *evolution scores*

Once the procedure is finished, an evaluation of the performance of the neural network is made. This is done using the evaluation scores.

By iterating and using different test groups, the procedure consistently moves through the entire data set. As shown on figure 20. This provides a different list of losses. Once the average loss has been made, it provides an idea of the general performance of the neural network. The code for the k-fold procedure can be found in Appendix 1.

4. Experiment results

This section addresses the experiments detailed in the methodology. This describes the statistical results for each experiment and describes what the statistics show.

4.1 Neural networks: Experimental settings (1)

This experiment explores the effect of a learning parameter on the performance of the neural network. This performance is measured by the loss of the neural network. This effect of the learning parameter is tested with the training data of 3 dynamic systems: Newtons cooling law, Van der pol Oscillator and Laub Loomis.

Further details on this experiment can be found in the *neural network: Experiment setting (1)* explained in the methodology.

4.1.1 Learning rate

Appendix 2, 6, 9 for the data of Newtons cooling law, Van der pol and Laub Loomis respectively, show the effect of different learning rate on the performance of the neural network. The statistical averages on the table and the boxplot show that a lower learning rate generally performs best. However, when the learning rate becomes too small it shows that the

loss begins to increase. This is because a small learning rate takes a longer time to converge to the local minimum. Smaller learning rates, combined with a higher number of epochs, would eventually converge to a smaller loss.

4.1.2 Batch size

In appendix 3a, 7a and 10a for the data of Newtons cooling law, Van der Pol and Laub Loomis respectively are boxplots showing the distribution of 30 data loss items. These boxplots show the effect of different batch sizes. These boxplots illustrated that larger batch sizes did not perform well. In appendix 3a, the largest batch size such as 300 for Newtons cooling law performed the worst. This is because the neural network generalises worse with a larger batch size.

4.1.3 Number of epochs

In appendix 4, 8, 11 for the data of Newtons cooling law, Van der Pol and Laub Loomis respectively show boxplots and statistical averages for 30 data loss items. The boxplots illustrate that the higher the number of epochs the better the neural network performs. However, when the number of epochs becomes too high, the optimiser begins to move away from the local minimum. This is shown in appendix 4a, where the highest number of epochs provides a worse result than it does for a small number of epochs. This is linked to the learning rate the neural network has. This is further detailed in the *discussions* section.

4.1.4 Number of layers

Appendix 5 for the data of Newton's cooling law shows the effect of the number of nodes and layers on the neural network's performance. It is illustrated that a neural network with more nodes and more layers perform better.

4.2 Dynamics Systems: experimental (1) and (2)

This experiment evaluates the performance of the neural network. This is done by analysing the performance of the neural network with data it has not seen before (test data). In neural networks, different inputs of data provide a different performance. In context to this, the performance is analysed by exploring the effect of different input of samples simulations into the neural network. In this experiment, it is presumed that state variables that are closer to

samples of state spaces in the training data provide higher accurate results than state spaces further away from samples in the training data. For example, in Newton cooling law the training data would consist of samples simulations corresponding to an initial set of temperatures [1 – 60]. This means that each sample simulation would begin from a specific temperature. The test data consists of samples of state spaces that are not within the training data. It is presumed that an initial state temperature of 0.05 or 65.56 produces less accurate results than of a sample with an initial state temperature of 10.67 or 61.89. This is because the initial state temperatures (10.67, 61.89) are closer to the initial state samples [1 – 60]. In addition, the performance is also evaluated by looking at different inputs of time steps within the simulation samples in the test data. For example, the accuracy of the result at time step 0, compared to time step 5.

In summary the experiment evaluates the performance of the neural network in 2 ways:

- Accuracy of the results of each simulation sample in the test data
- Accuracy of results of each time step within sample simulations in the test data.

The accuracy/performance of the neural network is measured by looking at the loss.

The details of this experiment are further explained within in section *Dynamic systems: Experimental setting (1)* and *Dynamic systems: Experimental setting (2)* found within the *Methodology*.

This section is structured by the 6 dynamical systems: Newtons cooling law, Van der Pol Oscillator, Laub Loomis, Biological model, Bouncing ball and Spiking neurons. The performance of the test data will be discussed below.

4.2.1 Van der Pol Oscillator

This is the Van der Pol Oscillator discussed within *Chapter 3: Methodology*. The numerical simulation in comparison with the predictions made by the neural network model can be found in appendix 12, and 13 for the training data and test data respectively.

4.2.1.1 Accuracy of results of a simulation sample

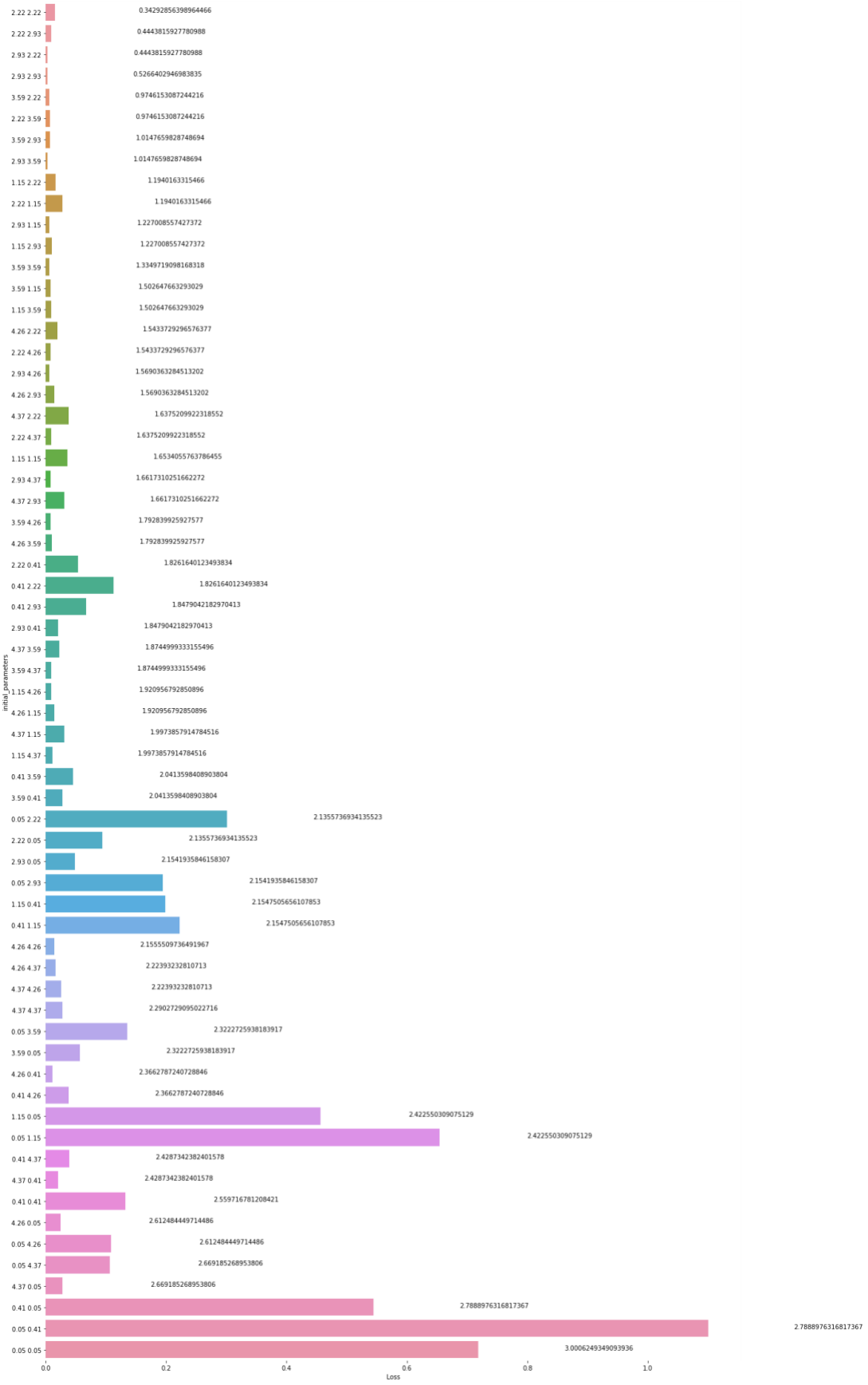


Figure 21: A bar graph showing samples of simulations and their loss for predictions of van der pol.

Figure 21 shows the bar graph where the x axis is the loss, and the y axis denotes the initial variable. The labels shown on the right-hand side of each bar measures the distance of a simulation sample within the test data from the samples in the training data. For example, the initial state $x = 2.22\, y = 2.22$ (first bar) has a distance of 0.345. The graph is sorted in ascending order of the distances. Before, beginning analysing the data it is bests to gain an understanding of what it means by the quantifier "distance".

4.2.1.2 "The Distance"

This additional section here is added to explain the quantifier "distance" and how it is measured. This is explained using the Van der Pol model. The training data has samples of simulations. Each simulation has a different initial state variable. For the Van der Pol model, we have 16 samples in the training data, and the initial parameters can be denoted as x-y value pairs (x, y). The 16 samples within the training data are plotted below.

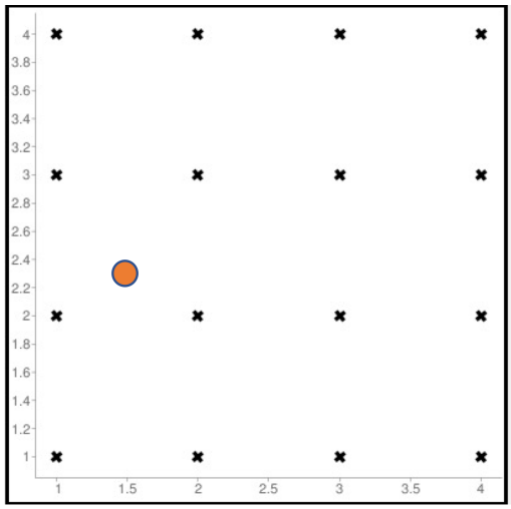


Figure 22: A graph showing the samples in the training data and a sample in the test data.

The crosses denote the samples within the training data and the red dot denotes a sample within the test data. We want to calculate the distance from the test sample in relation to the samples within the training data. This is done by applying a statistical distance (Mahalanobis distance). This allows us to take into account the variability between all of the samples within the training data.

Now back to analysing the accuracy of the results of a simulation sample. From figure 21, although the loss fluctuates as the distance ascends, it is reasonable to state that the initial state variable with a higher distance has a higher loss. The graph shows that generally the initial state variables with a high distance are situated at the bottom of the graph, whereas the lowest distances are situated at the top of the graph. This is shown by the initial variable $x = 0.05, y = 0.41$ $(0.05, 0.41)$. The initial variable (0.05, 0.41) has the second highest distance (2.789) and has the highest loss. Whereas the initial state variable (2.93, 2.93) is part of the lowest distances within the sample of test data and has a low loss. However, the graph fluctuates a lot. This is because they are initial variables which have the same distance but have a different loss. For example, $(0.4, 4.26)$ and $(4.26, 0.41)$ have equal distances (2.429). However, $(4.26, 0.41)$ has the lowest loss between the two. This perhaps is because the neural network learns better for a specific initial variable than it does for the other. For example, it might perform better on initial state x than it does for initial state y. Additionally, it may show that there are specific values that perform better for certain initial variables than it does for the other. These findings may suggest why the graph fluctuates a lot.

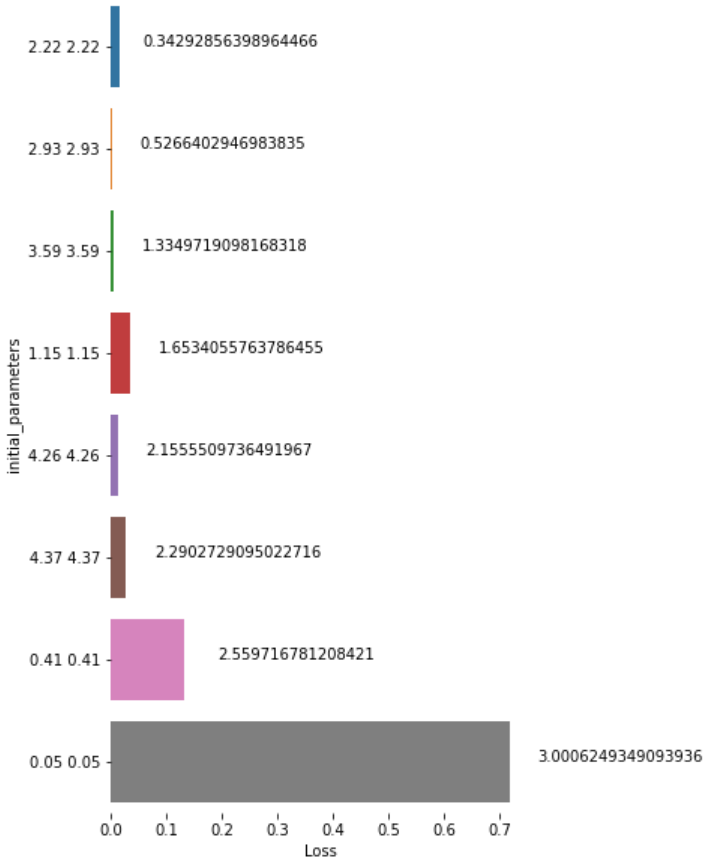


Figure 23: A bar graph showing samples of test data which do not have equal distances.

An additional graph is added, to support the idea that as the distances increases loss increases. This is done using a similar bar graph as figure 21, except this bar graph has removed all the samples with equal distances. This conveys the same information, as it did before: As the distance increases the loss also increases. However, there a distinct observation of (1.15, 1.15) which has a higher loss than the samples below. This further suggests that the neural network perhaps performs better for specific values for a certain initial variable than it does on the other.

Overall, the graph shows that

- A higher distance generally means a higher loss.
- The neural network model performs better for specific values corresponding to a specific state initial variable.

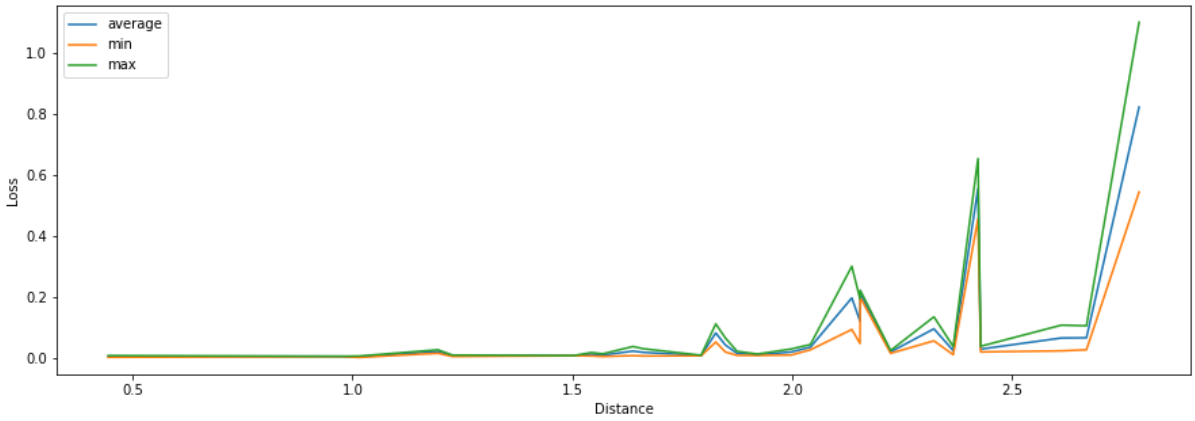


Figure 24: A line graph showing the samples with equal distances and their losses.

We want to focus more on the samples with equal distances. In this line graph, groups all the equal distances together and shows the average, minimum and maximum between the distances. This graph suggests that there is not much difference between the maximum loss taken from a group of equal distances and the average loss taken from a group of equal distances. This is the same information that is shown for the minimum loss taken from a group of equal distances. The graph shows that for some distances the loss is high. This further suggests that the neural networks perform better for some values for a specific initial variable. Additionally, the graph suggests that the higher the distance the higher the loss.

Overall, the line graph shows that

- There is not much of a difference between the maximum/minimum loss for a group of distances than the average.
- The higher the distance, the higher the loss.
- The neural network performs well for specific values corresponding to a specific initial variable.

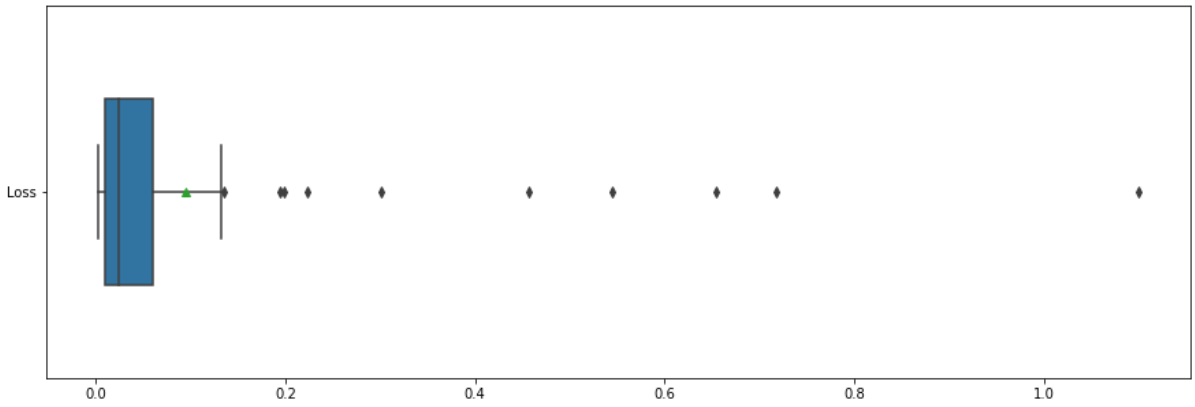


Figure 25: A boxplot showing the distribution of loss

Figure 25 shows the distribution of losses of each simulation's samples within the test data.

The box is short, this indicates that the (Interquartile range) IQR is small. This means there is less variability between the losses of each simulation sample. This suggests that the neural network performs generally well for all sample simulations despite the distance. The box plot has a positive skew. This means that most of the data is distributed closer to the minimum. Therefore, most sample simulations will have a small loss. However, there are 10 outliers. This means that 10 sample simulations within the test data does not perform as expected. From using figure 25, we can make a reasonable guess which outlier corresponds to the sample simulation. For example, (0.05, 0.41) shown in figure 25 is the highest loss. This corresponds to the highest outlier. (0.05, 0.41) is also has the highest distance. This further implies that the outliers shown within the boxplot mostly corresponds to samples with higher distances.

Overall, the boxplot shows that:

- The loss for each simulation is relatively consistent.
- The 10 outliers denoted in the graph mostly belong to the samples with higher distances.

4.2.1.3 Accuracy of results for a time step

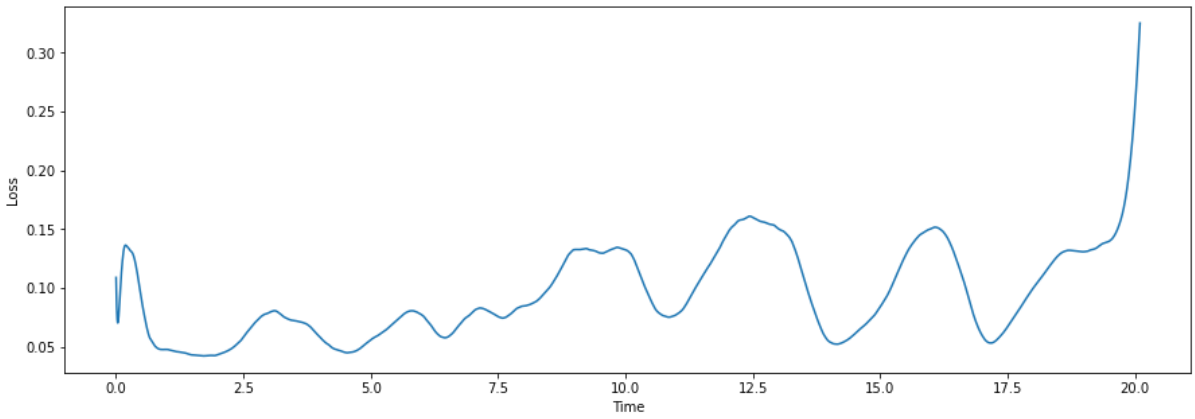


Figure 26: A line graph illustrating the average loss of samples for each time step in the test data

This graph shows that between each time step the accuracy fluctuates. The only distinguishable pattern is that the neural networks perform worse for the highest time step 20. These findings suggest that the neural network performs better for some time steps than it does for others. However, performance is worse for the highest time step.

Overall, the graph shows:

- There is inconsistency between the loss at each time step.
- The timestep at 20.0 performs the worst.

4.2.2 Newton's cooling law

This is Newton's cooling law discussed within *Chapter 3: Methodology*. The numerical simulations in comparison with the predictions made by the neural network model can be found in appendix 14a, and 14b for the training data and test data respectively.

4.2.2.1 Accuracy of results of a simulation sample

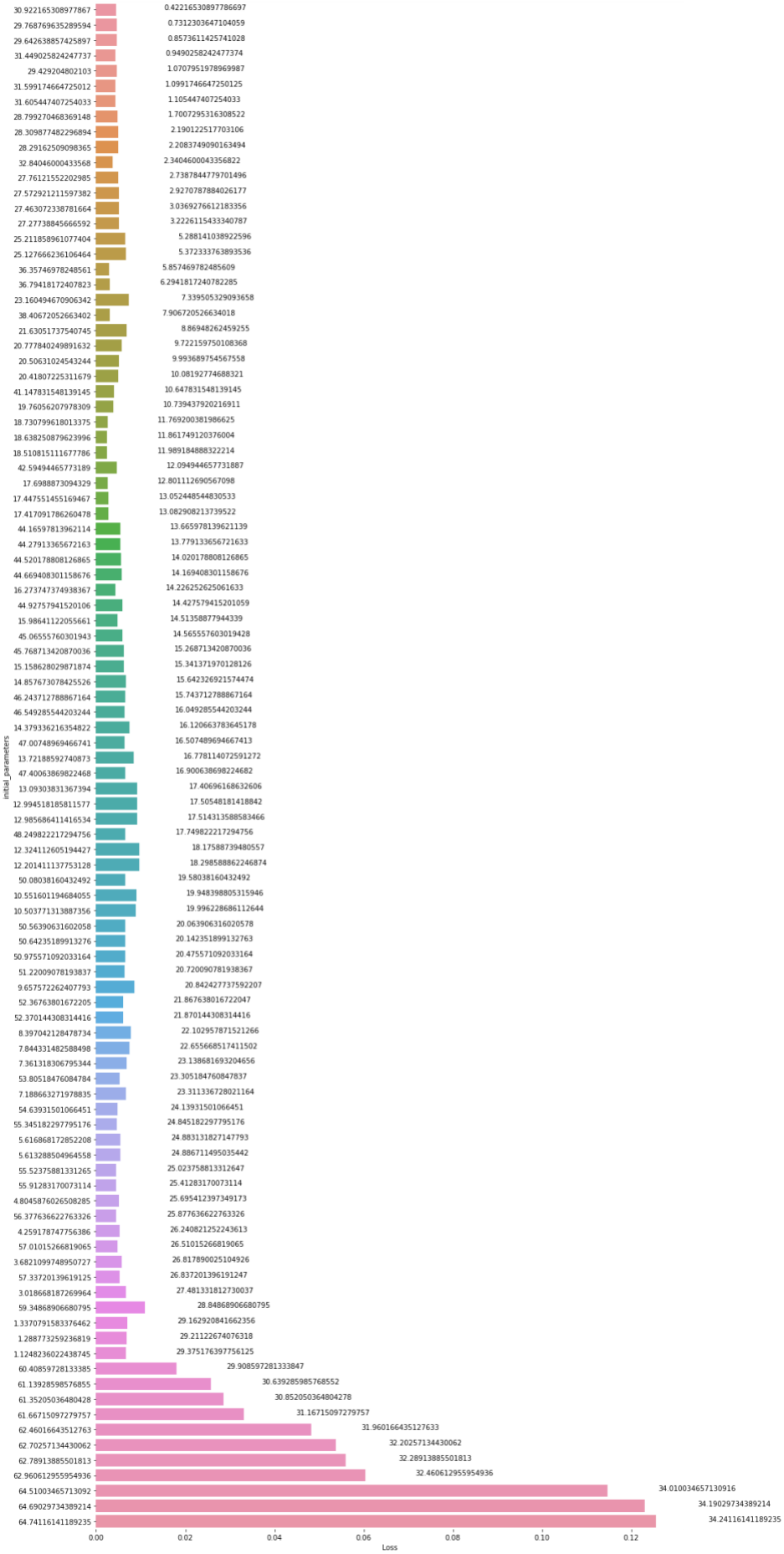

initial_parameters
Loss
0.00
0.02
0.04
0.06
0.08
0.10
0.12
30.922165308977867 0.42216530897786697
29.768769635289594 0.7312303647104059
29.642638857425897 0.8573611425741028
31.449025824247737 0.9490258242477374
29.429204802103 1.0707951978969987
31.599174664725012 1.0991746647250125
31.605447407254033 1.105447407254033
28.799270468369148 1.7007295316308522
28.309877482296894 2.190122517703106
28.29162509098365 2.2083749090163494
32.84046000433568 2.3404600043356822
27.76121552202985 2.7387844779701496
27.572921211597382 2.9270787884026177
27.463072338781664 3.0369276612183356
27.27738845666592 3.2226115433340787
25.211858961077404 5.288141038922596
25.127666236106464 5.372333763893536
36.35746978248561 5.857469782485609
36.79418172407823 6.2941817240782285
23.16049467090634 7.339505329093658
38.40672052663402 7.906720526634018
21.63051737540745 8.86948262459255
20.777840249891632 9.722159750108368
20.50631024543244 9.993689754567558
20.41807225311679 10.081927746883 21
41.147831548139145 10.647831548139145
19.76056207978309 10.739437920216911
18.730799618013375 11.769200381986625
18.638250879623996 11.861749120376004
18.510815111677786 11.989184888322214
42.59494465773189 12.094944657731887
17.6988873094329 12.801112690567098
17.447551455169467 13.052448544830533
17.417091786260478 13.082908213739522
44.16597813962114 13.665978139621139
44.27913365672163 13.779133656721633
44.520178808126865 14.020178808126865
44.669408301158676 14.169408301158676
16.273747374938367 14.226252625061633
44.92757941520106 14.427579415201059
15.98641122055661 14.51358877944339
45.06555760301943 14.565557603019428
45.76871342087003 6 15.268713420870036
15.158628029871874 15.341371970128126
14.857673078425526 15.642326921574474
46.243712788867164 15.743712788867164
46.549285544203244 16.049285544203244
14.379336216354822 16.120663783645178
47.00748969466741 16.507489694667413
13.72188592740873 16.778114072591272
47.40063869822468 16.900638698224682
13.09303831367394 17.40696168632606
12.994518185811577 17.505481814188842
12.985686411416534 17.514313588583466
48.249822217294756 17.749822217294756
12.324112605194427 18.17588739480557
12.201411137753128 18.298588862246874
50.08038160432492 19.58038160432492
10.551601194684055 19.948398805315946
10.503771313887356 19.996228686112644
50.56390631602058 20.063906316020578
50.64235189913276 20.142351899132763
50.975571092033164 20.475571092033164
51.22009078193837 20.720090781938367
9.657572262407793 20.842427737592207
52.36763801672205 21.867638016722047
52.370144308314416 21.87014430831 4416
8.397042128478734 22.102957871521266
7.844331482588498 22.655668517411502
7.361318306795344 23.138681693204656
53.80518476084784 23.305184760847837
7.188663271978835 23.311336728021164
54.63931501066451 24.13931501066451
55.345182297795176 24.845182297795176
5.616868172852208 24.883131827147793
5.613288504964558 24.886711495035442
55.52375881331265 25.023758813312647
55.91283170073114 25.41283170073114
4.8045876026508285 25.695412397349173
56.377636622763326 25.877636622763326
4.259178747756386 26.240821252243613
57.01015266819065 26.51015266819065
3.6821099748950727 26.817890025104926
57.33720139619125 26.837201396191247
3.018668187269964 27.481331812730037
59.34868906680795 28.84868906680795
1.3370791583376462 29.162920841662356
1.288773259236819 29.21122674076318
1.1248236022438745 29.375176397756125
60.40859728133385 29.908597281333847
61.13928598576855 30.639285985768552
61.35205036480428 30.852050364804278
61.66715097279757 31.16715097279757
62.46016643512763 31.960166435127633
62.70257134430062 32.20257134430062
62.78913885501813 32.28913885501813
62.960612955954936 32.460612955954936
64.51003465713092 34.010034657130916
64.69029734389214 34.190297343892 14
64.74116141189235 34.24116141189235

Figure 27: A bar graph illustrating the average loss of samples in the test data

In the graph the x axis denotes the temperature, and the y axis denotes the loss. The labels on the right-hand side of the bar graph show the distance of each sample from the simulation's samples from the training data.

This figure conveys the same information as it did for figure 21: As the distances increase, the loss increases. Therefore, it's reasonable to assume a sample simulation has a higher loss when a sample initial variable in test data exceeds the samples initial variables in the training data. This is shown by the initial state temperature 64.741 which has the highest distance (34.241). The initial temperature, which has the lowest the loss, is generally situated at the top, indicating it has a low distance. However, the graph between the highest distances and the lowest distances does fluctuate. This suggests that for some values applied to a specific initial variable performs better than it does for the other. The reason for this may be because the neural network performs better for some specific values for an initial variable.

Overall, the information on the graph shows:

- In general, the higher the distance, the higher the loss of the simulation sample.
- The neural network performs best for some values for initial variables than it does for other values.

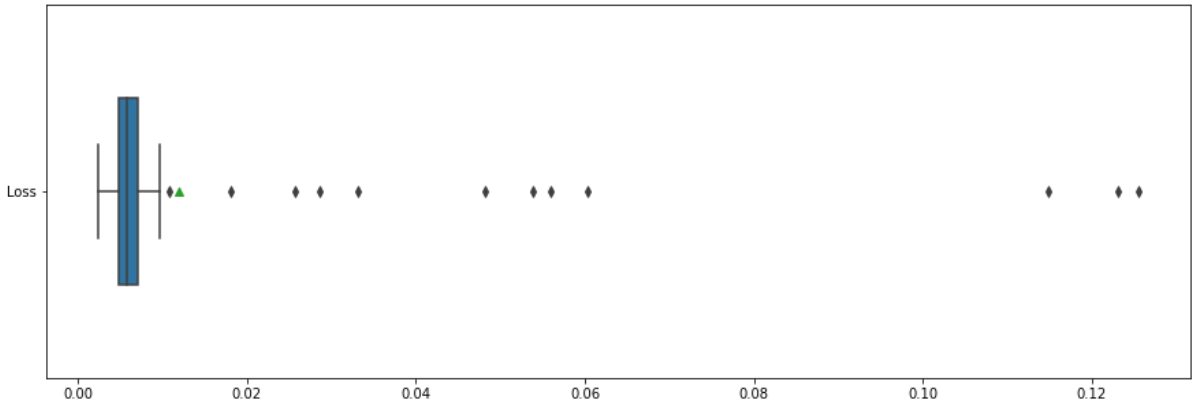


Figure 28: illustrates the distribution of loss for samples of simulations within the test data.

The box for this simulation is generally small. This indicates that the IQR is small, implying that there is less variability in the distribution of loss for a sample simulation. This suggests that samples with the test data have similar losses. The distribution of the data appears to be normal. This means that the loss of sample simulation either falls at the highest end or lowest

end of the loss. The mean of the data is approximately 0.015. There are around 12 outliers. Intuitively we can estimate which outliers correspond to a simulation sample. The highest outlier, which is approximately 0.125 corresponds to the simulation sample with an initial state variable of 64.742. This is because it is the highest loss within the bar graph and boxplot. The initial temperature 64.762 is also the highest distance. This suggests most of the outliers shown within the boxplots belong to samples with the highest distances.

Overall, from the boxplot it shows:

- The loss of each simulation sample is generally consistent
- Most of the outliers belong to test samples where the distance is higher.

Unlike the van der Pol model, there is no equal distance presented within the sample of simulations. As there is only one initial variable within Newton cooling law. Therefore, a graph which groups the equal distances is disregarded.

4.2.2.2 Accuracy of results for a time step

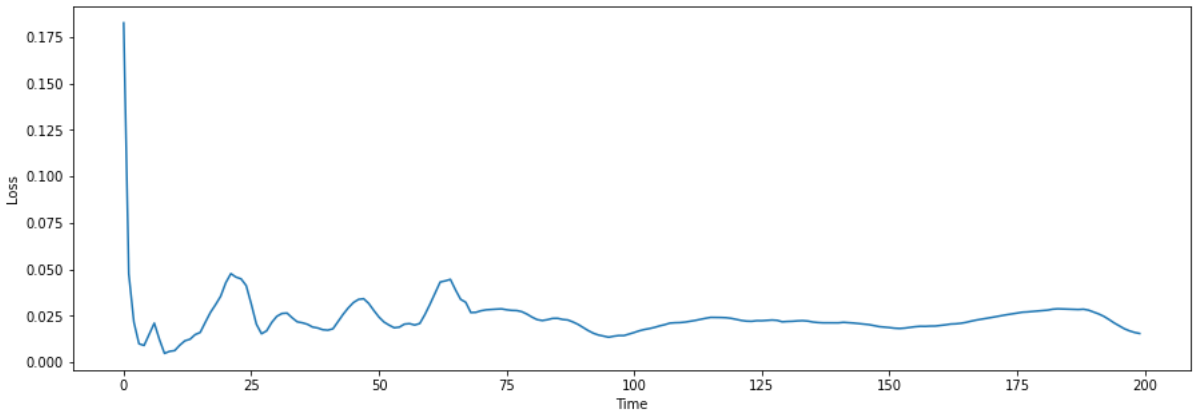


Figure 29: shows the average loss of a time step for the simulation's samples within the test data.

The graphs illustrate that as the time increases the loss decreases. This is shown as time step 0 has the highest loss. There are multiple samples all derived from different initial state variables. Each simulation sample has a different pattern in terms of how the temperature variable evolves. Perhaps the inconsistency between the loss for a time step is because the neural network is failing to learn the information captured at time step 0. This may be due to the many sample simulations within the neural networks which convey differing patterns at time steps 0. This makes it difficult for the neural network to learn.

Overall, this graph shows that

- generally, the further the time steps, the lower the loss.
- at time step 0, we have the highest loss.

4.2.3 Laub Loomis

This is Laub Loomis discussed within *Chapter 3: Methodology.* A sample of the training data and test data in comparison with the predictions made by the neural network can be found in appendix 15a, 15b respectively.

4.2.3.1 Accuracy of results of a simulation sample

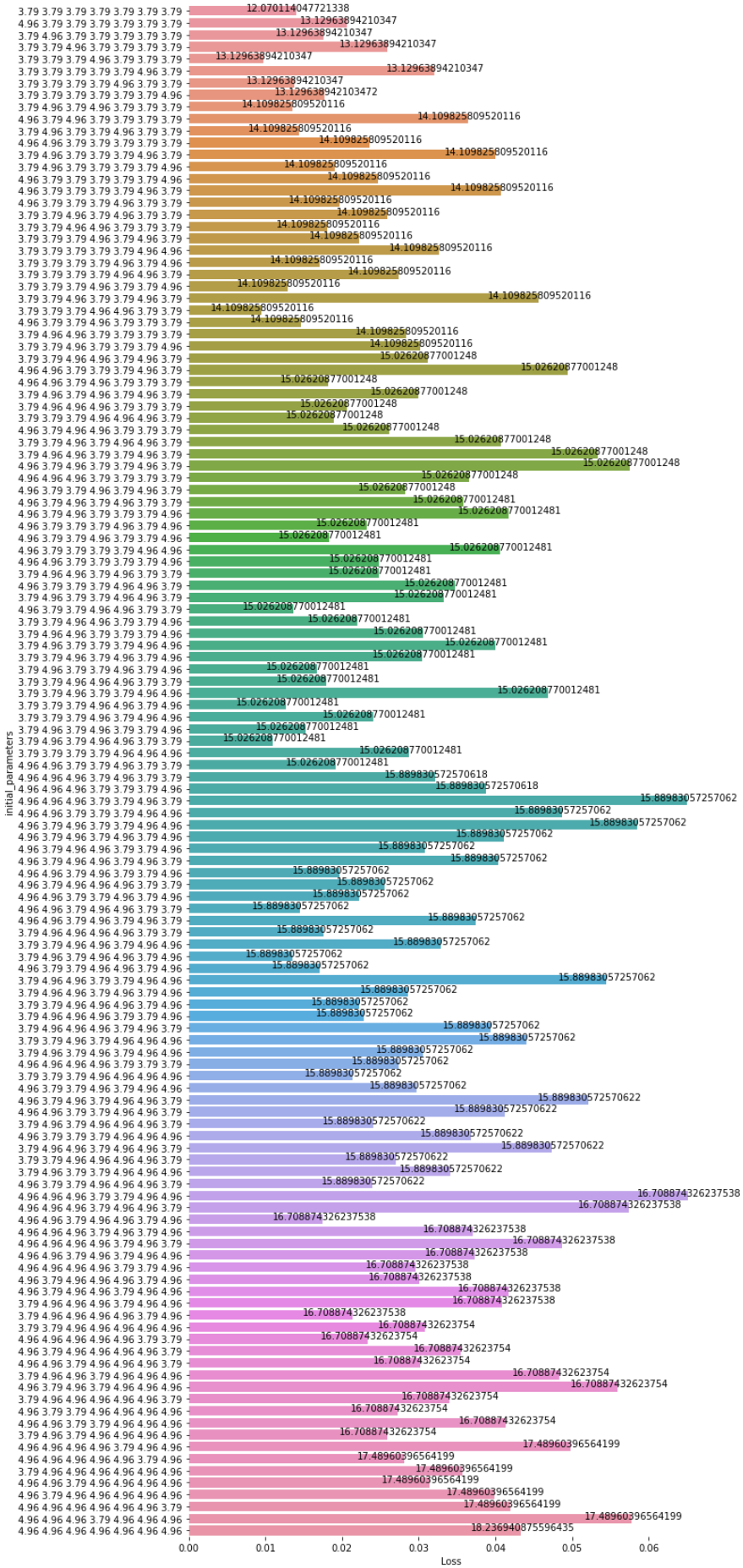

initial_parameters
Loss
0.00
0.01
0.02
0.03
0.04
0.05
0.06

Figure 30: Demonstrates the loss of each simulation sample. The labels on the right-side of the bar are the distances.

The information shown on the bar graph conveys a different information compared to the figures above for van der pol model and newtons cooling law. There is no consistent pattern that can be observed of how the distance of the sample affects the loss. For example, $(x_1 = 3.76, x_2 = 3.76, x_3 = 3.76, x_4 = 3.76, x_5 = 3.76, x_6 = 3.76, x_7 = 3.76)$ has a distance (12.07) and is alow distance. However, $(x_1 = 3.76, x_2 = 2.3, x_3 = 3.76, x_4 = 4.96, x_5 = 3.76, x_6 = 3.76, x_7 = 3.76)$ with a distance of (13.13) performs better. This suggest that the neural networks perform better for some values for specific initial state variables than it does on the other. The graphs also shows that samples with the same distances have different performances. This further suggests the idea that the neural networks perform best for certain initial variables for certain values.

Overall, the graph indicates that:

- There is an inconsistency between the distance of a sample and the loss. This may be due to the neural network performing better for some values for a specific initial variable.

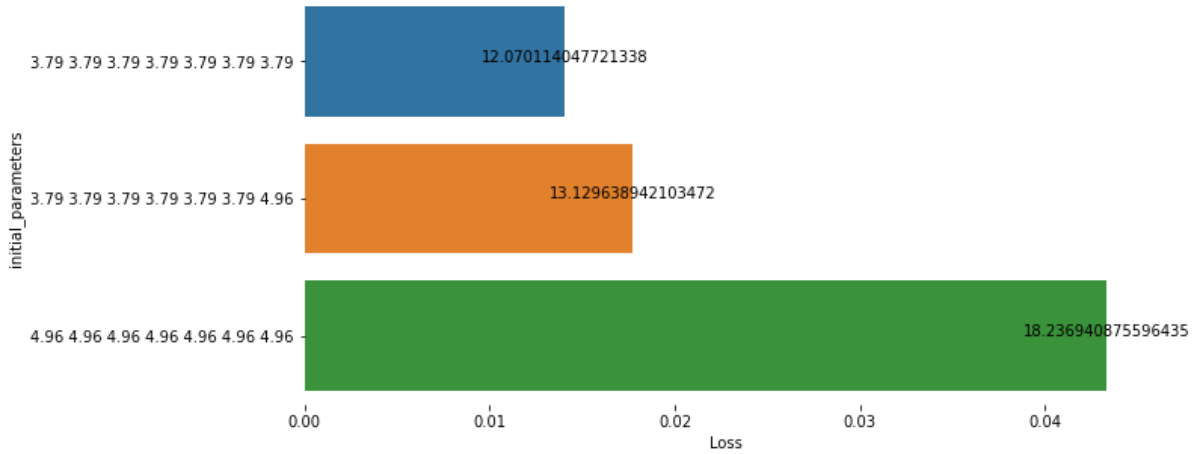


Figure 31: A bar graph showing samples which do not have equal distances.

This is similar to the figure demonstrated in figure 30. However, the initial variables with equal distances have been removed. This graph illustrates that the higher the distance, the higher the loss. This is a lot different from the information conveyed in the bar graph. However, we are using the same samples within the test data, except samples with equal distance have been removed. This further supports the idea that the neural network performs best for some values corresponding to initial state variables than it does for others.

Overall, the graph shows that:

- The higher the distance, the higher the loss.

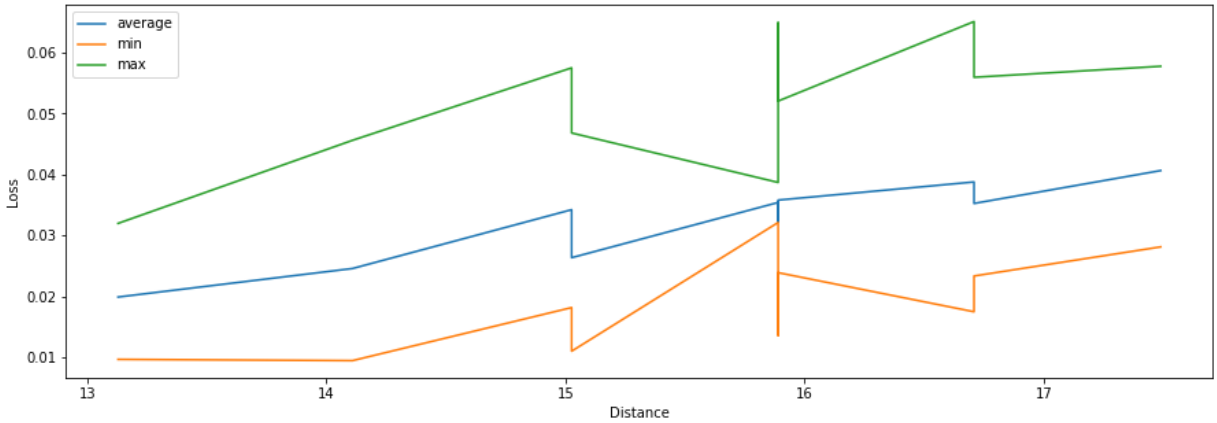


Figure 32: A line graph showing the samples with equal distances and their losses.

This is a graph showing all the samples with equal distances and the corresponding loss shown on the x axis. The graph indicates that lower distances have a low loss. However, there are places where the graph does flatulate a lot between the low and high distances of samples. This suggests that the neural network work predicts better for some values for an initial variable than it does for another.

Overall, the graph shows that:

- Samples with a low distance have a lower loss.
- The neural network predicts better for some values for an initial state variable.

4.2.3.2 Accuracy of results for a time step

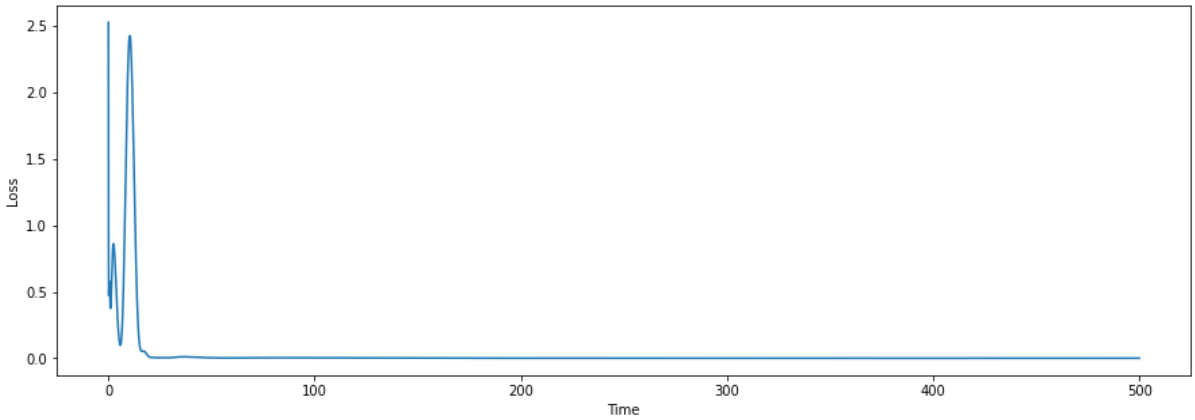


Figure 33: illustrates the average loss at each time step for all simulation samples.

This graph shows that the time steps close to 0 have a higher loss. This suggests that the neural network performs better at further time steps than it does at time steps at the beginning. Essentially this suggests that for each sample simulation there is more inconsistency in the behaviour at the lower time steps than it is for further time steps. This is perhaps because there are many executions (501). At a certain time step the behaviours of each simulations starts to converge to a certain point. For example, as $t \longrightarrow \infty$ the variables of Laub Loomis $(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ converge to a certain value. This holds true for all sample simulations no matter the initial variables. However, when it starts to converge is dependent on the initial variable, while some samples converge earlier others will converge later. Nonetheless, convergence sequence of data allows the neural network to consistently perform better for predicting results, as the patterns between samples simulation varies less.

To support our assumption, we take a look at how the variables evolve in the Laub Loomis by looking at a sample with initial variables (4.96, 4.96, 4.96, 4.96, 4.96, 4.96, 4.96) in the training data. This shows that there is a relationship between how variables evolve and the loss at each time step. Each sample simulation variable evolves into a pattern. In this case, it converges to a point. This pattern is less complex. Therefore, makes it easier for the neural network to generalise.

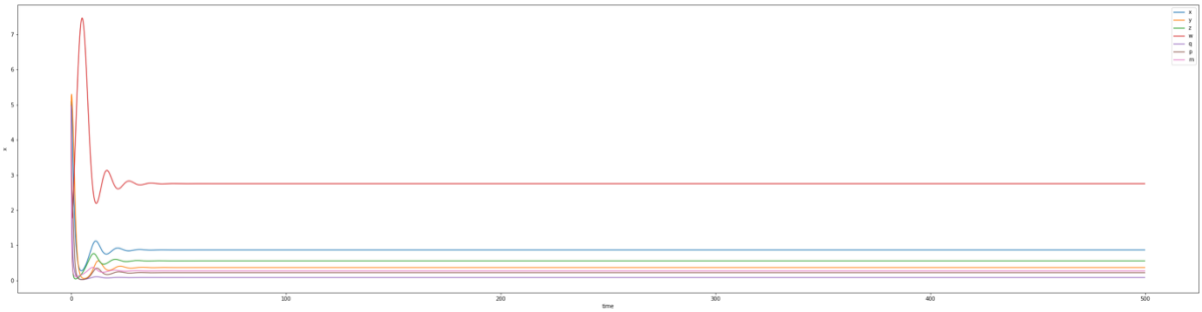


Figure 34: illustrates the evolution of variables for Laub Loomis as time increases

Overall, the graph shows that:

- As time increases the better performance of the neural network

4.2.4 Biological model

This is the Biological model discussed within *Chapter 3: Methodology.* A sample of the training data and test data in comparison with the predictions made by the neural network can be found in appendix 16a, 16b respectively.

4.2.4.1 Accuracy of results of a simulation sample

The loss of simulation samples cannot be shown within one page. However, it can be found on appendix 19.

The loss of simulation samples fluctuates a lot. However, it broadly shows that the higher the distance the worse the neural network performs. It is shown that samples with high distances are situated at the top have a much lower loss than the sample distances shown at the bottom. For example, the sample with initial variables $x_1 = 1.02, x_2 = 1.02, x_3 = 1.02, x_4 = 1.02, x_5 = 1.02\ , x_6 = 1.02\ , x_7 = 1.02, x_8 = 1.02, x_9 = 1.02$ $(1.02, 1.02, 1.02, 1.02, 1.02, 1.02, 1.02, 1.02, 1.02)$ with a distance of 5.994 has a lower loss than $(1.04, 1.04, 1.04, 1.04, 1.04, 1.04, 1.04, 1.04, 1.04)$ with a distance of 11.988. The graphs further illustrates that they are initial variables which have the same distance but different losses.

For example, $(\mathbf{1.04}, \mathbf{1.02}, 1.02, 1.02, 1.02, 1.02, 1.02, 1.02, 1.04,)$ and $(\mathbf{1.02}, \mathbf{1.04}, 1.02, 1.02, 1.02, 1.02, 1.02, 1.02, 1.04,)$ have the same distance (9.516). However, have different losses. This suggests the neural network for this model, learn values for specific initial variable better than it does for the other.

Overall, the bar graph shows that:

- Generally, an initial variable with higher distance has a higher loss.
- The neural network model performs well on certain values for an initial variable than it does for the other.

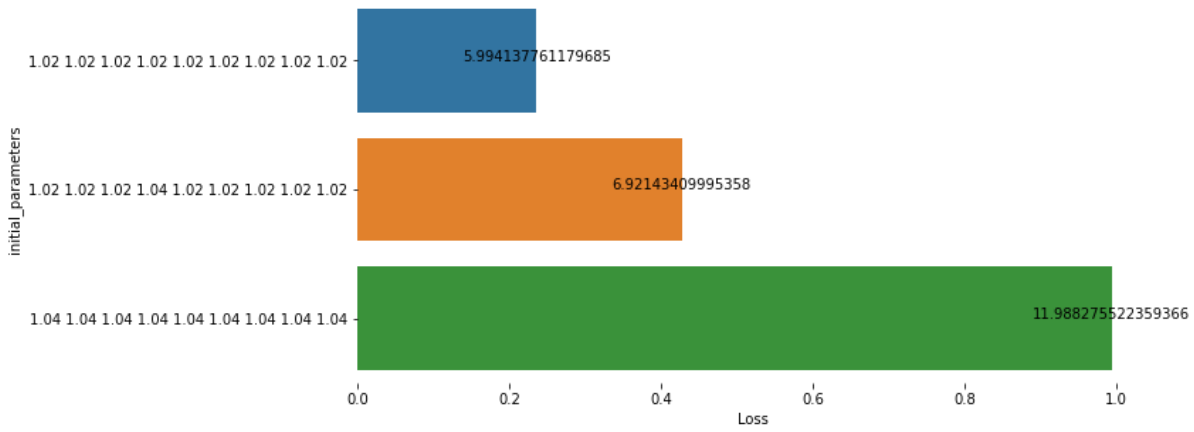


Figure 35: A bar graph showing samples which do not have equal distances.

This graph is similar to graph in appendix 19. However, it removes all the samples of simulations with equal distance. The information conveyed here is that the higher the distance the higher the loss.

Overall, the graph shows that:

- The higher the distance, the higher the loss.

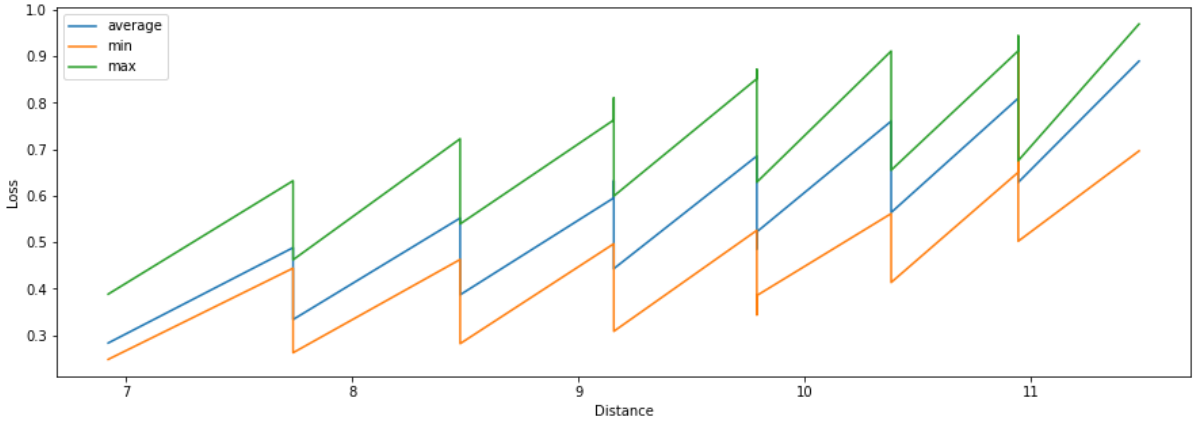


Figure 36: A line graph showing the samples with equal distances and their losses.

This line graph calculates the average, minimum and maximum for a group of equal distances. A particularly interesting property is that the graph shows that the relationship between the distance and loss is relatively the same for the minimum, maximum and the average. The graph is structured in a way where some distances have low points, and some distances have higher points. This produces a particular shape within the graph, where certain distances are low then go up. This further suggests that the neural network works better for some specific values for a particular initial variable than it does for others.

Overall, the graph indicates:

- The neural network performs well for some specific values corresponding to an initial variable than it does for another.

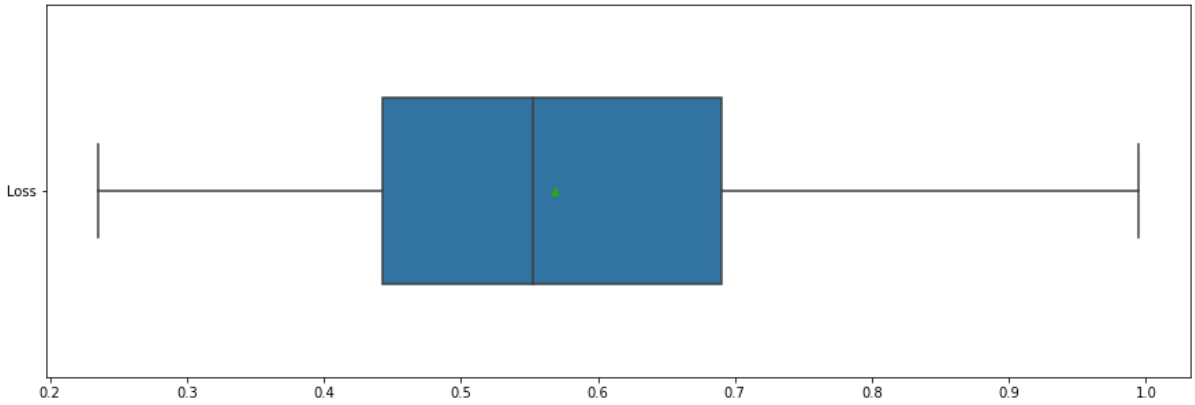


Figure 37: boxplot illustrates the distribution of losses for samples in the test data.

The box is long compared to other dynamic systems, implying the IQR is higher. This suggests that there is more variability in the loss for each sample. This implies that the neural network provides different accuracy for certain sample simulations. The boxplot has a positive skew. This means that most of the data is distributed at the minimum end of the loss.

Overall, the boxplot shows that:

- There is more variability in the loss for each sample.
- The distribution of data is situated at the lower end of the loss.

4.2.4.2 Accuracy of results for a time step

This shows the accuracy of results for the time step.

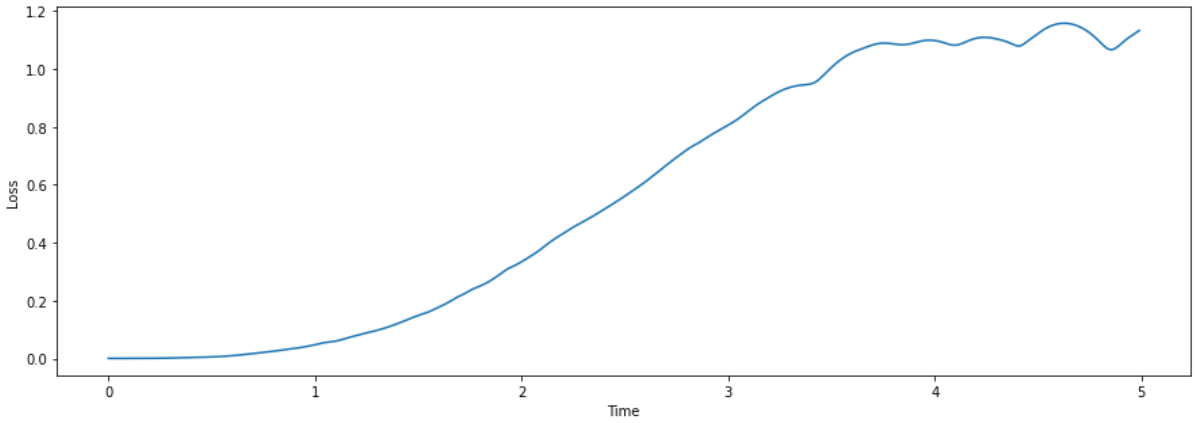


Figure 38: illustrates the average loss at each time step for all simulation samples.

Figure 38 illustrates that as time increases the loss also increases. This suggests that the neural network is failing to capture information at higher time steps. This may suggest that there is an inconsistent pattern between the simulation samples which differ as the time increases. This

makes it hard for the neural network to adjust its weight and bias to capture relationships of each sample of simulation.

Overall, this graph illustrates:

- As the time increases the loss increases.

4.2.5 bouncing ball

This is the bouncing ball hybrid automaton discussed within *Chapter 3: Methodology*. The numerical simulation in comparisons with the predictions made by the neural network model can be found in appendix 17a, and 17b for the training data and test data respectively.

4.2.5.1 Accuracy of results of a simulation sample

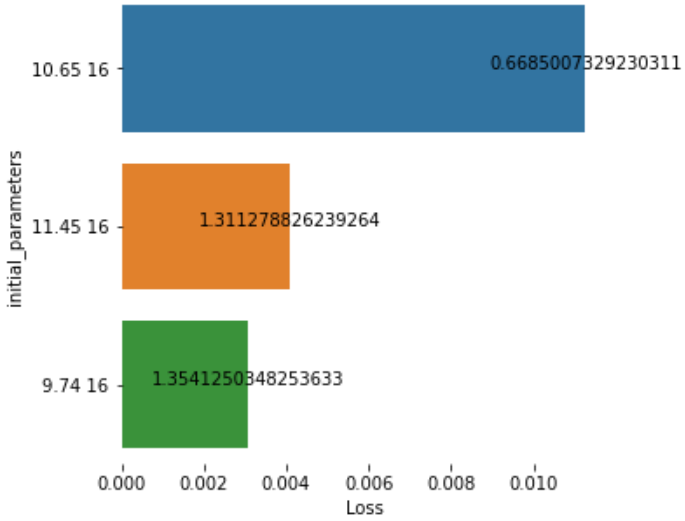


Figure 39: Demonstrates the loss of each simulation sample. The labels on the right-side of the bar are the distances.

Figure 39 is a bar graph showing the loss of each simulation sample within the test data.

This conveys a different information than it did for the other models in that it shows that the lower the distance of a sample simulation the higher the loss. However, the number of samples within the test data is not valid enough to make this statement. The reason for this result could just be because the neural networks perform better for certain values for certain initial variables.

Overall, the graphs show that.

- The neural network performs better for certain values corresponding to an initial variable than it does for the other.

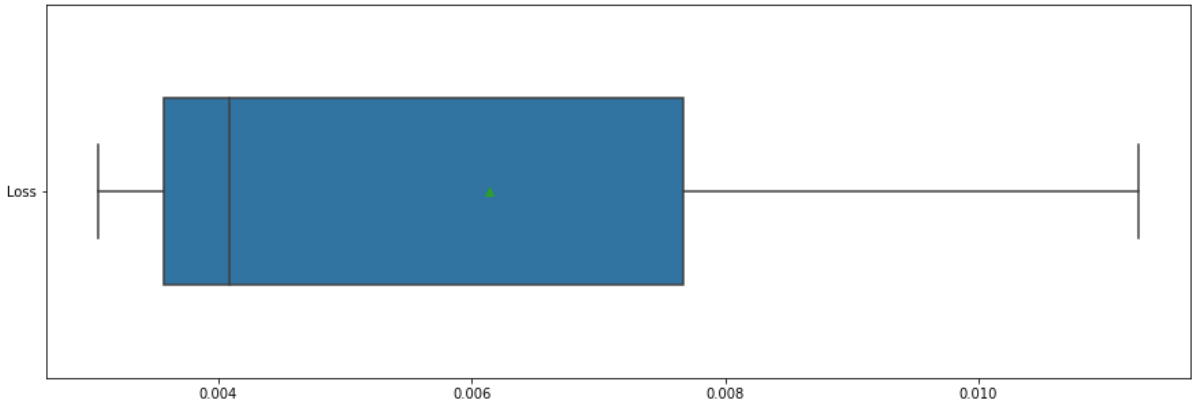


Figure 40: The distribution of loss for samples of simulations

The box illustrated on the figure 40 is long, this means the IQR is high. This information tells us that the distribution of data varies a lot. This suggests that neural network performance varies for each simulation sample. The data distribution has a positive skew. This implies that the distribution of our data is situated at the lower end of the loss. This implies the neural network provides a lower loss for most simulation samples.

Overall, the boxplot shows that:

- The distribution of data varies a lot.
- The distribution of data is at the lower end of the loss.

There are no samples simulation which have equal distances. Therefore, the line graph illustrating the equal distances is disregarded.

4.2.5.2 Accuracy of results for a time step

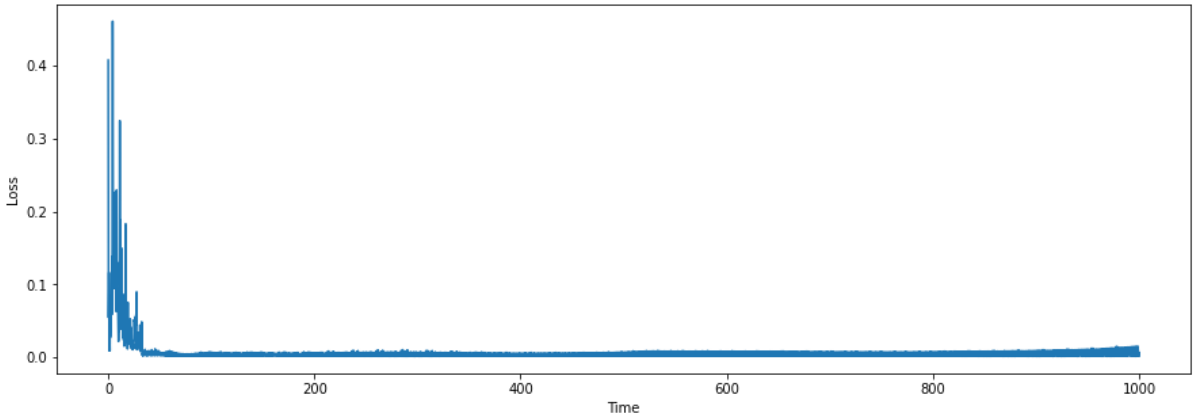


Figure 41: illustrates the average loss at each time step for all simulation samples.

Figure 41 shows that time steps closer to time step 0 and including time step 0 have the highest loss. Whereas the higher time steps have a lower loss. The reason for this is perhaps because there is a pattern within each sample which may be hard for the neural network to capture at the beginning, then it is at the end.

Overall, this graph shows that:

- Simulations closer to 0 have a higher loss.
- A higher time step has a lower loss.

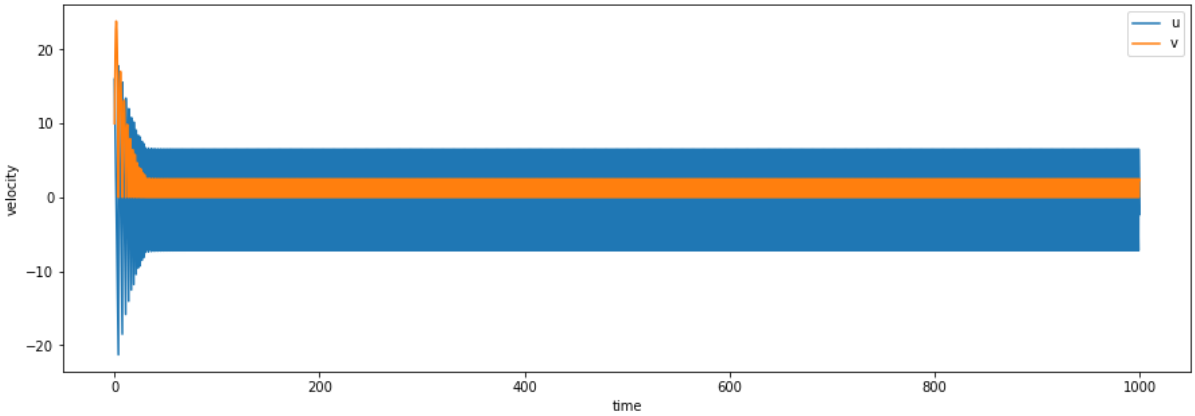


Figure 42: Illustrates of how variables in u, v evolves increase in the bouncing ball hybrid system

This graph shows how variables evolve in the bouncing ball. The hybrid system converges to a certain behaviour. The behaviour is called the zeno behaviour briefly discussed in the theoretical framework. The neural networks capture this behavioural pattern better than it does for the pattern in the beginning. This explains why the neural network performs worse at the beginning.

4.2.6 Spiking neurons

This is the spiking neurons hybrid automaton discussed within *Chapter 3: Methodology*. The numerical simulation in comparison with the predictions made by the neural network model can be found in appendix 18a, and 18b for the training data and test data respectively.

4.2.6.1 Accuracy of results of a simulation sample

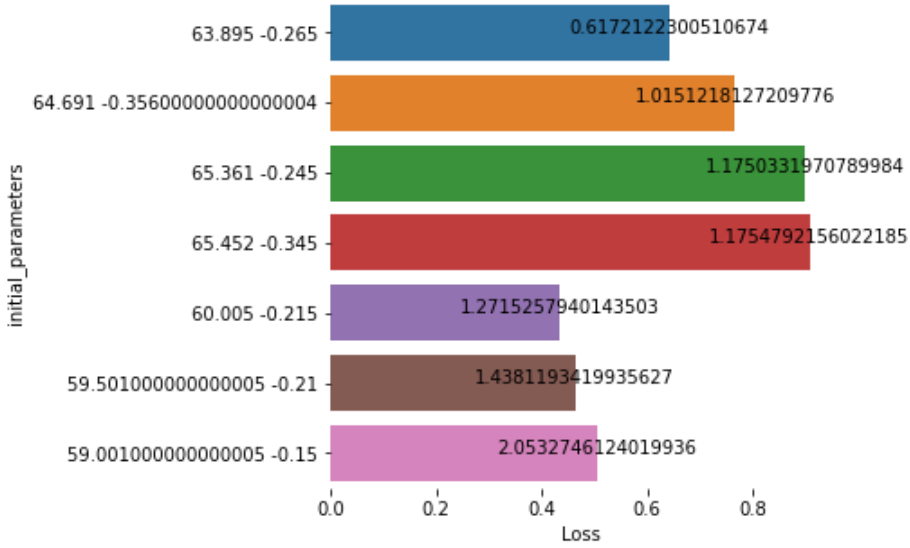


Figure 43: Demonstrates the loss of each simulation sample. The labels on the right-side of the bar are the distances.

The bar graph shows that higher distances have a lower loss. This is shown in the graph, where the higher distances are situated at the bottom and have a lower loss. For example (59.001, -0.15) and (59.501, -0.21) and (60.005, -0.215) have higher distances but lower losses. This implies that either higher distances perform better or that the neural network just performs better for different simulation samples. Essentially there are not enough sample simulations to state that a higher distance implies a lower loss. A test data with more samples simulations would have provided more accurate results to generalise how a higher distance impacts the loss. Therefore, we can only say that the neural network performs better for some specific values for a certain initial variable than it does for the other.

Overall, the graph shows that:

- Neural networks perform better for specific values corresponding to a specific initial variable than they do for the other.

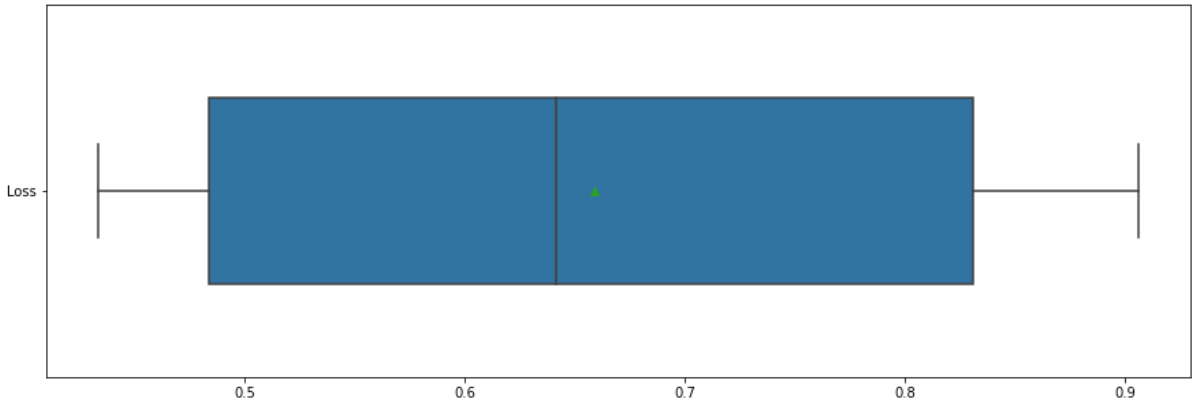


Figure 44: The distribution of loss for samples of simulations

The box illustrated on the boxplot is wide. This implies the IQR is high. This means that the distribution of data varies a lot. This implies that the neural network gives a different performance for different samples of simulations.

Overall, the boxplot shows that:

- The distribution of the data varies a lot.

No sample simulation within the test data has an equal distance. Therefore, the line graph showing the equal distance and the loss is disregarded.

4.2.6.2 Accuracy of results for a time step

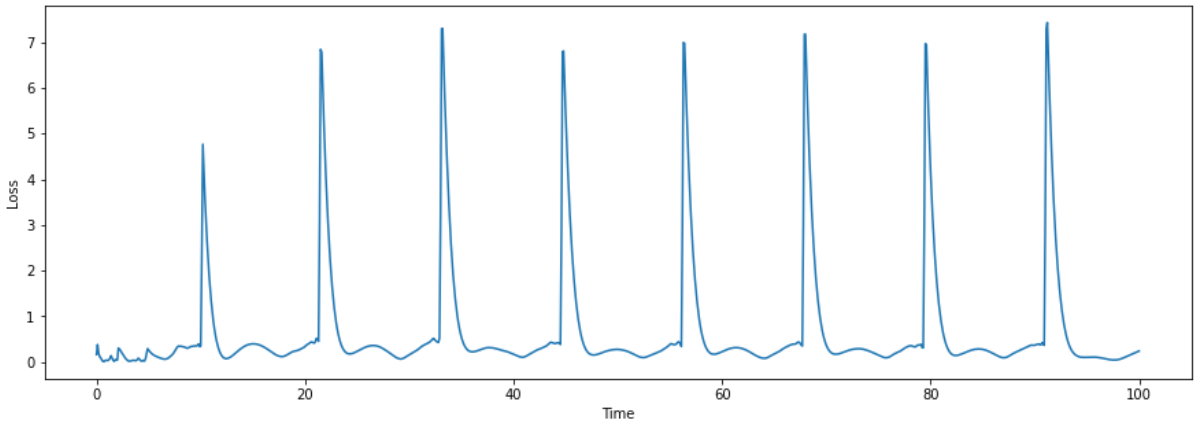


Figure 45: Illustrates the average loss at each time step for all simulation samples.

The graph shows that the loss for each time step is low at certain time steps, then high at others. The patterns shown in the graph may be because the neural network is capturing certain patterns in the behaviour at certain time steps than it does for others.

To further support this assumption, we look at how variables evolve in sample simulation u = 10, v = 16 (10,16). It appears that when there is a peak as variable v evolves through time the loss becomes higher.

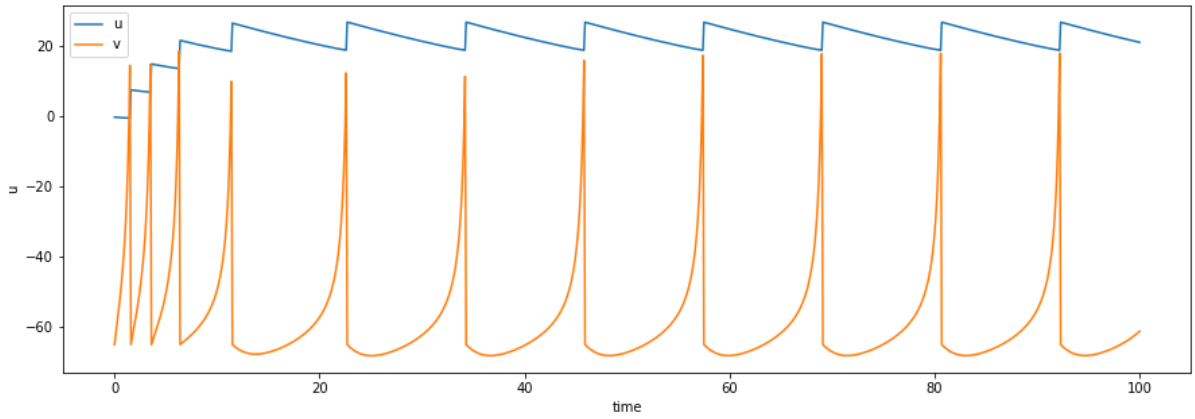


Figure 46: illustrates how variables u and v evolve with time in the hybrid system of spiking neurons

Overall, the graphs show that:

- For certain time steps we have a high loss.
- The high losses are due to the peaks exhibited on variable v as it evolves through time.

5. Discussion

This section expands on the reason for the *results* found in the experiments. This justifies the reason for the assumptions made while linking back to the research in the *theoretical framework* to explain it.

5.1 Neural networks: experimental setting (1)

In the *Neural networks: Experimental settings (1)* in the result section we found that.

1. Smaller learning rates perform better
2. The higher the number of epochs, the better the performance of the neural network

1. A smaller learning rate is performed better as it provides smaller steps to allow the cost function ($c(w,b) \approx 0$) to converge to the minimum. A learning rate which is too large may converge away from the local minimum. However, a smaller learning rate would need more steps to eventually converge to the local minimum.

2. In addition, the experiment found that the greater the number of epochs, the lower the loss becomes. However, too many epochs may lead to overfitting. The number of epochs which provided a lower loss may not have worked well for the test data. The attempt here is to make our neural network generalise the function. Too many epochs would lead to adjustments in the weights and biases that are suited directly to the training data and not the general function.

5.2 Dynamic systems: Experimental settings (1) and (2)

In the experiment of *Dynamic systems: experimental (1) and (2)* in the *results* section we found that:

1. A neural network performs better for some initial parameters than it works for the other.
2. A neural network performs better for some time steps.
3. In general, the samples with lower distances performed better.

1. This assumption is clearly evident as equal distances provide the different performances. The neural network has many parameters (weight and bias) to generalise a function. This project attempts to generalise many samples of a dynamic system. A given sample can be thought of as a function itself as variables within each sample evolve differently. There may be some initial values for a specific variable that has a consistent pattern in the way the variables evolve in each sample. This consistency makes it easier for the neural network to generalise.

2. Theoretically, the loss for each time step within the test data should be consistent. This is because the samples in the test data have the same number of executions/stop time as the sample in the training data. This means we are not exploring further time steps beyond what's in the training data, rather simulating a different state space. However, in practice variables within a sample evolve differently. They evolve into different patterns where some patterns may be easier for the neural network to generalise. This is clearly evident in the Laub Loomis, Bouncing ball and Spiking neurons dynamic systems as the loss for each time step was directly proportional to the evolution of variables in the system.

3. The continuous dynamic systems showed that the neural network performed better with smaller distances closer to the neural network. This is because neural networks are essentially

trained on training data. Samples with smaller distances from the samples in the training data will behave in a similar way to samples in the training data. Therefore, the neural networks will perform better with samples with smaller distances. However, samples with higher distances exhibits a different behaviour from samples within the training data. Therefore, the neural network may have a low performance.

6. Conclusions

This project has managed to achieve the aims introduced in the beginning. In this project, 6 dynamic systems used numerical simulations to produce test data and training data. The performance of a neural network was then evaluated. Additionally, knowledge about the architecture of a feed forward neural network was gained. This was done by exploring how parameters affect the performance of a neural network. Furthermore, an understanding of dynamics systems was gained. This includes the semantics and issues of modelling systems discussed in the *theoretical framework*. The knowledge of dynamics systems was strengthened as it was put into practice in the *methodology*. Concepts which were introduced in the *theoretical framework* were implemented such as numerical systems and hybrid automata. An implementation of a neural network was made. This can be found in the appendix 20.

The project proved that usage of neural networks can be used for simulating hybrid systems. The chosen 6 dynamic systems provided the required data to train the neural network. In return, the neural network can predict different state spaces given an input. This may aid in the process of co-simulation, where multiple systems are connected together to control a mechanism. A co-simulation algorithm controls the interactions between subsystems. The output from another system is used as input to a system. Instead of using a numerical solver to simulate the input, a neural network could be used in place to produce a different state space for each run of the co-simulation.

Although, the aims of the project have been achieved, they are more improvements that could be made. In addition, there are many ways this project could be expanded. The details concerning improvements are explained below:

1. One prominent issue in the project is the number of samples in the test data. Further samples should be explored to evaluate the performance of the neural network. This would provide more accurate results on the performance of the neural network. The number of samples of simulations was hindered by the number of resources on the computer and software used. The software used for hybrid automata was Stateflow. Although it was easy to simulate many samples, the issue was saving those samples into a CSV file. This was a tedious process as the output needed to be copied and pasted into excel. With more provided time, another piece of software would have been explored to provide more simulation samples. The process of simulating a dynamic system was easier than simulating a hybrid automaton. This was done using python. However, the number of resources on the computer made it hard to simulate many samples and calculate their loss without the computer behaving slowly.

2. The performance of the neural network corresponding to a dynamic system could have been better. This could be done by tuning the parameters to find the actual combination of parameters that worked well together to increase the performance. This was slightly explored in *Neural networks: Experimental settings (1)*. However, that experiment focused on how certain parameters affect the performance of neural networks and not looking at the best parameters to simulate a dynamic system.

3. This follows on from (2) in terms of performance. In *Dynamic systems: experimental (1) and (2)* we found out that the samples with lower distances generally provide a better accuracy. The reason for this was further addressed in the *discussion* section. This prompts the question of whether generalisation made by the neural network is good enough for a dynamical system. This may suggest that the neural network is slightly overfitting. There could be more techniques explored to prevent overfitting, such as early stopping, where we monitor the performance on a validation set; when the model stops improving on the test set, we stop the training. Another technique is regularization. This makes slight changes to the way the neural network algorithm works, allowing the algorithms to generalise better. There are many other techniques to apply to prevent overfitting.

Furthermore, this project could be expanded by introducing:

4. Forecasting. The project explored the neural network's performance on predicting a different state space than the samples of state spaces in the training data. However, it did not explore further time steps that go beyond the time steps in the training data. Forecasting could be

introduced whereby the neural network is provided with the same samples in the training data. However, further time steps will not be included in the training data for each sample. In addition, using the same methods we applied before: different samples of state spaces that are not in the training data are included, but this time further time steps will be included. This way we will essentially construct a neural network model which can simulate different state spaces and forecast future time steps.

5. Exploring safety states of hybrid automata is another way the project could be expanded. This is closely linked to verification of systems and is important for safety critical systems. For example, consider a basic controller hybrid automaton monitoring when the temperature of a computer is too high. The controller would not allow the computer to go beyond a certain state which is considered dangerous and ensures the computer temperature is within a safe state. A classification neural network could be introduced to classify whether a state is within a set of safe states or not. Furthermore, linking from (1) it could we could use forecasting to detect whether a state system is going to progress into a state which is not safe.

The performance of each dynamic system is provided below, measured using the technique of 5-fold cross validation detailed within the *methodology*.

Dynamic system	**Newtons Cooling law**	**Van der Pol Oscillator**	**Laub Loomis**	**Biological model**	**Bouncing ball**	**Spiking neurons**
Loss	0.0374	0.0147	0.00017	0.0232	0.2445	0.5294

The performance of the neural network gave relatively good results. However, the performance could be higher. This is particularly true for hybrid automata.

Bibliography

Alur, R. (2011). *Formal Verification of Hybrid Systems.* University of Pennsylvania.

Bielinskas, D. V. (2019, 08 04). *How Gradient Descent Works. Simple Explanation*. Retrieved from Youtube: https://www.youtube.com/watch?v=Gbz8RljxIHo

Frehse, G. (2015). *An Indroduction to Hybrid Automata, Numerical Simulation and Reachabilty Analysis.* Giéres, France.

Hindmarsh, A. C., & Petzold, L. R. (2005). *LSODA, Ordinary Differential Equation Solver for Stiff or Non-Stiff System.* NEA.

Karl Henrick Johansson, J. L. (2009). Vol.XV: Modeling of hybrid systems. In H. Unbehauen, *Control systems, robotics and automation.* Encyclopedia of Life Support Systems (EOLSS).

Keskar, N. S. (2017). *on large-batch training for deep learning : generlization gap and sharp minima.*

Lehrach, D. E. (2005). *Systems Biology in Practice: Concepts, Implementation and Application.* Wiley-VCH Verlag GmbH & Co. KGaA.

MathWorld Team. (2021, 04 30). *Van Der Pol Equation*. Retrieved from Wolframe MathWorld: https://mathworld.wolfram.com/vanderPolEquation.html

Michael T. Laub, a. W. (2017). *A Molecular Network That Produces Spontaneous Oscillations in Excitable Cells of Dictyostelium* (Vol. ix). (L. Shapiro, Ed.)

Nielsen, M. (2009). *Neural Networks and Deep Learning.*

Ripley, B. (1996). *Patterns Reconition and Neural Networks.* Cambridy Univeristy Press.

Shaikh, R. (2018). *Towards Data Science*. Retrieved from Cross Validation Explained: Evaluating estimator performance.: https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85

Theory of Hybrid Systems. (2021, 04 29). *5-dimensional switching linear system*. Retrieved from ths: rwth aachen university: https://ths.rwth-aachen.de/research/projects/hypro/5-dimensional-switching-linear-system/

Tiziano Villa, C. P. (2015). *Hybrid Automata*. Retrieved from Reykjavik University: http://www.ru.is/kennarar/luca/GSSI/SLIDES/HybridVilla.pdf

Wikimedia commons. (2020). Retrieved from Image : File:3d-function-2.svg: https://commons.wikimedia.org/wiki/File:3d-function-2.svg

Wikipedia. (2021, 04 2). *Ordinary differential equation*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Ordinary_differential_equation

Appendix

List of Figures:

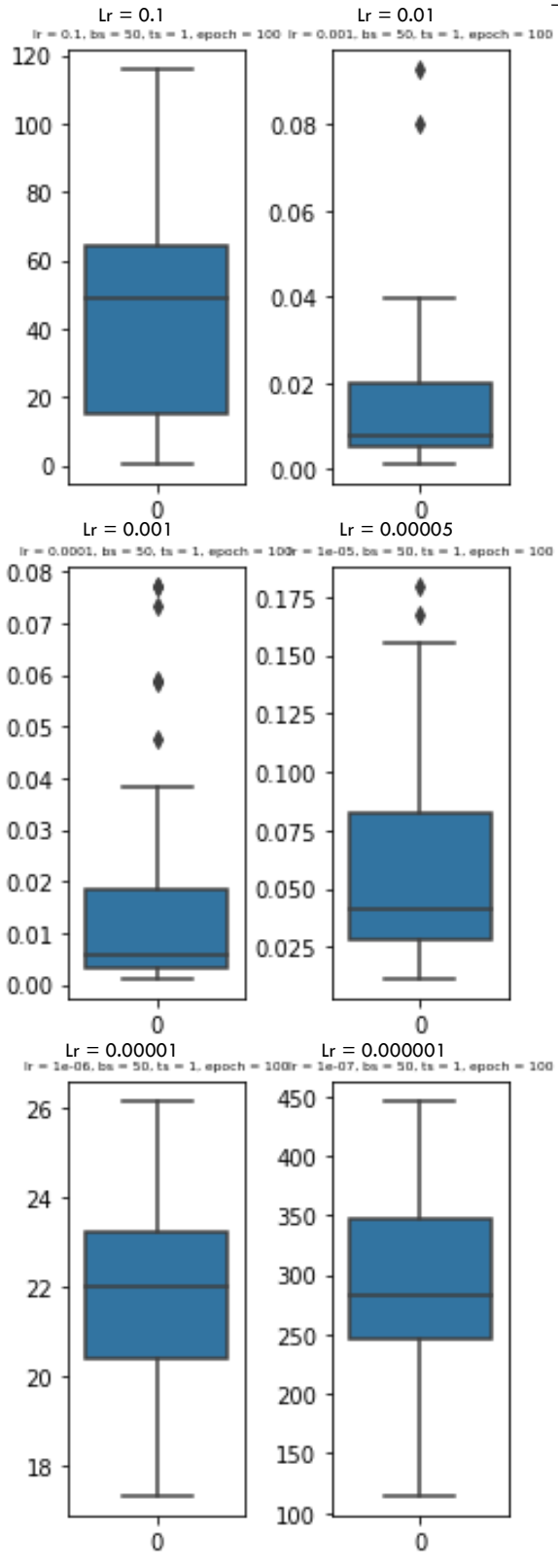
1. Implementation of cross validation

```
def cross_validation_evaluate(self, train_df, group):
    """
        cross_validation_evaluate:
            This uses the K-fold cross validation method. We we are given
            trainset dataframe. It would split the training dataframe into
            n number of groups. For each group in within n number of groups.
            It would act as a test data, while the others are used for training the data.
            We the evaluate the test data for each group. Overall the process is defined below:
                1. Take the group as a holdout or test data set
                2. Take the remaining groups as a training data set
                3. Fit a model on the training set and evaluate it on the test set
                4. Retain the evaluation score and discard the model
            The evaluation score should be roughly close together, if not we divide into more
            groups. This provides a clear score of our dataset.
        Args:
            train_df (<class 'pandas.core.frame.DataFrame'>): The dataset in pandas dataframe format
            group (int) : number of the group
        Return
            (<class 'list'>): This returns evaluations scores.

    """
    groups_df = self.cross_validation(train_df, group)

    evaluation_scores = []
    for idx, df in enumerate(groups_df):
        print("------------- Fold {} -------------" format(idx))
        a = [x for x in range(len(groups_df)) if x != idx]
        print("TEST: {} TRAIN: {}".format(idx, a))
        # trains contains all the groups of dataframes except for the
        # one we are currently iterating on
        train = [groups_df[x] for x in range(len(groups_df)) if x != idx]

        # This creates a new model, for every group
        if self.layer_list == None:
            model = prototype(self.num_inputs, self.num_targers, self.learning_rate)
        else:
            model = CustomeModel(self.num_inputs, self.num_targers, self.learning_rate, self.layer_list)

        # pandas concat dataframes together.
        train = pd.concat(train)
```

```
        # Turns inputs and targets into tensor
        train_inputs = self.df_to_tensor(train, self.inputs_cols)
        test_inputs = self.df_to_tensor(groups_df[idx], self.inputs_cols)
        train_targets = self.df_to_tensor(train, self.targets_cols)
        test_targets = self.df_to_tensor(groups_df[idx], self.targets_cols)

        # load tensor dataset
        train_dataset = TensorDataset(train_inputs, train_targets)
        # Put into into mini batch
        train_batch_loader = DataLoader(dataset=train_dataset, batch_size=self.batch_size, shuffle=True)

        # Train model
        model.train_model(train_batch_loader, self.num_epoches)

        # put test inputs into dataset.
        preds = model(test_inputs)
        loss_func = model.loss_function()
        evaluation_scores.append(loss_func(preds, test_targets))

    return evaluation_scores
```

2. Newton's cooling law: Effects of the Learning Rate on Neural Networks

a. Boxplots



This shows boxplots to illustrate the affect of different learning rates. This was done using Newtons cooling law training data.

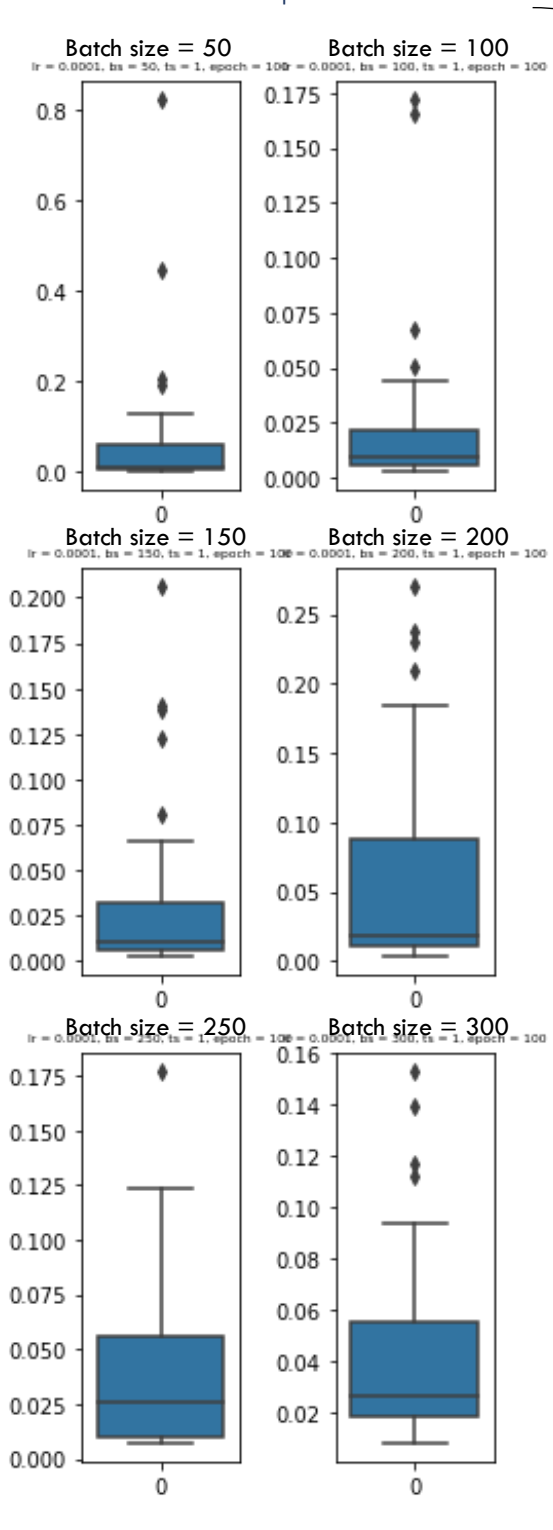
b. Statistical Averages

	lr = 0.1, bs = 50, ts = 1, epoch = 100	lr = 0.001, bs = 50, ts = 1, epoch = 100	lr = 0.0001, bs = 50, ts = 1, epoch = 100	lr = 1e-05, bs = 50, ts = 1, epoch = 100	lr = 1e-06, bs = 50, ts = 1, epoch = 100	lr = 1e-07, bs = 50, ts = 1, epoch = 100
Q1	15.178519	0.005070	0.003134	0.027918	20.418923	246.327290

Q2	48.861332	0.007779	0.005707	0.041373	22.023547	282.197952
Q3	64.062774	0.019982	0.018215	0.082201	23.225594	347.764763
Mean	45.787008	0.017896	0.019128	0.060511	21.934004	289.284983
Standard deviation	31.726222	0.021865	0.025181	0.045903	2.324425	76.848129

3. Newton's cooling law: Effects of batch size on neural networks

a. Boxplots



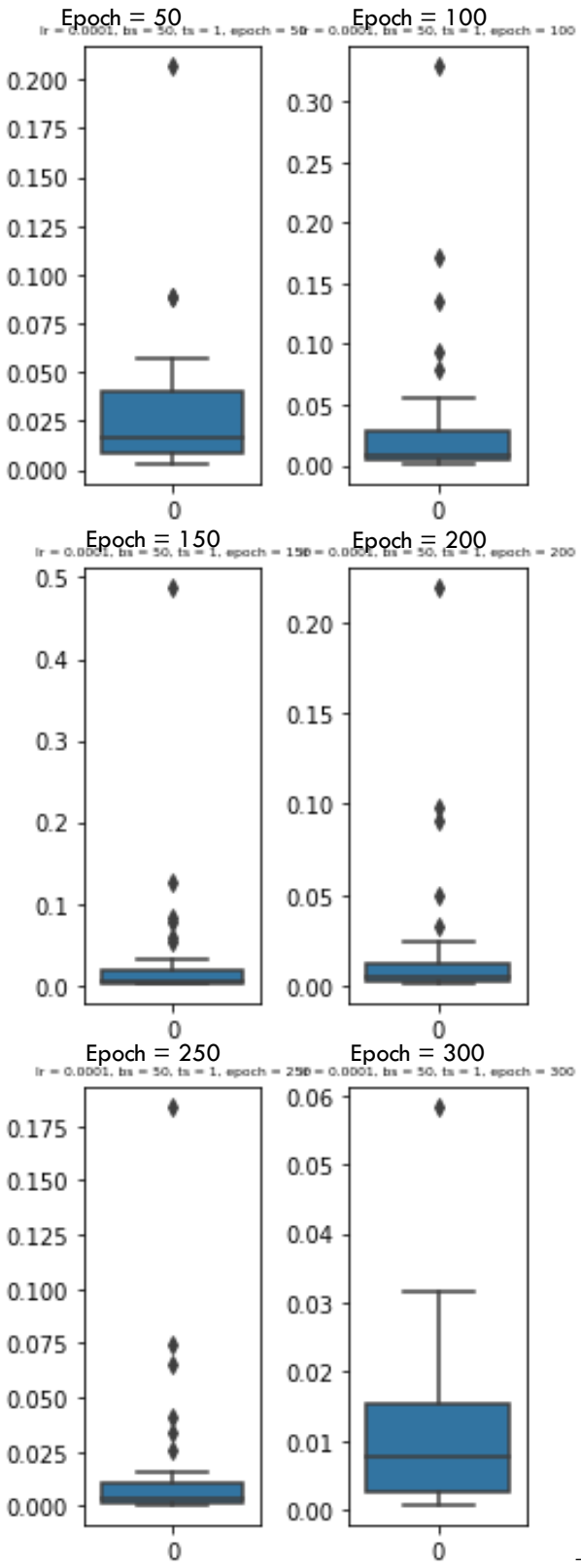
This shows boxplots to illustrate the affect of different batch sizes. This was done using Newtons cooling law training data.

b. Statistical Averages

	lr = 0.1, bs = 50, ts = 1, epoch = 100	lr = 0.001, bs = 50, ts = 1, epoch = 100	lr = 0.0001, bs = 50, ts = 1, epoch = 100	lr = 1e-05, bs = 50, ts = 1, epoch = 100	lr = 1e-06, bs = 50, ts = 1, epoch = 100	lr = 1e-07, bs = 50, ts = 1, epoch = 100
Q1	15.178519	0.005070	0.003134	0.027918	20.418923	246.327290
Q2	48.861332	0.007779	0.005707	0.041373	22.023547	282.197952
Q3	64.062774	0.019982	0.018215	0.082201	23.225594	347.764763
Mean	45.787008	0.017896	0.019128	0.060511	21.934004	289.284983
Standard deviation	31.726222	0.021865	0.025181	0.045903	2.324425	76.848129

4. Newton's cooling law: Effects of the number of epochs on the neural network

a. Boxplots



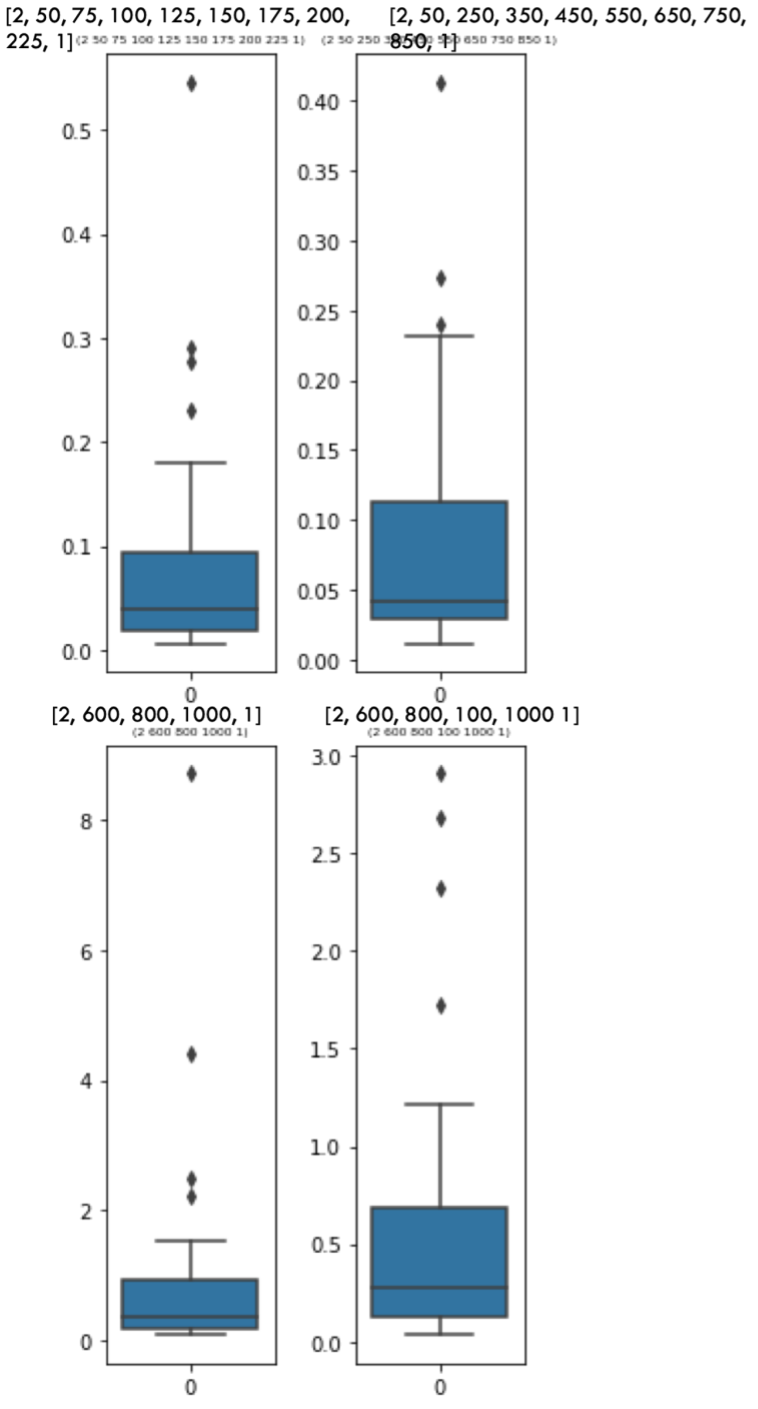
This shows boxplots to illustrate the affect of different number of epochs. This was done using Newtons cooling law training data.

b. Statistical Averages

	lr = 0.0001, bs = 50, ts = 1, epoch = 50	lr = 0.0001, bs = 50, ts = 1, epoch = 100	lr = 0.0001, bs = 50, ts = 1, epoch = 150	lr = 0.0001, bs = 50, ts = 1, epoch = 200	lr = 0.0001, bs = 50, ts = 1, epoch = 250	lr = 0.0001, bs = 50, ts = 1, epoch = 300
Q1	0.008240	0.003887	0.003124	0.002319	0.001713	0.002648
Q2	0.016005	0.008418	0.005383	0.005255	0.003475	0.007789
Q3	0.039807	0.027358	0.019243	0.012400	0.010932	0.015340
Mean	0.030704	0.037284	0.035079	0.020982	0.017224	0.011764
Standard deviation	0.040559	0.069100	0.090750	0.044686	0.036567	0.012630

5. Newton's cooling law: Effects of Layers on the Neural Network

a. Boxplots



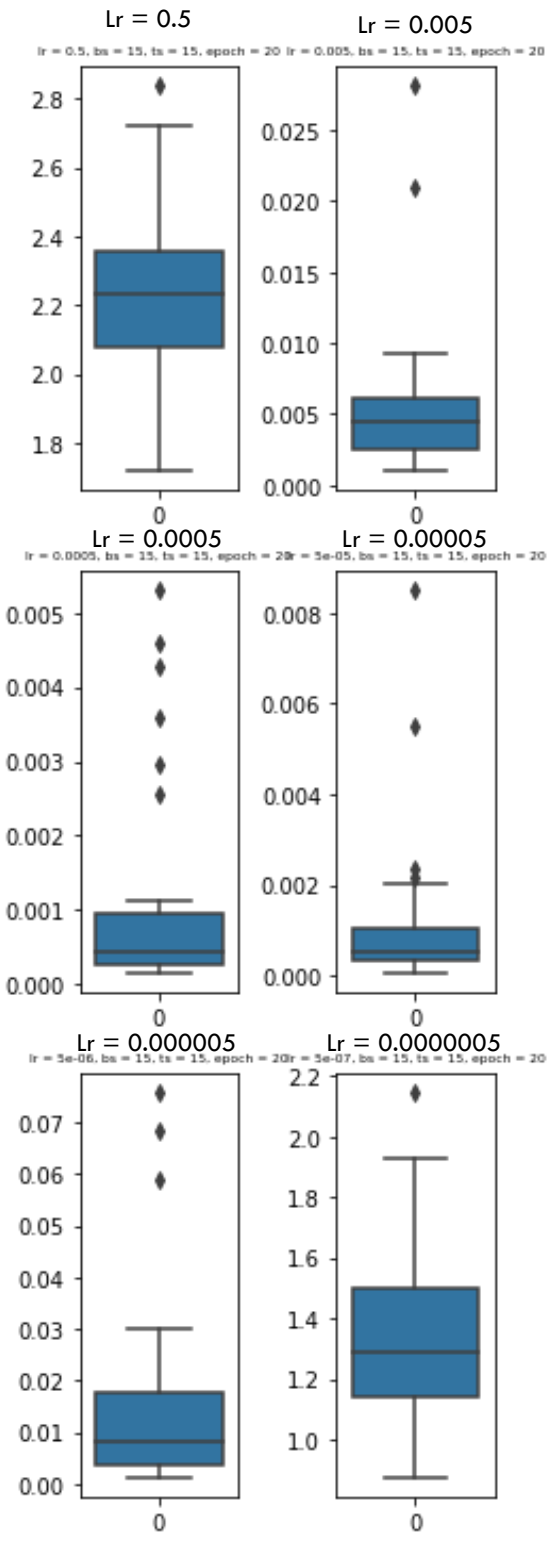
This shows boxplots to illustrate the affect on different numbers of neurons and layers on a neural network. This was explored using Newtons cooling law training data

b. Statistical Averages

	(2 50 75 100 125 150 175 200 225 1)	(2 50 250 350 450 550 650 750 850 1)	(2 600 800 1000 1)	(2 600 800 100 1000 1)
Q1	0.017968	0.029677	0.181464	0.128587
Q2	0.038959	0.041832	0.367685	0.273345
Q3	0.094561	0.113056	0.935951	0.681668
Mean	0.086426	0.086524	0.977586	0.617171
Standard deviation	0.117920	0.095246	1.729999	0.792256

6. Van der Pol: Effects of the Learning rate on the Neural Network

a. Boxplots



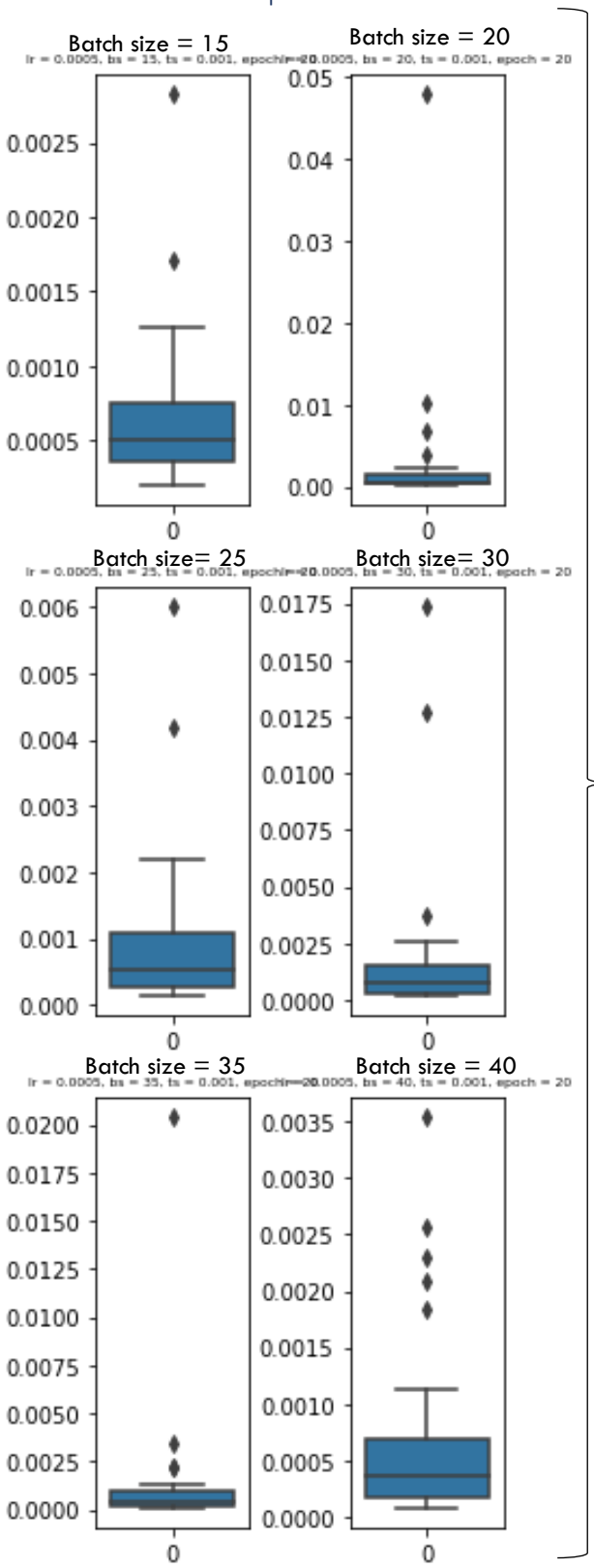
This shows boxplots. Each boxplot correspond to a different learning rates. This is to show the affect of different learning rate. This is done using training data of van der pol

b. Statistical Averages

	lr = 0.5, bs = 15, ts = 15, epoch = 20	lr = 0.005, bs = 15, ts = 15, epoch = 20	lr = 0.0005, bs = 15, ts = 15, epoch = 20	lr = 5e-05, bs = 15, ts = 15, epoch = 20	lr = 5e-06, bs = 15, ts = 15, epoch = 20	lr = 5e-07, bs = 15, ts = 15, epoch = 20
Q1	2.078812	0.002492	0.000251	0.000352	0.003591	1.141486
Q2	2.233722	0.004442	0.000436	0.000531	0.008248	1.289878
Q3	2.359528	0.006164	0.000956	0.001047	0.017755	1.498740
Mean	2.240288	0.005737	0.001117	0.001162	0.015224	1.355591
Standard deviation	0.236520	0.005684	0.001493	0.001749	0.019571	0.282192

7. Van der Pol: Effects of batch size on neural networks

a. Boxplots



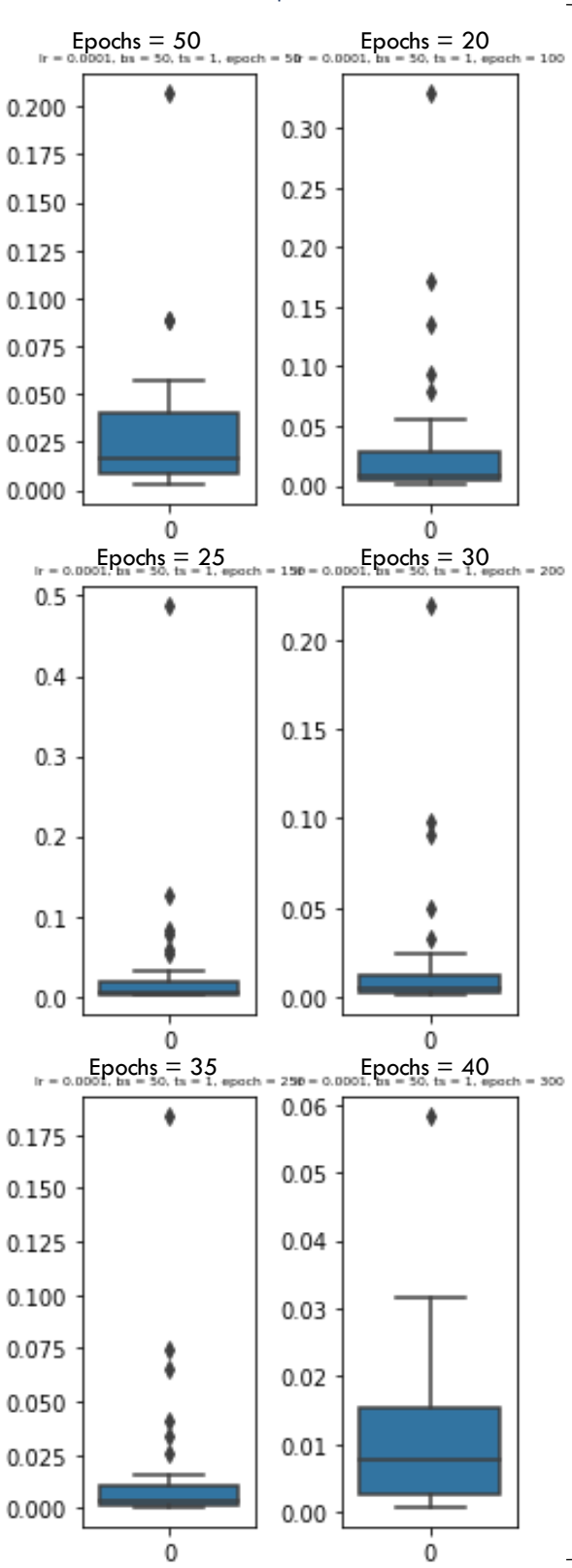
This shows boxplots. Each boxplot correspond to a different bacth sizes. This is to show the affect of different batch sizes. This is done using training data of van der pol

b. Statistical Averages

	lr = 0.0005, bs = 15, ts = 0.001, epoch = 20	lr = 0.0005, bs = 20, ts = 0.001, epoch = 20	lr = 0.0005, bs = 25, ts = 0.001, epoch = 20	lr = 0.0005, bs = 30, ts = 0.001, epoch = 20	lr = 0.0005, bs = 35, ts = 0.001, epoch = 20	lr = 0.0005, bs = 40, ts = 0.001, epoch = 20
Q1	0.000359	0.000305	0.000259	0.000325	0.000233	0.000181
Q2	0.000508	0.000560	0.000525	0.000757	0.000421	0.000362
Q3	0.000750	0.001388	0.001078	0.001568	0.000920	0.000680
Mean	0.000670	0.002892	0.000971	0.001890	0.001346	0.000716
Standard deviation	0.000534	0.008764	0.001278	0.003724	0.003675	0.000866

8. Van der Pol: Effects of the number of epochs on the neural network

a. Boxplots



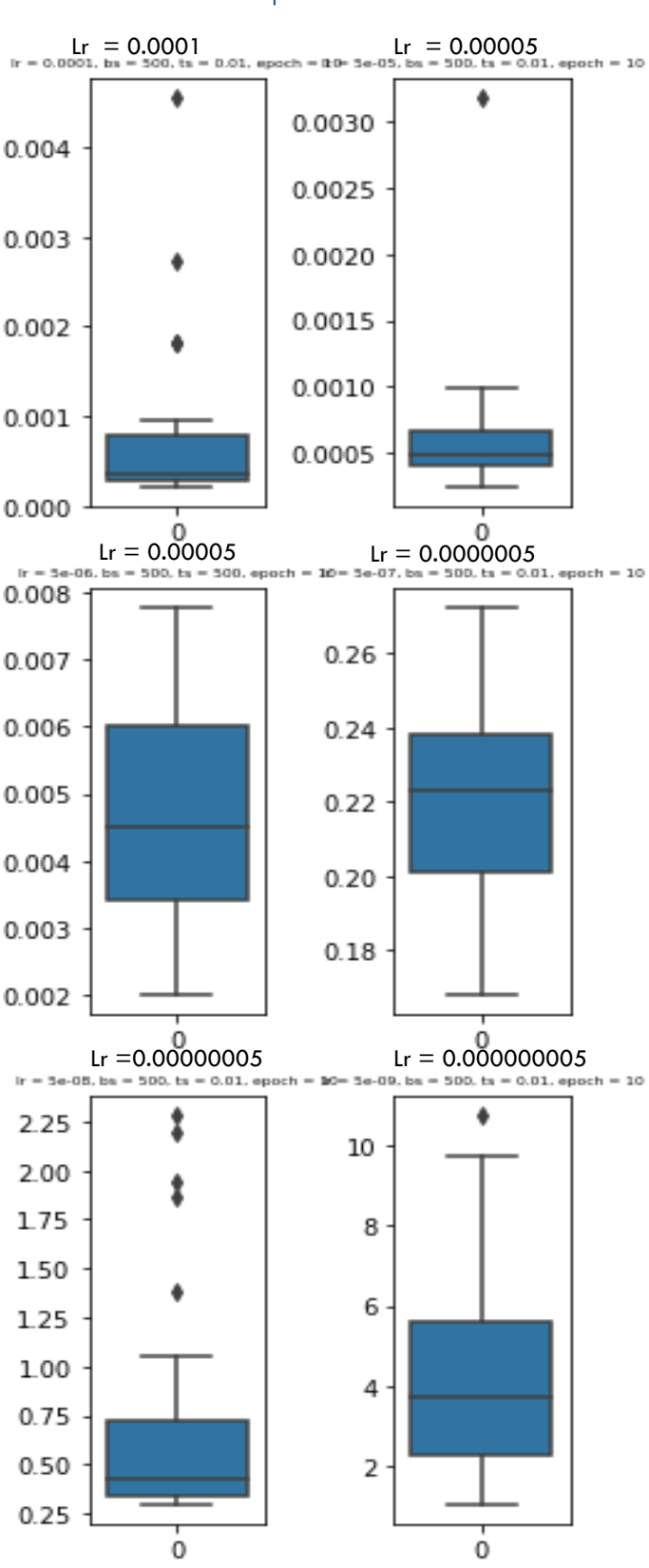
This shows boxplots. Each boxplot correspond to a different number of epochs. This is to show the affect of different number of epochs. This is done using training data of van der pol

b. Statistical Averages

	lr = 0.0001, bs = 50, ts = 1, epoch = 50	lr = 0.0001, bs = 50, ts = 1, epoch = 100	lr = 0.0001, bs = 50, ts = 1, epoch = 150	lr = 0.0001, bs = 50, ts = 1, epoch = 200	lr = 0.0001, bs = 50, ts = 1, epoch = 250	lr = 0.0001, bs = 50, ts = 1, epoch = 300
Q1	0.008240	0.003887	0.003124	0.002319	0.001713	0.002648
Q2	0.016005	0.008418	0.005383	0.005255	0.003475	0.007789
Q3	0.039807	0.027358	0.019243	0.012400	0.010932	0.015340
Mean	0.030704	0.037284	0.035079	0.020982	0.017224	0.011764
Standard deviation	0.040559	0.069100	0.090750	0.044686	0.036567	0.012630

9. Laub Loomis: Effects of the learning rate on the neural network

a. Boxplots



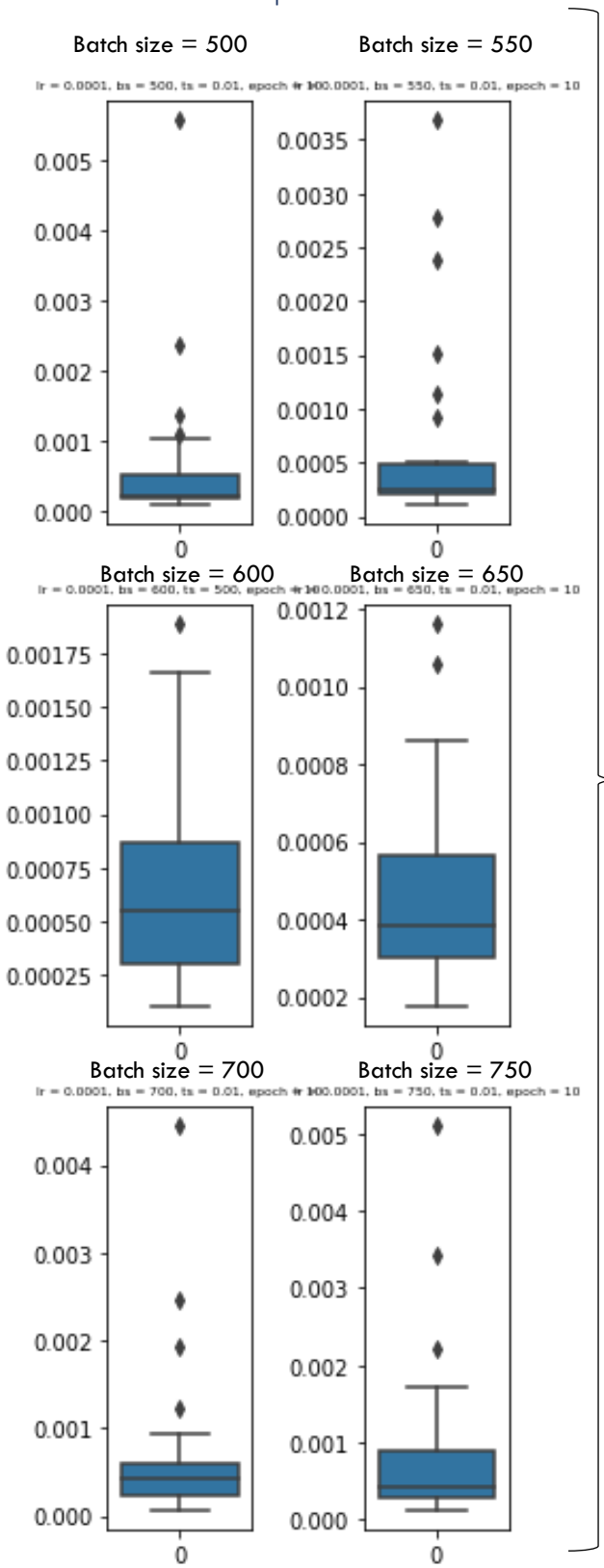
This shows boxplots. Each boxplot correspond to a different learning. This is to show the affect of different learning rate. This is done using training data of Laub Loomis

b. Statistical Averages

	lr = 0.0001, bs = 500, ts = 0.01, epoch = 10	**lr = 5e-05, bs = 500, ts = 0.01, epoch = 10**	**lr = 5e-06, bs = 500, ts = 500, epoch = 10**	**lr = 5e-07, bs = 500, ts = 0.01, epoch = 10**	**lr = 5e-08, bs = 500, ts = 0.01, epoch = 10**	**lr = 5e-09, bs = 500, ts = 0.01, epoch = 10**
Q1	0.000281	0.000397	0.003419	0.200893	0.344023	2.261743
Q2	0.000348	0.000485	0.004520	0.222949	0.422810	3.744069
Q3	0.000789	0.000654	0.006003	0.238083	0.726582	5.589386
Mean	0.000739	0.000618	0.004613	0.219355	0.703895	4.189723
Standard deviation	0.000920	0.000515	0.001752	0.028342	0.598935	2.342992

10. Laub Loomis: Effects of batch size on neural networks

a. Boxplots



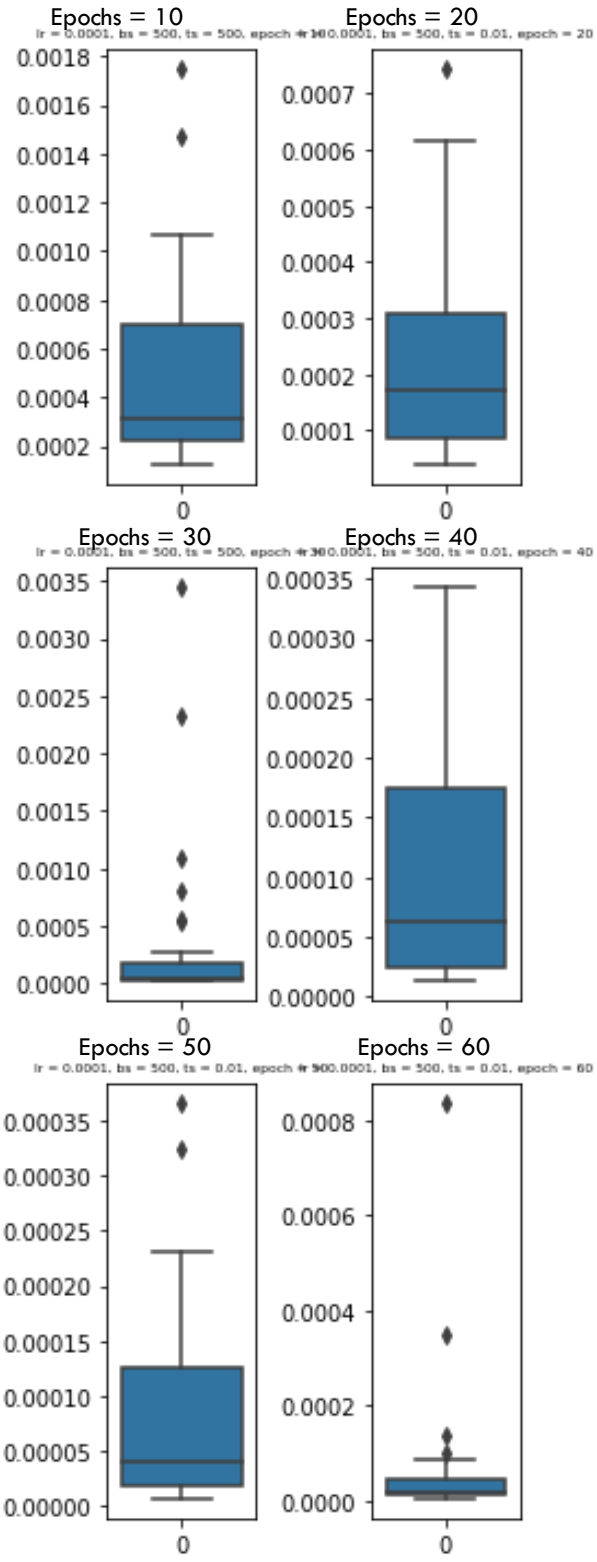
This shows boxplots. Each boxplot correspond to a batch sizes. This is to show the affect of different batch sizes. This is done using training data of Laub Loomis

b. Statistical Averages

	lr = 0.0001, bs = 500, ts = 0.01, epoch = 10	**lr = 0.0001, bs = 550, ts = 0.01, epoch = 10**	**lr = 0.0001, bs = 600, ts = 500, epoch = 10**	**lr = 0.0001, bs = 650, ts = 0.01, epoch = 10**	**lr = 0.0001, bs = 700, ts = 0.01, epoch = 10**	**lr = 0.0001, bs = 750, ts = 0.01, epoch = 10**
Q1	0.000181	0.000210	0.000306	0.000300	0.000238	0.000290
Q2	0.000221	0.000259	0.000551	0.000384	0.000431	0.000438
Q3	0.000516	0.000483	0.000870	0.000568	0.000594	0.000904
Mean	0.000604	0.000623	0.000681	0.000454	0.000698	0.000841
Standard deviation	0.001051	0.000862	0.000479	0.000243	0.000875	0.001070

11. Laub Loomis: Effects of the Number of epochs on the Neural Network

a. Boxplot



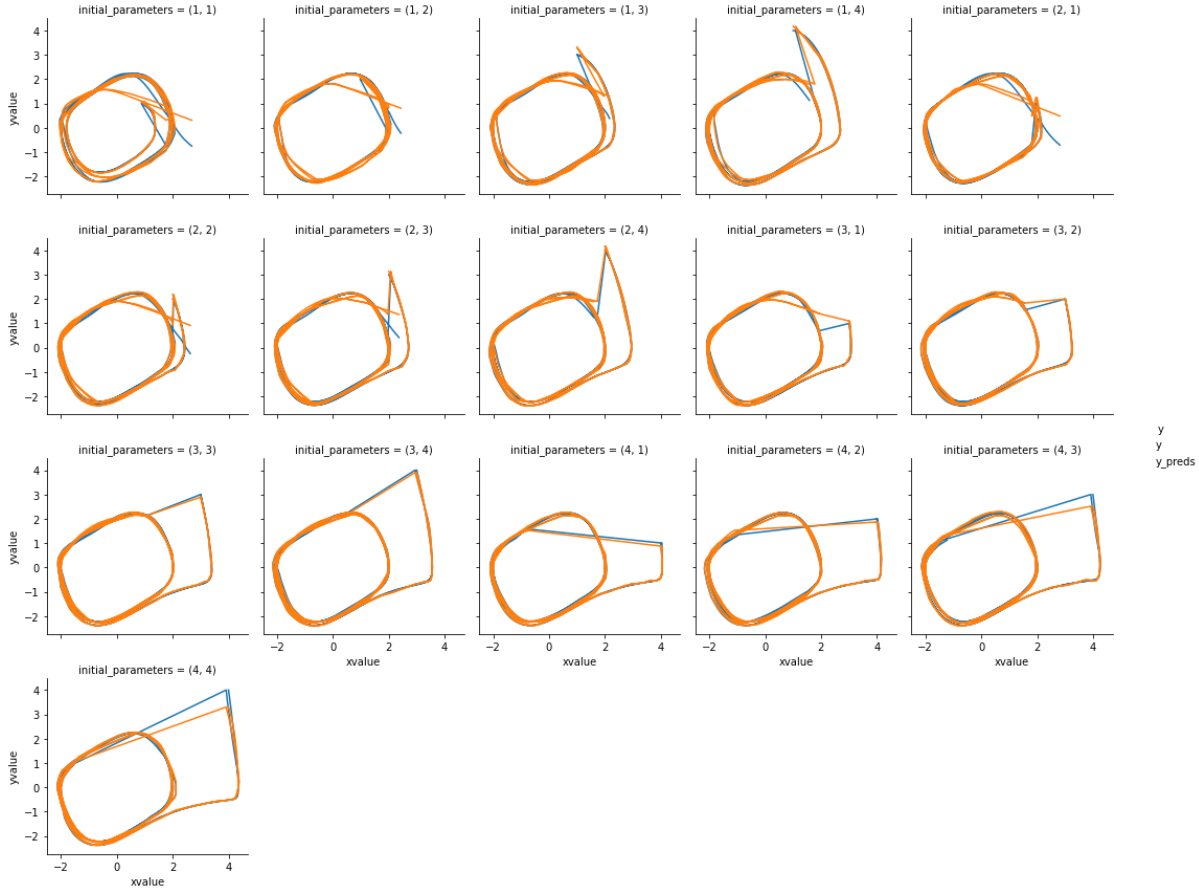
This shows boxplots. Each boxplot correspond to a different number of epochs. This is to show the affect of different number of epochs. This is done using training data of Laub Loomis.

b. Statistical Averages

	lr = 0.0001, bs = 500, ts = 500, epoch = 10	lr = 0.0001, bs = 500, ts = 0.01, epoch = 20	lr = 0.0001, bs = 500, ts = 500, epoch = 30	lr = 0.0001, bs = 500, ts = 0.01, epoch = 40	lr = 0.0001, bs = 500, ts = 0.01, epoch = 50	lr = 0.0001, bs = 500, ts = 0.01, epoch = 60
Q1	0.000223	0.000085	0.000034	0.000025	0.000019	0.000012
Q2	0.000314	0.000172	0.000053	0.000062	0.000040	0.000020
Q3	0.000703	0.000308	0.000183	0.000175	0.000125	0.000044
Mean	0.000492	0.000230	0.000345	0.000103	0.000082	0.000068
Standard deviation	0.000408	0.000187	0.000750	0.000092	0.000095	0.000159

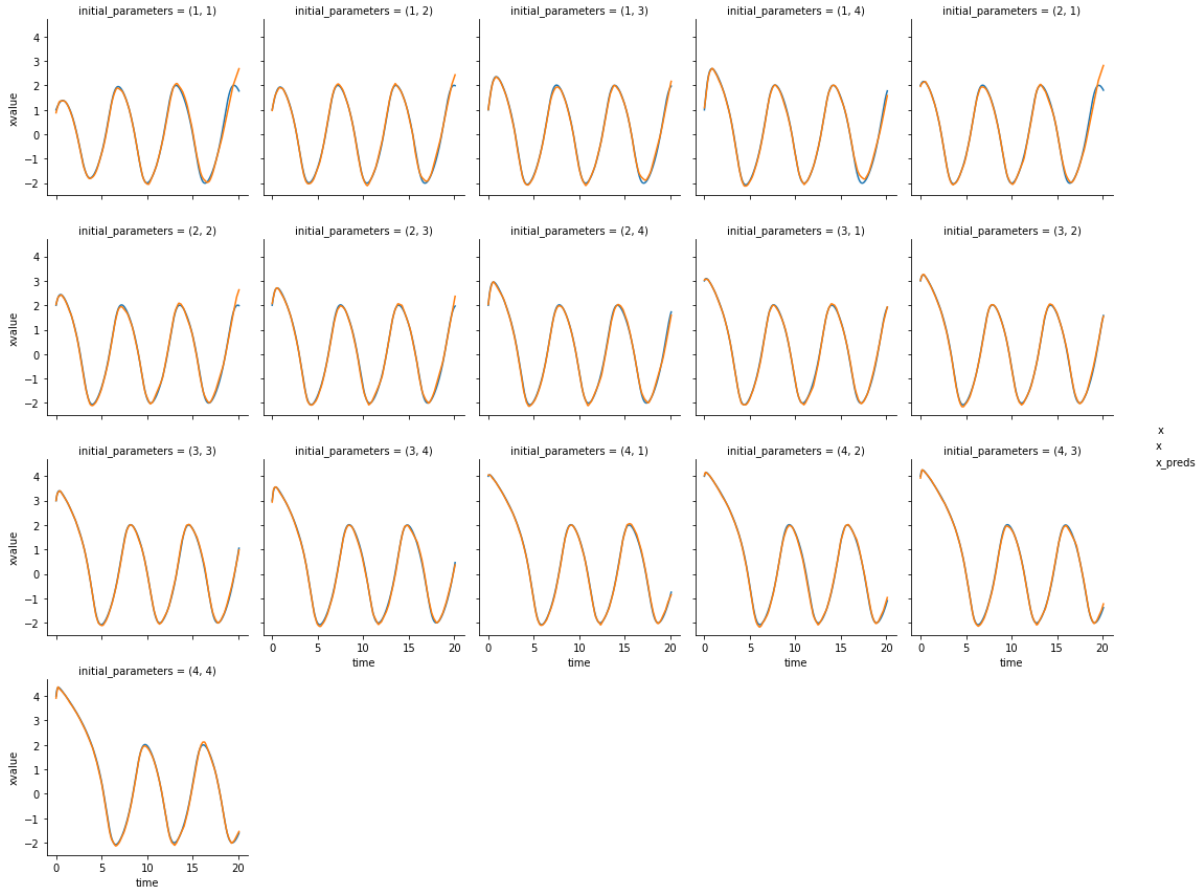
12. Van der Pol: Comparison of predictions and training data

a. Variable x plotted against y



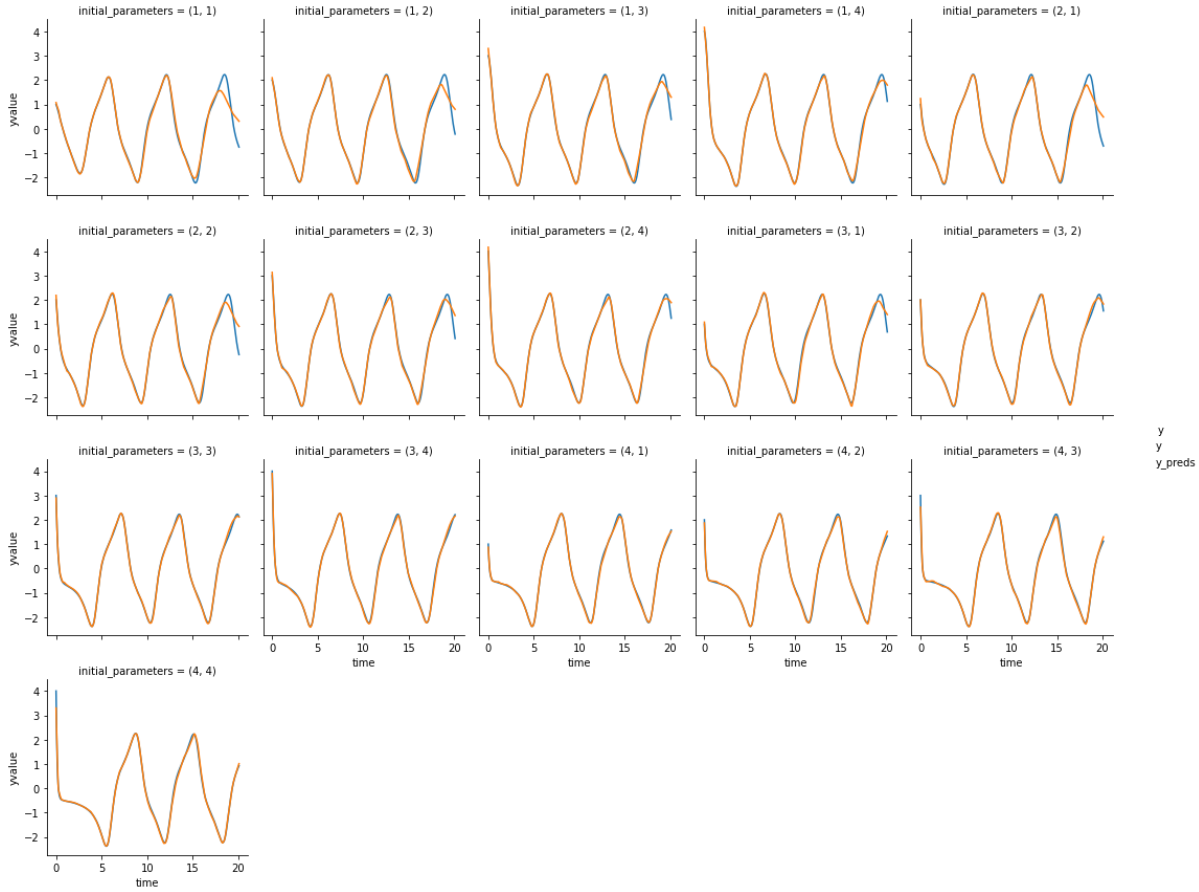
This figure above illustrates individual graphs. These individual graphs correspond to a state space. Within each graph it compares the van der pol oscillator function simulated using prediction of the neural network (denoted in orange) compared to the actual numerical simulation (**training data** - denoted in blue). The graphs are plotted against the x and y variables in van der pol.

b. Variable time plotted against x



This is similar to appendix 12.a expect the graphs to be plotted against time and x variable of the van der pol function.

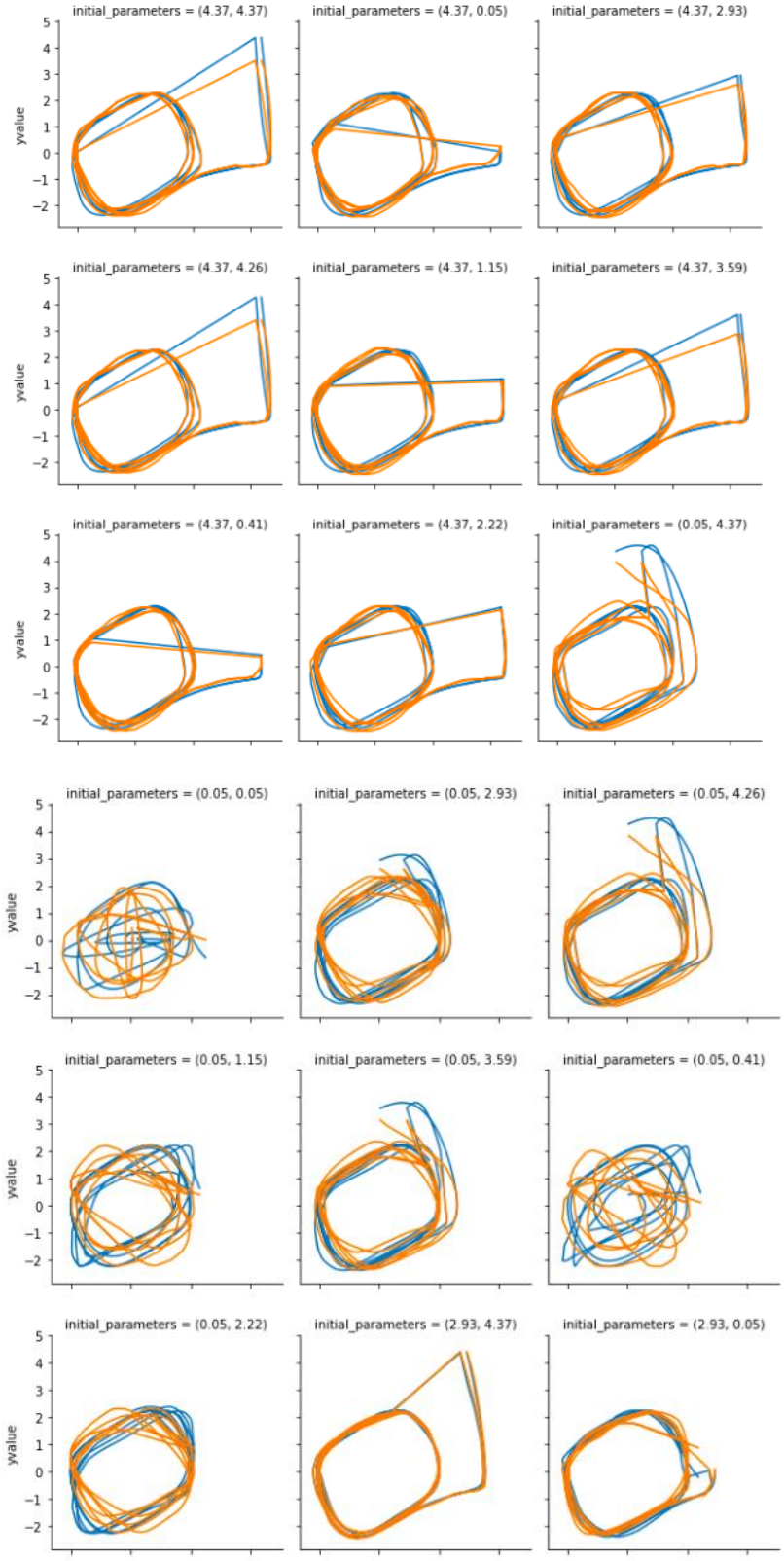
c. Variable time plotted against y

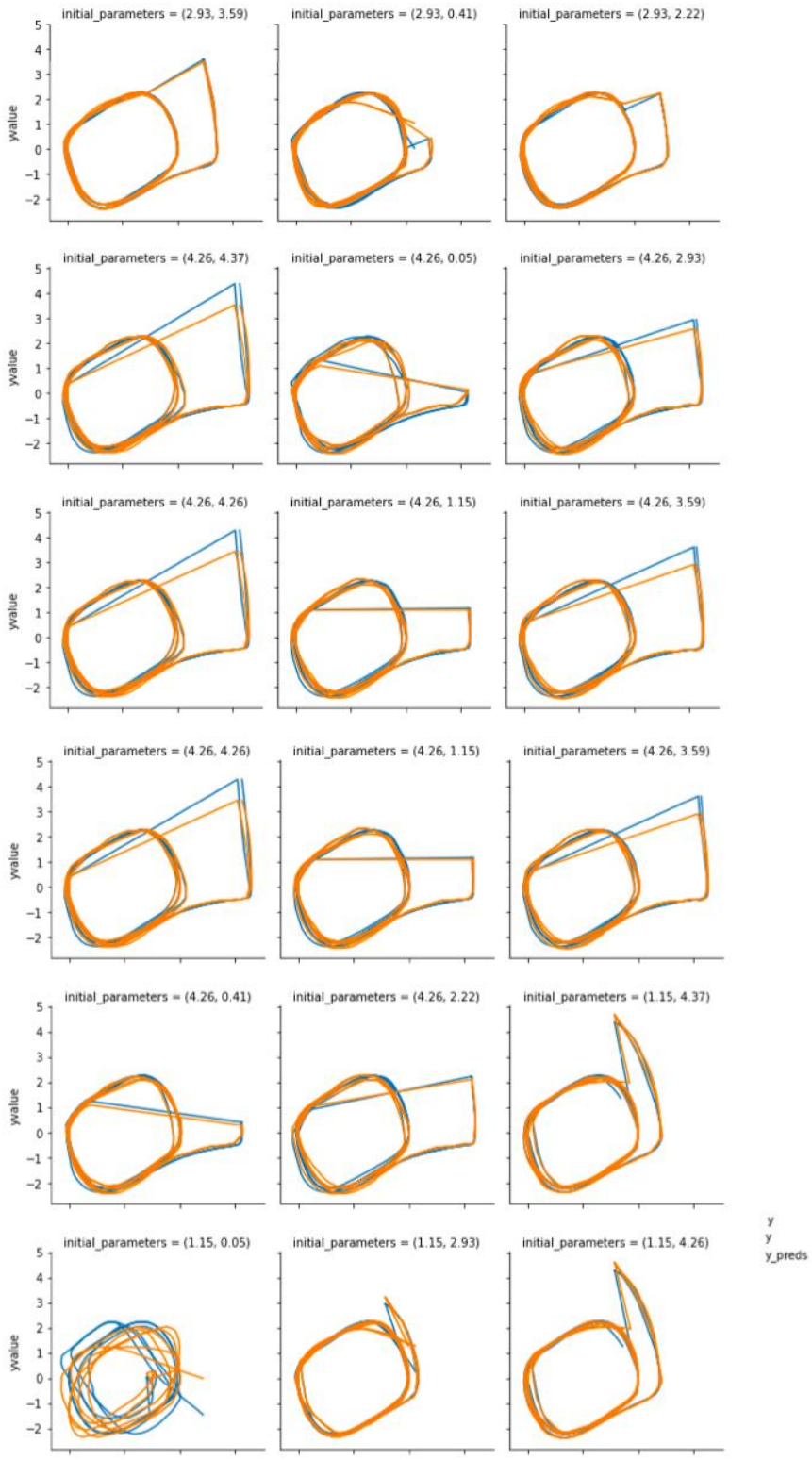


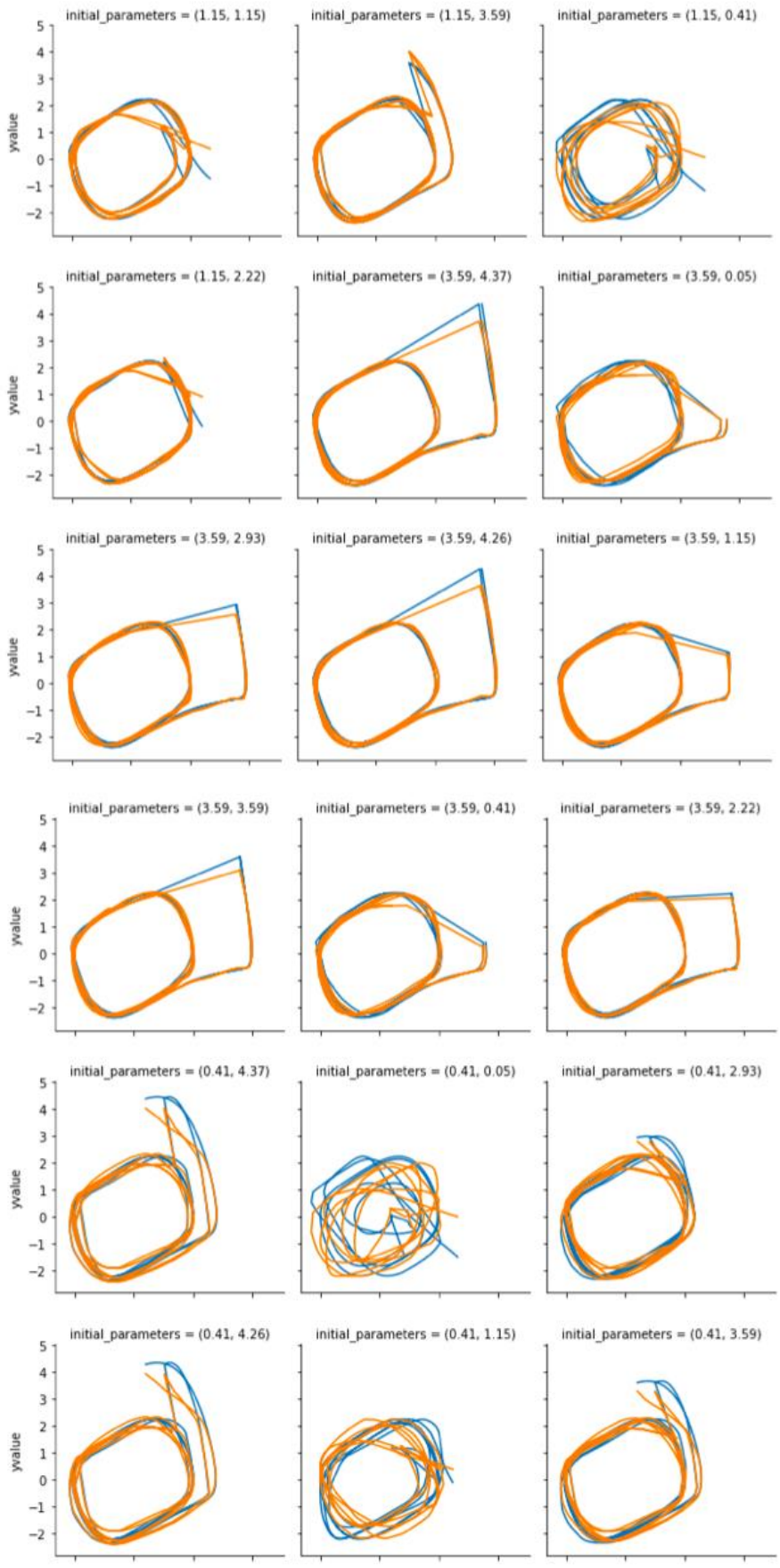
This is similar to appendix 12.a expect the graphs are plotted against time and y variable of the van der pol function.

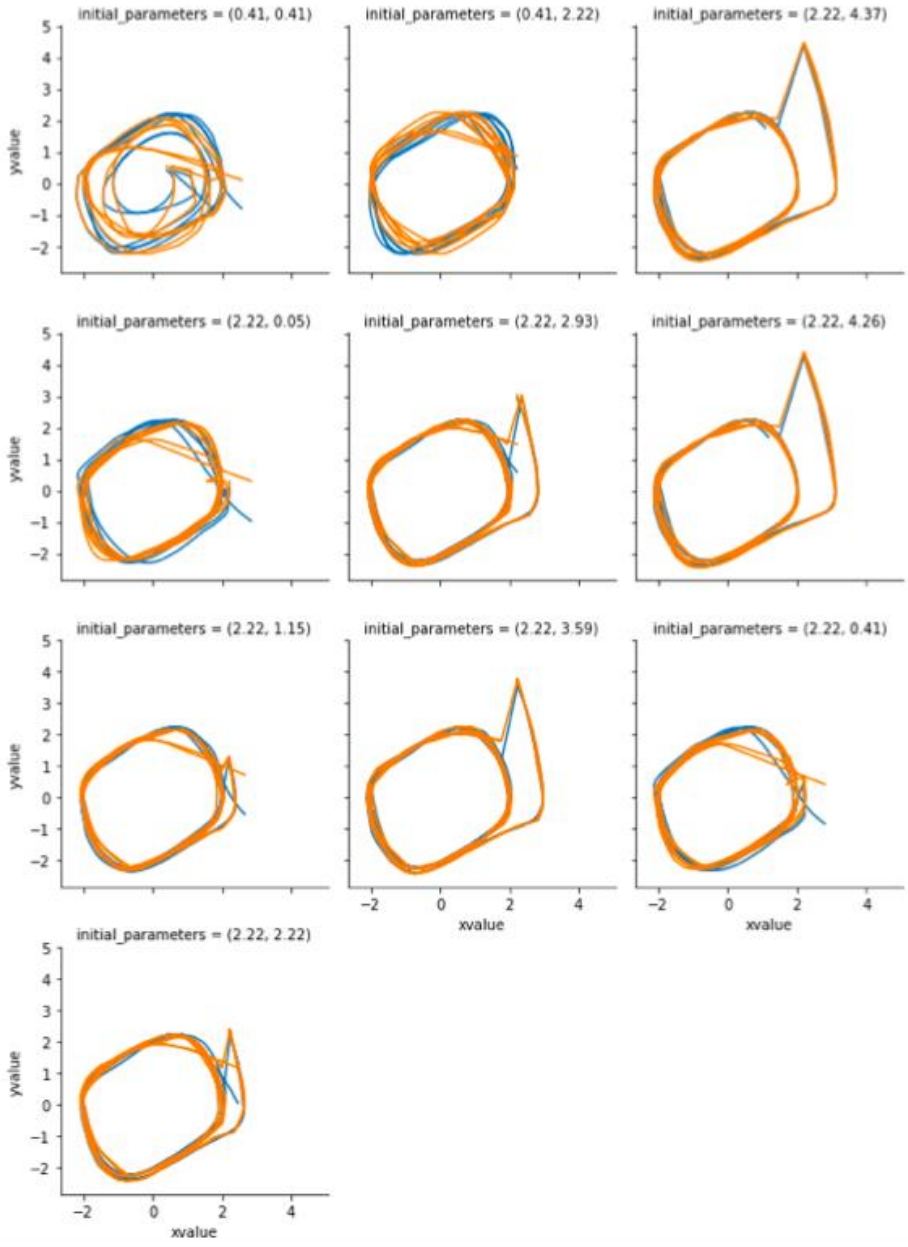
13. Van der Pol: Comparison of predictions and test data

a. Variables x plotted against y




initial_parameters = (2.93, 3.59)
initial_parameters = (2.93, 0.41)
initial_parameters = (2.93, 2.22)
initial_parameters = (4.26, 4.37)
initial_parameters = (4.26, 0.05)
initial_parameters = (4.26, 2.93)
initial_parameters = (4.26, 4.26)
initial_parameters = (4.26, 1.15)
initial_parameters = (4.26, 3.59)
initial_parameters = (4.26, 4.26)
initial_parameters = (4.26, 1.15)
initial_parameters = (4.26, 3.59)
initial_parameters = (4.26, 0.41)
initial_parameters = (4.26, 2.22)
initial_parameters = (1.15, 4.37)
initial_parameters = (1.15, 0.05)
initial_parameters = (1.15, 2.93)
initial_parameters = (1.15, 4.26)
yvalue
y
y
y_preds

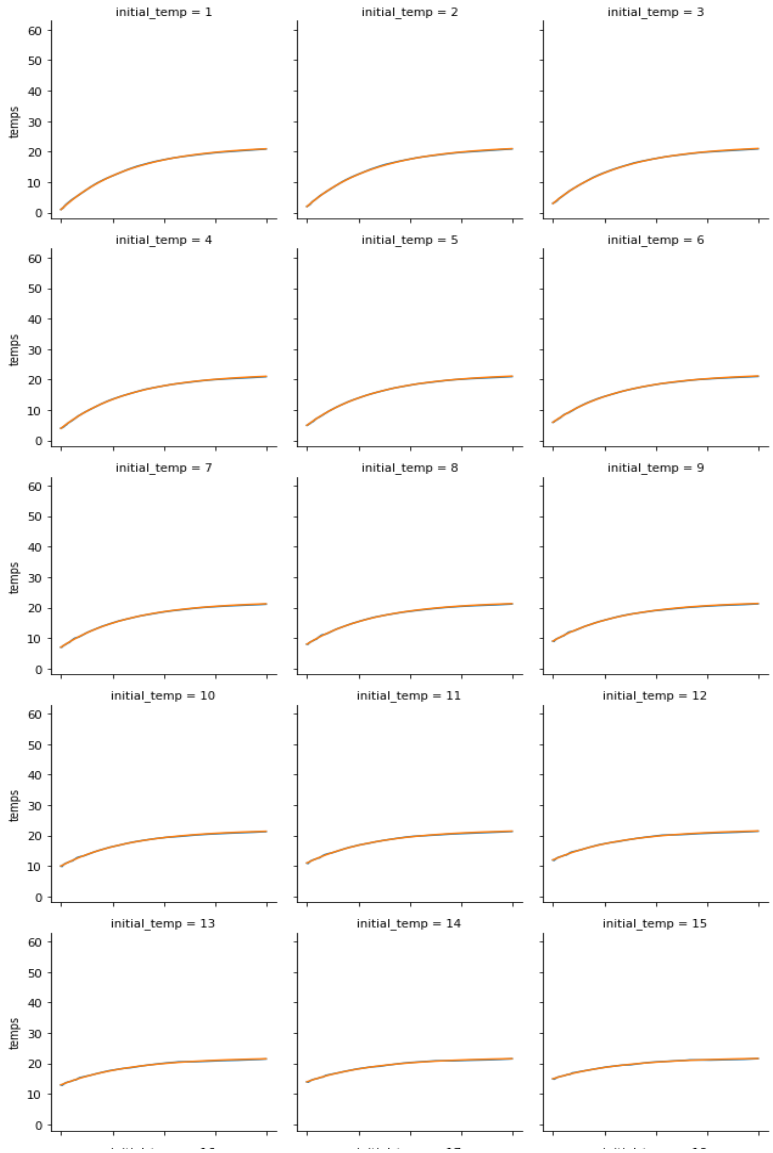

initial_parameters = (1.15, 1.15)
initial_parameters = (1.15, 3.59)
initial_parameters = (1.15, 0.41)
initial_parameters = (1.15, 2.22)
initial_parameters = (3.59, 4.37)
initial_parameters = (3.59, 0.05)
initial_parameters = (3.59, 2.93)
initial_parameters = (3.59, 4.26)
initial_parameters = (3.59, 1.15)
initial_parameters = (3.59, 3.59)
initial_parameters = (3.59, 0.41)
initial_parameters = (3.59, 2.22)
initial_parameters = (0.41, 4.37)
initial_parameters = (0.41, 0.05)
initial_parameters = (0.41, 2.93)
initial_parameters = (0.41, 4.26)
initial_parameters = (0.41, 1.15)
initial_parameters = (0.41, 3.59)
yvalue



This figure above illustrates individual graphs. These individual graphs correspond to samples in the test data of the van der pol model. This graph shows the predictions of simulations made by the neural network.

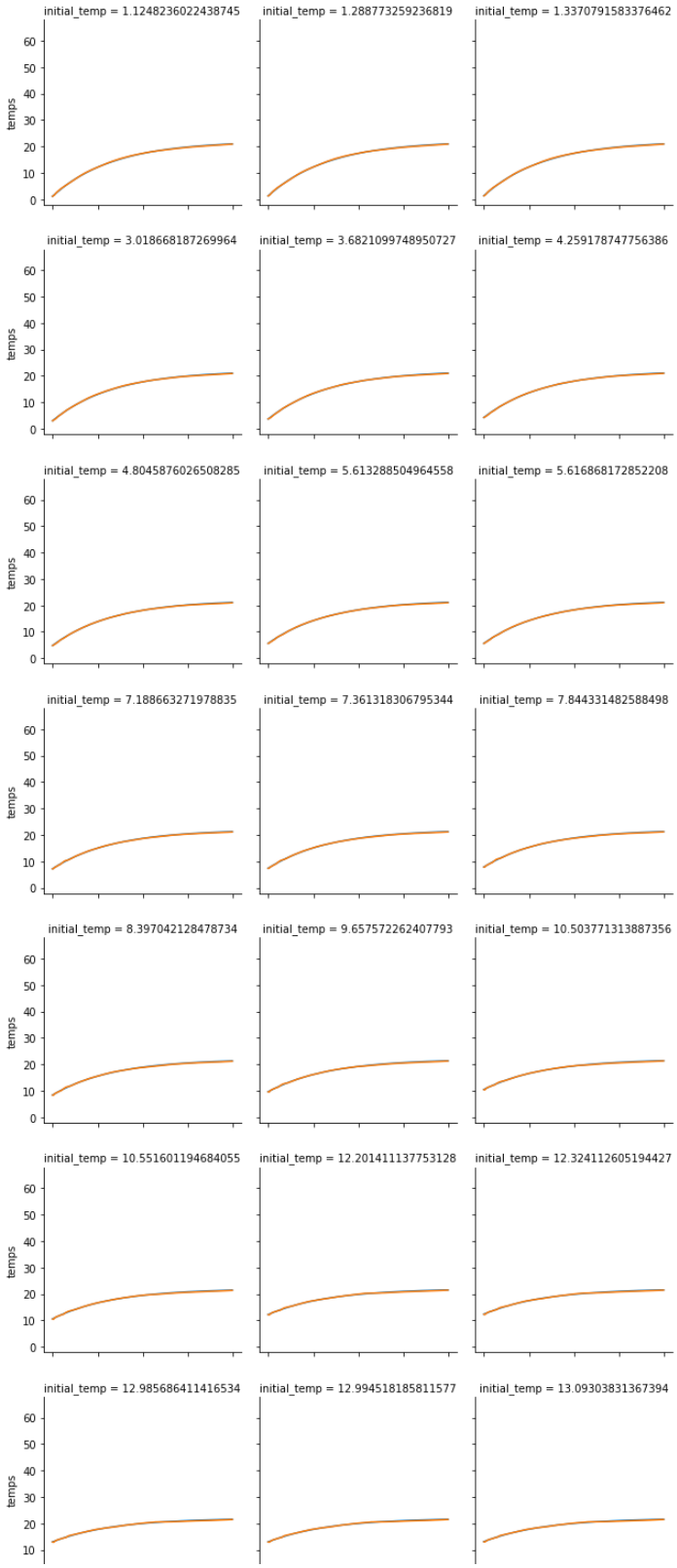
14. Newton's cooling Law: Predictions compared to numerical simulations

a. Training data



This graph looks at the first 15 samples in the training data for newton cooling law. Each individual graph plots one sample. Within each graph it is comparing the time (x axis) against the temperature (y axis)

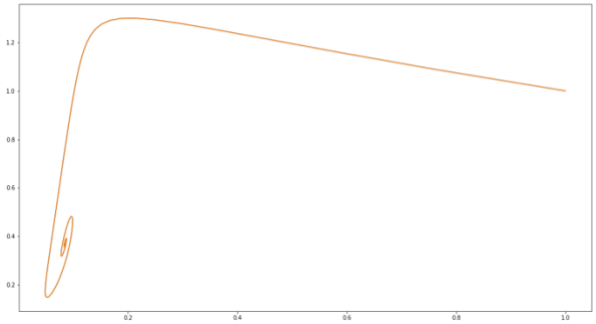
b. Test data

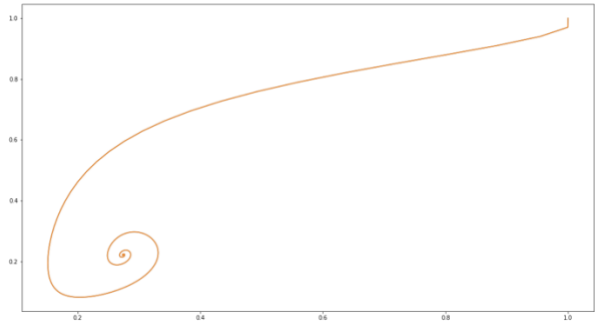


This graph looks at the first 21 samples in the test data for newton cooling law. Each individual graph plots one sample. Within each graph it is comparing the time (x axis) against the temperature (y axis)

15. Laub Loomis: Predictions compared to numerical simulations.

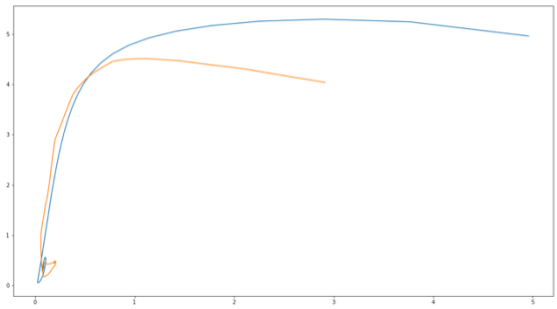
a. Training data

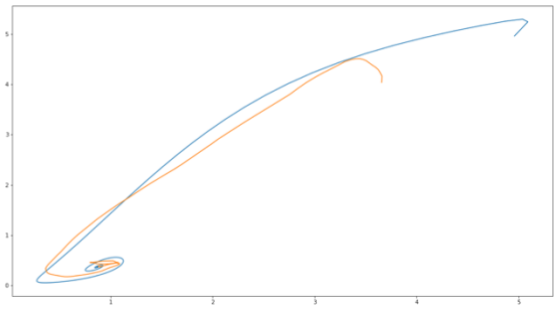




This figure shows the numerical simulations (training data) compared the predictions of the neural network. This orange denotes the predictions, and the blue denotes the numerical simulation. However, the lines may not be distinguishable, as the neural network produces accurate results. The first figure is plotted with the q and y of the variables (x axis = q, y axis =y). The second figure is plotted with x and y variables (x axis = x, y axis =y). The variables belong to one sample within the training data of the Laub Loomis where the initial state variables ($x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 1, x_6 = 1, x_7 = 1$)

b. Test data

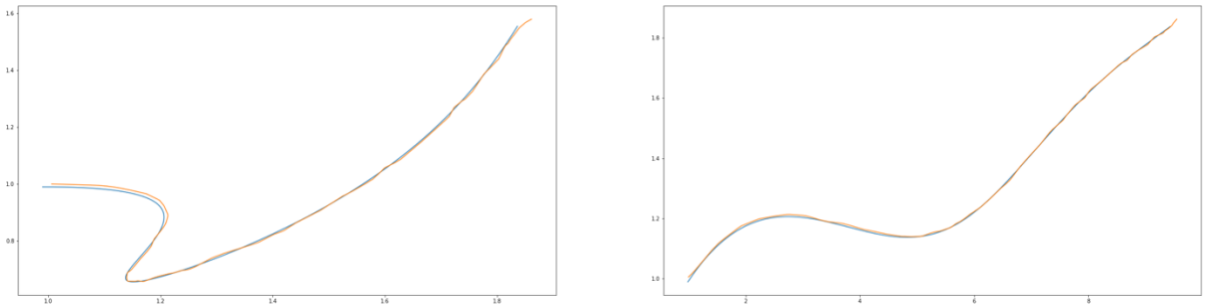




This figure is similar to the figure above (Appendix 15.a). However, we are comparing the testing data with the predictions of the neural network. The variables belong to one sample within the training data of the Laub Loomis where the initial state variables ($x_1 = 2.3, x_2 = 2.3, x_3 = 2.3, x_4 = 2.3, x_5 = 2.3, x_6 = 2.3, x_7 = 2.3$)

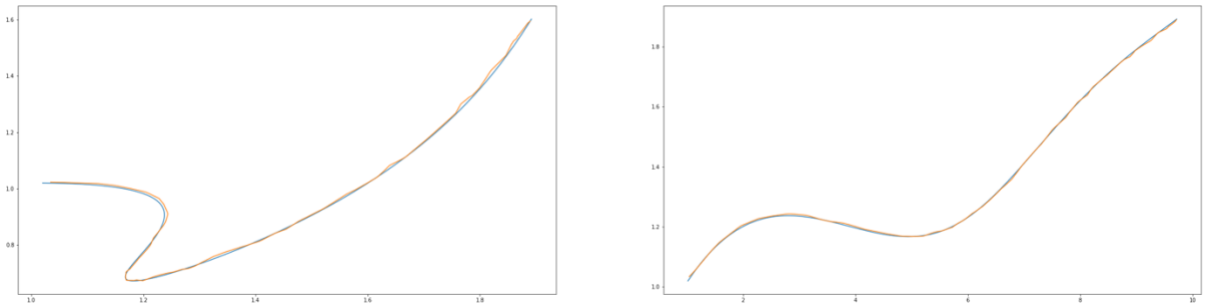
16. Biological Model: Predictions compared to numerical simulations

a. Training data



This is comparing the numerical simulations (training data) and predictions from the neural network. The first graph is plotted with x_1 and x_2, where x axis's is show variables of x_1 and y axis's shows x_2. The second graph is plotted against x_5 and x_1 where x axis is x_5 and y axis is x_1. These plots come from a sample within the training data where the initial state variables $(x_1 = 0.99, x_2 = 0.99, x_3 = 0.99, x_4 = 0.99, x_5 = 0.99,\ x\_6 = 0.99, x_7 = 0.99, x_8 = 0.99, x_9 = 1.01)$

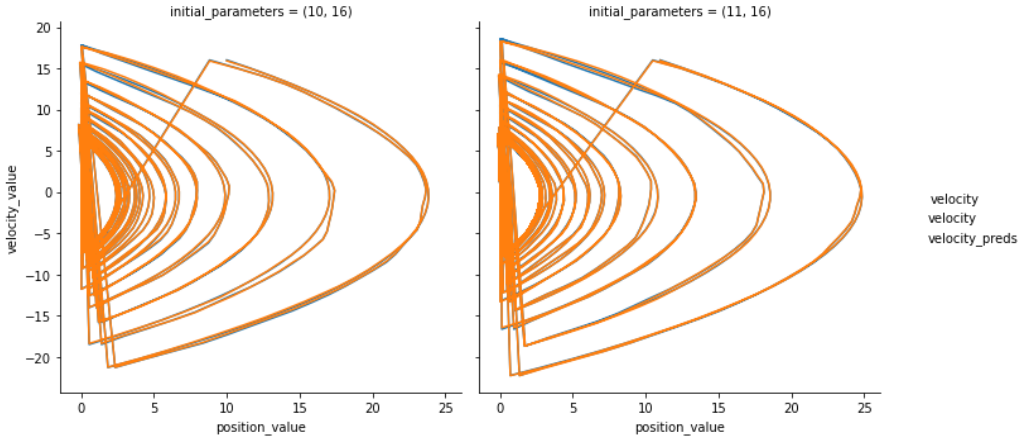
b. Test data



This is similar to the figure above (appendix 16.a). However, were comparing the testing data and predictions of the neural network. These plots are from the sample with initial state variables as $(x_1 = 1.02, x_2 = 1.02,, x_3 = 1.02,, x_4 = 1.02,, x_5 = 1.02, x_6 = 1.02, x_7 = 1.02,\ x_8 = 1.02,\ x_9 = 1.02,\ x_9 = 1.02,)$

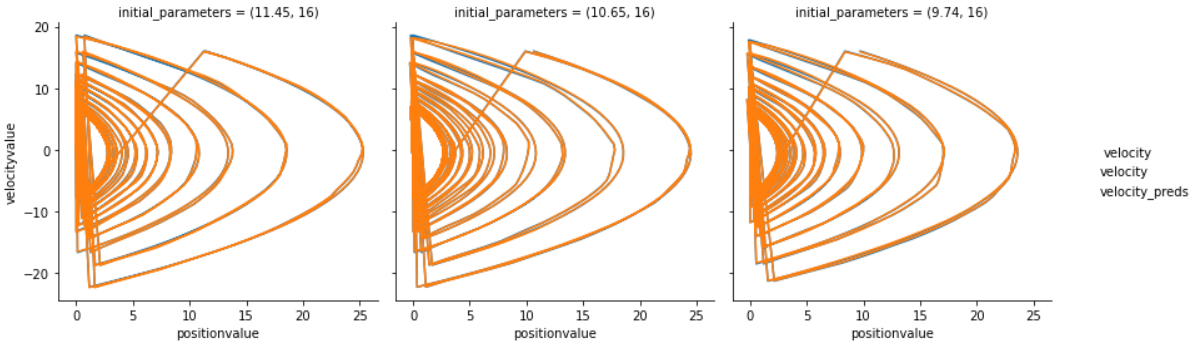
17. Bouncing ball: Predictions compared to numerical simulations

a. Training data



This is the training data compared with the predictions of the networks. Blue denotes the training data and orange denotes the predictions of the neural network

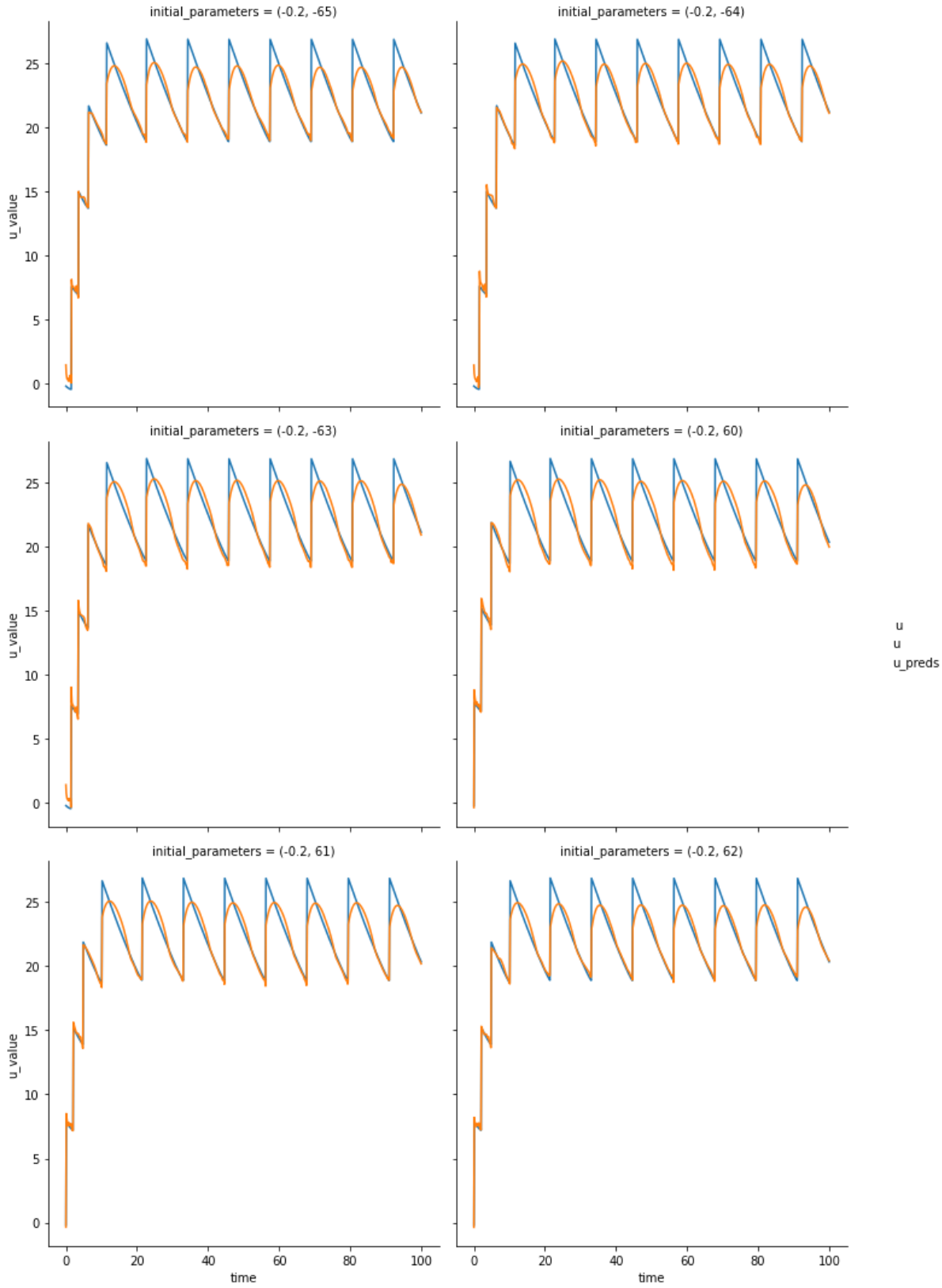
b. Test data



This is similar to appendix 17.a, but we are comparing predictions of the testing data.

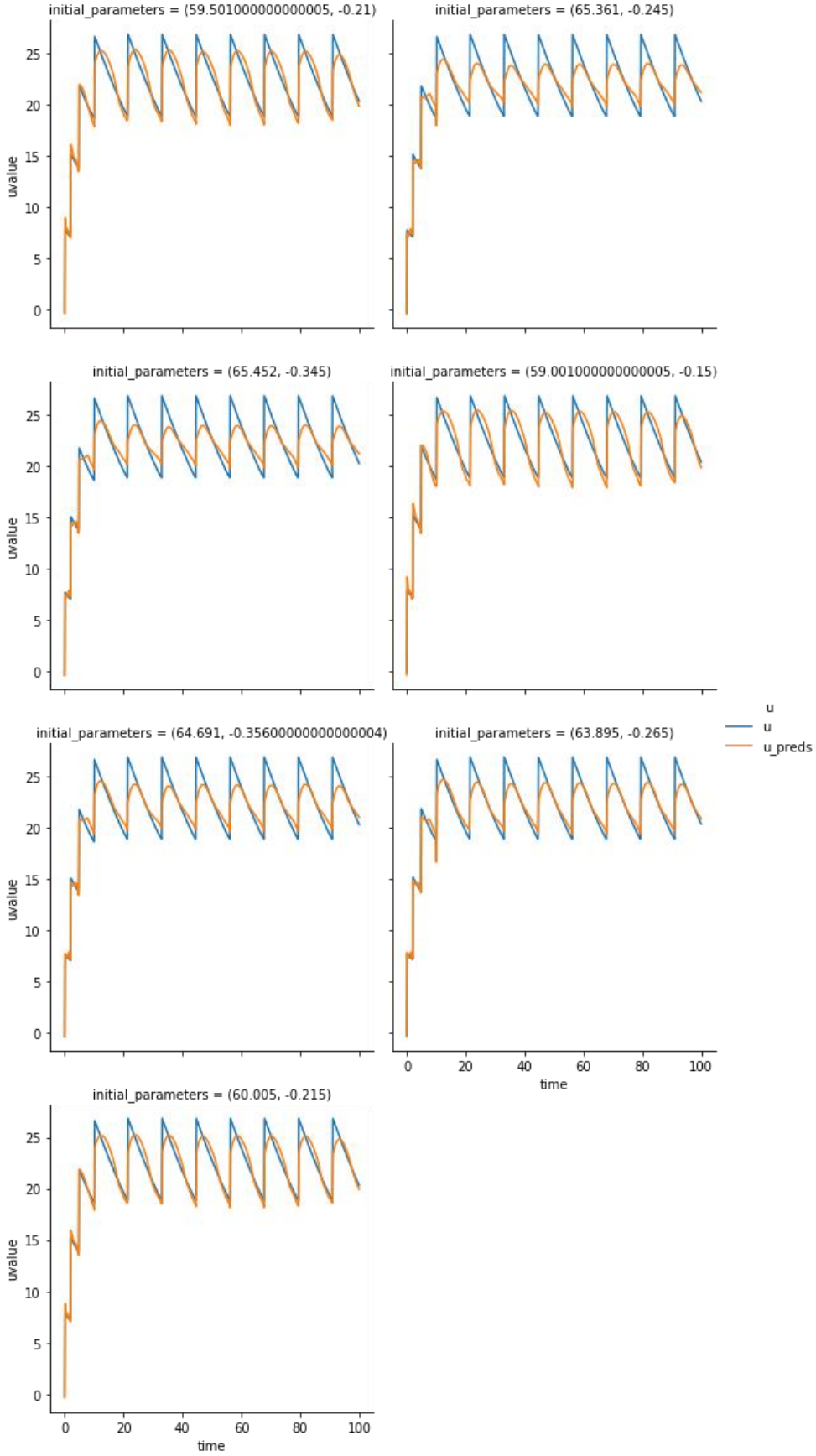
18. Spiking neurons: Predictions compared to numerical simulations

a. Training data



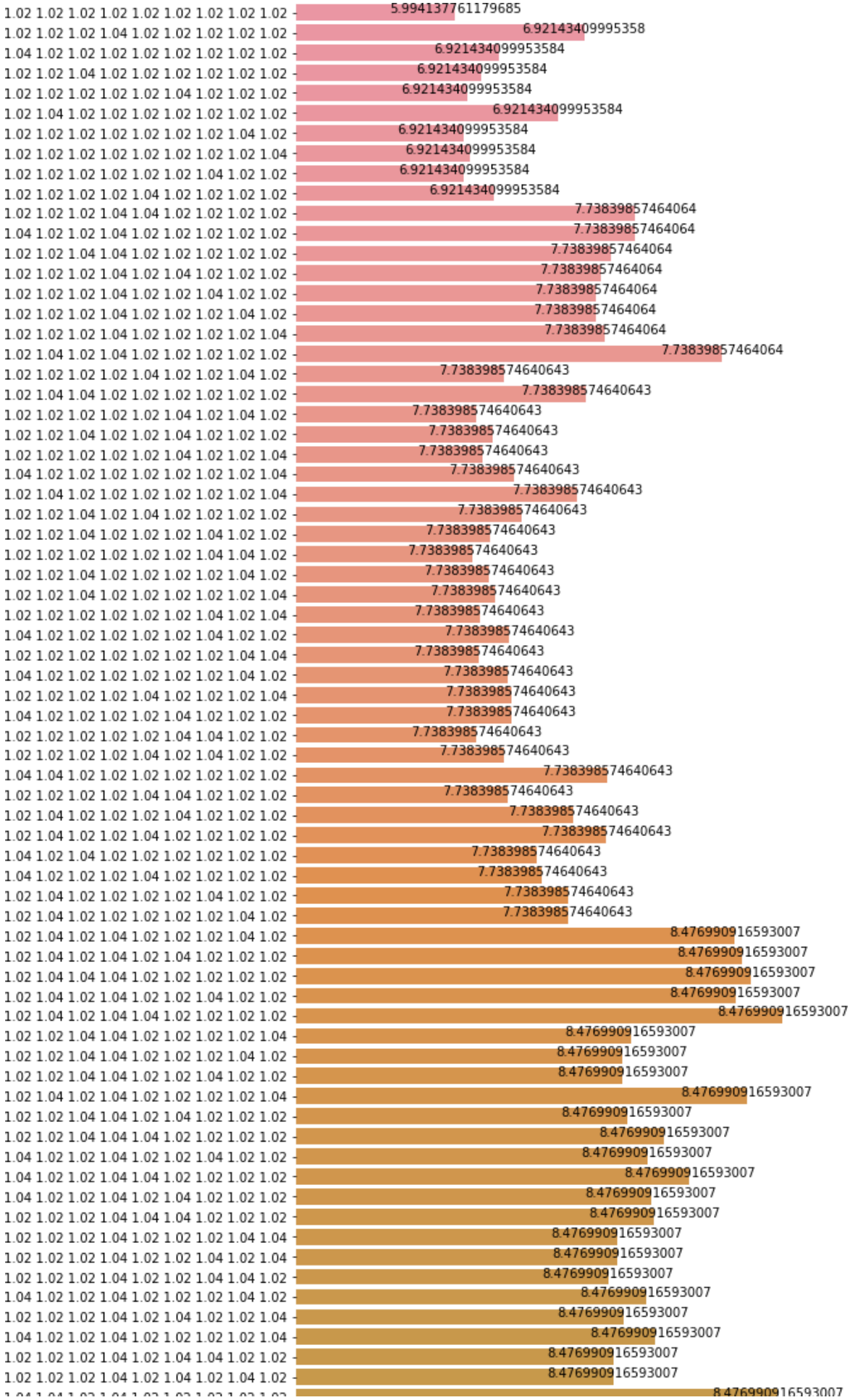
The training data of spiking neurons compared the predictions of the neural networks. The orange denotes the predictions and blue denotes the orange. This graph plots time against the u.

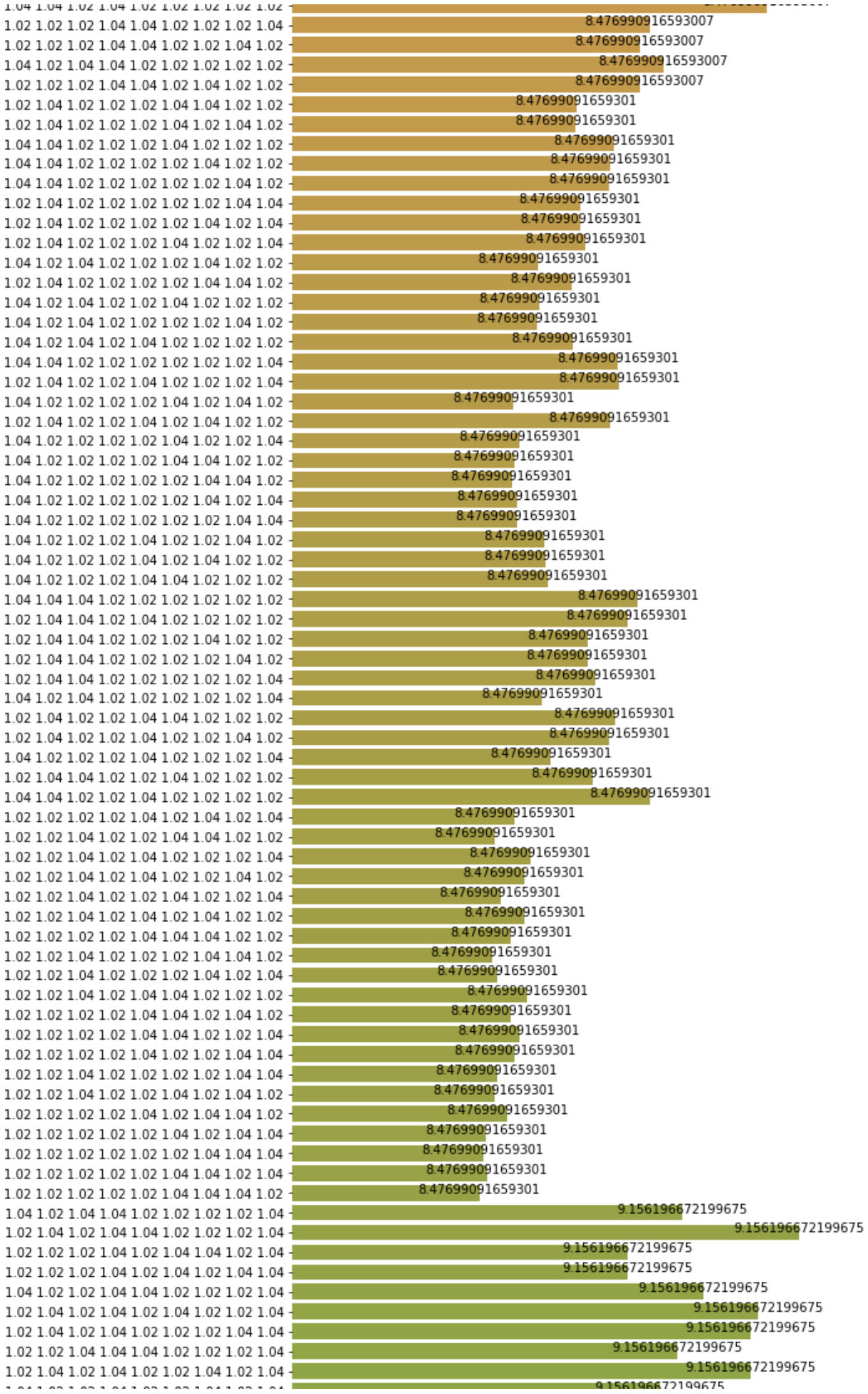
b. Test data

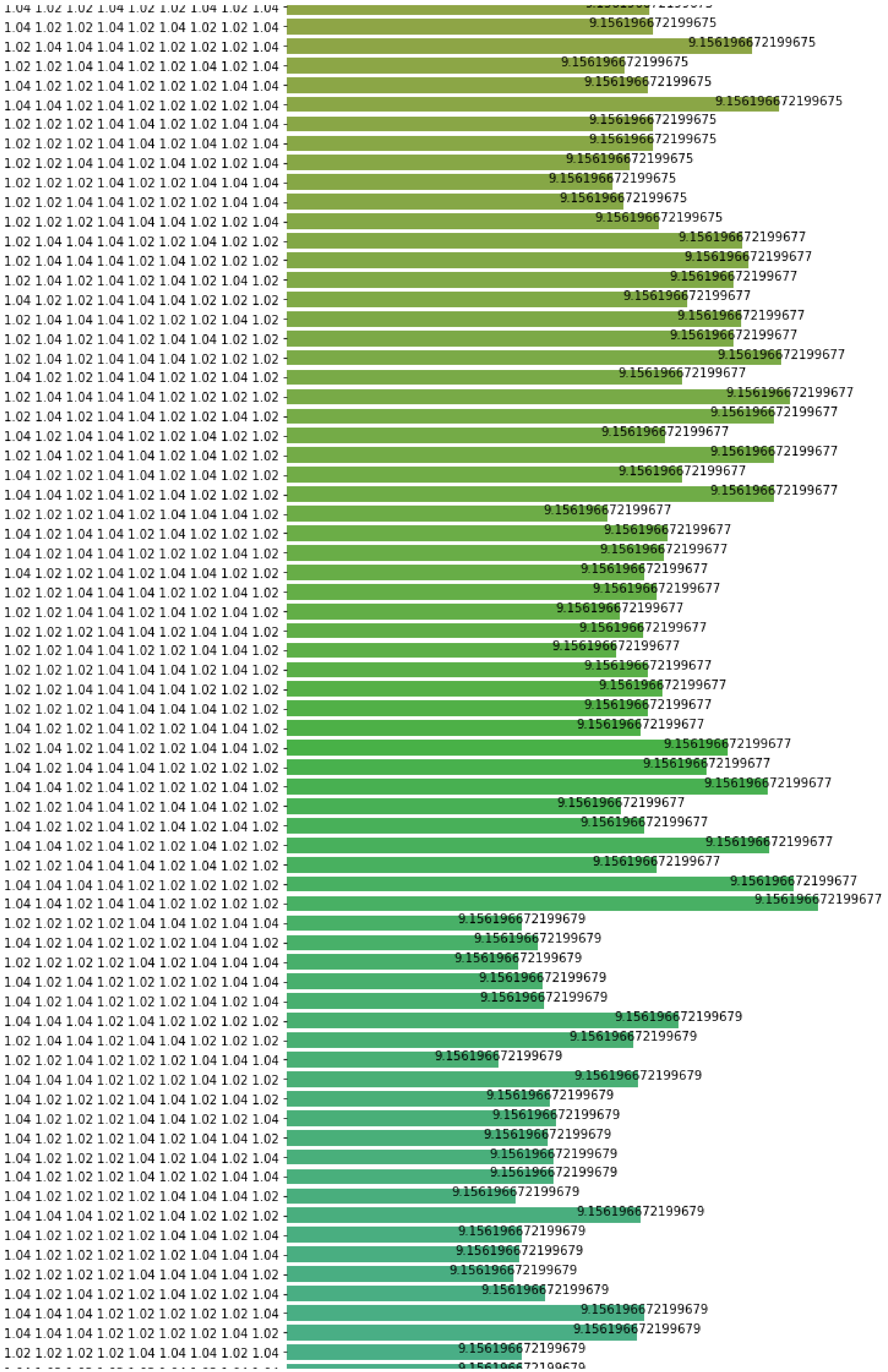


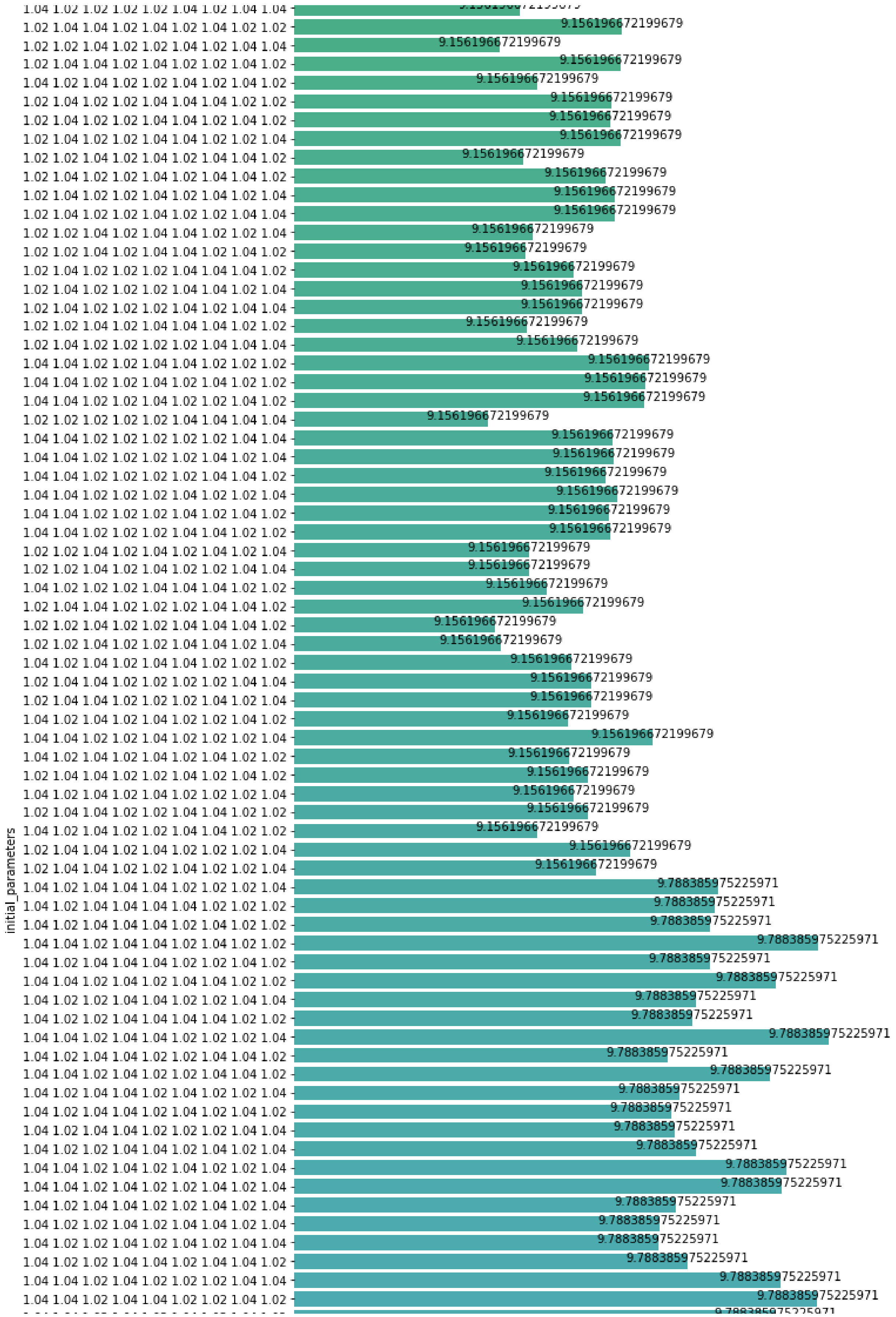
This is similar to appendix 18.a. However, were comparing the predictions and the testing data.

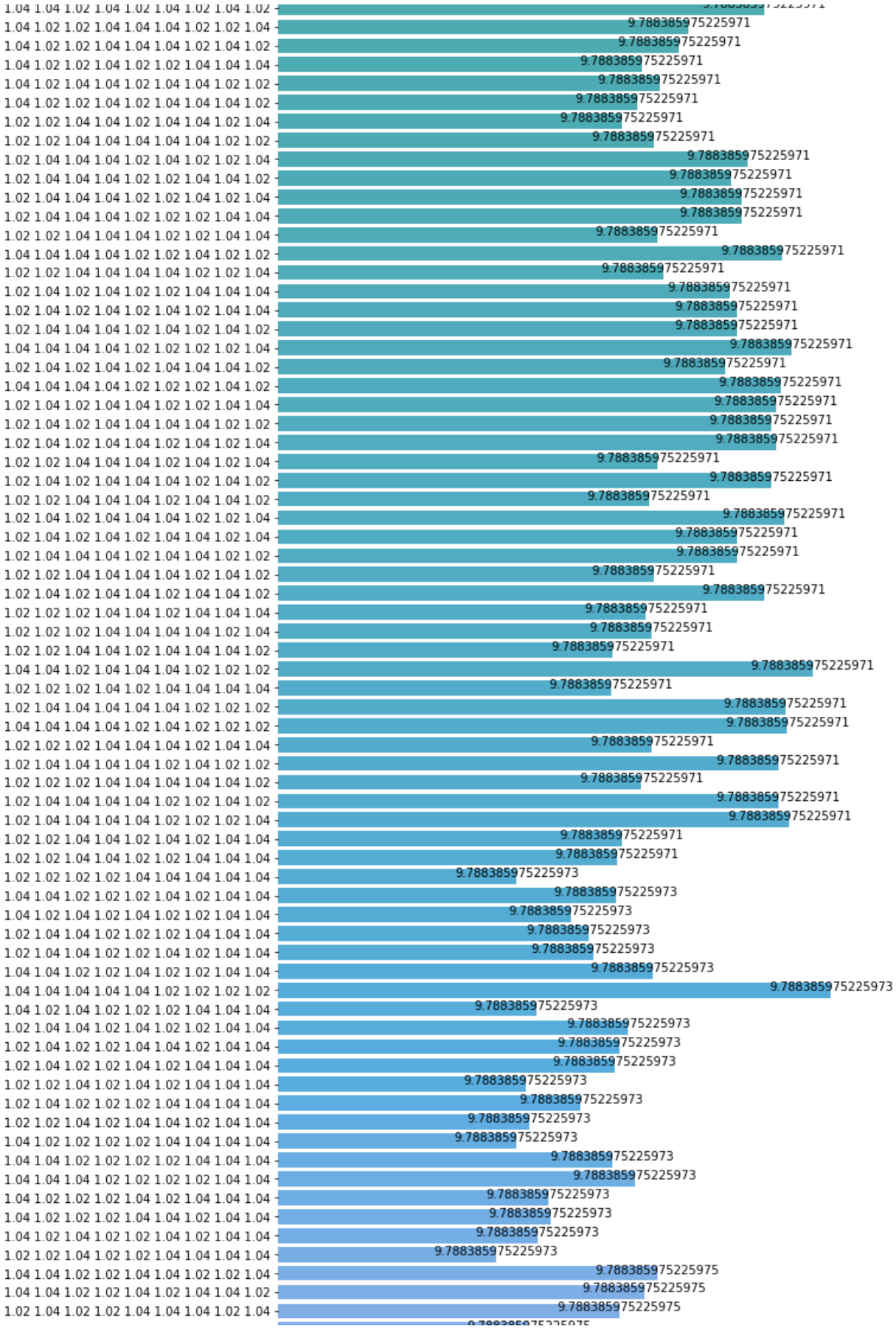
19. Loss of sample simulations within the test data of the biological model

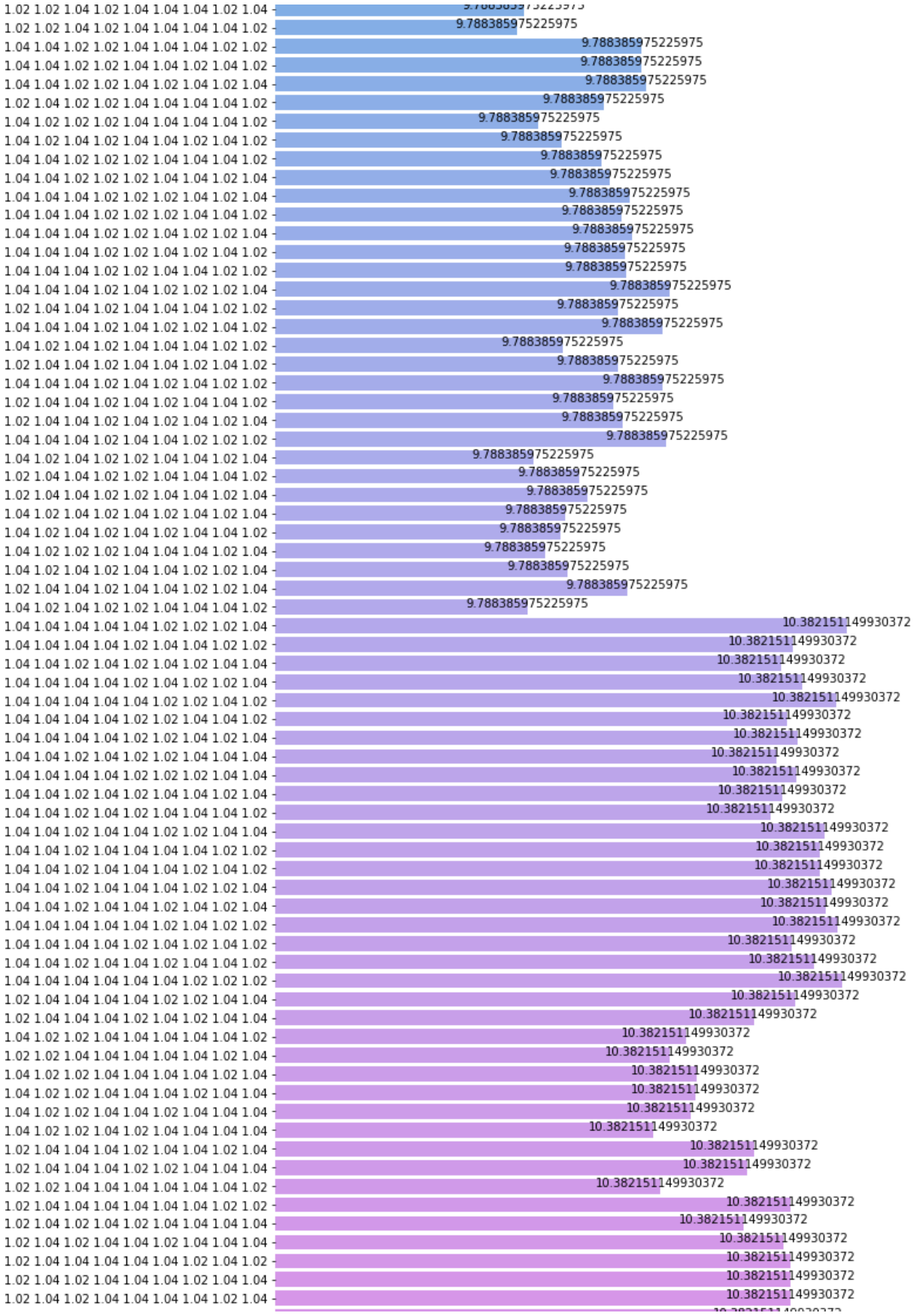

1.02 1.02 1.02 1.02 1.02 1.02 1.02 1.02 1.02
5.994137761179685
1.02 1.02 1.02 1.04 1.02 1.02 1.02 1.02 1.02
6.92143409995358
1.04 1.02 1.02 1.02 1.02 1.02 1.02 1.02 1.02
6.921434099953584
1.02 1.02 1.04 1.02 1.02 1.02 1.02 1.02 1.02
6.921434099953584
1.02 1.02 1.02 1.02 1.02 1.04 1.02 1.02 1.02
6.921434099953584
1.02 1.04 1.02 1.02 1.02 1.02 1.02 1.02 1.02
6.921434099953584
1.02 1.02 1.02 1.02 1.02 1.02 1.02 1.04 1.02
6.921434099953584
1.02 1.02 1.02 1.02 1.02 1.02 1.02 1.02 1.04
6.921434099953584
1.02 1.02 1.02 1.02 1.02 1.02 1.04 1.02 1.02
6.921434099953584
1.02 1.02 1.02 1.02 1.04 1.02 1.02 1.02 1.02
6.921434099953584
1.02 1.02 1.02 1.04 1.04 1.02 1.02 1.02 1.02
7.73839857464064
1.04 1.02 1.02 1.04 1.02 1.02 1.02 1.02 1.02
7.73839857464064
1.02 1.02 1.04 1.04 1.02 1.02 1.02 1.02 1.02
7.73839857464064
1.02 1.02 1.02 1.04 1.02 1.04 1.02 1.02 1.02
7.73839857464064
1.02 1.02 1.02 1.04 1.02 1.02 1.04 1.02 1.02
7.73839857464064
1.02 1.02 1.02 1.04 1.02 1.02 1.02 1.04 1.02
7.73839857464064
1.02 1.02 1.02 1.04 1.02 1.02 1.02 1.02 1.04
7.73839857464064
1.02 1.04 1.02 1.04 1.02 1.02 1.02 1.02 1.02
7.73839857464064
1.02 1.02 1.02 1.02 1.04 1.02 1.02 1.04 1.02
7.738398574640643
1.02 1.04 1.04 1.02 1.02 1.02 1.02 1.02 1.02
7.738398574640643
1.02 1.02 1.02 1.02 1.02 1.04 1.02 1.04 1.02
7.738398574640643
1.02 1.02 1.04 1.02 1.02 1.04 1.02 1.02 1.02
7.738398574640643
1.02 1.02 1.02 1.02 1.02 1.04 1.02 1.02 1.04
7.738398574640643
1.04 1.02 1.02 1.02 1.02 1.02 1.02 1.02 1.04
7.738398574640643
1.02 1.04 1.02 1.02 1.02 1.02 1.02 1.02 1.04
7.738398574640643
1.02 1.02 1.04 1.02 1.04 1.02 1.02 1.02 1.02
7.738398574640643
1.02 1.02 1.04 1.02 1.02 1.02 1.04 1.02 1.02
7.738398574640643
1.02 1.02 1.02 1.02 1.02 1.02 1.04 1.04 1.02
7.738398574640643
1.02 1.02 1.04 1.02 1.02 1.02 1.02 1.04 1.02
7.738398574640643
1.02 1.02 1.04 1.02 1.02 1.02 1.02 1.02 1.04
7.738398574640643
1.02 1.02 1.02 1.02 1.02 1.02 1.04 1.02 1.04
7.738398574640643
1.04 1.02 1.02 1.02 1.02 1.02 1.04 1.02 1.02
7.738398574640643
1.02 1.02 1.02 1.02 1.02 1.02 1.02 1.04 1.04
7.738398574640643
1.04 1.02 1.02 1.02 1.02 1.02 1.02 1.04 1.02
7.738398574640643
1.02 1.02 1.02 1.02 1.04 1.02 1.02 1.02 1.04
7.738398574640643
1.04 1.02 1.02 1.02 1.02 1.04 1.02 1.02 1.02
7.738398574640643
1.02 1.02 1.02 1.02 1.02 1.04 1.04 1.02 1.02
7.738398574640643
1.02 1.02 1.02 1.02 1.04 1.02 1.04 1.02 1.02
7.738398574640643
1.04 1.04 1.02 1.02 1.02 1.02 1.02 1.02 1.02
7.738398574640643
1.02 1.02 1.02 1.02 1.04 1.04 1.02 1.02 1.02
7.738398574640643
1.02 1.04 1.02 1.02 1.02 1.04 1.02 1.02 1.02
7.738398574640643
1.02 1.04 1.02 1.02 1.04 1.02 1.02 1.02 1.02
7.738398574640643
1.04 1.02 1.04 1.02 1.02 1.02 1.02 1.02 1.02
7.738398574640643
1.04 1.02 1.02 1.02 1.04 1.02 1.02 1.02 1.02
7.738398574640643
1.02 1.04 1.02 1.02 1.02 1.02 1.04 1.02 1.02
7.738398574640643
1.02 1.04 1.02 1.02 1.02 1.02 1.02 1.04 1.02
7.738398574640643
1.02 1.04 1.02 1.04 1.02 1.02 1.02 1.04 1.02
8.476990916593007
1.02 1.04 1.02 1.04 1.02 1.04 1.02 1.02 1.02
8.476990916593007
1.02 1.04 1.04 1.04 1.02 1.02 1.02 1.02 1.02
8.476990916593007
1.02 1.04 1.02 1.04 1.02 1.02 1.04 1.02 1.02
8.476990916593007
1.02 1.04 1.02 1.04 1.04 1.02 1.02 1.02 1.02
8.476990916593007
1.02 1.02 1.04 1.04 1.02 1.02 1.02 1.02 1.04
8.476990916593007
1.02 1.02 1.04 1.04 1.02 1.02 1.02 1.04 1.02
8.476990916593007
1.02 1.02 1.04 1.04 1.02 1.02 1.04 1.02 1.02
8.476990916593007
1.02 1.04 1.02 1.04 1.02 1.02 1.02 1.02 1.04
8.476990916593007
1.02 1.02 1.04 1.04 1.02 1.04 1.02 1.02 1.02
8.476990916593007
1.02 1.02 1.04 1.04 1.04 1.02 1.02 1.02 1.02
8.476990916593007
1.04 1.02 1.02 1.04 1.02 1.02 1.04 1.02 1.02
8.476990916593007
1.04 1.02 1.02 1.04 1.04 1.02 1.02 1.02 1.02
8.476990916593007
1.04 1.02 1.02 1.04 1.02 1.04 1.02 1.02 1.02
8.476990916593007
1.02 1.02 1.02 1.04 1.04 1.04 1.02 1.02 1.02
8.476990916593007
1.02 1.02 1.02 1.04 1.02 1.02 1.02 1.04 1.04
8.476990916593007
1.02 1.02 1.02 1.04 1.02 1.02 1.04 1.02 1.04
8.476990916593007
1.02 1.02 1.02 1.04 1.02 1.02 1.04 1.04 1.02
8.476990916593007
1.04 1.02 1.02 1.04 1.02 1.02 1.02 1.04 1.02
8.476990916593007
1.02 1.02 1.02 1.04 1.02 1.04 1.02 1.02 1.04
8.476990916593007
1.04 1.02 1.02 1.04 1.02 1.02 1.02 1.02 1.04
8.476990916593007
1.02 1.02 1.02 1.04 1.02 1.04 1.04 1.02 1.02
8.476990916593007
1.02 1.02 1.02 1.04 1.02 1.04 1.02 1.04 1.02
8.476990916593007
8.476990916593007

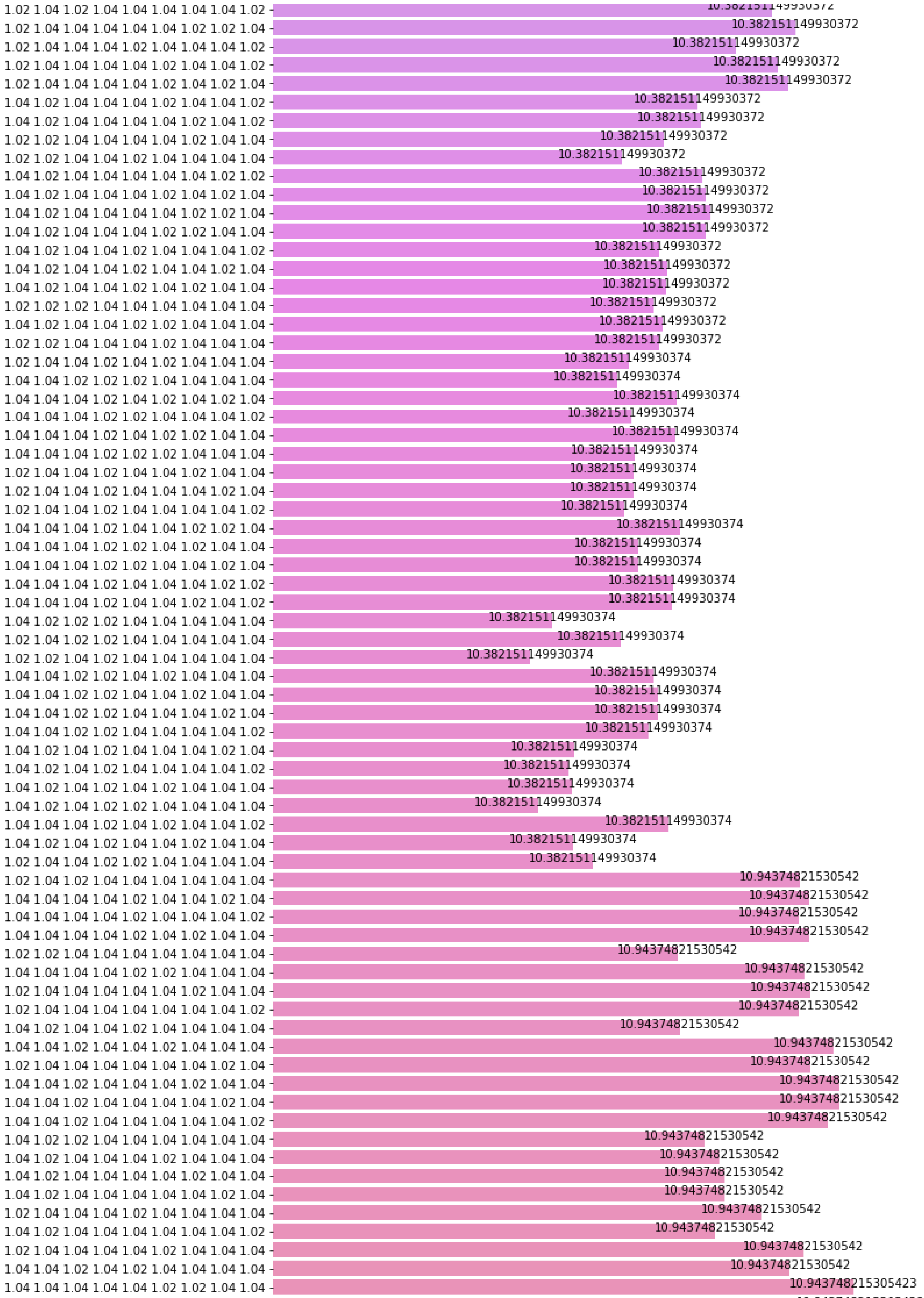

1.04 1.04 1.02 1.04 1.02 1.02 1.02 1.02 1.02
8.476990916593007
1.02 1.02 1.02 1.04 1.04 1.02 1.02 1.02 1.04
8.476990916593007
1.02 1.02 1.02 1.04 1.04 1.02 1.02 1.04 1.02
8.476990916593007
1.04 1.02 1.04 1.04 1.02 1.02 1.02 1.02 1.02
8.476990916593007
1.02 1.02 1.02 1.04 1.04 1.02 1.04 1.02 1.02
8.47699091659301
1.02 1.04 1.02 1.02 1.02 1.04 1.04 1.02 1.02
8.47699091659301
1.02 1.04 1.02 1.02 1.02 1.04 1.02 1.04 1.02
8.47699091659301
1.04 1.04 1.02 1.02 1.02 1.04 1.02 1.02 1.02
8.47699091659301
1.04 1.04 1.02 1.02 1.02 1.02 1.04 1.02 1.02
8.47699091659301
1.04 1.04 1.02 1.02 1.02 1.02 1.02 1.04 1.02
8.47699091659301
1.02 1.04 1.02 1.02 1.02 1.02 1.02 1.04 1.04
8.47699091659301
1.02 1.04 1.02 1.02 1.02 1.02 1.04 1.02 1.04
8.47699091659301
1.02 1.04 1.02 1.02 1.02 1.04 1.02 1.02 1.04
8.47699091659301
1.04 1.02 1.04 1.02 1.02 1.02 1.04 1.02 1.02
8.47699091659301
1.02 1.04 1.02 1.02 1.02 1.02 1.04 1.04 1.02
8.47699091659301
1.04 1.02 1.04 1.02 1.02 1.04 1.02 1.02 1.02
8.47699091659301
1.04 1.02 1.04 1.02 1.02 1.02 1.02 1.04 1.02
8.47699091659301
1.04 1.04 1.02 1.02 1.02 1.02 1.02 1.02 1.04
8.47699091659301
1.02 1.04 1.02 1.02 1.04 1.02 1.02 1.02 1.04
8.47699091659301
1.04 1.02 1.02 1.02 1.02 1.04 1.02 1.04 1.02
8.47699091659301
1.02 1.04 1.02 1.02 1.04 1.02 1.04 1.02 1.02
8.47699091659301
1.04 1.02 1.02 1.02 1.02 1.04 1.02 1.02 1.04
8.47699091659301
1.04 1.02 1.02 1.02 1.02 1.04 1.04 1.02 1.02
8.47699091659301
1.04 1.02 1.02 1.02 1.02 1.02 1.04 1.04 1.02
8.47699091659301
1.04 1.02 1.02 1.02 1.02 1.02 1.04 1.02 1.04
8.47699091659301
1.04 1.02 1.02 1.02 1.02 1.02 1.02 1.04 1.04
8.47699091659301
1.04 1.02 1.02 1.02 1.04 1.02 1.02 1.04 1.02
8.47699091659301
1.04 1.02 1.02 1.02 1.04 1.02 1.04 1.02 1.02
8.47699091659301
1.04 1.02 1.02 1.02 1.04 1.04 1.02 1.02 1.02
8.47699091659301
1.04 1.04 1.04 1.02 1.02 1.02 1.02 1.02 1.02
8.47699091659301
1.02 1.04 1.04 1.02 1.04 1.02 1.02 1.02 1.02
8.47699091659301
1.02 1.04 1.04 1.02 1.02 1.02 1.04 1.02 1.02
8.47699091659301
1.02 1.04 1.04 1.02 1.02 1.02 1.02 1.04 1.02
8.47699091659301
1.02 1.04 1.04 1.02 1.02 1.02 1.02 1.02 1.04
8.47699091659301
1.04 1.02 1.04 1.02 1.02 1.02 1.02 1.02 1.04
8.47699091659301
1.02 1.04 1.02 1.02 1.04 1.04 1.02 1.02 1.02
8.47699091659301
1.02 1.04 1.02 1.02 1.04 1.02 1.02 1.04 1.02
8.47699091659301
1.04 1.02 1.02 1.02 1.04 1.02 1.02 1.02 1.04
8.47699091659301
1.02 1.04 1.04 1.02 1.02 1.04 1.02 1.02 1.02
8.47699091659301
1.04 1.04 1.02 1.02 1.04 1.02 1.02 1.02 1.02
8.47699091659301
1.02 1.02 1.02 1.02 1.04 1.02 1.04 1.02 1.04
8.47699091659301
1.02 1.02 1.04 1.02 1.02 1.04 1.04 1.02 1.02
8.47699091659301
1.02 1.02 1.04 1.02 1.04 1.02 1.02 1.02 1.04
8.47699091659301
1.02 1.02 1.04 1.02 1.04 1.02 1.02 1.04 1.02
8.47699091659301
1.02 1.02 1.04 1.02 1.02 1.04 1.02 1.02 1.04
8.47699091659301
1.02 1.02 1.04 1.02 1.04 1.02 1.04 1.02 1.02
8.47699091659301
1.02 1.02 1.02 1.02 1.04 1.04 1.04 1.02 1.02
8.47699091659301
1.02 1.02 1.04 1.02 1.02 1.02 1.04 1.04 1.02
8.47699091659301
1.02 1.02 1.04 1.02 1.02 1.02 1.04 1.02 1.04
8.47699091659301
1.02 1.02 1.04 1.02 1.04 1.04 1.02 1.02 1.02
8.47699091659301
1.02 1.02 1.02 1.02 1.04 1.04 1.02 1.04 1.02
8.47699091659301
1.02 1.02 1.02 1.02 1.04 1.04 1.02 1.02 1.04
8.47699091659301
1.02 1.02 1.02 1.02 1.04 1.02 1.02 1.04 1.04
8.47699091659301
1.02 1.02 1.04 1.02 1.02 1.02 1.02 1.04 1.04
8.47699091659301
1.02 1.02 1.04 1.02 1.02 1.04 1.02 1.04 1.02
8.47699091659301
1.02 1.02 1.02 1.02 1.04 1.02 1.04 1.04 1.02
8.47699091659301
1.02 1.02 1.02 1.02 1.02 1.04 1.02 1.04 1.04
8.47699091659301
1.02 1.02 1.02 1.02 1.02 1.02 1.04 1.04 1.04
8.47699091659301
1.02 1.02 1.02 1.02 1.02 1.04 1.04 1.02 1.04
8.47699091659301
1.02 1.02 1.02 1.02 1.02 1.04 1.04 1.04 1.02
9.156196672199675
1.04 1.02 1.04 1.04 1.02 1.02 1.02 1.02 1.04
9.156196672199675
1.02 1.04 1.02 1.04 1.04 1.02 1.02 1.02 1.04
9.156196672199675
1.02 1.02 1.02 1.04 1.02 1.04 1.04 1.02 1.04
9.156196672199675
1.02 1.02 1.02 1.04 1.02 1.04 1.02 1.04 1.04
9.156196672199675
1.04 1.02 1.02 1.04 1.04 1.02 1.02 1.02 1.04
9.156196672199675
1.02 1.04 1.02 1.04 1.02 1.04 1.02 1.02 1.04
9.156196672199675
1.02 1.04 1.02 1.04 1.02 1.02 1.02 1.04 1.04
9.156196672199675
1.02 1.02 1.04 1.04 1.04 1.02 1.02 1.02 1.04
9.156196672199675
1.02 1.04 1.02 1.04 1.02 1.02 1.04 1.02 1.04
9.156196672199675

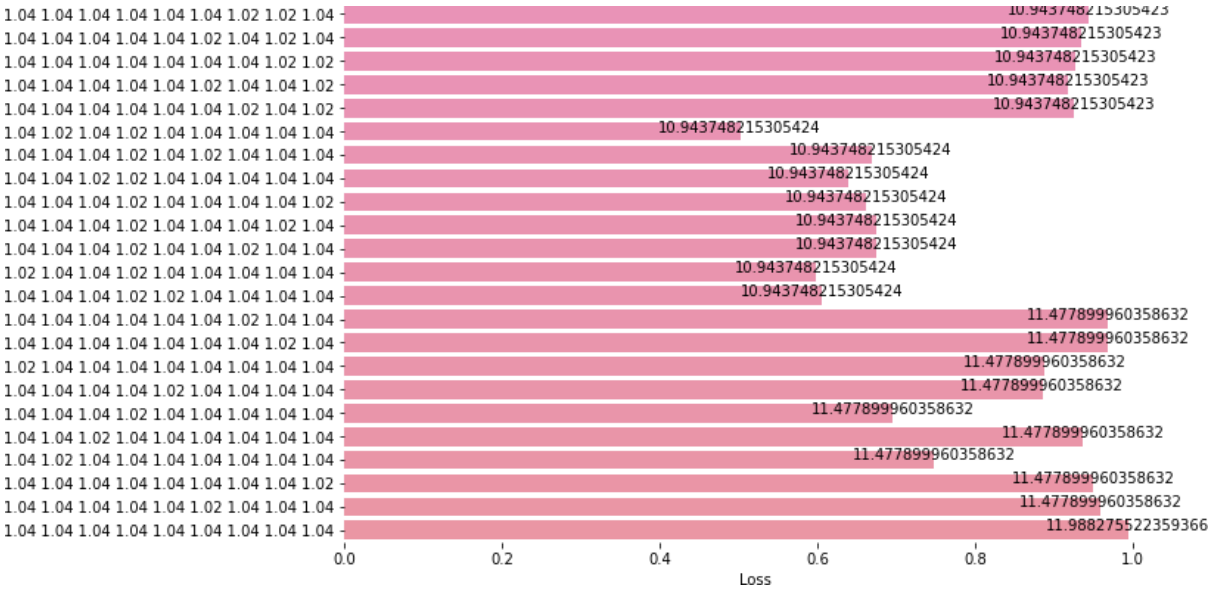

1.04 1.02 1.02 1.04 1.02 1.04 1.02 1.02 1.04
9.156196672199675
1.02 1.04 1.04 1.04 1.02 1.02 1.02 1.02 1.04
9.156196672199675
1.02 1.02 1.04 1.04 1.02 1.02 1.04 1.02 1.04
9.156196672199675
1.04 1.02 1.02 1.04 1.02 1.02 1.02 1.04 1.04
9.156196672199675
1.04 1.04 1.02 1.04 1.02 1.02 1.02 1.02 1.04
9.156196672199675
1.02 1.02 1.02 1.04 1.04 1.02 1.02 1.04 1.04
9.156196672199675
1.02 1.02 1.02 1.04 1.04 1.02 1.04 1.02 1.04
9.156196672199675
1.02 1.02 1.04 1.04 1.02 1.04 1.02 1.02 1.04
9.156196672199675
1.02 1.02 1.02 1.04 1.02 1.02 1.04 1.04 1.04
9.156196672199675
1.02 1.02 1.04 1.04 1.02 1.02 1.02 1.04 1.04
9.156196672199675
1.02 1.02 1.02 1.04 1.04 1.04 1.02 1.02 1.04
9.156196672199675
1.02 1.04 1.04 1.04 1.02 1.02 1.04 1.02 1.02
9.156196672199677
1.02 1.04 1.04 1.04 1.02 1.04 1.02 1.02 1.02
9.156196672199677
1.02 1.04 1.02 1.04 1.02 1.04 1.02 1.04 1.02
9.156196672199677
1.04 1.02 1.02 1.04 1.04 1.04 1.02 1.02 1.02
9.156196672199677
1.02 1.04 1.04 1.04 1.02 1.02 1.02 1.04 1.02
9.156196672199677
1.02 1.04 1.02 1.04 1.02 1.04 1.04 1.02 1.02
9.156196672199677
1.02 1.04 1.02 1.04 1.04 1.04 1.02 1.02 1.02
9.156196672199677
1.04 1.02 1.02 1.04 1.04 1.02 1.02 1.04 1.02
9.156196672199677
1.02 1.04 1.04 1.04 1.04 1.02 1.02 1.02 1.02
9.156196672199677
1.02 1.04 1.02 1.04 1.04 1.02 1.02 1.04 1.02
9.156196672199677
1.04 1.02 1.04 1.04 1.02 1.02 1.04 1.02 1.02
9.156196672199677
1.02 1.04 1.02 1.04 1.04 1.02 1.04 1.02 1.02
9.156196672199677
1.04 1.02 1.02 1.04 1.04 1.02 1.04 1.02 1.02
9.156196672199677
1.04 1.04 1.02 1.04 1.02 1.04 1.02 1.02 1.02
9.156196672199677
1.02 1.02 1.02 1.04 1.02 1.04 1.04 1.04 1.02
9.156196672199677
1.04 1.02 1.04 1.04 1.02 1.04 1.02 1.02 1.02
9.156196672199677
1.04 1.02 1.04 1.04 1.02 1.02 1.02 1.04 1.02
9.156196672199677
1.04 1.02 1.02 1.04 1.02 1.04 1.04 1.02 1.02
9.156196672199677
1.02 1.02 1.04 1.04 1.04 1.02 1.02 1.04 1.02
9.156196672199677
1.02 1.02 1.04 1.04 1.02 1.04 1.02 1.04 1.02
9.156196672199677
1.02 1.02 1.02 1.04 1.04 1.02 1.04 1.04 1.02
9.156196672199677
1.02 1.02 1.04 1.04 1.02 1.02 1.04 1.04 1.02
9.156196672199677
1.02 1.02 1.02 1.04 1.04 1.04 1.02 1.04 1.02
9.156196672199677
1.02 1.02 1.04 1.04 1.04 1.04 1.02 1.02 1.02
9.156196672199677
1.02 1.02 1.02 1.04 1.04 1.04 1.04 1.02 1.02
9.156196672199677
1.04 1.02 1.02 1.04 1.02 1.02 1.04 1.04 1.02
9.156196672199677
1.02 1.04 1.02 1.04 1.02 1.02 1.04 1.04 1.02
9.156196672199677
1.04 1.02 1.04 1.04 1.04 1.02 1.02 1.02 1.02
9.156196672199677
1.04 1.04 1.02 1.04 1.02 1.02 1.02 1.04 1.02
9.156196672199677
1.02 1.02 1.04 1.04 1.02 1.04 1.04 1.02 1.02
9.156196672199677
1.04 1.02 1.02 1.04 1.02 1.04 1.02 1.04 1.02
9.156196672199677
1.04 1.04 1.02 1.04 1.02 1.02 1.04 1.02 1.02
9.156196672199677
1.02 1.02 1.04 1.04 1.04 1.02 1.04 1.02 1.02
9.156196672199677
1.04 1.04 1.04 1.04 1.02 1.02 1.02 1.02 1.02
9.156196672199677
1.04 1.04 1.02 1.04 1.04 1.02 1.02 1.02 1.02
9.156196672199677
1.02 1.02 1.02 1.02 1.04 1.04 1.02 1.04 1.04
9.156196672199679
1.04 1.02 1.04 1.02 1.02 1.02 1.04 1.04 1.02
9.156196672199679
1.02 1.02 1.02 1.02 1.04 1.02 1.04 1.04 1.04
9.156196672199679
1.04 1.02 1.04 1.02 1.02 1.02 1.02 1.04 1.04
9.156196672199679
1.04 1.02 1.04 1.02 1.02 1.02 1.04 1.02 1.04
9.156196672199679
1.04 1.04 1.04 1.02 1.04 1.02 1.02 1.02 1.02
9.156196672199679
1.02 1.04 1.04 1.02 1.04 1.04 1.02 1.02 1.02
9.156196672199679
1.02 1.02 1.04 1.02 1.02 1.02 1.04 1.04 1.04
9.156196672199679
1.04 1.04 1.04 1.02 1.02 1.02 1.04 1.02 1.02
9.156196672199679
1.04 1.02 1.02 1.02 1.04 1.04 1.02 1.04 1.02
9.156196672199679
1.04 1.02 1.02 1.02 1.04 1.04 1.02 1.02 1.04
9.156196672199679
1.04 1.02 1.02 1.02 1.04 1.02 1.04 1.04 1.02
9.156196672199679
1.04 1.02 1.02 1.02 1.04 1.02 1.04 1.02 1.04
9.156196672199679
1.04 1.02 1.02 1.02 1.04 1.02 1.02 1.04 1.04
9.156196672199679
1.04 1.02 1.02 1.02 1.02 1.04 1.04 1.04 1.02
9.156196672199679
1.04 1.04 1.04 1.02 1.02 1.04 1.02 1.02 1.02
9.156196672199679
1.04 1.02 1.02 1.02 1.02 1.04 1.04 1.02 1.04
9.156196672199679
1.04 1.02 1.02 1.02 1.02 1.02 1.04 1.04 1.04
9.156196672199679
1.02 1.02 1.02 1.02 1.04 1.04 1.04 1.04 1.02
9.156196672199679
1.04 1.02 1.04 1.02 1.02 1.04 1.02 1.02 1.04
9.156196672199679
1.04 1.04 1.04 1.02 1.02 1.02 1.02 1.02 1.04
9.156196672199679
1.04 1.04 1.04 1.02 1.02 1.02 1.02 1.04 1.02
9.156196672199679
1.02 1.02 1.02 1.02 1.04 1.04 1.04 1.02 1.04
9.156196672199679


initial_parameters
9.156196672199679
9.788385975225971


1.04 1.04 1.02 1.04 1.02 1.04 1.02 1.04 1.02
1.04 1.02 1.02 1.04 1.04 1.04 1.02 1.02 1.04
1.04 1.02 1.02 1.04 1.04 1.04 1.02 1.04 1.02
1.04 1.02 1.02 1.04 1.02 1.02 1.04 1.04 1.04
1.04 1.02 1.04 1.04 1.02 1.04 1.04 1.02 1.02
1.04 1.02 1.02 1.04 1.02 1.04 1.04 1.04 1.02
1.02 1.02 1.04 1.04 1.02 1.04 1.04 1.02 1.04
1.02 1.02 1.04 1.04 1.04 1.04 1.04 1.02 1.02
1.02 1.04 1.04 1.04 1.02 1.04 1.02 1.02 1.04
1.02 1.04 1.04 1.04 1.02 1.02 1.04 1.04 1.02
1.02 1.04 1.04 1.04 1.02 1.02 1.04 1.02 1.04
1.02 1.04 1.04 1.04 1.02 1.02 1.02 1.04 1.04
1.02 1.02 1.04 1.04 1.04 1.02 1.02 1.04 1.04
1.04 1.04 1.04 1.04 1.02 1.02 1.04 1.02 1.02
1.02 1.02 1.04 1.04 1.04 1.04 1.02 1.02 1.04
1.02 1.04 1.02 1.04 1.02 1.02 1.04 1.04 1.04
1.04 1.04 1.04 1.04 1.02 1.02 1.02 1.02 1.04
1.02 1.04 1.02 1.04 1.04 1.04 1.02 1.04 1.02
1.04 1.04 1.04 1.04 1.02 1.02 1.02 1.04 1.04
1.02 1.04 1.04 1.04 1.04 1.02 1.02 1.02 1.04
1.02 1.04 1.02 1.04 1.04 1.04 1.04 1.02 1.02
1.02 1.02 1.04 1.04 1.04 1.02 1.04 1.02 1.04
1.02 1.02 1.04 1.04 1.04 1.02 1.04 1.04 1.02
1.02 1.04 1.02 1.04 1.04 1.04 1.04 1.02 1.02
1.02 1.04 1.04 1.04 1.02 1.04 1.04 1.02 1.04
1.02 1.02 1.04 1.04 1.04 1.02 1.04 1.02 1.04
1.02 1.04 1.02 1.04 1.04 1.04 1.02 1.04 1.02
1.02 1.02 1.04 1.04 1.04 1.02 1.04 1.04 1.02
1.02 1.04 1.02 1.04 1.04 1.04 1.04 1.02 1.02
1.02 1.02 1.02 1.04 1.04 1.02 1.04 1.04 1.04
1.02 1.02 1.04 1.04 1.02 1.04 1.04 1.04 1.02
1.04 1.04 1.02 1.04 1.04 1.04 1.02 1.02 1.02
1.02 1.02 1.02 1.04 1.02 1.04 1.04 1.04 1.04
1.02 1.02 1.04 1.04 1.02 1.04 1.04 1.04 1.02
1.02 1.04 1.04 1.04 1.04 1.04 1.02 1.02 1.02
1.04 1.04 1.04 1.04 1.02 1.04 1.02 1.02 1.02
1.02 1.02 1.02 1.04 1.04 1.04 1.02 1.04 1.04
1.02 1.04 1.04 1.04 1.04 1.02 1.04 1.02 1.02
1.02 1.02 1.02 1.04 1.04 1.04 1.04 1.04 1.02
1.02 1.04 1.04 1.04 1.04 1.02 1.02 1.04 1.02
1.02 1.04 1.04 1.04 1.04 1.02 1.02 1.02 1.04
1.04 1.04 1.02 1.04 1.04 1.04 1.02 1.02 1.02
1.02 1.02 1.04 1.04 1.02 1.04 1.04 1.04 1.04
1.02 1.02 1.04 1.04 1.02 1.04 1.04 1.04 1.04
1.02 1.02 1.02 1.02 1.04 1.04 1.04 1.04 1.04
1.04 1.04 1.02 1.02 1.02 1.04 1.02 1.04 1.04
1.04 1.02 1.04 1.02 1.04 1.02 1.02 1.04 1.04
1.02 1.04 1.04 1.02 1.02 1.02 1.04 1.04 1.04
1.02 1.04 1.02 1.02 1.04 1.04 1.02 1.04 1.04
1.02 1.04 1.04 1.02 1.02 1.04 1.02 1.04 1.04
1.04 1.04 1.02 1.02 1.04 1.02 1.02 1.04 1.04
1.04 1.04 1.04 1.04 1.04 1.02 1.02 1.02 1.02
1.04 1.02 1.04 1.02 1.02 1.02 1.04 1.04 1.04
1.02 1.04 1.04 1.02 1.04 1.02 1.02 1.04 1.04
1.02 1.04 1.02 1.02 1.04 1.04 1.02 1.04 1.04
1.02 1.04 1.04 1.02 1.02 1.04 1.02 1.04 1.04
1.02 1.02 1.04 1.02 1.04 1.02 1.04 1.04 1.04
1.02 1.04 1.02 1.02 1.02 1.04 1.04 1.04 1.04
1.02 1.02 1.04 1.02 1.04 1.04 1.02 1.04 1.04
1.04 1.02 1.04 1.02 1.02 1.04 1.04 1.04 1.04
1.04 1.04 1.02 1.02 1.02 1.02 1.04 1.04 1.04
1.04 1.02 1.02 1.02 1.04 1.02 1.04 1.04 1.04
1.04 1.02 1.02 1.02 1.04 1.04 1.02 1.04 1.04
1.04 1.02 1.04 1.02 1.02 1.04 1.02 1.04 1.04
1.02 1.02 1.04 1.02 1.02 1.04 1.04 1.04 1.04
1.04 1.04 1.02 1.02 1.04 1.04 1.02 1.02 1.04
1.04 1.04 1.02 1.02 1.04 1.02 1.04 1.04 1.02
1.02 1.04 1.02 1.02 1.04 1.04 1.04 1.02 1.04
9.788385975225971
9.788385975225973
9.788385975225975


1.02 1.02 1.04 1.02 1.04 1.04 1.04 1.02 1.04
9.788385975225975
1.02 1.02 1.04 1.02 1.04 1.04 1.04 1.04 1.02
9.788385975225975
1.04 1.04 1.02 1.02 1.04 1.04 1.04 1.02 1.02
9.788385975225975
1.04 1.04 1.02 1.02 1.04 1.04 1.02 1.04 1.02
9.788385975225975
1.04 1.04 1.02 1.02 1.04 1.02 1.04 1.02 1.04
9.788385975225975
1.02 1.04 1.02 1.02 1.04 1.04 1.04 1.04 1.02
9.788385975225975
1.04 1.02 1.02 1.02 1.04 1.04 1.04 1.04 1.02
9.788385975225975
1.04 1.02 1.04 1.02 1.04 1.04 1.02 1.04 1.02
9.788385975225975
1.04 1.04 1.02 1.02 1.02 1.04 1.04 1.04 1.02
9.788385975225975
1.04 1.04 1.02 1.02 1.02 1.04 1.04 1.02 1.04
9.788385975225975
1.04 1.04 1.04 1.02 1.02 1.02 1.04 1.02 1.04
9.788385975225975
1.04 1.04 1.04 1.02 1.02 1.02 1.04 1.04 1.02
9.788385975225975
1.04 1.04 1.04 1.02 1.02 1.04 1.02 1.02 1.04
9.788385975225975
1.04 1.04 1.04 1.02 1.02 1.04 1.02 1.04 1.02
9.788385975225975
1.04 1.04 1.04 1.02 1.02 1.04 1.04 1.02 1.02
9.788385975225975
1.04 1.04 1.04 1.02 1.04 1.02 1.02 1.02 1.04
9.788385975225975
1.02 1.04 1.04 1.02 1.04 1.04 1.04 1.02 1.04
9.788385975225975
1.04 1.04 1.04 1.02 1.04 1.02 1.04 1.04 1.02
9.788385975225975
1.04 1.04 1.04 1.02 1.04 1.02 1.04 1.02 1.02
9.788385975225975
1.02 1.04 1.04 1.02 1.04 1.04 1.02 1.02 1.02
9.788385975225975
1.04 1.04 1.04 1.02 1.04 1.04 1.02 1.02 1.02
9.788385975225975
1.04 1.02 1.04 1.02 1.02 1.04 1.04 1.02 1.04
9.788385975225975
1.02 1.04 1.04 1.02 1.02 1.04 1.04 1.04 1.02
9.788385975225975
1.02 1.04 1.04 1.02 1.02 1.04 1.04 1.02 1.04
9.788385975225975
1.04 1.02 1.04 1.02 1.04 1.02 1.04 1.02 1.04
9.788385975225975
1.04 1.02 1.04 1.02 1.04 1.02 1.04 1.04 1.02
9.788385975225975
1.04 1.02 1.02 1.02 1.04 1.04 1.04 1.02 1.04
9.788385975225975
1.04 1.02 1.04 1.02 1.04 1.04 1.02 1.02 1.04
9.788385975225975
1.02 1.04 1.04 1.02 1.04 1.04 1.02 1.02 1.04
9.788385975225975
1.04 1.02 1.04 1.02 1.02 1.04 1.04 1.04 1.02
9.788385975225975
1.04 1.04 1.04 1.04 1.04 1.02 1.02 1.02 1.04
10.382151149930372
1.04 1.04 1.04 1.04 1.02 1.04 1.04 1.02 1.02
10.382151149930372
1.04 1.04 1.02 1.04 1.02 1.04 1.02 1.04 1.04
10.382151149930372
1.04 1.04 1.04 1.04 1.02 1.04 1.02 1.02 1.04
10.382151149930372
1.04 1.04 1.04 1.04 1.04 1.02 1.02 1.04 1.02
10.382151149930372
1.04 1.04 1.04 1.04 1.02 1.02 1.04 1.04 1.02
10.382151149930372
1.04 1.04 1.04 1.04 1.02 1.02 1.04 1.02 1.04
10.382151149930372
1.04 1.04 1.02 1.04 1.02 1.02 1.04 1.04 1.04
10.382151149930372
1.04 1.04 1.04 1.04 1.02 1.02 1.02 1.04 1.04
10.382151149930372
1.04 1.04 1.02 1.04 1.02 1.04 1.04 1.02 1.04
10.382151149930372
1.04 1.04 1.02 1.04 1.02 1.04 1.04 1.04 1.02
10.382151149930372
1.04 1.04 1.02 1.04 1.04 1.02 1.02 1.04 1.04
10.382151149930372
1.04 1.04 1.02 1.04 1.04 1.04 1.04 1.02 1.02
10.382151149930372
1.04 1.04 1.02 1.04 1.04 1.04 1.02 1.04 1.02
10.382151149930372
1.04 1.04 1.02 1.04 1.04 1.02 1.04 1.04 1.02
10.382151149930372
1.04 1.04 1.04 1.04 1.04 1.02 1.04 1.02 1.02
10.382151149930372
1.04 1.04 1.04 1.04 1.02 1.04 1.02 1.04 1.02
10.382151149930372
1.04 1.04 1.02 1.04 1.04 1.02 1.04 1.04 1.02
10.382151149930372
1.04 1.04 1.04 1.04 1.04 1.04 1.02 1.02 1.02
10.382151149930372
1.02 1.04 1.04 1.04 1.04 1.04 1.02 1.04 1.04
10.382151149930372
1.02 1.04 1.04 1.04 1.02 1.04 1.02 1.04 1.04
10.382151149930372
1.04 1.02 1.02 1.04 1.04 1.04 1.04 1.04 1.02
10.382151149930372
1.02 1.02 1.04 1.04 1.04 1.04 1.04 1.02 1.04
10.382151149930372
1.04 1.02 1.02 1.04 1.04 1.04 1.04 1.02 1.04
10.382151149930372
1.04 1.02 1.02 1.04 1.04 1.04 1.02 1.04 1.04
10.382151149930372
1.04 1.02 1.02 1.04 1.04 1.02 1.04 1.04 1.04
10.382151149930372
1.04 1.02 1.02 1.04 1.02 1.04 1.04 1.04 1.04
10.382151149930372
1.02 1.04 1.04 1.04 1.04 1.02 1.04 1.04 1.02
10.382151149930372
1.02 1.04 1.04 1.04 1.02 1.02 1.04 1.04 1.04
10.382151149930372
1.02 1.02 1.04 1.04 1.04 1.04 1.04 1.04 1.02
10.382151149930372
1.02 1.04 1.04 1.04 1.04 1.04 1.04 1.02 1.02
10.382151149930372
1.02 1.04 1.02 1.04 1.02 1.04 1.04 1.04 1.04
10.382151149930372
1.02 1.04 1.02 1.04 1.04 1.02 1.04 1.04 1.04
10.382151149930372
1.02 1.04 1.04 1.04 1.04 1.04 1.02 1.04 1.02
10.382151149930372
1.02 1.04 1.02 1.04 1.04 1.04 1.02 1.04 1.04
10.382151149930372
1.02 1.04 1.02 1.04 1.04 1.04 1.04 1.02 1.04
10.382151149930372


1.02 1.04 1.02 1.04 1.04 1.04 1.04 1.04 1.02
1.02 1.04 1.04 1.04 1.04 1.04 1.02 1.02 1.04
10.382151149930372
1.02 1.04 1.04 1.04 1.02 1.04 1.04 1.04 1.02
10.382151149930372
1.02 1.04 1.04 1.04 1.04 1.02 1.04 1.04 1.02
10.382151149930372
1.02 1.04 1.04 1.04 1.04 1.02 1.04 1.02 1.04
10.382151149930372
1.04 1.02 1.04 1.04 1.04 1.02 1.04 1.04 1.02
10.382151149930372
1.04 1.02 1.04 1.04 1.04 1.04 1.02 1.04 1.02
10.382151149930372
1.02 1.02 1.04 1.04 1.04 1.04 1.02 1.04 1.04
10.382151149930372
1.02 1.02 1.04 1.04 1.02 1.04 1.04 1.04 1.04
10.382151149930372
1.04 1.02 1.04 1.04 1.04 1.04 1.04 1.02 1.02
10.382151149930372
1.04 1.02 1.04 1.04 1.04 1.02 1.04 1.02 1.04
10.382151149930372
1.04 1.02 1.04 1.04 1.04 1.04 1.02 1.02 1.04
10.382151149930372
1.04 1.02 1.04 1.04 1.04 1.02 1.02 1.04 1.04
10.382151149930372
1.04 1.02 1.04 1.04 1.02 1.04 1.04 1.04 1.02
10.382151149930372
1.04 1.02 1.04 1.04 1.02 1.04 1.04 1.02 1.04
10.382151149930372
1.04 1.02 1.04 1.04 1.02 1.04 1.02 1.04 1.04
10.382151149930372
1.02 1.02 1.02 1.04 1.04 1.04 1.04 1.04 1.04
10.382151149930372
1.04 1.02 1.04 1.04 1.02 1.02 1.04 1.04 1.04
10.382151149930372
1.02 1.02 1.04 1.04 1.04 1.02 1.04 1.04 1.04
10.382151149930372
1.02 1.04 1.04 1.02 1.04 1.02 1.04 1.04 1.04
10.382151149930374
1.04 1.04 1.02 1.02 1.02 1.04 1.04 1.04 1.04
10.382151149930374
1.04 1.04 1.04 1.02 1.04 1.02 1.04 1.02 1.04
10.382151149930374
1.04 1.04 1.04 1.02 1.02 1.04 1.04 1.04 1.02
10.382151149930374
1.04 1.04 1.04 1.02 1.04 1.02 1.02 1.04 1.04
10.382151149930374
1.04 1.04 1.04 1.02 1.02 1.02 1.04 1.04 1.04
10.382151149930374
1.02 1.04 1.04 1.02 1.04 1.04 1.02 1.04 1.04
10.382151149930374
1.02 1.04 1.04 1.02 1.04 1.04 1.04 1.02 1.04
10.382151149930374
1.02 1.04 1.04 1.02 1.04 1.04 1.04 1.04 1.02
10.382151149930374
1.04 1.04 1.04 1.02 1.04 1.04 1.02 1.02 1.04
10.382151149930374
1.04 1.04 1.04 1.02 1.02 1.04 1.02 1.04 1.04
10.382151149930374
1.04 1.04 1.04 1.02 1.02 1.04 1.04 1.02 1.04
10.382151149930374
1.04 1.04 1.04 1.02 1.04 1.04 1.04 1.02 1.02
10.382151149930374
1.04 1.04 1.04 1.02 1.04 1.04 1.02 1.04 1.02
10.382151149930374
1.04 1.02 1.02 1.02 1.04 1.04 1.04 1.04 1.04
10.382151149930374
1.02 1.04 1.02 1.02 1.04 1.04 1.04 1.04 1.04
10.382151149930374
1.02 1.02 1.04 1.02 1.04 1.04 1.04 1.04 1.04
10.382151149930374
1.04 1.04 1.02 1.02 1.04 1.02 1.04 1.04 1.04
10.382151149930374
1.04 1.04 1.02 1.02 1.04 1.04 1.02 1.04 1.04
10.382151149930374
1.04 1.04 1.02 1.02 1.04 1.04 1.04 1.02 1.04
10.382151149930374
1.04 1.04 1.02 1.02 1.04 1.04 1.04 1.04 1.02
10.382151149930374
1.04 1.02 1.04 1.02 1.04 1.04 1.04 1.02 1.04
10.382151149930374
1.04 1.02 1.04 1.02 1.04 1.04 1.04 1.04 1.02
10.382151149930374
1.04 1.02 1.04 1.02 1.04 1.02 1.04 1.04 1.04
10.382151149930374
1.04 1.02 1.04 1.02 1.02 1.04 1.04 1.04 1.04
10.382151149930374
1.04 1.04 1.04 1.02 1.04 1.02 1.04 1.04 1.02
10.382151149930374
1.04 1.02 1.04 1.02 1.04 1.04 1.02 1.04 1.04
10.382151149930374
1.02 1.04 1.04 1.02 1.02 1.04 1.04 1.04 1.04
10.382151149930374
1.02 1.04 1.02 1.04 1.04 1.04 1.04 1.04 1.04
10.94374821530542
1.04 1.04 1.04 1.04 1.02 1.04 1.04 1.02 1.04
10.94374821530542
1.04 1.04 1.04 1.04 1.02 1.04 1.04 1.04 1.02
10.94374821530542
1.04 1.04 1.04 1.04 1.02 1.04 1.02 1.04 1.04
10.94374821530542
1.02 1.02 1.04 1.04 1.04 1.04 1.04 1.04 1.04
10.94374821530542
1.04 1.04 1.04 1.04 1.02 1.02 1.04 1.04 1.04
10.94374821530542
1.02 1.04 1.04 1.04 1.04 1.04 1.02 1.04 1.04
10.94374821530542
1.02 1.04 1.04 1.04 1.04 1.04 1.04 1.04 1.02
10.94374821530542
1.04 1.02 1.04 1.04 1.02 1.04 1.04 1.04 1.04
10.94374821530542
1.04 1.04 1.02 1.04 1.04 1.02 1.04 1.04 1.04
10.94374821530542
1.02 1.04 1.04 1.04 1.04 1.04 1.04 1.02 1.04
10.94374821530542
1.04 1.04 1.02 1.04 1.04 1.04 1.02 1.04 1.04
10.94374821530542
1.04 1.04 1.02 1.04 1.04 1.04 1.04 1.02 1.04
10.94374821530542
1.04 1.04 1.02 1.04 1.04 1.04 1.04 1.04 1.02
10.94374821530542
1.04 1.02 1.02 1.04 1.04 1.04 1.04 1.04 1.04
10.94374821530542
1.04 1.02 1.04 1.04 1.04 1.02 1.04 1.04 1.04
10.94374821530542
1.04 1.02 1.04 1.04 1.04 1.04 1.02 1.04 1.04
10.94374821530542
1.04 1.02 1.04 1.04 1.04 1.04 1.04 1.02 1.04
10.94374821530542
1.02 1.04 1.04 1.04 1.02 1.04 1.04 1.04 1.04
10.94374821530542
1.04 1.02 1.04 1.04 1.04 1.04 1.04 1.04 1.02
10.94374821530542
1.02 1.04 1.04 1.04 1.04 1.02 1.04 1.04 1.04
10.94374821530542
1.04 1.04 1.02 1.04 1.02 1.04 1.04 1.04 1.04
10.94374821530542
1.04 1.04 1.04 1.04 1.04 1.02 1.02 1.04 1.04
10.943748215305423



20.

```
imports
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch import from_numpy
import torch.optim as optim # get optimisers
import torch.nn.functional as F # Relu function
from torch.utils.data import TensorDataset, DataLoader # For mini batches
from abc import ABC, abstractmethod
from sklearn.model_selection import train_test_split
from typing import Optional

class Training():
    """
        This class is used for training a neural network model
    """
    def device(self):
        """
            device:
                Returns the device being used
            Returns:
                (class <device>)
        """
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        return device

    def loss_function(self):
        """
            loss_function:
                This will is used to return the loss function
```

```
52.
53.             Returns:
54.             (<class 'torch.nn.MSELoss>'): This will return the loss function. This function
55.                 is currently using nn.MSELoss()
56.         """
57.         return nn.MSELoss()
58.
59.     @abstractmethod
60.     def gradient(self):
61.         pass
62.
63.     @abstractmethod
64.     def forward(self):
65.         pass
66.
67.     def train_model(self, training_batch_loader, num_epochs, test_inputs=None, test_targets=None):
68.         """
69.             training:
70.                 This will be used to train our model, we first define our gradient
71.                 we are using. We loop through the number of epoch, within that loop
72.                 go through the number of batches, and get the inputs and targets outputs
73.                 for each batch. We then caculate the loss using our loss_function.
74.
75.             Args:
76.                 training_batch (<class 'torch.utils.data.dataloader.DataLoader'>): This is the number of batches needed to
    train our
77.                                 model
78.                 num_epochs (int): The number of cycles repeated through the full training dataset.
79.
80.             Returns:
81.                 (float) : The loss
82.                 (<class 'pandas.core.frame.DataFrame'>) : A dataframe showing training and test loss at each epochs
83.         """
84.         opt = self.gradient()
85.         loss_function = self.loss_function()
86.
87.         # io is a boolean value to denote if the user would want to see the test and train loss at each epoch
88.         io = (test_inputs  != None and test_targets != None)
89.
90.         train_epochs = []
91.         test_epochs = []
92.
93.         for epoch in range(num_epochs):
94.             for inputs, outputs in training_batch_loader:
95.                 inputs = inputs.to(device=self.device())
96.                 outputs = outputs.to(device=self.device())
97.
98.                 # forward in the network
```

```
99.                 preds = self.forward(inputs)
100.
101.                # This will caculate the loss
102.                loss = loss_function(preds, outputs)
103.
104.                opt.zero_grad()
105.                loss.backward()
106.
107.                opt.step()
108.
109.            print(f"Epoch: {epoch}  Train Loss: {loss.item()}", end=" ")
110.
111.            if io:
112.                test_loss_func = self.loss_function()
113.                self.eval() # put on evaluation mode, that denotes trainin is false
114.                test_preds = self.forward(test_inputs) # put inputs into the function
115.
116.                # reshape is used to flatten the test targets
117.                test_loss = test_loss_func(test_preds, test_targets)
118.                print(f"Test Loss {test_loss.item()}", end=" ")
119.
120.                test_epochs.append(test_loss.item())
121.                train_epochs.append(loss.item())
122.
123.                self.train()
124.            print("")
125.
126.        if io:
127.            return pd.DataFrame(data = {"Train" : train_epochs, "Test" : test_epochs})
128.        else:
129.            return loss.item()
130.
131. class prototype(nn.Module, Training):
132.     """
133.         prototype:
134.             This is a neural network class. This will be used for predicting
135.             continous dynamics of a system. This is a regression neural network.
136.
137.         Attributes:
138.             layer_0 (<class 'torch.nn.modules.linear.Linear'>): The first layer within our neural networks
139.                     it uses the activation function y= xA^T + b
140.                     where x=input A=weights, b=bias.
141.             layer_1 (<class 'torch.nn.modules.linear.Linear'>): The second layer within our neural networks
142.                     it uses the activation function y= xA^T + b
143.                     where x=input A=weights, b=bias.
144.             layer_2 (<class 'torch.nn.modules.linear.Linear'>): The is the third layer within our neural network.
145.                     This is also the last layer of our neural network.
146.                     This uses the activation function to predict the
```

```
                output
        device (<class 'torch.device'>): Where the tensor caculations will be executed. Either
                the CPU or a GPU. Where the user has a GPU, the neural
                network is caculated there. If not the CPU, is used
                instead.
        learning_rate (float):  This is the learning rate we want our gradient descent to perform
    """
    def __init__(self, num_inputs, num_classes, learning_rate):
        """
        __init__:
            This is used to initilise our class when creating an object

        Args:
            num_inputs (int): The number of inputs we are using wihtin our class
            num_classes (int): The number of classes we are using
            learning_rate (float): The learning for our optimiser

        """
        super(prototype, self).__init__()

        self.layer_0 = nn.Linear(num_inputs, 50)
        self.layer_1 = nn.Linear(50, 100)
        self.layer_2 = nn.Linear(100, 200)
        self.layer_3 = nn.Linear(200, 400)
        self.layer_4 = nn.Linear(400, num_classes)
        self.learning_rate = learning_rate

    def forward(self, inputs):
        """
        feed:
            The input data (inputs) is fed in the forward direction through the network.
            It goes through the first layer (layer_0), both applying the activation function
            Each hidden layer accepts the input data, processes it as per the
            activation function and passes to the successive layer.

        Args:
            inputs (<class 'torch.Tensor'>): A tensor of inputs for the neural networks

        Returns:
            (<class 'torch.Tensor'>): A tensor of predictions provided by our neural network
        """
        inputs = inputs.reshape(inputs.shape[0], -1) # To ensure outputs is all within 1d vector
        outputs = self.layer_0(inputs) # apply the linear function
        outputs = F.relu(outputs) # Then apply the activation function to layer_0

        outputs = self.layer_1(outputs) # apply the linear function to layer_1
        outputs = F.relu(outputs) # apply the activation function to layer_1

```

```
195.         outputs = self.layer_2(outputs) # apply the linear function to layer_2
196.         outputs = F.relu(outputs) # apply the activation function to layer_2
197.
198.         outputs = self.layer_3(outputs) # apply the linear function to layer_3
199.         outputs = F.relu(outputs) # apply the activation function to layer_3
200.
201.         outputs = self.layer_4(outputs) # applying linear function to layer_4
202.         return outputs
203.
204.     def gradient(self):
205.         """
206.             gradient:
207.                 This will return the gradient we are using
208.             Returns:
209.                 () The gradient optimiser to use
210.         """
211.         return optim.Adam(self.parameters(), lr=self.learning_rate)
212.
213. class CustomeModel(nn.Module, Training):
214.     """
215.         CustomeModel:
216.             This would be used as way to allow the user to change layers within the neural network.
217.             This would acts as an antonmous process which defines models for testing the effect on different layers
218.             with different parameters
219.     """
220.
221.     def __init__(self, num_inputs, num_targets, learning_rate, layer_list,):
222.
223.         """
224.             init:
225.                 would initiliase the class
226.             Args:
227.                 layer_list (A list of dictionarys size 2): This would be the defined layers within the custome model.
228.                         The dictionary within layer list has to be in format: [2, 50, 100, 200, 400, 1]
229.                         where the first position denotes the first layer which has a 2 nodes and 50 output features.
230.                         The second position denotes the second layer which has a 50 nodes and 100 outputs and so on.
231.                 inputs: The number of inputs within for the model.
232.                 targets: The number of targets within for the model.
233.         """
234.         super(CustomeModel, self).__init__()
235.
236.         if len(layer_list) < 1:
237.             raise ValueError("They should be at least 1 layer")
238.
239.         # Throws error if length of first layer is not the same as len of inputs
240.         num_inputs_first_layer = layer_list[0]
241.         if (num_inputs_first_layer != num_inputs):
```

```
242.             raise ValueError("First layer has {} features, length of inputs is {}. This should be the
    same.".format(num_inputs_first_layer, num_inputs))
243.
244.         # Throws error if length of last layer is not the same as lengh of targets
245.         num_inputs_last_layer = layer_list[len(layer_list)-1]
246.         if (num_inputs_last_layer != num_targets):
247.             raise ValueError("Last layer has {} features, length of targets is {}. This should be the
    same.".format(num_inputs_last_layer, num_targets))
248.
249.         num_layers = len(layer_list)
250.
251.         self.linear = nn.ModuleList([nn.Linear(layer_list[n], layer_list[n+1]) for n in range(num_layers-1)])
252.
253.         self.num_inputs = num_inputs
254.         self.num_targets = num_targets
255.         self.learning_rate = learning_rate
256.
257.     def forward(self, inputs):
258.         inputs = inputs.reshape(inputs.shape[0], -1) # To ensure outputs is all within 1d vector
259.         for i in range(0, len(self.linear)):
260.             layer = self.linear[i]
261.             inputs = layer(inputs)
262.             if (i != len(self.linear)-1):
263.                 inputs = F.relu(inputs)
264.         return inputs
265.
266.     def gradient(self):
267.         """
268.             gradient:
269.                 This will return the gradient we are using
270.             Returns:
271.                 () The gradient optimiser to use
272.         """
273.         return optim.Adam(self.parameters(), lr=self.learning_rate)
```