A Mathematical Model for Quorum Sensing in Pseudomonas aeruginosa Using Machine Learning

Acknowledgements

I would like to express my gratitude towards my supervisor - Jongrae Kim for providing me with valuable guidance and resources that accelerated my progress in this project.

Table of Contents

Abstract

The harmful pathogen Pseudomonas aeruginosa has been researched for years, yet even with the increased understanding of its behaviour, it continues to burden healthcare services and infect vulnerable patients. A characteristic behaviour which contributes to the pathogen's virulence is known as quorum sensing. It allows the bacteria to remain undetected by host immune systems, and only cause harm once their colonies grow to overwhelmingly large populations. Quorum sensing is also known to be affected by a multitude of internal and external factors. It was therefore assumed that the variability of this behaviour could be captured by the modification of coefficients in a mathematical model that was proved to be successful in describing the quorum sensing behaviour. This project explored the use of machine learning methods in the forecasting of this behaviour. Specifically, neural networks were used in a multi-output regression problem to estimate the ordinary differential equation coefficients of the mathematical model.

The project consisted of building technical foundation with the utilized software through simpler examples. Artificial neural networks were successfully developed for estimating the gradient of a generic linear equation, and the natural frequency (w_n) and damping ratio (ξ) of a mass/spring/damper system – with the respective R^2 values being 0.995 and 0.979. The insight gained was used to enhance the Pseudomonas aeruginosa mathematical model. The result was an artificial neural network capable of predicting 4 out of 11 coefficients ranging $\pm$ 40% from their original values stated in literature, with a mean concentration error of 0.0637 units. The predictive capabilities of the enhanced model give it life-saving potential in clinical applications through the possible application of machine learning in diagnosis and treatment planning of infected patients.

Nomenclature

Abbreviations

AI	Artificial intelligence
ANN	Artificial Neural network
API	Application programming interface
IDE	Integrated development environment
LR	Learning rate
MAE	Mean absolute error
ML	Machine learning
MSE	Mean squared error
ODE	Ordinary differential equation
PA	Pseudomonas aeruginosa
QS	Quorum sensing
R^2	Coefficient of determination

Formula variables

A	3-oxo-C12-HSL autoinducer molecule
R	LasR transcriptional activator protein
R_0	Basal rate of LasR formation
A_0	Basal rate of 3-oxo-C12-HSL (A) formation
K_R	Michaelis constant, substrate concentration at which reaction rate is $\frac{V_R}{2}$
k_R	Natural degradation rate of LasR
V_R	Max rate achieved by system at saturated LasR conditions
V_A	Max rate achieved by system at saturated 3-oxo-C12-HSL conditions
K_A	Michaelis constant, substrate concentration at which reaction rate is $\frac{V_A}{2}$
ρ	Local density (volume fraction) of cell
δ	Diffusion conductance through cell membrane
k_E	Natural degradation rate of extracellular 3-oxo-C12-HSL
k_A	Natural degradation rate of intracellular 3-oxo-C12-HSL
m	Mass in a mass/spring/damper system
k	Stiffness in a mass/spring/system
c	Damping coefficient in a mass/spring/damper system
w_n	Natural frequency of a mass/spring/damper system
ξ	Damping ratio of a mass/spring/damper system

List of Figures and Tables

1. Introduction

Pseudomonas aeruginosa (PA) are a gram-negative bacteria - like the common bacteria E-coli (Campa, Bendinelli and Friedman, 1993). They are in the top 3 most common opportunistic pathogens (Stover et al., 2000). “Opportunistic” means that they take advantage of an individual’s weakened immune system to inflict harm. Thus, PA are mostly acquired by ill patients in hospitals. The bacteria were identified as the cause of 51,000 infections in the USA each year, with roughly 440 cases ending with death (Azam & Khan, 2018; Bassetti et al., 2018). Some of the maladies with which PA are frequently associated include traumatic burns, Cystic Fibrosis and ventilator-associated pneumonia. However, it is ultimately dependent on the patient and their condition as to how and where the pathogen will be most prominent in its infection.

The bacteria exhibit a characteristic known as QS (*quorum sensing).* QS allows bacteria to regulate their gene expression based on their population density. It can be thought of as a communication mechanism that allows bacteria to control a diverse array of physiological phenomena such as virulence (severity of illness caused), motility (ability to travel in fluid) and biofilm formation (Miller and Bassler, 2001; Papenfort and Bassler, 2016).

QS in PA is causative of disruption of the epithelial barrier and degradation of elastin, collagen and other matrix proteins (Lee and Zhang, 2014), amplifying the risk of lung-related illnesses. Hentzer (2003) stated that fighting the pathogen by attempting to mitigate the communication mechanism as opposed to using compounds that kill or inhibit growth is more beneficial. This is due to the lower chance of the pathogens developing resistance – rendering treatment easier in the future.

In Dockery’s (2001) research, a mathematical model demonstrating a simplified version of the QS mechanism through a system of ODEs (ordinary differential equations) was developed. The equations contain various constants inferring molecule concentrations and reaction rates. Their combination defines the rate of production of 2 molecules, from which the pathogen’s QS behaviour can be observed. It has been shown that QS can be affected by different factors such as oxygen concentration (Smith and Iglewski, 2003) and presence of medicines such as resperine used to combat biofilm formation (Parai et al., 2018). In this project it has therefore been assumed that the variability in the bacteria’s QS mechanism can be represented by the modification of the equation constants in Dockery’s, (2001) mathematical model.

Machine learning (ML) methods have been growing in popularity to solve complex problems, from programming of autonomous vehicles to optimized personalization of adverts for consumers. This research aims to utilize these novel techniques to enhance the mathematical model created by Dockery (2001), in being able to predict the constants of the ODE system, from user-generated molecule concentration data.

The problem can be defined as multi-output regression as the aim is to predict a set of continuous variables (ODE constants) from input data. Similar work in this domain is sparse, rendering the ML approach unique in its nature. The research shows immense potential as using the model to forecast the QS behaviour in a real-life situation could provide a medical application in diagnosis and treatment planning of infected patients.

1.1 Aims

The aim of the project is to utilize modern machine learning methods to enhance a mathematical model describing the QS behaviour in PA. The enhancement aims to account for the inherent variability of the communication mechanism due to probable external and internal factors.

1.2 Objectives

1. Build a technical foundation through practice with relevant examples
2. Understand and replicate the mathematical model presented in the literature
3. Produce datasets used for model training and testing
4. Enhance the mathematical model from literature using machine learning

1.3 Report Layout

The report structure is organised in a logical manner to provide the reader with the background, theory and methods necessary for understanding the work carried out in this project. The chapter breakdown is as follows:

- **Chapter 2 – Literature Review –** provides biological and technical background, as well as analysis into the research undertaken to tackle similar problems
- **Chapter 3 – Software Tools and Methods –** lists the software tools used in the project, as well as methods involved in the analysis of results
- **Chapter 4 – Practice Cases –** details the practice cases of predicting constants in a linear equation and mass/spring/damper system of ODEs.
- **Chapter 5 – P.aeruginosa model enhancement –** details the iterative approach taken into developing the enhanced PA mathematical model model.
- **Chapter 6 – Conclusions and Future Work –** gives a summary of the results and details of suggested future work in the domain

An appendix containing the supervisor meeting log and related code produced during this project is included at the end of the report.

2. Literature Review

2.1 Introduction

This chapter begins with providing a background into the biology of the bacteria and the associated quorum sensing mechanism. Dockery's (2001) mathematical model is then presented along with its ability to visualise the characteristic behaviour. Next, to build familiarity with machine learning, a brief introduction into the relevant theory is provided. The chapter concludes with analysis of examples where researchers utilized machine learning methods to tackle similar problems as the one outlined in this project.

2.2 Pseudomonas aeruginosa

Adding to its potential for harm is PA's highly resistant nature to a plethora of antibiotics of all classes, meaning that it is increasingly difficult to treat infections caused by this pathogen as the infections are becoming more severe and more likely to result in death (Bassetti et al., 2018). The result is an increased cost of 70% (16,265 vs 4,933 euros) when comparing the mean cost of admission per patient for resistant and non-resistant strains (Morales et al., 2012). This annunciates the urgent need for research in order for pharmaceutical companies and healthcare services to develop the treatment necessary to diminish these statistics.

An evolutionary survival mechanism of PA is the production of an extracellular polymer matrix termed "biofilm" that surrounds the bacteria. Biofilms are attributed to two thirds of PA infections and also increase the pathogen's resistance to antibiotics. The result can be a 10-1000 fold increase in resistance because of the biofilm. It renders PA to also be the most frequent colonizer of medical equipment such as catheters and joint replacements, by allowing them to endure harsh sterilisation procedures (Taylor, Yeung and Hancock, 2014). The contributing properties of the biofilm include the presence of antibiotic-degrading enzymes in the matrix, and creation of a protective environment where the bacteria can live in a low metabolic state, and a harsh anaerobic environment caused by antibiotics (Taylor, Yeung and Hancock, 2014).

2.3 Quorum Sensing

The characteristic was first discovered by Hastings and Nelson (1977), where the researchers attributed QS to "bioluminescence" (light production) in the marine bacteria - *Vibrio fischeri.* In this instance, the behaviour is observed as when the microorganisms are grown to a high enough population density, their colonies illuminate a blue-green light.

QS bacteria are able to perform cell-to-cell communication through the production of chemical signalling molecules called autoinducers. These increase as a function of the bacteria cell density, i.e. their population size (Miller and Bassler, 2001; Pérez-Velázquez, Gölgeli and García-Contreras, 2016). At low concentration levels, the autoinducer disperses into the surroundings without any effect. However, once there is a high enough concentration in the colony, and a threshold is reached, what follows is a sudden change in gene expression – accompanied by an exponential increase in the production of the autoinducer molecule. This allows the bacteria to alter their gene expression as a whole population rather than individually, permitting the output of the transcribed genes to be significantly exerted. The fundamentals of the mechanism are the same for different microorganisms. It is the biochemical details that differ from one type of bacteria to another (Dockery, 2001; Pérez-Velázquez, Gölgeli and García-Contreras, 2016).

In healthy individuals, secretion of virulent toxins by pathogens instigates a generally efficient immune response from the host – resulting in eradication of the pathogens. However, by regulating the release of these toxins to only be detectable once the population of the colony is regarded as large enough, it is ensured that the immune system defences are overwhelmed. The result is an increased likelihood of survival and infection from the pathogen (de Kievit and Iglewski, 2000; Dockery, 2001).

2.4 Quorum Sensing in Pseudomonas aeruginosa

The QS mechanism in PA is hierarchical and interconnected, consisting of at least 4 individual systems. These affect a multitude of virulence factors including, biofilm structure and dynamics, cytotoxicity and antibiotic resistance (Lee and Zhang, 2014). However, within this research project, a simpler model, consisting of only two systems was considered.

The two systems of interest are the *las* and *rhl*. In *las*, the autoinducer molecule - 3-oxo-C12-HSL (A) - is produced by a synthase enzyme - LasI. It binds to a transcriptional activator protein - LasR (R) - to form a complex, which goes onto bind to target DNA sequences and stimulate their transcription. The outcome is the production of LasB and LasA enzymes which deteriorate human lung tissue and are highly dangerous in patients suffering from lung infections (Dockery, 2001). The autoinducer/transcriptional protein complex regulates the expressions of several genes as well as promotes the activity of *lasI* - a gene coding for the synthesis of the autoinducer, which inherently results in a positive feedback loop.

The second system – *rhl* system – is dependent on the *las* system as the generated 3-oxo-C12-HSL (A) / LasR (R) protein dimer stimulates the transcription of the transcriptional protein of the *rhl* system. The autoinducer molecule and transcriptional protein for this system are labelled “C4-HSL” and “RhlR” respectively (Dockery, 2001; Smith and Iglewski, 2003). The product of the *rhl* system is a “rhamnolipid” that degrades the function of human respiratory epithelium (Dockery, 2001). *Figure* 2.1 illustrates a simplified schematic of the system, omitting certain components as they are not used in the mathematical model outlined in Section 2.5.

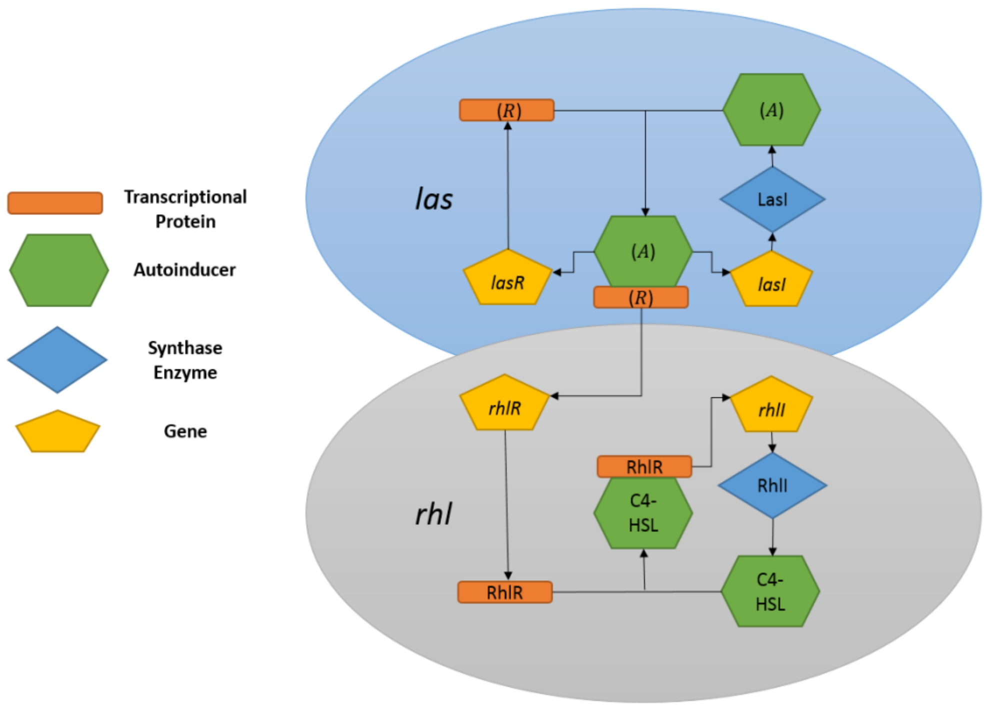


Figure 2.1. Simplified schematic diagram of the las and rhl systems in Pseudomonas aeruginosa

2.5 Mathematical model of quorum sensing in P. aeruginosa

Dockery (2001) generated a mathematical model of the QS behaviour. The model is described as eight-dimensional ODE system - defined by equations (2.1), (2.2) and (2.3). These incorporate several assumptions such as being based solely on the *las* system (represented by the blue oval in *Figure* 2.1). However, the cell QS ‘switch’ point – where the autoinducer concentration shows an explosive increase (*Figure* 2.2) – can still be interpreted from it, and thus, it was used as the foundation for the machine learning model developed on top of it.

$$\frac{dR}{dt} = \frac{V_R RA}{K_R + RA} - k_R + R_0 \tag{2.1}$$

where: R is the concentration of LasR - the transcriptional activator protein; A is the concentration of 3-oxo-C12-HSL - the autoinducer molecule; V_R is the a component of the assumed Michaelis-Menten type reaction representing the maximum rate of reaction in saturated substrate conditions for R; K_R is the Michaelis constant and represents the substrate concentration at which the reaction rate is half of V_R; k_R is the natural degradation rate of LasR; and R_0 is the basal rate of formation of R

$$\frac{dA}{dt} = \frac{V_A RA}{K_A + RA} + A_0 - d(\rho)A \tag{2.2}$$

where: V_A is the a component of the assumed Michaelis-Menten type reaction representing the maximum rate of reaction in saturated substrate conditions for A; K_A is the Michaelis constant and represents the substrate concentration at which the reaction rate is half of V_A; A_0 is the basal rate of formation of A; and $d(\rho)$ is the decay rate of A defined by equation (2.3).

$$d(\rho) = k_A + \frac{\delta}{\rho}\left(\frac{k_E(1-\rho)}{\delta + k_E(1-\rho)}\right) \tag{2.3}$$

where: k_A is the natural degradation rate of intracellular A; δ is the diffusion conductance through the cell membrane; ρ is the local density (volume fraction) of cells; and k_E is the natural degradation rate of extracellular A.

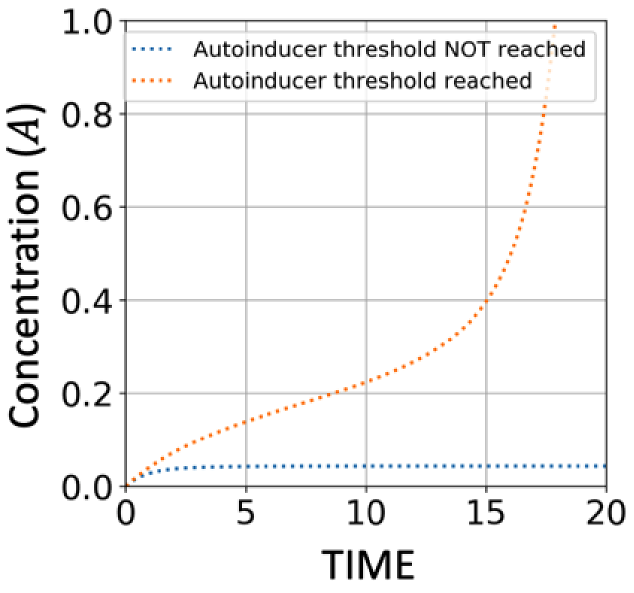


Figure 2.2. Autoinducer concentration (A) plots over time for ODE coefficient configurations showing QS switch point

Using equations (2.1), (2.2) and (2.3) it is possible to show the QS switching behaviour as when ρ is high enough, there is a surge in concentration of the autoinducer A upon the simultaneous integration of (2.1) and (2.2) - illustrated in *Figure* 2.2. Dockery (2001) attributed this to the decreased elimination of A in the extracellular space.

The system is indeed simplified as the study of particular chemical reaction timescales discounted the need for incorporating components from mRNA molecules. This is because their lifespan is significantly shorter compared to the respective synthase enzymes they are responsible for transcribing (shown as blue diamonds in *Figure* 2.1) (Dockery, 2001), (Pérez-Velázquez, Gölgeli and García-Contreras, 2016). Furthermore, Dockery, (2001) extended the mathematical model further from the one presented by equations (2.1), (2.2) and (2.3), rendering it more realistic through the inclusion of a spatial variable. However, the resulting increased complexity was not possible to incorporate in this project given the permitted time scale.

2.6 Machine Learning

ML is an application of AI (artificial intelligence). According to the Merriam-Webster dictionary, AI is defined as "a branch of computer science dealing with the simulation of intelligent behaviour in computers" (Merriam-Webster, 2019). ML involves the inferring of patterns and construction of rules from data by passing it through user-generated models that employ techniques from a combination of computer science and statistics. Deep learning is yet a further subset of ML in which data is ran through a layered analysis of filters before obtaining the final output from the model. *Figure* 2.3 represents the groupings of the above concepts. Deep learning models are almost always coined as "Artificial Neural Networks" (ANNs), although the association of the neural network with the brain is highly misleading as the brain is still far from being comprehended (Chollet, 2017).

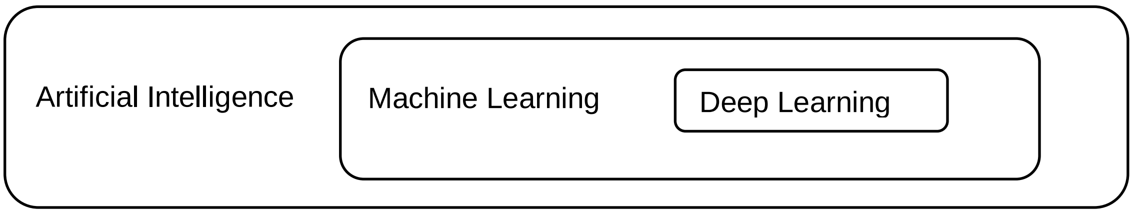


Figure 2.3. A grouped representation of Artificial Intelligence, Machine Learning and Deep Learning

There exist several forms of deep learning which include:

- supervised learning; where the input data to the model is labelled with target values of corresponding outputs. This is also the form of deep learning used in this project.
- unsupervised learning; where the data is unlabelled, and the model develops relationships about the different data and groups it together based on the identified patterns
- reinforcement learning; where a model takes actions in an environment on the basis of trial-and-error to maximize a cumulative reward (Chollet, 2017)

ANNs based on supervised learning have been successfully employed in image categorization tasks (outputs being discrete variables), such as presence of objects such as cars, aeroplanes, cats or dogs in an image (Guillaumin, Verbeek and Schmid, 2010), (Kang, 2015). They have also been used in prediction continuous variables as a regression model. Examples include the prediction of stock performance (Nicholas Refenes, Zapranis and Francis, 1994) or forecasting electricity consumption using time-series data (Kaytez et al., 2015).

The layered analysis in ANNs comes from the interconnection between "neurons" present in the layers of the network. A simple schematic of a typical ANN is illustrated in *Figure* 2.4. The network begins with an input layer, into which data is fed, proceeded by hidden layers and a final output layer. The neurons in the layers are connected by weights and biases, which have the function of transforming the information inferred from the input data. In order for an ANN to perform well, it has to be trained by a means of a training algorithm, where the weights and biases between the interconnected neurons are optimized to minimize an error function between a model's predicted outputs and the true values (Kalogirou, 2000; Sözen and Arcaklioglu, 2007) .

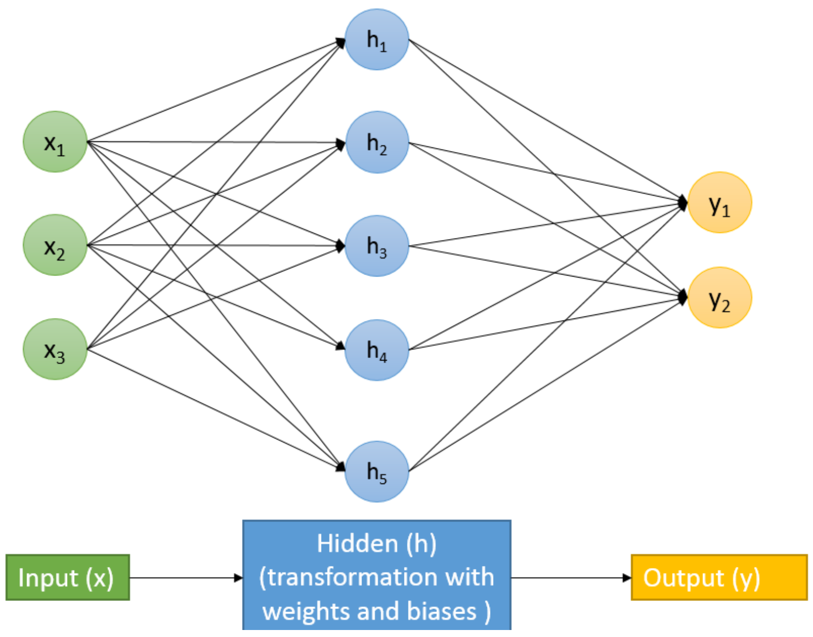


Figure 2.4. Simple Schematic of ANN with one hidden layer

The simplest and most standard architecture of an ANN is the feedforward neural network. In this instance, the information inferred from data in terms of weights and biases is relayed in one direction from the input to the output nodes, without any cycles or loops (Zell, 1994). The error between predicted and desired outputs is minimized by calculating derivatives of the loss function.

A concept called back propagation is most commonly implemented in standard ANNs, where the calculated derivatives are decomposed for all the layers in the network, and the weights and biases of nodes are updated via the application of an optimizing algorithm (for example stochastic gradient descent). The process is iterated until the error converges to a stable value, wherein further reductions can only be achieved by tuning the model's hyperparameters or modifying the input data (Chollet, 2017).

Model hyperparameters constitute of the model architecture, such as the number of hidden nodes or layers in an ANN. Other optimisable hyperparameters are the optimization algorithm parameters employed in reducing the loss function such as the learning rate (LR). This is the rate at which node weights are adjusted with respect to the loss gradient. Hyperparameters need to be tuned during model development to ensure that the model is fully optimized for its function (Chollet, 2017).

2.7 Multi-output Regression

The problem presented in this research project can be represented as a MO (multi-output) regression problem. This is due to the aim of predicting several constants in the mathematical model (outputs) from the time-series molecule concentration data (inputs). In recent years, a multitude of techniques have been tested to tackle this problem. Borchani et al. (2015) explain how using a single model is more computationally efficient than constructing multiple single-target regression models. However, this comes at a cost of an increased challenge of the model build. The researchers attributed this claim to the fact that MO regression models not only consider relationships between inputs and the targets, but also between the targets themselves. In summary the two approaches are identified as:

1) Problem transformation

   The approach assumes that a solution can be achieved by breaking down a multiple target model into individual models that predict each target separately.
2) Algorithm transformation

   This approach aims to use a single model that captures all input/output and output/output inter-dependencies.

Clustering, kernel regression, ANNs and tree & graph structures are ML methods that have been used as forms of algorithm transformations, indicating the prowess ML techniques exhibit in solving these complex problems (Borchani et al, 2015).

Antanasijević et al., (2018) showed the promise of using ANNs in predicting air-pollutant emissions using emission data from 26 European countries based on socioeconomic inputs such as GDP. The mutual correlation between the outputs was low ($r < 0.55$), which augmented the complexity of the task for the ANN presented in the study. In contrast, when the input data is highly correlated, it introduces confusion into the ANN learning process. Examples from the research include job density and population density which showed perfect correlation (r = 0.99). Moreover, the input data consisted of only 6 different features – a low number compared to the autoinducer concentration trajectory data used in this project.

The type of data used within this project also differs in terms of its nature. The input data is in a time-series format, as opposed to being feature-based, and thus, checking the correlation between time steps would not prove insightful. Moreover, the data is user-generated, meaning that the outputs are randomly generated, and thus again, assessing correlation would not provide any value. Nevertheless, some statistical analysis should always be undertaken to evaluate the data's representation.

A variation of a feed forward neural network using backpropagation (explained in section 2.6) called a general regression neural network – a specific type of radial basis function network – was used by Antanasijević et al., (2018). The claims were that these networks have faster training times and perform better in noisy environments and so could serve as a possible approach in this project.

Punjani and Abbeel, (2015) make use of an ANN in regression prediction of highly-dimensional helicopter dynamics. The researchers evaluate the performance of their deep learning approach by assessing the error against baseline models. It is critical to have a reference point when evaluating any model, as frequently it is difficult to even beat trivial models that continuously predict the mean of a target dataset. Additionally, the increased performance measurement serves as a quantitative metric that gives a more intuitive assessment into the capabilities of the model.

2.8 Summary

The preceding literature review established a solid background of the relevant biology and provided motivation for tackling the problems caused by PA infections. The associated mathematical model was detailed alongside its capability to successfully replicate the QS mechanism in PA, making it viable for use in this project.

ML theory was explained to grant a general idea of how the tools employed in this project work, as well as other instances where they have been used. The applicable MO regression problem was described and examples of where ML has been used in tackling it have been analysed. The resulting analysis invokes that there is a need for investigation of the data before the construction of a model, as well as the need for a reference point from which the effects of any data/model modifications can be measured against. These considerations were taken into account throughout this project and applied appropriately.

3. Software Tools and Methods

3.1 Tools

3.1.1 Python

All coding work carried out in this research was done using the open source programming language; Python v 3.6.1, in an IDE (integrated development environment) – PyCharm 2019.1 (Community Edition)

3.1.2 SciPy (Python library)

SciPy is an open source Python library for mathematics, science and engineering. Within this project, it was used in the integration of ODEs, of which the outputs served as datasets into the ANNs. SciPy version 1.1.0 was used in this project.

3.1.3 Keras (Python library)

Keras is an open source Python library and an API (application programming interface) for application of deep learning in Python. Keras version 2.2.4 was used in this project.

3.1.4 Talos (Python library)

Talos is an open source Python library that serves as a hyperparameter optimization tool that integrates with Keras. Talos version 0.4.3 was used in this project.

3.2 Methods

3.2.1 Data generation

ScipPy was used to integrate ODEs of which the solutions included the input variables over a time vector of a predefined length. The ODE coefficients (target labels), from which the input data was generated were also saved and separately stored. The generated data is unitless and presented as such within the figures of this report. This is due to the arbitrary nature of the ODE coefficients it was generated from.

3.2.2 Training, test and validation dataset splitting

The ML convention is to split the input dataset into 3 groups – training, testing, and validation sets (Chollet, 2017). A schematic of the method is pictured in *Figure 3.1*.

The training dataset is used to train the model. The model infers all relationships and transformation weights and biases from this data after being trained for a specified number of iterations.

The validation set is used to give an unbiased assessment into the performance of the model during training – being instantly evaluated on data that it has not been exposed to. The main purpose of the validation set for fine-tuning the model hyperparameters, and to ensure that the model is performing to an acceptable standard on unseen data - not being biased towards the data used during training and leading to overfitting.

The test set provides a final evaluation of the model performance. This due to the possibility of the model becoming biased towards the validation data during the adjustment of hyperparameters.

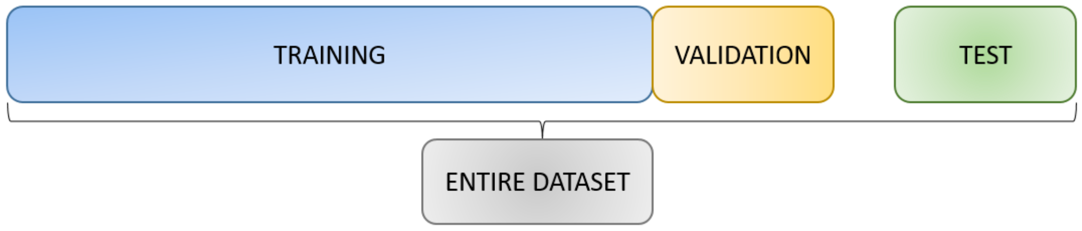


Figure 3.1. Visualisation of dataset splting

3.2.3 Data standardisation

Data standardisation is a form of data processing that ensures all the features of a dataset have a common scale. This is necessary as features containing a larger scale impose a greater influence on the result compared to lower-scaled ones, providing a misleading insight into their importance (Chollet, 2017). The conventional method of standardisation is expressed in equation (3.1). This form of standardisation centres the feature about a mean of 0 and a standard deviation of 1. Standardisation is carried out with respect to the training data. The calculated means and standard deviations are then used to transform the testing data.

$$x' = \frac{x_i - \bar{x}}{\sigma} \tag{3.1}$$

where: x' is the standardised data point, x_i is the intial data point, $\bar{x}$ is the mean of all the data points for that feature, and σ is the standard deviation of those features.

3.2.4 Hyperparameter optimisation

The Python library; Talos was used to optimise the Keras ANN hyperparameters. The hyperparameters studied in this project and their definitions are as follows:

- Epochs – the number of iterations in which the entire dataset is passed forward and backward through the ANN once.
- Batch size – the number of training samples in a batch.
- Hidden nodes – the number of hidden nodes in a given layer that contain weight and bias transformation functions.
- Learning rate – the rate at which the model is altered in response to the error every time the model weights and biases are amended.

- Weight regularizer – a parameter that limits the weights of nodes, encouraging the development of a simpler model less prone to overfitting.

3.2.5 K-fold cross validation

This method comprises splitting the training and validation data into K subsets. A model is trained K times, and for each instance, the training and validation data are adequately split to provide K model assessments. This allows every sample in the dataset to be used in training and in validation. After the process has ended, the overall performance metrics of the models are combined to calculate an average – providing a more detailed evaluation of the model. The process is illustrated in *Figure 3.2*

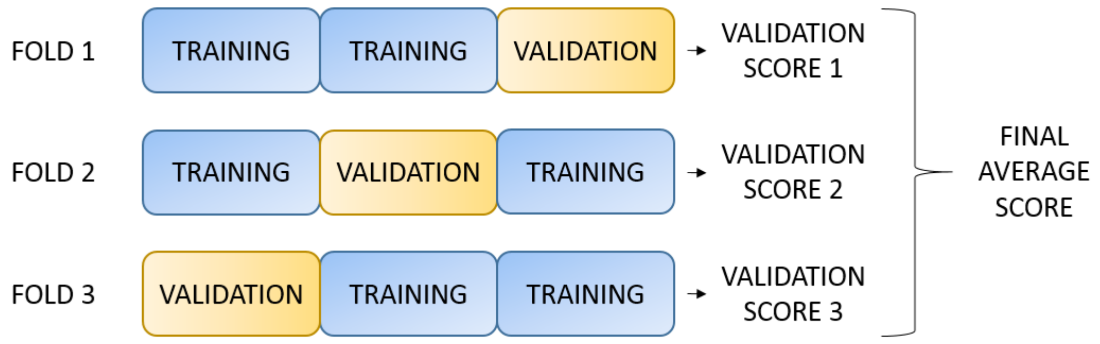


Figure 3.2. K-fold cross validation schematic (K = 3)

3.2.6 Loss functions and metrics

The primary loss function used in model optimization was MSE (mean squared error) defined in (3.2), giving the average squared error between predicted and true values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2 \qquad (3.2)$$

where: n is the total number of samples, y_i is the prediction and $\widehat{y_i}$ is the true value

In addition to monitoring loss, the metrics R^2 (coefficient of determination) - equation (3.3), and MAE (mean absolute error) - equation (3.4) were used as alternatives for measuring model performance. These are standard metrics used in regression models (Chollet, 2017), (Lowe, Emsley and Harding, 2006).

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \qquad (3.3)$$

where: SS_{res} is the sum of the squared residuals from predicted and true values and SS_{tot} is the total sum of squared residuals from calculated means and true values.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}| \qquad (3.4)$$

4. Practice Cases

4.1 Linear equation

The first practice case involved prediction of a single gradient constant in the generic linear equation (4.1). This simple task served to develop familiarity with the software.

$$\frac{dy}{dt} = m \tag{4.1}$$

where: y is the dependent variable, m is the gradient and t is the independent time variable

4.1.1 Dataset information

The user-generated input data involved the integral of equation (4.1) over a vector of 10 time points from t = 0 to t = 100. The initial conditions were y = 0 when t = 0. The output labels consisted of 2000 m values in the range $-1000 \leq m \leq 1000$. The data was shuffled before being used in model building to ensure a stochastic mixture of the data during training. Next the data was split into training (56.25%), validation (18.75%) and testing (25%) datasets. *Figure 4.1* illustrates a schematic of the ANN for one sample.

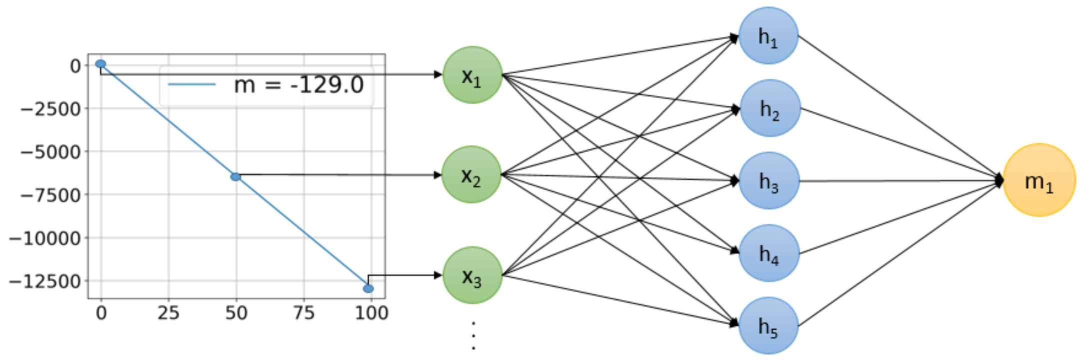


Figure 4.1. Schematic of an ANN predicting a single gradient value output from an input sample

4.1.2 Results

Evaluation of the model against never-seen before test data gave a R^2 value of 0.995 and a MAE of 4.56. Table 4.1 shows the ANN performance when deployed to predict test data with m values of -1500 and 1500. In summary, this exercise achieved its goal of establishing familiarity with the data generation procedure and Keras workflow.

Table 4.1. ANN performance on gradient value prediction with new data

True value	Predicted value	Error
1500	1497.72	-2.28
-1500	-1524.48	-24.48

4.2 Mass/spring/damper system

The second practice test case was based on the standard engineering mass spring damper system comprising of a second order ODE – defined in equation (4.2).

$$\ddot{x} = -\frac{k}{m}x - \frac{c}{m}\dot{x} \tag{4.2}$$

where: $\ddot{x}$ is the acceleration, $\dot{x}$ is the velocity, x is the displacement, k is the stiffness, m is the mass and c is the damping coefficient.

The initial objective was to predict the 3 constants; m, k and c. However, the problem was later simplified into the prediction of 2 variables describing the behaviour of the system. Namely these were the natural frequency w_n – defined in equation (4.3), and the damping ratio ξ – defined in equation (4.4).

$$w_n = \sqrt{\frac{k}{m}} \tag{4.3}$$

$$\xi = \frac{c}{2\sqrt{mk}} \tag{4.4}$$

4.2.1 Initial Dataset

The input data used in this case was the displacement (x) output of the integral of equation (4.2), culminating to a total of 1,000 samples, whilst the velocity output ($\dot{x}$) was dismissed. Further information on the dataset included:

- Sampling frequency set at 1Hz, sampling for a total of 100 units, creating a vector of total length t = 100.
- The initial conditions of displacement; x_0 = 2 and velocity; $\dot{x}_0$ = 3 units.
- The output targets set to ranges of; $1 \leq m \leq 2{,}000$, $1 \leq k \leq 10{,}000$ and $1 \leq c \leq 3{,}500$.
- The generated dataset being split into subsequent training (64%), validation (16%) and testing (20%) datasets.

The data was grouped based on the calculated value of ξ to provide a representation into the spread of the different damping behaviours - shown in *Figure 4.2*.

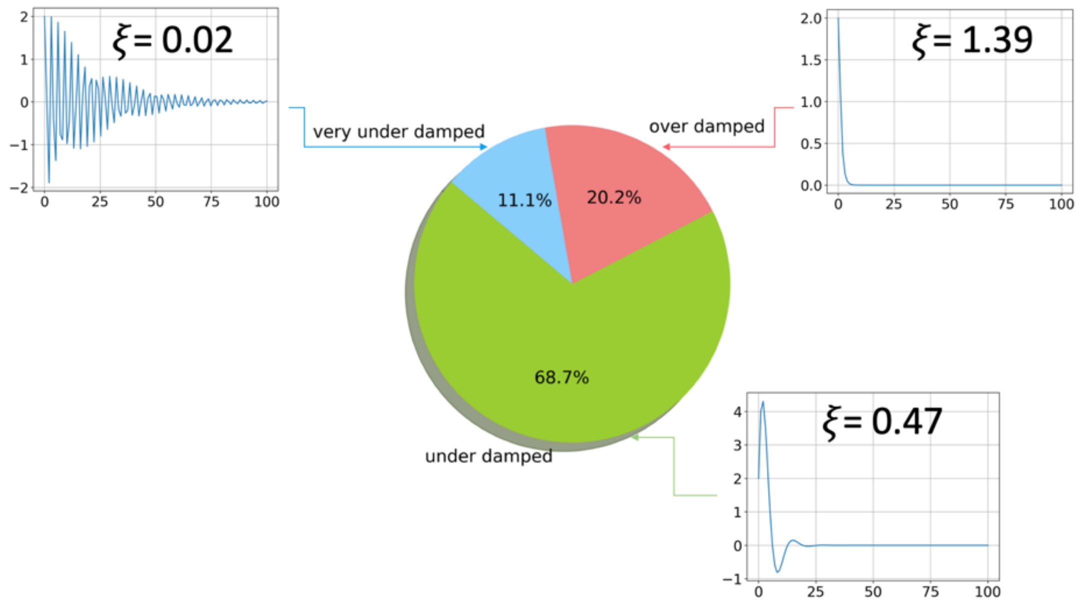


Figure 4.2. Pie chart displaying the distribution of the different damping behaviours with exemplar displacement trajectories. Very under damped: $\xi \leq 0.1$ under damped: $0.1 < \xi < 1$, over damped: $1 \geq \xi$

An initial model was created with manually configured hyperparameters. The loss and metric convergence are shown in *Figure 4.3*, and the spilt dataset results in *Figure 4.4.*

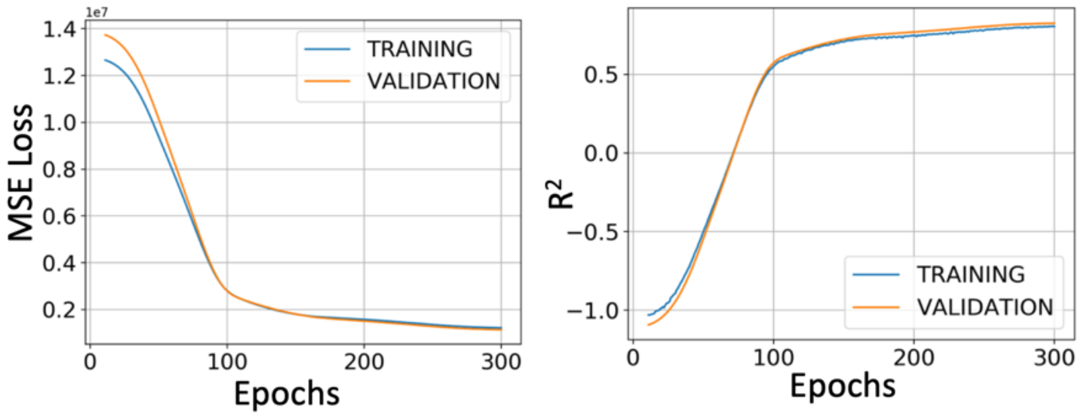


Figure 4.3. MSE and R^2 convergence for initial, manually configured ANN

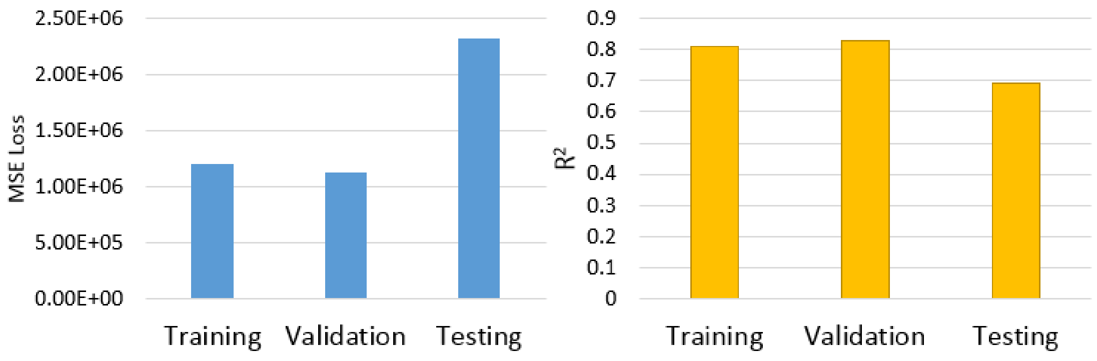


Figure 4.4. Training, validation and test dataset MSE and R^2 comparison for initial, manually configured ANN

Figure 4.4 suggests that the training and validation dataset scores were in good agreement with each other (a -7% and +2% difference for MSE and R^2 respectively). On the other hand, the testing scores were significantly worse (a +92% and -14% difference for MSE and R^2 compared to the training dataset). This is due to a concept known as overfitting and is defined as building a model that infers excessive information from the training data, making it biased towards it (Chollet, 2017). In this case the model hyperparameters were selected such that they complemented the validation data, explaining the model's high performance with respect to that dataset.

The model was tested on a random combination of newly generated target values. As the targets m, k and c do not uniquely determine the displacement trajectory, it would suffice if the model was capable of predicting the trajectory with a different combination of the ODE coefficients. A comparison plot is shown in *Figure 4.5.*

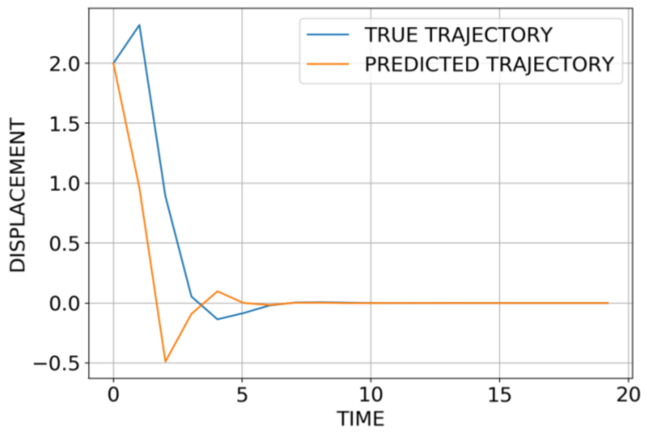


Figure 4.5. Comparison of initial predicted displacement trajectory and the true target. True coefficients: m = 1500, k = 2200 and c = 2500. Predicted coefficients: m = 1325, k = 5011 and c = 2314

The predicted trajectory is correct in terms of reaching a steady state at the same time t ~ 6 units. However, it was speculated that an even better prediction can be achieved with respect to the predicted oscillatory trajectory.

4.2.2 <u>Dataset modification</u>

Certain modifications were incorporated when regenerating the datasets to provide the model with more representative and better-quality data. The modifications and reasoning included:

- Changing target outputs from m, k and c to w_n and ξ – as w_n and ξ are variables which explicitly govern the oscillatory behaviour of the system.
- Enforcing a more representative distribution of the 3 damping behaviours (*Figure 4.6*) – to give a more balanced spread of input data into the ANN

- Increasing the sampling frequency to 2Hz and shortening the total time period to 50 units – as in most cases the trajectories reached a steady state after t = 50 units.
- Reducing the ODE coefficient range for all coefficients to be in the range of $1 \leq coeff. \leq 1000$ – to generate an equal range of the ODE coefficients.

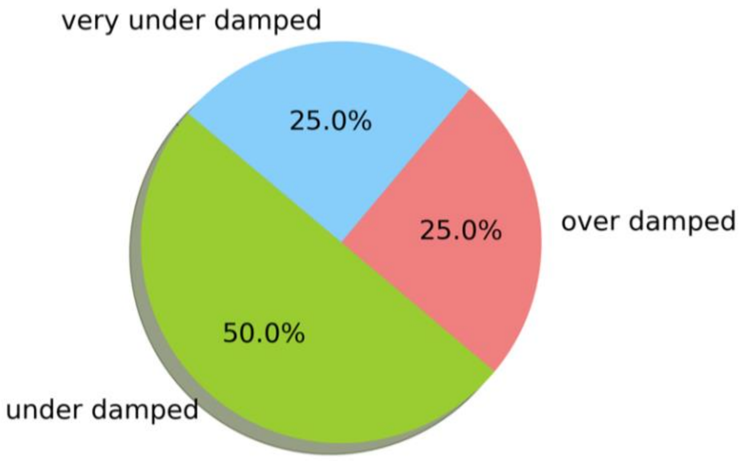


Figure 4.6. Modified distribution of the 3 damping behaviours

K-fold cross-validation was carried out with K = 4 folds to gain insight into the variability of the predictions made by the ANN. The revelation was that the differing model behaviour between the folds with respect to the MSE loss and R^2 metric – illustrated in *Figure 4.7*. This observation suggested that there existed significant variation between the subsets of the training and validation datasets for each fold in the cross-validation, resulting in the model performing relatively well in some cases (K = 2 and 3), and poorly in others (K =1). This resulted in a skewed mean validation R^2 score of ~ 0.52. Such variation in model performance is unacceptable, and thus had to be rectified.

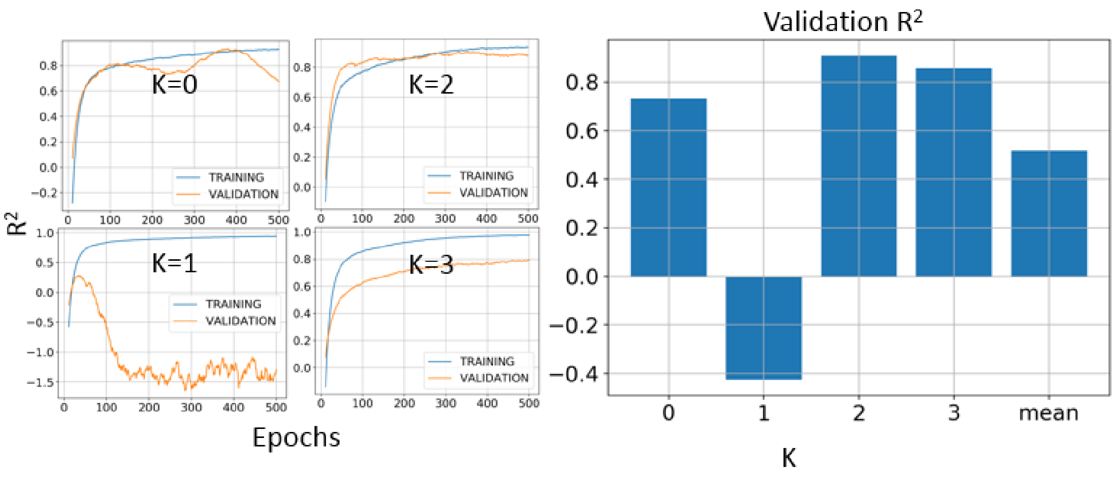
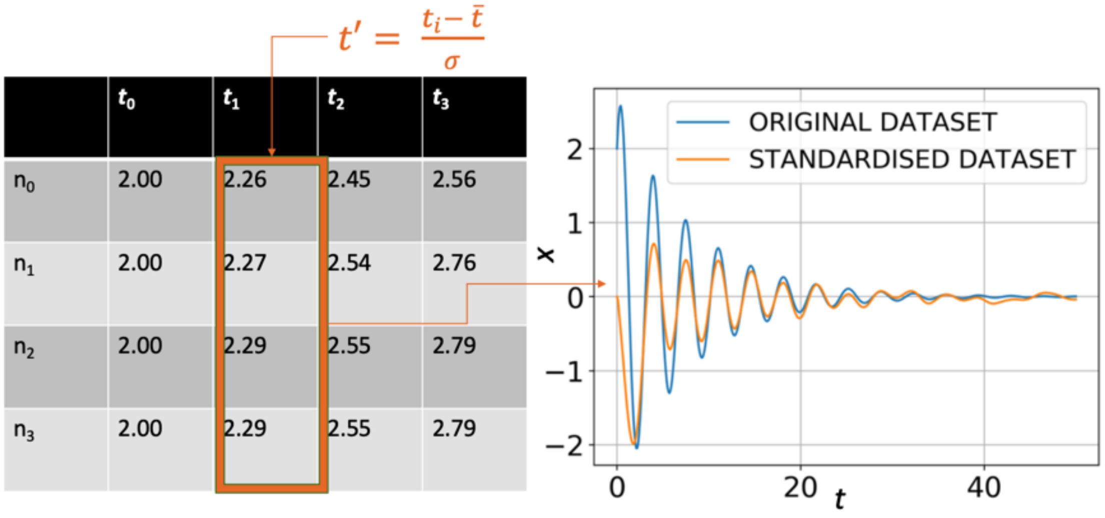


Figure 4.7. Incongruous R^2 metric conversion history with calculated mean for all folds.

4.2.3 Standardisation

Data standardisation was applied to investigate the impact on the model performance. *Figure 4.8* showcases the transformation applied to the data after standardisation.



$$t' = \frac{t_i - \bar{t}}{\sigma}$$

	t_0	t_1	t_2	t_3
n_0	2.00	2.26	2.45	2.56
n_1	2.00	2.27	2.54	2.76
n_2	2.00	2.29	2.55	2.79
n_3	2.00	2.29	2.55	2.79

Figure 4.8. Standardisation procedure and sample trajectory comparison between non-standardised and standardised data

Although the variation in between the folds was still present, on average the model was performing better with respect to the R^2 metric - shown in *Figure 4.9*. The updated average mean from all the folds was ~ 0.79 – a 52% increase from the average result presented in *Figure 4.7* The conclusion was that the standardisation should be employed from then on.

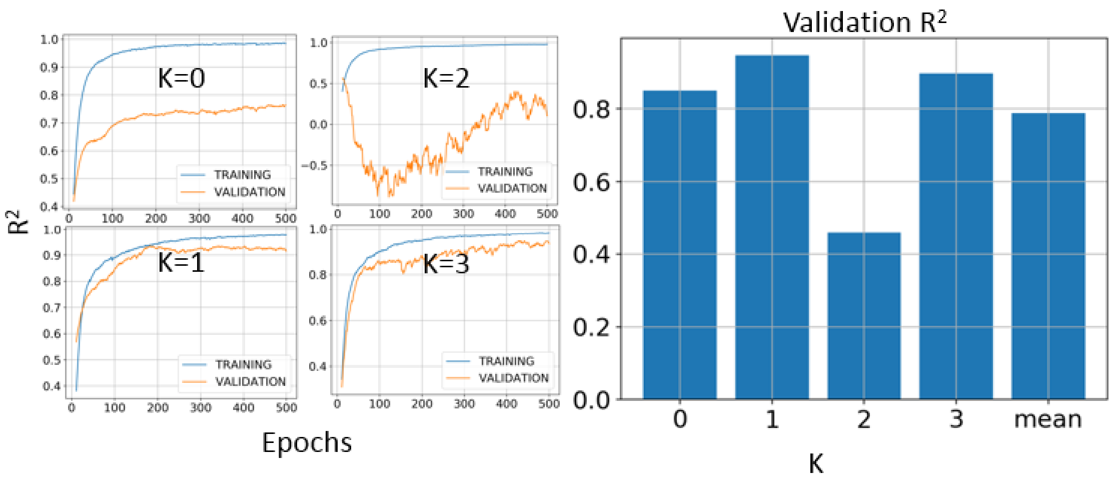


Figure 4.9. R^2 metric conversion history with calculated mean for all folds after dataset standardisation

4.2.4 Further Dataset Modification

In order to tackle the incoherent behaviour with respect to validation R^2 metrics, further modifications were applied to the dataset with the following reasoning:

- Reducing the ODE coefficient range for all coefficients to be in the range of $0.1 \leq m/k/c \leq 10$ – to further constrain the range of the ODE coefficients.
- Increasing the number of samples in the dataset from 1,000 to 10,000 – to provide the ANN with more data from which to infer relationships.
- Removing the constraint on the distribution of damping behaviours from *Figure 4.6* as it was thought to jeopardize the data rather than improve the distribution of the different damping behaviours. The result being an updated distribution, shown in *Figure 4.10*
- A condition was set such that $\xi < 10$ to oust any outlier samples which would influence confusion from the dataset.

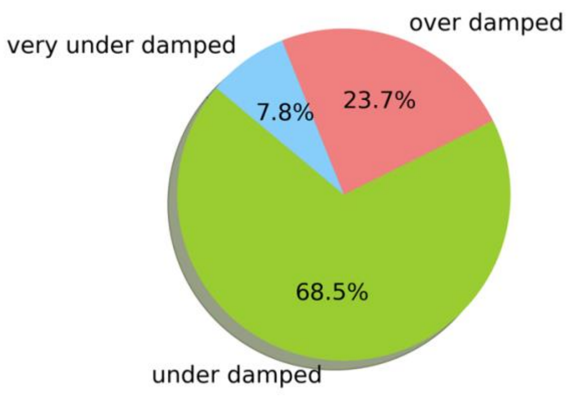


Figure 4.10. Updated distribution of the 3 damping behaviours

A model with the same ANN hyperparameters was tested again through K-fold cross-validation with K = 4.

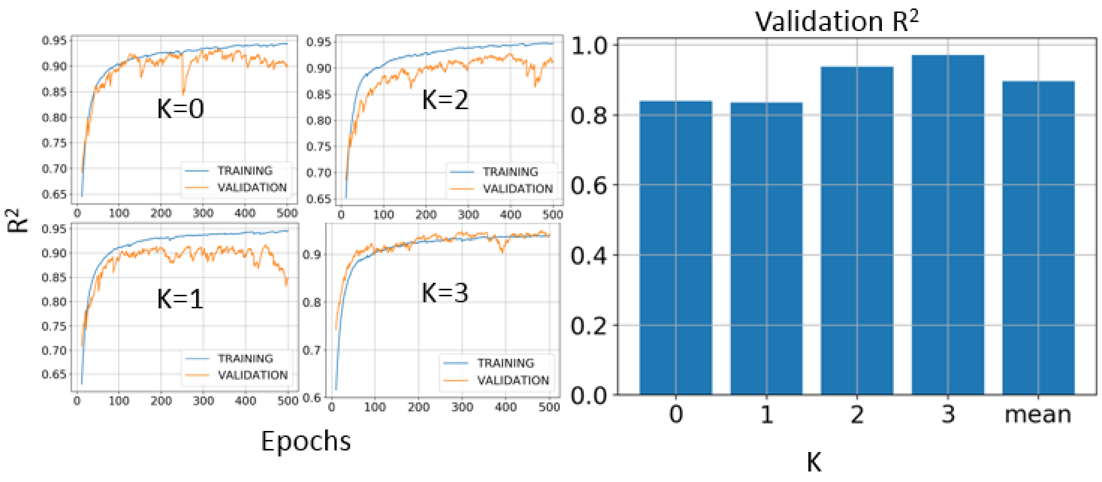


Figure 4.11. R^2 metric history after further dataset modifications for all folds with calculated mean.

Figure 4.11 shows significant improvement in model coherence between the different folds. This meant the dataset was of good enough quality to move on to the hyperparameter optimisation of the ANN. The mean MAE metric for validation data from all was ~ 0.105, which served as a reference point for hyperparameter optimisation detailed in Section 4.2.5.

4.2.5 Hyperparameter optimisation with Talos

The primary step in hyperparameter tuning was into the number of layers in the ANN. *Figure 4.12* shows KDE (kernel density estimation) plots to visualise the distribution of the MAE scores. The additional parameters that were investigated using Talos were the batch size, number of hidden nodes in each layer and number of epochs.

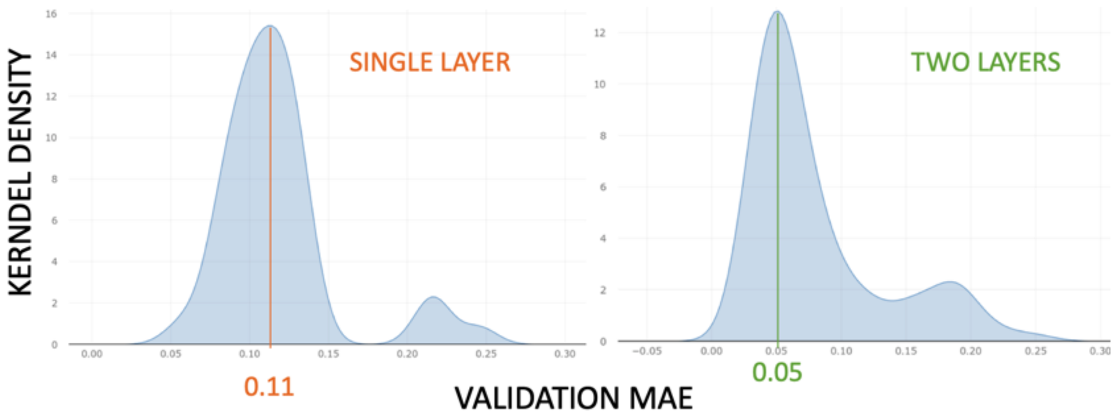


Figure 4.12. KDE plot comparing the distributions of validation MAE metrics for optimisation cycles considering only a single and a 2-layer configuration.

Figure 4.12 showcases the advantage of using a two-layer configuration in an ANN. The right-hand plot, corresponding to the two-layer model architecture KDE plot shows a distribution centred on a lower MAE value (0.05) in comparison to the left-hand single-layer configuration (0.11). Moreover, the lowest validation MAE score achieved in the two-layer optimisation was 0.028 - a significantly lower value in comparison to the baseline of 0.105. This was due to the ANN's ability to form more intricate connections from the input data – in the form of increased combinations of various weights and biases. *Figure 4.13* displays a correlation heatmap of the investigated hyperparameters alongside a regression plot of the number of hidden nodes in the first layer and the resultant MAE for the two-layer optimisation cycle.

The most impactful parameter was the number of hidden neurons in the first layer of the ANN. The regression plot in *Figure 4.13* indicates that the configurations resulting in the lowest validation MAE were the layers consisting of 32 and 64 hidden nodes. The other hyperparameters of batch size, epochs and second neuron hidden nodes did not exert significant impact, and thus, to render the model build quicker, lower epochs and higher batch sizes were preferred.

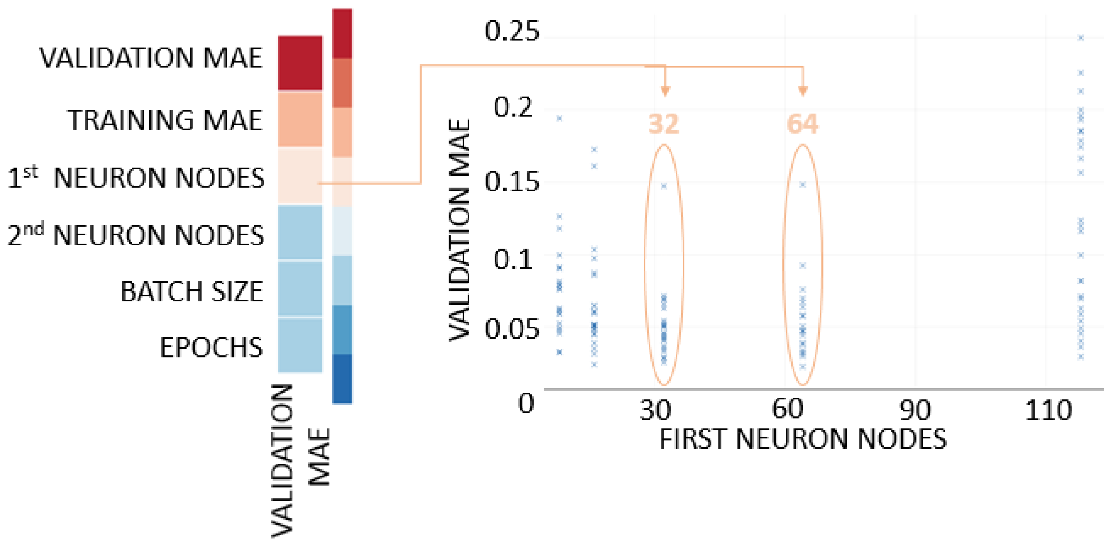


Figure 4.13. Talos output correlation heatmap with corresponding regression plot investigating the number of hidden nodes in the first ANN layer for the two-layer optimisation study

Next, the learning rate of the model was studied whilst imposing a tighter range for the number of hidden nodes in the first layer. *Figure 4.14* displays the result of the Talos optimisation cycle with the learning rate incorporated as the changing parameter. The conclusion was that the model favoured a mid-range learning rate (between 0.5 and 1 of the normalised default value), and a higher number of hidden nodes in the second layer. It is noteworthy that there was no significant impact from the difference in the number of epochs. Additionally, the lowest validation MAE obtained in this study was 0.034 – slightly higher than the previous cycle (0.028). The model that achieved this score had the default learning rate of 1 – suggesting that the variation introduced by the varying learning rate was disadvantageous.

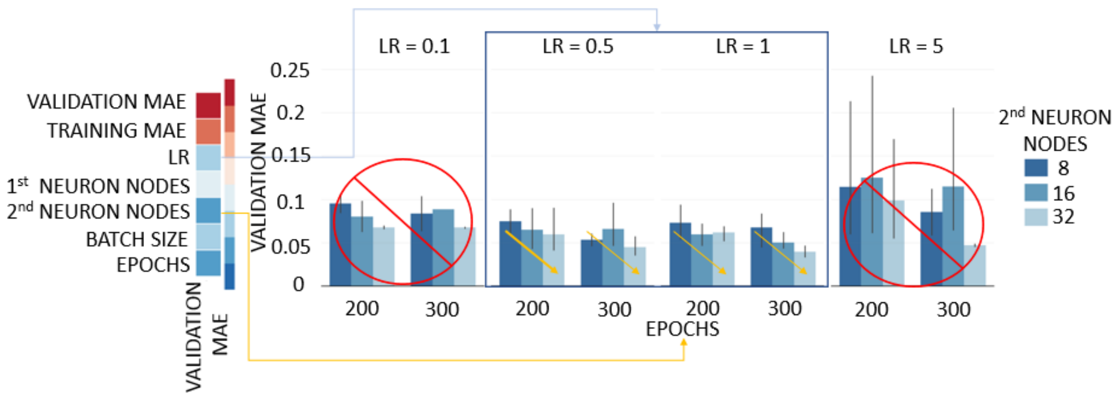


Figure 4.14. Talos correlation heatmap for the learning rate investigation. Corresponding bar charts indicate that the mid-range of the learning rate (blue arrow) and higher numbers of hidden nodes in the second layer (yellow arrows) produce favourable results in terms of a lower MAE.

The next study incorporated the investigation of weight regularizers in both layers.

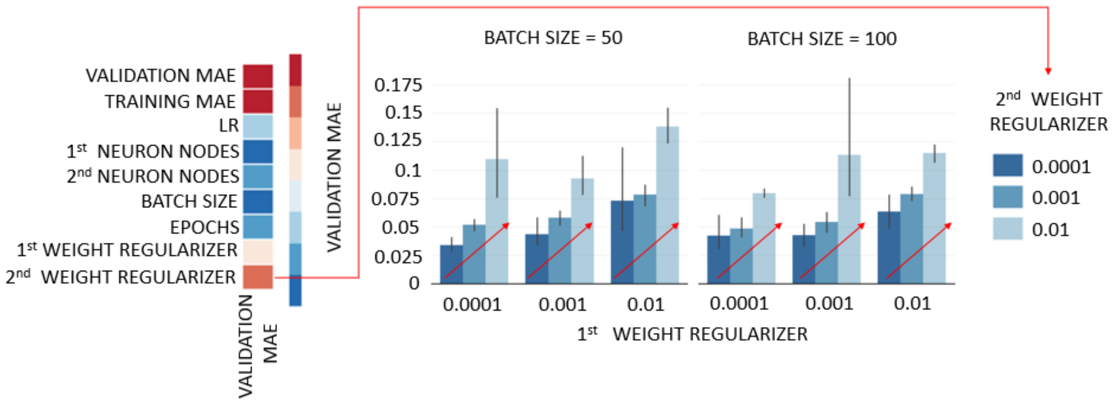


Figure 4.15. Talos correlation heatmap of the weight regularization investigation. Corresponding bar charts indicate that a lower 2nd weight regularizer correlates with a lower MAE

Figure 4.15 showcases how higher values of weight regularizers correlated with a lower validation MAE, indicating that the lower values were preferred.

Figure 4.16 displays the lowest validation MAE scores for all the optimisation investigations, including the baseline. The benefit of using Talos as an optimisation tool is clearly visible, the result being that a two-layer network configuration with minor weight regularizers proving to be the best ANN setup. The lowest validation MAE was in the weight regularization study with a value of 0.023.

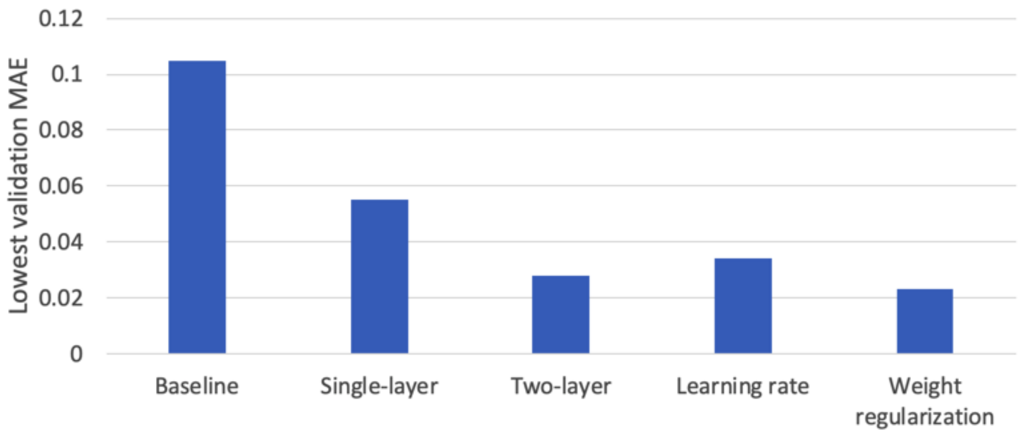


Figure 4.16. Bar chart displaying the lowest validation MAE scores from each optimisation study

4.2.6 Final model

Table 4.2 displays the optimal hyperparameter configuration resulting from the optimisation studies. *Figure 4.17* demonstrates the benefit of the optimisation process on the model. The coherence between the different folds as well the as high R^2 metric (0.979) outlines the robustness and prediction accuracy of the final ANN.

Table 4.2. Optimum hyperparameter configuration resulting from optimisation study with Talos

Epochs	Batch size	Learning rate	First neuron nodes	Second neuron nodes	First Neuron weight regularizer	Second Neuron weight regularizer
500	50	0.5	32	128	0.0001	0.0001

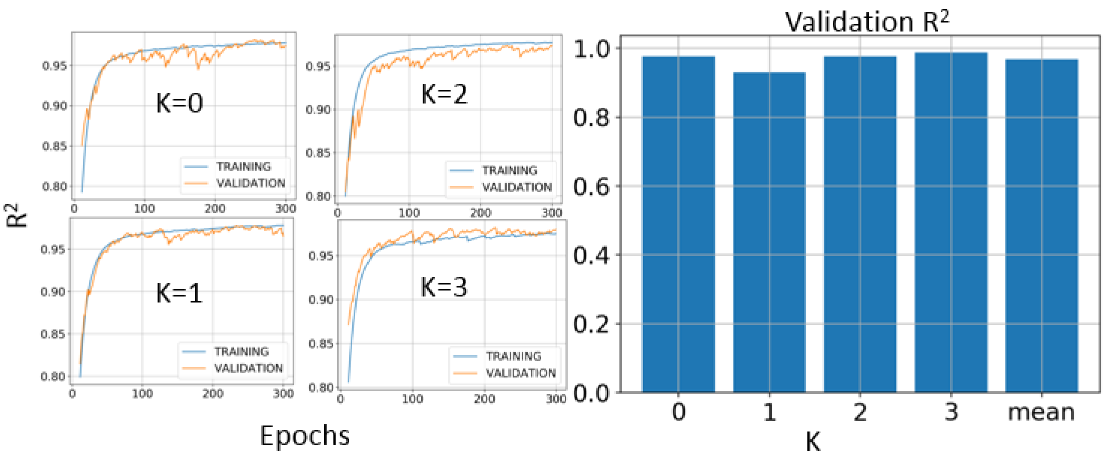


Figure 4.17. K-fold cross-validation analysis of the identified optimal hyperparameter configuration.

Figure 4.18 shows the optimised ANN in action. The errors between the true and predicted target values were marginal (a mean of 0.06 and 0.029 units for w_n and ξ respectively). The largest error was for highly under damped cases where ξ < 0.1. This was due to the significant under representation (7.8%) of that behaviour in the entire dataset as was seen in *Figure 4.10*.

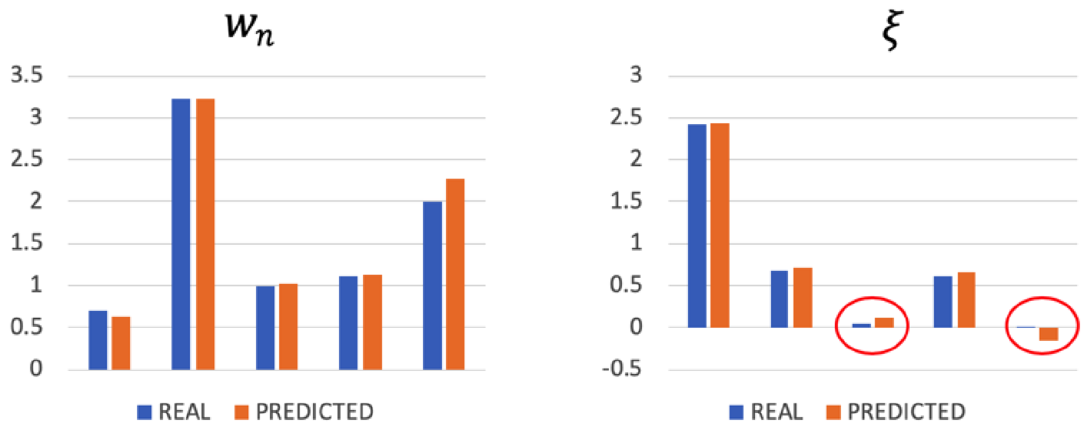


Figure 4.18. ANN deployment results on arbitrary targets. Highlighted cases of very under damped and largest error systems

4.2.7 *Discussion*

The practice examples served their purpose of developing familiarity with the software and tools as well as highlighted important aspects of ANN development unique to this project. It was shown that for a task of predicting the gradient in a generic linear ODE, a simple model without data pre-processing is sufficient. The achieved R^2 value being 0.995. A more complex problem of a second order ODE system required more thought and effort to accomplish a working ANN. The resultant R^2 value being 0.979. The lessons taken away from the practice examples are as follows:

- Using w_n and ξ as target outputs as opposed to m, k and c infers the importance to use a set of target variables of which relationships in the input data provide enough representation of the output targets
- Using K-fold cross-validation serves as a useful validation tool to ensure the generated model works accordingly on a variety of data
- Standardization improves model performance in multi-output regression
- Enforcing a specific distribution in the data may hinder the model performance
- Increasing the dataset size is beneficial as it provides more resources for the model to develop relationships between the input and output targets, improving its performance
- The Talos Python library proves to be a beneficial tool for optimizing and understanding the ANN hyperparameters

The insight gained from this experience was employed in the enhancing of the PA mathematical model, detailed in Chapter 5.

5. P. aeruginosa model enhancement

5.1 Model replication

It was necessary in this study to ensure that the mathematical model used was validated as equivalent to that of Dockery's (2001). As mentioned in 2.5, the researchers illustrated the QS 'switch' behaviour from a negligible concentration rise to an explosive growth in the autoinducer concentration. This was achieved through the varying of the variable of local cell density ρ. *Figure 5.1* shows for how low ρ values, the concentration of A reaches a steady state at a negligible level. However, upon, the increase of ρ to 0.15 there is an explosive increase in the autoinducer concentration. *Figure 5.1* is analogous to the study carried out by Dockery (2001), thus validating the ODEs implemented in Python.

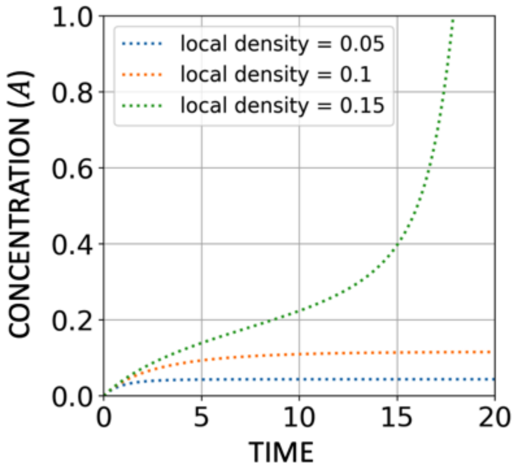


Figure 5.1. (A) concentration over time plot for different local density (ρ) values, ρ = 0.15 showing an explosive growth in autoinducer concentration and QS 'switch'

5.2 Dataset information.

As there are 11 constants that can be modified from the original system of ODEs, a systemic method of iteratively building up the number of constant outputs and their respective ranges was employed. In the first instance, coefficients ρ and δ were selected as outputs with their respective ranges being $\pm$40% from the values used by Dockery (2001), whilst all the other coefficients were kept equal to the values used in the literature. The system of ODEs defined by equations (1.1), (1.2) and (1.3) was integrated. Only the integral output of (1.2) (A concentration) was used as the input data, whilst the R concentration data was discarded. This more closely reflected a real-life scenario of solely monitoring the autoinducer concentration in a lab environment. Additional dataset parameters included:

- input data being generated over a total time of 40 units with a sampling frequency of 2.5Hz, resulting in each sample containing 100 time points.

- output target labels (ODE coefficients) being generated to 4 decimal points.
- the total dataset consisting of 10,000 samples.

To ensure a good representation in the generated data, an equal distribution of the two autoinducer concentration growth patterns was assimilated. This was done by calculating the difference between the initial and final values of each integral. *Figure 5.2* shows the distribution of the differences, clearly indicating the significant gap in the concentration differences. A difference of < 1 indicated that the threshold had not been reached whilst the contrary indicated the explosive autoinducer production. The final dataset contained 4,027 small steady state solutions and 5,973 exploding growth solutions. This was regarded as a representative distribution of observed behaviours. The dataset was then split into training (80%) and testing data (20%).

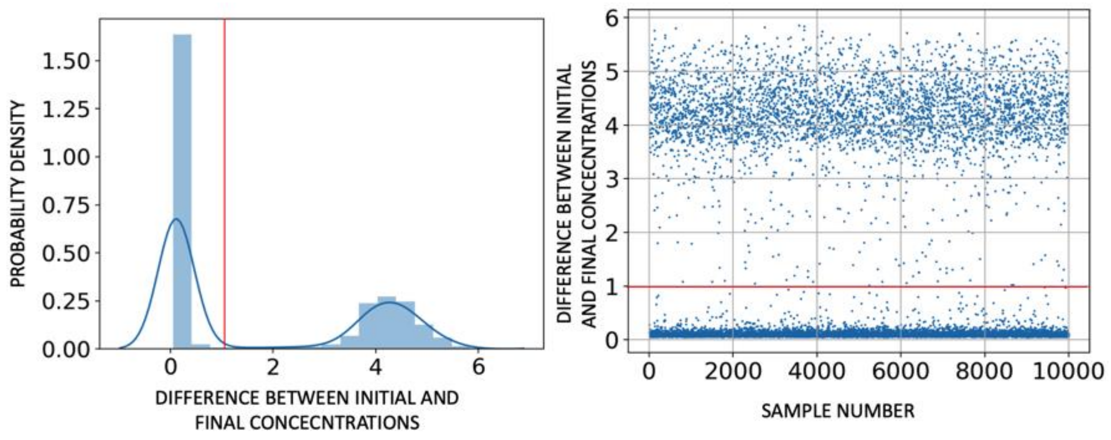


Figure 5.2. Distribution and scatter plot of the difference between final and initial autoinducer concentrations - illustrating the two types of behaviors. Red lines separate the two observed growth patterns.

5.3 Model build

Following the lessons learned from Chapter 4, the Talos Python library was used to build an ANN with optimized hyperparameters. These are detailed in *Table 5.1.* Correspondingly, *Figure 5.3* shows the K-fold cross validation output for K= 4 folds. Coherence between the different validation folds is maintained to an acceptable degree. However, the R^2 score plateaus at a relatively low value of 0.74 compared to the final models produced in Chapter 4. The final MAE score on the test data was 0.022.

Table 5.1. Optimum hyperparameter configuration for enhanced QS mathematical model for the initial dataset

Epochs	Batch size	Learning rate	First neuron nodes	Second neuron nodes	First Neuron weight regularizer	Second Neuron weight regularizer
300	200	1	16	64	0	0

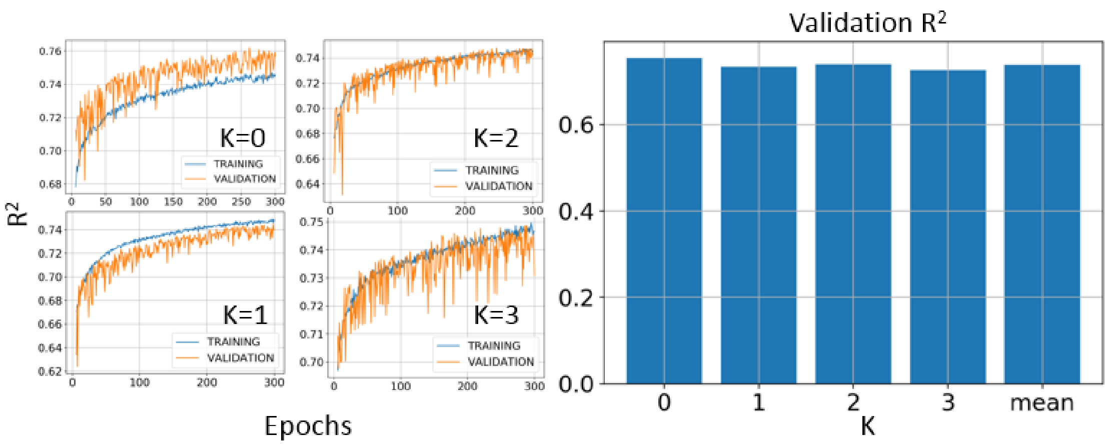


Figure 5.3 K-fold cross-validation analysis of the identified optimal hyperparameter configuration

The main goal was to build an ANN capable of computing the correct trajectory - without the strict importance of a high accuracy of the predicted coefficients. The model performance was thus tested by comparing the trajectories defined by the output constants against true trajectories defined by the original values. *Figure 5.4* displays a comparison of 3 randomly generated trajectories and the ANN-predicted solutions *Figure 5.5* displays the mean percentage error between the true and predicted target coefficients. It was demonstrated that the ANN was effectively predicting concentration trajectories, albeit not necessarily alongside correct target coefficients.

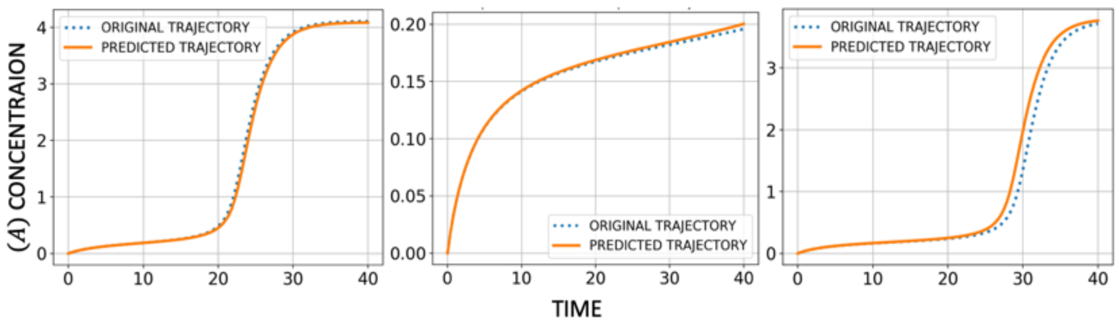


Figure 5.4. Comparison of true and ANN-predicted concentration trajectories for the initial dataset

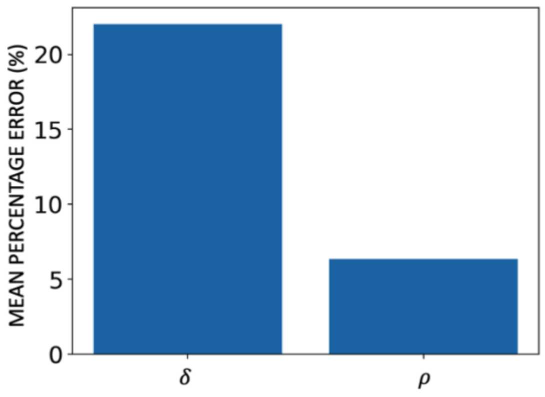


Figure 5.5. Bar chart showing the mean percentage error for the prediction of two target ODE coefficients

Another metric for measuring the error was implemented. This included calculation of the mean of the errors for each timepoint between the true and predicted concentrations. The final calculated value was **0.0512 concentration units**. This deemed the model accurate enough with respect to correctly predicting autoinducer concentration trajectories.

5.4 Increase in output targets and their ranges

To iteratively build on the model capabilities, two more ODE constants - k_A and k_E - in the range of $\pm$ 40% from the values used by Dockery (2001), were added to the vector of targets of the ANN. Using the same model parameters as those presented in *Table 5.1*, the model's performance expectedly decreased, with a mean error for each time point between true and predicted concentrations rising to **0.0637 concentration units**. *Figure 5.6* displays the mean percentage error of the updated four ODE constants. An expected rise in the error margin is observed in comparison to *Figure 5.5.*

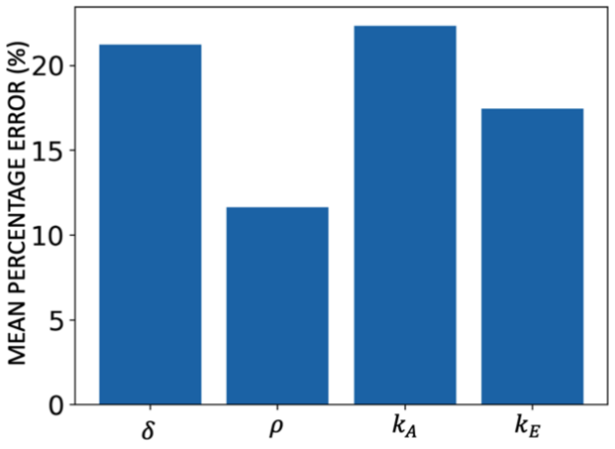


Figure 5.6. Bar chart showing the mean percentage error for the prediction of four target ODE coefficients

The rise in the error margin was regarded as negligible, and thus the project progressed into adding more complexity to the ANN. The range of the four predicted ODE constants was increased from $\pm$ 40% to $\pm$ 80%. Again, the same model hyperparameters from *Table 5.1* were implemented and tested on the new target outputs. On this occasion, the mean concentration error for each time point between true and predicted concentrations exhibited a drastic rise to **0.908 concentration units**. *Figure 5.7* displays comparisons of true and model-predicted trajectories alongside a bar chart illustrating the mean percentage error of the 4 predicted coefficients in the test data. The verdict was that the increase in the target output dimensionality caused a significant decrease in model performance down to an unacceptable standard.

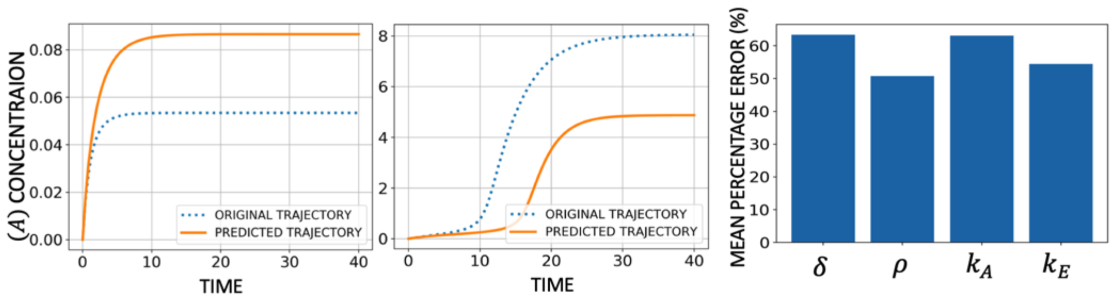


Figure 5.7. Comparison of true and model-predicted concentration trajectories and bar chart of mean percentage prediction error

The methods employed to improve the ANN performance included:

- Further hyperparameter optimisation with the Talos library
- Implementation of a convolution neural network (not discussed in this document)
- Use of initially discarded R concentration in parallel with A concentration data
- Increase of dataset size to 100,000 samples

None of these amendments successfully rectified the model performance, and due to time constraints, further investigation was not possible. However, if more time was made available, an approach aimed at reducing data dimensionality such as principal component analysis would be investigated.

5.5 Discussion

A systematic approach to building an ANN by gradually increasing its complexity was adopted. This consisted of increasing the number of target outputs as well as their ranges, respective to the original values used in the literature. In the first instance of predicting two ODE coefficients (ρ and δ) with a $\pm$ 40% range, the R^2 metric did not exceed a value of 0.74. However, when the output trajectories of the predicted coefficient output were compared against trajectories from the correct combination of ODE constants, successful similarity was observed. This led to a development of new metric – calculating the average error in the concentration at each time point for every sample in a test dataset.

The same hyperparameter configuration was deemed to be good enough to predict an additional 2 outputs (k_A and k_E), increasing the conventional and new metrics only by a small margin. However, when the range of the target values was expanded to $\pm$ 80%, the constructed ANN did not perform well, even with additional optimisation, data and further investigation into the problem. Provided more time, suggested work would entail dimensionality reduction through a process such as principal component analysis.

6. Conclusions and Future Work

6.1 Achievements

The main achievements of this project were:

- The building of a technical foundation in machine learning from practice examples of a linear equation and 2nd order ODE mass/spring/damper system.
- The successful replication and validation of the mathematical model presented in the literature by Dockery (2001)
- The successful construction of datasets used in ANN development
- The development of an enhanced mathematical model, capable of accurately predicting concentration trajectories by the estimation of 4 ODE coefficients varying $\pm$ 40% from the values presented in literature

6.2 Discussion

The initial practice example of predicting the gradient in a generic linear equation successfully built a foundation into the construction of datasets ANNs, as well as established a technical familiarity with the SciPy and Keras Python libraries. Dataset generation was done through the integration of the linear ODE (4.1), with the solutions and corresponding target labels saved accordingly. The constructed ANN did not require any significant pre-processing of the data or model hyperparameter optimisation and achieved an R^2 score of 0.995. This was due to the simple nature of the problem and single target prediction (as opposed to multiple), rendering the problem fairly low-dimensional.

The subsequent case study involving the mass/spring/damper system – a common engineering system and 2nd order ODE model – posed a greater challenge. The targets were modified from the coefficients in the system (m, k and c) to the physical characteristics - w_n and ξ. This was due to the integral solution of the ODE not being unique to a specific set of m, k and c values, meaning that different combinations of coefficients could output identical displacement trajectories. Switching to w_n and ξ as output targets, simplified the problem in terms of dimensionality, as the oscillatory behaviour of the system is readily described by these two parameters.

The implementation of data standardisation proved to be largely beneficial in this project. Upon its implementation, improvement in ANN performance through a 52% increase in R^2 was observed. Furthermore, the use of K-fold cross-validation gave insight into the robustness of a constructed ANN when trained on different subsets of a training dataset.

It is paramount to ensure the dataset used for model building is representative of the investigated problem. However, in this study, the enforcement of a specific distribution of 3 damping behaviours proved to impede the model by preventing an effective loss optimisation as well as induced variable performance during cross-validation. Generating data with a random distribution was more beneficial and eradicated the previously witnessed variability. In addition, the freedom to generate datasets proved beneficial, as there was an observed improvement in the ANN performance when more data was supplied for its training. However, it should be noted that the freedom also imposes an obligation for the user for ensuring a representative distribution of the data – increasing the magnitude of the challenge.

Once a working model was established with good quality datasets, the Talos Python library demonstrated the potential of optimising model hyperparameters to achieve an optimum ANN configuration. Additionally, it further smoothened the variability in performance between different input data subsets during cross-validation. Undertaking hyperparameter optimisation with Talos diminished the MAE from 0.105 to 0.023 (78%). The final model performed most poorly in prediction of targets for highly underdamped cases ($\xi < 0.1$). This was attributed to the small presence of the behaviour in the dataset used in the ANN training.

For the enhancement of the PA mathematical model, an iterative approach to increasing the ANN complexity was undertaken. This consisted of gradually increasing the number of targets as well as their respective ranges once an acceptable model was constructed. After employing previously gained insights such as data standardisation and Talos optimisation, the R^2 metric for the initial and most basic case were relatively low (0.74) compared to the achievements of prior examples. However, high performance was demonstrated when the model was tested in prediction of concentration trajectories. Although the predicted coefficients were slightly different to the true values, the reproduced autoinducer trajectories were nearly identical with a mean concentration error from all time points of 0.0512 units. Difficulty was experienced when expanding the range of the predicted targets to $\pm$ 80%. Upon this complexity increase, it was found that further hyperparameter optimisation as well as several other ML techniques did not provide any improvement ANN performance. Due to the time constraints, there was not enough opportunity for further investigation. A study into dimensionality reduction techniques such as principal component analysis is suggested as the next steps in the project. These would simplify the data inputted into the ANN and consequently, the problem at hand.

6.3 Conclusion

The potential of using ML methods and the deep learning approach of an ANN has been demonstrated in a MO regression problem. The work carried out in this study is unique in its nature because of the self-generation of data as well as its time-series quality. Nevertheless, conventional procedures of data processing and model development still contributed to producing an ANN with a high prediction accuracy. Through an iterative approach in both, optimisation and increasing model complexity, it was demonstrated how increasing insight can be obtained to tackle a highly dimensional problem.

The prowess of ANNs in regression problems have been shown through the following:

- Prediction of gradient constant values in a simple linear ODE from a total dataset of 2000 samples. The resultant R^2 of the model being 0.995
- Prediction of w_n and ξ parameters of a mass/spring/damper system from a dataset of 10,000 samples. The resultant R^2 of the model being 0.979
- Successful prediction of 4 ODE constants, ranging $\pm$ 40% from the values presented by Dockery (2001) in literature.

Increasing problem complexity by escalating the constant range to $\pm$ 80% rendered the model ineffective and not able to rectify through various approaches in the given timeframe. However, further investigation into dimensionality reduction is suggested

6.4 Future work

The project poses the potential for using modern ML methods to enhance the mathematical model for QS presented by Dockery (2001). The work showcased in this document is basic in its nature and can be iterated on, with the possibility of extending it to a real-life clinical application with the potential to save lives. Suggested future work to further improve the showcased model performance includes:

- Comparison of ANNs with other ML techniques such as random forest models
- Application of dimensionality reduction using methods such as principal component analysis to simplify the input dataset used for ANN training
- Transformation of the MO regression problem into separate single target problems, where a separate ANN would be built for each target. A comparison against the collective approach presented within this report could then be made
- Validation against real data obtained from experiment, to evaluate whether the approach is capable of predicting real-life observation and therefore has the potential of being used in a medical application

7. References

Antanasijević, D., Pocajt, V., Perić-Grujić, A. and Ristić, M. (2018). Multiple-input–multiple-output general regression neural networks model for the simultaneous estimation of traffic-related air pollutant emissions. *Atmospheric Pollution Research*, 9(2), pp.388-397.

Azam, M. and Khan, A. (2018). Updates on the pathogenicity status of Pseudomonas aeruginosa. Drug Discovery Today.

Bassetti, M., Vena, A., Croxatto, A., Righi, E. and Guery, B. (2018). How to manage Pseudomonas aeruginosa infections. *Drugs in Context*, 7, pp.1-18.

Borchani, H., Varando, G., Bielza, C. and Larrañaga, P. (2015). A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(5), pp.216-233.

Campa, M., Bendinelli, M. and Friedman, H. (1993). Pseudomonas aeruginosa as an Opportunistic Pathogen. Boston, MA: Springer US.

Chollet, F. (2017). *Deep learning with Python*. Shelter Island, New York: Manning Publications Co.

Dockery, J. (2001). A Mathematical Model for Quorum Sensing in Pseudomonas aeruginosa. *Bulletin of Mathematical Biology*, 63(1), pp.95-116.

Guillaumin, M., Verbeek, J. and Schmid, C. (2010). Multimodal semi-supervised learning for image classification. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Hastings, J. and Nealson, K. (1977). Bacterial Bioluminescence. Annual Review of Microbiology, 31(1), pp.549-595.

Kalogirou, S. (2000). Applications of artificial neural-networks for energy systems. *Applied Energy*, 67(1-2), pp.17-35.

Kang, Tim. (2015), "Using Neural Networks for Image Classification". *Master's Projects*. 395. https://scholarworks.sjsu.edu/etd_projects/395

Lee, J. and Zhang, L. (2014). The hierarchy quorum sensing network in Pseudomonas aeruginosa. *Protein & Cell*, 6(1), pp.26-41.

Lowe, D., Emsley, M. and Harding, A. (2006). Predicting Construction Cost Using Multiple Regression Techniques. *Journal of Construction Engineering and Management*, 132(7), pp.750-758.

Meriam-Webster. (2019). Definition of artificial intelligence. (2019). In: *artificial intelligence noun*. [online] Available at: https://www.merriam-webster.com/dictionary/artificial%20intelligence [Accessed 3 Apr. 2019].

Miller, M. and Bassler, B. (2001). Quorum Sensing in Bacteria. *Annual Review of Microbiology*, 55(1), pp.165-199.

Morales, E., Cots, F., Sala, M., Comas, M., Belvis, F., Riu, M., Salvadó, M., Grau, S., Horcajada, J., Montero, M. and Castells, X. (2012). Hospital costs of nosocomial multi-drug resistant Pseudomonas aeruginosa acquisition. *BMC Health Services Research*, 12(1).

Nicholas Refenes, A., Zapranis, A. and Francis, G. (1994). Stock performance modeling using neural networks: A comparative study with regression models. *Neural Networks*, 7(2), pp.375-388.

Papenfort, K. and Bassler, B. (2016). Quorum sensing signal–response systems in Gram-negative bacteria. *Nature Reviews Microbiology*, 14(9), pp.576-588.

Parai, D., Banerjee, M., Dey, P., Chakraborty, A., Islam, E. and Mukherjee, S. (2018). Effect of reserpine on Pseudomonas aeruginosa quorum sensing mediated virulence factors and biofilm formation. *Biofouling*, 34(3), pp.320-334.

Pérez-Velázquez, J., Gölgeli, M. and García-Contreras, R. (2016). Mathematical Modelling of Bacterial Quorum Sensing: A Review. *Bulletin of Mathematical Biology*, 78(8), pp.1585-1639.

Punjani, A. and Abbeel, P. (2015). Deep learning helicopter dynamics models. *2015 IEEE International Conference on Robotics and Automation (ICRA)*.

Smith, R. and Iglewski, B. (2003). Pseudomonas aeruginosa quorum sensing as a potential antimicrobial target. *Journal of Clinical Investigation*, 112(10), pp.1460-1465.

Sözen, A. and Arcaklioglu, E. (2007). Prediction of net energy consumption based on economic indicators (GNP and GDP) in Turkey. *Energy Policy*, 35(10), pp.4981-4992.

Stover, C., Pham, X., Erwin, A., Mizoguchi, S., Warrener, P., Hickey, M., Brinkman, F., Hufnagle, W., Kowalik, D., Lagrou, M., Garber, R., Goltry, L., Tolentino, E., Westbrock-Wadman, S., Yuan, Y., Brody, L., Coulter, S., Folger, K., Kas, A., Larbig, K., Lim, R., Smith, K., Spencer, D., Wong, G., Wu, Z., Paulsen, I., Reizer, J., Saier, M., Hancock, R., Lory, S. and Olson, M. (2000). Complete genome sequence of Pseudomonas aeruginosa PAO1, an opportunistic pathogen. *Nature*, 406(6799), pp.959-964.

Taylor, P., Yeung, A. and Hancock, R. (2014). Antibiotic resistance in Pseudomonas aeruginosa biofilms: Towards the development of novel anti-biofilm therapies. *Journal of Biotechnology*, 191, pp.121-130.

Zell, A. (1994). *Simulation Neuronaler Netze [Simulation of Neural Networks] (In German)*. 1st ed. Bonn: Addison-Wesley, p.73.

8. Appendix

8.1 Data generation code with 4 output targets varying (ρ, δ, k_A and k_E) +/- 40% from their initial values

```python
# data generation code that varies only in terms of p, d, kE and kA and has four targets by 40%

import random
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint #shortens scipy.integrate.odeint to just odeint


##############################################################################
# defining the ODE to be integrated in this function
def model(Z, t):
    # Z a list of solutions for R and A
    R = Z[0]
    A = Z[1]

    # defining the ODE for R - according to equation (20) in the 2001 paper
    dRdt = (-kR * R) + ((VR * R * A) / (KR + (R * A))) + R0

    # defining the equation for decay rate - according to equation (21) in the 2001 paper
    dp = kA + ((d / p) * ((kE * (1 - p)) / (d + (kE * (1 - p)))))

    # defining the ODE for A - according to equation (21) in the 2001 paper
    dAdt = ((VA * R * A) / (KA + (R * A))) + A0 - (dp * A)

    # create a list of the 2 equations, 1st one is R, second is A
    dZdt = [dRdt, dAdt]
    return dZdt
##############################################################################

# hardcoding constants with values from literature
VR = 2.0
VA = 2.0
KR = 1.0
KA = 1.0
R0 = 0.05
A0 = 0.05
kR = 0.7

##############################################################################

#initial condition
Z0 = [0,0]

# defining time period
fin_time = 40
time_step = 100 # also defines the number of time points each data set will have
t = np.linspace(0,fin_time, time_step)

# empty lists for R and A FOR PLOTTING
R_plot = []
A_plot = []
```

```python

# defining how many datasets I want to produce
tot_data = 2000
##############################################################################
# initialising empty target lists
d_list = []
p_list = []
kA_list = []
kE_list = []

const_list_str = ['d', 'p', 'kA', 'kE']
##############################################################################

# empty list for analysing differences between initial and final concentrations
diff_R = []
diff_A = []

# empty containers fot storing the actual ODE solutions
R_sols = []
A_sols = []

# container for solutions when trying to do 3D array data (R and A in the same container)
sols = []

for i in range(tot_data):
    # Miscellaneous
    d = round(random.uniform(0.12, 0.28), 4)  # Diffusion conductance through cell membrane
    p = round(random.uniform(0.06, 0.16), 4)  # Local density (volume fraction) of cell
    kA = round(random.uniform(0.012, 0.028), 4) # Natural degradation rate of intracellular 3-oxo-C12-HSL (autoinducer)
    kE = round(random.uniform(0.06, 0.14), 4) # Natural degradation rate of extracellular 3-oxo-C12-HSL (autoinducer)

    const_list_float = [d, p]
    # appending list for targets
    d_list.append(d)
    p_list.append(p)
    kA_list.append(kA)
    kE_list.append(kE)

    # solving the ode
    Z = odeint(model, Z0, t)
    Z = np.transpose(Z, axes=None) # comment out if I want to plot!!!

    sols.append(Z) # this appends both of the solved ODEs into what would be a 3D array of shape (1000, 2, 100)

    # appending solutions FOR PLOTTING ONLY
    R = Z[0]
    A = Z[1]

        # appending empty list with concentration trajectory solutions
        R_sols.append(R)
        A_sols.append(A)

        # appending the differences between the final and initial values

        diff_R.append(abs(R[0]-R[-1]))
        diff_A.append(abs(A[0]-A[-1]))

```

```python

            # plotting the ODEs to check behaviour
            # COMMENT THIS OUT TO COMPLETE CODE
            plt.plot(t,Z[0], linewidth = 2, linestyle = '-',label = 'R' )
            plt.plot(t,Z[1], linewidth = 2, linestyle = ':',label = 'A')
            plt.title('TOXIN CONCENTRATION OVER TIME')
            plt.xlabel('TIME')
            plt.legend(loc='best')
            plt.ylabel('CONCENTRATION')
            plt.grid()

            plt.show()

        ###############################################################################

        # plotting the distribution of distances
        sns.distplot(diff_R, hist=True, rug=False, label = 'R')
        sns.distplot(diff_A, hist=True, rug=False, label = 'A')
        plt.xlabel('difference between concentrations')
        plt.ylabel('probability density')
        plt.legend(loc = 'best')
        plt.show()

        # plotting the scatter of concentration differences
        plt.scatter(range(len(diff_R)), diff_R, label = 'R', s = 2, marker = 'x')
        plt.scatter(range(len(diff_A)), diff_A, label = 'A', s = 2, marker = '.')
        plt.xlabel('sample number')
        plt.ylabel('absolute difference betwen final and initial values')
        plt.legend(loc = 'best')
        plt.grid()
        plt.show()

        ###############################################################################

        # seeing how many of the differences are greater than 1 to define QS switch
        R_switch = sum(i > 1 for i in diff_R)
        A_switch = sum(i > 1 for i in diff_A)

        print('total of R QS switches = ' + str(R_switch))
        print('total of A QS switches = ' + str(A_switch))

        ###############################################################################

        # stacking and saving the input and target arrays.
        targets = np.column_stack((d_list, p_list))
        targets = np.column_stack((targets, kA_list))
        targets = np.column_stack((targets, kE_list))
        sols = np.asarray(sols)
        A_sols = np.asarray(A_sols)

        # save data to files
        #np.save('0_p_d_ODE_sols_3D_4dp_100000', sols)
        #np.savetxt('0_p_d_kA_kE_ODE_sols_TEST_2000.txt', A_sols, delimiter= ',' )
        #np.savetxt('0_p_d_kA_kE_targets_TEST_2000.txt', targets, delimiter= ',')
```

8.2 ANN construction code with 4 output targets varying (ρ, δ, k_A and k_E) +/- 40% from their initial values

```python
# main code building the optimised ANN

import sys
sys.path.insert(0, '../../tools')
import tools
from keras import models
from keras import layers
import numpy as np
from keras import optimizers
import matplotlib.pyplot as plt

##############################################################################
# read in data
data_filename = '0_p_d_kA_kE_ODE_sols.txt'
label_filename = '0_p_d_kA_kE_targets.txt'

# read in data
data = tools.read_data(data_filename)
labels = tools.read_data(label_filename)
##############################################################################
# splitting data into training and test
# setting ratio dictating how we are going to divide the data into training and testing
ratio = 0.8
train_data, test_data, train_labels, test_labels = tools.data_split(data,labels, ratio)
train_data = np.asarray(train_data)
test_data = np.asarray(test_data)
train_labels = np.asarray(train_labels)
test_labels = np.asarray(test_labels)

##############################################################################
# standardising by the test data to make mean 0 and std = 1
# here we are normalising inputs with method 1 - where each time step can be considered as a unique feature

train_data_mean = train_data.mean(axis = 0)
train_data -= train_data_mean
test_data -= train_data_mean

train_data_std = train_data.std(axis = 0)
train_data /= train_data_std
test_data /= train_data_std

train_data[:,0] = 0 # changing all the values at t = 0 to 0s because they are nan [:,:,0] if working with 3D array
test_data[:,0] = 0 # same for test data
##############################################################################
num_epochs = 300
batch_size = 50

# building of model
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(train_data.shape[1],),
                           kernel_initializer='normal'))
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(4, activation = 'linear'))
```

```

rmsprop = optimizers.RMSprop(lr=0.001)

model.compile(optimizer=rmsprop, loss='mse', metrics=[tools.coeff_determination, 'mae'])

# run the neural network model
history = model.fit(train_data, train_labels, epochs= num_epochs,
                        batch_size = batch_size, validation_data=(test_data, test_labels))
##########################################################################

# extract history data from the model.
history_dict = history.history
loss_values = history_dict['loss'] # Train data loss values
val_loss_values = history_dict['val_loss'] # Validation data loss values
train_mae = history_dict['mean_absolute_error']  # Train data MAE values
val_mae = history_dict['val_mean_absolute_error']  # Validation data MAE values
train_r2 = history_dict['coeff_determination'] # Train data R2 VALUES
val_r2 = history_dict['val_coeff_determination'] # Validation data R2 values

epochs = range(1, len(loss_values) + 1)

##########################################################################

# Plotting of raw training and validation data
# MSE
plt.plot(epochs[10:], loss_values[10:], label = 'TRAINING')
plt.plot(epochs[10:], val_loss_values[10:], label = 'VALIDATION')
plt.title('TRAINING VS VALIDATION LOSS')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.show()

# R2
plt.plot(epochs[10:], train_r2[10:], label = 'TRAINING')
plt.plot(epochs[10:], val_r2[10:], label = 'VALIDATION')
plt.title('TRAINING VS VALIDATION MAE')
plt.xlabel('Epochs')
plt.ylabel('R2')
plt.legend()
plt.show()

# MAE
plt.plot(epochs[10:], train_mae[10:], label = 'TRAINING')
plt.plot(epochs[10:], val_mae[10:], label = 'VALIDATION')
plt.title('TRAINING VS VALIDATION MAE')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()
plt.show()

##########################################################################

# using smoothing function in tools Python file to provide smoother plots of training history
smooth_loss = tools.smooth_curve(loss_values)
smooth_val_loss = tools.smooth_curve(val_loss_values)
smooth_train_r2 = tools.smooth_curve(train_r2)
smooth_val_r2 = tools.smooth_curve(val_r2)
smooth_train_mae = tools.smooth_curve(train_mae)
```

```
        smooth_val_mae = tools.smooth_curve(val_mae)

        # Plotting of raw training and validation data
        # MSE
        plt.plot(epochs[10:], smooth_loss[10:], label = 'TRAINING')
        plt.plot(epochs[10:], smooth_val_loss[10:], label = 'VALIDATION')
        plt.title('TRAINING VS VALIDATION LOSS')
        plt.xlabel('Epochs')
        plt.ylabel('MSE')
        plt.legend(loc='best', prop={'size': 18})
        plt.tick_params(axis='both', labelsize=18)
        plt.grid()
        plt.show()

        # R2
        plt.plot(epochs[10:], smooth_train_r2[10:], label = 'TRAINING')
        plt.plot(epochs[10:], smooth_val_r2[10:], label = 'VALIDATION')
        plt.title('TRAINING VS VALIDATION R2')
        plt.xlabel('Epochs')
        plt.ylabel('R2')
        plt.legend(loc='best', prop={'size': 18})
        plt.tick_params(axis='both', labelsize=18)
        plt.grid()
        plt.show()

        # MAE
        plt.plot(epochs[10:], smooth_train_mae[10:], label = 'TRAINING')
        plt.plot(epochs[10:], smooth_val_mae[10:], label = 'VALIDATION')
        plt.title('TRAINING VS VALIDATION MAE')
        plt.xlabel('Epochs')
        plt.ylabel('MAE')
        plt.legend(loc='best', prop={'size': 18})
        plt.tick_params(axis='both', labelsize=18)
        plt.grid()
        plt.show()

        ##############################################################################

        # evaluate the model
        test_mse_score, test_r2_score, test_mae_score = model.evaluate(test_data, test_labels)
        print('RESULTS FOR TEST DATA ARE:')
        print('THE TEST LOSS IS: ' + str(test_mse_score))
        print(' THE TEST R2 IS: ' + str(test_r2_score))
        print(' THE TEST MAE IS: ' + str(test_mae_score))

        ##############################################################################

        # serialize model to JSON
        model_json = model.to_json()
        with open("model.json", 'w') as json_file:
            json_file.write(model_json)

        # serialize weights to HDF5
        model.save_weights("model.h5")
```

8.3 ANN evaluation code with 4 output targets varying (ρ, δ, k_A and k_E) +/- 40% from their initial values

```
import sys
sys.path.insert(0, '../../tools')
import tools
import numpy as np
```

```python
from keras.models import model_from_json
from random import randint
import matplotlib.pyplot as plt
from scipy.integrate import odeint #shortens scipy.integrate.odeint to just odeint
from statistics import mean

##############################################################################
# read in data
data_filename = '0_p_d_kA_kE_ODE_sols.txt'
label_filename = '0_p_d_kA_kE_targets.txt'

# read in data
data = tools.read_data(data_filename)
labels = tools.read_data(label_filename)
##############################################################################
# splitting data into training and test
# setting ratio dictating how we are going to divide the data into training and testing
ratio = 0.8
train_data, test_data, train_labels, test_labels = tools.data_split(data,labels, ratio)
train_data = np.asarray(train_data)
test_data = np.asarray(test_data)
train_labels = np.asarray(train_labels)
test_labels = np.asarray(test_labels)

##############################################################################
# standardising by the test data to make mean 0 and std = 1
# here we are normalising inputs with method 1 - where each time step can be considered as a unique feature

train_data_mean = train_data.mean(axis = 0)
train_data -= train_data_mean
test_data -= train_data_mean

train_data_std = train_data.std(axis = 0)
train_data /= train_data_std
test_data /= train_data_std

train_data[:,0] = 0 # changing all the values at t = 0 to 0s because they are nan [:,:,0] if working with 3D array
test_data[:,0] = 0 # same for test data
##############################################################################
# now read in prediction data
data_filename = '0_p_d_kA_kE_ODE_sols_TEST_2000.txt'
label_filename = '0_p_d_kA_kE_targets_TEST_2000.txt'
prediction_data = tools.read_data(data_filename)
prediction_labels = tools.read_data(label_filename)
orig_pred_data = prediction_data
orig_pred_labels = prediction_labels

prediction_data = np.asarray(prediction_data)
prediction_labels = np.asarray(prediction_labels)

##############################################################################
# standardizing prediction data
prediction_data -= train_data_mean
prediction_data /= train_data_std

```

```python
prediction_data[:,0] = 0 # changing all the values at t = 0 to 0s because they are nan
##############################################################################
# read in model in JSON format
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
##############################################################################

# load weights into new model
loaded_model.load_weights('model.h5')
print('loaded model')

#compile model
loaded_model.compile(optimizer='rmsprop', loss='mse', metrics=[tools.coeff_determination,'mae'])
##############################################################################
# running the model on testing data
val_loss, val_metric, val_mae = loaded_model.evaluate(prediction_data, prediction_labels, verbose=0)
print('the test loss is ' + str(val_loss))
print('the test metric is ' + str(val_metric))
print('the test mae is ' + str(val_mae))
##############################################################################
# print results
print('TESTING NEURAL NETWORK on standardised  = ' + str(prediction_labels[0])) # testing on first sample in prediction data ste
prediction_data_vec = np.asarray(prediction_data[0])
prediction_data_vec = prediction_data_vec.reshape(-1,100)
result = loaded_model.predict(prediction_data_vec)
print('STANDARDIZED RESULT IS ' + str(result))
##############################################################################

# hardcoding constants with values from figure 2
VR = 2.0
VA = 2.0
KR = 1.0
KA = 1.0
R0 = 0.05
A0 = 0.05
kE = 0.1
kR = 0.7
kA = 0.02

    # defining the ODE to be integrated in this function
    def model(Z, t):
        # Z a list of solutions for R and A
        R = Z[0]
        A = Z[1]

        # defining the ODE for R - according to equation (20) in the 2001 paper
        dRdt = (-kR * R) + ((VR * R * A) / (KR + (R * A))) + R0

        # defining the equation for decay rate - according to equation (21) in the 2001 paper
        dp = kA + ((d / p) * ((kE * (1 - p)) / (d + (kE * (1 - p)))))

        # defining the ODE for A - according to equation (21) in the 2001 paper
```

```
            dAdt = ((VA * R * A) / (KA + (R * A))) + A0 - (dp * A)

            # create a list of the 2 equations, 1st one is R, second is A
            dZdt = [dRdt, dAdt]
            return dZdt
        ##############################################################################
        # plotting some of the result real trajectories to those outputted by the constants predicted by NN
        # initial condition
        Z0 = [0,0]

        # defining time period
        fin_time = 40
        time_step = 100 # also defines the number of time points each data set will have
        t_long = np.linspace(0,fin_time, time_step)
        t = t_long[0::10]
        ##############################################################################

        # for loop that plots some of the sample of prediction data and resultant model data
        for i in range(100):
            # assigning random index
            index = randint(0,len(prediction_data))

            # plotting the original real data
            plt.plot(t_long, orig_pred_data[index], linewidth=1, linestyle=':', label='real data d = '
                                                    + str(orig_pred_labels[index][0]) + ' p = ' + str(orig_pred_labels[index][1]))
            plt.title('Comparison of real vs model predicted trajectories')
            plt.xlabel('time')
            # plt.legend(loc='best')
            plt.ylabel('concentration')
            plt.grid()

            # solving the constants by inputting the data
            # Miscellaneous
            print('TESTING NEURAL NETWORK on standardised again = ' + str(prediction_labels[index]))
            prediction_data_vec = np.asarray(prediction_data[index])
            prediction_data_vec = prediction_data_vec.reshape(-1, 100)
            result = loaded_model.predict(prediction_data_vec)
            print('STANDARDIZED RESULT IS again' + str(result))
            d = result[0][0]# Diffusion conductance through cell membrane
            p = result[0][1]  # Local density (volume fraction) of cell

            # solving the ode
            Z = odeint(model, Z0, t_long)
            Z = np.transpose(Z, axes=None)  # comment out if I want to plot!!!

            # appending solutions FOR PLOTTING ONLY
            R = Z[0]
            A = Z[1]
            # appending the solutions to check the error after.
            #A = A[0::10]

            # plotting the ODEs to check behaviour
            #plots against time
            plt.plot(t_long,A, linewidth = 1, linestyle = '-',label = 'model prediction d = ' + str(result[0][0]) +
                     ' p = ' + str(result[0][1]))
            plt.xlabel('time')
```

```python
        plt.legend(loc='best')
        plt.ylabel('concentration')
        plt.tick_params(axis = 'both', labelsize = 18)
        plt.show()

    ###############################################################################
    # for loop that will iterate through all the prediction data and calculate the mean absolute error nad r2?
    # by checking how different each time point is from the concentration

    sols = []
    for i, j in enumerate(orig_pred_labels):

        # solving the constants by inputting the data
        # Miscellaneous
        print('TESTING NEURAL NETWORK on standardised again = ' + str(prediction_labels[i]))
        prediction_data_vec = np.asarray(prediction_data[i])
        prediction_data_vec = prediction_data_vec.reshape(-1, 100)
        result = loaded_model.predict(prediction_data_vec)
        print('STANDARDIZED RESULT IS again' + str(result))
        d = result[0][0]  # Diffusion conductance through cell membrane
        p = result[0][1]  # Local density (volume fraction) of cell

        # solving the ode
        Z = odeint(model, Z0, t_long)
        Z = np.transpose(Z, axes=None)  # comment out if I want to plot!!!

        # appending solutions FOR PLOTTING ONLY
        R = Z[0]
        A = Z[1]
        sols.append(A)
    ###############################################################################
    # section calculating the average error for constants
    # initialising empty target lists
    pred_d_list = []
    pred_p_list = []

    for i,j in enumerate(orig_pred_data):
        print('TESTING NEURAL NETWORK on standardised again = ' + str(prediction_labels[i]))
        prediction_data_vec = np.asarray(prediction_data[i])
        prediction_data_vec = prediction_data_vec.reshape(-1, 100)
        result = loaded_model.predict(prediction_data_vec)
        print('STANDARDIZED RESULT IS again' + str(result))
        d = result[0][0]  # Diffusion conductance through cell membrane
        p = result[0][1]  # Local density (volume fraction) of cell

        pred_d_list.append(d)
        pred_p_list.append(p)

    # calculating the errors for the different coefficients
    err_d = []
    err_p = []
    for i,j in enumerate(orig_pred_labels):
        err_d.append(abs(j[0] - pred_d_list[i]))
        err_p.append(abs(j[1] - pred_d_list[i]))

    d_mean = mean(err_d)
    p_mean = mean(err_p)
    mean_scores = [d_mean, p_mean]
    const_list = ['d', 'p']
```

```python
        # plotting bar chart of the different constant means from the test data
        plt.bar(const_list, mean_scores)
        plt.title('MEAN CONSTANT ERROR')
        plt.xlabel('Constant')
        #plt.xticks(x_pos, const_list)
        plt.ylabel('MEAN ABSOLUTE ERROR')
        plt.show()
        ###############################################################################
        # calculating the error at each time point
        ers_all = []
        for i, j in zip(orig_pred_data,sols):
            ers_ind = [] # empty list to calculate each error
            for x, y in zip(i, j):
                ers_ind.append(abs(x-y))
            ers_all.append(ers_ind)

        ers_mean = []
        for i in ers_all:
            ers_mean.append(mean(i))
        total_er = mean(ers_mean)
        print('THE AVERAGE ABSOLUTE ERROR OF ALL THE AVERAGE ERRORS AT EACH TIME POINT IS ' + str(total_er))
```

All other related code can be found following this link:

https://github.com/jan-zajac/DISSERTATION_GIT