Bioprocess Simulation and Optimisation Using Machine Learning

A dissertation submitted to

The University of Manchester

for the degree of

Master of Engineering

in Chemical Engineering with Energy and Environment

by Mostafizor Rahman

Student ID: 9548302

Supervisor: Dr. Dongda Zhang

School of Chemical Engineering and Analytical Science

The University of Manchester

May 2020

Acknowledgments

I would like to express my sincerest gratitude to my supervisor Dr. Dongda Zhang for his encouragement, guidance, patience and invaluable feedback throughout the project. I would also like to express my thanks to Max Mowbray for his support and insightful suggestions throughout every stage of the project. I am indebted to my friend and fellow research partner Gabriel Grant-Rush for his advice and support and with whom I have had the pleasure to work with in this project.

Abstract

A potential strategy to combat fossil fuel shortage and global warming is to develop renewable and sustainable methods of developing products that are conventionally derived from non-renewable sources; these products are referred to as bioproducts. The bioproduct of interest in this study is lutein which can be synthesized from microalga *Desmodesmus* Sp. Lutein possesses a wide variety of useful properties that have extensive applications in the pharmaceutical, food and cosmetic industries. However, there is a major obstacle preventing the industrialisation of the lutein production bioprocess which is determining the optimal operating conditions of the process and to subsequently keep the process within the desired operating limits by implementing a robust control scheme. To achieve the critical tasks of optimisation and control, it is necessary to construct a mathematical model that can accurately simulate the dynamic behaviour of the microalgal lutein production bioprocess. Consequently, the aim of this study is to develop such a mathematical model.

Artificial Neural Networks (ANNs) are the modelling strategy of choice, in particular, two models which fall under the umbrella term of ANNs were developed: Feedforward Neural Networks (FNNs) and Recurrent Neural Networks (RNNs). In previous studies, FNNs have already demonstrated the ability to simulate the microalgal lutein production bioprocess. However, RNNs have never been employed to simulate the bioprocess, therefore, this is a novel area of research. To ensure the accuracy of the constructed ANNs, various strategies were employed. Firstly, the experimental data was rescaled using the standardisation approach. Next, to obtain enough training data to construct the ANNs, the experimental data was artificially replicated by embedding random noise. Finally, the optimal structure of the ANNs was determined by employing a robust hyperparameter selection procedure.

The simulation accuracy of the optimised ANNs was subsequently verified by testing on unseen experimental data. It was found that both FNNs and RNNs can accurately simulate the short-term and long-term dynamic behaviour of the microalgal lutein production bioprocess. Consequently, they can be utilised for optimisation and control of the bioprocess. Furthermore, it was discovered that RNNs are the superior modelling strategy as they consistently outperformed FNNs during the testing simulations. Thus, this research improves upon previous work by developing a model that can simulate the dynamic behaviour of the lutein production bioprocess with more accuracy than FNNs. Furthermore, the construction of a mathematical model that can accurately perform optimisation and control facilitates the transition of the microalgal lutein production bioprocess from laboratory scale to industrial scale.

Table of Contents

Abbreviations

Abbreviation	Definition
ANN	Artificial Neural Network
FNN	Feedforward Neural Network
RNN	Recurrent Neural Network
GP	Gaussian Process
MSE	Mean Squared Error
MAPE	Mean Absolute Percentage Error

Nomenclature

Symbol	Definition	Units
X	Biomass Concentration	g L^{-1}
N	Nitrate Concentration	mg L^{-1}
Lu	Lutein Production	mg L^{-1}
Li	Light Intensity	µmol m^{-2} s^{-1}
F_c	Nitrate Inflow Concentration	mol L^{-1}
$Y_{i,e}$	Experimental Data	-
$Y_{i,p}$	Predicted Data	-
$\boldsymbol{W}_l$	Weight Matrix for Layer l	-
$\boldsymbol{w}_{l,j}$	Weight Vector for Neuron j in Layer l	-
$b_{l,j}$	Bias Value for Neuron j in Layer l	-
$\boldsymbol{a}_l$	Activation Function in Layer l	-
α	Learning Rate	-
$\boldsymbol{u}_{l,j}$	Hidden State Weight Vector for Neuron j in Layer l	-
$\boldsymbol{h}$	Hidden State Vector	-
t	EPOCH	-
$GL(t)$	Generalisation Loss at EPOCH t	-
$E_{opt}(t)$	Average MSE of Optimum EPOCH Value	-
$E_{va}(t)$	Average MSE after Current EPOCH t	-
β	Generalisation Loss Threshold	-
μ	Mean	-
σ	Standard Deviation	-

1. Introduction

The world is faced by an impending energy crisis which will begin not when the last barrel of oil or other non-renewable fossil fuels are extracted from the earth, but when the demand for fossil fuels far outweighs the supply capacity [1]. This crisis is expected to inflict the world as early as 2025 [1]. Furthermore, the combusting of fossil fuels to generate energy leads to climate change due to the release of large quantities of greenhouse gases such as carbon dioxide [2]. Due to these pressing issues, it is imperative to find renewable, sustainable and economically viable methods of producing fuels and products that are conventionally derived from fossil fuels and other non-renewable sources. There is not one simple solution to this complex problem, there are many potential solutions that must be researched, developed and subsequently implemented in tandem.

One promising solution to this problem is the production of biofuels and bioproducts from microalgae. Microalgae can be used to produce a variety of biofuels such as biodiesel and biohydrogen [3]. Furthermore, microalgae-based technologies have been developed that are capable of producing bioproducts such as animal feeds and nutritious food supplements; this market is forecasted to see considerable global growth [4] [5]. More specifically, in this study, lutein is the bioproduct of interest [6]. Lutein is a carotenoid pigment which is found in eukaryotic organisms such as plants, algae and fungi where it provides colour; it can also be found in prokaryotic organisms such as bacteria [7]. Lutein possesses a wide variety of useful properties that have extensive applications in the pharmaceutical, food and cosmetic industries [8]. Lutein protects the human eyes from macular degeneration which makes it valuable to the pharmaceutical industry [9]. It can also prevent the development of chronic health problems that plague the general population such coronary heart disease, diabetes, cancer and dermatological conditions [9]. Furthermore, lutein is used to provide colouring to cosmetics, foods and drugs [7]. Due to the practical utility of lutein spanning a range of industries, global demand for lutein is growing at a respectable rate. The predicted compound annual growth rate of lutein is 5.2% between 2018-2025 with a forecasted market capitalisation of $396.4 million by 2024 [9].

The successful industrialisation of the lutein production bioprocess is prevented by its high operating costs which can be attributed to the conventional source of lutein: marigold [10]. Marigold is a poor source of lutein as it possesses a small lutein content of 0.03% and this problem is amplified by its slow growth rates [11]. The combination of these undesirable properties results in bioprocesses that produce low yields of lutein and have high energy costs due to the requirement of intensive separation operations [11]. In response to these issues, extensive research has been performed on various strains of microalgae in order to find potential replacements to marigold. The result of this research was the identification of microalga *Desmodesmus* sp. which not only demonstrates a significantly higher growth rate than marigold but also possesses a lutein content of 5.05 mg/g which is almost 17x higher than marigold [11]. In addition, microalga *Desmodesmus* sp. has strong thermotolerant abilities which is an essential attribute as microalgae cultivation is best suited to locations where the water temperature regularly breaches 45 ^{o}C; thermotolerance is a requirement to sustain growth during these breaches [12] [13].

The switch from marigold to microalga *Desmodesmus* sp. has undoubtedly improved the economic viability of industrial microalgal lutein production. However, there still remains a major obstacle that must be overcome, which is the problem of determining the optimal operating conditions of the bioprocess [13]. Moreover, due to the sensitivity of the microalgal lutein production bioprocess to variations and fluctuations in operating conditions, it is vital to subsequently design a robust control scheme that can reliably keep the bioprocess within the optimal operating limits. Through the identification of the optimal operating conditions and the subsequent implementation of a control scheme, the efficiency, profitability and safety of the bioprocess can be ensured [14]. To perform the critical tasks of optimisation and control, it is necessary to construct a mathematical model that can reliably simulate the dynamic behaviour of the microalgal lutein production bioprocess [15].

Therefore, the aim of this research is to construct such a mathematical model. As explained in the initial research proposal, there are two main approaches that can be employed to construct such a mathematical model: physical modelling and machine learning which is a branch of data-driven modelling [16]. There exists a wide array of modelling techniques in the physical modelling toolbox that can be utilised to simulate the microalgal lutein production bioprocess such as the Monod Model and the Droop Model [17]. Although physical models can be employed to successively simulate the microalgal lutein production bioprocess as demonstrated in the research performed by del Rio-Chanona et al. [18], the act of constructing such a physical model is no elementary task. It is, in fact, an extremely difficult procedure particularly in the case of bioprocesses.

The construction of physical models requires a comprehensive understanding of the kinetics driving the bioprocess [13]. As explained by del Rio-Chanona et al. [18], compared to conventional chemical processes that are common in the field of Chemical Engineering, the kinetics driving a typical bioprocess are significantly more complex. Additionally, in the case of bioprocesses that require the absorption of light to function, such as those centring around the growth of microalgae in a photobioreactor (PBR), the complexity is amplified even further. This amplification in complexity is due to an occurrence known as light attenuation where there is non-uniformity in the distribution of light intensity across the PBR. This results in highly non-linear models that are functions of both the spatial and temporal domains and are thus troublesome to solve numerically. Consequently, alternative modelling strategies are required that are easier to construct and solve numerically than physical models, furthermore, these models must demonstrate comparable or superior predictive capabilities.

An alternative to physical models are machine learning models which resolve many of the issues surrounding physical models. Physical models are examples of 'white box' models which are developed based on fundamental principles such as the conservation of energy, mass and momentum [19]. Therefore, as mentioned earlier, a comprehensive understanding of the kinetics driving the bioprocess is required to construct a physical model that adequately simulates the dynamic process. On other hand, machine learning models are 'black box' models that do no not require any understanding at all of the kinetics to construct [19]. The black box itself is simply a mathematical function/model accompanied by a range of mathematical optimisation techniques. There are a variety of mathematical models that can be employed inside a black box such as simple linear functions, polynomial functions, artificial neural networks (ANNs) and gaussian processes (GPs). The sole requirement to construct a black box model is an experimental dataset that describes the operation of the bioprocess over a period of time. Using this dataset, the black box optimises the parameters of the mathematical model such that it can accurately map the inputs of the dataset to the outputs. This optimisation or 'learning' process is achieved through a variety of machine learning techniques and is implemented programmatically on a computer.

In this report, artificial neural networks (ANNs) will be the model of choice from the vast array of modelling techniques available in the machine learning toolbox. ANN is an umbrella term that encompasses many different types of neural network architectures. Some example architectures are feedforward neural networks, recurrent neural networks and convolutional neural networks [20]. In this work, the focus will be on constructing feedforward neural networks (FNNs) and recurrent neural networks (RNNs) and subsequently comparing their performance to determine which is the superior modelling strategy.

FNNs were chosen because they have been shown to have superior predictive capabilities in comparison to physical models [18]. RNNs were chosen because there is currently no work in the literature which investigates the performance of RNNs in the dynamic simulation of the lutein production bioprocess; thus, it is a completely novel area of research. Furthermore, unlike FNNs, RNNs were specifically designed to simulate sequential data such as time-series data. Consequently, it follows that RNNs should theoretically exceed the performance of ANNs in the simulation of bioprocesses which are time dependent; this will be an interesting and important question to answer from the research performed in this report. GPs are also an exciting topic of research as they have been proven to be a potential replacement to physical models [21]. Furthermore, GPs possess a major advantage over FNNs and RNNs as they can provide a measure of uncertainty for their predictions [22]. However, due to time constraints, GPs have unfortunately not been investigated in this report. Nevertheless, they are an interesting point of future research.

This report is split into 6 sections starting with the introduction and motivations behind the research which has been presented above. In Section 2, the aims of the project will be clearly defined, and a list of objectives will be formulated that outline the steps that need to be taken in order to achieve these aims. In Section 3, the theory behind ANNs will be discussed in detail. This will hopefully demystify the subject and provide the reader with a clear understanding of what ANNs are and how they operate from a general and mathematical perspective. In Section 4, the procedure for constructing and optimising data-driven models (FNNs and RNNs) will be discussed in detail. In Section 5, the optimisation and simulation results of the FNN and RNN will be presented and compared with the aim to identify the superior modelling technique. In Section 6, final conclusions will be made including recommendations for future work.

2. Objectives

The aim of this research project is to construct a mathematical model that can simulate the dynamic behaviour of the microalgal lutein production bioprocess. This mathematical model will be constructed using machine learning techniques, specifically, artificial neural networks (ANNs). ANNs comprise many different neural network architectures, therefore, the specific architectures that will be investigated in this project are feedforward neural networks (FNNs) and recurrent neural networks (RNNs). The simulation results of the optimised FNN and RNN will be compared in order to determine the superior modelling technique. The research aims have been formulated into a list of structured objectives below.

1. **To construct an FNN and an RNN capable of simulating the dynamic behaviour of the microalgal lutein production bioprocess.**
2. **To optimise the hyperparameters of the FNN and RNN by employing a comprehensive hyperparameter selection procedure for improved predictive performance.**

3. **To test the predictive capabilities of the optimised FNN and RNN through online and offline simulation.**
4. **To determine the superior modelling strategy by comparing the online and offline simulation results of the FNN and RNN.**

3. Theory of Artificial Neural Networks (ANNs)

In this section the theory behind the operation of FNNs and RNNs will be explained from both a general and mathematical perspective. This section will begin with a discussion of the theoretical aspects of FNNs followed by identifying the limitations of FNNs, finally leading into RNNs which are a direct answer to those limitations.

3.1. Feedforward Neural Networks (FNNs) – General Theory

The human brain consists of an interconnected web of billions of neurons. Each individual neuron is connected to thousands of neighbouring neurons resulting in an expansive neural network. All neurons contain an axon and dendrites; the dendrites receive signals while the axon transmits signals [23]. Neurons are connected by their axons and dendrites; these connections are referred to as synapses [23]. Simplified neural networks can be simulated on a computer; these simulations are called ANNs. The idea behind ANNs is to construct a computer program that can process information, learn and think like a human. A schematic of the typical structure of an ANN is shown in Figure 1 below.

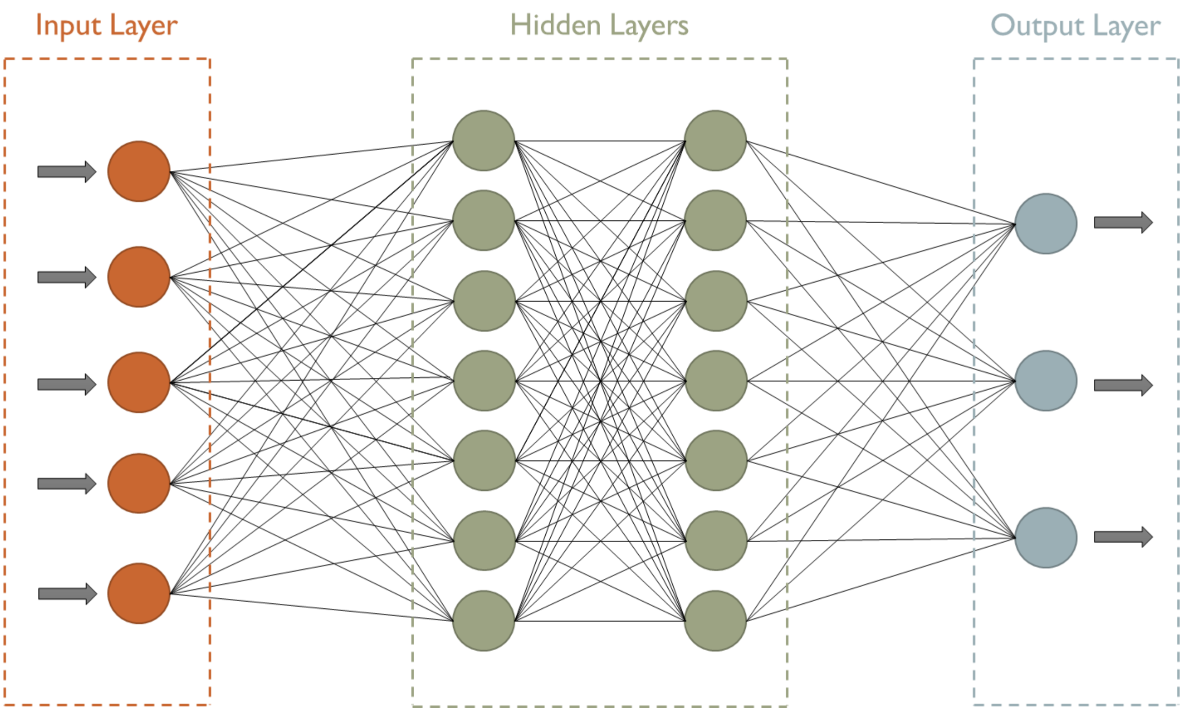


Figure 1. Feedforward ANN Structure.

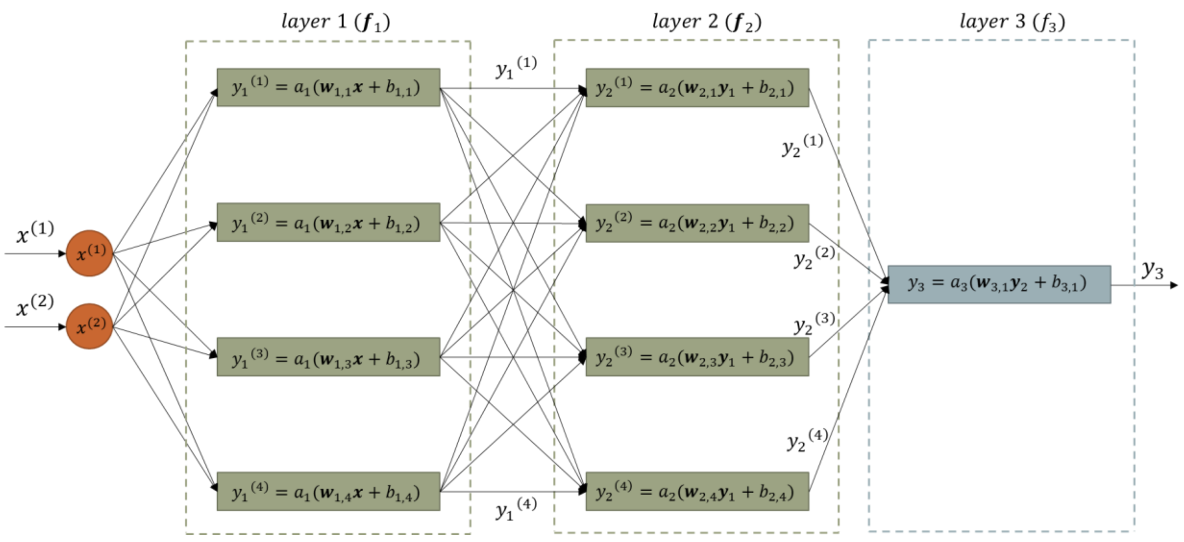
As can be seen in Figure 1 above, ANNs consist of nodes (neurons) which are connected by lines (synapses). The nodes are separated into 3 sections: an input layer, hidden layers and an output layer. The input layer contains the input nodes which represent input values to the network; these input values could be any real number and are commonly referred to as input features. These real numbers can represent a wide range of things such as pixels in images, letters in a string of text and state variables in a bioprocess. The hidden layers consist of hidden neurons that receive the input values and transform them by performing a series of mathematical operations. These processed input values then leave the hidden layer and are passed to the output layer. The output layer produces the output predictions from the network by performing further mathematical transformations.

If the input values to an ANN represent words in the English language, then the output values from the network could represent translated words in another language such as French. In this case the ANN is a translation system. Another example more pertinent to the topic of research in this paper is an ANN that simulates a bioprocess. In this case, the ANN would receive state variables as inputs that describe the state of the system at the current point in time *t*. The output predictions from the network would then be the state variables at the next point in time *t+1*.

FNNs are a subcategory of ANNs in which information propagates forwards through the network - only touching a given node once - hence the name ‘feedforward’. There exist other variations of ANNs where information can loop back through the network from a succeeding node to a prior node, an example of which are RNNs. RNNs will be discussed in more detail in sections 3.3 and 3.4 of this report.

3.2. Feedforward Neural Networks (FNNs) – Mathematical Theory

In this section the mathematical operations that occur under the hood of a typical FNN will be presented and explained.



layer 1 (f_1)
layer 2 (f_2)
layer 3 (f_3)
$x^{(1)}$
$x^{(2)}$
$y_1^{(1)} = a_1(w_{1,1}x + b_{1,1})$
$y_1^{(2)} = a_1(w_{1,2}x + b_{1,2})$
$y_1^{(3)} = a_1(w_{1,3}x + b_{1,3})$
$y_1^{(4)} = a_1(w_{1,4}x + b_{1,4})$
$y_1^{(1)}$
$y_1^{(4)}$
$y_2^{(1)} = a_2(w_{2,1}y_1 + b_{2,1})$
$y_2^{(2)} = a_2(w_{2,2}y_1 + b_{2,2})$
$y_2^{(3)} = a_2(w_{2,3}y_1 + b_{2,3})$
$y_2^{(4)} = a_2(w_{2,4}y_1 + b_{2,4})$
$y_2^{(1)}$
$y_2^{(2)}$
$y_2^{(3)}$
$y_2^{(4)}$
$y_3 = a_3(w_{3,1}y_2 + b_{3,1})$
y_3

Figure 2. FNN Mathematical Structure [24].

Figure 2 above shows a schematic of a simple FNN consisting of an input layer, 2 hidden layers (layer 1 and layer 2) and an output layer (layer 3). The FNN receives an input vector, $\boldsymbol{x}$, consisting of two input features $x^{(1)}$ and $x^{(2)}$ and the network outputs just one value, y_3. Each hidden layer consists of 4 hidden nodes represented by rectangles. Each hidden node possesses a vector of weights, $\boldsymbol{w_{l,j}}$, and a single scalar value, $b_{l,j}$, referred to as the bias. The weight vector in each neuron, $\boldsymbol{w_{l,j}}$, contains a scalar weight value, $w_{l,j,k}$, for each input value, $x^{(k)}$, that it receives from the input layer. Similarly, the weight vectors for each hidden node in the second layer contain a number of weights equivalent to the number of hidden nodes in the first hidden layer. Note that the subscript *l* denotes the number of hidden layers, the subscript *j* denotes the number of neurons in each hidden layer and the subscript *k* denotes the number of weights in each neuron in each hidden layer. Focussing on the first node in the first hidden layer, the node applies a linear transformation to the input vector, $\boldsymbol{x}$, resulting in a single scalar value, *s*, as follows [24].

$$s = \boldsymbol{w}_{1,1}\boldsymbol{x} + b_{1,1} = \mathrm{w}_{1,1,1}x^{(1)} + \mathrm{w}_{1,1,2}x^{(2)} + b_{1,1} \tag{1}$$

An activation function, a_1, is subsequently applied to the scalar value, *s*, yielding the output from the hidden node, ${y_1}^{(1)}$, which is simply a scalar value:

$${y_1}^{(1)} = a_1(s) \tag{2}$$

An activation function is required to introduce non-linearity into the FNN which then allows the FNN to model non-linear data [24]. Typical activation functions are the sigmoid and hyperbolic tangent function:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4}$$

Equations (1) and (2) can be combined into the following equation as seen inside the rectangles in Figure 2 above [24].

$${y_1}^{(1)} = a_1(\boldsymbol{w}_{1,1}\boldsymbol{x} + b_{1,1}) \tag{5}$$

The output value from the node then becomes an input value to each of the nodes in the following layer. The mathematical operation shown in Equation (5) above occurs in every node in both hidden layers 1 and 2. However, the activation function in the output layer can differ from the hidden layers and in many cases the activation function in the output layer is omitted, thus, only a linear transformation is performed.

Each hidden layer can be represented as a single vector function, $\boldsymbol{f}$, by using a combination of vector and matrix notation as follows [24].

$$\boldsymbol{f}_l(\boldsymbol{x}) = \boldsymbol{a}_l(\boldsymbol{W}_l\boldsymbol{x} + \boldsymbol{b}_l) \tag{6}$$

where $\boldsymbol{f}_l$ is a vector containing the outputs from each node in layer l, $\boldsymbol{x}$ is the input vector containing all of the input features to the network, $\boldsymbol{a}_l$ is the activation function in layer l, $\boldsymbol{W}_l$ is a matrix containing the weight vectors for each node in layer l and $\boldsymbol{b}_l$ is a vector containing the biases for each node in layer l.

Therefore, it follows that an FNN, such as that presented in Figure 2, is simply a nested non-linear function [24].

$$y_3 = f_3\left(\boldsymbol{f}_2(\boldsymbol{f}_1(\boldsymbol{x}))\right) \tag{7}$$

Note that since the FNN in Figure 2 only outputs one value, the output layer is represented by a scalar function, f_3, as opposed to a vector function like the 2 hidden layers which produce multiple outputs.

Using the predicted outputs from the network, $y_{i,p}$, an average loss can be calculated such as the Mean Squared Error (MSE) by comparing the predicted values, $y_{i,p}$, to the real/experimental values, $y_{i,\mathrm{e}}$, as follows [24].

$$l = \frac{1}{N}\sum_{i}^{N}(y_{i,p} - y_{i,\mathrm{e}})^2 \tag{8}$$

where *N* is the total number of predicted datapoints and *i* represents a single datapoint. Note that $y_{i,\mathrm{e}}$ is simply the nested function that is shown in Equation 7 above, or in other words it is the FNN. The experimental values are referred to as output labels which the network must attempt to predict. The loss function quantifies the accuracy of these predictions.

Using this loss function, the weights and biases are adjusted in the network in order to minimise the loss. This adjustment process is achieved via an algorithm known as backpropagation with gradient descent [24]. Backpropagation essentially enables the calculation of the gradient of the loss function with respect to each of the individual weights and biases (parameters) in a neural network using the chain rule [24]. In order to minimise the loss, the parameters in the network must be iteratively adjusted in the direction that reduces the magnitude of the loss function. The gradient reveals this direction to us; therefore, it is pivotal in the optimisation process.

Thus, the backpropagation algorithm calculates the partial derivative of the loss function with respect to each of the weights and biases in the network. The weights and biases are then adjusted in a direction negative to the partial derivative multiplied by a learning rate, α, as denoted in equations 9 and 10 below respectively [24].

$$w^{k+1} \leftarrow w^{k} - \alpha \frac{\partial l}{\partial w} \tag{9}$$

$$b^{k+1} \leftarrow b^{k} - \alpha \frac{\partial l}{\partial b} \tag{10}$$

This optimisation procedure is typically terminated when the reduction in the loss function between iterations is lower than a specified tolerance value. Through this optimisation process, the network 'learns' to simulate a given phenomenon such as a dynamic bioprocess. To summarise, the backpropagation optimisation procedure fits the nested function in equation 7 (which is the FNN) to a dataset by adjusting its parameters in order to minimise the loss function in Equation 8. Thus, constructing and subsequently training an FNN can be thought of as a non-linear regression process.

The learning rate, α, is a hyperparameter that controls the magnitude of the weight update per iteration. A higher learning rate means that the weights will be adjusted to a larger extent per iteration, whereas a smaller learning rate means that the extent of the weight adjustment will be smaller. Hyperparameters are values that are not adjusted by the backpropagation optimisation procedure; therefore, it is in the hands of the machine learning engineer to choose a suitable value. Machine learning engineers typically employ a separate optimisation procedure to tune the hyperparameters. A comprehensive hyperparameter optimisation procedure will be presented in section 4 of this report.

There exist four more hyperparameters that are relevant to the research performed in this report. The first two hyperparameters are the number of hidden layers and the choice of activation function in each layer. The third hyperparameter is known as the number of EPOCHS. An EPOCH is simply one complete pass through the experimental dataset during the training phase. The final hyperparameter is the batch size. The batch size is the number of datapoints processed by the neural network before the loss is calculated using Equation 8 and subsequently backpropagated through the network to update the parameters. A batch size equal to one means that the weights are updated after every datapoint in the training dataset. When the batch size is equal to 1, the training and optimisation procedure is called

Stochastic Gradient Descent [25]. On the other hand, when the batch size is equal to the size of the entire dataset, it is referred to as Batch Gradient Descent [25]. Finally, when the batch size is more than one datapoint but less than the total number of datapoints in the training set, it is referred to as mini-batch gradient descent [25]. Again, the hyperparameter optimisation framework will be presented in section 4 of this report.

3.3. Recurrent Neural Networks (RNNs) – General Theory

The drawback of FNNs is that they possess no perception of order in time. FNNs only operate on the present input that they are exposed to, even if that input belongs to a larger sequence of data, such temporal data from the dynamic operation of a bioprocess [26]. It would be useful if the neural network could use its memory of previous datapoints in a sequence to influence its future predictions. This would, in theory, improve its predictive performance as sequential data from a bioprocess are inevitably linked between timesteps. This can be achieved by using recurrent neural networks (RNNs).

Unlike FNNs, RNNs contain loops which essentially allows information to persist between timesteps [26]. Consequently, through the use of loops, information can be passed from the network at a previous point in time to the network at a future point in time [27]. As shown in Figure 3 below, if we unfold the loop, an RNN can be visualised as numerous copies of the same FNN in successive timesteps with each FNN passing a message to itself in a future timestep [27].

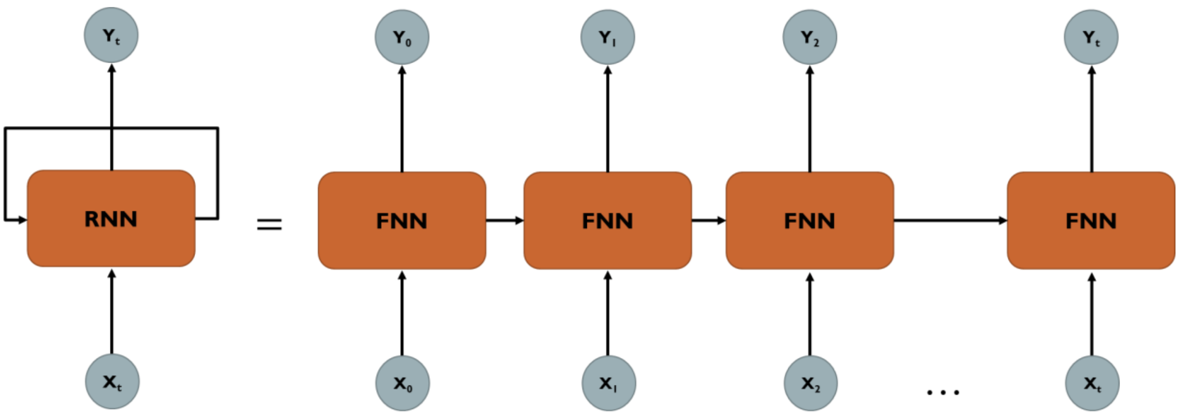


Figure 3. An RNN can be thought of as sequential copies of the same FNN each passing a message to its future self.

Focussing on the first FNN in the unfolded loop, this is the FNN at the first timestep, t_0. It receives its first input, $\boldsymbol{X_o}$, and produces an output prediction, $\boldsymbol{Y_o}$, for the next timestep, t_1. However, the network also possesses a hidden state, $\boldsymbol{H}$, which is essentially its memory of all the previous sequential data [26]. This memory is passed to the network at the next timestep along with the input data at the next timestep; this data will be used to make the prediction at the following timestep t_2.

3.4. Recurrent Neural Networks (RNNs) – Mathematical Theory

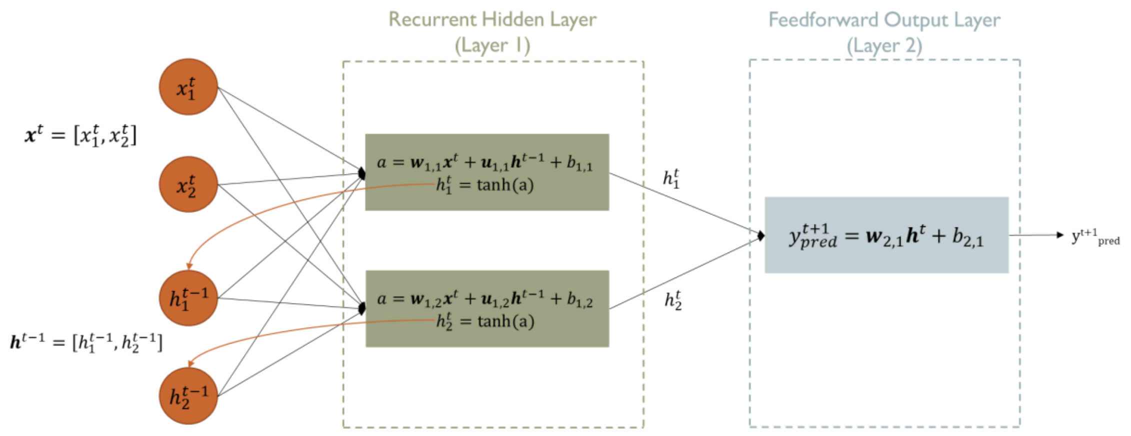

Recurrent Hidden Layer (Layer 1)
Feedforward Output Layer (Layer 2)
$\boldsymbol{x}^t = [x_1^t, x_2^t]$
x_1^t
x_2^t
h_1^{t-1}
h_2^{t-1}
$\boldsymbol{h}^{t-1} = [h_1^{t-1}, h_2^{t-1}]$
$a = \boldsymbol{w}_{1,1}\boldsymbol{x}^t + \boldsymbol{u}_{1,1}\boldsymbol{h}^{t-1} + b_{1,1}$
$h_1^t = \tanh(a)$
$a = \boldsymbol{w}_{1,2}\boldsymbol{x}^t + \boldsymbol{u}_{1,2}\boldsymbol{h}^{t-1} + b_{1,2}$
$h_2^t = \tanh(a)$
h_1^t
h_2^t
$y_{pred}^{t+1} = \boldsymbol{w}_{2,1}\boldsymbol{h}^t + b_{2,1}$
y^{t+1}_{pred}

Figure 4. RNN Mathematical Structure.

A schematic representation of an RNN consisting of one recurrent hidden layer followed by a standard feedforward output layer is shown in Figure 4 above. The hidden layer consists of two neurons and the major difference between a standard FNN is that each neuron in the hidden layer contains a hidden state, h. The hidden state represents the memory of the neuron and is simply a scalar value. Focussing on the first neuron in the hidden layer, the neuron sequentially receives feature vectors consisting of two features as inputs: x_1^t and x_2^t. It also receives a hidden state vector containing the hidden states of the neurons in the same layer at the previous timestep: h_1^{t-1} and h_2^{t-1}.

The neuron firstly performs a linear transformation on the feature vector and the hidden state vector. The output of that linear transformation, a, is then passed to an activation function which is conventionally the hyperbolic tangent function as shown in Equation 4 above. The result of the activation function is the updated hidden state for the neuron: h^t_1. The updated hidden state from both neurons are then passed to the output neuron which performs a further linear transformation yielding the output prediction at the next timestep y^{t+1}_{pred}. The updated hidden states also loop back to the input layer to be used as inputs alongside the feature vector at the next timestep.

In practice, an RNN operates much like the unfolded loop in Figure 3. The RNN receives the entire sequence of temporal data and calculates the predictions at each timestep almost simultaneously; there is one FNN in the unfolded loop for every datapoint in the input sequence. For each prediction in a given timestep the FNN will utilise the hidden state from the previous timestep. However, the first FNN in the sequence does not receive a hidden state as there is no previous FNN in the sequence, thus, the hidden state does not exist. This means that the first FNN receives a hidden state filled with zeroes. Note that the number of FNNs in the unfolded loop is referred to as the sequence length.

4. Procedure for Data-Driven Model Construction

In this section, the procedure for constructing both FNNs and RNNs will be presented and discussed. The procedure for constructing FNNs and RNNs is for the most part analogous. Thus, it can be assumed that the information presented in this section applies to both FNNs and RNNs unless explicitly stated otherwise.

4.1. Software

The programming language of choice to construct the ANNs is Python. Python was chosen as the ecosystem is supported by a wide range of machine learning and data manipulation libraries. The main libraries that were utilised in this project are PyTorch, NumPy, Pandas and Scikit-learn. PyTorch is a machine learning library that provides all of the functions necessary to construct ANNs such as activation functions, linear functions, loss functions and optimisation algorithms. NumPy is a library that allows for the easy manipulation and construction of large multi-dimensional matrices which are essential to represent the vast number of parameters present in a typical ANN. Pandas and Scikit-learn provide functions that allow for the easy loading, manipulation and pre-processing of large amounts of data.

4.2. Experimental Data

As mentioned earlier in this report, all that is required to construct a machine learning (black box) model is an experimental dataset that describes the operation of the bioprocess that one wishes to model. The dataset employed in this project consists of data from 8 experiments. The experimental data describes the growth of microalga *Desmodesmus* Sp. in a photobioreactor through the evolution of three key state variables: biomass concentration (X), nitrate concentration (N) and lutein production (Lu) over an operational period of 144 hours with measurements taken at 12 hour intervals. Each experiment begins with an initial biomass concentration of approximately 0.077 g/L, an initial nitrate concentration of either approximately 780 mg/L or 2700 mg/L, and an initial lutein concentration of 0 mg/L. For each experiment the incident light intensity (Li) is constant throughout the entire operational period and can take on values between 150-600 µmol m^{-2} s^{-1}. Furthermore, at the 60th hour, a continuous flow of nitrate is turned on; the resulting nitrate inflow rate (F_R) is 3 mL/h in 7 out of 8 experiments and 60 mL/h in just one experiment. Finally, each experiment has a nitrate inflow concentration (F_C) of either 0.1 or 0.5 M which is constant throughout the entire operational period.

When constructing ANNs, it is typical to have a training dataset and a testing dataset. The training dataset is used to train the network by tuning its parameters via gradient descent and backpropagation. It is also used to optimise the hyperparameters of the network via an optimisation procedure that will be presented in Section 4.6 of this report. The predictive capabilities of the trained and optimised network are subsequently evaluated on the testing dataset which the network has never been directly exposed to. In this project, 6 out of 8 of the experimental datasets were used for training and the remaining 2 datasets were used for testing. The train-test split is shown in Table 1 below.

Table 1. Table Showing the Training and Testing Datasets.

Operating Conditions	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Exp 6	Test 1	Test 2
Initial Biomass Conc (g/L)	0.077	0.077	0.077	0.077	0.078	0.077	0.077	0.077
Initial Nitrate Conc (mg/L)	767.76	764.27	2700.64	783.20	784.40	783.30	2649.39	783.50
Nitrate Inflow Rate (mL/h)	3.0	60.0	3.0	3.0	3.0	3.0	3.0	3.0
Nitrate Inflow Conc (M)	0.5	0.5	0.5	0.1	0.1	0.1	0.5	0.1
Light Intensity (μmol m^{-2} s^{-1})	150	300	600	150	480	600	480	300

The testing datasets were chosen such that the interpolative capabilities of the network could be tested. During interpolation, the network uses known operating conditions to estimate an unknown value with operating conditions that lie in-between the known operating conditions. On the other hand, in extrapolation, the network uses its knowledge of the current operating conditions to predict a completely unknown value that lies outside of the range of known operating conditions.

The test sets will be referred to as Test Set 1 and Test Set 2; the operating conditions of these test sets are shown in Table 1 above. The combination of operating conditions in both test sets is unique and the network must interpolate between known combinations of operating conditions to simulate these test sets, thus, verifying the predictive capabilities of the ANNs.

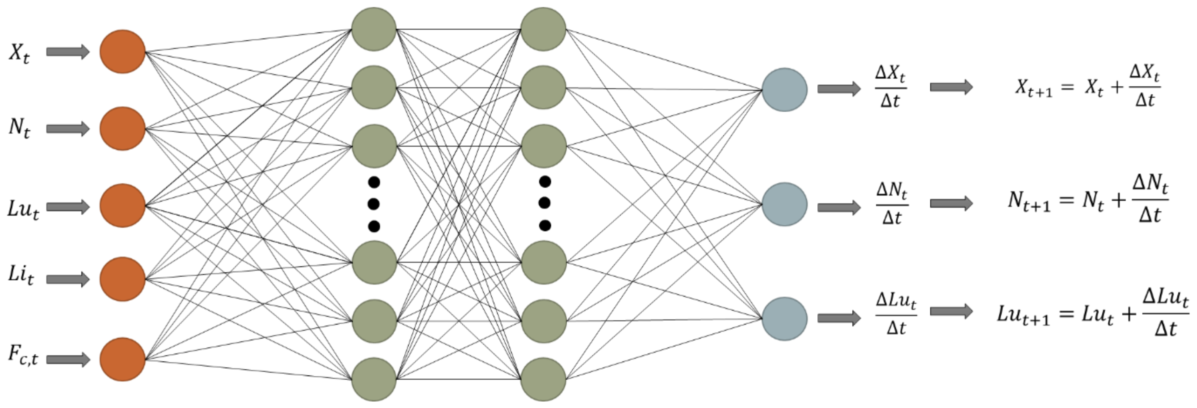
4.3. Feature Selection

Both the FNN and the RNN receive 5 inputs features:

- Biomass Concentration (X)
- Nitrate Concentration (N)
- Lutein Production (Lu)
- Light Intensity (Li)
- Nitrate Inflow Concentration (F_C)

The biomass concentration, nitrate concentration and lutein production are state variables that are measured every 12 hours. Light intensity and nitrate inflow concentration are operating conditions that remain constant during experimentation. The nitrate inflow rate was not provided as an input feature to the network as it is the same for 5 out of 6 of the experiments in the training dataset. Furthermore, both experiments in the testing dataset possess the same nitrate inflow rate. Moreover, the switching-on of the nitrate inflow rate at the 60th hour is conveyed by the nitrate concentration; before the 60th hour the nitrate concentration decreases and after the 60th hour the nitrate concentration begins to increase. Consequently, providing the nitrate inflow rate as an input feature would add unnecessary complexity into the model and remove weighting from the other more important input features.

In the case of the FNN, it operates by receiving 5 input features at time *t*. Using these 5 input features, the network predicts the value of the 3 state variables at the next time step, *t+1*. However, the network does not predict the value of the state variables at the next time step directly, instead, it predicts the rate of change of the state variables. Using the rates of change, the value of the state variables at the next timestep are calculated by adding the rates of change to the value of the state variables at the previous timestep. This procedure is demonstrated schematically in Figure 5 below.



X_t
N_t
Lu_t
Li_t
$F_{c,t}$
$\frac{\Delta X_t}{\Delta t}$
$X_{t+1} = X_t + \frac{\Delta X_t}{\Delta t}$
$\frac{\Delta N_t}{\Delta t}$
$N_{t+1} = N_t + \frac{\Delta N_t}{\Delta t}$
$\frac{\Delta Lu_t}{\Delta t}$
$Lu_{t+1} = Lu_t + \frac{\Delta Lu_t}{\Delta t}$

Figure 5. Input-Output Structure of FNNs.

As proven by del Rio-Chanona et al. [28], using rates of change to predict the value of the state variables at the next time step as opposed to predicting the value of the state variables directly is preferred as it results in superior predictive performance. Note that for RNNs there would be additional inputs representing the hidden states for each neuron in the first hidden layer as explained in Section 3.4. Furthermore, the second hidden layer would receive the hidden states of the neurons in the same layer at the previous timestep as well as the outputs from the neurons in the first hidden layer. However, the outputs from the RNN would also be the rates of change of the three state variables, just like the FNN. Thus, to ensure that the networks predict rates of change of the state variables as opposed to the state variables directly, the output labels must also be the rates of change of the three state variables. These output labels are calculated for each input vector by subtracting the value of the state variables at the current timestep, t, from the state variables at the subsequent timestep *t+1*.

4.4. Data Pre-Processing

As mentioned previously, a data-driven model has no knowledge of the underlying kinetics driving a bioprocess; it is a black-box, it has no physical or empirical knowledge. Therefore, a data-driven model relies solely on the dataset that it is presented with to build a mathematical model that accurately simulates the bioprocess in question. Therefore, it is critical to ensure that the dataset is processed correctly before it is used to train the model. A poorly processed dataset can result in unsuccessful training and a highly inaccurate model. Consequently, a robust data pre-processing procedure will be presented in this section to ensure successful training of the ANNs.

4.4.1. Data Rescaling

Referring to Table 1 above, it is clear that the magnitude of the input features varies massively. For example, the nitrate concentration values are hundreds to thousands of times larger than the biomass concentration, lutein production, light intensity and nitrate inflow concentration values. This is an issue as it can introduce imbalances in the weight updates during the training process [24]. As explained in Section 3.2, the partial derivative of the loss function with respect to each of the parameters in the network is calculated during the training process. Since the magnitude of the nitrate concentration feature is so much larger than the other input features, it will dominate the weight updates [24]. Thus, less emphasis will be placed on the other features by the optimisation algorithm during the training process even if those features have a significant effect on the operation of the bioprocess. Consequently, it is necessary to scale the input features such that they have similar magnitudes. This will improve the quality and speed of the training process.

There are two main methods that can be employed to scale the input features: normalisation and standardisation. In normalisation, the actual range of an input feature is scaled into a smaller range [24]. For example, a feature that has a range of [250, 900] is condensed into a smaller range of [-1, 1]. The normalisation transformation is represented mathematically in Equation 11 below, this transformation is applied to every datapoint for a given feature [24].

$$\bar{x}^{j} = \frac{x^{(j)} - min^{(j)}}{\max^{(j)} - min^{(j)}} \tag{11}$$

where $\min^{(j)}$ and $\max^{(j)}$ are the minimum and maximum value of feature j in the dataset, $x^{(j)}$ is the unscaled value and $\bar{x}^{j}$ is the resulting normalised value. Standardisation is the scaling of a feature in a dataset such that the feature set possesses the properties of a standard normal distribution or, in other words, a mean, μ, of 0 and a standard deviation, σ, of 1 [24]. The standardisation transformation is represented mathematically in Equation 12 below.

$$\hat{x}^{(j)} = \frac{x^{(j)} - \mu^{(j)}}{\sigma^{(j)}} \tag{12}$$

Ultimately, standardisation was chosen as the scaling method for a variety of reasons. Firstly, it was found in a previous study that standardisation performed better than normalisation when modelling the lutein production bioprocess [14]. Furthermore, normalisation demonstrates difficulties with features that have outliers i.e. datapoints that are much larger or smaller than the rest of the datapoints in the range [14]. In this scenario, normalisation will compress the datapoints in an effort to include the outliers, this will cause the majority of the scaled datapoints to be very close together, essentially misrepresenting the original dataset [14]. This misrepresentation will reduce the quality of the training and the accuracy of the resulting ANN. The fact that normalisation is prone to compressing the datapoints in order to maintain outliers can cause problems with certain activation functions such as the sigmoid activation function [29]. Sigmoid activation functions demonstrate deteriorated learning with datasets that contain many outliers, which can be the case with normalised datasets. This issue is very relevant to this project as the activation function utilised in the hidden layers for the FNN is a sigmoid activation function (which will be discussed in more detail in Section 4.8.1).

4.4.2. Data Replication

Currently, there are 6 experiments in the training dataset, each containing 13 datapoints. However, the last datapoint in each experiment cannot be used as an input to the network as there is no subsequent value to predict, hence, there are effectively 12 datapoints per experiment. Thus, there are a total of 72 datapoints in the training dataset which is much too small to adequately train an ANN which consists of hundreds of parameters and therefore requires thousands of datapoints for effective training; significantly more data needs to be incorporated into the training dataset [30]. Small datasets are troublesome as they will most likely lead to overfitting which results in a model that has poor predictive capabilities; overfitting will be explained in more detail in the following Section. It is not practically feasible to perform more experiments in the laboratory as hundreds of more experiments will need to be conducted to obtain enough data which take 2-3 weeks per experiment [28].

A solution to this lack of data is to artificially replicate the training dataset 50 times by embedding 3% random noise and an additional 50 times with 5% random noise [14]. This is theoretically feasible as there is a level of uncertainty caused by the stochastic nature of biological systems which can be reflected in the magnitude of the random noise values [14]. This replication process results in 7272 datapoints which is 100 times larger than the original un-replicated dataset and is more than sufficient to adequately train an FNN. Furthermore, 7272 datapoints translates into 606 sequences of data containing 12 datapoints each (i.e. 606 individual experiments) for the RNN which is again, more than sufficient to successfully train an RNN. To prevent confusion, it is important to note that an FNN treats each input individually, therefore it receives one datapoint at a time as it does not make connections between past and current datapoints. On the other hand, an RNN receives datapoints in sequences of 12 as it utilises it knowledge of previous datapoints in a sequence to influence its future predictions as explained in Section 4.

4.5. Parameter Optimisation

The loss function used to quantify the performance of the network during the training process was the Mean Squared Error (MSE) as shown in Equation 8 above. The adaptive moment (ADAM) optimiser is chosen to update the parameters in the network via stochastic gradient descent and backpropagation. The ADAM optimiser has a range of benefits over other common optimisers such as AdaGrad and RMSProp [31]. One of the most significant advantages is that it is more computationally efficient due to lower memory requirements which will help to reduce training times significantly [31]. Moreover, it is designed to handle optimisation problems with large datasets and large numbers of parameters [31].

4.6. Hyperparameter Selection

In this section, a comprehensive procedure for selecting the optimal values of the following hyperparameters will be presented:

- Number of Hidden Layers
- Number of Hidden Neurons and EPOCHS
- Learning Rate
- Batch Size

A comprehensive hyperparameter selection procedure is essential to avoid the common pitfalls that plague ANNs and to thus construct a model that can accurately model the microalgal lutein production bioprocess. One of the major pitfalls that must be avoided during the construction and training of ANNs are the problems of overfitting and underfitting. Overfitting is a phenomenon where the ANN simulates the training data extremely well, but it fails to accurately simulate the testing data [24]. Overfitting occurs when the model fits too closely to the training data; it essentially misses the true trend by focussing on the noise present in the data [24]. On the other hand, underfitting is when the model fails to simulate the data that it was trained with, which also translates to an inability to simulate the testing datasets [24].

4.6.1. Selection of Number of Hidden Layers

The number of hidden layers considered are 1, 2 and 3 hidden layers for both the FNN and the RNN. For each hidden layer value, an FNN and RNN will be constructed and the hyperparameters optimised using the procedure laid out in Sections 4.6.2 to 4.6.4 below resulting in 6 ANNs; 3 FNNs and 3 RNNs. The performance of the optimised FNNs and RNNs will be compared by validation on the testing dataset to come to a decision on the optimal number of hidden layers for each ANN architecture. The performance of the optimised FNN and RNN will finally be compared with the ultimate aim being to determine the superior modelling strategy.

The activation function utilised in the hidden layers of the FNN will be the sigmoid activation function. The sigmoid function is chosen as it is a simple function that approximately captures the non-linearity in the experimental data [32]. Furthermore, the sigmoid function is differentiable which is essential for gradient descent [32]. For the RNN the activation function employed in the hidden layers will be the hyperbolic tangent function. The hyperbolic tangent function is chosen for the same reasons as the sigmoid function. However, it is preferred over the sigmoid function in certain cases as it provides a greater rate of learning during the training process; this will be explained in more detail in Section 5.1.2 of this report [32]. For both the FNN and the RNN, the output layer will be linear, thus, there will be no additional activation function in the output layers.

4.6.2. Selection of Number of Hidden Neurons and EPOCHS

The number of hidden neurons is an important hyperparameter that can cause overfitting if tuned incorrectly. The main causes of overfitting are a dataset that is too small in comparison to the number of features fed to the network and a model that is 'too complex' [24]. In the case of the dataset utilised in this project, the first cause has already been addressed through the data replication process. Moving on to the second cause, the term 'too complex' is quite an abstract term and requires some explanation. In the context of ANNs, complexity manifests itself in the form of parameters. Too many parameters implies that the network is too complex and too few parameters implies that the network is too simple which is also an issue as simplicity can lead to underfitting [24]. The number of parameters is directly related to the number of hidden neurons, as each hidden neuron possesses a range of parameters (weights and biases) proportional in number to the number of inputs that the neuron receives. Thus, to prevent overfitting and underfitting, the optimisation procedure needs to strike a good balance between too few and too many neurons. However, even if a good balance has been determined, overfitting and underfitting can still occur if the number of EPOCHS is tuned incorrectly. Too many EPOCHS can lead to overfitting whilst too few EPOCHS can lead to underfitting. Consequently, the number of hidden neurons and EPOCHS must be optimised simultaneously.

In this project, the optimisation framework chosen to select the number of hidden neurons and EPOCHS is Grid Search with Stopping Conditions [33].

Grid Search with Stopping Conditions Optimisation Procedure

- In this optimisation procedure, the number of hidden neurons is varied within a pre-determined range and the number of EPOCHS is determined using a stopping condition [33] [34].
- The number of neurons considered are 2, 4, 8, 12, 16 & 20.
- For networks with more than one hidden layer, unsymmetrical hidden neuron configurations are considered.
- For each hidden neuron value, the network is trained for a maximum of 500 EPOCHS. However, the training process can stop at any point before the limit of 500 EPOCHS if the Generalisation Loss, *GL(t)*, between EPOCHS is greater than a specified value, β [33]. The network is then tested yielding an average MSE value over all of the predictions made on the test set.
- The hidden neuron and EPOCH configuration with the lowest average MSE is chosen as the optimal hyperparameter configuration.

The stopping condition is defined by the generalisation loss, *GL(t)*, at the tth EPOCH as follows [33] :

$$stop\ after\ the\ first\ EPOCH\ t\ with\ GL(t) > \ \beta$$

GL(t) is defined in Equation 13 below [33].

$$GL(t) = 100 \times \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1\right) \tag{13}$$

The value for β is set to 1%. *E* refers to the objective function which is the Mean Squared Error (MSE). $E_{va}(t)$ is the average MSE for all predictions made on the validation set after the current EPOCH, *t*. $E_{opt}(t)$ is the lowest average MSE for all predictions made on the validation set up to the current EPOCH, *t*. After the optimisation process is terminated by the stopping condition above, the EPOCH at which $E_{opt}(t)$ occurred is taken as the optimum number of EPOCHS. In the case that the stopping condition is never activated, the maximum number of EPOCHS is set to 500, after which the optimisation process is automatically terminated. In simple words, $GL(t)$ is the increase in prediction error between EPOCHS.

Thus, when the increase in prediction error between EPOCHS is greater than 1%, the training process is terminated as it is assumed that overfitting is occurring. Finally, the EPOCH at which $E_{opt}(t)$ occurred is taken as the optimal number of EPOCHS.

Two other optimisation approaches were also considered: Grid Search without Stopping Conditions (Unmodified Grid Search) and Grid Search with K-Fold Cross Validation (K-Fold). In the Unmodified Grid Search approach, the number of EPOCHS is varied inside a predetermined range instead of using a stopping condition [34]. The number of EPOCHS considered are 15, 30, 50, 100, 150, 200, 300, 400 & 500 and the number of hidden neurons considered are 4, 8, 12, 16, & 20. For each hidden neuron value, the network is trained at each value of EPOCHS and subsequently tested yielding an average MSE value over all of the predictions made on the test set. Thus, a total of 1745 EPOCHS of training are completed for each hidden neuron value (sum of EPOCH values). In the K-Fold approach, the training dataset is split into 6 folds, with each fold containing a unique combination of 5 out of 6 experiments [35]. The ranges for the hidden neurons and EPOCHS are the same as those used in the Unmodified Grid Search approach. For each hidden neuron and EPOCH combination, the network is trained using the first fold and then tested using the omitted experiment. The next fold is then used for training and the omitted experiment used for testing. This procedure is repeated until all of the folds have been used for training. An average error value across all folds is then calculated.

In practice, it was found that the stopping condition approach is much less computationally intensive than the Unmodified Grid Search approach and the K-Fold approach. This is because, for each hidden neuron value in the range, a maximum of only 500 EPOCHS of training are undertaken as opposed to a compulsory 1745 EPOCHS in the Unmodified Grid Search approach. Furthermore, in the K-Fold approach, 6 times more training EPOCHS must be undertaken for each value in the range of hidden neurons in comparison to the Unmodified Grid Search approach; 10,470 EPOCHS compared to just 1745 EPOCHS. This is because, for each hidden neuron and EPOCH combination, the training procedure must be repeated 6 times, once per fold. Additionally, in the stopping condition approach, the optimal number of EPOCHS occurs well before the limit of 500, further decreasing the computational intensity. Due to the lower computational intensity, Grid Search with Stopping Conditions was chosen as the preferred hyperparameter optimisation strategy.

4.6.3. Selection of Learning Rate

After optimising the number of hidden neurons and EPOCHS, the learning rate is optimised. As explained in Section 3.2, the learning rate defines the size of the weight updates during backpropagation. The larger the learning rate, the faster the model will learn [36]. However, the drawback of a large learning rate is that the model may arrive at a sub-optimal solution as the size of the weight adjustments results in overshooting the minimum point, thus, the network effectively oscillates around the optimum point; it never converges [36]. On the other hand, a smaller learning rate results in a model that takes longer to train (more EPOCHS) [36]. However, the advantage of a smaller learning rate is that there is a higher chance that the network will converge, thus, a higher chance that a global optimum solution will be obtained [36]. Consequently, the choice of learning rate is a trade-off between the training speed and the optimality of the solution.

The learning rate is optimised using a grid search approach where the learning rate is varied in the range 0.0001-1, which can be divided further into the following sub-ranges: 0.0001-0.0009; 0.001-0.009; 0.01-0.09; 0.1-1.0. Within each of the first three sub-ranges, the network is trained and tested at 9 equidistant points and in the last sub-range it is trained at 10 equidistant points. This results in an average MSE value at each point. The learning rate which yields the lowest average MSE value is taken as the optimum value.

4.6.4. Selection of Batch Size

After optimising the learning rate, the last hyperparameter that must be optimised is the batch size. As explained in Section 3.2, the batch size denotes the number of datapoints that are processed by the network before the loss function is calculated and the weights subsequently updated by the optimisation algorithm. The batch size effects the speed in which training occurs. A larger batch size results in faster training (lower amount of time per EPOCH) as the loss function is calculated less frequently per EPOCH. Critically, this also means that backpropagation is performed a lower number of times per EPOCH. In addition, the batch size can affect a wide range of other factors such as the generalisation capability and quality of the optimised network; these factors tend to deteriorate as the batch size becomes too large [16] .The batch size is optimised using a grid search approach in the same way as the learning rate. The range of batch sizes considered are: 5, 10, 15, 20, 30, 40, 50, 100, 200, 300, 400, 500 for the FNN and 3, 5, 8, 10, 15, 20, 30, 40, 50, 100, 200, 300, 400, 500 for the RNN.

4.7. Introduction to Testing Approaches

The optimised ANNs must be validated using unseen data (Test Set 1 & Test Set 2) to assess their predictive capabilities and to determine if any issues such as overfitting or underfitting have occurred. In this section, two different methodologies to test the predictive capabilities of the FNN and RNN will be introduced: online and offline simulation. A mathematical indicator to quantify the prediction accuracy of the networks, namely the Mean Absolute Percentage Error (MAPE), will also be introduced.

4.7.1. One-Step-Ahead Prediction (Online Simulation)

Online simulation is a testing approach where the ANNs receive experimental inputs at each timestep (every 12 hours) and use these inputs to predict the state of the system at the next timestep, hence, the ANNs only need to predict a maximum of 12 hours ahead (one-step-ahead prediction). Consequently, it is expected that the ANNs will demonstrate accurate predictive capabilities during online testing [14]. In the case of the FNN, it receives one input (feature vector) at a time from a single timestep of a single experiment and predicts the outputs at the next timestep. The order in which it receives the inputs is not important as an FNN does not make links between past and future predictions. On the other hand, the RNN receives an entire sequence of 12 inputs spanning 144 hours which represents the operation of an entire experiment. This is because, the RNN uses its knowledge of previous datapoints in the sequence to influence its future predictions by passing a hidden state to itself at each subsequent timestep. Consequently, the RNN operates like the unfolded loop in Figure 3, where there are 12 copies of the same FNN: one for each datapoint in the sequence. Each FNN in the sequence receives the experimental inputs at the current timestep and a hidden state from the FNN at the previous timestep; the FNN uses these inputs to predict the outputs at the next timestep. Note that the first FNN in the sequence receives a hidden state filled with zeros as there is no prior hidden state to feed to the FNN.

Also note that it is possible to feed an entire sequence of 12 points to the RNN as the experimental data has already been recorded, consequently, all 12 points are immediately available. However, during the operation of a real bioprocess, datapoints will only be available every 12 hours; consequently, it is not possible to feed all 12 datapoints at once to the RNN (which is a sequence of 12 FNNs). In this case, the sequence length of the RNN would simply be changed from 12 to 1, such that the RNN is essentially just a single FNN. However, it is important to realise that this FNN differs from a standard FNN in that it possesses a hidden state. During operation, the hidden state would be manually fed from the previous timestep to the next timestep. Hence, only one input is fed to the RNN (which is now a single FNN that possesses a hidden state) at a time along with the hidden state from the previous timestep.

4.7.2. Multi-Step-Ahead Prediction (Offline Simulation)

In offline simulations, the ANNs are only provided with the initial datapoint in a sequence. Using this initial datapoint, the ANNs simulate the entire bioprocess spanning 144 hours, therefore, they predict multiple steps ahead (multi-step-ahead prediction). For example, the ANN is provided with the initial datapoint at t = 0h. Using this datapoint, the ANN predicts the outputs at the next timestep, t = 12h. However, instead of providing the ANN with the experimental inputs at t = 12h, the output predictions at t = 12h are provided to the ANN [14]. Using these output predictions as inputs, the ANN predicts the value at t = 24 h. This process is repeated until the entire process has been simulated. Offline prediction is a more rigorous test of the ANNs predictive capability as the prediction errors accumulate as the simulation progresses [14]. Consequently, a well-developed model will be able to accurately simulate the bioprocess by limiting the magnitude of the prediction errors.

The FNN operates as explained in Section 4.7.1 above, the only difference being that the inputs for a given timestep are the predicted state variables at the previous timestep along with the Li and F_C which are constant values that do not change during the simulation. Thus, the FNN only receives one set of experimental values which are the initial conditions at t = 0h. The RNN operates with a sequence length of 1 as explained in Section 4.7.1 above. However, like the FNN, it only receives one set of experimental values which are the initial conditions at t = 0h. The inputs for the following timesteps are the predicted state variables at the previous timestep, Li, F_c and the hidden state, $\boldsymbol{h}$, at the previous timestep.

4.7.3. Analysis of Predictive Performance

The prediction accuracy of the network will be assessed using the Mean Absolute Percentage Error (MAPE) by comparing the predicted values from the network to the experimental (real) values. The MAPE is defined in equation 14 below.

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{Y_{i,e} - Y_{i,p}}{Y_{i,e}}\right| \times 100\% \tag{14}$$

where $Y_{i,e}$ is the experimental value, $Y_{i,p}$ is the predicted value and N is the total number of predictions.

5. Results and Discussion

In this section, the hyperparameter optimisation results for the FNN and the RNN will be presented and their significance discussed. The simulation results will then be presented where the optimised networks will be trained and tested in an online and offline framework to assess their predictive capabilities. Finally, the simulation results of the FNN and RNN will be compared in order to determine the superior modelling strategy.

5.1. Hyperparameter Optimisation Results

By performing hyperparameter optimisation, the first and second objectives set forth at the beginning of the report will be achieved. Furthermore, this paves the way to achieving the overarching aim of the project which is to construct a mathematical model that can simulate the dynamic behaviour of the microalgal lutein production bioprocess. The impact of this achievement, as explained in the introduction, is that it will enable optimisation and control of the bioprocess which will improve the economic viability of the bioprocess. This is the biggest hurdle preventing the industrialisation of the lutein production bioprocess.

5.1.1. Results of Hidden Layer Optimisation

The only way to determine the optimal number of hidden layers is to complete the entire hyperparameter optimisation process outlined in Section 4.6 for each number of hidden layers in the range considered: 1, 2 & 3. This results in 6 optimised networks, 3 FNNs and 3 RNNs. The networks are then trained and tested on Test Set 1 and Test Set 2 to determine the optimal number of hidden layers. This is the procedure that was followed, and it was found that for the FNN, the predictive capability of the network improved from 1 to 2 hidden layers. This suggests that an FNN with 1 hidden layer is not complex enough to model the training data; there may be some level of underfitting occuring as there are not enough parameters in the model. This results in poor predictive performance on the testing dataset. Thus, increasing the number of hidden layers increases the number of parameters and hence the complexity of the model. This allows the model to fit more closely to the training data, however, not too closely which would lead to overfitting. Interestingly, it was found that increasing the number of hidden layers from 2 to 3 actually decreased the predictive capability of the model. This suggests that an FNN with 3 hidden layers is too complex as it contains too many parameters which leads to overfitting. Therefore, it can be concluded that the optimal number of hidden layers for the FNN is 2 hidden layers. Consequently, the hyperparameter optimisation results presented in the following sections will be for an FNN that consists of two hidden layers.

In the case of the RNN, it was found that the predictive capability decreased as the number of hidden layers is increased beyond 1. This indicates that an RNN with 1 hidden layer is sufficiently complex to accurately model the training and testing data. It also suggests that overfitting is occurring beyond 1 hidden layer as the RNN contains too many parameters and is thus too complex. The fact that the RNN begins to overfit with only 2 hidden layers whereas the FNN begins to overfit with 3 hidden layers can be explained by the presence of hidden states. Unlike FNNs, each neuron in an RNN possesses a hidden state. This hidden state has an associated hidden state vector, $\boldsymbol{u}$, as explained in Section 3.4. Consequently, per neuron, an RNN contains significantly more parameters which explains why an RNN overfits with a lower number of hidden layers in comparison to the FNN. Therefore, it can be concluded that the optimal number of hidden layers for the RNN is 1. Hence, the hyperparameter optimisation results presented in the following sections will be for an RNN that consists of 1 hidden layer.

5.1.2. Results of Hidden Neurons and EPOCHS Optimisation

The optimisation results for the number of hidden neurons and EPOCHS in the case of the FNN with 2 hidden layers are shown in Figure 6 below. The x-axis displays the hidden neuron configurations and the y-axis displays the average MSE, $E_{opt}(t)$, at the optimum number of EPOCHS, *t*. The optimum number of EPOCHS for each hidden neuron configuration is displayed at the top of the corresponding bar. The average MSE is simply the Mean Squared Error (MSE) averaged over all predictions on the testing dataset (refer to appendix for sample calculation).

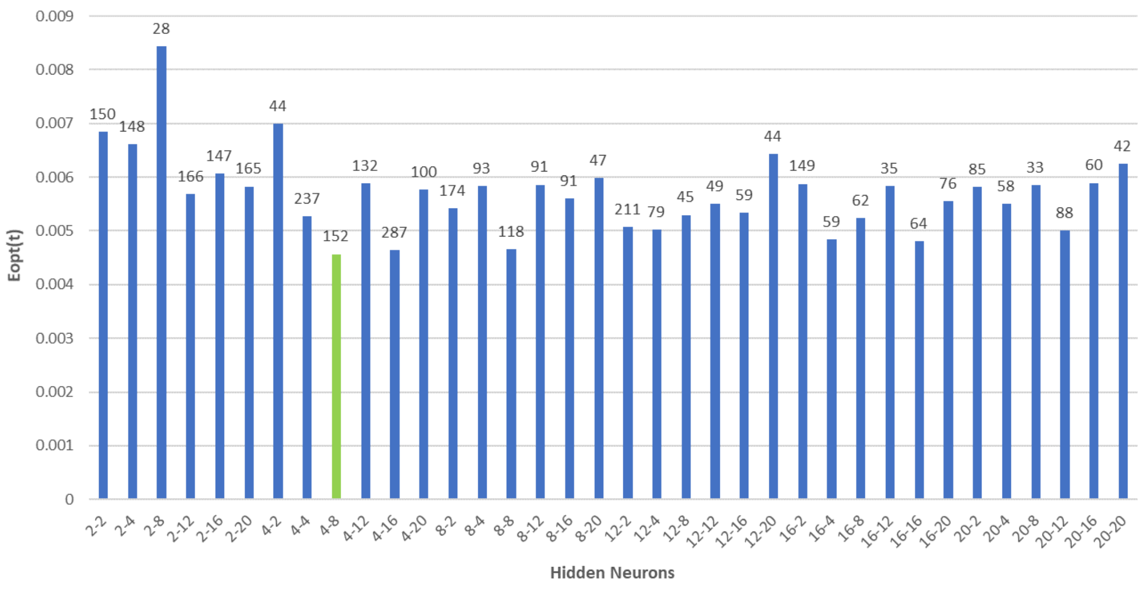


Figure 6. Hidden Neurons and EPOCHS Optimisation Results for FNN.

From Figure 6 above, it can be seen that the optimal number of hidden neurons in the first hidden layer is 4, and in the second is 8 (4-8). The corresponding optimal number of EPOCHS for this hidden neuron configuration is 152. In general, the number of parameters increases with the total number of neurons. However, the configuration of the neurons also affects the number of parameters. For example, networks with the following configurations: 2-8 and 8-2, have the same total number of hidden neurons. However, the number of parameters are 63 and 75 respectively (refer to appendix for sample calculations). Thus, to get a better idea of the relationship between the complexity of the network and the average error, the optimisation results in Figure 6 have been rearranged such that the x-axis is in ascending order with respect to the total number of parameters; this is shown in Figure 7 below. Note that the number of parameters for a given hidden neuron combination are listed in red below the optimal number of EPOCHS.

As can be seen in Figure 7 below, the average error gradually decreases from the first hidden neuron combination, 2-2, up to a minimum point at 4-8. Thus, the accuracy of the FNN is generally increasing as the number of parameters increases. This is because, the larger number of parameters allows the model to fit more closely to the training data. Large fluctuations in error such as that seen for a hidden neuron combination of 2-8 are most likely due to the stopping condition employed in the optimisation procedure. The generalisation loss stopping condition tends to terminate the optimisation procedure prematurely [33]. This early termination is reflected by the low number of EPOCHS of 28 in comparison to the neighbouring hidden neuron combinations. This low number of EPOCHS has most likely resulted in underfitting which is reflected by an extremely high average error which can be seen visually in Figure 7. To resolve this issue, a more robust stopping condition is required which is a limitation of the research performed in this report. Thus, an interesting topic of future research would be the investigation of more robust stopping conditions.

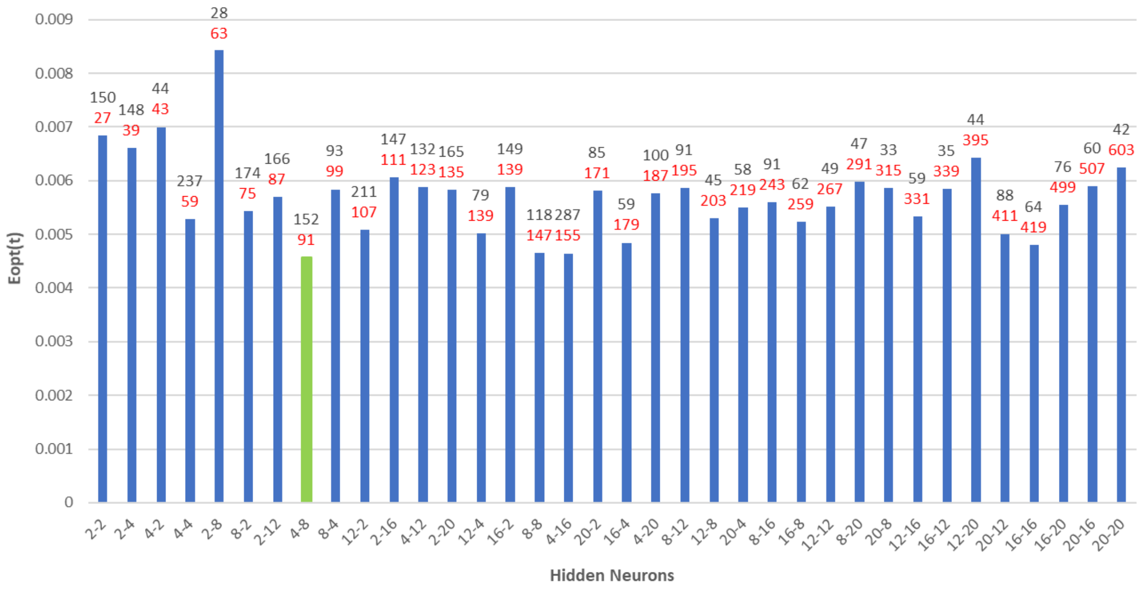


Figure 7. Hidden Neurons and EPOCHS Optimisation Results for FNN (in ascending order of parameters).

Beyond the optimum hidden neuron combination of 4-8, the average error for the hidden neuron combinations is either higher or similar in magnitude to the optimal combination. This implies that beyond the optimal configuration, an increase in the number of parameters has no benefit to the predictive performance of the network and may actually be hindering its performance due to overfitting. An excessive number of parameters leads to overfitting as the model essentially fits to the 'noise' in the training dataset which decreases its predictive capability on the unseen testing dataset.

For a model with a correctly tuned number of hidden neurons, overfitting and underfitting can still occur if the number of EPOCHS is tuned incorrectly. If the number of EPOCHS is too small, underfitting will occur and if the number of EPOCHS is too large then overfitting will occur. Furthermore, for a model that has too many neurons and thus too many parameters, overfitting will occur faster with each EPOCH. Thus, the optimal number of EPOCHS will most likely be lower as overfitting will occur at a lower number of EPOCHS. This phenomenon can be seen in Figure 7 above. For FNNs with 171 parameters or more, the optimal number of EPOCHS is 100 or less in all cases. Whereas for FNNs with less than 171 parameters, the optimal number of EPOCHS frequently exceeds 100 with combinations such as 4-4 and 4-16 training for well above 200 EPOCHS.

The optimisation results for the RNN consisting of 1 hidden layer are shown in Figure 8 below. In this case, the number of parameters is directly proportional to the number of hidden neurons, therefore, the x-axis is already in ascending order with respect to the number of parameters.

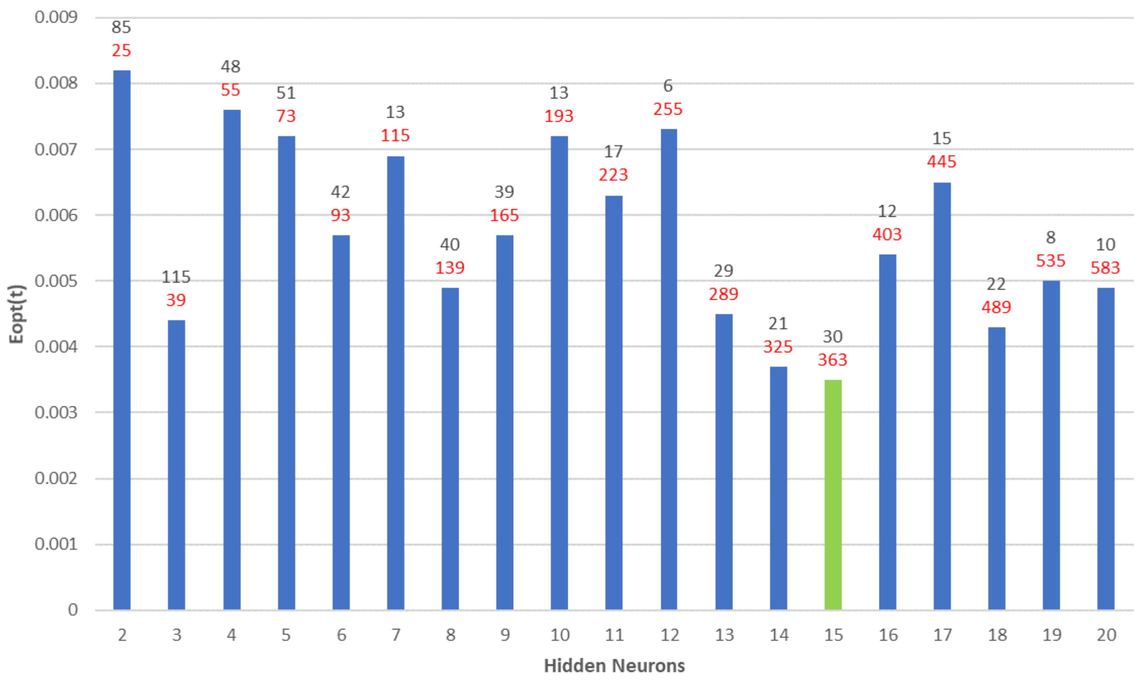


Figure 8. Hidden Neurons and EPOCHS Optimisation Results for RNN.

The optimum number of hidden neurons for the RNN is 15 and the optimal number of training EPOCHS is 30 as shown in Figure 8 above. The first thing that is evident from the optimisation results for the RNN is that the optimal number of training EPOCHS are generally much lower in comparison to the FNN. This suggests that the RNN is overfitting at a much lower number of EPOCHS than the FNN. This seems quite unusual as the number of parameters for a 1 hidden layer RNN are comparable to that of a 2 hidden layer FNN. There are a variety of possible reasons for this occurrence.

The most likely reason is due to the difference in the rate of learning of the RNN in comparison to the FNN. During the optimisation process, it was observed that the loss of the RNN decreases much more quickly than the loss of the FNN. This means that the RNN essentially learns faster which in mathematical terms means that the RNNs weight updates are larger and more efficient; the RNN takes less weight updates to arrive at the optimal solution. This increased learning speed is due to the difference in activation functions between the FNN and the RNN. The RNN uses the hyperbolic tangent activation function in its hidden layers which produces a larger output range than the sigmoid activation function [32]. The output range of the hyperbolic tangent activation function is [-1, 1] whereas the output range for the sigmoid function is [0, 1] [32]. This larger output range results in higher gradients which increase the learning speed as the weight updates are larger. Another reason for the faster learning speed of the RNN is that it has access to its past memory through the hidden state values that it receives which the FNN does not have access to. Consequently, the RNN possesses more efficient parameter adjustments as it utilises more information to make its predictions. Therefore, the fact that the RNN requires less EPOCHS to train and less EPOCHS to overfit can be attributed to the RNNs faster learning ability.

Furthermore, focussing specifically on the optimal networks, the optimal RNN consists of 363 parameters whereas the optimal FNN consists of only 91. This also explains why there is such a large difference in the optimal number of EPOCHS between these networks. The RNN is vastly more complex as it consists of exponentially more parameters. Thus, the RNN overfits at a lower number of EPOCHS, hence, possesses a lower optimal EPOCH value. A further contributing factor to the low EPOCH values for the RNNs could again be the generalisation loss stopping condition terminating the optimisation procedure early. Hence, it is again emphasised that the first point of further research should focus on alternative stopping conditions. Finally, a noteworthy result from the optimisation results is that the average error for the optimal RNN configuration is much lower than the optimal FNN: approximately 0.0035 vs 0.0045. This suggests that the RNN will display better predictive performance; the networks will be tested and compared later in Section 5.2.

5.1.3. Results of Learning Rate Optimisation

In this stage, an FNN and an RNN are constructed using the optimal hyperparameter values determined thus far. The FNN and RNN are trained at each value in the range of learning rates and subsequently tested on the testing dataset yielding an average MSE value. Note that the learning rate utilised during the hidden neurons and EPOCHS optimisation stage was 0.001 for both the FNN and the RNN. This learning rate was chosen through initial testing before the hyperparameter optimisation stage as it was found that the FNN and the

RNN performed well at this value of learning rate. From the optimisation results it will be interesting to see how far away from the optimum value this initial learning rate is.

The learning rate optimisation results for both the FNN and the RNN are shown in Figure 9 below; the results for the FNN are on the left and for the RNN on the right. The FNN consists of 2 hidden layers with 4 neurons in the first hidden layer, 8 neurons in the second hidden layer and is trained for 152 EPOCHS. The RNN consists of 1 hidden layer, 15 hidden neurons and is trained for 30 EPOCHS. As explained in Section 4.6.3, the learning rate range was split into 4 sub-ranges. The optimisation results in Figure 9 only display the sub-range that contains the global optimum solution for readability.

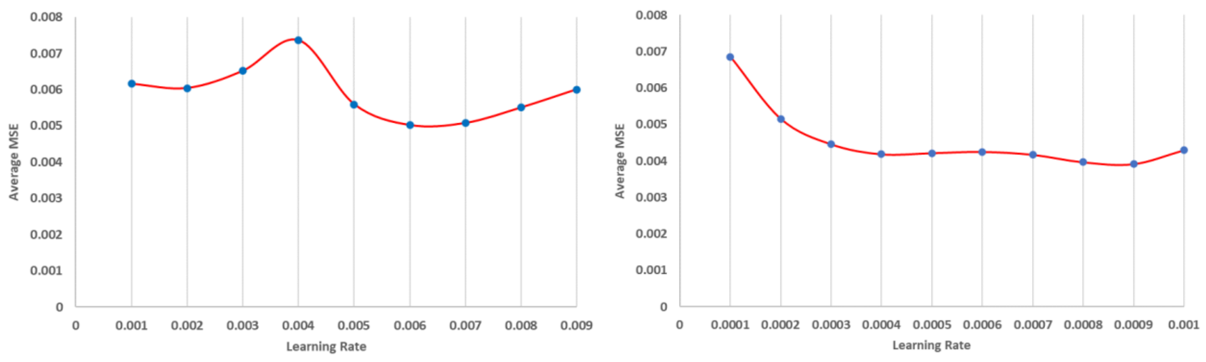


Figure 9. Learning Rate Optimisation Results for FNN (Left) and RNN (Right).

From Figure 9 above, focussing on the results for the FNN, it can be seen that the error initially increases from a learning rate of 0.001 up to a maximum point at a learning rate of 0.004. After this maximum point, the error rapidly decreases up to a global minimum point at learning rate of 0.006 after which the error begins to gradually increase again. Consequently, the optimum learning rate was taken to be 0.006. Note that the error decreases slightly from a learning rate of 0.006 to a learning rate of approximately 0.0065. Thus, a further search could be carried out in the range between 0.006-0.007, however, it was decided that this was not worth the further computational effort for such a minor decrease in error. Also note that the initial learning rate of 0.001 that was chosen through trial and error was within the subrange of the optimal learning rate. Furthermore, when compared to the initial learning rate, the optimum learning rate decreased the average error from approximately 0.006 to 0.005; an improvement of almost 20%. Note that the optimum learning rate is in the middle of the overall range, however, it is still small in magnitude. Thus, this implies that smaller weight updates are favoured which increases the training time and moves the network towards the optimum value slowly; thus, preventing the overshooting of the minimum point and the arrival at a sub-optimal solution.

Moving onto the RNN, it can be seen that error initially decreased rapidly from a learning rate of 0.0001 to a local minimum point at a learning rate of 0.0004. The error then increases slightly up to a learning rate of 0.0006. After this, the error decreased to a global minimum point at a learning rate of 0.0009. Thus, the optimum learning can be taken as 0.0009. Note that the optimum learning rate is only 1 datapoint away in the search range from the initial learning rate (0.001) that was chosen through initial testing. Furthermore, the true optimum point decreases the prediction error from approximately 0.0045 to 0.004; an improvement of just above 10%. Just like the FNN, the learning rate is low in magnitude, thus, the points made above also apply to the RNN. Furthermore, the RNN has an optimal learning rate that is approximately 7 times lower than the FNN. This is because, as explained in Section 5.1.2, the magnitude of the derivative of the loss function with respect to the parameters in the network are larger for the RNN than the FNN; hence a lower learning rate is required to reduce the size of the weight updates for the RNN and to therefore prevent overshooting the minimum point.

In the seminal study by del Rio-Chanona et al. [14] which laid the foundations for FNN construction and optimisation when applied to the lutein production bioprocess, no consideration was given to the selection or optimisation of the learning rate. In this study, optimising the learning rate resulted in a performance increase of 20% for the FNN and 10% for the RNN. From these values we can conclude that the learning rate is an important hyperparameter as the performance gains are substantial. Furthermore, in this study, the initial learning rate was selected through initial testing such that it provided good predictive performance. If the choice of the initial learning rate was poor and therefore further away from the optimal learning rate, then the performance increase would be even more substantial. Therefore, this study improves upon the work performed by del Rio-Chanona et al. [14] by highlighting the importance of the learning rate and by providing a comprehensive optimisation strategy.

5.1.4. Results of Batch Size Optimisation

After optimisation the learning rate, the batch size is optimised. During the previous hyperparameter optimisation phases, the FNN utilised a batch size of 50 and the RNN utilised a batch size of 8. Again, these batch sizes were chosen before the hyperparameter optimisation phase through initial testing; it was found that the networks performed well with these batch sizes. Note that a batch size of 50 for the FNN means that it receives 50 datapoints before the weights are updated. However, for the RNN, a batch size of 8 translates into 96 datapoints as the RNN is provided with sequences of data rather than individual datapoints. Thus, a batch size of 1 for the RNN practically means that it receives 1 sequence of data consisting of 12 individual points (i.e. a 144 hour experiment). The batch size optimisation results for the FNN and RNN are presented in Figure 10 below.

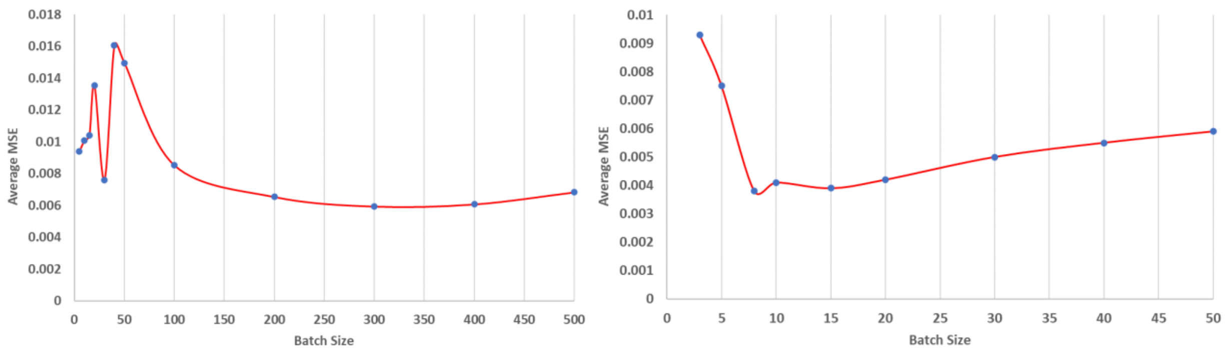


Figure 10. Batch Size Optimisation Results for FNN (Left) and RNN (Right).

From Figure 10 above, focussing on the FNN, it can be seen that the error initially oscillates from a batch size of 5 up to a batch size of 40. After this, the error decreases rapidly up to a global minimum point at a batch size of 300. In comparison to the initial batch size of 50, the optimal batch size decreased the average error from approximately 0.008 to 0.006; an improvement of 25%.

Moving onto the RNN (note that the optimisation results for the RNN are only displayed up to a batch size of 50 for readability), it can be seen that the error decreases rapidly from a batch size of 3 to global minimum point at a batch size of 8. After this, the error continuously increases in a gradual manner. Thus, the optimal batch size is 8 which is the same as the initial batch size. Furthermore, the average error of the fully optimised RNN is much lower than the fully optimised FNN; approximately 0.0035 vs 0.006. Hence, it is expected that the RNN will perform better during simulation phase that is to follow.

Returning to the study by del Rio-Chanona et al. [14], another weakness of this work is that the selection and optimisation of the batch size is not considered. In this study, optimising the batch size for the FNN resulted in a performance increase of 25% and for the RNN the initial batch size was also the optimal batch size. A performance increase of 25% is substantial and again, as in the case of the learning rate, if the initial choice of batch size was poor then the performance increase would be even more significant. Thus, this study further improves on the work performed by del Rio-Chanona et al. [14] by highlighting the importance of the batch size and by providing a comprehensive optimisation strategy.

5.1.5. Hyperparameter Comparison

In this section, the most important hyperparameter for both the FNN and the RNN will be identified by quantifying the performance improvement provided by optimising each individual hyperparameter. The performance improvement metric is simply the percentage reduction in prediction error (average MSE) between the worst performing hyperparameter value in the defined range to the best performing hyperparameter value. The performance improvement values will then be compared to determine which hyperparameter provides the greatest reduction in prediction error and can therefore be considered the most important hyperparameter. The performance improvement values are summarised in Table 2 below (please refer to the appendix for sample calculations).

Table 2. Performance Improvement from Hyperparameter Optimisation (the worst performing hyperparameter value in the pre-defined range is used as a benchmark for the performance improvement calculation).

	Hidden Neurons & EPOCHS	**Learning Rate**	**Batch Size**
FNN - Performance Improvement	45.2%	99.8%	63.1%
RNN - Performance Improvement	57.3%	93.2%	69.1%

From Table 2 above, it is immediately evident that the most important hyperparameter for both the FNN and the RNN is the learning rate. This is because, the learning rate provides the greatest potential performance improvement, thus, if tuned incorrectly the performance of the network will be hindered greatly. The 2nd most important hyperparameter is the batch size for both the FNN and the RNN. Finally, the least significant hyperparameter appears to be the number of hidden neurons and EPOCHS. However, it is important to note that the number of hidden neurons and EPOCHS is still an important hyperparameter that must be tuned correctly as the magnitude of the performance improvement is large. These results again highlight the weaknesses in the study performed by del Rio-Chanona et al. [14] as the two most important hyperparameters appear to be the learning rate and the batch size. However, these hyperparameters were not considered in the study.

5.1.6. Training Optimised Neural Networks

The optimised FNN with a hidden neuron configuration of 4-8, a learning rate of 0.006 and a batch size of 300 was trained for 152 EPOCHS. Next, the optimised RNN consisting of 1 hidden layer, 15 hidden neurons, a learning rate of 0.0009 and a batch size of 8 was trained for 30 EPOCHS. The predictive performance of the optimised and trained networks was subsequently validated by testing on Test Set 1 and Test Set 2 in both an online and offline framework. The results of this testing are presented in Section 5.2 below.

5.2.Simulation Results

By testing the optimised ANNs in both an online and offline framework and subsequently comparing their performance in order to determine the superior modelling strategy, the 3rd and 4th objectives set forth at the beginning of the report will be achieved. Furthermore, if the ANNs can accurately simulate the bioprocess in both an online and offline framework, then the overarching aim has also been achieved which is to construct a mathematical model that can simulate the dynamic behaviour of the microalgal lutein production bioprocess.

5.2.1. Validation with Online Simulation

In Figure 11 below, the online simulation results for Test Set 1 are presented. The simulation results for both the FNN and the RNN have been presented side-by-side for easy comparison. The results for the FNN are presented on the left and for the RNN on the right. The blue datapoints represent the experimental values and the red datapoints represent the ANNs predictions. The Mean Absolute Percentage Error (MAPE) has also been presented on each graph to quantify the overall accuracy of the simulations.

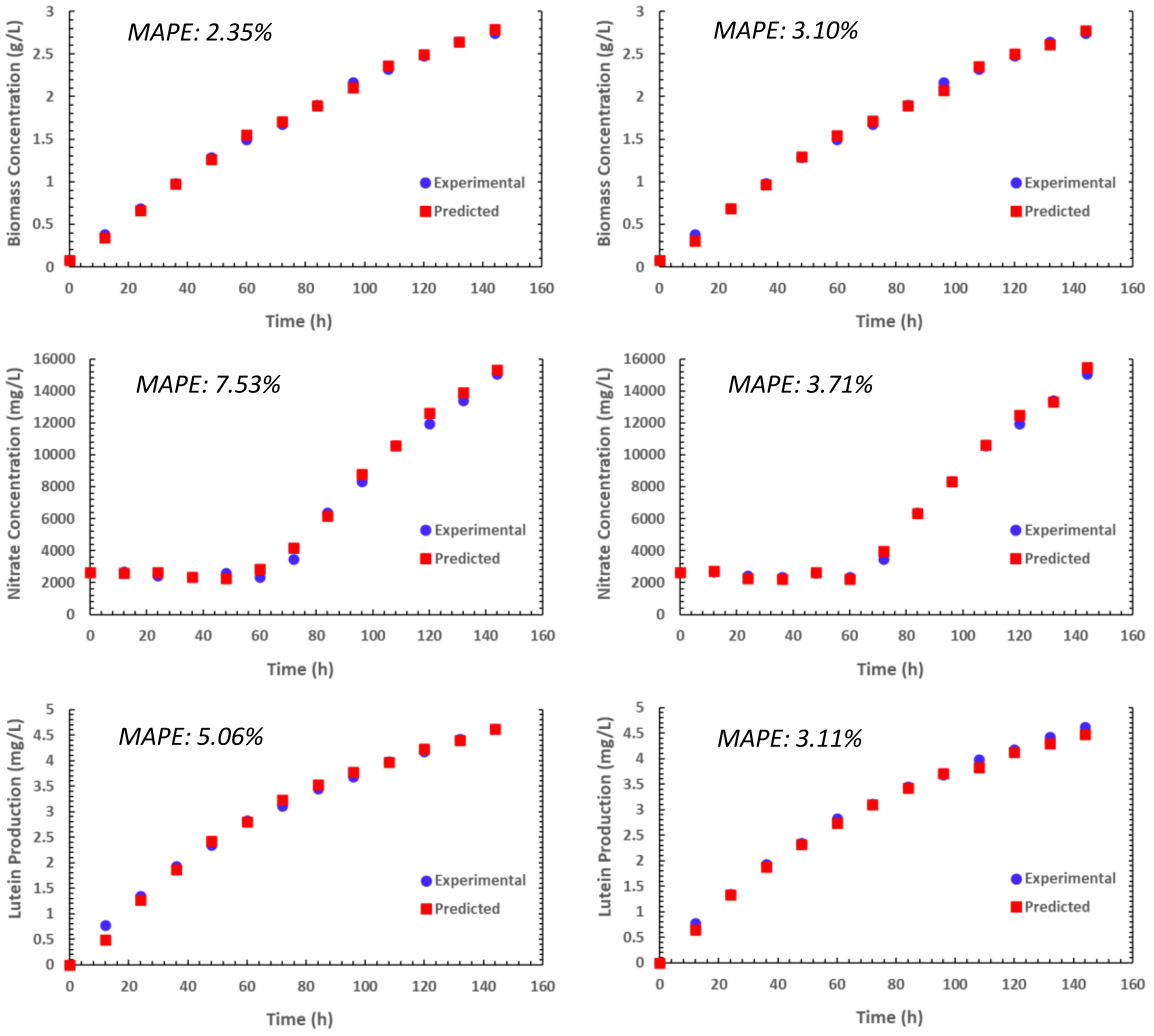


Figure 11. Test Set 1 Online Simulation Results (Left: FNN, Right: RNN).

Focussing on the biomass concentration simulations in Figure 11 above, both the FNN and the RNN follow the experimental results extremely closely over the entire 144-hour period. However, the FNN is slightly more accurate with a MAPE of 2.35% compared to 3.10% for the RNN. Moving onto the nitrate concentration simulations, initially the FNNs predictions follow the experimental datapoints closely, however, after the 40th hour the predictions begin to deviate slightly which is reflected in the higher MAPE value of 7.53%. The RNNs predictions on the other hand follow the experimental results almost perfectly up to the 60th hour after which the predictions also begin to deviate. However, compared to the FNN these deviations are lower in magnitude and also occur less frequently; this is reflected in the lower MAPE for the RNN which is 3.71%. Finally, moving on to the lutein production simulation, the RNN again outperforms the FNN with a MAPE of 3.11% compared to 5.06 % for the FNN. This difference can be attributed to the FNNs prediction at the 12th hour which deviates by a large margin from the experimental datapoint; far more than the RNN. From this analysis it can be concluded that both of the ANNs can be successfully utilised for online simulation of Test Set 1, however, the RNN marginally outperforms the FNN.

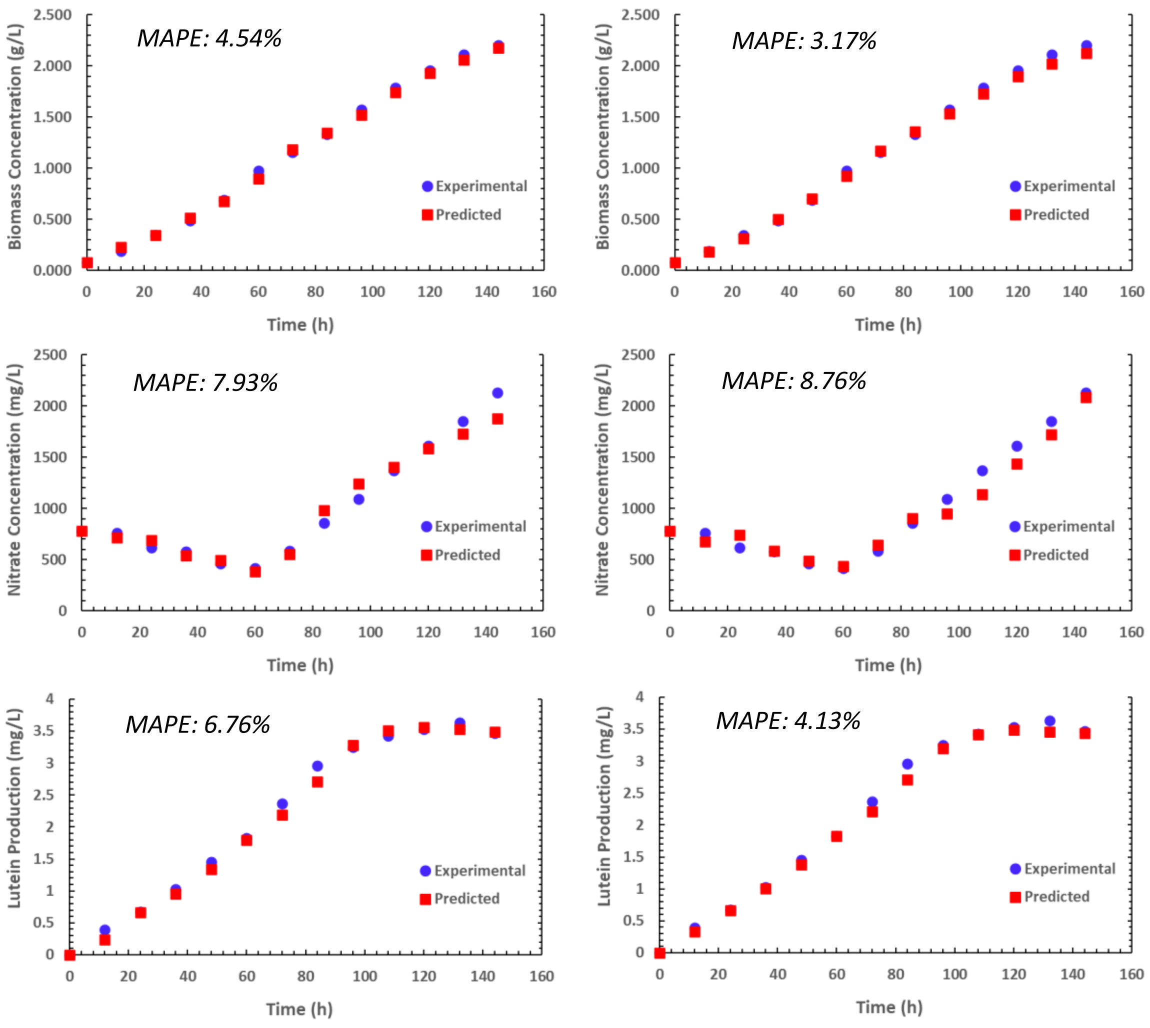


Figure 12. Test Set 2 Online Simulation Results (Left: FNN, Right: RNN)

The online simulation results for Test Set 2 are presented in Figure 12 above. From Figure 12, it can be seen visually and through the low MAPE values that the ANNs can accurately simulate the experiments with only minor deviations much like the simulation results for Test Set 1. However, a noteworthy experiment is the nitrate concentration experiment where deviations occur frequently and are large in comparison to the other experiments. As explained by del Rio-Chanona et al. [14], a reason for this could be that the increase in nitrogen concentration caused by nitrogen injection at the 60th hour is not well represented. This is due to the lack of variety of nitrogen inflow concentrations in the training dataset.

This hypothesis is strengthened by the fact that deviations begin to occur frequently after the 60th hour where the nitrogen inflow is turned on. This can be resolved by increasing the variety of nitrogen inflow concentrations in the training dataset. From Figure 12, it can be seen that the RNN again outperforms the FNN in 2/3 simulations. However, it is again important to note that both ANNs can accurately simulate Test Set 2 in an online framework.

In summary, for both Test Set 1 and Test Set 2, the MAPE across all simulations for both networks is below 10%. Consequently, it can be concluded that both the FNN and the RNN can simulate the short-term dynamic behaviour the microalgal lutein production bioprocess. Hence, they can be used for short-term optimisation and control. In addition, from the analysis performed above it is clear that the RNN outperforms the FNN by a small margin. However, both of the ANNs display highly accurate online predictive performance, thus, they are both suitable for this task.

5.2.2. Validation with Offline Simulation

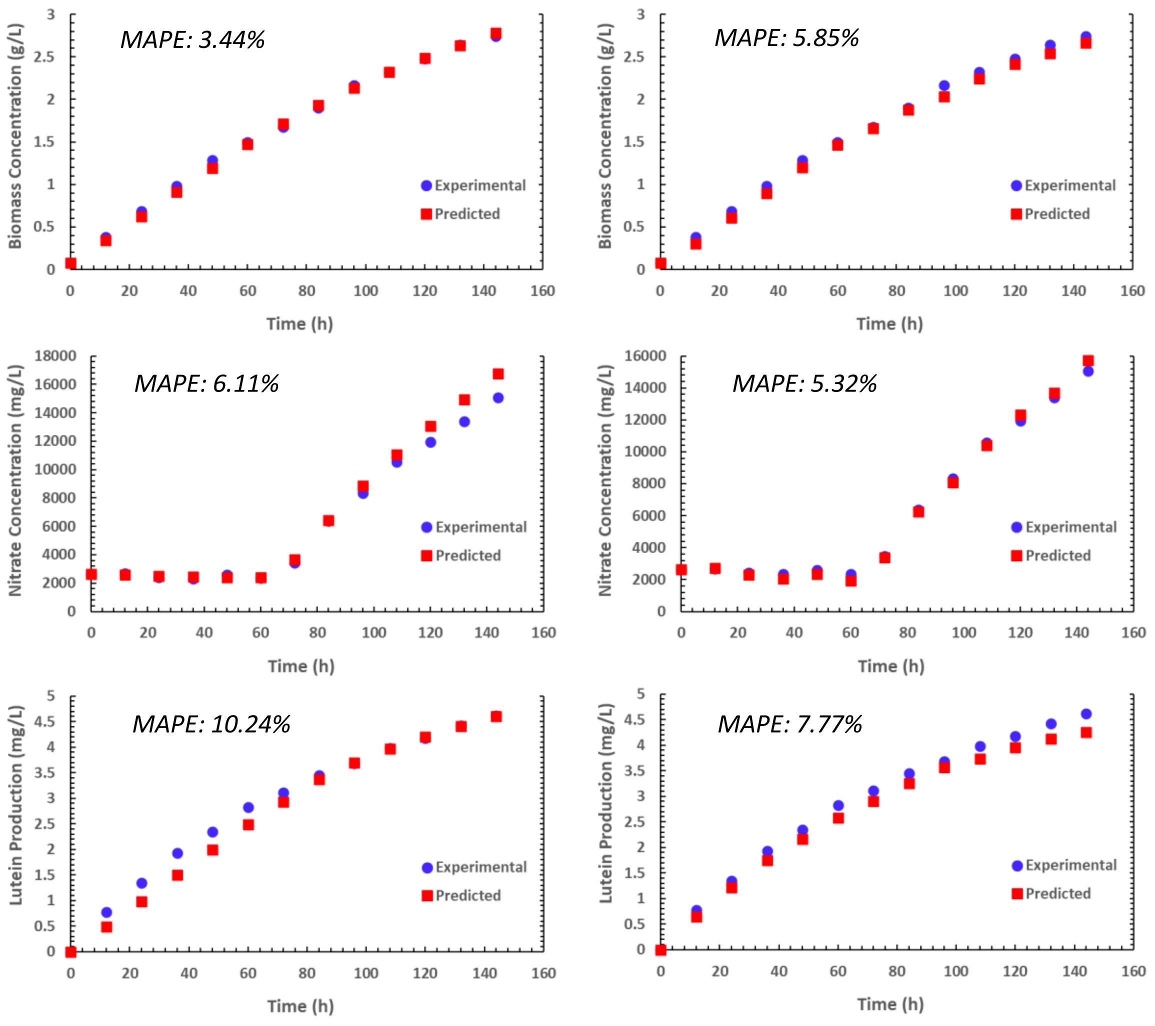


Figure 13. Test Set 1 Offline Simulation Results (Left: FNN, Right: RNN).

The offline simulation results for Test Set 1 are presented in Figure 13 above. Focussing on the biomass concentration simulations, both the FNN and the RNN simulate the experimental data extremely accurately with only minor deviations. In this case, the FNN outperforms the RNN with a MAPE of 3.44% compared to 5.85% for the RNN. Moving onto the nitrate concentration simulations, the FNN simulates the experimental data extremely accurately with only minor deviations up the 96th hour. However, after the 96th hour, the deviations become more noticeable and continuously increase in magnitude as time progresses. This continual decrease in accuracy can be explained by error accumulation. As explained in Section 4.7.2, in offline simulation the models are only provided with the initial datapoints at t = 0h. From these initial datapoints, the models simulate the entire bioprocess. Thus, the prediction errors accumulate from t = 0h all the way up to t = 144h. In the case of the FNN, the error accumulation becomes noticeable at the 96th hour and beyond. On the other hand, the RNN accurately simulates the bioprocess all the way up to the 132nd hour before any major deviations occur. After the 132nd hour, a single major deviation is noticeable which is again likely due to error accumulation. Thus, in this case, the RNN outperforms the FNN as reflected by the MAPE values; 5.32% for the RNN compared to 6.11% for the FNN.

Finally, moving on to the lutein production simulations, the RNN outperforms the FNN by quite a large margin. The RNN yields a MAPE of 7.77% compared to 10.24% for the FNN. During the first 72 hours of operation, the deviation of the predicted results from the experimental results was consistently large for the FNN. However, after the 72nd hour, this the deviation almost disappears completely and the FNN accurately simulates the remaining part of the experiment. This is strange as it suggests that error accumulation is large early in the experiment, however, the FNN manages to almost eliminate this accumulated error after 72 hours of operation. On the other hand, the results for the RNN are more consistent with the theory of error accumulation. The RNN manages to simulate the experiment accurately during the early operational hours and the accuracy begins to decrease as the experiment progresses. However, these deviations are considerably smaller than the deviations seen in the FNN during the early operational hours. From the results in Figure 13, it can be concluded that both the FNN and the RNN can accurately simulate Test Set 1 in an offline framework. However, the RNN provides some slight performance improvements.

The offline simulation results for Test Set 2 are provided in Figure 14 below. Focussing on the biomass concentration simulation, for the first 48 hours the RNN manages to simulate the experiment almost perfectly whereas the FNNs predictions are seen to deviate. These deviations look minor due to the scale of the graph (the values near the end of the operational period are much larger than the initial values), however, the magnitude of these deviations is large. For the FNN, the deviation of the 1st prediction at the 12th hour is 23%, the 2nd 14%

and the 3rd is also 14%. On the other hand, the RNNs deviations for these same three predictions are all below 10% with the 1st and 3rd predictions having deviations below 5%. Near the end of the operational period, the accuracy begins to decrease for both networks due to error accumulation, however, the RNNs deviations are larger than the FNNs. Nevertheless, the RNN outperforms the FNN with a MAPE of 6.79% compared to 7.15% for the FNN as the RNNs superior performance during the first 48 hours outweighs the FNNs superior performance during the later operational hours.

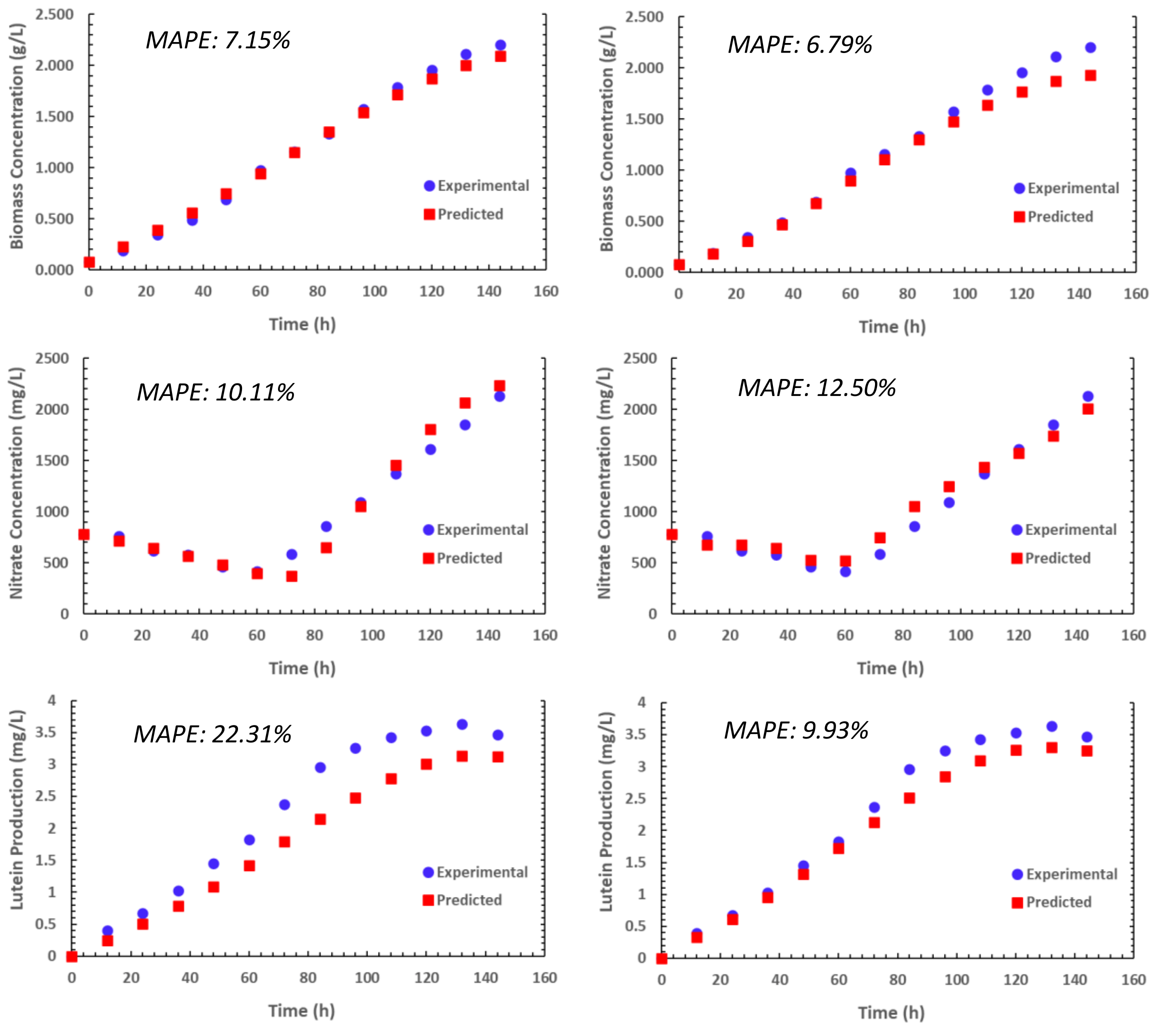


Figure 14. Test Set 2 Offline Simulation Results (Left: FNN, Right: RNN).

Next, for the FNN and the RNN, the nitrate concentration simulations demonstrate the worst predictive performance so far with MAPEs above 10% in both cases; the FNN outperforms the RNN with a MAPE of 10.11% compared to 12.49% for the RNN. This poor performance can be explained by the contribution of two factors. Firstly, accumulation of error which is present during offline prediction. Secondly, this error is heightened by the fact that the increase in nitrogen concentration at the 60th hour is not well represented as explained in Section 5.2.2. Furthermore, as stated earlier, this can be resolved by increasing the variety of nitrogen inflow concentrations in the training data. A more comprehensive hyperparameter optimisation procedure may slightly improve the performance, however, the root of the problem lies in the experimental dataset. Finally, the lutein production simulation is the most significant difference in performance thus far between the FNN and the RNN. The RNN drastically outperforms the FNN with a MAPE of 9.93% compared to 22.31% for FNN. The RNN manages to accurately simulate the process up to the 60th hour after which the deviations become large. However, for the FNN, the deviations are large from the onset and they continue to increase in magnitude throughout the simulation. From the results in Figure 14, it can be concluded that the RNN significantly outperforms the FNN and that the FNN may not be suitable for offline prediction of Test Set 2 as the MAPE surpasses 20% for the lutein production simulation.

In summary, for both Test Set 1 and Test Set 2, the large majority of simulations possess a MAPE of less than 10%. Consequently, it can be concluded that both the FNN and the RNN can simulate the long-term dynamic behaviour of the lutein production bioprocess. Hence, they can be used for long-term optimisation and control. In addition, for the RNN only one simulation exceeds a MAPE of 10%. On the other hand, for the FNN three simulations exceed a MAPE of 10% with one simulation exceeding 20%. This clearly demonstrates that RNNs are superior for offline simulation and thus long-term optimisation and control.

5.2.3. Comparison of FNN vs RNN

Referring back to the online simulation results in Section 5.2.1, it was found that the FNN and RNN can accurately simulate the lutein production bioprocess in an online framework. It was also found that the RNN provides a minor improvement to the FNN in terms of prediction accuracy. However, the procedure to construct, optimise, train and subsequently implement an RNN in real-time is significantly more complex in comparison to the FNN. Thus, it is concluded that the slight performance improvement provided by the RNN is not worth the additional complexity. Consequently, FNNs are recommended for online simulation applications. Referring now to the offline simulation results in Section 5.2.2, it was found that the RNN provides major improvements in predictive performance in comparison to the FNN. Only one offline simulation for the RNN yielded a MAPE above 10%. Whereas for the FNN, three simulations yielded a MAPE above 10% with one simulation exceeding 20%. Consequently, it is concluded that the major performance improvement provided by the RNN is worth the additional complexity. Therefore, RNNs are recommended for offline simulation applications. Finally, it can be concluded that the RNN is the superior modelling strategy as it outperforms the FNN in both of the online simulation tests and both of the offline simulation tests.

6. Conclusions

This project aims to construct a mathematical model that can simulate the dynamic behaviour of the microalgal lutein production bioprocess. The mathematical models employed are FNNs and RNNs. The secondary aim is to determine which mathematical model is superior. To achieve these aims, firstly three FNNs and three RNNs of varying numbers of hidden layers were constructed using the Python programming language and a host of supporting libraries. A comprehensive data pre-processing strategy was then employed to ensure successful training and highly accurate models. In the first stage of the data pre-processing procedure, the input data was rescaled using the standardisation approach. Next, to generate enough training data, the experimental dataset was artificially replicated by embedding random noise. This artificially increased the size of the experimental dataset by 100 times which would be practically infeasible to do in the laboratory. A robust hyperparameter optimisation procedure was then employed to determine the optimal structure of the ANNs and to therefore maximise their predictive performance. From the hyperparameter optimisation results, it was found that the optimal configuration for the FNN is 2 hidden layers with 4 neurons in the first hidden layer, 8 neurons in the second hidden layer, a learning rate of 0.006, a batch size of 300 and 152 EPOCHS of training. For the RNN it was found that the optimal configuration is 1 hidden layer with 15 hidden neurons, a learning rate of 0.0009, a batch size of 8 and 30 EPOCHS of training. Furthermore, it was discovered that the most important hyperparameter is the learning rate as it provides a potential performance improvement of beyond 90% for both the FNN and the RNN if tuned correctly.

The optimised FNN and RNN were then tested in an online and offline framework using an unseen testing dataset. For both the FNN and the RNN, the MAPE across all online simulations was less than 10%. Therefore, it can be concluded that both the FNN and the RNN can simulate the short-term dynamic behaviour of the lutein production bioprocess. Furthermore, the RNN provided a slight performance improvement over the FNN. However, it was decided that this slight performance improvement is not worth the extra complexity involved in constructing, optimising and deploying the RNN in real-time. Consequently, FNNs are recommended for online simulation applications. Moving onto the offline simulation results, the MAPE across the majority of the simulations is less than 10%. Therefore, it can be concluded that both the FNN and the RNN can simulate the long-term dynamic behaviour of the lutein production bioprocess. Furthermore, for the RNN only one simulation exceeds a MAPE of 10% whereas for the FNN three simulations exceed a MAPE of 10% with one simulation exceeding a MAPE of 20%. Therefore, the RNN provides a significant performance improvement over the FNN. Consequently, RNNs are recommended for offline simulation applications. Even though FNNs are recommended for online simulation purposes, RNNs still outperform FNNs in both online and offline simulation. Thus, it is evident that RNNs are the superior modelling strategy.

For future work it is recommended to explore more robust stopping conditions. The generalisation loss stopping condition utilised in this report has a tendency to terminate the optimisation procedure prematurely. Therefore, the ANNs optimised using this stopping condition are susceptible to underfitting. A further recommendation for future research is to obtain an experimental dataset that has a larger variety of nitrate inflow concentrations. This will improve the ANNs ability to accurately simulate the lutein production bioprocess as explained in Section 5.2.1. This is necessary if the ANNs are to be used in industry for real-time optimisation and control. Finally, another interesting topic of future research is the investigation of alternative machine learning techniques such as Gaussian Processes (GPs). GPs possess an extremely useful advantage over ANNs in that they can provide a measure of the uncertainty in their predictions. Furthermore, GPs can be constructed with significantly smaller experimental datasets than ANNs.

7. References

[1] D. J. MacKay, Sustainable Energy - without the hot air, Cambridge: UIT Cambridge Ltd., 2009.

[2] J. Houghton, Global Warming The Complete Briefing, New York: Cambridge University Press, 2009.

[3] L. Brennan and P. Owende, “Biofuels from microalgae—A review of technologies for production, processing, and extractions of biofuels and co-products,” *Renewable and Sustainable Energy Reviews,* vol. 14, no. 2, pp. 557-577, 2010.

[4] W. Chu, “Biotechnological applications of microalgae,” *IeJSME,* vol. 6, no. 126, pp. 24-37, 2012.

[5] D. Zhang, N. Xiao, K. Mahbubani, R.-C. E. del and V. Vassiliadis, “Bioprocess Modelling of Biohydrogen Production by Rhodopseudomonas palustris: Model Development and Effects of Operating Conditions on Hydrogen Yield and Glycerol Conversion Efficiency,” *Chemical Engineering Science,* vol. 130, pp. 68-78, 2015.

[6] S. Ho, Y. Xie, M. Chan, C. Liu, C. Chen, D. Lee, C. Huang and J. Chang, “Effects of nitrogen source availability and bioreactor operating strategies on lutein production with Scenedesmus obliquus FSP-3.,” *Bioresour. Technol.,* vol. 184, pp. 131-138, 2015.

[7] A. Kaczor and M. Baranska, Carotenoid Production by Bacteria, Microalgae, and Fungi., Oxford: John Wiley & Sons, 2016.

[8] J. Fabregas, A. Otero, A. Maseda and A. Dominguez, “Two-stage cultures for the production of Astaxanthin from Haematococcus pluvialis,” *Journal of Biotechnology,* vol. 89, no. 1, pp. 65-71, 2001.

[9] A. Alan, “Global Lutein market to witness a CAGR of 5.2 % during 2018-2024,” Energias Market Research , 25 April 2019. [Online]. Available: https://www.globenewswire.com/news-release/2019/04/25/1809676/0/en/Global-Lutein-market-to-witness-a-CAGR-of-5-2-during-2018-2024.html. [Accessed 27 April 2020].

[10] W. Y. Hong, H. S. Cheng and W. M. Te, “The Comparison of Lutein Production by Scenesdesmus sp. in the Autotrophic and the Mixotrophic Cultivation,” *Applied Biochemistry and Biotechnology,* vol. 164, pp. 353-361, 2011.

[11] Y. Xie, S. Ho, C. Chen, C. Chen, I. Ng, K. Jing, J. Chang and L. Y, “Phototrophic cultivation of a thermo-tolerant Desmodesmus sp. for lutein production: effects of nitrate concentration, light intensity and fed-batch operation.,” *Bioresour Technol.,* vol. 144, pp. 435-444, 2013.

[12] A. Franz, F. Lehr, C. Posten and G. Schaub, “Modeling microalgae cultivation productivities in different geographic locations - estimation method for idealized photobioreactors.,” *Biotechnology Journal,* vol. 7, no. 4, pp. 546-557, 2012.

[13] E. Del Rio-Chanona, N. R. Ahmed, D. Zhang, K. Jing and Y. Lu, “Kinetic Modeling and Process Analysis for Desmodesmus sp.,” *American Institue of Chemical Engineers,* vol. 63, pp. 2546-2554, 2017.

[14] E. Del Rio-Chanona, F. Fiorelli, D. Zhang, N. R. Ahmed, K. Jing and N. Shah, “An Efficient Model Construction Strategy,” *Biotechnology and Bioengineering,* vol. 114, no. 11, pp. 2518-2527, 2017.

[15] D. Zhang, P. Dechatiwongse, E. del Rio-Chanona, K. Hellgardt, G. C. Maitland and V. Vassilos, “Analysis of the cyanobacterial hydrogen photoproduction process via model identification and process simulation,” *Chemical Engineering Science,* vol. 128, pp. 130-146, 2015.

[16] M. Rahman, “DATA-DRIVEN MODELLING AND OPTIMISATION FOR SUSTAINABLE BIO-PRODUCTION,” The University of Manchester, Manchester, 2019.

[17] E. Del Rio-Chanona, Z. Dongda, Y. Xie, E. Manirafasha and K. Jing, “Dynamic Simulation and Optimization for Arthrospira platensis Growth and C-Phycocyanin Production,” *Industrial and Engineering Chemistry Research,* vol. 54, no. 43, pp. 10606-10614, 2015.

[18] E. Del Rio-Chanona, N. R. Ahmed, J. Wagner, Y. Lu, D. Zhang and K. Jing, “Comparison of physics-based and data-driven modelling,” *Biotechnology and Bioengineering,* pp. 1-12, 2019.

[19] S. Estrada-Flores, I. Merts, B. D. Ketelaere and J. Lammertyn, “Development and validation of “grey-box” models for refrigeration applications: A review of key concepts,” *International Journal of Refrigeration,* vol. 29, no. 6, pp. 931-946, 2006.

[20] A. Tch, “The mostly complete chart of Neural Networks, explained,” Towards Data Science, 4 August 2017. [Online]. Available: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464. [Accessed 10 May 2020].

[21] E. Bradford, A. M. Schweidtmann, D. Zhang, K. Jing and E. Del Rio-Chanona, “Dynamic modeling and optimization of sustainable algal production with uncertainty using multivariate Gaussian processes,” *Computers and Chemical Engineering,* vol. 118, pp. 143-158, 2018.

[22] C. E. Rasmussen and C. K. Williams, Gaussian Processes for Machine Learning, Massachusetts: MIT Press, 2006.

[23] K. Cherry, “An Overview of the Different Parts of a Neuron,” Very Well Mind, 7 April 2019. [Online]. Available: https://www.verywellmind.com/structure-of-a-neuron-2794896. [Accessed 19 May 2020].

[24] A. Burkov, The Hundred-Page Machine Learning Book, Andriy Burkov, 2019.

[25] J. Brownlee, “Difference Between a Batch and an Epoch in a Neural Network,” Machine Learning Mastery, 20 July 2018. [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/. [Accessed 27 April 2020].

[26] C. Nicholson, “A Beginner's Guide to LSTMs and Recurrent Neural Networks,” Pathmind, [Online]. Available: https://pathmind.com/wiki/lstm. [Accessed 27 April 2020].

[27] C. Olah, “Understanding LSTM Networks,” colahs blog, 27 August 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 27 April 2020].

[28] E. Del Rio-Chanona, E. Manirafasha, D. Zhang, Q. Yue and K. Jing, “Dynamic Modelling and Optimisation of Cyanobacterial C-phycocyanin Production Process by Artificial Neural Network,” *Algal Research,* vol. 13, pp. 7-15, 2016.

[29] M. Nielsen, Neural Networks and Deep Learning, California: Determination Press, 2014.

[30] A. Witek-Krowiak, K. Chojnacka, D. Podstawczyk, A. Dawiec and K. Pokomeda, “Application of response surface methodology and artificial neural network methods in modelling and optimization of biosorption process.,” *Bioresour Technol. ,* vol. 160, pp. 150-160, 2014.

[31] J. Brownlee, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,” Machine Learning Mastery, 3 July 2017. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Accessed 27 April 2020].

[32] V. Nigam, “Understanding Neural Networks. From neuron to RNN, CNN, and Deep Learning,” Towards Data Science , 11 September 2018. [Online]. Available: https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90. [Accessed 10 May 2020].

[33] L. Prechelt, “Automatic early stopping using cross validation: quantifying the criteria,” *Neural Networks,* vol. 11, no. 4, pp. 761-767, 1998.

[34] J. Brownlee, “How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras,” Machine Learning Mastery, 9 August 2016. [Online]. Available: https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/. [Accessed 27 April 2020].

[35] T. Hastie, R. Tibshirani and J. Friedman, The Elements of Statistical Learning, New York: Springer Science+Business Media, 2017.

[36] J. Brownlee, “How to Configure the Learning Rate When Training Deep Learning Neural Networks,” Machine Learning Mastery, 23 January 2019. [Online]. Available: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/. [Accessed 27 April 2020].

8. Appendices

Average Mean Squared Error (MSE) Calculation

Below is an example of how the average MSE which is displayed on the hyperparameter optimisation graphs is calculated.

The testing dataset consists of two experiments: Test Set 1 and Test Set 2. During testing, the ANNs simulate the biomass concentration, nitrate concentration and lutein production over 144 hours for both Test Set 1 and Test Set 2. Consequently, there are a total of 6 simulations produced each time testing occurs. Each simulation has an associated MSE value which is calculated using Equation 8. This is done programmatically in Python; no manual calculations are performed. Therefore, there are now 6 MSE values associated to the 6 simulations produced by the ANN:

- MSE_{X1}: MSE for biomass concentration predictions on Test Set 1
- MSE_{N1}: MSE for nitrate concentration predictions on Test Set 1
- MSE_{Lu1}: MSE for lutein production predictions on Test Set 1
- MSE_{X2}: MSE for biomass concentration predictions on Test Set 2
- MSE_{N2}: MSE for nitrate concentration predictions on Test Set 2
- MSE_{Lu2}: MSE for lutein production predictions on Test Set 2

These MSE values are then averaged to obtain a single average MSE value which quantifies the ANNs performance over the 6 simulations:

$$Average\ MSE = \frac{MSE_{X1} + MSE_{N1} + MSE_{Lu1} + MSE_{X2} + MSE_{N2} + MSE_{Lu2}}{6}$$

FNN and RNN Parameter Calculation

An FNN with a 2-8 configuration consists of 63 parameters. The calculations performed to arrive at this value are shown below.

In the first hidden layer, each neuron receives 5 input values from the input layer. Therefore, each neuron consists of 5 weights corresponding to each input value plus an additional bias value. Hence, each neuron consists of 6 parameters in the first hidden layers. Therefore, the number of parameters in the first hidden layer can be calculated as follows:

$$2\ neurons\ \times 6\ parameters\ per\ neuron = 12\ parameters$$

In the second hidden layer, each neuron receives 2 input values from the first hidden layer (as there are 2 neurons in the first hidden layer). Therefore, each neuron consists of 2 weights corresponding to each input value plus an additional bias value. Hence, each neuron consists of 3 parameters in the second hidden layer. Therefore, the number of parameters in the second hidden layer can be calculated as follows:

$$8\ neurons \times 3\ parameters\ per\ neuron = 24\ parameters$$

The output layer consists of 3 neurons and each neuron receives 8 input values from the second hidden layer (as there are 8 neurons in the second hidden layer). Therefore, each neuron consists of 8 weights corresponding to each input value plus an addition bias value. Hence, each neuron contains 9 parameters in the output layer. Consequently, the number of parameters in the output layer can be calculated as follows:

$$3\ neurons \times 9\ parameters\ per\ neuron = 27\ parameters$$

$$\rightarrow Total\ Number\ of\ Parameters = 12 + 24 + 27 = 63$$

An RNN with 1 hidden layer and 6 hidden neurons consists of 93 parameters. The calculations performed to arrive at this value are shown below.

In the hidden layer, each neuron receives 5 input values from the input layer. Therefore, each neuron consists of 5 weights corresponding to each input value plus an additional bias value. Furthermore, each neuron receives a hidden state vector which contains a weight for each hidden neuron in the hidden layer. Consequently, in this case, each neuron consists of an additional 6 weights. Hence, the total number of parameters per neuron in the hidden layer is 12. Therefore, the number of parameters in the hidden layer can be calculated as follows:

$$6\ neurons\ \times 12\ parameters\ per\ neuron = 72\ parameters$$

The output layer consists of 3 neurons and each neuron receives 6 input values from the hidden layer (as there are 6 neurons in the hidden layer). Therefore, each neuron consists of 6 weights corresponding to each input value plus an additional bias value. Hence, each neuron contains 7 parameters in the output layer. Consequently, the number of parameters in the output layer can be calculated as follows:

$$3\ neurons \times 7\ parameters\ per\ neuron = 21\ parameters$$

$$\rightarrow Total\ Number\ of\ Parameters = 72 + 21 = 93$$

<u>**Hidden Neurons and EPOCHS Performance Improvement Calculations**</u>

<u>FNN</u>

For the FNN, the best performing hidden neuron configuration is 4-8 accompanied by an optimal EPOCH value of 152. The average MSE for this optimal hyperparameter configuration is 0.0046. The worst performing hidden neuron configuration is 2-8 accompanied by an optimal EPOCH value of 28. The average MSE for this worst performing hyperparameter configuration is 0.0084. Consequently, the performance improvement provided by optimising the hidden neurons and EPOCHS is calculated as follows:

$$Performance\ Improvement = \left(\frac{0.0084 - 0.0046}{0.0084}\right) \times 100\% = 45.2\%$$

<u>RNN</u>

For the RNN, the best performing value for the number of hidden neurons is 15 accompanied by an optimal EPOCH value of 30. The average MSE for this optimal hyperparameter configuration is 0.0035. The worst performing value for the number of hidden neurons is 2 accompanied by an optimal EPOCH value of 85. The average MSE for this worst performing hyperparameter configuration is 0.0082. Consequently, the performance improvement provided by optimising the hidden neurons and EPOCHS is calculated as follows:

$$Performance\ Improvement = \left(\frac{0.0082 - 0.0035}{0.0082}\right) \times 100\% = 57.3\%$$

<u>**Learning Rate Performance Improvement Calculations**</u>

<u>FNN</u>

For the FNN, the best performing value of learning rate is 0.006 which yields an average MSE of 0.0050. The worst performing value of learning rate is 0.07 which yields an average MSE of 3.26. Consequently, the performance improvement provided by optimising the learning rate is calculated as follows:

$$Performance\ Improvement = \left(\frac{3.2552 - 0.0050}{3.2552}\right) \times 100\% = 99.8\%$$

RNN

For the RNN, the best performing value of learning rate is 0.0009 which yields an average MSE of 0.0039. The worst performing value of learning rate is 0.6 which yields an average MSE of 0.0570. Consequently, the performance improvement provided by optimising the learning rate is calculated as follows:

$$Performance\ Improvement = \left(\frac{0.0570 - 0.0039}{0.0570}\right) \times 100\% = 93.2\%$$

Batch Size Performance Improvement Calculations

FNN

For the FNN, the best performing value of batch size is 300 which yields an average MSE of 0.0059. The worst performing value of batch size is 40 which yields an average MSE of 0.016. Consequently, the performance improvement provided by optimising the batch size is calculated as follows:

$$Performance\ Improvement = \left(\frac{0.0160 - 0.0059}{0.0160}\right) \times 100\% = 63.1\%$$

RNN

For the RNN, the best performing value of batch size is 8 which yields an average MSE of 0.0038. The worst performing value of batch size is 500 which yields an average MSE of 0.0123. Consequently, the performance improvement provided by optimising the batch size is calculated as follows:

$$Performance\ Improvement = \left(\frac{0.0123 - 0.0038}{0.0123}\right) \times 100\% = 69.1\%$$

Python Code

All of the python code written during the research project can be found on GitHub: https://github.com/Mostafizor/Bioprocess-Simulation-using-Machine-Learning

Optimisation and Training Time

The time taken to fully optimise and train a 2 hidden layer FNN is approximately 30 minutes on an Intel Core i5 8th generation CPU. The time taken to optimise and train a 2 hidden layer RNN is approximately the same.