Tutor

Dr. David Curcó Cantarell
Departament d'Enginyeria Química I Química Analítica

Grau en Enginyeria Química

Treball Final de Grau

Application of neural networks to model double tube heat exchangers.

Max Gómez Cáceres

June 2023

UNIVERSITAT DE BARCELONA

CONTENTS

SUMMARY I

RESUM III

SUSTAINABLE DEVELOPMENT GOALS V

1. INTRODUCTION 1
1.1. Artificial intelligence 3
1.2. Machine Learning 3
1.3. Deep learning 4
1.3.1. Activation Functions 5
1.3.2. Hyperparameters 6
1.3.3. Training 7
1.3.4. Bias-Variance Tradeoff 8
1.4. Recurrent neural networks 11
1.5. Long-short term memory neural networks 13

2. OBJECTIVES 16

3. METHODS 17
3.1. Software Tools 17
3.2. Simulation Introduction 19
3.3. Simulation strategy 19
3.3.1. System Study 21
3.3.2. Data Storage 22
3.4. Data Pre-processing 23
3.4.1. Loading and Checking 23
3.4.2. Data Segmentation and Standardization 24
3.4.3. Data Windowing 24
3.5. Neural network model 27

4. NEURAL NETWORK DESIGN PROCESS AND RESULTS 28
4.1. Quality measurements 28
4.2. Proof of Concept 29
4.3. Exploratory Design 30
4.3.1. 30-second horizon prediction model 30

4.3.2. 60-second prediction horizon model 33
4.3.3. 90-second prediction horizon model 35
4.4. Detail Engineering and Testing 38
4.4.1. Testing Approach 38
4.4.2. Results 39
5. CONCLUSIONS 42
REFERENCES AND NOTES 44
ACRONYMS 47
APPENDICES 49
APPENDIX 1: NEURAL NETWORK PARAMETERS 51
APPENDIX 2: STANDARDIZATION CONSTANTS 55
APPENDIX 3: TEST RESULTS 57

SUMMARY

Artificial Intelligence is experiencing dramatic growth in recent times. AI models such as ChatGPT have become controversial topics as they continously transform our world. Nevertheless, the true nature of AI is still widely not yet understood by society. Nowadays, Artificial Intelligence is still seen by many as an obscure and foreign concept, even mysterious and threatening. However, this couldn't be further from the truth. At their essence, they are just mathematical tools which rely on centuries-old knowledge: algebra and calculus.

In this project, a neural network model has been created to solve a chemical engineering problem, the predictive model of a double tube heat exchanger.

This model is a neural network that predicts future system outputs (inner stream output temperature) from the past values of the input variables of the system (inner and outer streams input temperatures and outer stream flow rate).

The data used to train the model was obtained in a simulation written in the Python programming language. Afterwards, the optimal design parameters of the neural network were found experimentally by training different models and testing their performance. This was done in three stages: a proof of concept, a general design stage and a detailed design stage.

The model has been successful in predicting the future state of the system with high exactitude while being circa. 3000 times faster than a conventional simulation.

Keywords: artificial intelligence, neural networks, simulation, dynamics, Python, software architecture

RESUMEN

La Inteligencia Artificial está experimentando un crecimiento dramático recientemente. Los modelos de IA como ChatGPT se han convertido en temas controvertidos mientras transforman continuamente nuestro mundo. Sin embargo, la sociedad aún no comprende la verdadera naturaleza de la IA. Hoy en día, la Inteligencia Artificial todavía es vista por muchos como un concepto oscuro y extraño, incluso misterioso y amenazante. Sin embargo, esto no podría estar más lejos de la realidad. En esencia, son solo herramientas matemáticas que se basan en conocimientos centenarios: álgebra y cálculo.

En este proyecto se ha creado un modelo de red neuronal para resolver un problema de ingeniería química, el modelo predictivo de un intercambiador de calor de doble tubo.

Este modelo es una red neuronal que predice los valores de salida futuros del sistema (temperatura de salida de la corriente interna) a partir de los valores pasados de las variables de entrada del sistema (temperaturas de entrada de las corrientes interna y externa y caudal de la corriente externa).

Los datos utilizados para entrenar el modelo se obtuvieron en una simulación escrita en el lenguaje de programación Python. Posteriormente, los parámetros óptimos de diseño de la red neuronal se encontraron experimentalmente entrenando diferentes modelos y probando su rendimiento. Esto se realizó en tres etapas: una prueba de concepto, una etapa de diseño general y una etapa de diseño detallado.

El modelo ha predicho exitosamente el estado futuro del sistema con exactitud y aproximadamente 3000 veces más rápido que una simulación convencional.

Palabras clave: inteligencia artificial, redes neuronales, simulación, dinámica, Python, arquitectura de software

SUSTAINABLE DEVELOPMENT GOALS

The impact of this project is mainly centred on the “Prosperity” and “Planet” sections of the Sustainable Development Goals. The employment of artificial intelligence in industry is one of the drivers of the fourth industrial revolution or “Industry 4.0”. This new paradigm of industry employs new advances in informatics to improve manufacturing efficiency. More efficient factories waste less raw material and electricity, which produces both economic and ecological advantages. Innovation in industry is a core tenet of Goal 9: Industries, Innovation, and Infrastructure.

The efficiency of an industrial process is heavily influenced by its control systems. The predictive nature of the neural networks allows action to be taken before the system enters a wasteful or dangerous state. Improved control strategies directly translate into a reduction of electricity use (Goal 7: Affordable and Clean Energy), and raw material wastage (Goal 12: Responsible Consumption and Production). Additionally, predictive models can detect potential future unsafe conditions and enable safety actions to be taken earlier than with traditional feedback control. In some cases, such as in a reactor runaway, taking proper safety measures early enough is vital to preventing an accident. Chemical industry accidents are devastating to the people and the environment, so preventing them is crucial.

In cases where a simulation-based predictive system was already in place, a neural network can learn from the simulation and replace it. Neural network models require less computational power than simulations, which is directly correlated to a reduction in electricity use. Such actions to increase energy efficiency are related to the Goal 7: Affordable and Clean Energy. The better use of energy and resources translates directly into a reduction in greenhouse gas emissions.

Extracting and transporting resources and generating electricity have associated CO2 emissions. Consequently, using resources and power in a more efficient manner translates directly into a reduction in carbon dioxide emissions. Since CO2 is a main driver in climate change, reducing its emissions is vital to stop global warming and the damage it is causing to the people and environment. This furthers Goal 13: Climate Action.

1. INTRODUCTION

The dynamic systems that govern double tube heat exchangers make their control a challenging task. The core problem is that in these systems, the gain, dead time and time constant vary depending on flow rate. This makes these systems strongly nonlinear, so they cannot be correctly described by conventional linear partial differential equations.

These heat exchangers also have long dead times and a tendency to exhibit oscillatory behaviour when controlled. Dead times are known to make systems harder to control, especially so in cases where these delays are long and even more so if they are not constant. Double tube heat exchangers have both issues.

Classical PID uses feed-back control, which in this case would make decisions based on the output temperature. Since the effects of disturbances and control actions on the output temperature are delayed, the controller is always making decisions based on outdated information. This makes PID controllers unsuitable for these systems.

To successfully control such systems, feed-forward control elements need to be used. [36]

Feed-forward control relies on measuring disturbances and making a prediction of their effect before they affect the system. In a heat exchanger, this would be done by measuring the temperature and flow rate of the streams that feed into the exchanger and using them to predict the future output temperature.

Feed-forward control requires a mathematical model of the plant to determine what control actions should be taken. This model calculates the outputs of the system depending on its inputs.

One example of feed-forward control strategy is Model Predictive Control (MPC). In this strategy, the mathematical model of the system is used to predict its future behaviour and optimize control actions over a finite time horizon.

Consequently, the performance of the control strategy relies on the quality of the model. This model must be accurate and fast to calculate since it is necessary to perform iterative calculations during optimization.

The model of these systems is obtained via system identification, which is more challenging in nonlinear systems such as double tube heat exchangers. In these cases, the model may be based on fundamental mass and energy balances. However, this requires simulating these balances, which is computationally intensive and, in some cases, too slow to allow iterative optimization. As an alternative, empirical approaches have been studied. Their objective is to fit the empirical data into a function that is faster to calculate than the simulation. In cases where the balances cannot be calculated, only empirical approaches can be used.

A novel approach that is steadily being developed is using artificial intelligence as empirical models. [37][38]

These techniques consist of creating a surrogate model (the artificial neural network) which is in essence a regression from empirical data. The model behaves like a black box that disregards the physical balances of the system it is modelled after. The result is a function that correlates inputs to outputs directly, a sort of black-box system identification transfer function.

1.1. ARTIFICIAL INTELLIGENCE

Artificial intelligence is the simulation of human intelligence by a machine. It refers to the development of computer systems that can perform tasks that usually require human intelligence such as learning, reasoning, problem-solving and decision-making. [1]

The study of artificial intelligence began in 1956, when the term "artificial intelligence" was coined by John McCarthy, an American computer scientist, at The Dartmouth Conference.[2].

1.2. MACHINE LEARNING

Machine learning is the use of algorithms that allow AI models to learn from data and improve their performance on specific tasks. [3] Training is done by optimizing the model parameters to improve its performance. A successfully trained model would be able to react accordingly to new inputs.

Three main approaches have been developed for machine learning: supervised learning, unsupervised learning, and reinforcement learning.

In supervised learning the training data is labelled with the correct answer. The difference between the data and the result predicted by the method is then minimized using optimization algorithms such as gradient descent.

Supervised learning has two main applications: classification and regression.

Classification is used to assign inputs into discrete groups. They are assigned into categories or labels such as {True, False} or {0,1,2,3}.

Regression is used to find correlations between variables to predict a continuous output variable. It follows the same principles as conventional linear regression.

In unsupervised learning, the data is unlabelled. The objective is to find hidden patterns in the data, such as clusters of data points.

Reinforcement learning is performed in an interactive dynamic environment. The program must achieve a goal and it is given feedback, being rewarded “points” on a function which it tries to maximize. [4]

1.3. DEEP LEARNING

Deep learning is a subset of machine learning. It focuses on the development of artificial neural networks, a type of machine-learning algorithm that is based on the structure of the human brain.

Perceptrons are the building blocks of neural networks. Neural networks have perceptrons organized in layers: an input layer, an output layer, and an arbitrary number of hidden layers.

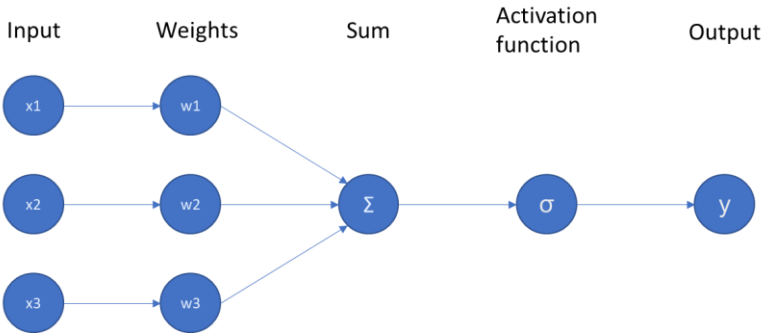

Figure 1. An illustration of a perceptron

Perceptrons can distinguish between two classes of inputs. They consist of input nodes, which receive input signals, and a single output node, which computes a weighted sum of the inputs and applies an activation function to produce an output signal. Each input has a weight that impacts the effect it has on the output. These weights determine the behaviour towards inputs and its capability to properly classify them. The optimal weights can't be known beforehand and must be obtained using optimization algorithms.

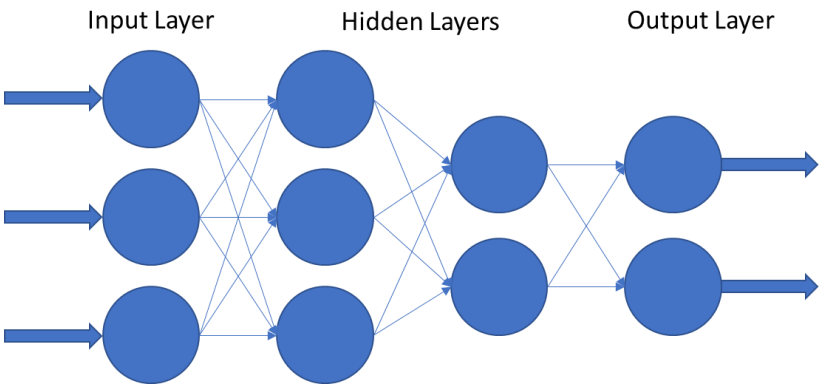

Figure 2. An illustration of a multilayer perceptron. Each circle is a perceptron.

Multiple perceptrons can be connected in parallel (where multiple perceptrons share inputs) and in series (the input of a perceptron is the output of another perceptron) to produce a multilayer feed-forward perceptron. Multilayer perceptrons are the simplest type of neural network. The individual perceptrons of the multilayer perceptron are often called neurons.

Multilayer feed-forward perceptrons with at least 2 layers of depth are in theory universal function approximators [14]. This means that these algorithms can approximate any arbitrary function if they have enough neurons. Depending on the activation function used they can perform either regression or classification, while single perceptrons can only perform classification. [5]

Consequently, a multilayer perceptron can be created to make a regression or a prediction from data which follows functions that are not solvable analytically, such as the partial differential equations governing the behaviour of dynamic systems.

However, the number of required neurons may be impractical in some cases, which induced the development of diverse architectures which incorporate features such as feed-back loops or convolution.

1.3.1. Activation Functions

Activation functions define an output based on an input or set of inputs. The impact of each input is determined by their weight, a parameter which is optimized during training. Non-linear functions are used because neural networks with linear activation functions do not exhibit universal function approximation capabilities. [6]

A widespread activation function is the Rectified Linear Unit (ReLU) which is a ramp function defined as: $f(x) = \max(0, x)$.

Also common are sigmoid-like activation functions such as the logistic $f(x) = (1 + e^{-x})^{-1}$ and the hyperbolic tangent $f(x) = {(e^{x} - e^{-x})}/{(e^{x} + e^{-x})}$

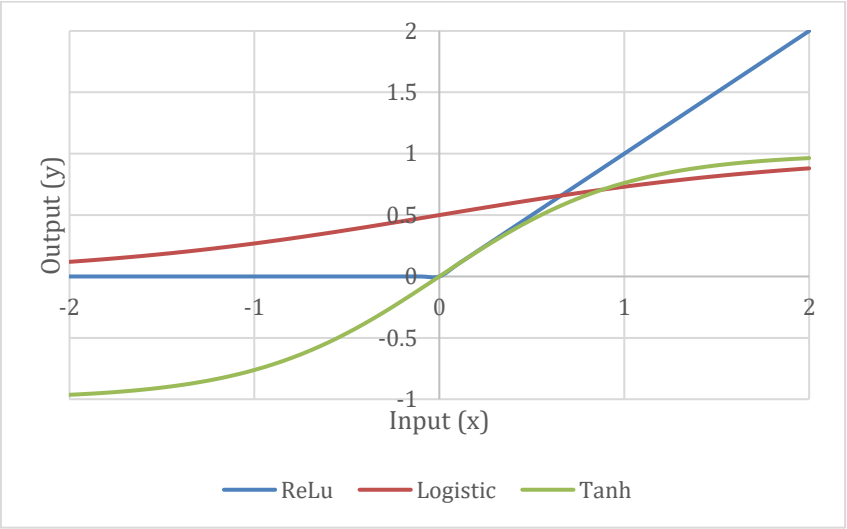

Figure 3. Common activation functions

1.3.2. Hyperparameters

Hyperparameters are the design parameters of the neural network that describe its structure and control the learning process. They are not optimized during training and remain constant.

The structure is described by hyperparameters like number of neurons, number of layers and activation functions.

The learning process is described by the learning rate (the rate at which weights are changed by the optimizer), which optimizers are used, and the number of epochs (the number of times the training data is inputted into the network while training), among others. Some optimizers such as Adam automatically adjust the learning rate for each layer, thus incorporating the hyperparameter into the training. [7]

However, hyperparameters can also be optimized (hyperparameter optimization). However, using an optimization algorithm such as gradient descent or grid search would require the whole training process to be repeated each time a hyperparameter is changed. The optimization of all hyperparameters is unfeasibly time-consuming due to the "combinatorial explosion" (the number of possible combinations increases exponentially with the number of variables). Consequently, it is common that many hyperparameters are chosen based on heuristics and only a few most significant ones are optimized.

1.3.3. Training

To train neural networks they are usually first initialized with small random weights and zero bias. Training neural networks require data, which must be split into three segments: training, testing data and validation data. Training and validation data are both used to fit the model: The "train" set is used to train the model (optimization with backpropagation), and the "validation" set is used to fine-tune the properties of the model with hyperparameter optimization. [8]. The "test" set is used to evaluate the performance of the model on completely unseen data.

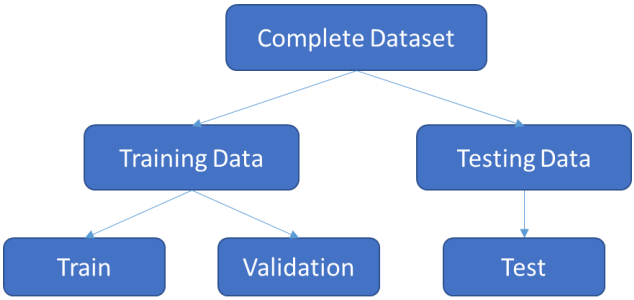

Figure 4. Data segmentation

First, the training data set is divided into batches. The first data entry of the first is inputted into the algorithm and the outputs are produced. Then, the loss is calculated out of the deviation from the expected value using a loss function. An example of a loss function is the squared loss, which is also used in the least-squares method of linear regression.

The process is repeated for all data entries in the batch and the cost function is calculated, which is often the average of all losses of the batch.

The objective is to minimize the cost function by iteratively adjusting the appropriate weights and biases for all neurons. This optimization is performed by optimizers based on gradient descent such as SGD (Stochastic gradient descent) or Adam (adaptive moment estimation).

The adjustments are performed with backpropagation. Backpropagation works by calculating the gradient of the cost function with respect to each weight in the network. This gradient is a vector in the direction of the highest slope. The negative of this vector points towards the path that minimizes the function the most.

First, this gradient is calculated for the last row of neurons, finding the direction these weights should change. Since the output of a neuron is determined by the outputs of the preceding neurons and the weights of the connections, the optimal path for these preceding outputs also must be calculated. These outputs also depend on the weights and outputs of the preceding neurons, the process is repeated until the first layer of neurons is reached. This gradient is then calculated using Leibniz's chain rule of derivatives. [9][10][11]

The resulting gradient is a vector of N dimensions, one for each weight to optimize. Afterwards, a step is performed in this direction. The size of the step is determined by the learning rate.

Higher learning rates produce larger steps, making the system in some cases converge faster but risk overstepping the solution (the system would fail to converge). Smaller learning rates lower the risk of overstepping, (more robust training) but at the cost of being slower.

This process is then repeated for all batches in the training data set. A pass through all batches of the training data is an epoch. After all batches have been passed through, the performance of the neural network is tested with the validation data and the next epoch begins. The process is then repeated until it reaches a pre-defined number of epochs or until it is interrupted by an early-stopping algorithm.[12]

1.3.4. Bias-Variance Tradeoff

The bias-variance trade-off is the relationship between the ability of a model to fit the training data (bias) and its ability to generalize to new, unseen data (variance). [13]

This is the reason why the dataset is divided into training, validation, and testing data. The model is trained only with training data and validation data (both of which will be referred to as training data in the context of overfitting and underfitting), keeping testing data as unknown new data to test the model with. A good model should be accurate both in the training data and in the testing data.

During training, the neural network is expected to find relationships between data points in the training data which can be then applied to new data. This can fail in two different ways: underfitting and overfitting.

Underfitting is the phenomenon in which the model is unable to capture all the complex patterns in the training data, which produces a high bias. These models are often exceedingly simple and perform poorly on training and test datasets.

Overfitting is the opposite phenomenon. It is caused by models with too complex structures, where they just "memorize" the training data instead of finding patterns. This produces a high variance. These models perform very well on training datasets (as they have memorized them) but poorly on testing datasets (as they have not learned anything).

These phenomena can also be found in conventional regression, where it is easier to illustrate. Here, the R-squared value is analogous to the inverse of the training error or loss.

As an example, suppose a parabola with the following function:

$y = 0.4x^2 - 2x + 3 + noise$ where noise is a random number between -0,5 and +0,5.

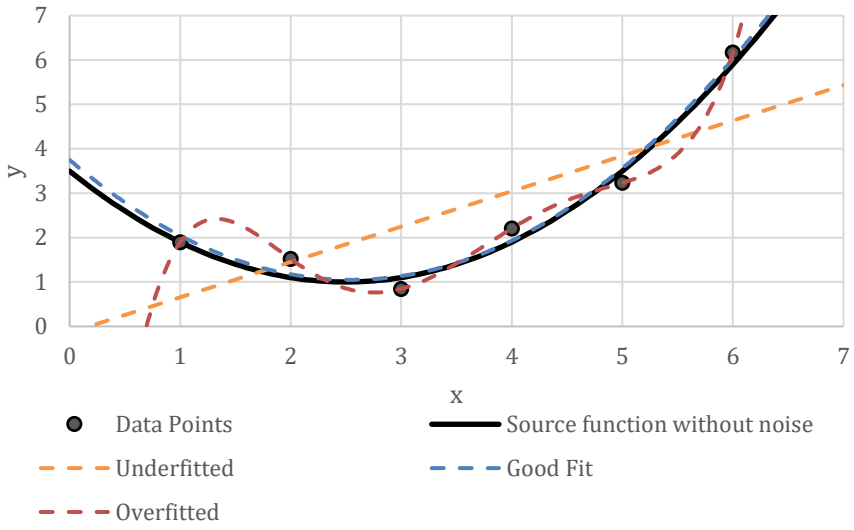

Figure 5. Underfitted and overfitted polynomial regression models

The under-fitted model ($y = 0{,}7962x + 0{,}1409$) is too simple to capture changes in slope and is thus unable to reproduce a parabola. It has a R-squared value of 0,6159.

The overfitted model is a 5th-degree polynomial, the highest possible degree with 6 data points.

$(y = 0{,}0776x^5 - 1{,}3595x^4 + 8{,}941x^3 - 26{,}789x^2 + 35{,}392x - 14{,}366)$

This model captures all noise, it is unable to discern between the underlying trend and the random noise. Consequently, it has an R-squared value of 1. The training error is 0, as the model has a theoretically perfect, unbiased fit. However, it would fail to predict the value at x = 1,5 despite having a perfect fit at x = 1 and x = 2.

The good fit model ($y = 0{,}4169x^2 - 2{,}1222x + 3{,}75$) has a higher training loss than the overfitted model, an R-squared value of 0,9761, lower than the 1 of the overfitted model. However, this is not a negative property. This bias is caused by the ability of the model to ignore the noise and only focus on the underlying trend.

1.4. RECURRENT NEURAL NETWORKS

The neural network architecture used in this project is Long Short-Term Memory, a type of Recurrent Neural Network.

Recurrent Neural Networks (RNN) are a type of neural network that is best used to find patterns in sequences of data. Sequential data include human language, genes, stock market prices, industrial sensor data…

The main difference between feed-forward neural networks ("basic" neural networks) and RNNs is that RNNs include feed-back transfers of information. Just like normal perceptrons, RNN (including LSTM) neurons can be joined in series and in parallel. The hidden layer may contain an arbitrary number of neurons and layers of neurons.

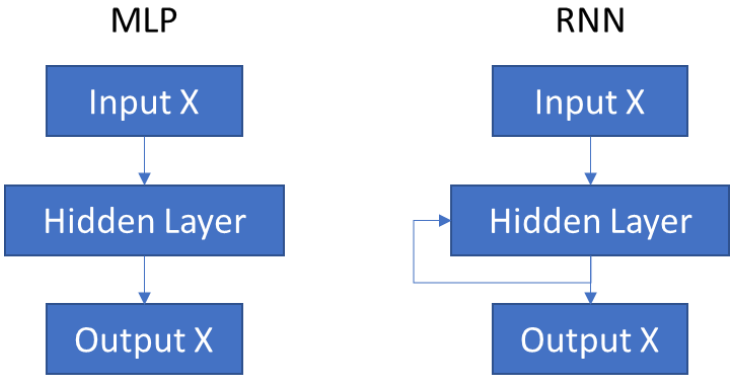

Figure 6. Comparison between a multi-layer perceptron (feed-forward) and a recurrent neural network (feed-back)

Feed-forward neural networks do not have memory, and thus they can only make predictions on the present conditions. The inclusion of feed-back loops allows the model to remember past states and include them into the predictions. [12]

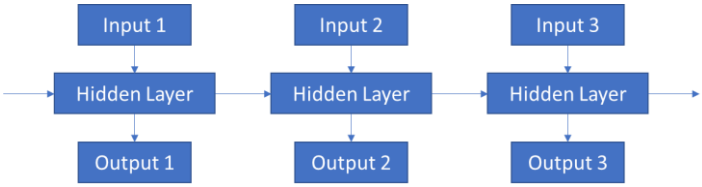

Figure 7. Unfolded Recurrent Neural Network

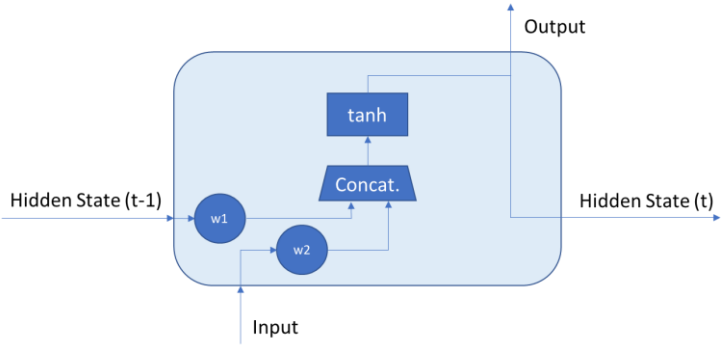

Figure 8. RNN Hidden Layer

However, RNNs exhibit phenomena known as “vanishing gradient” and “exploding gradient”. These problems happen during backpropagation. Backpropagation in an RNN is done by performing backpropagation on each unit of the unfolded recurrent neural network. It is “backpropagation through time”. The gradients obtained by backpropagation in the first units must propagate through all units of the sequence until the current time step. After each layer, small gradients shrink and end up vanishing after a few time steps. The model has “forgotten” what happened in that first unit. [26] The opposite is exploding gradient, which happens with large gradients that increase through time until the model becomes unstable.[24] [25]

1.5. LONG-SHORT TERM MEMORY NEURAL NETWORKS

Long Short-Term Memory is a variety of RNN built to avoid the vanishing gradient problem which is common in basic RNN since they allow the gradient to flow unchanged. However, they are still vulnerable to the less common exploding gradient problem. Research is being conducted on different algorithms to reduce this issue [31]. Nevertheless, only one instance of exploding gradient happened during the training of all the models of this study so there was no need to apply these algorithms.

This architecture keeps the RNNs ability to capture long-term dependencies. As a type of RNN, LSTM are used in sequential and time-series data, such as stock price prediction, weather prediction, and language translation (Google Translate). [25] [28]

It also has applications in the field of control and robotics, such as the predictive control of a corn processing plant [30] and the development of robotic hands [29].

The main difference between LSTM and RNN is that LSTM cells incorporate memory cells. Each LSTM neuron has two streams of data (called "states"), the hidden state (the "output" of the neuron and an input of the next cell, which works the same as in RNN) and the cell state (the output of the memory cell of this neuron and input of the memory cell and the next neuron).

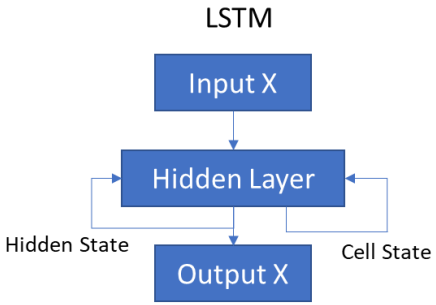

Figure 9. Folded architecture of a LSTM neural network

LSTM use gates to control the information that enters and leaves the memory cell. These gates allow data to flow unmodified and with no impact on the cell output unless these gates are "open". This behaviour makes the memory cell act like a "long-term memory", while the hidden state behaves like a "short-term memory". This is the namesake of this architecture "Long Short-Term Memory".

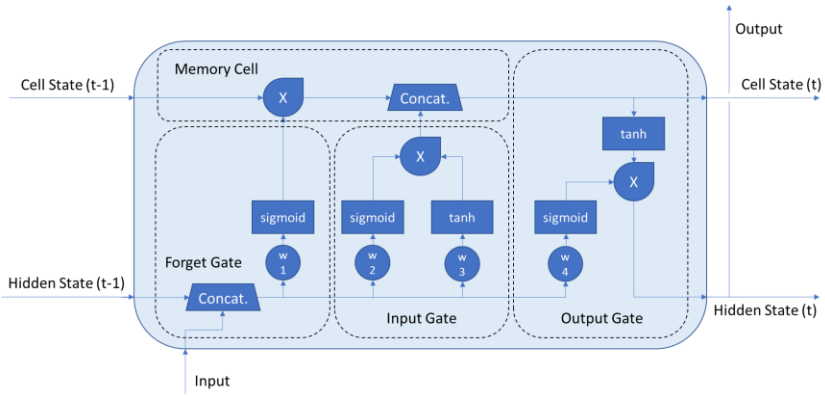

Figure 10. LSTM Hidden Layer architecture

The gates consist of sigmoid and hyperbolic tangent activation functions. The sigmoid activation functions control the gates. They have a range of (0,1), where an output of 0 represents a closed gate and 1 an open gate. The behaviour of all three gates is controlled by the current input and the hidden state of the previous cell. The hyperbolic tangent functions are the "normal" activation functions that process inputs into outputs.

The Forget Gate has the task of deleting the content of the memory cell ("forgetting"). The hidden state of the previous neuron and the current input are concatenated and multiplied by a weight (w1). By consequence, the choice of how much information to delete depends on both the input and the hidden state. If the output of the sigmoid function is 0, this 0 is multiplied by the Cell State of the previous cell. The result of this multiplication is 0, deleting its contents. Results between 0 and 1 allow for partial deletion of the memory cell.

The Input Gate has the task of selectively adding information into the memory cell. The choice of how much information to add is done by a sigmoid activation function like the one in the forget gate. The actual information added into the memory cell is processed by the hyperbolic tangent function. Both outputs are multiplied together.

If the result of the sigmoid function is 0, the result of the hyperbolic tangent function is discarded, and the memory cell is not modified. On the contrary, a sigmoid output of 1 would allow the result of the tanh function to be completely added into the memory cell.

The Output Gate selectively returns information from the memory cell to the hidden state, which is the output of the neuron. This gate can withhold the information in the output cell until a point in the future. This behaviour allows LSTM neural networks to work with delayed systems. This gate is also controlled by the input and a hidden state, which means that the time this information is stored (the dead time) depends on the input. This would in theory allow the neural network to adapt to different dead times, which depend on the input.

LSTM layers are often composed of multiple "stacked" LSTM units. Instead of having just one cell for each time step, there is an arbitrary number of them. However, the general structure remains the same as seen in Figure 10, but the hidden states, cell states, weights and outputs become vectors instead of scalars.

All units receive the same inputs but produce different outputs, which adds a dimension to these outputs.

A LSTM layer with only one unit returns a vector $Output = (y_{t1},\ y_{t2},\ y_{t3})$

A LSTM layer with more than one unit returns a matrix, for example a layer with two units would return $Output = \begin{bmatrix} y_{t1,u1} & y_{t2,u1} & y_{t3,u1} \\ y_{t1,u2} & y_{t2,u2} & y_{t3,u2} \end{bmatrix}$

The number of units determines the capacity and complexity of the layer. This is analogous to the degree of the polynomial used in the polynomial regression example. More layers allow capturing more complex patterns but at the cost of increased computational costs and higher risk of overfitting.

2. OBJECTIVES

The objectives of this project are divided into two main sections: research and application.

First, bibliographic research into neural networks with emphasis on their underlying mathematical characteristics. The purpose of this research is to demystify artificial intelligence and learning how it really operates from a more scientific perspective.

Second, developing a neural network to solve a chemical engineering problem. Especially, the goal was to provide a solution to a problem that was challenging to solve with conventional methods.

The problem to be solved is the modelling of a double tube heat exchanger. A model that predicts the future state of the system from past disturbances is needed to apply a feed-forward loop. The objective is to create a neural network that predicts with reasonable accuracy and speed the effects any disturbance would have on the future state of the system. In essence, the neural network is desired to behave like an empirical, time-domain version of a transfer function.

It is desirable to obtain a model that predicts far enough into the future to allow seeing the whole dynamic response to most disturbances, from the moment of the disturbance to at least the point at which the system reaches a stationary state.

To achieve these objectives, a simulation must be performed which produces compatible data that the model can learn and improve from.

Afterwards, the model must be designed. Finding the appropriate design parameters is done mainly in an experimental fashion. As such, experiments are to be performed to optimize these parameters.

3. METHODS

3.1. SOFTWARE TOOLS

The programming language Python was used to develop all software programs of this Project.

Python was chosen because it is the leading programming language in Computer Science applications in Science and Engineering. It is also the first programming language in terms of general popularity according to the TIOBE Community Index. [15]

Python is a free high-level language which focuses on readability and ease of use. Consequently, the user can focus on writing complex applications and algorithms quickly. [16]

However, this abstraction comes at a cost. Since it is not possible to optimize memory management, programs written in Python are usually slower than those written in lower-level languages such as C.

Another important advantage of Python is its very rich free library environment. Libraries are collections of pre-written functions that can be downloaded and used in programs at the users' discretion. Once downloaded, they can be loaded into the program by adding the line "import (library) as (abbreviation)" at the top of the program. Afterwards, they can be integrated seamlessly into the code as they behave like a regular user-written function. [17]

These functions often perform more computationally complex functions by calling a function written and compiled in a lower-level language, often C. In consequence, they allow the user to benefit from the speed of lower-level languages while maintaining the simplicity and readability of Python.

The Python libraries used in this project are free to download.

Libraries Used

NumPy: Adds the “array” data structure, which is absent from base Python. This data structure is common in lower-level languages such as C or C++. They are similar to Python lists, but their length is fixed. Arrays are a faster alternative to lists when working with vectors and matrices. It also includes additional mathematical functions such as sine and cosine. [18]

Pandas: Adds the “DataFrame” data structure, which is very similar to Excel spreadsheets. This library also allows the user to read and write Excel files in several formats such as “.csv” and “.xlsx”. Consequently, Pandas is used frequently in data analysis and data science applications. [19]

Matplotlib: Graphical library that allows plotting NumPy arrays and Pandas dataframes directly. [20]

Scikit-learn: Machine learning library built on NumPy, SciPy and matplotlib that provides tools to assist in data processing and modelling. [21]

TensorFlow: Artificial intelligence library which provides the “tensor” data structure. Tensors are in essence arrays optimized for deep learning purposes. This library also allows the usage of the GPU to accelerate the training of the models using CUDA. [22]

Keras: Deep learning library associated with TensorFlow. Keras provides the user with all the tools needed to build a neural network, such as optimizers, activation functions and loss functions. [23]

The code was written using Visual Studio Code, a free code editor. The also free “Python” plug-in was downloaded from the “Plugin Store” menu of the editor. This plug-in allows executing Python programs inside the editor by only clicking a button.

The neural network is written in Jupyter notebook format. Jupyter notebooks are a type of Python file based on cells that can be executed independently, similar to Wolfram MATHEMATICA. The support for Jupyter notebooks is included in the Python plugins for Visual Studio Code.

3.2. SIMULATION INTRODUCTION

The main objective of the artificial intelligence is to make a prediction of the future state of the system according to present and past sensor data. To achieve this, empirical data of the system behaviour is required.

The empirical data takes the form of a time series which includes the reading of all sensors. Step disturbances are performed in the disturbance and manipulated variables in regular intervals inside the operating range of the system.

This data can be produced with either experimentally in a laboratory or in a computer simulation. Since a physical system was unavailable, a simulation of that system was used to produce the data. This is called a digital twin and it is a widely used technique in the Industry 4.0 paradigm.

3.3. SIMULATION STRATEGY

The system is a double tube heat exchanger with co-current turbulent streams. There is perfect convection in the radial axis, which means that the only temperature difference in the radial axis is between both streams. This temperature difference is the driving force for the heat exchange between the inner and outer tubes. This heat exchange produces a temperature gradient along the axial axis. For each differential in the axial length, the heat exchange between streams is calculated using the macroscopic heat exchange equation.

Inner tube balance:

$$\frac{\left(T_{in(x,t+1)} - T_{in(x,t)}\right)}{dt} = \frac{\left(W_{in(t)} * Cp_{in} * \left(T_{in(x-1,t)} - T_{in(x,t)}\right) - Q_{(x,t)}\right)}{mass_{in(x,t)} * Cp_{in(x,t)}}$$

Outer tube balance:

$$\frac{\left(T_{out(x,t+1)} - T_{out(x,t)}\right)}{dt} = \frac{\left(W_{out(t)} * Cp_{out} * \left(T_{out(x-1,t)} - T_{in(x,t)}\right) + Q_{(x,t)}\right)}{\left(mass_{out(x,t)} * Cp_{out(x,t)}\right)}$$

Heat flow between streams:

$$Q_{(x,t)} = U * A * \left(T_{in(x,t)} - T_{out(x,t)}\right)$$

Accordingly, both streams are simulated with one spatial coordinate (axial position “length”) and one time coordinate. The heat exchange over time and axial coordinate for each stream is calculated in separate energy balances. The heat exchange between streams is modelled as a “generation” variable in the energy balance.

The system is divided along the axial axis into 100 sections, with each length section having one node for the inner tube and one for the outer tube, totalling 200 nodes. Simulations with less sections (e.g 50) were tried but produced less accurate results. Simulations with more than 100 sections produced identical results, indicating diminishing results in accuracy.

The time axis is divided into 0,25 second steps. Step sizes at or above 0,5 seconds produced instabilities in which the simulation diverged towards infinity.

The microscopic heat transfer equation is solved using the explicit Euler method. At first, the simulation is initialized to let it reach a steady state, 1000 seconds was proven to be enough. Once the initialization period is over, the main loop begins. For producing training data, random step disturbance signals are performed once every a random number of seconds.

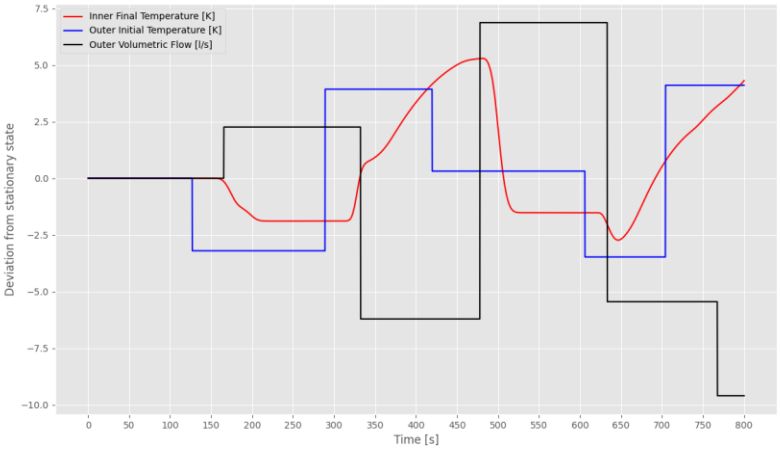

Figure 11. Extract of the simulation

3.3.1. System Study

To understand the behavior of the system, a simulation was run in which only one signal was produced. Afterwards, the response of the system to this disturbance was observed.

However, the response varied significantly depending on the value of other variables. Disturbance signals of +10 K in the outer tube fluid initial temperature were performed under different outer tube fluid volumetric flows, from 2.5 to 20 L/s under co-current flow.

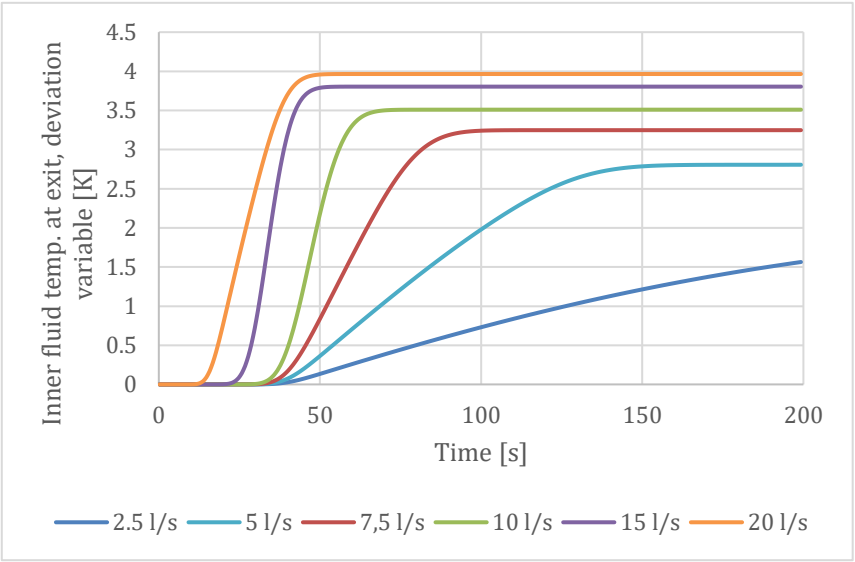

Figure 12. Response to a disturbance of +10 K in the outer stream initial temperature

As seen in figure 12, under flows higher than 10 L/s the response was of a dead time and a relatively fast sigmoid shaped increase of temperature. This dead time increases as volumetric flow decreases. However, at 10 L/s the dead time stops increasing, remaining at 35 seconds. Further reducing flow reduced the slope of the sigmoid function. Static gain increases with flow rate.

The interval between signals of a variable can be constant or randomized. Both options have been used successfully in the literature [39] [40].

However, randomized intervals reduce the risk of overfitting by making it more difficult for the model to attempt to predict the periodicity of signals. If the signals are always produced at the same time or under predictable intervals the model may learn these intervals and produce predictions using that information. Since the objective is to make a model that makes predictions using sensed disturbances data, not predicting disturbances, this would be undesirable.

Consequently, randomized intervals have been used in the simulation. Different intervals have been tested, depending on the number of disturbed variables. Experiments with more disturbed variables have been done with higher intervals, to preserve a similar time between signals of any variable. The ideal time between signals must be found empirically as it has been observed to depend on the architecture of the neural network. As an initial proposal, 50 seconds was chosen, which is higher than the delay time (35 seconds) and lower than the time it takes for the 10 L/s flow to achieve steady-state conditions (70 seconds). Afterwards it had to be increased to 100 and 150 seconds.

3.3.2. Data Storage

Background noise in the sensor has been added, with an amplitude of 0.05 K for all temperatures and 0.025 L/s for flow rates.

The stored data is kept as the sensor data as is, not converted to “disturbance variable” format. Disturbance variable format consists of storing the values of variables as the difference from their value in the stationary state at the beginning of the simulation. Even though using converting to disturbance variables is widespread in control theory, it is not adequate in this case since it conflicts with standardization, a data pre-processing technique used to train the neural network.

However, simulations used for illustrative purposes use disturbance variable format since it allows for better visualizations of the dynamics of the system.

The data of the disturbance variables and the controlled variable are stored in a “.csv” file. To reduce the size of the file and the computational load of training the model, the storage of this data is done in 1-second intervals. Since the simulation is done in 0,25-second steps, data is stored every 4 simulation steps. In the other 3 steps, the data is only used to calculate the simulation, not stored in the “.csv” file.

3.4. DATA PRE-PROCESSING

Data pre-processing is important before training any machine learning model, and in some cases, like in this project, it is required. It represents the actions done to convert raw data into a usable form.

3.4.1. Loading and Checking

First, the data is loaded. The ".csv" files with the simulation data are read and stored into Pandas Data Frames, which behave like an Excel table.

Once the data is loaded, the data is checked for errors or outliers. This has been done by using pandas' "describe()" function, which creates a table with descriptive statistics of the data (minimum, maximum, average, standard deviation, percentiles…)

Table 1. Example of a "describe()" statistics table for one of the simulations

	Outer Temp. [K]	Initial Inner Temp. [K]	Outer Ws [L/s]	Inner Temp. [K]
count	800000	800000	800000	800000
mean	25.00	69.98	10.04	55.08
std	2.92	5.75	5.70	4.76
min	19.98	59.98	0.00	44.20
25%	22.47	65.05	5.25	51.49
50%	24.94	69.86	9.96	54.81
75%	27.55	75.07	14.87	58.29
max	30.02	80.00	20.01	72.67

Anomalous or missing values can be detected using this table. This was useful because it allowed to detect an instability in the simulation. At first, the simulation was run in 0,5-second steps. This produced results that looked fine at the beginning but after several hundred thousand seconds of simulation, it became unstable and diverged into infinity. This was detected by the anomalously large "max" values in the variables.

3.4.2. Data Segmentation and Standardization

Afterwards, the data is separated into training, validation, and test segments. The ratio used for this division is 70% train, 20% validation and 10% test.

The next step is standardization. It is a normalization technique that adjusts the scales of the variable in a way that all vary in the same range. This ensures that the scale of the variable (such as which units are used) does not impact the model. It has been proven to improve the quality of the models even though it slows down training slightly [32].

Standardization is done by calculating the average and the standard deviation of all variables in the testing data. Validation and training data are not used because they are treated as "unknown new data". If they were used, they would affect the results of the normalization, which is not desired. The model should not know anything about validation and test data until validation and testing.

Afterwards, all values of the variables in all 3 data segments are expressed in (train data) standard deviations from the (train data) mean. These values are calculated only once during training and stored for future use. Afterwards, they are used to convert the predictions back from standard deviations into temperature. Also, all future inputs into the model must be standardized using the stored values for standard deviation and mean from the training set.

3.4.3. Data Windowing

Afterwards, the data (in time-series format) is converted into a supervised learning dataset.

Supervised learning datasets consist of inputs and labels. Inputs are the data given to the model to predict the labels. In this case, the objective is for the neural network to make a prediction of the next m seconds (labels), using the last n+1 seconds as inputs.

This is done by using a sliding time-window. The "WindowGenerator" available in the TensorFlow documentation was used. [33]

The Window Generator creates a slice of the dataset that contains the data from t-n to t+m so that there is only the relevant data (inputs and labels). This is then repeated for t = t+1, until the end of the dataset is reached. The result is a collection of small time series which are treated independently.

Time	Input	Label
1	3.3	2.60
2	3.3	2.65
3	3.3	2.7
4	5	2.75
5	5	2.8
6	5	2.85
7	5	2.9
8	5	2.95
9	5	3.5
10	5	4

	Time	Input	Label
t-n	1	3.3	2.60
t-3	2	3.3	2.65
t-2	3	3.3	2.7
t-1	4	5	2.75
t	5	5	2.8
t+1	6	5	2.85
t+2	7	5	2.9
t+m	8	5	2.95

	Time	Input	Label
t-n	2	3.3	2.65
t-3	3	3.3	2.7
t-2	4	5	2.75
t-1	5	5	2.8
t	6	5	2.85
t+1	7	5	2.9
t+2	8	5	2.95
t+m	9	5	3.5

Figure 13. Data Windowing example

For the first window, the values of the inputs and the labels from t=1 to t=5 are given to the model to make predictions of the labels at t=6 to t=8. Then these labels are revealed and the differences between the predicted values and the real ones are calculated.

Afterwards, the values of the inputs and the labels from t=2 to t=6 are used to predict t=7 to t=9…

The time column is not shown to the machine, so it does not know that these windows belong to the same data sequence. It cannot "cheat" by learning what is the label at t=8 in the first window, copying it and then pasting it in the t=8 of the second window.

However, this comes with a problem.

At time t, the system conditions for t+1, t+2… t+m are estimated from the system conditions at t-n, t-n+1, t-n+2… t. If a disturbance were to happen at t+1, the whole prediction would be wrong as the system would implicitly assume that all input variables would remain constant from t to t+m.

There is no way to get around this problem as the information of this "random" disturbance (which comes from out of the system and is thus unpredictable from inside the system) does not exist at time t. A way to avoid this problem from arising is to set up disturbance intervals in a way that the time between them is always higher than m. However, this would come at a cost of reducing the available data points significantly.

As an example of this problem, suppose a system where the label variable follows the equation $(label_t - label_{t-1}) = 0{,}01 * input$

This system is subject to random disturbances in the input variable. The input is kept constant at 5 until at t=6 a step disturbance signal is done with a resulting value of -10.

Table 2. Data Windowing Unpredictability Issue

	Time	Input	Label	Prediction
t-n	1	5	5.45	-
t-3	2	5	5.50	-
t-2	3	5	5.55	-
t-1	4	5	5.60	-
t	5	5	5.65	-
t+1	6	-10	5.55	5.70
t+2	7	-10	5.45	5.75
t+m	8	-10	5.35	5.80

When t=5, the system is only given information of the inputs and labels from t=1 to t=5. As it sees a constant input of 5, the model thinks that the input will remain constant (a continuation of the trend). It has no way of knowing about the t=6 random disturbance. Since the disturbances are random, they cannot be predicted. If the disturbances followed a trend, they could be predicted, but that it is not the case here.

By consequence, this prediction is unavoidably wrong. However, if not too frequent, these incorrect predictions are treated as noise and ignored by the neural network.

3.5. NEURAL NETWORK MODEL

The model takes the inputs at time=t and makes the predictions for t=t+1 to t=t+m in one step, instead of iterating the predictions one by one. This is called a single-shot approach. [34]

The model consists of a LSTM layer with an arbitrary number of units and a Dense layer. The dense layer is a layer of "regular" perceptrons that convert the LSTM output vectors into scalars (the output of the model).

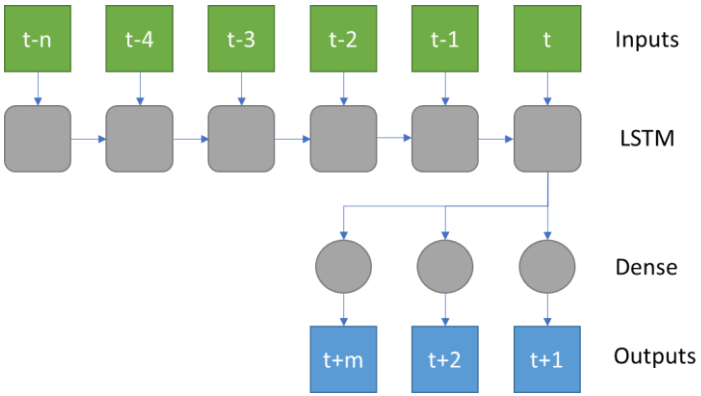

Figure 14. LSTM model illustration

The model is trained for up to 15 epochs or until stopped by its early stopping algorithm. After each epoch, the training loss of the model diminishes but there is a point at which validation loss stops diminishing. At this point the neural network has captured all possible data and additional epochs make the model start overfitting. After this, the validation loss starts increasing again (Figure 14). When the program detects that the validation loss has stopped diminishing, the training is completed.

Further details on the structure of the model can be found in the annex.

4. NEURAL NETWORK DESIGN PROCESS AND RESULTS

4.1. QUALITY MEASUREMENTS

The quality of the models is measured in mean absolute error, which is in standard deviations. The value of a standard deviation depends on which dataset is used. The interval between disturbances in the training data is the variable that affects the standard deviation the most.

The conversion from std. dev. into Kelvin is as follows: $MAE\ [K] = MAE\ [std] * C_{reg}$

where C_{reg}= 4,7 (0,2) K if interval = 50 ; 5,2 (0,2) if interval = 100 ; 5,2 (0,2) if interval = 150.

The development of the models has been done in two stages. First, a basic engineering stage. This is a "discovery" stage where different techniques are tested in a "trial-and-error" fashion to see what works and what doesn't. Here, the quality of the models is measured using their accuracy on their own testing data. Because of the data windowing technique used to prepare the data, there are unpredictable disturbances that reduce the accuracy of the predictions. The error caused by these predictions produces a "MAE floor".

$$MAE_{DiscStage} = MAE_{real} + MAE_{floor}$$

At this point, techniques that do not work properly produce high errors that are evidently above the MAE floor. In essence, this allowed to compare which models work and which don't.

However, comparing the performance of the best models is difficult in this stage since their error is very small compared to the MAE floor.

These models have been downloaded and tested further in a detail engineering stage. There, they have been made to predict a series of predefined tests which have no unpredictable disturbances. In that environment, only the real prediction error is measured. With these measurements, the best hyperparameters for these models are chosen.

4.2. PROOF OF CONCEPT

The first task was to observe if this neural network can learn from the data. Essentially, it is to verify if the choice of architecture (LSTM) was incompatible with the data, or if the way the data was generated was wrong. The results for this first model are successful.

Table 2. Proof of concept results

Number of units	15	24	32	36	42	60
Outer Temp. Dist.	0,0227	0,0197	0,017	0,0171	0,0174	0,0339
Inner Temp. Dist.	0,0213	0,0179	0,0165	0,0149	0,0139	0,0157

The MAE is as low as 0.0170 std (0.08 K) for the 32 units outer temperature disturbed model and 0.0139 std (0.065 K) for the 42 units inner temperature disturbed model. Even though both errors are very small, it seems that the model struggles more with streams where both temperature and flow rate change (outer temperature dist. In this case)

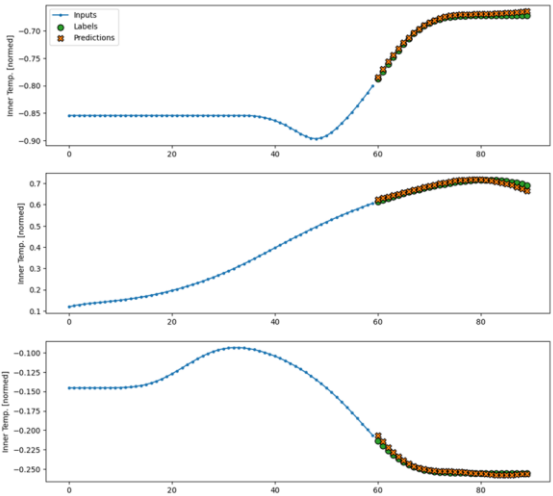

Figure 15. Example predictions for the proof of concept

The window generator produces charts of some example predictions to visually evaluate their quality. A time t is chosen at random, and a prediction is made with the previous n (in this case 60) time steps. In these charts, t = 0 to 59 are used to make a prediction of t = 60 to 89. The green dots represent the labels (“real data”) and the red crosses are the predictions.

Both the MAE values found in Table 2 and the predictions shown in Figure 15 demonstrate that this approach has been successful. Hence, the proof of concept is validated.

4.3. EXPLORATORY DESIGN

4.3.1. 30-second horizon prediction model

In this stage the model is made to work under increasingly complex situations. The first step is increasing the number of disturbed variables. A simulation was conducted in which both the inner and outer streams temperatures experience disturbances. The outer stream volumetric flow is also disturbed, while the inner stream volumetric flow is constant. In this case, the model must also learn the interaction between initial inner temperature and initial outer temperature and the effect each combination of them, under different outer volumetric flows, have on the final inner temperature. Since these flows have different dead times, the model needs to be able to work with simultaneous disturbances in which their effects are not equally delayed. This is an important problem in control theory.

Since these conditions are more complicated to work with, an initial hypothesis was that a higher number of units would be needed to produce a good fitting.

To test the hypothesis, an experiment was done but with higher number of units. Since the model has proven to be successful in the two variables scenario, it has been deemed appropriate to follow a more careful approach in taking measurements. When not deemed unfeasible by computational time issues, the experiments have been done in triplicate and the results expressed in terms of their average and standard deviation.

The length of the simulation was increased from 400.000 seconds to 800.000 seconds to ensure that there is enough data.

The MAE of this model indicates that it has learned to work with 3 variables successfully, with an average absolute error of 0,07 K. The results can be seen in Table 3.

Table 3. 30-second horizon predictions MAE

Number of units	10	24	42	64	90	120
Replica 1	0,0247	0,0234	0,0175	0,0149	0,0148	0,0211
Replica 2	0,0298	0,0221	0,0229	0,0197	0,0158	0,0167
Replica 3	0,0249	0,0173	0,0167	0,0195	0,0228	0,0164
Average	0,026	0,021	0,019	0,018	0,018	0,018
Std. Deviation	0,003	0,003	0,003	0,003	0,004	0,003

The prediction plots also display satisfactory results, as seen in Figure 16.

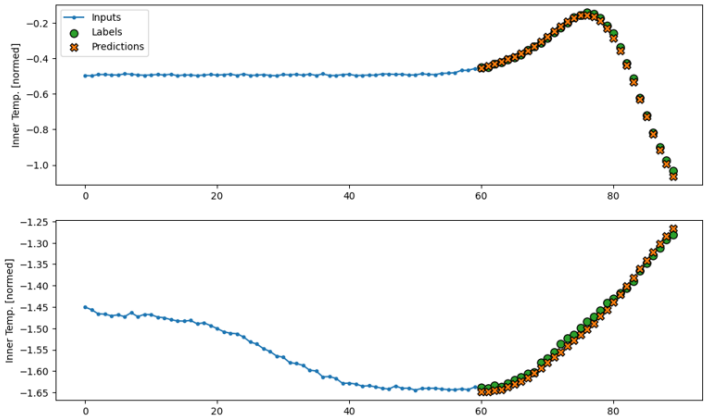

Figure 16. Example predictions for the 30-second prediction horizon model

However, working with more variables increased the number of unpredictable changes that produced completely wrong predictions.

In this example (Figure 17), the outer volumetric rate increased violently sometime slightly after t=60. Since the model only knew about the conditions at t=0 to t=59, it could not know that the volumetric flow was going to change at t=63. For the AI, this information “does not exist”, it is in the future. So, it made a prediction that would have been correct if the volumetric flow did not change.

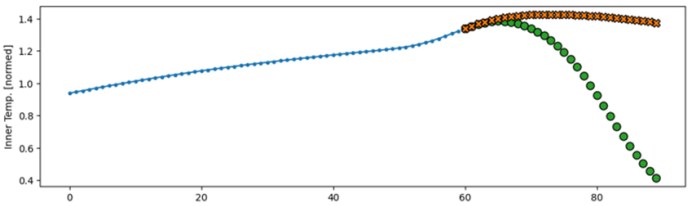

Figure 17. Example of the unpredictability issue producing wrong predictions

The next step has been to increase the range of the predictions to more than 30 seconds. This prediction range is slightly too low for this system and gives it a flaw. For outer stream volumetric flows lower than 10 L/s, the dead time is higher than 30 seconds, up to 37. By consequence, there is a 7 second delay between when the sensor obtains information of the disturbance and when its effects enter the predicted time range.

At volumetric flows above 10 L/s this is not a problem as the dead time is under 30 seconds. Whenever a disturbance happens, at least some of its effects are immediately visible in the predictions.

To ensure at least some information of the effect of any disturbance is immediately seen, this predicted range must be higher than 37 s. First, a prediction range (m) of 40 s has been tried.

At first, the same conditions from the previous experiments were replicated. The model with 42 units was used, predicting 40 seconds from the previous 60 seconds of data. There should be no need to increase the number of units as the complexity of the underlying system is the same. The obtained MAE was 0,0259 with a standard deviation of 0,003.

This error is substantially higher than in the previous experiment. While an increase in error is expected since predictions far into the future carry more uncertainties, this change seems too high for such a small increase in the prediction horizon.

Another problem is that the training time increased, which was expected to become a problem if the prediction horizon were to be extended further. A measure to reduce this training time is increasing batch size. Models with higher batch sizes are significantly faster to train but are less accurate. The significance of this loss in accuracy must be found empirically.

To test the impact of increasing batch size, the 42-units model with n=60 and m=40 was trained with a batch size of 64 and of 128.

The results (Table 4) indicate that the loss in accuracy by increasing the batch size to 128 is very low, below one standard deviation (0,003). On the contrary, training times improved significantly. By consequence, all further experiments have been done with a batch size of 128.

Table 4. Batch size experimentation

Batch Size	MAE	Train time by Epoch [s]
32	0,0259	90
64	0,0265	60
128	0,0277	40

4.3.2. 60-second prediction horizon model

At a prediction horizon of 60 seconds, a different problem arose. At this point, the time between disturbances is smaller than the prediction horizon. By consequence, more than half of the data windows included at least one “unexpected change” that made them at least partially unusable. The available useful data windows would be less than half of the ones provided.

Predictably, the results obtained using the same conditions as the previous experiment (42 units, batch size = 128 but with n=60 and m=60) were unacceptable. The MAE was 0,0794 and the prediction error was visually evident in the plot.

At this point, it was decided that the Mean Squared Error loss function was not performing correctly. Mean Squared Error punishes larger errors more harshly. With these long prediction horizons, unpredictable disturbances had enough time to produce very large deviations from the prediction.

These unavoidable “noise” errors were given too much importance by the loss function, more than the actual errors made in the normal predictions. This made the model attempt to fit these unpredictable disturbances, which is overfitting.

Additionally, different further approaches to reduce the error have been tried.

The first one was increasing the length of the simulation. Its length has been doubled to 1,6 million seconds. This increases the total amount of available windows, to compensate for the ones lost. It did not produce successful results.

Table 5. Effects of increasing the length of the simulation.

Length [s]	24 units	42 units	64 units
800 000	0,0652	0,0635	0,0637
1 600 000	0,0663	0,0643	0,0635

Another option was increasing the number of previous data steps used. Since the prediction horizon is larger, it needs more capacity to capture long term trends. Increasing n from 60 to 90 yielded a slight improvement in accuracy. Further increases produced no improvement.

Table 6. Effects of increasing the number of past data points.

Length [s]	24 units	42 units	64 units
800 000	0,0629	0,0611	0,0604
1 600 000	0,0626	0,0606	0,0600

And finally, increasing the time between disturbances to 100 seconds.

Table 7. Experiment from Table 6 repeated with 100 seconds between disturbances.

Length [s]	24 units	42 units	64 units
800 000	0,0326	0,0316	0,0309
1 600 000	0,0314	0,0315	0,0320

This reduces the proportion of wasted windows but at the cost of also reducing the total amount of disturbances. By consequence, there are more empty data windows where nothing happens. It should also be noted that this approach also changes the testing data. Thus, the MAE obtained with this approach cannot be compared with MAE obtained in other methods, as MAE can only be compared among models with the same testing data. Since the performance between these approaches cannot be directly compared in this stage, models with both approaches have been saved for the testing stage.

Both interval = 50 s and interval = 100 s produced very similar predictions despite having different MAE. This makes evident that the quality of models with different training sets cannot be compared by their performance in their respective training sets. In both cases the predictions are very accurate except when unpredictable disturbances happen. (Figure 18)

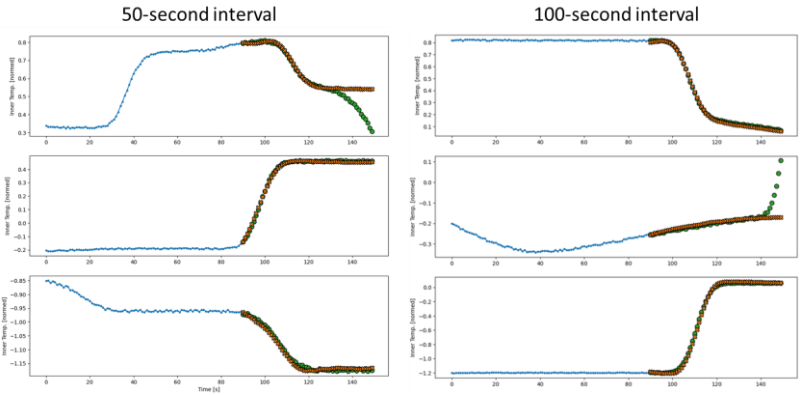

Figure 18. Comparison of the predictions of models with both disturbance intervals

4.3.3. 90-second prediction horizon model

Since the model was able to expand its prediction range from 30 to 60 seconds, an attempt to increase it to 90 seconds has been done. Here, datasets with 100- and 150-seconds delay between disturbances have been used.

Using MSE as a loss function produced completely wrong predictions. In Figure 19 it can be seen how the model "invents" disturbances where there should be none.

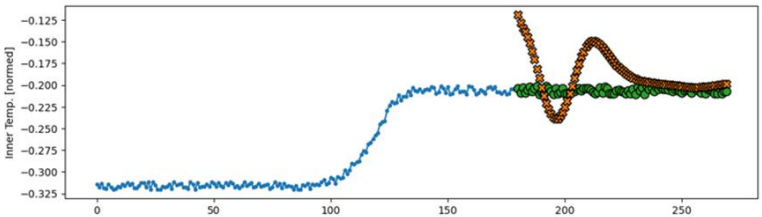

Figure 19. Wrong prediction of a model with a MAE of 0,11.

So, it seems that MAE is the only option for longer time horizons. The successful techniques used to predict a 60 second horizon were used for the 90 second horizon model.

Data from the last 90 seconds was used, as using 120 seconds did not improve the results.

The trend is very similar to the one observed when increasing the prediction horizon from 30 to 60 seconds. Using the same disturbance interval yields significantly higher MAE (more than double), even though the loss in real accuracy is much lower since this increase is mostly caused by a larger MAE floor (caused by unpredictable disturbances) rather than real prediction error.

However, increasing this range reduces the number of disturbances in the data, which means less information. At 150 second intervals, 800 000 seconds of data isn't enough to "saturate" the neural network with data as using 1 600 000 produces better results.

Table 8. Different interval results

	MAE using 100 s intervals			MAE using 150 s intervals		
Length [s]	24 units	42 units	64 units	24 units	42 units	64 units
800 000	0,0825	0,0826	0,0831	0,0566	0,0563	0,0565
1 600 000	0,0831	0,0821	0,0828	0,0537	0,0539	0,0534

Like in the 60-second intervals, models with both interval lengths have been stored so they can be tested in the next stage. There, the performance across intervals can be compared.

The plotted predictions (Figure 20) follow the same trend that was observed in the previous experiments. Even though MAE is higher, it is caused by those unavoidably wrong predictions. In those cases where there are no unexpected disturbances, the predictions are very accurate.

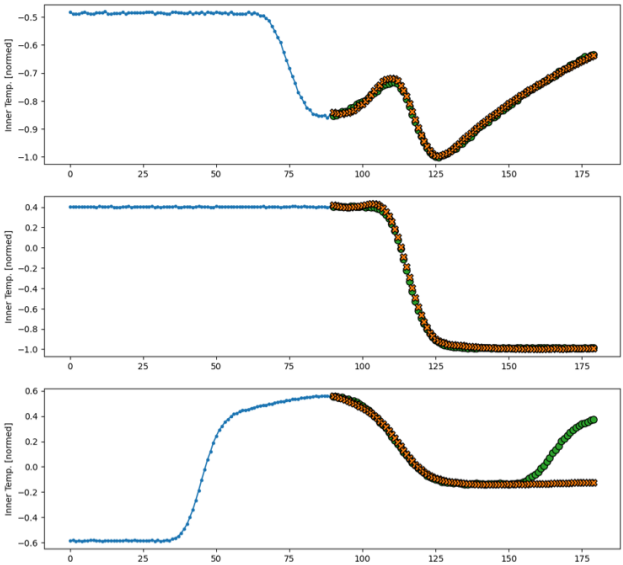

Figure 20. 90-second prediction horizon example predictions

The prediction horizon won't be extended further beyond this point as 90 seconds is enough to capture the dynamic trends of the system. 90 seconds is enough for any disturbance to reach stationary state while flow rate is over 7,5 L/s, which is more than half of the range of that variable. However, if it were needed, it seems theoretically possible to increase this horizon further or even indefinitely, given high enough disturbance intervals and long enough simulations.

4.4. DETAIL ENGINEERING AND TESTING

4.4.1. Testing Approach

Models with different number of units (identified with a letter “s”), disturbance intervals (“i), and simulation length (“l”) have been trained with the techniques that have performed the best in the previous stage. Afterwards, they have been tested with 8 intentionally made testing simulations. A description of the tests and all results is available in the annex.

In these tests, one or more disturbances are performed and immediately after the last disturbance the model is tasked with predicting the future state of the system. The average MAE of each model over the 8 tests is used to compare their performance.

The 60- and 90-second prediction horizon models are tested, since the 30-second prediction horizon is hard to test under these conditions and served more as a proof of concept.

All models have been trained using the Mean Average Error loss function, a batch size of 128 and at least 800 000 seconds of data with randomized disturbance intervals. The models’ number of units range from 24 to 150.

The MAE has been converted from standard deviations to Kelvin using the stored standardization constants of each model, which can be found in the annex. The values in the following tables are in Kelvin, so they are in the order of 5 times higher than if they were in standard deviations like in the previous stage.

4.4.2. Results

Table 9. 60-second prediction horizon results

	MAE [K] by number of units					
Model	24	42	64	90	120	150
i = 100, l = 800	0,049	0,046	0,045	0,034	0,028	0,035
i = 100 , l = 1600	0,046	0,033	0,030	0,032	0,027	0,026
i = 50 , l = 800	0,047	0,035	0,034	0,027	0,033	0,027
i = 50, l = 1600	0,034	0,030	0,029	0,025	0,028	0,033

Table 10. 90-second prediction horizon results

	MAE [K] by number of units					
Model	24	42	64	90	120	150
i = 150, l = 800	0,054	0,047	0,033	0,031	0,036	0,036
i = 150 , l = 1600	0,044	0,041	0,036	0,031	0,026	0,028
i = 100 , l = 800	0,064	0,055	0,048	0,030	0,034	0,030
i = 100, l = 1600	0,056	0,037	0,028	0,034	0,039	0,044

As expected, the results of these tests allowed for better comparisons among models since there is no MAE floor due to unavoidable predictions (explained by how the difference between models with different “i” is much smaller than in the previous stage). This also highlighted the differences in MAE between models with different number of units.

The performance of all models was very satisfactory since their MAE is close to the absolute minimum of 0,025 K, which is the noise in the measured variable.

The best model for the 60-second horizon is the 90-unit, with 1 600 000 seconds of data and 50-second disturbance intervals.

In the 90-second horizon, it is the 120-unit model with 1 600 000 seconds of data and 150-second disturbance intervals.

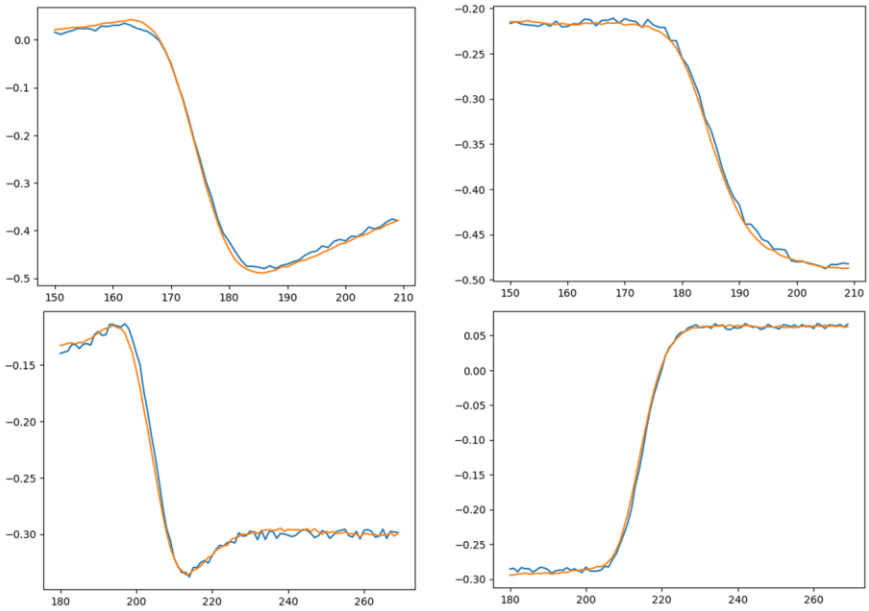

Figure 21. Prediction examples of the best models for the 60- and 90-second horizons, with the real value in blue and the prediction in orange

This conflicts with what was observed in the previous stage, where quality seemed to stagnate after 24 units and remained constant afterwards. Here, this trend continued further up to 64 units.

Consequently, overfitting is less of an issue than expected, with models starting to overfit (increasing MAE) at 120-150 units. The ideal number of units across all models is 90-120, higher than the 42 that was seen in the development stage. However, the 42-unit models still perform very close to optimal, since the optimum in the number of units is a wide range rather than an exact number.

The 1 600 000-second models perform marginally better than their 800 000-second counterparts and with an adequate number of units their performance is almost identical. It is likely that at 800 000 seconds the neural network is already “saturated” with data and good results could still be obtained with shorter simulations.

In the 60-second prediction horizon, the shorter interval (50) performed better than the longer one (100). However, in the 90-second interval, the longer interval (150) is better. Thus, it seems that the ideal disturbance interval does not scale linearly with the prediction horizon and should still be found empirically. The optimal MAE is similar in both intervals. However, it is found at different number of units, closer to 120 for the longer intervals and 90 for the shorter ones.

The optimal MAE obtained in both prediction horizons is essentially the same, even though the 90-second horizon has higher errors in the suboptimal models. By consequence, it is possible to extend the prediction horizon without losses in quality if the hyperparameters are correctly optimized. Thus, it should be possible to extend this horizon further if it were needed.

All models produced results extremely fast, doing 200 complete horizon predictions in the order of 3 ms. This equals to 15 μs per prediction or 66.000 predictions per second. This equals to over 8 000 000 seconds of simulation per second, which would have taken on the order of an hour to simulate on the original simulator.

5. CONCLUSIONS

The results were very satisfactory. The mean error in the predictions of the best models were less than 5% over the background noise, which is an almost perfect fit despite working with 3 input variables in a complicated dynamic system. Additionally, these predictions have been done on the order of 3000 times faster than conventional simulations.

Consequently, neural networks are an adequate alternative to conventional system identification, with the advantage of being theoretically applicable to any system. This makes them a promising tool for modelling non-linear systems with variable dead times, a difficult problem in the field of control.

The performance of the model has been observed to rely on the data (quantity and quality) and the neural network structure (architecture and hyperparameters).

The data required to train the model was obtained via a simulation. The length of the simulation required to achieve optimal results is short enough to viably simulate, but it may be too long to obtain experimentally in the laboratory. Further work would need to be done in optimizing the data generation strategy to reduce the length of the required dataset.

The Long-Short Term Memory (LSTM) neural network architecture has been proven to be a good fit for systems with varying dead time, as it has made correct predictions under different flow rates.

The appropriate hyperparameters had to be found empirically, even though the range at which the model performed close to optimal was wide. For example, batch size was increased from 32 to 128 with a minimal loss in quality but a significant improvement in training speed.

The ideal number of units is around 90 for these models, but the results were satisfactory in the range of 24-150. Under 24 units the model was too simple and failed to capture the trends in the data, underfitting.

On the other end, excessively high number of units produced models that fitted the training data too closely (overfitting), failing to generalize on new data. The early stopping algorithm prevented the model from overfitting up until 150 units.

However, the choice of loss function had a significant impact in the quality of the model. Between the two standard regression loss functions, Mean Squared Error (MSE) and Mean Absolute Error (MAE), MAE produced the best results. This is explained by the fact that the data windowing technique used to process the data caused some unavoidably wrong predictions, which disturbed the training process when using MSE but were treated as noise and disregarded with MAE.

The prediction horizon was expanded up to 90 seconds (enough to capture the effects of most disturbances in this system) without loss in quality, only requiring changing the hyperparameters. As such, this horizon may be expanded further or even indefinitely.

Since the model makes predictions on the order of thousands of times faster than the simulation, it is fast enough to allow real-time optimization of future control actions. This may be applied in Model Predictive Control or be used to reduce the computational load (and thus power consumption) of already existing models. However, the applications are not limited to MPC, as it may be used for any application that requires performing system identification on dynamic systems with non-linearities or variable dead times.

REFERENCES AND NOTES

1. Ballester, O. (2021). An Artificial Intelligence Definition and Classification Framework for Public Sector Applications. DG.O2021: The 22nd Annual International Conference on Digital Government Research. https://www.semanticscholar.org/paper/An-Artificial-Intelligence-Definition-and-Framework-Ballester/432797f689029e81fdcd1e24b30226d8772e4bad
2. Skinner, R.E. (2012). Building the Second Mind: 1956 and the Origins of Artificial Intelligence Computing. https://www.semanticscholar.org/paper/Building-the-Second-Mind%3A-1956-and-the-Origins-of-Skinner/2ff3f19ffb557f7d5eebb91224c3d7f877ccc7ca
3. Niklas Kühl, Max Schemmer, Marc Goutier, Gerhard Satzger, Artifcial intelligence and machine learning, 16 May 2022. https://link.springer.com/content/pdf/10.1007/s12525-022-00598-0.pdf
4. Morales, E.F., & Escalante, H.J. (2022). A brief introduction to supervised, unsupervised, and reinforcement learning. Biosignal Processing and Classification Using Computational Learning and Intelligence. https://www.semanticscholar.org/paper/A-brief-introduction-to-supervised%2C-unsupervised%2C-Morales-Escalante/8c1bb43628ac320166b635b8aa521c13b83e25a8
5. Bagus Suteja, Application of Neural Network in Letter Recognition Using the Perceptron Method. https://journalinstal.cattleyadf.org/index.php/Instal/article/download/9/10
6. Johannes Lederer, Activation Functions in Artificial Neural Networks: A Systematic Overview, January 26, 2021. https://arxiv.org/abs/2101.09957
7. Diederik P. Kingma, Jimmy Ba, Adam: A Method for Stochastic Optimization, 22 Dec 2014. https://arxiv.org/abs/1412.6980
8. Borislava Vrigazova, The Proportion for Splitting Data into Training and Test Set for the Bootstrap in Classification Problems, 1 May 2021. https://www.semanticscholar.org/paper/The-Proportion-for-Splitting-Data-into-Training-and-Vrigazova/1dc9bd7eaf7ac9ba03f854d75bfddd545b6eb1d3
9. Buscema, Massimo. (1998). Back Propagation Neural Networks. Substance use & misuse. 33. 233-70. 10.3109/10826089809115863. https://www.researchgate.net/publication/13731614_Back_Propagation_Neural_Networks
10. Jun Fang Mingze Xu∗ Hao Chen Bing Shuai Zhuowen Tu Joseph Tighem, An In-depth Study of Stochastic Backpropagation, 30 Sep 2022. https://arxiv.org/pdf/2210.00129.pdf
11. Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0
12. Federico Girosi, Michael Jones, Tomaso Poggio • Regularization Theory and Neural Networks Architectures, 1995. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.9258
13. Qipei Li, Ming Yan, Jie Xu, 2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS) Optimizing Convolutional Neural Network Performance by Mitigating Underfitting and Overfitting. https://www.semanticscholar.org/paper/Optimizing-Convolutional-Neural-Network-Performance-Li-Yan/125c91e11fbb24a905322c2098a9688e9a12d3a2
14. Kurt Hornik,Maxwell Stinchcombe and Halbert White. Multilayer Feedforward Networks are Universal Approximators, 9 March 1989 https://cognitivemedium.com/magic_paper/assets/Hornik.pdf
15. TIOBE Index. https://www.tiobe.com/tiobe-index/
16. Akshit J. Dhruv, Reema Patel and Nishant Doshi. Python: The Most Advanced Programming Language for Computer Science Applications. https://www.scitepress.org/Papers/2020/103079/103079.pdf

17. Python Documentation https://docs.python.org/3/faq/general.html#what-is-python
18. Numpy https://numpy.org/
19. Pandas https://pandas.pydata.org/
20. Matplotlib https://matplotlib.org/
21. Scikit-learn https://scikit-learn.org/stable/
22. Tensorflow https://www.tensorflow.org/?hl=es-419
23. Keras https://keras.io/
24. Ralf C. Staudemeyer, Eric Rothstein Morris, Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks, 12 Sep 2019, https://arxiv.org/abs/1909.09586
25. Mehmet BULUT ,Hydroelectric Generation Forecasting with Long Short Term Memory (LSTM) Based Deep Learning Model for Turkey https://arxiv.org/ftp/arxiv/papers/2109/2109.09013.pdf
26. Gang Chen ∗ Department of Computer Science and Engineering, SUNY at Buffalo, 14 Jan 2018. A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation https://arxiv.org/pdf/1610.02583.pdf
27. Robin M. Schmidt, Recurrent Neural Networks (RNNs): A gentle Introduction and Overview, 23 Nov 2019 https://arxiv.org/abs/1912.05911
28. Jaeyoung Kim, Mostafa El-Khamy, Jungwon Lee, Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition, 5 Jun 2017 https://arxiv.org/abs/1701.03360
29. Emirhan Inanc, Yigit Gurses, Abdullah Habboush, Yildiray Yildiz, Anuradha M. Annaswamy, Neural Network Adaptive Control with Long Short-Term Memory, 5 Jan 2023 https://arxiv.org/abs/2301.02316
30. Jiaqi Meng , Chengbo Li , Jin Tao , Yi Li , Yi Tong, Yu Wang, Lei Zhang, Yachao Dong, and Jian Du,* RNN-LSTM-Based Model Predictive Control for a Corn-to-Sugar Process https://www.mdpi.com/2227-9717/11/4/1080/pdf
31. Sida Xing, Feihu Han, Suiyang Khoo, Extreme-Long-short Term Memory for Time-series Prediction, 15 Oct 2022 https://arxiv.org/abs/2210.08244
32. M. Shanker, M.Y. Hu, M.S. Hung Effect of data standardization on neural network training, https://www.sciencedirect.com/science/article/abs/pii/0305048396000102
33. Fernandez Roblero, Jaime & Little, Suzanne & O'Connor, Noel. (2019). A Single-Shot Approach Using an LSTM for Moving Object Path Prediction. 1-6. 10.1109/IPTA.2019.8936126. https://www.tensorflow.org/tutorials/structured_data/time_series
34. Fernandez Roblero, Jaime & Little, Suzanne & O'Connor, Noel. (2019). A Single-Shot Approach Using an LSTM for Moving Object Path Prediction. 1-6. 10.1109/IPTA.2019.8936126. https://www.researchgate.net/publication/338069503_A_Single-Shot_Approach_Using_an_LSTM_for_Moving_Object_Path_Prediction
35. Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang, On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, 15 Sep 2016 https://arxiv.org/abs/1609.04836
36. Yuvraj Bhushan Khare Yaduvir Singh, PID Control of Heat Exchanger System, .6, October 2010 https://www.ijcaonline.org/volume8/number6/pxc3871742.pdf
37. Artificial neural network-based FCS-MPC for three-level inverters, Xinliang Yang, Kun Wang,, 29 September 2022 https://www.semanticscholar.org/paper/Artificial-neural-network-based-FCS-MPC-for-Yang-Wang/b54c07070a6c416be59f52443e9bf6429516b2eb
38. Divas Karimanzira, Thomas Rauschenbach, Deep Learning Based Model Predictive Control for a Reverse Osmosis Desalination Planthttps://www.scirp.org/journal/paperinformation.aspx?paperid=104606
39. Zarzycki, K.; Ławryńczuk, M. LSTM and GRU Neural Networks as Models of Dynamical Processes Used in Predictive Control: A Comparison of Models Developed for Two Chemical Reactors. Sensors 2021, 21, 5625. https://doi.org/10.3390/s21165625https://www.mdpi.com/1424-8220/21/16/5625

40. Munoz, S.A.; Park, J.; Stewart, C.M.; Martin, A.M.; Hedengren, J.D. Deep Transfer Learning for Approximate Model Predictive Control. Processes 2023, 11, 197. https://doi.org/10.3390/pr11010197, https://www.mdpi.com/2227-9717/11/1/197

ACRONYMS

MAE: Mean Absolute Error

MSE: Mean Standard Error

RNN: Recurrent Neural Network

LSTM: Long-Short Term Memory

APPENDICES

APPENDIX 1: USING THE FILES

The folder with all the required files (Simulator, neural network model and testing program) along with the models and simulations used are available in the following public GitHub repository:

https://github.com/Maxekai/TFG

To execute the programs, it is required to have a Python 3 interpreter installed in the computer. It is available in the Microsoft Store or in the Python page https://www.python.org/downloads/

To open, edit and execute the files a code editor is necessary. The free "Visual Studio Code" code editor was used to develop this project, but any other code editor should also work.

Visual Studio Code is available to download here: https://code.visualstudio.com/

To execute these files in the editor, open the Extensions Store by clicking on its icon on the left of the screen or use the shortcut (Ctrl+Shift+X) and download the "Python" extension. The correct one should appear the first in the list, and its author is Microsoft.

This extension allows executing the files inside the editor and support for Jupyter Notebooks.

Afterwards, open a terminal either by clicking on "Terminal" on the top menu or by using the shortcut (Ctrl+Shit+ñ). There, paste the following text: pip install -r requirements.txt

This command installs all required libraries to run the program.

Execute the simulator by clicking the play button on the upper right corner or the screen. For the Jupyter notebooks, they can be executed all at once by clicking the Run All button on the top of the screen or cell by cell by clicking their respective "Play" button.

APPENDIX 2: NEURAL NETWORK PARAMETERS

Data Properties:

Fixed Settings:

The heat exchanger itself is kept unchanged. Its physical properties (length,area,U…) are constant in all experiments.

The physical properties of the fluids (density,heat capacity…) are also kept constant, except for the temperature.

Valve Range: The valve that controls the outer fluid flow rate has a constant range of 0 to 20 L/s.

Disturbed variable range: In the experiments where they are disturbed, outer fluid initial temperature has a range of -5 K to +5 K from the initial steady state, -10 K to +10 K for the inner fluid initial temperature and -1 L/s to +1 L/s for the inner fluid flow rate.

Initial State: The system is always initialized with a 7 L/s, 70 °C inner fluid stream and a 10 L/s, 25 °C outer fluid stream.

Manipulated Settings:

Disturbance variables: Different experiments are performed with numbers of disturbed variables. However, the possible values of these variables compared to the initial stationary state is fixed at [-10, +10] L/s for outer stream flow rate, [-5,+5] K for outer stream input temperature and [-10,+10] K for inner stream input temperature.

Time between disturbances: The average time between disturbances of any kind is manipulated by changing the time between disturbances of each variable.

Randomness of the time between disturbances: Some experiments have been done with a constant time between each disturbance and others with a random interval.

Noise: Some experiments have been done with noise in the non-constant variables. The noise is an arbitrary value fixed at 0,05 K for temperatures and 0,025 L/s for volumetric flows.

Neural Network Properties:

Fixed Settings:

Features = 1. This is the number of variables to predict. The model only predicts the final inner temperature.

Return Sequences = False. This makes the LSTM layer return an output only at the last time step. This output is the prediction for the following m time steps. This can be observed in Figure 14T, where the LSTM cells from t-n to t-1 do not return any output but only pass information to the next cell.

Optimizer = Adam. This is a state-of-the-art optimizer, the “default” choice.

Metrics = Mean Absolute Error (MAE). This is how the quality of the model is measured during validation and testing. It is not used to train the model, but to calculate the expected error of the predicted values.

Manipulated Settings:

Label Width: The “m” number of predicted time steps.

Input Width: The “n+1” number of time steps used to make a prediction.

Number of Units: The number of stacked LSTM cells, which is related to the complexity of the model.

Input data size: The number of simulation time-steps loaded into the model.

Loss = Mean Squared Error (MSE) and Mean Absolute Error (MAE). At first MSE was use, but in the models with a longer prediction window MSE did not yield good results and MAE was used instead.

Batch Size = This parameter affects the learning process; higher values make learning faster but less accurate. The simpler models, where training time is not excessive, have been trained with a batch size of 32. This is at the lower end of typical batch sizes, giving the best quality. [35]

The more complicated models have been done with batch sizes of 64 and 128, which have been shown to produce significantly faster results with minimal loss in accuracy.

APPENDIX 3: STANDARDIZATION CONSTANTS

These constants can be copied and pasted them into the program to apply them.

I =50, I=800k

train_mean = pd.Series({'Outer Temp.':25.041515,'Initial Inner Temp.':69.902352,'Outer Ws':10.039647,'Inner Temp.':55.038887})

train_std = pd.Series({'Outer Temp.':2.929728,'Initial Inner Temp.':5.753829,'Outer Ws':5.711073,'Inner Temp.':4.768334})

i=50, I=1600k

train_mean = pd.Series({'Outer Temp.':24.981220,'Initial Inner Temp.':70.028579,'Outer Ws':10.030917,'Inner Temp.':55.138814})

train_std = pd.Series({'Outer Temp.':2.909863,'Initial Inner Temp.':5.762679,'Outer Ws':5.771887,'Inner Temp.':4.784221})

i=100 , I=800k

train_mean = pd.Series({'Outer Temp.':24.995166,'Initial Inner Temp.':70.154394,'Outer Ws':9.860415,'Inner Temp.':55.6092745})

train_std = pd.Series({'Outer Temp.':2.885494,'Initial Inner Temp.':5.796503,'Outer Ws':5.754449,'Inner Temp.':5.250952})

i=100, l=1600k

train_mean = pd.Series({'Outer Temp.':24.985298,'Initial Inner Temp.':69.970662,'Outer Ws':9.854125,'Inner Temp.':55.485196})

train_std = pd.Series({'Outer Temp.':2.868629,'Initial Inner Temp.':5.775671,'Outer Ws':5.751491,'Inner Temp.':5.213089})

i=150, l=800k

train_mean = pd.Series({'Outer Temp.':25.057913,'Initial Inner Temp.':69.808034,'Outer Ws':9.720709,'Inner Temp.':55.678122})

train_std = pd.Series({'Outer Temp.':2.900207,'Initial Inner Temp.':5.835023,'Outer Ws':5.837478,'Inner Temp.':5.589536})

i=150, l=1600k

train_mean = pd.Series({'Outer Temp.':25.103154,'Initial Inner Temp.':69.712887,'Outer Ws':9.888763,'Inner Temp.':55.491773})

train_std = pd.Series({'Outer Temp.':2.866545,'Initial Inner Temp.':5.794157,'Outer Ws':5.807717,'Inner Temp.':5.437240})

APPENDIX 4: TEST RESULTS

All results are expressed in standard deviations, m= prediction horizon in seconds, n= past seconds used for the predictions, s= number of units, i= disturbance interval, l= simulation length

	Test 1	Test 2	Test 3	Test 4	
m60n90s24i100l800	0.02051	0.00522	0.00499	0.00479	
m60n90s42i100l800	0.00819	0.00484	0.00559	0.00561	
m60n90s64i100l800	0.01217	0.00466	0.00579	0.00548	
m60n90s90i100l800	0.00753	0.00634	0.00352	0.00510	
m60n90s120i100l800	0.00625	0.00272	0.00373	0.00635	
m60n90s150i100l800	0.00410	0.00347	0.00593	0.00699	
	Test 5	Test 6	Test 7	Test 8	Average
m60n90s24i100l800	0.00724	0.00811	0.01176	0.01278	0.00943
m60n90s42i100l800	0.00758	0.00589	0.01067	0.02201	0.00880
m60n90s64i100l800	0.01010	0.00874	0.00695	0.01450	0.00855
m60n90s90i100l800	0.00498	0.00444	0.00624	0.01378	0.00649
m60n90s120i100l800	0.00745	0.00552	0.00587	0.00551	0.00542
m60n90s150i100l800	0.00489	0.00519	0.00597	0.01631	0.00661

	Test 1	Test 2	Test 3	Test 4
m60n90s24i100l1600	0.00974	0.00399	0.00611	0.00595
m60n90s42i100l1600	0.00915	0.00304	0.00449	0.00403
m60n90s64i100l1600	0.00683	0.00283	0.00534	0.00418
m60n90s90i100l1600	0.00475	0.00322	0.00591	0.00549
m60n90s120i100l1600	0.00729	0.00273	0.00364	0.00563
m60n90s150i100l1600	0.00509	0.00274	0.00366	0.00538

	Test 5	Test 6	Test 7	Test 8	Average
m60n90s24i100l1600	0.01084	0.00732	0.01058	0.01595	0.00881
m60n90s42i100l1600	0.00474	0.00561	0.00719	0.01173	0.00625
m60n90s64i100l1600	0.00795	0.00484	0.00714	0.00751	0.00583
m60n90s90i100l1600	0.00725	0.00675	0.00858	0.00708	0.00613
m60n90s120i100l1600	0.00481	0.00417	0.00472	0.00803	0.00513
m60n90s150i100l1600	0.00557	0.00453	0.00673	0.00579	0.00494

	Test 1	Test 2	Test 3	Test 4
m60n90s24i50l800	0.01995	0.00667	0.00680	0.00573
m60n90s42i50l800	0.01226	0.00427	0.00572	0.00476
m60n90s64i50l800	0.01108	0.00464	0.00504	0.00485
m60n90s90i50l800	0.00594	0.00577	0.00502	0.00644
m60n90s120i50l800	0.00942	0.00856	0.00748	0.00516
m60n90s150i50l800	0.00769	0.00381	0.00420	0.00622

	Test 5	Test 6	Test 7	Test 8	Average
m60n90s24i50l800	0.01039	0.00782	0.01065	0.01071	0.00984
m60n90s42i50l800	0.00656	0.00611	0.00811	0.01148	0.00741
m60n90s64i50l800	0.00430	0.00536	0.00767	0.01364	0.00707
m60n90s90i50l800	0.00509	0.00483	0.00503	0.00779	0.00574
m60n90s120i50l800	0.00525	0.00638	0.00595	0.00748	0.00696
m60n90s150i50l800	0.00344	0.00423	0.00493	0.01046	0.00562

	Test 1	Test 2	Test 3	Test 4	
m60n90s24i50l1600	0.00945	0.00459	0.00513	0.00559	
m60n90s42i50l1600	0.00757	0.00332	0.00648	0.00473	
m60n90s64i50l1600	0.00694	0.00671	0.00698	0.00449	
m60n90s90i50l1600	0.00728	0.00427	0.00421	0.00377	
m60n90s120i50l1600	0.00478	0.00431	0.00585	0.00561	
m60n90s150i50l1600	0.00722	0.00796	0.00655	0.00455	
	Test 5	Test 6	Test 7	Test 8	Average
m60n90s24i50l1600	0.00848	0.00679	0.00748	0.00933	0.00710
m60n90s42i50l1600	0.00477	0.00533	0.00729	0.01003	0.00619
m60n90s64i50l1600	0.00401	0.00497	0.00787	0.00669	0.00608
m60n90s90i50l1600	0.00369	0.00464	0.00576	0.00886	0.00531
m60n90s120i50l1600	0.00439	0.00489	0.00576	0.01066	0.00578
m60n90s150i50l1600	0.00390	0.00588	0.00871	0.01111	0.00699

	Test 1	Test 2	Test 3	Test 4	
m90n90s24i150l800	0.01598	0.00728	0.00531	0.00515	
m90n90s42i150l800	0.00988	0.00562	0.00636	0.00488	
m90n90s64i150l800	0.01158	0.00393	0.00289	0.00408	
m90n90s90i150l800	0.00631	0.00394	0.00469	0.00335	
m90n90s120i150l800	0.00790	0.00606	0.00405	0.00465	
m90n90s150i150l800	0.00726	0.00544	0.00385	0.00399	
	Test 5	Test 6	Test 7	Test 8	Average
m90n90s24i150l800	0.01463	0.00903	0.00997	0.01012	0.00969
m90n90s42i150l800	0.01120	0.00879	0.00943	0.01125	0.00843
m90n90s64i150l800	0.00548	0.00626	0.00493	0.00829	0.00593
m90n90s90i150l800	0.00543	0.00494	0.00582	0.00953	0.00550
m90n90s120i150l800	0.00790	0.00498	0.00421	0.01237	0.00652
m90n90s150i150l800	0.00662	0.00668	0.00734	0.00993	0.00639

	Test 1	Test 2	Test 3	Test 4	
m90n90s24i100l800	0.01907	0.01178	0.00565	0.00525	
m90n90s42i100l800	0.01193	0.01082	0.00694	0.00575	
m90n90s64i100l800	0.00731	0.00631	0.00695	0.00495	
m90n90s90i100l800	0.00757	0.00412	0.00359	0.00399	
m90n90s120i100l800	0.00818	0.00484	0.00609	0.00686	
m90n90s150i100l800	0.00665	0.00416	0.00345	0.00429	
	Test 5	Test 6	Test 7	Test 8	Average
m90n90s24i100l800	0.01587	0.00932	0.00827	0.02257	0.01222
m90n90s42i100l800	0.00803	0.00589	0.00962	0.02413	0.01039
m90n90s64i100l800	0.01239	0.00965	0.00797	0.01758	0.00914
m90n90s90i100l800	0.00627	0.00423	0.00586	0.00951	0.00564
m90n90s120i100l800	0.00687	0.00564	0.00466	0.00819	0.00642
m90n90s150i100l800	0.00931	0.00517	0.00602	0.00671	0.00572

	Test 1	Test 2	Test 3	Test 4	
m90n90s24i100l1600	0.01453	0.00572	0.00587	0.00557	
m90n90s42i100l1600	0.00879	0.00366	0.00519	0.00417	
m90n90s64i100l1600	0.00880	0.00369	0.00410	0.00387	
m90n90s90i100l1600	0.00907	0.00344	0.00353	0.00419	
m90n90s120i100l1600	0.01047	0.00604	0.00375	0.00534	
m90n90s150i100l1600	0.00670	0.00502	0.00600	0.00669	
	Test 5	Test 6	Test 7	Test 8	Average
m90n90s24i100l1600	0.00955	0.01129	0.00858	0.02531	0.01080
m90n90s42i100l1600	0.00687	0.00660	0.00878	0.01312	0.00715
m90n90s64i100l1600	0.00631	0.00517	0.00481	0.00585	0.00532
m90n90s90i100l1600	0.00742	0.00531	0.00542	0.01337	0.00647
m90n90s120i100l1600	0.00729	0.01062	0.00491	0.01121	0.00745
m90n90s150i100l1600	0.00961	0.00982	0.01174	0.01216	0.00847

	Test 1	Test 2	Test 3	Test 4
m90n90s24i150l1600	0.01120	0.00557	0.00585	0.00423
m90n90s42i150l1600	0.00737	0.00529	0.00470	0.00416
m90n90s64i150l1600	0.00854	0.00443	0.00518	0.00421
m90n90s90i150l1600	0.00877	0.00346	0.00299	0.00365
m90n90s120i150l1600	0.00452	0.00295	0.00341	0.00405
m90n90s150i150l1600	0.00446	0.00317	0.00401	0.00405

	Test 5	Test 6	Test 7	Test 8	Average
m90n90s24i150l1600	0.00766	0.00613	0.01108	0.01247	0.00802
m90n90s42i150l1600	0.00775	0.00907	0.00733	0.01436	0.00750
m90n90s64i150l1600	0.00565	0.00645	0.00788	0.01124	0.00670
m90n90s90i150l1600	0.00610	0.00857	0.00507	0.00763	0.00578
m90n90s120i150l1600	0.00382	0.00419	0.00537	0.00996	0.00478
m90n90s150i150l1600	0.00469	0.00452	0.00506	0.01084	0.00510