CHALMERS
UNIVERSITY OF TECHNOLOGY

# PREDICTION OF BRAKE SQUEAL: A DEEP LEARNING APPROACH

## ANALYSIS BY MEANS OF RECURRENT NEURAL NETWORKS

MASTER'S THESIS IN COMPLEX ADAPTIVE SYSTEMS

NADJA GROCHEVAIA

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

MASTER'S THESIS 2020:05

# Prediction of Brake Squeal: A Deep Learning Approach

Analysis by Means of Recurrent Neural Networks

NADJA GROCHEVAIA


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Prediction of Brake Squeal: A Deep Learning Approach
Analysis by Means of Recurrent Neural Networks
NADJA GROCHEVAIA



Supervisor: Staffan Johansson, Volvo Car Corporation
Examiner: Marina Rafajlovic, Department of Physics

Master's Thesis 2020:05
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Prediction of Brake Squeal: A Deep Learning Approach
Analysis by Means of Recurrent Neural Networks
NADJA GROCHEVAIA
Department of Physics
Chalmers University of Technology

## ABSTRACT

Noise, vibration and harshness (NVH) is a principal field of research for the automotive industry. Current research methods of brake NVH involve the finite element method and complex eigenvalue analysis, both of which fall short in terms of their noise prediction capabilities. Lately, research has shifted towards deep learning with the application of machine learning algorithms to detect, characterise and predict noise in commercial brake systems. This thesis investigates the possibility of implementing novel data science techniques to predict the vibrational behaviour of brake structure by means of deep neural network models, more specifically recurrent neural network (RNN) architectures. Two versions of RNN with an encoder-decoder architecture were evaluated: the long short-term memory (LSTM) and the gated recurrent unit (GRU) networks. The networks were applied on two datasets of normal force between the brake pad and the disc, measured in Newton: the sinusoidal data signal that corresponds to the brake squeal and the quiet reference data. The effect of the multifeatured data on prediction accuracy was investigated as well. The results showed that the LSTM network produced the most reliable results on the sinusoidal data signal with a prediction length of 10 ms, which generated a weighted mean absolute percent error (wMAPE) of 61.57% and a mean absolute error (MAE) of the normal force of 0.1647 N. The corresponding results obtained by the GRU model were a wMAPE of 21.77% and a MAE of 0.1804 N. The highest wMAPE and MAE values of 91.01% and 0.0442 N, respectively, were obtained by the LSTM network on the multifeatured sinusoidal data signal with a length of 2.5 ms. In general, shorter prediction lengths generated higher accuracy and lower MAE scores. Moreover, predictions based on multifeatured datasets generated overall slightly better results compared to the single featured data. Overall, the outlook of data-driven applications on friction-induced dynamics seem promising. Future work, should focus on classification of various types of brake signals for a better understanding of the brake squeal phenomenon.

Keywords: AI, ANN, brake squeal, deep learning, encoder-decoder, GRU, LSTM, RNN, time series forecasting.

# ACKNOWLEDGEMENTS

Firstly, I wish to express my sincere appreciation to my supervisor at Volvo Cars, Staffan Johansson, for his reliable guidance, invaluable insight and undivided support. His encouraging words and sense of professionalism allowed me to finish this thesis in a timely manner.

I would also like to express my gratitude toward Tomas Björklund, Deep learning specialist at Volvo Cars, for his technical expertise and patience. Our discussions have had a profound impact on my overall understanding of the subject matter.

Furthermore, I wish to pay special regards and thanks to postdoctoral research fellow Marina Rafajlovic at Chalmers University of Technology, for taking on the role as Examiner.

Finally, I would like to acknowledge my friends and family for their unconditional love and support, in particular my domestic partner John. They have been rallying behind me from the very beginning and this thesis would not have been possible without their unwavering encouragement.

Nadja Grochevaia, Gothenburg, October 2020

# CONTENTS

# List of Figures

# List of Tables

# 1 | INTRODUCTION

The automotive industry is constantly aiming to improve commercial road vehicles with superior safety innovations, quality and comfort. The development process is highly intricate, with numerous components continuously being manufactured and improved in order to meet new industry standards. A principal field of research within new automotive systems is noise, vibration and harshness (NVH), which deals with disturbances experienced by passengers in the cabin of a vehicle [1]. Researchers have identified various forms of NVH, such as aerodynamic, electrical and mechanical. Although there are several causes of NVH, one of the biggest concerns for automotive consumers is the noise and vibrations produced by friction brakes in commercial road vehicles [2]. Problems relating to NVH often lead to considerable costs for both the manufacturer and the consumer; therefore, industry researchers are continuously developing new numerical methods and experimental setups to gain a better understanding of NVH. The current research methodology includes the finite element method and complex eigenvalue analysis, both of which fall short in their noise prediction capabilities with regard to human experience [2].

During the last decade, numerous researchers alongside the automotive industry have been struggling with identifying the root cause of self-excited frictional vibrations in dynamically unstable machine structures [2]. With ever more sophisticated research methods, studies have illustrated the highly nonlinear and chaotic nature of NVH. Today, the majority of industry trials are still being conducted experimentally due to insufficient prediction accuracy of the current numerical methods [3]. Despite the substantial progress made within data-driven sciences, the numerical modelling approach has yet to show any conclusive results in terms of NVH. This has shifted the research focus toward deep learning, where machine learning algorithms are being used to detect, characterise and predict brake noise in commercial brake systems [3].

A recurrent neural network (RNN) is a type of artificial neural network (ANN) that is capable of processing sequential data. The architecture of the RNN enables an internal memory state to change throughout time, i.e., a pattern of the current time series is constantly updated and stored [4]. The two most common RNN units are the long short-term memory (LSTM) and the gated recurrent unit (GRU). A network that processes sequential data is tasked with remembering past observations and uses these to predict future values. This means that a basic RNN model consisting single memory units might perform poorly due to excessive workloads. [4]. In order to circumvent this type of problem the so called encoder-decoder architecture is introduced. Instead of using a single multi-tasking cell, the encoder-decoder model uses a combination of two specialised cells: one for learning past values and another reserved for encoder memory states that is used for predictions of future values [5].

The data acquisition process involves brake signal measurements by means of a standard brake dynamometer, i.e., a set of sensors fitted on to each brake pad measures the pressure at a fixed position [2]. Traditionally, brake noise is recorded using a microphone during a matrix test for a variety of braking applications [6]. Throughout this procedure the brake pad is treated as a passive component, since it does not provide any information of the real brake efficiency, nor does it detect any malfunctions [7]. In this thesis, a new means of brake data acquisition is introduced, made possible by the so called ITT Smart Pad® designed by ITT Friction Technologies. The the embedded sensors in the bake pad collects data in real-time, thus transforming an otherwise passive component into an active one. The ITT Smart Pad® is capable of gathering real-time data of normal force, temperature, torque etc. With this type of information, new modes of analysis can be developed which might lead to a deeper understanding of the principles that governs brake noise [7].

This thesis aims to investigate the possibility of implementing novel data science techniques on classical mechanical engineering problems such as prediction of brake noise by means of deep neural network models. The goal is to determine the adequacy of such models for prediction of time dependent data series, i.e. whether a RNN is suitable for capturing typical brake noise patterns.

The NVH group at Volvo Cars would like to investigate the possibility of applying deep learning methods to friction-excited vibrational structures, such as the physics between brake pads and discs. The objective for the future is to implement a prediction model for mitigation of brake squeal directly into the car itself.

The first paper to provide insight into brake squeal using ANN was published by Stender et al. [3]. Their work showed promising results in applications of deep learning methods for detection, classification and prediction of brake squeal. Nevertheless, classification of brake squeal is excluded from this thesis. Instead, the focus is set on modelling an encoder-decoder RNN for prediction of the normal force signal. Moreover, a comparison between LSTM and GRU units will be conducted, along with an analysis of the maximum predicted signal length without loss of accuracy. Furthermore, a hyperparameter search will be performed to determine the efficiency of the model. Finally, the overall prediction performance of the model with univariate input signals will be compared to that of a multivariate model.

Since the primary objective of this thesis is to investigate the efficacy of deep learning techniques to predict brake noise patterns; hence, conventional research methods, such as FEM and CEA, will not be considered. Given the wide variety of RNN architectures suitable for time series forecasting, this thesis will be limited to the encoder-decoder model. Also, time series forecasting will only be conducted on normal force signals resulting from friction between the brake pad and disc. Each sensor on the ITT Smart Pad® are addressed individually, whereas the instance of all five sensors being engaged simultaneously is treated separately. Furthermore, classification of brake noise based on the fast Fourier transform of a signal is ruled out. Lastly, all computations will be performed on the CPU as in the classical approach, due to relatively small datasets. Thus, usage of the GPU is deemed redundant and excluded from this thesis.

This thesis consists of six chapters. Background presents a brief review of the brake noise phenomenon, followed by an introduction of machine learning and its implementation on brake noise problems. Theoretical prerequisites will go deeper into the architecture of ANN, where the requisite theory for implementation of neural networks is presented. Then, the development of machine learning algorithms, such as the RNN and the encoder-decoder architecture is discussed, along with relevant methods for optimisation and regularisation. In Methodology, the ITT Smart Pad® raw dataset is described, together with machine learning algorithms and data preprocessing methods. Also, the implemented deep learning models are explained in greater detail. In Results, the model comparison and the hyperparameter search are presented, as well as illustrations of predicted signals of varying lengths. Lastly, Conclusion is put forward as a closing remark. Please see appendix A for a more in-depth historical review and appendix B for additional theory.

# 2 | BACKGROUND

This chapter gives a short review of the brake noise problem and the application of machine learning for brake squeal predictions. Although the interest in machine learning is widespread, the coverage of the topic with respect to brake noise is limited [3].

## 2.1 REVIEW OF BRAKE NOISE

Brake noise caused by commercial road vehicles has been an area of concern within the automotive industry for decades. Throughout the years a large body of research has been published on brake noise using increasingly more sophisticated experimental techniques [3, 8, 9, 10]. In 1961, Spurr [9] introduced the first kinematic constraint source model suggesting that brake noise arises due to frictionally induced dynamic instabilities caused by surface asperities. The frictional forces produced by the model varies with time, despite the friction coefficient being constant. Furthermore, North [8] established himself as the first researcher to treat brake squeal as a high-tonal frictionally induced vibration and characterised it as a type of dynamic instability. He also pioneered the use of complex eigenvalue analysis to describe the inherent mechanisms of dynamic instability.

According to Day [2], most brake noises occur during a braking application due to flexural vibrations within the disc or drum of a vehicle. The combination of material and geometric properties of each moving part greatly influence the frequency of the brake noise. For instance, flexural vibrations produced by both the disc and drum of a passenger car tend to have frequency components in the range of 1–16 kHz which is more commonly known as squeal. This type of brake noise is particularly intrusive since it constitutes the most sensitive portion of the human auditory range [2]. For more information, the single mass model with two degrees of freedom is described in appendix B.1.

The automotive brake system is driven into a self-excited state of instability due to complex, non-stationary and multiphysical constraints. Furthermore, the chaotic nature of self-excited instabilities in brake systems tend to generate numerical predictions of limited quality. Today, there is still limited knowledge of the contributing factors to the cause of these instabilities. However, research show that brake squeal is highly nonlinear, multiscalar and unpredictable [3].

In an era of big data and machine learning, the implementation of deep neural networks in brake systems was more or less inevitable. Prediction of a sequence with multiple times steps based on previously observed values is efficiently handled by means of time series forecasting models [11]. Thus, the application of deep learning methods

in automotive brake systems is a step toward mitigation of brake squeal and ultimately, minimised costs.

## 2.2 MACHINE LEARNING FOR BRAKE NOISE

Machine learning has seen an accelerated rate of development in the past few decades and the application of artificial neural networks has improved many aspects of modern day research disciplines. Various research areas such as natural language recognition, image processing [3], object detection, etc., have shown vast improvements as a result of enhanced processing capabilities and the discovery of cutting edge algorithms [4]. However, the overall development of numerical methods for brake systems has stagnated, which has shifted the focus toward implementation of machine learning methods within the field of mechanical engineering. Also, the ability to collect large datasets from commercial brake tests has played a significant role in this development [3].

Huge advancements have been made in autonomous vehicles with cars now capable of guiding themselves through rugged landscapes, avoid pedestrians and keep steady speeds. However, limited progress has been made in brake systems in the past decade with respect to application of machine learning [3]. When developing next generation brakes, major attention is drawn to material and design improvements, as well as obtaining a more stable frictional constant between the brake disc and pad, independent of running conditions. Furthermore, future brake systems powered by deep learning are expected to have the potential of transforming brakes from a mere mechanical component to an active one that incorporates real time software [7]. Thus, deep learning offers the potential capability of teaching the brake system how to act in the next instance based on historical data of the trained network, given the condition to primarily mitigate brake squeal.

The idea of implementing deep learning to predict the propensity of brake squeal has gained traction in recent years. The first paper that gave insight into the brake squeal problem using deep learning was published in 2020 by the aforementioned Stender et al. [3]. Their work described applications of RNN as well as convolutional neural networks for vibrational detection, classification and prediction. The authors used image processing to classify brake noise, which prompts the neural network to train on different images of noise in the frequency-time domain. They observed promising results where the existence of noise could both be predicted and classified, based on a set of specific load of parameters such as pressure, disc velocity, temperature etc.

Time series forecasting emerged from applications on financial time series, where deep learning has been used for over 40 years. This particular application of deep learning has been reviewed by Sezer et al. [12]. The authors studied over 200 scientific papers throughout the period from 2005 to 2019, where they examined the most commonly used models and programming languages for implementation of deep learning. The study showed that more than half of the papers were utilising RNNs, in particular LSTM units, which constituted more than 60% of the applied architectures. Meanwhile, architectures such as convolutional neural networks or deep multiple layer perceptrons

are widely used for classification purposes. According to Sezer et al. [12], the most ubiquitous framework was Python in combination with Keras and TensorFlow libraries which corresponded to 60% of the preferred development environments. Another observation was that LSTM with an encoder-decoder structure tends to provide a more successful forecast.

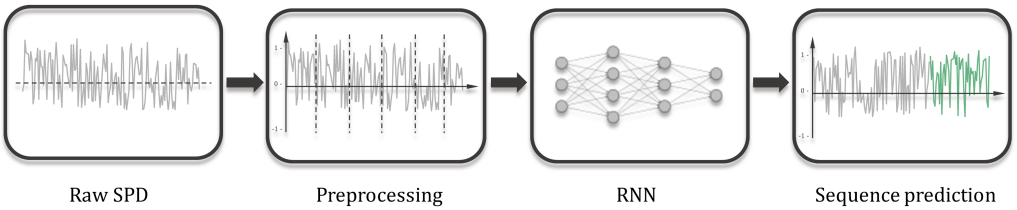



**Figure 2.1:** Flowchart of the time series forecasting process on the raw Smart Pad® data (SPD). The analysed normal force data is collected by ITT Smart Pad® and saved as MAT-files. The preprocessing procedure includes, e.g., normalisation and partitioning of the data into smaller sequences that can be fed into the network. Several RNN models are applied to the prepared data and accuracy scores are generated for analysis.

The overarching strategy for brake signal forecasting has become evident as a result of this literature review. However, the complexity lies within the endeavour of finding the right model fit for brake squeal prediction. Still, the application of various RNN models show great promise, especially in tandem with an encoder-decoder structure. Before time series forecasting can be applied, the raw sequential data generally require a range of transformations. This type of preprocessing play an intricate part in the construction of deep learning models. A diagram of the whole process is illustrated in figure 2.1.

# 3 | THEORETICAL PREREQUISITES

This chapter introduces the core concepts of machine learning with emphasis on two RNN architectures, namely the LSTM and GRU networks. Various regularisation and optimisation methods for mitigation of overfitting in deep learning networks are presented. The aim of this chapter is to familiarise the reader with the requisite theory for comprehending this thesis.

## 3.1 ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANNs) are dynamic computational models that take on different forms based on the type of problem. According to [13], the architecture of the ANN is inspired by interconnected brain cells, or neurons. The nervous impulses between these neurons create a processing network that repeatedly sends information through various connections. This allows separately weighted information to enter the cell body, where it is processed and summed. The output is then transmitted to the connection point which forwards the information to the next neuron. An impulse is generated only if a specific excitation threshold is met, i.e., only a particular set of values is propagated through the network, and over time a distinct configuration representing the task is established [13].

The artificial neuron is a mathematical representation of the biological neuron, and it forms the foundation of an ANN. An illustration of the artificial neuron is presented in figure 3.1.

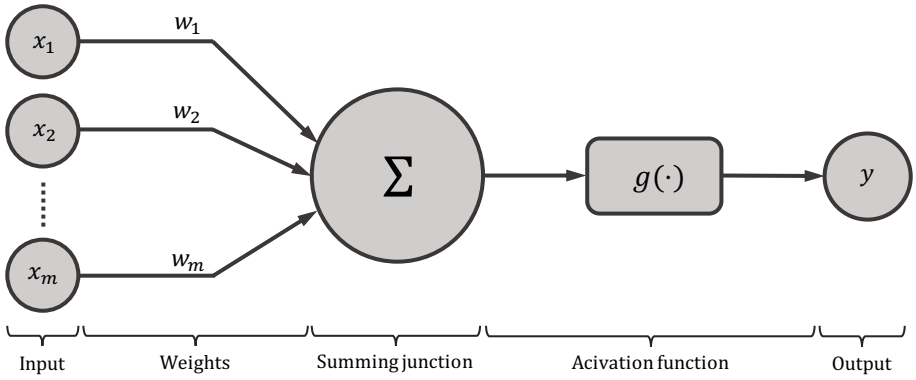



**Figure 3.1:** A graphical representation of an artificial neuron, where $x_i$ for $i = 1,\ldots,m$ are inputs, $w_i$ are weights, $g$ is the activation function and $y$ is the output given by equation (3.1).

As stated in [14], the neuron accepts multiple binary inputs $x_i$ for $i = 1,\ldots,m$ and as-

signs a weight $w_i$ to each input $x_i$ based on their respective priority. The summing junction collects all of the weighted entries and generates an impulse within the cell. Then, the activation threshold $\theta$ determines whether the neuron produces an excitatory potential or remains inhibited. Finally, the activation function $g$ limits the output to a specific range of values for computational efficiency and can be represented by different types of functions such as linear, step, bipolar step, etc. Thus, the output signal $y$ for the artificial neuron is [14]:

$$y = g\left(\sum_{i=1}^{m} w_i x_i - \theta\right).$$

One of the earliest interpretations of a simple connected neural network was the so called perceptron. The perceptron output described in [13] is continuously compared to the target values for all training samples. The perceptron algorithm uses the following updating scheme, also known as Hebb's rule:

$$\begin{aligned}
\hat{y} &\leftarrow \operatorname{sgn}\left(\boldsymbol{w}^T \boldsymbol{x}^{(\mu)} - \theta\right), \\
\boldsymbol{w} &\leftarrow \boldsymbol{w} + \eta\left(y^{(\mu)} - \hat{y}\right)\boldsymbol{x}^{(\mu)}, \\
\theta &\leftarrow \theta + \eta\left(y^{(\mu)} - \hat{y}\right)(-1),
\end{aligned}$$

where $\hat{y}$ is the predicted output from the perceptron, $\boldsymbol{w} = [w_1\ w_2\ \dots\ w_m]^T$ are the weights, $\boldsymbol{x}^{(\mu)} = [x_1^{(\mu)}\, x_2^{(\mu)} \dots x_m^{(\mu)}]^T$ are the inputs and $y^{(\mu)}$ is the target value. Note that the learning rate, which determines how fast the training process approaches convergence is denoted by $\eta \in (0,1)$ and $\mu \in \mathbb{Z}^+$ is the pattern index for the training set [13].

The development of both the artificial neuron and the perceptron gave rise to an alternative ANN, called adaptive linear element (ADALINE). According to [13], ADALINE uses the delta rule, also referred to as the gradient descent method as its primary updating scheme which performs the minimisation of the squared error. Moreover, the squared error between the target value $y^{(\mu)}$ and the output $\hat{y}$ from the network related to the total of $p$ training samples is given by:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{p}\left(y^{(\mu)} - \hat{y}\right)^2. \tag{3.1}$$

Now, given all training samples $p$ provided for training process and the squared error in equation (3.1), the gradient descent method is defined as:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}},$$

where

$$\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = -\sum_{\mu=1}^{p}\left(y^{(\mu)} - \hat{y}\right)\boldsymbol{x}^{(\mu)},$$

represents the gradient of the squared error in equation (3.1) in the case when the activation function $g$ is linear. The tendency to update weights distinguishes ADALINE

from the perceptron [13]. The stopping criterion for ADALINE training process using mean square error (MSE) with respect to all training samples is given by:

$$\bar{E}(\boldsymbol{w}) = \frac{1}{2p}\sum_{\mu=1}^{2}\left(y^{(\mu)} - \hat{y}\right)^2,$$

and

$$\left|\bar{E}\left(\boldsymbol{w}^{\text{current}}\right) - \bar{E}\left(\boldsymbol{w}^{\text{previous}}\right)\right| \leq \epsilon,$$

where $\epsilon$ is the required precision for the weights convergence [13].

A combination of several interconnected layers of perceptrons forms a feed forward ANN called a multilayer perceptron (MLP). A notable feature of MLPs is the addition of nonlinear activation functions [13]. This is discussed in greater details in section 3.3. MLP uses a generalisation of the delta rule known as backpropagation as its primary learning algorithm (see section 3.2). A schematic illustration of the MLP during forward propagation is presented in figure 3.2.

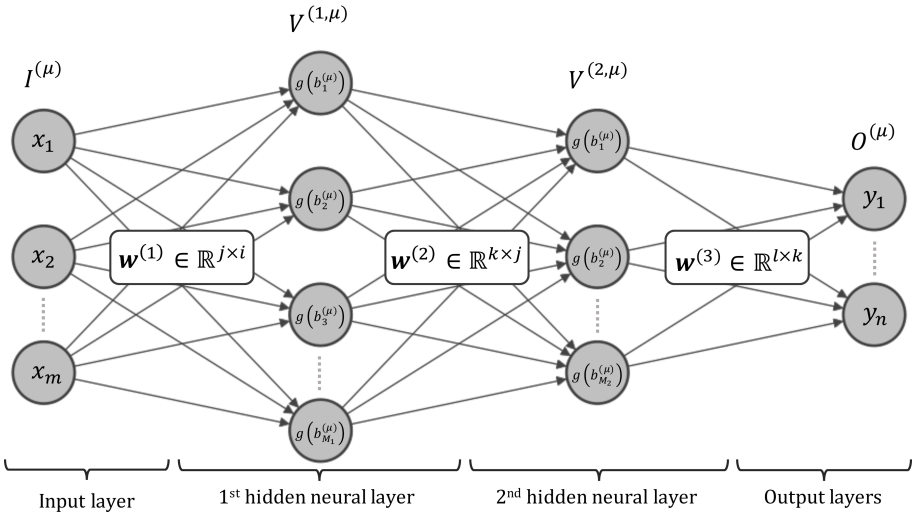



**Figure 3.2:** Architecture of the MLP network consisting of the input layer $I^{(\mu)}$ with $m$ inputs, two hidden layers $V^{(1,\mu)}$ and $V^{(2,\mu)}$, respectively, and the output layer $O^{(\mu)}$ with $n$ outputs. The weight matrix connecting the layers is denoted as $\boldsymbol{w}^{(L)}$ for $L \in \mathbb{Z}^+$ and the indices correspond to the number of neurons in each layer. Note that $L = 3$ in this particular case.

## 3.2 BACKPROPAGATION

The learning algorithm described in [13] is divided into two parts. The forward phase is initiated by the input layer $I^{(\mu)}$. The entries $x_i$ are propagated through the network via the hidden layers $V^{(1,\mu)}$ and $V^{(2,\mu)}$ until the output layer $O^{(\mu)}$ is evaluated [13]. Hence,

forward propagation can be written as [13]:

$$
\begin{aligned}
V_j^{(1,\mu)} &= g\left(b_j^{(\mu)}\right), & b_j^{(\mu)} &= \sum_{i=1}^{m} w_{j,i}^{(1)} x_i - \theta_j^{(1)}, \\
V_k^{(2,\mu)} &= g\left(b_k^{(\mu)}\right), & b_k^{(\mu)} &= \sum_{j=1}^{M_1} w_{k,j}^{(2)} V_j^{(1,\mu)} - \theta_k^{(2)}, \\
\hat{y}_l^{(\mu)} &= g\left(b_l^{(\mu)}\right), & b_l^{(\mu)} &= \sum_{k=1}^{M_2} w_{l,k}^{(3)} V_k^{(2,\mu)} - \theta_l^{(3)}.
\end{aligned}
$$

Moreover, the global performance of training process with respect to all patterns $p$ in the training set is determined by the MSE function as follows:

$$
E\left(y_l^{(\mu)}, \hat{y}_l^{(\mu)}\right) = \frac{1}{2p} \sum_{l=1}^{n} \sum_{\mu=1}^{p} \left(y_l^{(\mu)} - \hat{y}_l^{(\mu)}\right)^2. \tag{3.2}
$$

The MSE function given by equation (3.2), measures the local performance in terms of results produced by the output neuron $l$, with respect to the training sample $\mu$. By dividing the squared error by all samples $p$ in the training set, the MSE for the global performance of the network is obtained. Although there are various error functions that can be used depending on the nature of the problem, the MSE function is still the most common option for regression problems [13].

The backward phase of the learning algorithm involves computation of the derivative of the error function with respect to the weights of $V^{(1,\mu)}$, $V^{(2,\mu)}$ and $O^{(\mu)}$ layers [13]. Thus, the derivative of the error with respect to $w_{l,k}^{(3)}$ of the output layer is:

$$
\begin{aligned}
\frac{\partial E}{\partial w_{l,k}^{(3)}} &= \frac{\partial E}{\partial y_l^{(\mu)}} \cdot \frac{\partial y_l^{(\mu)}}{\partial B_l^{(\mu)}} \cdot \frac{\partial B_l^{(\mu)}}{\partial w_{l,k}^{(3)}} \\
&= \sum_{\mu} \left(y_l^{(\mu)} - \hat{y}_l^{(\mu)}\right) \cdot g'\left(B_l^{(\mu)}\right) \cdot V_k^{(2,\mu)}.
\end{aligned} \tag{3.3}
$$

The chain rule is then applied to the two hidden layers [13]:

$$
\begin{aligned}
\frac{\partial E}{\partial w_{k,j}^{(2)}} &= \frac{\partial E}{\partial y_l^{(\mu)}} \cdot \frac{\partial y_l^{(\mu)}}{\partial B_l^{(\mu)}} \cdot \frac{\partial B_l^{(\mu)}}{\partial V_k^{(2,\mu)}} \cdot \frac{\partial V_k^{(2,\mu)}}{\partial b_k^{(\mu)}} \cdot \frac{\partial b_k^{(\mu)}}{\partial w_{k,j}^{(2)}} \\
&= \sum_{\mu,l} \left(y_l^{(\mu)} - \hat{y}_l^{(\mu)}\right) \cdot g'\left(B_l^{(\mu)}\right) \cdot w_{l,k}^{(3)} \cdot g'\left(b_k^{(\mu)}\right) \cdot V_j^{(1,\mu)}
\end{aligned} \tag{3.4}
$$

and

$$
\begin{aligned}
\frac{\partial E}{\partial w_{j,i}^{(1)}} &= \frac{\partial E}{\partial y_l^{(\mu)}} \cdot \frac{\partial y_l^{(\mu)}}{\partial B_l^{(\mu)}} \cdot \frac{\partial B_l^{(\mu)}}{\partial V_k^{(2,\mu)}} \cdot \frac{\partial V_k^{(2,\mu)}}{\partial b_k^{(\mu)}} \cdot \frac{\partial b_k^{(\mu)}}{\partial V_j^{(1,\mu)}} \cdot \frac{\partial V_j^{(1,\mu)}}{\partial b_j^{(\mu)}} \cdot \frac{\partial b_j^{(\mu)}}{\partial w_{j,i}^{(1)}} \\
&= \sum_{\mu,l,k} \left(y_l^{(\mu)} - \hat{y}_l^{(\mu)}\right) \cdot g'\left(B_l^{(\mu)}\right) \cdot w_{l,k}^{(3)} \cdot g'\left(b_k^{(\mu)}\right) \cdot w_{k,j}^{(2)} \cdot g'\left(b_j^{(\mu)}\right) \cdot x_i^{(\mu)}.
\end{aligned} \tag{3.5}
$$

To obtain an expression for the error with respect to the thresholds, the partial derivatives in equations (3.3)-(3.5) are evaluated with respect to $\theta^{(3)}$, $\theta^{(2)}$, and $\theta^{(1)}$ [13]. All expressions can then be simplified even further through the introduction of local gradients:

$$\begin{aligned}
\delta_l^{(3,\mu)} &= g'\left(B_l^{(\mu)}\right) \cdot \left(y_l^{(\mu)} - \hat{y}_l^{(\mu)}\right), \\
\delta_k^{(2,\mu)} &= \sum_l \delta_l^{(3,\mu)} \cdot w_{l,k}^{(3)} \cdot g'\left(b_k^{(\mu)}\right), \\
\delta_j^{(1,\mu)} &= \sum_k \delta_k^{(2,\mu)} \cdot w_{k,j}^{(2)} \cdot g'\left(b_j^{(\mu)}\right),
\end{aligned}$$

which are substituted into both the error functions with respect to weights and subsequently, thresholds [13]. Thus, the various errors in equations (3.3)-(3.5) are simplified as follows [13]:

$$\begin{aligned}
\frac{\partial E}{\partial w_{l,k}^{(3)}} &= \sum_\mu \delta_l^{(3,\mu)} \cdot V_k^{(2,\mu)}, & \frac{\partial E}{\partial \theta_{l,k}^{(3)}} &= \sum_\mu \delta_l^{(3,\mu)} \cdot (-1), \\
\frac{\partial E}{\partial w_{k,j}^{(2)}} &= \sum_\mu \delta_k^{(2,\mu)} \cdot V_k^{(1,\mu)}, & \frac{\partial E}{\partial \theta_{k,j}^{(2)}} &= \sum_\mu \delta_k^{(2,\mu)} \cdot (-1), \\
\frac{\partial E}{\partial w_{j,i}^{(1)}} &= \sum_\mu \delta_j^{(1,\mu)} \cdot x_i^{(\mu)}, & \frac{\partial E}{\partial \theta_{j,i}^{(1)}} &= \sum_\mu \delta_j^{(1,\mu)} \cdot (-1).
\end{aligned} \tag{3.6}$$

The expressions in equation (3.6) can be used to minimise differentiable error functions in any gradient based optimisation algorithm. The updating scheme for weights and thresholds in [13] are given by:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial E}{\partial \boldsymbol{w}}, \tag{3.7}$$

$$\theta \leftarrow \theta - \eta \frac{\partial E}{\partial \theta}. \tag{3.8}$$

Since the error is continuously propagated backward, these simplifications guarantees that the algorithm is computationally efficient even for deeper networks, such as RNNs [13].

## 3.3 ACTIVATION FUNCTIONS

The choice of the activation function affects the characteristics of an ANN due to the format of the output layer determining the prediction values. According to [13], the backpropagation algorithm is only feasible if the corresponding activation functions are differentiable. In general, activation functions are categorised into two fundamental groups: partially differentiable and fully differentiable. Those of the first group are applicable to feed forward networks, where the first order derivatives are nonexistent, such as the bipolar or Heaviside's step function. The latter group consists of functions with well defined first order derivatives. The three activation functions presented in figure 3.3 are examples of functions of class $\mathbb{C}^\infty$, i.e., they are infinitely differentiable.

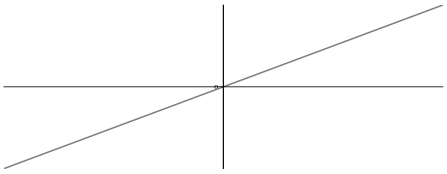
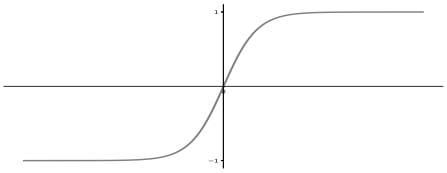
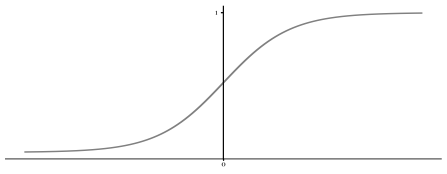
| Plot | Formula |
|---|---|
|  | Linear: $g(x) = x \in \mathbb{R}$ |
|  | Hyperbolic tangent: $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \in [-1, 1]$ |
|  | Sigmoid: $g(x) = \frac{1}{1 + e^{-x}} \in [0, 1]$ |

**Figure 3.3:** Illustration of the three most widely used activation functions in RNNs and their mathematical.

Both sigmoid and hyperbolic tangent functions are applicable during training since both of them are differentiable. However, in this thesis hyperbolic tangent is the primary function used throughout the various networks. Although the linear function has a constant derivative, it is only used in the output layer for scaling purposes. This ensures comparability between predicted values and target values [13].

## 3.4 RECURRENT NEURAL NETWORKS

The recurrent neural network (RNN) was developed as a generalisation of deep learning methods for sequential characters [15]. The RNN differs from previous ANN architectures in that it has a dynamic memory component for its input sequences. Instead of processing each data point separately, the output from a neuron at a previous time step affects the calculations in the current neuron. This dynamic memory feature is the main reason why RNNs tend to outperform static ANNs on problems based on temporally dependent time series [15].

Conventional RNNs have a dynamic looping mechanism that generates internal memory states, also known as hidden states. These can be altered through recurrent connections from the previous output unit onto itself, effectively creating a folded network [13]. Figure 3.4 depicts an unrolled RNN with a single layer and its corresponding features.

Given a continuous sampling quantity $x(t)$ at time instance $t_i \in \mathbb{R}^+$, and under the assumption of uniform sampling $t_{i+1} = t_i + \Delta t$, the univariate times series are the following [16]:

$$\boldsymbol{x} = (x(t_1), x(t_2), \ldots, x(t_n)).$$

Multivariate time series $\boldsymbol{X}(t) \in \mathbb{R}^{m \times n}$ measures multiple quantities, or features, in a similar fashion [16].

Since the entries in a RNN are sequential, each new cell in the network accounts for the historical data from previous time steps. This means that historical data needs to be continuously fed into the network. Thus, the RNN has an additional time-dependent memory state $\boldsymbol{h}(t_i)$ solely dedicated to time series processing. At time $t_i$ the current memory state is evaluated with respect to the current input $\boldsymbol{W}^{(1)}$, which is fused with the memory state of the previous time step weighted with $\boldsymbol{W}^{(2)}$ as follows [15]:

$$\begin{aligned} \boldsymbol{h}(t_i) &= g(\boldsymbol{h}(t_{i-1}), \boldsymbol{x}(t_i)) \\ &= g\big(\boldsymbol{W}^{(1)}\boldsymbol{x}(t_i) + \boldsymbol{W}^{(2)}\boldsymbol{h}(t_{i-1}) + \boldsymbol{\theta}\big), \end{aligned} \tag{3.9}$$

where $g(\cdot)$ is the nonlinear, fully differentiable activation function and $\boldsymbol{\theta}$ denotes threshold values. The updated memory state in the multilayered RNN is then weighted with $\boldsymbol{W}^{(3)}$, producing an output at current time step which is forwarded in time as $\boldsymbol{h}(t_{i+1})$, or to the next layer as $\boldsymbol{y}(t_i)$ [15]. Upon termination, the predicted output produced by the RNN is the following:

$$\hat{\boldsymbol{y}}(t_i) = g\big(\boldsymbol{W}^{(3)}\boldsymbol{h}(t_i) + \boldsymbol{\theta}\big).$$

A schematic illustration of this forward propagation process in presented in figure 3.4.

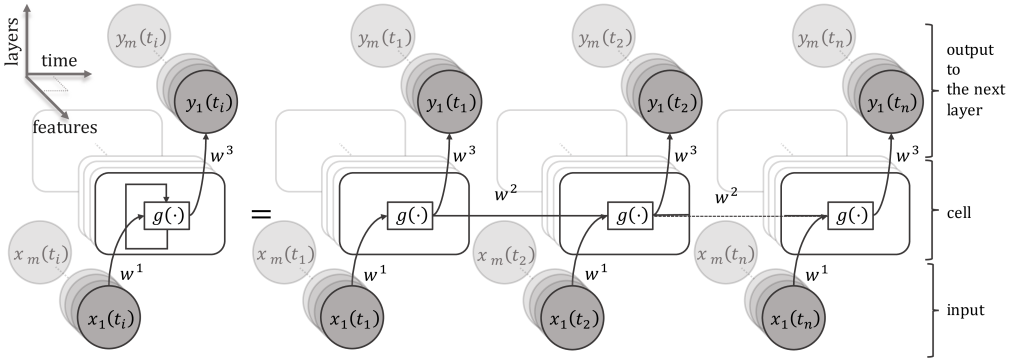



**Figure 3.4:** Illustration of a RNN unrolled along time $t$. The entries at each time step $t_i$ are multi-featured time series $\boldsymbol{X} \in \mathbb{R}^{m \times n}$. The memory state is generated and processed after the entries are weighted by $\boldsymbol{W}^{(1)}$, $\boldsymbol{W}^{(2)}$, and activated by $g(\cdot)$. If the RNN have several stacked layers, the output can serve as input for the next layer.

Due to the dynamic nature of the RNN architecture, the basic backpropagation algorithm presented in section 3.2, required minor modifications to the updating scheme. Thus, the backpropagation through time (BPTT) was developed [17]. It was suggested

to treat the RNN as a basic feed forward network, enabling the application of back-propagation between each time-dependent layer. Hence, the MSE in equation (3.2) is modified to compute the error $E_t$ over a number of time series elements as follows [17]:

$$E_t\big(\boldsymbol{y}(t_i), \hat{\boldsymbol{y}}(t_i)\big) = \frac{1}{2n}\sum_{i=1}^{n}\big(\boldsymbol{y}(t_i) - \hat{\boldsymbol{y}}(t_i)\big)^2, \tag{3.10}$$

where $\hat{\boldsymbol{y}}(t_i)$ is the network estimation and $\boldsymbol{y}(t_i)$ is the real output.

Furthermore, the error gradients of hidden layers evaluated in equations (3.3)-(3.5), are instead represented as gradients of temporal layers in the RNN. Since the depth of the network determines the number of partial derivatives, longer time series tend to produce so called vanishing or exploding gradients, i.e., the partial derivatives of the error with respect to the input values $\boldsymbol{x}(t)$ decreases or increases exponentially [18]. Hence, the computations performed by the network optimiser become increasingly more complex as the distance between $t_i$ and $t_n$ increases, due to more information being processed from the previous time step [18]. The optimisers used in this thesis are described in greater detail in section 3.7.1. To avoid this issue, Hochreiter and Schmidhuber [19] developed an extension of the RNN that is more adept at handling longer time sequences – the long short-term memory (LSTM) network.

### 3.4.1 LONG SHORT-TERM MEMORY NEURAL NETWORK

The LSTM cell introduces a secondary memory state, known as the cell state $\boldsymbol{c}(t_i)$ along with three additional gates [20]. This allows the LSTM unit to determine whether it should hold the memory passed through these dedicated gates, or forget it all together. The architecture of a basic LSTM unit is presented in figure 3.5.

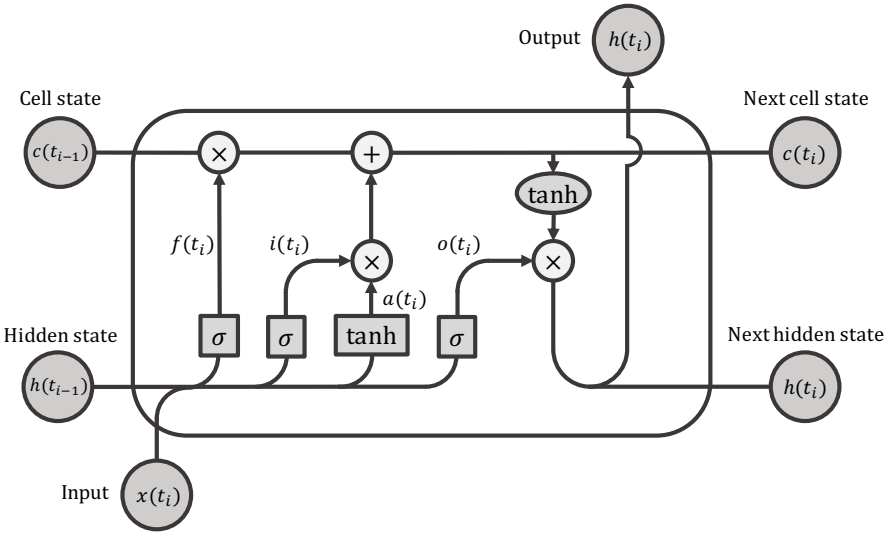



**Figure 3.5:** Architecture of the LSTM-cell. The previous hidden state $\boldsymbol{h}(t_{i-1})$ and inputs $\boldsymbol{x}(t_i)$ go through the gates $\boldsymbol{f}(t_i)$, $\boldsymbol{i}(t_i)$, $\boldsymbol{o}(t_i)$, and updates the memory state $\boldsymbol{c}(t_{i-1})$, which in turn updates the next hidden state $\boldsymbol{h}(t_i)$ prior to being fed into the next LSTM cell.

The input gate $\boldsymbol{i}(t_i)$ filters irrelevant information, preventing perturbations from occurring in the current memory. Similarly, the output gate $\boldsymbol{o}(t_i)$ prevents perturbations from occurring in the subsequent cell [20]. The forget gate $\boldsymbol{f}(t_i)$ on the other hand, opens and closes during learning which allows new information to be added to $\boldsymbol{c}(t_i)$ or the memory to be deleted from $\boldsymbol{h}(t_{i-1})$. This process is known as a constant error carousel and it effectively mitigates the vanishing or exploding gradient problem through truncation [21].

The cell state $\boldsymbol{c}(t_i)$ and the hidden state $\boldsymbol{h}(t_i)$ from current unit is forwarded in time to the next LSTM cell according to [20]:

$$\boldsymbol{f}(t_i) = \sigma\Big(\boldsymbol{W}_{\boldsymbol{x}}^{(f)}\boldsymbol{x}(t_i) + \boldsymbol{W}_{\boldsymbol{h}}^{(f)}\boldsymbol{h}(t_{i-1}) + \boldsymbol{\theta}_f\Big) \qquad \text{forget gate} \tag{3.11}$$

$$\boldsymbol{i}(t_i) = \sigma\Big(\boldsymbol{W}_{\boldsymbol{x}}^{(i)}\boldsymbol{x}(t_i) + \boldsymbol{W}_{\boldsymbol{h}}^{(i)}\boldsymbol{h}(t_{i-1}) + \boldsymbol{\theta}_i\Big) \qquad \text{input gate} \tag{3.12}$$

$$\boldsymbol{a}(t_i) = \tanh\Big(\boldsymbol{W}_{\boldsymbol{x}}^{(a)}\boldsymbol{x}(t_i) + \boldsymbol{W}_{\boldsymbol{h}}^{(a)}\boldsymbol{h}(t_{i-1}) + \boldsymbol{\theta}_a\Big) \qquad \text{input activation} \tag{3.13}$$

$$\boldsymbol{c}(t_i) = \boldsymbol{i}(t_i) \odot \boldsymbol{a}(t_i) + \boldsymbol{f}(t_i) \odot \boldsymbol{c}(t_{i-1}) \qquad \text{cell state} \tag{3.14}$$

$$\boldsymbol{o}(t_i) = \sigma\Big(\boldsymbol{W}_{\boldsymbol{x}}^{(o)}\boldsymbol{x}(t_i) + \boldsymbol{W}_{\boldsymbol{h}}^{(o)}\boldsymbol{h}(t_{i-1}) + \boldsymbol{\theta}_o\Big) \qquad \text{output gate} \tag{3.15}$$

$$\boldsymbol{h}(t_i) = \boldsymbol{o}(t_i) \odot \tanh(\boldsymbol{c}(t_i)) \qquad \text{hidden state} \tag{3.16}$$

where $\odot$ is the element-wise vector product, $\boldsymbol{W}_{\boldsymbol{x}}^{(\text{gate})}$ are the input weights and $\boldsymbol{W}_{\boldsymbol{h}}^{(\text{gate})}$ are the recurrent weights for all respective gates.

The learning process of a LSTM network resembles that of a basic RNN, but with some additional steps [21]. Since there are four regular and four recurrent weights per gate, the total number of gradients amounts to eight in the LSTM backpropagation calculation. Considering the error function in equation (3.10), the partial derivatives of the error for a single time step $t_i$ are the following [21]:

$$\begin{aligned}\frac{\partial E_{t_i}}{\partial \boldsymbol{W}_{\boldsymbol{x}}^{(\text{gate})}} &= \frac{\partial E_{t_i}}{\partial \hat{\boldsymbol{y}}(t_i)} \cdot \frac{\partial \hat{\boldsymbol{y}}(t_i)}{\partial V_j} \cdot \frac{\partial V_j}{\partial V_j^{\text{in}}} \cdot \frac{\partial V_j^{\text{in}}}{\partial \boldsymbol{W}_{\boldsymbol{x}}^{(\text{gate})}} \\ &= (\hat{\boldsymbol{y}}(t_i) - \boldsymbol{y}(t_i)) \cdot \tanh(\boldsymbol{c}(t_i)) \cdot g'(\cdot) \cdot \boldsymbol{x}(t_i),\end{aligned} \tag{3.17}$$

and

$$\begin{aligned}\frac{\partial E_{t_i}}{\partial \boldsymbol{W}_{\boldsymbol{h}}^{(\text{gate})}} &= \frac{\partial E_{t_i}}{\partial \hat{\boldsymbol{y}}(t_i)} \cdot \frac{\partial \hat{\boldsymbol{y}}(t_i)}{\partial V_j} \cdot \frac{\partial V_j}{\partial V_j^{\text{in}}} \cdot \frac{\partial V_j^{\text{in}}}{\partial \boldsymbol{W}_{\boldsymbol{h}}^{(\text{gate})}} \\ &= (\hat{\boldsymbol{y}}(t_i) - \boldsymbol{y}(t_i)) \cdot \tanh(\boldsymbol{c}(t_i)) \cdot g'(\cdot) \cdot \boldsymbol{h}(t_{i-1}),\end{aligned} \tag{3.18}$$

where $\partial \boldsymbol{V}_j = \{\partial \boldsymbol{a}(t_i), \partial \boldsymbol{i}(t_i), \partial \boldsymbol{f}(t_i), \partial \boldsymbol{o}(t_i)\}$ for $j = 1, \ldots, 4$ is the set of partial derivatives for each gate [21]. Note that the superscript *in* refers to the interior product and $g'(\cdot)$ is the derivative of the activation function with respect to gate.

As the LSTM network unrolls along time, the total error of the network is the sum of all gradients:

$$\frac{\partial E_t}{\partial \boldsymbol{W}^{(\text{gate})}} = \sum_i \frac{\partial E_{t_i}}{\partial \boldsymbol{W}^{(\text{gate})}} \tag{3.19}$$

where $\boldsymbol{W}^{(\text{gate})}$ refers to both regular and recurrent weights [21]. The updating scheme for the weights and thresholds utilises the method of stochastic gradient descent according to equations (3.7) and (3.8) and it is discussed further in section 3.7.

In order to measure the performance of a model, analysis of the training error and validation data must be conducted; if the model performs well on the training set and poorly on the validation set, then the model is considered underfitted. Thus, the model learns at a slow rate, or not at all [22]. Conversely, if the model performs well on both the training and validation set, then it is considered well fitted. However, if the performance on the validation increases again passed a certain inflexion point, then the model is overfitted [22]. There are several methods one could implement to circumvent poorly fitted networks; these are described in greater detail in section 3.6.

The purpose of training a network varies greatly depending on the problem. Some areas of concern involve outputting class labels for an input sequence, so called supervised learning, or predicting time steps in a real valued sequence. These kinds of problems are classified as multi-step time series forecasting, or simply sequence to sequence regression problems. This type of prediction is particularly complex, especially for input and output sequences with large discrepancies in length and non-monotonic relationships. As the complexity increases, the more computational resources are required. Hence, Cho [5] developed an alternative version to LSTM, namely the gated recurrent unit (GRU) which is computationally more efficient.

### 3.4.2 GATED RECURRENT UNIT NEURAL NETWORK

The GRU bears a resemblance to LSTM but with fewer parameters; instead of $\boldsymbol{f}(t_i), \boldsymbol{i}(t_i)$ and $\boldsymbol{o}(t_i)$ gates in LSTM, the GRU uses $\boldsymbol{z}(t_i)$ and $\boldsymbol{r}(t_i)$ denoting the update and reset gate, respectively. An illustration of the structural architecture is presented in figure 3.6.

The forward propagation through GRU unit is the following:

$$\boldsymbol{z}(t_i) = \sigma\Big(\boldsymbol{W}_{\boldsymbol{x}}^{(z)}\boldsymbol{x}(t_i) + \boldsymbol{W}_{\boldsymbol{h}}^{(z)}\boldsymbol{h}(t_{i-1}) + \boldsymbol{\theta}_z\Big) \qquad \text{update gate} \qquad (3.20)$$

$$\boldsymbol{r}(t_i) = \sigma\Big(\boldsymbol{W}_{\boldsymbol{x}}^{(r)}\boldsymbol{x}(t_i) + \boldsymbol{W}_{\boldsymbol{h}}^{(r)}\boldsymbol{h}(t_{i-1}) + \boldsymbol{\theta}_r\Big) \qquad \text{reset gate} \qquad (3.21)$$

$$\boldsymbol{a}(t_i) = \tanh\Big(\boldsymbol{W}_{\boldsymbol{x}}^{(a)}\boldsymbol{x}(t_i) + \boldsymbol{r}(t_i) \odot \boldsymbol{W}_{\boldsymbol{h}}^{(a)}\boldsymbol{h}(t_{i-1}) + \boldsymbol{\theta}_r\Big) \qquad \text{memory activation} \qquad (3.22)$$

$$\boldsymbol{h}(t_i) = \boldsymbol{z}(t_i) \odot \boldsymbol{h}(t_{i-1}) + (1 - \boldsymbol{z}(t_i)) \odot \boldsymbol{a}(t_i) \qquad \text{hidden state} \qquad (3.23)$$

where $\odot$ is the element-wise vector product, $\boldsymbol{W}_{\boldsymbol{x}}^{(gate)}$ are input weights and $\boldsymbol{W}_{\boldsymbol{h}}^{(gate)}$ are the recurrent weights for all respective gates. Equations (3.20)-(3.23) indicate the order traversal of each gate in figure 3.6.

The reset gate $\boldsymbol{r}(t_i)$ together with the activation $\boldsymbol{a}(t_i)$ prompts the hidden state to ignore previously held memory; thus, only new information from the input is used due to the reset gate being almost zero. This feature is similar to the forget gate in LSTM, which effectively filters out any irrelevant information from the hidden state. Moreover, the update gate $\boldsymbol{z}(t_i)$ has a similar functionality; it regulates the information that enters the current unit [5].

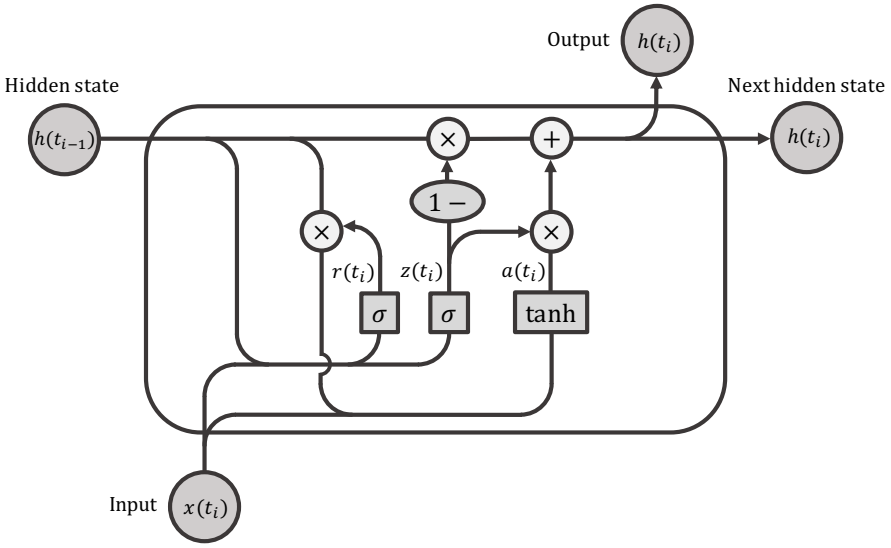



**Figure 3.6:** Architecture of the GRU-cell. The previous hidden state $\boldsymbol{h}(t_{i-1})$ and inputs $\boldsymbol{x}(t_i)$ go through the gates $\boldsymbol{r}(t_i)$, $\boldsymbol{z}(t_i)$ and $\boldsymbol{a}(t_i)$ to update the next hidden state $\boldsymbol{h}(t_i)$, which is fed into next GRU cell.

Despite, GRU having a simpler architecture with fewer parameters compared to LSTM, it still shows great promise in terms of performance. In a study carried out by Chung et al. [23] showed that GRU outperformed LSTM with respect to CPU times and parameter updates for a given dataset.

## 3.5 ENCODER-DECODER

The encoder-decoder model was introduced by Sutskever [24] in 2014. The algorithm was initially intended for natural language processing where it showed tremendous potential, but recently it has been proven that the encoder-decoder model is applicable to other types of data as well [5]. A schematic illustration of the encode-decoder architecture is depicted in figure 3.7.

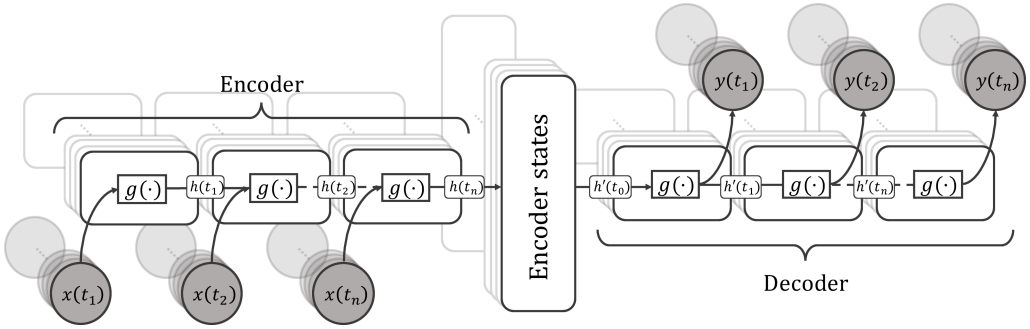



**Figure 3.7:** Architecture of the unrolled encoder-decoder neural network with a single layer. The encoder encodes inputs $\boldsymbol{x}(t_n)$ and generates encoder states which are passed further to the decoder as the initial hidden state. The decoder is able to produce an output sequence $\boldsymbol{y}(t_m)$.

The encoder-decoder architecture consists of two different single- or multilayer RNNs; the first RNN represents the encoder which processes the input $\boldsymbol{x}(t_i)$ of length $t_n$, while $\boldsymbol{h}(t_i)$ is updated sequentially according to equation (3.9) [22]. As the encoder nears the end of the input sequence, the network generates a vector of all past entries that have passed through the hidden state $\boldsymbol{h}(t_n)$, which is transferred to the next RNN. The second RNN constitutes the decoder; it uses $\boldsymbol{h}(t_n)$ generated by the encoder as its initial hidden state $\boldsymbol{h}'(t_0)$ and produces the output sequence $\boldsymbol{y}(t_m)$ [22]. Note that $t_m$ in the decoder can have different lengths compared to the $t_n$ in the encoder.

Moreover, the decoder is constructed in such a way that it accepts empty inputs, and together with $\boldsymbol{h}'(t_m)$ an output $\boldsymbol{y}(t_m)$ is produced and forwarded to the subsequent cell [22]. Then, the decoder output is set to the real target sequence in preparation for model training. In essence, the encoder maps a source sequence of variable length to a fixed-length vector, whereas the decoder maps the representation of the vector back to a target sequence of variable length.

## 3.6 REGULARISATION FOR DEEP LEARNING

The primary objective of regularisation methods is to ensure that an algorithm performs well on any given input and not exclusively on the training data. The most common strategies in machine learning focus on reducing the test error and are collectively referred to as regularisation. In terms of deep learning, these methods are generally based on regularisation of estimators, i.e., reducing the variance at the expense of increased bias. In doing so, a model plagued by overfitting where the variance constitute the bulk of the estimation error, may instead match the true data-generating process. There are currently various forms of regularisation methods and more effective strategies are constantly being developed [4].

### 3.6.1 L2 REGULARISATION

The method of L2-regularisation forces the weights to tend toward zero by adding a penalty term to the error or loss function [4]. Thus, the regularised loss determined by the cross-entropy function is:

$$\tilde{J}\left(\boldsymbol{w}^{(L)}, \boldsymbol{\theta}^{(L)}\right) = \frac{1}{p}\sum_{\mu} E\left(\hat{y}^{(\mu)}, y^{(\mu)}\right) + \frac{\lambda}{2p}\sum_{L}\left\| w^{(L)}\right\|^2, \tag{3.24}$$

where $E$ is the loss function, $p$ is the number of samples, $L$ is the number of layers and $\lambda$ is the regularisation parameter. The norm in equation (3.24) is computed as follows:

$$\left\|\boldsymbol{w}^{(L)}\right\|^2 = w_{11}^2 + w_{12}^2 + \ldots + w_{n^{(L+1)}n^{(L)}}^2.$$

The loss function increases in proportion to the weights. Hence, L2-regularisation conditions the network to favour small weights due to the added penalty term.

### 3.6.2 EARLY STOPPING

For large models with an inherent tendency of overfitting, a common occurrence is that the training error tend to decrease as the validation error increases over time. This

indicates that a model with a superior validation error could be obtained by reverting back to a parameter setting at an earlier point in time. As the validation error improves, a copy of the model parameters is stored. Instead of returning the latest parameters, the algorithm terminates when the currently best recorded validation error cannot be improved further for a specific number of iterations. This regularisation method is more commonly known as the early stopping and it is widely used in deep learning [4].

### 3.6.3 DROPOUT

Dropout is an inexpensive technique that addresses the issues of overfitting and slow computation times in deep neural networks. The algorithm uses the concept of randomly dropping neurons, along with their corresponding connections during training; thus, reinforcing existing favourable neurons in the network and preventing them from excessively co-adapting [4]. During training, dropout samples from a large number of various thinned networks. Then, it averages the predictions of all the thinned networks by applying a single unthinned network with considerably smaller weights. According to Srivastava et. al [25], dropout improves the performance of a neural network for several modes of application.

## 3.7 OPTIMISATION FOR DEEP LEARNING MODELS

The gradient descent method described in section 3.2 is one of the most common optimisation techniques used in deep learning models. The method uses the derivative to determine the minimum of the error function, which in itself is a computationally expensive task, particularly in the context of deep learning. Another disadvantage is that the model tends to slow down significantly for critically small gradients. In order to circumvent this problem, the stochastic gradient descent method (SGD) was introduced. In SGD, the inputs are randomly shuffled prior to each epoch, which theoretically does not always yield smaller errors. However, the oscillating nature of SGD prevents the gradients of the error function from getting stuck in local minima [4].

### 3.7.1 ADAPTIVE MOMENTS ESTIMATION

In 2014, Kingma et. al [26] developed the adaptive moments estimator (ADAM), an optimisation method designed specifically for training deep neural networks. ADAM borrows features from other optimisation algorithms, such as Momentum and RMSprop. The algorithm adjusts the learning rate by means of squared gradients and exploits momentum through moving averages of gradients [4].

ADAM uses estimates of the first- and second-order moment to tweak the learning rates associated with each individual weight of the neural network. These estimates are equivalent to the moving averages $m(t_i)$ and $v(t_i)$, which are defined as follows [26]:

$$m(t_i) \leftarrow \beta_1 \cdot m(t_{i-1}) + \big(1-\beta_1\big) \cdot g(t_{i-1}),$$
$$v(t_i) \leftarrow \beta_2 \cdot v(t_{i-1}) + \big(1-\beta_2\big) \cdot g(t_{i-1})^2,$$

where $g_{t_i}$ is the gradient, and $\beta_1, \beta_2 \in [0,1]$ are hyperparameters that regulate the exponential decay of each moving average. Since the first- and second-order estimates are biased toward zero, these are therefore modified according to [26]:

$$\begin{aligned} \hat{m}(t_i) &\leftarrow \frac{m(t_i)}{1-\beta_1(t_i)}, \\ \hat{v}(t_i) &\leftarrow \frac{v(t_i)}{1-\beta_2(t_i)}. \end{aligned} \tag{3.25}$$

Thus, the updating scheme for previously evaluated moving averages is given by [26]:

$$\boldsymbol{w}(t_i) \leftarrow \boldsymbol{w}(t_{i-1}) - \eta \frac{\hat{m}(t_i)}{\sqrt{\hat{v}(t_i)} + \epsilon},$$

where $\hat{m}(t_i)$ and $\hat{v}(t_i)$ are the modified moving averages in equation (3.25). Note that this updating scheme is suitable for thresholds as well.

## 3.8 PREPROCESSING OF TIME SERIES

The data need to be prepared before any deep learning techniques can be implemented [11]. One common preprocessing technique is scaling, which consists of methods such as normalisation and standardisation. Normalisation of data prevents larger values from overriding smaller ones, which reduces the overall complexity of the data [11]. Thus, by scaling data to a specific range of values, the integrity of the data is maintained without any major loss of information. The inputs $\boldsymbol{x}(t_i)$ are normalised by means of min-max normalisation as follows [27]:

$$x^{\text{norm}}(t_i) = \lambda_1 + (\lambda_2 - \lambda_1)\left(\frac{x(t_i) - x_{\min}}{x_{\max} - x_{\min}}\right), \tag{3.26}$$

where $\lambda_1, \lambda_2$ is the normalisation range. Typically, normalisation ranges are set to $[0,1]$ or $[-1,1]$, depending on the type of problem as well as the deep learning model that is being used. Standardisation is yet another preprocessing method that is widely used on multi-featured data of varying scales. However, this technique is not required in this thesis due the raw data being prestandardised.

# 4 | METHOD

In this chapter, a description of preprocessing of the brake pad data is presented, along with the implementation of the time series forecasting model. Also, the RNN encoder-decoder architectures consisting of LSTM and GRU units are defined.

## 4.1 DATA DESCRIPTION

The brake pad data used in this thesis was collected by ITT Friction Technologies. According to ITT [7], the proprietary brake pad technology (ITT Smart Pad®) is capable of delivering real time data during brake applications. The brake pad can be equipped with several embedded sensors that are distributed on the surface of the backplate facing the brake disc and the friction material. The sensors register signals excited by the contact between the brake pad and the brake disc, such as normal force, shear force and temperature [7]. However, only normal force signals were analysed in this thesis. A schematic illustration of the data collection process is presented in figure 4.1.

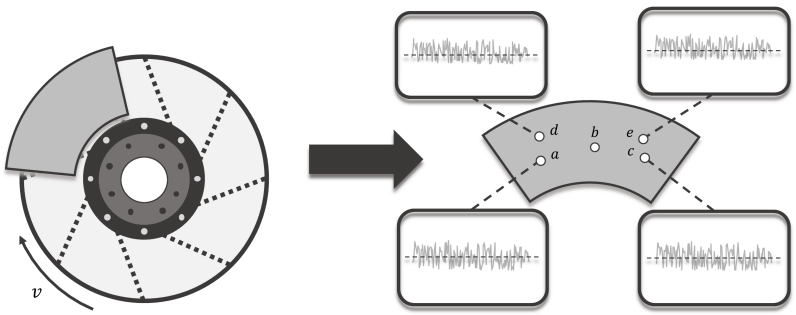



**Figure 4.1:** Schematic illustration of the data collection process by means of ITT Smart Pad®. The several sensors such as (*a*, *b*, *c*, *d* and *e*) are embedded in the brake pad, collect signals such as normal force, shear force, temperature, etc.

In total, data of 20 different brake applications in drag condition with varying pressures, given a constant temperature and velocity, were made available for analysis. The metrics consist of force values in N measured between the Smart Pad® and the brake disc. The duration of each brake application is 25 seconds and the signals generated by the Smart Pad® were sampled at 40 kHz, i.e., a sampling rate $F_s$ of 40k samples/second. Hence, the bandwidth equals 20 kHz, which encompasses the entire frequency spectrum of brake squeal (1–16 kHz). This yields signals with one million data points, at approximately 30 MB each.

The MAT-files containing each brake application were loaded into Python by means of the SciPy library. The files include signals from each individual sensor, along with their

fast Fourier transform (FFT) counterparts. These signals were then stored in separate arrays in preparation for preprocessing.

In order to gain a better understanding of the data, the short-time Fourier transform (STFT) was applied to each signal. The STFT allows for visualisation of time-localised frequencies, whereas the standard FFT shows averaged frequencies over the whole time interval [28]. For visual purposes, two types of signals were chosen: a growing sinusoidal signal with visible periodic self-oscillations, showed in grey in figure 4.2, and a noisy reference signal with a wide range of frequency components, showed in black. Both these signals are depicted in figure 4.2. Note that brake squeal is observed in the sinusoidal signal, particularly noticeable at the end of the signal .

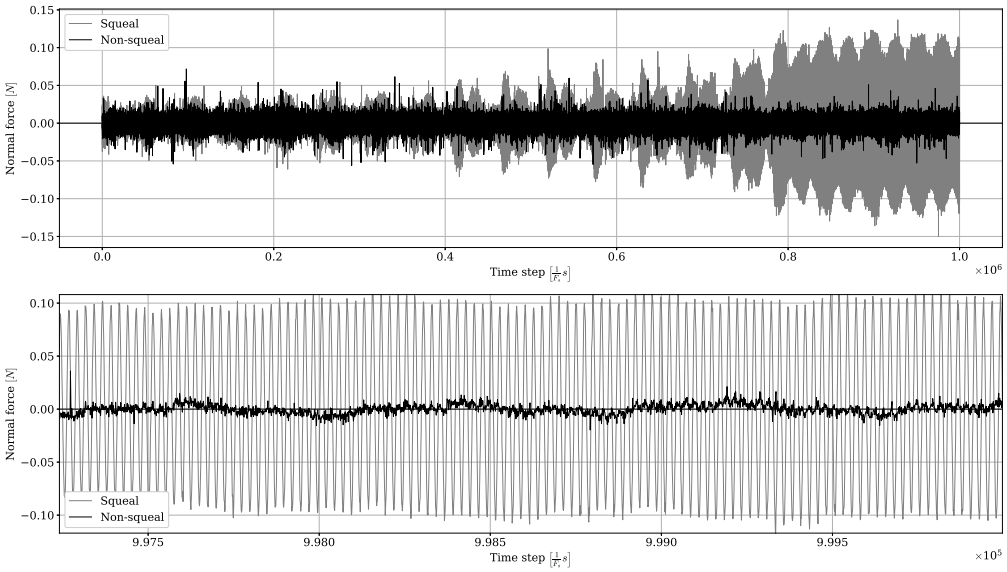



**Figure 4.2:** Two raw normal force signals generated by sensor *a* in the ITT Smart Pad®. The growing sinusoidal signal with visible periodic self-oscillations is shown in grey can be classified as squeal. The noisy reference signal with a wide range of frequency components is shown in black and can be classified as non-squeal signal. The plot on the bottom is a magnification of the one on the top.

Each individual signal from figure 4.2 is presented separately in figure 4.3, in addition to their respective STFTs. The STFT of a signal was generated by means of the signal package of the SciPy library in Python. Note that the signal on the bottom right shows a slowly modulated frequency around 1.6 kHz, which could be classified as squeal through the decibel conversion of the normal force signal.

Moreover, the FFT was performed on the raw normal force signal to verify the amplitude of significant frequency components. The FFT-data utilised in this thesis was provided by ITT Friction Technologies, which could be observed in the figure 4.4. However, this type of analysis is beyond the scope of this thesis. Instead, the objective is to investigate the learning processes of different RNN models and perform time series forecasting on various normal force signals.

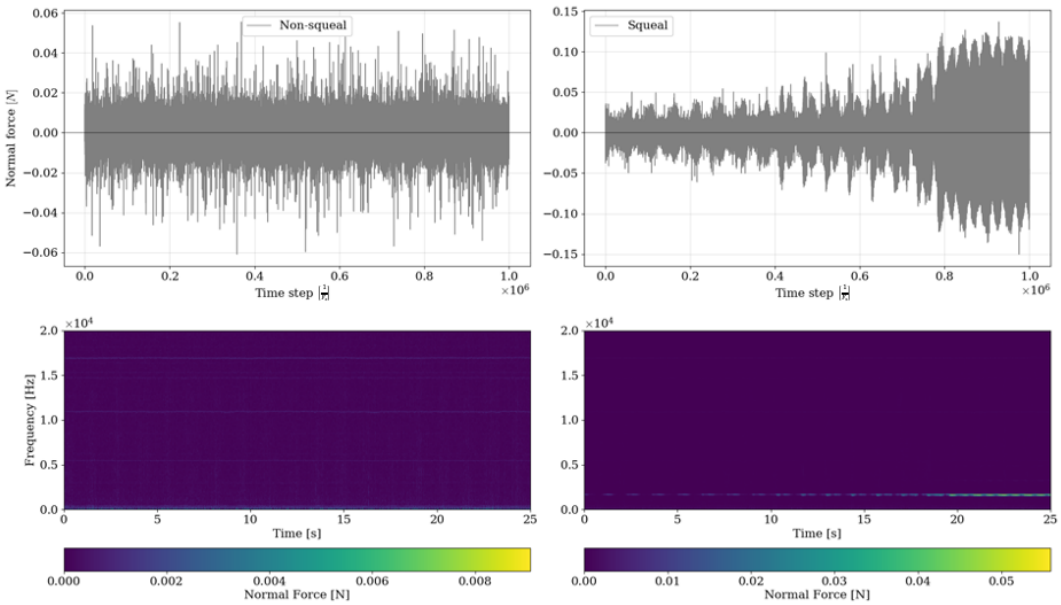



**Figure 4.3:** Two normal force signals with different properties. The signal on the left is the reference signal that is the sum of multiple low amplitude signals with frequencies of approximately 5, 11 and 17 kHz. The signal on the right suggests that frictionally induced sinusoidal vibrations of approximately 1.6 kHz with increased amplitude towards the end of the measurement are present, which is observed in the short time Fourier transform (STFT) to the bottom right. This specific feature can be classified as squeal.

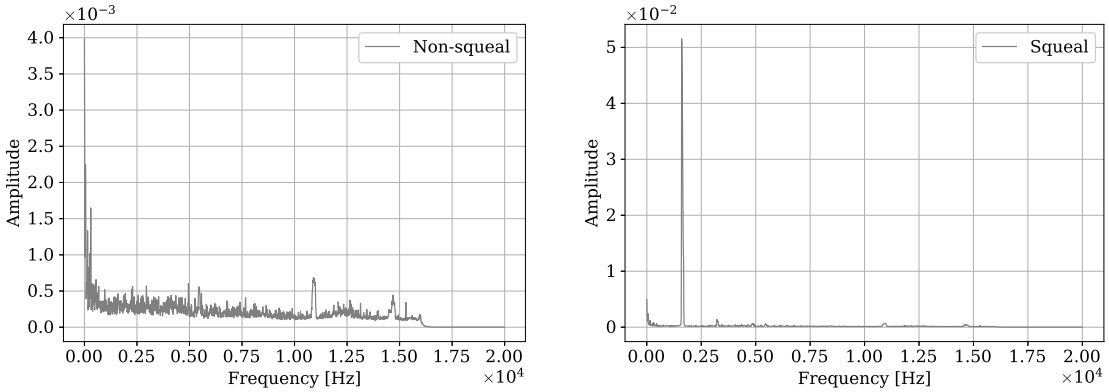



**Figure 4.4:** Fast Fourier transform of two normal force signals. The large magnitude of the frequency component associated with signal to the right suggests the presence of squeal. The signal to the left is a combination of various low-amplitude frequency components.

## 4.2 DATA PREPROCESSING

Prior to implementation of deep learning networks, the time series data was thoroughly analysed. The data should not exhibit properties of white noise, i.e., it should not entirely consist of sequences of random numbers since completely random se-

quences cannot be predicted [11]. Evaluation of the average mean and variance of the signals in the dataset revealed that none of the signals were white noise. See appendix B.2 for the formal definition of white noise.

Each signal in the dataset was normalised between -1 and 1 as described in section 3.8. This procedure was easily implemented by means of the `MinMaxScaler` method of the scikit-learn library in Python. Then, the data was reshaped accordingly to fit the encoder-decoder architecture. The RNN encoder-decoder accepts data as three-dimensional arrays; each sample is stored row-wise with sample lengths determined by the column size and multiple tables constitute the number of features of the reshaped data. The length of each sample corresponds to the sum of the encoder and decoder lengths. Thus, each variable is equivalent to a sampled data point in the sequence. Note that the decoder input is represented by a zero vector which forces the model to memorise the entire input sequence fed into the encoder.

## 4.3 SOFTWARE AND HARDWARE

This project was developed in the integrated development environment Pycharm, with Python 3.7 as the primary interpreter. The open-source neural network library, Keras, was used to construct the various deep neural networks in this thesis. While Keras is highly versatile, it does not offer functionality for tensor products. TensorFlow was used as the backend engine. Hence, Keras in combination with TensorFlow were the ideal choice for modelling intricate neural networks, such as RNNs with encoder-decoder architectures. Moreover, the hardware specifications of the computer system used in this thesis are presented in table 4.1.

| | |
|---|---|
| Operating System: | Windows 10 |
| Processor: | Intel® Core™ i5-6500 CPU @ 3.2 GHz |
| Memory (RAM): | 16.0 GB |
| Graphics Card: | Nvidia GeForce GTX 950 |

**Table 4.1:** Hardware specifications of the computer system used in this thesis.

## 4.4 EXPERIMENTAL SETUP

The experimental data was partitioned into three subsets: train, validation and test in a 70 : 15 : 15 ratio. The training data was used for development and training of the model, i.e., the network learns the parameters and patterns associated with the data [11]. Likewise, the validation data was used to evaluate whether the model has memorised these variables in the training set. Since the model was not subjected to the validation data during training; thus, it acted as an adequate measure of how well the model was fitted. Also, the test data was used to determine the most appropriate deep learning model and for calculating the accuracy of the model.

For proper normalisation of the raw data, the scaler had to be fitted on the training

set first. The normalisation parameters stored in the scaler were then applied to the rest of the data (validation and test). This procedure was essential to avoid an overly optimistic evaluation of the model. If rescaling was performed on the entire dataset simultaneously, the trained model would likely lose some information about future forecasts, which could result in biased predictions [11].

To compare the performance of the two RNN models implemented in this thesis and how multifeatured data affects the prediction accuracy, a simple experiment was setup according to table 4.2. Different lengths of the encoder input and the prediction of several decoder outputs will provide some insight into the model's performance accuracy and time complexity of the analysed algorithms. The experiment was performed on two types of signals, previously described in section 4.1.

| **Encoder input** | 100 | 200 | 300 | 400 |
|---|---|---|---|---|
| **Decoder output** | 100 | 200 | 300 | 400 |

**Table 4.2:** Experimental setup for performance comparison of the LSTM and GRU RNN encoder-decoder models. Encoder input and decoder output time steps are expressed in $1/F_s$, where $F_s$ is the sampling rate of the signal. Thus, these values are equivalent to 2.5, 5.0, 7.5 and 10 ms.

After finding the most efficient model, another type of experiments was conducted to determine the ratio between encoder-decoder lengths that produces the lowest forecast error. The experiment was performed on a quiet signal, i.e., a single featured signal with without high-amplitude oscillations. The experimental setup is presented in the table 4.3.

| | **E. input** | | | |
|---|---|---|---|---|
| | 100 | 200 | 300 | 400 |
| **D. output** | 20 | 40 | 60 | 80 |
| | 50 | 100 | 150 | 200 |
| | 100 | 200 | 300 | 400 |
| | 120 | 240 | 360 | - |

**Table 4.3:** Additional experimental setup for performance of the LSTM RNN encoder-decoder model on a quiet signal. The encoder sequence length is set to values from 100 to 400 data points. The decoder output lengths were chosen as a percentage of the encoder length. The encoder input (E.input) and the decoder output (D.output) time steps are expressed in $1/F_s$, where $F_s$ is the sampling rate of the signal. Thus, these values are equivalent to predictions of time steps between 0.5 ms to 10 ms.

A heat map representation of different length ratios, ranging from 10 to 200, was made to gain a better understanding of the effect of varying encoder-decoder lengths on accuracy of the prediction and its time complexity. Furthermore, the most efficient model with the highest prediction accuracy and lowest time complexity was discovered by means of a hyperparameter search. The model's efficiency was determined

based on parameters such as batch size, number of hidden neurons, learning rate and regularisation parameter.

In order to compare models for different encoder-decoder lengths, some of the hyperparameters were held constant throughout the experiment (see table 4.4).

| Hidden neurons | Learning rate | Batch size | Decay | Dropout | Epochs |
|---|---|---|---|---|---|
| 32 | 0.001 | 128 | 0.0001 | 0.1 | 200 |

**Table 4.4:** Default hyperparameters for model comparison.

The initial choice of the parameters was arbitrary. The number of hidden neurons is the same in all layers for both the encoder and decoder. Therefore, the depth of the architecture is constant throughout this thesis, i.e, the number of trainable parameters are the same, namely 25377 in the LSTM model and 19041 in the GRU model. The number of trainable parameters $P$ was determined based on the number of cell gates $G$, number of hidden neurons $H$ and input dimension $I$:

$$P = G(H(H + I) + H). \tag{4.1}$$

In this case, $G = 4$ for LSTM and $G = 3$ for GRU, given $H = 32$ and $I = 1$. Though, $I = 32$ for the layers in the decoder, since all encoder states are sent to the decoder in the network. By calling `summary()` in Keras, the trainable parameters are easily obtained for the evaluation.

## 4.5 MODEL DEVELOPMENT

The architecture of the RNN encoder-decoder used in this thesis was inspired by the time series forecasting model created by Chevalier [29]. Due to its simplicity, the model was slightly rewritten to fit the purpose of this thesis. Model is depicted in figure 3.7, but instead of one layer of hidden neurons, two layers were implemented in both the encoder and decoder for more effective learning. Note that 32 hidden neurons were used in each layer, but different numbers of hidden neurons were tested as well (see section 4.5.1). The pseudocode for of the RNN encoder-decoder model is presented in the algorithm 1.

The layers are categorised by the index $l = 0, \ldots, L$, where $l = 0$ denotes the layer of input terminals and $l = L$ represents layer of outputs. The encoder and decoder are represented by $E$ and $D$, respectively, and the corresponding cell states are abbreviated as $c$, while the hidden states are denoted by $h$. Note that the LSTM cells in algorithm 1 are completely interchangeable with GRU cells.

The implementation of the model in the algorithm 1 follows a standard procedure, where a framework is first instantiated, i.e., a model is constructed prior to populating it with data. Thus, the framework acts as an operations manual for the various tensors in the model. Moreover, the justification of using a multilayer encoder-decoder model was based on the findings put forward by Lambert [30]. He showed that a

stacked RNN encoder-decoder architecture produced better results compared to its single-layer counterparts. Hence, deep architectures with multiple layers are able to learn complex patterns and generate higher level representations of the data.

**Algorithm 4.1:** Pseudocode for RNN encoder-decoder model

```
Initialise shape of input and output layers;
for l = 1,...,L do
    build encoder: E^(l) ← LSTM cell;
end
Concatenate encoder E^(l) into a single RNN layer;
Set in E: return_state = True ;
Generate encoder cell states E_c^(L) and outputs E_h^(L);
for l = 1,...,L do
    build decoder: D^(l) ← LSTM cell;
end
Concatenate decoder D^(l) into a single RNN layer;
Set in D: return_state = True and return_sequences = True;
Set initial decoder states to encoder states: D_c^(0) ← E_c^(L);
Generate decoder outputs D_h^(L);
Build RNN model: set shape as input and D_h^(L) as output;
```

The passage of cell states and hidden output states from encoder to decoder is governed by the argument `return_state` in Keras. It determines whether the last cell state and the hidden output state are returned or not. Moreover, the argument `return_sequence` gives access to the hidden sate outputs in its entirety. Finally, the model is connected with dense layers that convert the hidden state outputs into prediction outputs $\hat{\boldsymbol{y}}(t)$ by means of the linear activation function $g(\hat{y}) = \hat{y}$. The choice of activation function used in the dense layer is significant since it defines the format of the prediction output. Still, the linear activation function presented in figure 3.3 is the most widely used function for time series regression problems.

### 4.5.1 HYPERPARAMETER SEARCH

After identifying an effective architecture, the model still has to be tuned since the performance could almost always be improved upon [11]. There are various techniques used during hyperparameter search, but for the purpose of this thesis, a sequential search was performed. Each parameter, such as batch size, number of hidden nodes, learning rate $\eta$, regularisation and parameter $\lambda$, were changed one at the time, while keeping all other parameters constant. The diagnostics of the model training history was recorded simultaneously, using training and validation loss curves as an indication of the model performance.

Due to the stochastic nature of the network training process, several runs of the same architecture are required to mitigate the variance of the results. In theory, the estimated model should at least be repeated 30 times or more [11]. However, due to time constraints and limited computer resources, a total of three runs for the hyperparameter search was performed. The averaged results can provide some insight into how

different parameter combinations affect the model performance.

## 4.6 MODEL EVALUATION

The goal of the network is to minimise the error between the actual and the predicted values of the test set. By comparing the behaviours of the loss curves of training versus validation sets, the model can be regularised by some of the methods earlier mentioned in section 3.6.

There are several types of performance measurements for the evaluation of the models. The most common loss measurement for regression is the MSE and it is computed according to equation (3.10). The MSE measures the overall expected deviation or variance between predicted and actual values [4]. Other metrics used for monitoring the performance of the network is the mean absolute error (MAE), which calculates the average magnitude of the errors in a set of predictions, without consideration of their direction. The MAE shows the average absolute distance between the predicted and the actual values, according to [13]:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|\boldsymbol{y}(t_i) - \hat{\boldsymbol{y}}(t_i)\right|, \tag{4.2}$$

where the MAE is expressed in the same unit as the raw data.

The MSE and the MAE are scale-dependant performance metrics, where the calculated error is in the same scale as the dataset [31]. Since the network is applicable on multifeatured datasets simultaneously, comparing performance of the model based on different scales of the data would be odd and difficult to comprehend. However, in this thesis only data of the same scale is investigated and the MAE can still be used for result comparisons between network models.

To compare the forecast accuracy across time series, there are other so called scale-free performance metrics that express the accuracy of the forecast in percent. The most commonly used scale-independent accuracy measurement is the mean absolute percent error (MAPE) [31]:

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{\boldsymbol{y}(t_i) - \hat{\boldsymbol{y}}(t_i)}{\boldsymbol{y}(t_i)}\right|.$$

The MAPE calculates the relative error by dividing the absolute error of actual and predicted values by the true value. This is then averaged over the predicted data values. However, if the data contains values, such as $\boldsymbol{y}(t_i) = 0$ or $\boldsymbol{y}(t_i) \approx 0$, this will result in undefined or extreme values [31]. The solution to this problem is to generalise the MAPE by weighting the percentage errors by the mean of actual values of the time series. This alternative metric is known as MAE/Mean ratio, or weighted MAPE (wMAPE), and it is computed as [31]:

$$\text{wMAPE} = 100\frac{\frac{1}{n}\sum_{i=1}^{n}\left|\boldsymbol{y}(t_i) - \hat{\boldsymbol{y}}(t_i)\right|}{\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{y}(t_i)}.$$

This metric is applicable as long as the time series does not show seasonality or trends [31]. The wMAPE is used as the accuracy parameter in this thesis, measured in percent.

# 5 | RESULTS AND DISCUSSION

In this chapter, the findings obtained the experiments are presented and discussed. The results from the model comparisons of the prediction length and time complexity are summarised, along with the analysis of the hyperparameter search. These are discussed in an attempt to investigate the applicability of brake pad data to recurrent neural networks to predict brake squeal.

## 5.1 MODEL COMPARISON

The aim of this thesis was to find an appropriate RNN model for the time series forecasting and evaluate the applicability of the brake pad signal data on the prediction of brake squeal. In order to compare the prediction capabilities of the LSTM and the GRU RNN encoder-decoder architectures, an experimental scheme was set up based on various prediction lengths, according to table 4.2.

Each model was trained three times for each encoder-decoder length ratio for 200 epochs and the average values were recorded. In order to compare the models, the MSE and the MAE scores, along with the wMAPE, were recorded. Furthermore, the time complexities during the training and evaluation process were recorded as well. Encoder inputs and decoder outputs are given in $1/F_s$, while the MSE and the MAE denote the magnitudes of error between the forecast and the actual values, measured in $N^2$ and N, respectively. The accuracy wMAPE is measured in percent.

The results obtained from the LSTM and the GRU models on the single reference signal (quiet signal) and on the single sinusoidal signal are presented in tables 5.1 and 5.4.

**Table 5.1:** LSTM RNN encoder-decoder results for various input-output ratios. The four models were trained on the reference data signal, with 25 377 trainable model parameters.

| **E. input** | **D. output** | $\mathbf{t}_{\text{train}}$ [s] | **MSE** [$N^2$] | **MAE** [N] | $\mathbf{t}_{\text{test}}$ [s] | Accuracy [%] |
|---|---|---|---|---|---|---|
| 100 | 100 | 685.5 | 0.0152 | 0.0858 | 0.2921 | 58.91 |
| 200 | 200 | 2604 | 0.0235 | 0.1203 | 0.5154 | 54.26 |
| 300 | 300 | 2729 | 0.0350 | 0.1435 | 0.6346 | 29.43 |
| 400 | 400 | 3171 | 0.0494 | 0.1720 | 0.8211 | 28.90 |

The results obtained from the LSTM model applied on the multifeatured reference and the sinusoidal brake data, respectively, are presented in tables 5.5 and 5.6.

In terms of time series forecasting performance, the most efficient network was the

**Table 5.2:** GRU RNN encoder-decoder results for various input-output ratios. The four models were trained on the reference data signal with 19 041 trainable parameters.

| **E. input** | **D. output** | $\mathbf{t}_{\text{train}}$ [s] | **MSE** [$N^2$] | **MAE** [N] | $\mathbf{t}_{\text{test}}$ [s] | Accuracy [%] |
|---|---|---|---|---|---|---|
| 100 | 100 | 1034 | 0.0303 | 0.1260 | 0.2583 | 50.54 |
| 200 | 200 | 3808 | 0.0411 | 0.1511 | 0.3557 | 43.13 |
| 300 | 300 | 9492 | 0.0570 | 0.1889 | 0.6672 | 11.55 |
| 400 | 400 | 10707 | 0.0660 | 0.2038 | 0.7145 | 10.95 |

**Table 5.3:** LSTM RNN encoder-decoder results for various input-output ratios. The four models were trained on the sinusoidal data signal with 25 377 trainable model parameters.

| **E. input** | **D. output** | $\mathbf{t}_{\text{train}}$ [s] | **MSE** [$N^2$] | **MAE** [N] | $\mathbf{t}_{\text{test}}$ [s] | Accuracy [%] |
|---|---|---|---|---|---|---|
| 100 | 100 | 642.3 | 0.0034 | 0.0459 | 0.2095 | 90.68 |
| 200 | 200 | 2054 | 0.0142 | 0.0533 | 0.3126 | 87.67 |
| 300 | 300 | 2607 | 0.0241 | 0.0652 | 0.6091 | 70.40 |
| 400 | 400 | 3125 | 0.0549 | 0.1647 | 0.7914 | 61.57 |

LSTM RNN encoder-decoder model with an accuracy of 90.68% and a MAE of 0.0459 N for the forecast of 100 time steps, corresponding to 2.5 ms of the sinusoidal signal. Similarly, the GRU RNN encoder-decoder model had an accuracy of 89.66% with MAE of 0.0499 N. The accuracy scores and the training times of the LSTM model were better for all prediction lengths compared to the GRU model. The lower MAE scores for the LSTM model indicate that the model is able to find the right pattern in the data more often than the GRU model. Analysis of the time complexities during training of the models revealed that the LSTM RNN was more efficient than the GRU RNN, despite having more parameters, as previously discussed in the section 3.4.2. However, while evaluating the time complexity of the test set, the GRU RNN performed slightly better on a select few encoder-decoder ratios.

The same experimental setup was used on the reference data signal for investigation of the network's ability to learn and predict other types of signal patterns. The obtained results were inadequate since the signal was more or less random compared to the signal with high-amplitude self-oscillations. These types of patterns are more complex and therefore, more difficult to process by the DNNs. The LSTM network is still superior to the GRU network, producing better results in terms of prediction accuracy and time complexity. It seems that LSTMs better suited for capturing long term temporal dependencies. Thus, the LSTM RNN encoder-decoder produced better results for all variations of input-output ratios.

According to the theory in section 3.4.2, networks comprised of GRU units should be faster than LSTM networks, due to fewer trainable parameters, especially for shorter input sequences. However, this was only reflected in the evaluation times, where the GRU RNN was marginally faster than the LSTM RNN. In terms of accuracy, the GRU RNN performed notably worse compared to the LSTM RNN. It seems that the trainable parameters play a significant role in performance of accuracy measures between

**Table 5.4:** GRU RNN encoder-decoder results for various input-output ratios. The four models were trained on the sinusoidal data signal with 19 041 trainable model parameters.

| **E. input** | **D. output** | $\mathbf{t}_{\text{train}}$ [s] | **MSE** [$N^2$] | **MAE** [N] | $\mathbf{t}_{\text{test}}$ [s] | Accuracy [%] |
|---|---|---|---|---|---|---|
| 100 | 100 | 818.3 | 0.0039 | 0.0499 | 0.2027 | 89.66 |
| 200 | 200 | 2098 | 0.0163 | 0.0939 | 0.3175 | 85.57 |
| 300 | 300 | 8876 | 0.2356 | 0.1651 | 0.4311 | 66.91 |
| 400 | 400 | 9203 | 0.1871 | 0.1804 | 0.5913 | 21.77 |

the two models. Furthermore, the previously mentioned research on LSTM and GRU was conducted in the context of language modelling. Thus, these results might not be completely comparable with the brake signal predictions in this thesis.

Following experiment investigated the effect of using multifeatured data signal for time series prediction. The experiment was performed on a reference signal and a sinusoidal signal. The additional four features were extracted from the embedded sensors (b,c,d and e) on the brake pad (see figure 4.1). The results of the LSTM RNN encoder-decoder based on five signals are presented in tables 5.5 and 5.6.

**Table 5.5:** LSTM RNN encoder-decoder results for various input-output ratios of the multifeatured reference data signal.

| **E. input** | **D. output** | $\mathbf{t}_{\text{train}}$ [s] | **MSE** [$N^2$] | **MAE** [N] | $\mathbf{t}_{\text{test}}$ [s] | Accuracy [%] |
|---|---|---|---|---|---|---|
| 100 | 100 | 699.3 | 0.0161 | 0.0909 | 0.3012 | 61.24 |
| 200 | 200 | 1643 | 0.0308 | 0.1311 | 0.5581 | 51.58 |
| 300 | 300 | 2795 | 0.0305 | 0.1325 | 0.6791 | 32.88 |
| 400 | 400 | 3504 | 0.0548 | 0.1806 | 0.8251 | 30.17 |

**Table 5.6:** LSTM RNN encoder-decoder results for various input-output ratios of the multifeatured sinusoidal data signal.

| **E. input** | **D. output** | $\mathbf{t}_{\text{train}}$ [s] | **MSE** [$N^2$] | **MAE** [N] | $\mathbf{t}_{\text{test}}$ [s] | Accuracy [%] |
|---|---|---|---|---|---|---|
| 100 | 100 | 664 | 0.0032 | 0.0442 | 0.2512 | 91.01 |
| 200 | 200 | 2165 | 0.0199 | 0.0556 | 0.3362 | 89.93 |
| 300 | 300 | 2632 | 0.0309 | 0.0691 | 0.6198 | 74.37 |
| 400 | 400 | 3190 | 0.0624 | 0.1653 | 0.8010 | 55.91 |

According to the obtained results in tables 5.5 and 5.6, the multifeatured data provided, on average, more accurate predictions than single featured data. However, the training time was slightly longer, and due to time constraints following experiments were solely performed on single signal data for time series predictions.

An alternative experiment was setup according to table 4.3, where the effect of various encoder inputs and decoder outputs on prediction accuracy was recorded. The

results of these trials are presented in table 5.7. The various prediction lengths varied between 20 and 400. Since prediction lengths of 400 time steps produced relatively low accuracy scores, even longer predictions were excluded from this thesis. For the sake of computational efficiency, each examined architecture was only trained once.

**Table 5.7:** LSTM RNN encoder-decoder model performance results for additional 15 input-output ratios of the reference data signal. Each model was trained for 200 epochs.

| **E. input** | **D. output** | $\mathbf{t}_{\text{train}}$ [s] | **MSE** [$N^2$] | **MAE** [N] | $\mathbf{t}_{\text{test}}$ [s] | Accuracy [%] |
|---|---|---|---|---|---|---|
| 100 | 20 | 514.7 | 0.0115 | 0.0748 | 0.1861 | 66.52 |
| 100 | 50 | 479.8 | 0.0145 | 0.0789 | 0.1901 | 64.14 |
| 100 | 100 | 685.5 | 0.0152 | 0.0858 | 0.2921 | 58.91 |
| 100 | 120 | 749.3 | 0.0264 | 0.1130 | 0.3031 | 47.12 |
| 200 | 40 | 1021 | 0.0111 | 0.0801 | 0.3817 | 65.86 |
| 200 | 100 | 2050 | 0.0201 | 0.0867 | 0.4262 | 61.11 |
| 200 | 200 | 2604 | 0.0235 | 0.1203 | 0.5154 | 54.26 |
| 200 | 240 | 1830 | 0.0331 | 0.1354 | 0.5176 | 40.54 |
| 300 | 60 | 2320 | 0.0128 | 0.0786 | 0.5055 | 50.97 |
| 300 | 150 | 2109 | 0.0152 | 0.0877 | 0.5167 | 48.33 |
| 300 | 300 | 2729 | 0.0350 | 0.1435 | 0.6346 | 29.43 |
| 300 | 360 | 2927 | 0.0561 | 0.1860 | 0.6401 | 29.19 |
| 400 | 80 | 3032 | 0.0300 | 0.1212 | 0.5497 | 47.54 |
| 400 | 200 | 2859 | 0.0211 | 0.1068 | 0.6579 | 43.99 |
| 400 | 400 | 3171 | 0.0494 | 0.1720 | 0.8211 | 28.90 |

The alternative experiment showed that longer encoder-decoder sequences result in less accurate predictions and worse time complexities. Note that training and evaluation times increase in proportion to the complexity of the task. The study was also conducted on shorter encoder-decoder sequences to gain a better understanding of how encoder input lengths affect the prediction accuracy. The results are summarised as heatmaps in figures 5.1 and 5.2. The heatmaps illustrate the distribution of the model's performance scores (MSE) and the evaluation times for the analysed encoder-decoder ratios.

Heatmaps provide a visual representation of how different encoder-decoder lengths affect the prediction performance of the model and its evaluation time. The observations from the heatmap in figure 5.1 indicate that shorter encoder inputs yield lower accuracy scores for the constant decoder output length. However, some of the MSE scores were lower for the same decoder output prediction, despite encoder lengths being longer. This might be due to the stochastic nature of the model, as well as the fact that the models only ran once. The heatmap in figure 5.2 shows that longer encoder-decoder lengths require longer evaluation times in general.

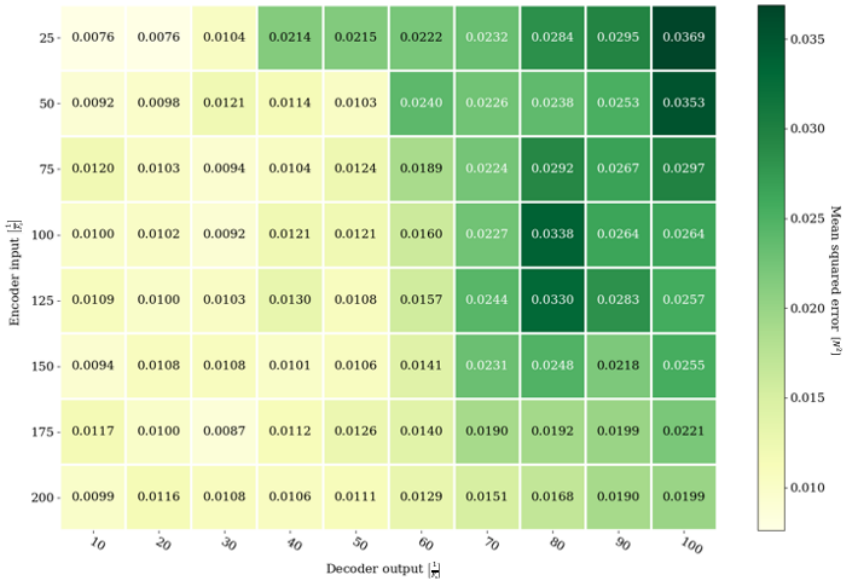



**Figure 5.1:** Heatmap of the MSE scores for various input-output ratios with the LSTM RNN encoder-decoder model. The encoder lengths vary from 25 to 200, while the decoder lengths vary from 10 to 100.


**Figure 5.2:** Heatmap of the evaluation time $\mathbf{t}_{\text{test}}$ scores for various input- output ratios with the LSTM RNN encoder-decoder model. The $\mathbf{t}_{\text{test}}$ was collected during the evaluation of the test set. The encoder lengths vary from 25 to 200, while the decoder lengths from 10 to 100.

It is difficult to get an intuitive sense of the learning process and discern potential over-

fitting from the results above. Thus, some of the encoder-decoder ratios have been visualised together with their learning curves. Figures 5.3-5.6 allows for a visual interpretation of the predictions for the 100-20 and 100-100 input-output ratios, along with their respective MSE loss function curves using early stopping method described in section 3.6.2.

Figure 5.3 shows the plot of the test dataset prediction, in addition to the learning curve. The prediction, previously unseen by the network, follows the real data signal well. The performance score for the 100-20 input-output ratio, presented in table 5.7, show that the accuracy of the model is 66.52 %. The learning curves are converging simultaneously towards the same loss function value, which is an indication of a well fitted model.

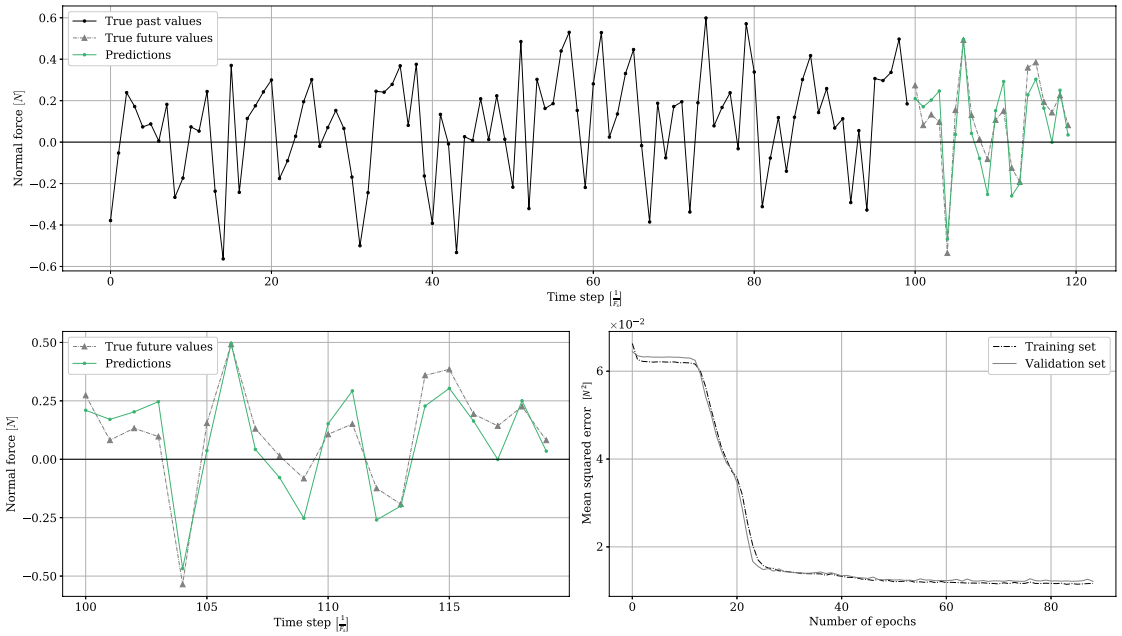



**Figure 5.3:** LSTM RNN encoder-decoder prediction on the reference data signal with a 100-20 input-decoder ratio, in addition to the learning curve. The learning curve demonstrates a well fitted model.

Furthermore, the prediction, previously unseen by the network, in figure 5.4 follows the real data signal very well. The learning curve converges quickly towards the minimum error due to the low complexity of the problem, and the obtained accuracy score measured 93.93%, which is the highest accuracy obtained in this entire thesis.

The prediction in figure 5.5 follows the real data signal fairly well, with an accuracy of 58.91%. However, the network fails to map some of the signal points, which indicates that the model is underfitted and might require additional training or other hyperparameters. The time series prediction in figure 5.6 follows the real, previously unseen, data signal well with an accuracy score of 90.68%.

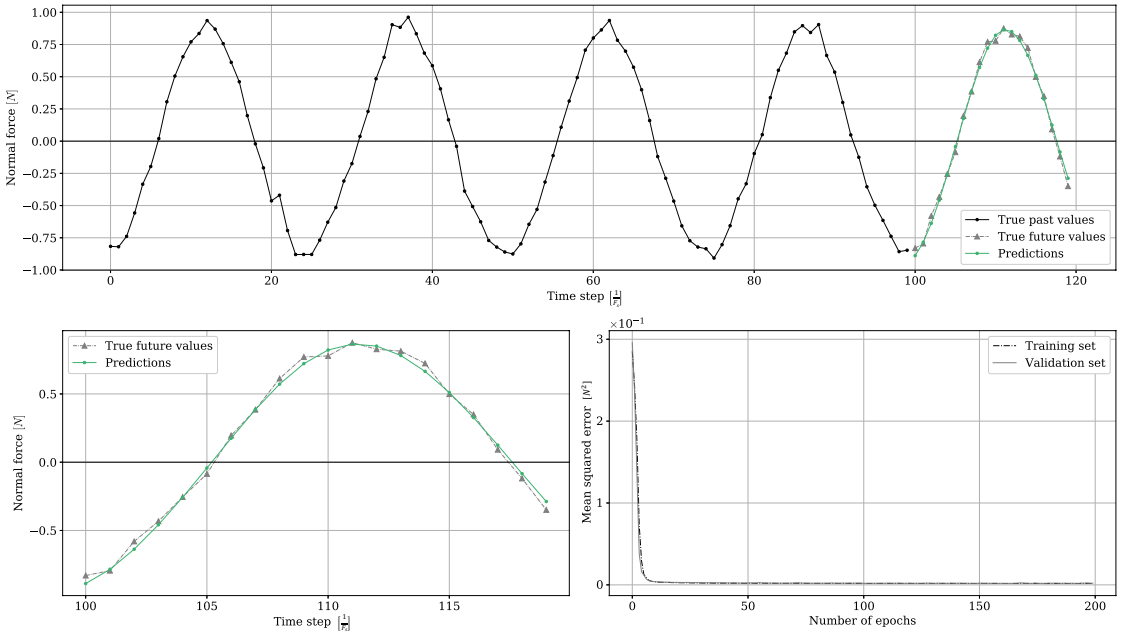



**Figure 5.4:** LSTM RNN encoder-decoder prediction on the sinusoidal data signal with a 100-20 input-output ratio, in addition to the learning curve. The learning curve demonstrates a well fitted model.

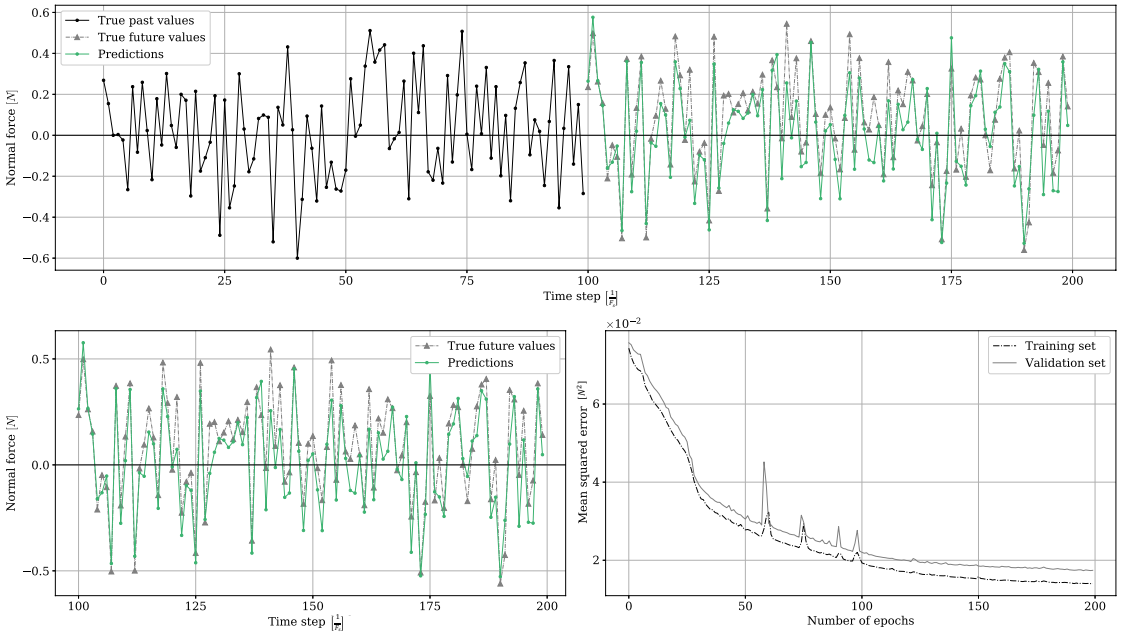



**Figure 5.5:** LSTM RNN encoder-decoder prediction on the reference signal with a 100-100 input-output ratio, in addition to the learning curve. The learning curve demonstrates a somewhat underfitted model that requires further training.

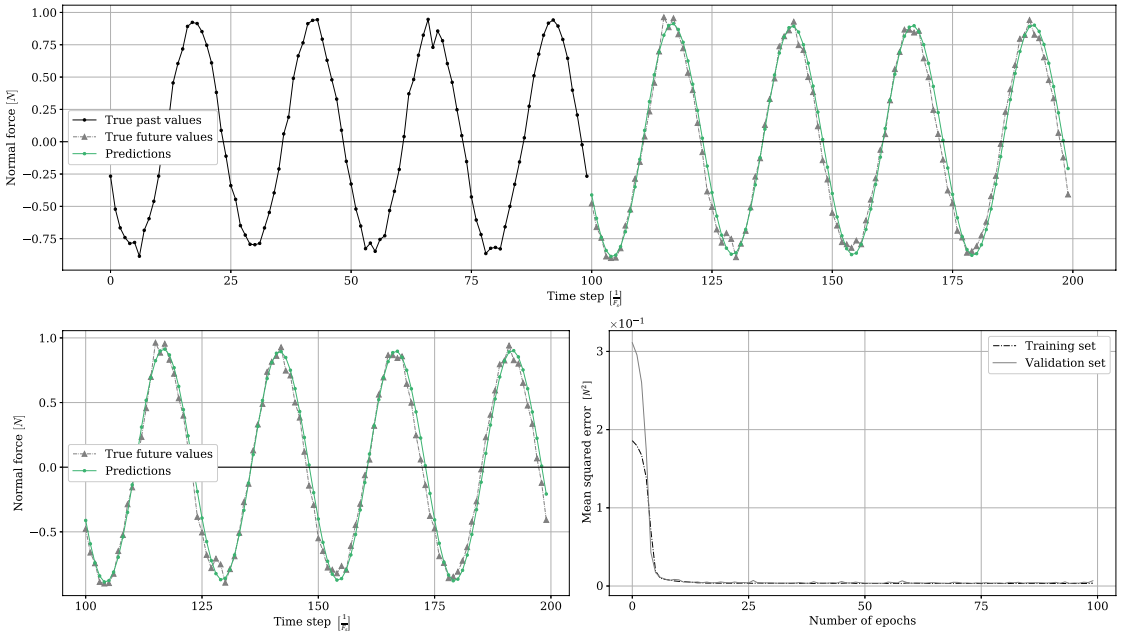



**Figure 5.6:** LSTM RNN encoder-decoder prediction on the sinusoidal data signal with a 100-100 input-output ratio, in addition to the learning curve. The learning curve demonstrates a well fitted model.

## 5.2 HYPERPARAMETER SEARCH

The results from the hyperparameter search are summarised in table 5.8. A total of four additional values were tested for each of the four parameters. For the sake of comparison, all other variables in the model were held constant during this exercise, according to table 4.4. The models were trained for 100 epochs.

### 5.2.1 BATCH SIZE

According to the batch size analysis, the lowest MSE and MAE scores were provided if the data is divided into eight samples per training iteration. To gain a better understanding of the effects the batch sizes have on the training process, the learning curves are presented in figure 5.7. However, smaller batch sizes resulted in much higher time complexities for the learning process, which is observed in the figure 5.8b.

### 5.2.2 NUMBER OF HIDDEN NODES

Analysis of the number of hidden neurons showed that the lowest MSE and MAE scores were provided for the network with 128 hidden nodes in its architecture. The effects of varying the number of hidden nodes has on the training process are presented in figure 5.9. The learning process was slower when using fewer hidden nodes, since the network still did not converge after reaching 100 epochs. Though, using too many hidden nodes makes the network too complex, causing fluctuations in the learning curve. This is observed in figure 5.8a.

**Table 5.8:** Hyperparameter search results. The values are averaged over three runs. The best results are highlighted in bold, holding all other parameters constant (see table 4.4).

**(a)** Batch size

| Batch size | **MSE** [N$^2$] | **MAE** [N] |
|---|---|---|
| **8** | **0.0117** | **0.0760** |
| 16 | 0.0121 | 0.0776 |
| 32 | 0.0122 | 0.0778 |
| 64 | 0.0139 | 0.0847 |
| 256 | 0.0325 | 0.1329 |

**(b)** Number of hidden nodes

| Neurons | **MSE** [N$^2$] | **MAE** [N] |
|---|---|---|
| 16 | 0.0327 | 0.1334 |
| 32 | 0.0234 | 0.1096 |
| 64 | 0.0177 | 0.0972 |
| 100 | 0.0204 | 0.0998 |
| **128** | **0.0158** | **0.0903** |

**(c)** Learning rate, $\eta$

| $\eta$ | **MSE** [N$^2$] | **MAE** [N] |
|---|---|---|
| 0.00008 | 0.0534 | 0.1866 |
| 0.0001 | 0.0494 | 0.1779 |
| 0.0008 | 0.0184 | 0.0955 |
| 0.001 | 0.0296 | 0.1267 |
| **0.01** | **0.0128** | **0.0799** |

**(d)** Regularisation parameter, $\lambda$

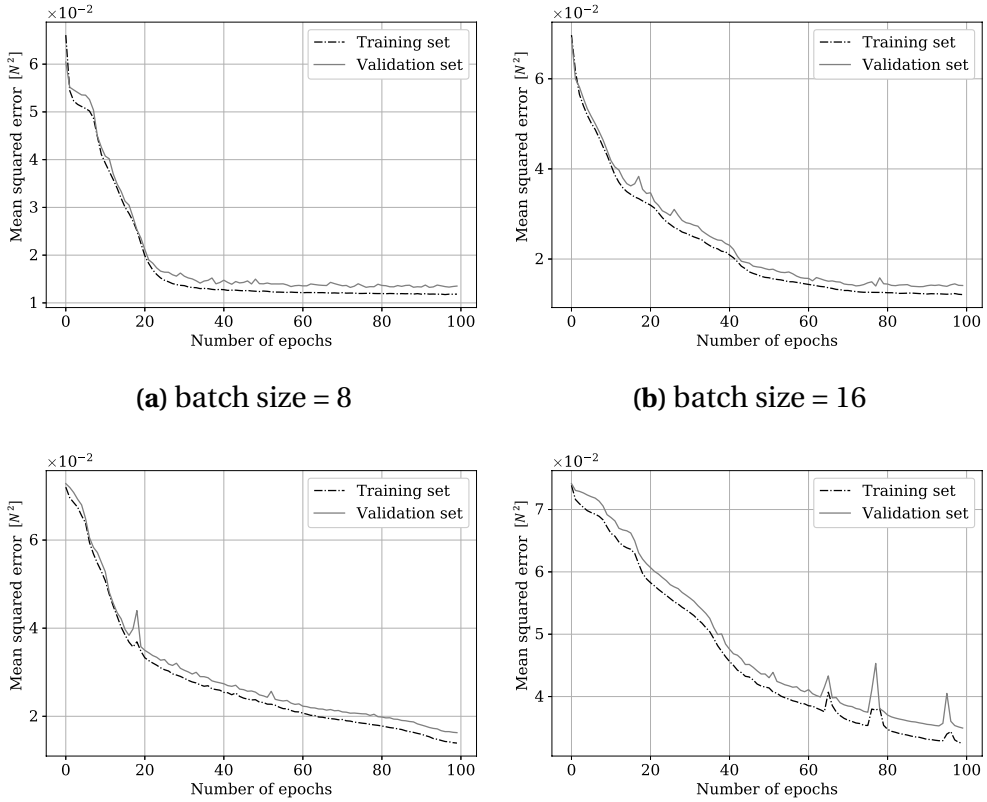
| $\lambda$ | **MSE** [N$^2$] | **MAE** [N] |
|---|---|---|
| **0.000001** | **0.0288** | **0.1184** |
| 0.00001 | 0.0361 | 0.1256 |
| 0.0001 | 0.0734 | 0.2179 |
| 0.001 | 0.0751 | 0.2248 |
| 0.01 | 0.0751 | 0.2255 |

### 5.2.3 LEARNING RATE

Analysis of the learning rate showed that the lowest MSE and MAE scores were obtained for learning rates of 0.01. The results are represented by learning curves, presented in figure 5.11. It is evident that the learning rate affects the rate of the network convergence. Small learning rates, such as $\eta = 0.0001$ was proven to be too slow. This would require training much longer than 100 epochs. Though, a learning rate of 0.01 generated spikes in the learning curve, it still produced the lowest MSE score.

### 5.2.4 REGULARISATION PARAMETER

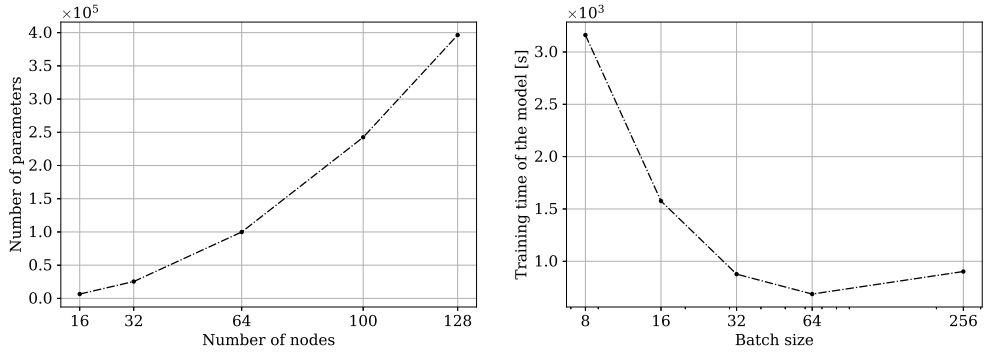
According to the analysis of the regularisation parameter, the lowest MSE and MAE scores were obtained for the network using a regularisation parameter of 0.000001. The findings are summarised in figure 5.12.




**(a)** batch size = 8 **(b)** batch size = 16

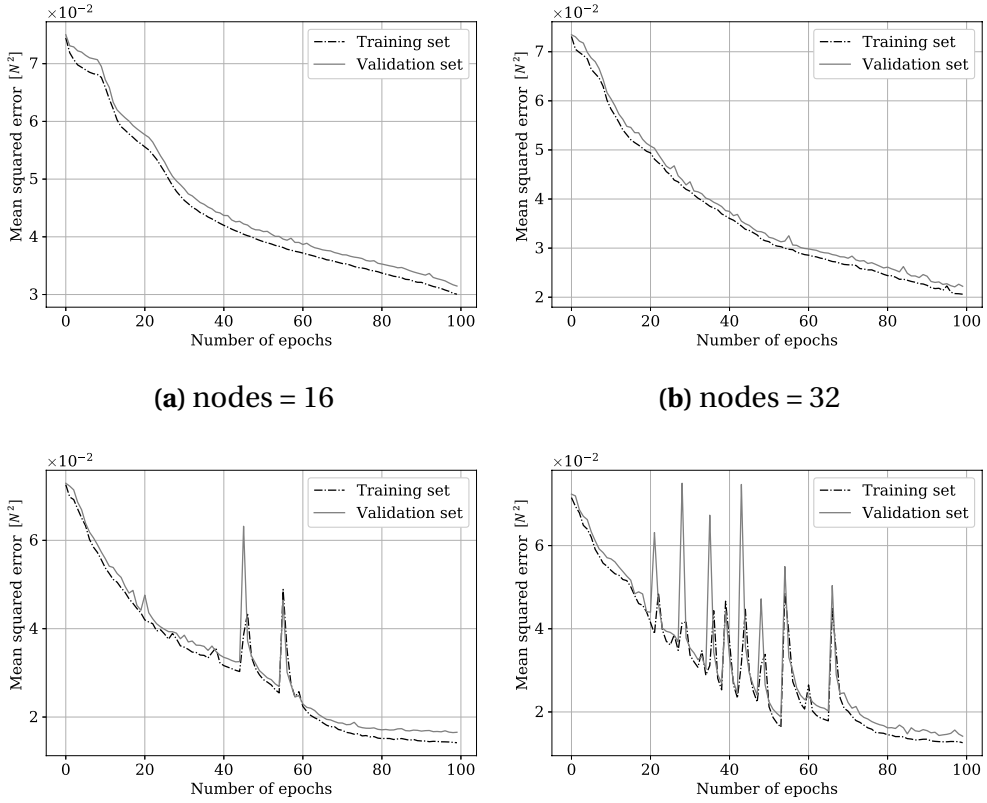
**(c)** batch size = 64 **(d)** batch size = 256

**Figure 5.7:** Learning curves of training and validation sets for various batch sizes. Convergence is reached faster with smaller batch sizes, which also produces the lowest MSE score. The sudden spike in the learning curve is a sign of an underfitted model, i.e., the training and validation sets do not provide enough information to efficiently learn the problem.




**(a)** **(b)**

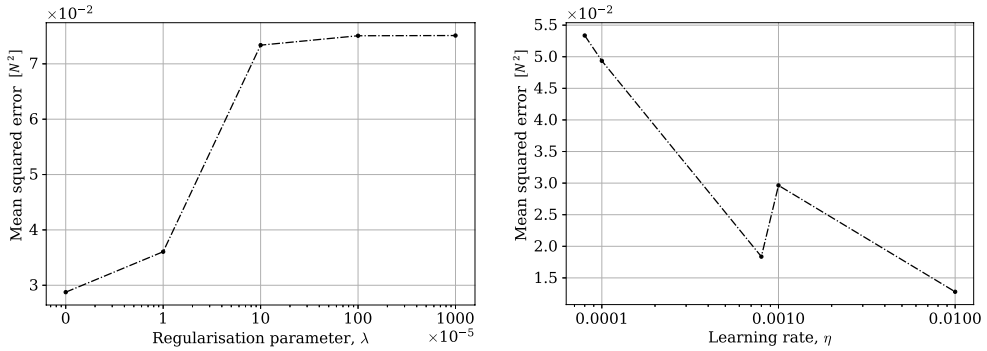
**Figure 5.8:** Effects of tuning hyperparameters. Panel (a) shows the effect of increasing number of hidden nodes on the number of trainable parameters. Panel (b) shows the effect of increasing batch size on the time complexity of the training process. Since this analysis is directly linked to the computational complexity of the algorithm, these results were expected.




**(a)** nodes = 16 **(b)** nodes = 32

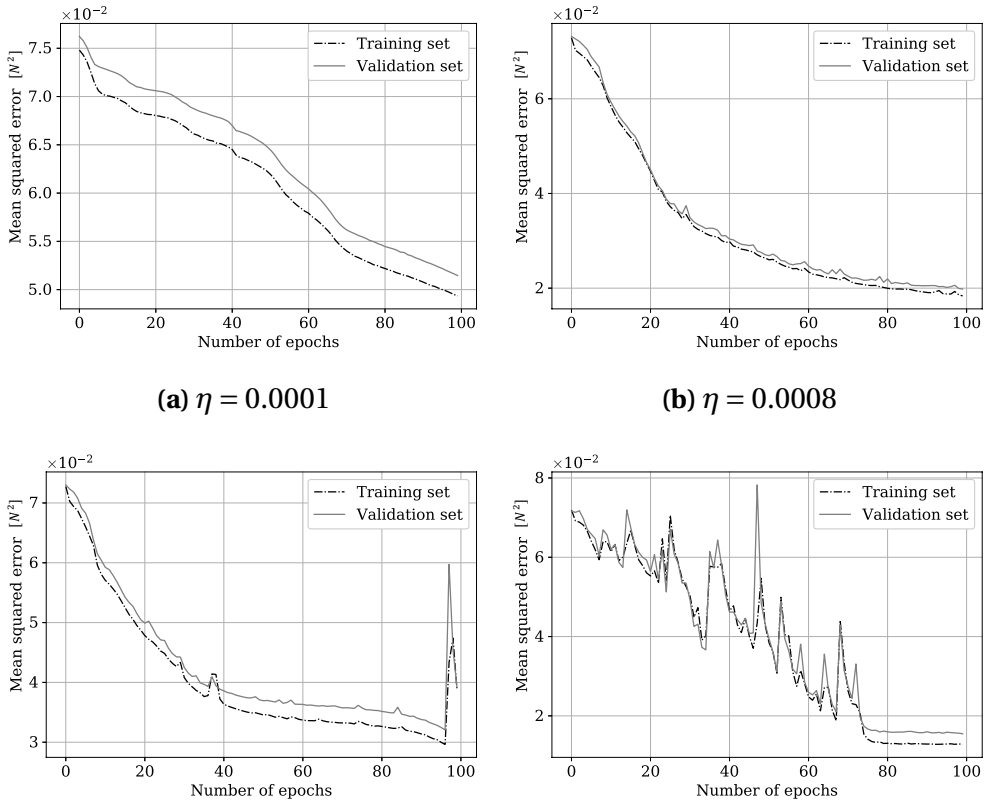
**(c)** nodes = 64 **(d)** nodes = 128

**Figure 5.9:** Learning curves of training and validation sets for various numbers of hidden nodes. A low number of hidden nodes produces smooth convergence towards low MSE scores. However, large numbers of hidden nodes makes the network more complex, causing spikes in the learning curves, particularly for 64 and 128 hidden nodes.




**(a)** **(b)**

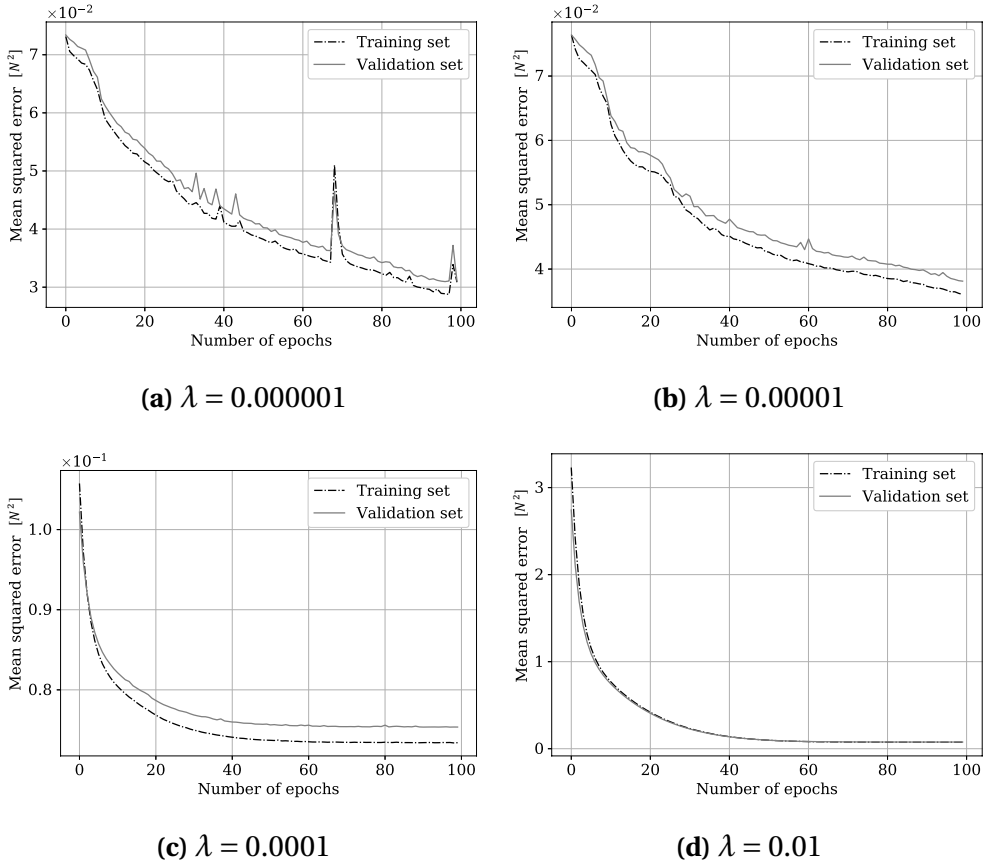
**Figure 5.10:** Effects of tuning hyperparameters. Panel (a) shows the effect of the increasing regularisation parameter on the MSE score. Panel (b) shows the effect of the increasing learning rate.




**(a)** $\eta = 0.0001$ **(b)** $\eta = 0.0008$

**(c)** $\eta = 0.001$ **(d)** $\eta = 0.01$

**Figure 5.11:** Learning curves of training and validation sets for various values of the learning rate, $\eta$. Learning rates between 0.0008 and 0.001 produces smooth convergence towards low MSE scores. The spikes in panel (d) indicate that the learning rate is too high, causing numerical instability. However, this learning rate enables the most optimal MSE score.




**(a)** $\lambda = 0.000001$ **(b)** $\lambda = 0.00001$

**(c)** $\lambda = 0.0001$ **(d)** $\lambda = 0.01$

**Figure 5.12:** Learning curves of training and validation sets for various values of the regularisation parameter, $\lambda$. Low values of the regularisation parameter provide the best MSE scores.

# 6 | Conclusion

In this chapter, a short summary of all findings and reflections accumulated throughout this thesis, is presented. The outlook for future work within the field of NVH concludes this chapter.

The model comparison was performed through analysis of accuracy scores for two different types of data signals: single featured and multifeatured, for which various input-output ratios were compared. The best performing architecture was the LSTM RNN with two layers, which produced an accuracy score of 91.01% for prediction of the multifeatured sinusoidal data signal of length 2.5 ms. Similarly, the single featured input gave an accuracy score of 90.68%. Yet, the network struggled to forecast the reference signal; an accuracy score of only 61.24% was obtained with the multifeatured input, whereas the single featured input produced 58.91%.

The results in figure 5.1 indicate that accuracy scores are, on average, directly proportional to input lengths, and inversely proportional to output lengths. Furthermore, results show that the prediction accuracy is higher for time series of the sinusoidal data, compared to the reference data. Hence, the sinusoidal data signals are less complex, easier to learn, and more reliable. Moreover, the multifeatured input signals generally provide better results, particularly in the case of sinusoidal data signals. This gives reason to believe that the smart pad technology could prove itself useful in the research of NVH. Though, the hyperparameter search revealed the importance of tuning the these parameters and the impact they have on the learning curves. Thus, hyperparameters should be prioritised during tuning of the algorithm.

The implementation of the network in a real application on board a vehicle is limited by the evaluation time for brake signal predictions. Evaluation times have to be sufficiently short, which is primarily determined by both hardware and software. Although the GRU RNN encoder-decoder performed slightly better in terms of evaluation times $t_{test}$, compared to its LSTM counterpart; the accuracy scores were still relatively low. However, the LSTM RNN encoder-decoder produced significantly higher accuracy scores, particularly in terms of MAE scores. If an optimal set of hyperparameters could be found for the GRU model, then the network could likely reach, or even surpass the performance of the LSTM model. Likewise, if the LSTM model were to be implemented on a more powerful computer system, then the evaluation times could probably improve beyond those of the GRU model.

Since the body of literature in regard to RRNs for brake signal predictions as part of brake squeal mitigation is fairly limited, it is difficult to compare the results obtained in this thesis with any expected results. Instead, the aim of this thesis was to determine whether RNNs are suitable for predictions of different data signals with various

characteristics (single featured, multifeatured, "quiet" and sinusoidal). The conclusion that could be drawn from this experiment is that RNNs hold great promise for NVH research, and that RNNs can be employed to to learn vibrational responses in the brake system. However, RNNs should be researched and analysed further.

## 6.1 FUTURE WORK

In the future, research could be extended to studies of the potential of combining classifiers with time series forecasting networks, and whether these are applicable on different types of brake signals, simultaneously. Thus, the study should examine whether the network is capable of classifying the type of brake signal, e.g., sinusoidal or "quiet", and in combination with a time series forecasting network, predicting future brake signal patterns, effectively mitigating brake squeal. Furthermore, investigation of deeper architectures are encouraged for greater prediction accuracy.

The research for using multifeatured data for prediction of brake signals showed promising results. However, implementation of other features, such as temperature and humidity, should be considered as another course of the research in the future.

Moreover, implementation of the model was done mainly on the CPU, since a relatively small data set was used. However, future projects should focus on implementation of time series forecasting models on the GPU for rapid manipulations of large data sets with higher time complexities.

# BIBLIOGRAPHY


[1] R. C. Dante, *Handbook of Friction Materials and their Applications*. Boston: Woodhead Publishing, 2016.

[2] A. Day, *Braking of Road Vehicles*. Oxford: Butterworth-Heinemann, 2014.

[3] M. Stender, M. Tiedemann, D. Spieler, and S. Oberst, "Deep learning for brake squeal: vibration detection, characterization and prediction," Hamburg, Germany, Tech. Rep., 2020.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[5] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. ACL, 2014, pp. 1724–1734.

[6] *Disc and Drum Brake Dynamometer Squeal Noise Test Procedure*, apr 2013. [Online]. Available: https://doi.org/10.4271/J2521_201304

[7] I. Inc., "Innovative itt smart pad® brake pad supported by "por fesr piemonte 2014-2020 fund"," *ITT Blog*, 2019, https://www.itt.com/newsroom/itt-blog.

[8] M. North, "A survey of published work on vibration in braking systems," *M.I.R.A. Bulletin*, no. 4, pp. 8–12, 1969.

[9] R. Spurr, "A theory of brake squeal," *Proceedings of the Institution of Mechanical Engineers: Automobile Division*, vol. 15, no. 1, pp. 33—52, 1961.

[10] N. Hoffmann, M. Fischer, R. Allgaier, and L. Gaul, "A minimal model for studying properties of the mode-coupling type instability in friction induced oscillations," *Mechanics Research Communications*, vol. 29, no. 4, pp. 197–205, 2002.

[11] J. Brownlee, *Introduction to time series forecasting with python*, 1st ed. Jason Brownlee, 2019, http://machinelearningmastery.com.

[12] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, "Financial time series forecasting with deep learning," Ankara, Turkey, Tech. Rep., 2019.

[13] I. N. da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, *Artificial Neural Networks*. Switzerland: Springer International Publishing, 2017.

[14] W. S. McCulloch and W. H. Pitts, "A logical calculus of the ideas immanent in ner-

vous activity," *The Bulletin in Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[15] M. I. Jordan, *Serial order: A parallel distributed processing approach*. San Diego: University of California, 5 1986.

[16] G. Dorffner, "Neural networks for time series processing," *Neural Network World*, vol. 6, pp. 447–468, 1996.

[17] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 10 1990.

[18] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 10, pp. 157–166, 3 1994.

[19] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 4 1998.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory." *Neural computation*, vol. 78, no. 10, pp. 1735–1780, 10 1997.

[21] F. A. Gers, J. A. Schmidhuber, and F. A. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 10 2000.

[22] J. Brownlee, *Long Short-Term Memory Networks With Python*, 1st ed. Jason Brownlee, 2019, http://machinelearningmastery.com.

[23] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[24] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *NIPS'14: Proceedings of the 27th International Conference on Neural Information Processing Systems*, vol. 2, p. 3104–3112, 2014.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[27] I.A.Basheer and M.Hajmeer, "Artificial neural networks: fundamentals, computing, design, and application," *Journal of Microbiological Methods*, vol. 43, pp. 3–31, 2000.

[28] N. Kehtarnavaz, *Digital Signal Processing System Design*. Academic Press, 2008, http://machinelearningmastery.com.

[29] G. Chevalier, "Signal forecasting with a sequence-to-sequence rnn model in tensorflow," https://github.com/guillaume-chevalier/seq2seq-signal-prediction, Accessed: 26-05-2020.

[30] J. Lambert, "Stacked rnns for encoder-decoder networks : Accurate machine understanding of images," 2016.

[31] S. Kolassa and W. Schütz, "Advantages of the mad/mean ratio over the mape," *Foresight: The International Journal of Applied Forecasting*, no. 6, pp. 40–43, 2007. [Online]. Available: https://EconPapers.repec.org/RePEc:for:ijafaa:y:2007:i:6:p:40-43

[32] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. New York: John Wiley and sons, 1949.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, no. 323, pp. 533–536, 10 1986.

[34] T. Hamabe, I. Yamazaki, K. Yamada, H. Matsui, S. Nakagawa, and M. Kawamura, "Study of a method for reducing drum brake squeal," *SAE International*, vol. 108, no. 6, pp. 523–529, 1999.

# A | Historical Review

This chapter provides some additional historical review about the foundation of the ANN that could be interesting for the reader.

## A.1 Development of artificial neural networks

The foundation of ANN was first introduced in 1943 by McCulloch and Pitts [14] in an effort to describe brain activity and the different processes occurring in the neuron by means of mathematical models. The result came to be known as the McCulloch-Pitts artificial neuron or the linear threshold function [14].

In 1949, Hebb [32] introduced a theory that described a training method for neural networks, which came to be known as Hebb's rule. The method explained the pattern learning process in an ANN, i.e., the weights within a given ANN are distributed such that the shared connection between two simultaneously activated neurons is reinforced [32].

In 1960, Widrow and Hoff developed an alternative ANN architecture named the adaptive linear element (ADALINE). The model was capable of classifying more complex patterns and perform linear regression. Despite the simple nature of the new architecture, the algorithm improved the efficiency of conventional neural networks considerably. The specific features that set ADALINE apart was the delta learning rule, which later was implemented in the multilayer perceptron as well as in signal processing [13].

Although, the advancements of the ANN, perceptron and ADALINE were impressive at the time, these were still linear classifiers limited to recognition of linearly separable classes. The single neuron network did not have a negative feedback structure for the values produced by the output layer; hence, forward propagation only occurred in one direction. This lead to the development of the backpropagation algorithm, allowing a new type of learning process for deeper networks to emerge.

In 1986, Rumelhart et al. [33] pioneered another learning strategy for neural networks. Their method applied the chain rule to compute the gradient of the error function and where each term is evaluated in reverse order, i.e., from the output to the input layer, hence the name backpropagation.

Around the time backpropagation was introduced, Jordan [15] developed a generalisation of deep learning methods for sequential characters – the recurrent neural network (RNN). The RNN differs from previous ANN architectures in that it has a dynamic memory component for its input sequences; instead of processing each data point separately, the output from a neuron at a previous time step affects the calculations in the

current neuron.

Due to the dynamic nature of the RNN architecture, the basic backpropagation algorithm required minor modifications to the updating scheme for it to function properly. Thus, in 1990 Werbos [17] developed an algorithm that unfolds the network back in time, the so called backpropagation through time (BPTT). Werbos suggested treating the RNN as a basic feed forward network, enabling the application of backpropagation between each time-dependent layer.

# B | ADDITIONAL THEORY

The purpose of this chapter is to give the reader additional tools for a deeper understanding of the theory presented in this thesis. The benchmark model that represents the brake system is explained as well as the definition of the white noise.

## B.1 TWO DEGREE OF FREEDOM MODEL

The frequency range of brake noise seems to be influenced by the dimensions of the brake pad and the inherent vibrational mode of the disc; if the brake pad is short in comparison to the natural wavelength of the disc, then the entire pad assembly can be modelled as a rigid beam with two degrees of freedom, presented in figure B.1. Thus, the overall shape of a disc brake enables vibrations to travel between two closely spaced modes, generating a type of instability termed binary flutter [2].

In order to study the various mechanisms of the binary flutter, a minimal single mass model with two degrees of freedom is introduced as in figure B.1. The conveyor belt

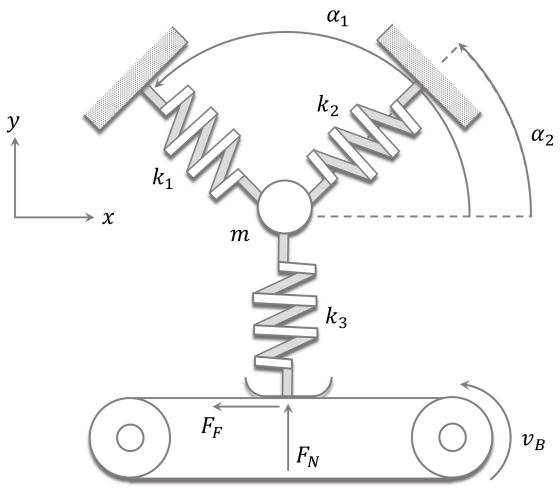



**Figure B.1:** Single mass model with two degrees of freedom representing the brake system. The conveyor belt demonstrating the brake disc, is operating at constant velocity $v_B$ while a constant force $F_N$ is applied perpendicularly to the point mass $m$, illustrating the brake pad. The mass is secured with two linear springs $k_1$ and $k_2$ at the angles $\alpha_1$ respective $\alpha_2$, whereas $k_3$ acts as a model for the perpendicular stiffness between the mass and the surface of the belt. The law of friction is applied, where $F_F$ is the frictional force with a constant friction.

demonstrating the brake disc, is operating at constant velocity $v_B$ while a constant force $F_N$ is applied perpendicularly to the point mass $m$, illustrating the brake pad.

The mass is secured with two linear springs $k_1$ and $k_2$ at the angles $\alpha_1$ respective $\alpha_2$, whereas $k_3$ acts as a model for the perpendicular stiffness between the mass and the surface of the belt. Note that Coulomb's law of friction is applied, where $F_F$ is the frictional force with a constant friction [34].

The single mass model with two degree of freedom representing the brake system, could be considered as a generalisation of the stick-slip phenomenon with the added complexity of displacements normal to the friction surface. Hence, the equations of motion relating to figure B.1 are represented as kinematic constraint model as follows [10]:

$$\begin{bmatrix} m & 0 \\ 0 & m \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} + \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} F_F \\ F_N \end{bmatrix},$$

or more concisely,

$$\boldsymbol{M}\ddot{\boldsymbol{X}} + \boldsymbol{K}\boldsymbol{X} = \boldsymbol{F},$$

where $x$, $y$ are the degrees of freedom, $\boldsymbol{M}$ and $\boldsymbol{K}$ represent mass and stiffness, respectively, and $\boldsymbol{F}$ is the force acting on the system. Note that the coefficients of the stiffness matrix $\boldsymbol{K}$ are given by [10]:

$$\begin{aligned} k_{11} &= k_1 \cos^2 \alpha_1 + k_2 \cos^2 \alpha_2, \\ k_{12} &= k_{21} = k_1 \sin \alpha_1 \cos \alpha_1 + k_2 \sin \alpha_2 \cos \alpha_2, \\ k_{22} &= k_1 \sin^2 \alpha_1 + k_2 \sin^2 \alpha_2. \end{aligned}$$

This type of kinematic constraint model illustrates the underlying properties of mode coupling, i.e, the frictional energy from the surface of the conveyor belt in figure B.1 is converted into vibrational energy in other parts of the system, effectively coupling the friction force with vibrational modes. This transfer of energy between different vibrational modes is crucial in the pursuit of understanding binary flutter and ultimately brake noise [10].

## B.2 WHITE NOISE

White noise is defined by the time series of independent and identically distributed values, with zero mean $\mu$ and unit variance $\sigma^2$, as $X \in \mathcal{N}(\mu = 0, \sigma^2 = 1)$. Despite the unpredictability of white noise, the analysed time dependent signal is expected to contain some random component on top of the signal generated by underlying process, according to [11]:

$$\boldsymbol{x}(t_i) = \tilde{\boldsymbol{x}}(t_i) + X(t_i), \tag{B.2}$$

where $\tilde{\boldsymbol{x}}(t_i)$ is the filtered signal with additional noise $X(t_i)$ generate signal $\boldsymbol{x}(t_i)$ for the analysis. After predictions being made, the forecast errors should be white noise, which means that the model succeeded to harness all information in the sequence and that no further improvements to the model can be done [11].