Using Long-Short-Term-Memory Recurrent Neural Networks to Predict Aviation Engine Vibrations

by

AbdElRahman Ahmed ElSaid
Bachelor of Science, Cairo University, 2007

A thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota
December
2016

This thesis, submitted by AbdElRahman Ahmed ElSaid in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work as been done and is hereby approved.

Dr. Travis Desell, Chairperson

Mr. James Higgins

Dr. Wen-Chen Hu

Dr. Marina Kim

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the School of Graduate Studies at the University of North Dakota and is hereby approved.

Dr. Grant McGimpsey
Dean of the School of Graduate Studies

December 8, 2016
Date

PERMISSION

Title	Using Long-Short-Term-Memory Recurrent Neural Networks to Predict Aviation Engine Vibrations
Department	Department of Computer Science
Degree	Master of Science

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the Chairperson of the department or the dean of the School of Graduate Studies. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.

AbdElRahman Ahmed ElSaid
December 2016

Contents

List of Figures v

List of Tables vi

Acknowledgements vii

Abstract ix

Notations x

1 Introduction 1

2 Related Work 3
I Aircraft Engine Vibration . . . 3
II Time Series Prediction . . . 4
II.I Statistical Prediction Models . . . 5
II.I.1 Autoregressive (AR) Model . . . 5
II.I.2 Moving Average (MA) Model . . . 6
II.I.3 ARMA (Autoregressive Moving Average) Model . . . 7
II.I.4 ARIMA (Autoregressive Integrated Moving Average) Model . . . 7
II.I.5 SARIMA (Seasonal ARIMA) Model . . . 8
II.II Artificial Neural Network (ANN) Prediction Models . . . 8
II.II.1 RNN for Predicting Flight Parameters . . . 10
II.II.2 LSTM RNN . . . 13
II.III Hybrid Forecasting Models . . . 14
II.IV Summary . . . 15

3 Methodology 17
I Experimental Data . . . 17
I.I Data Correlation . . . 17
I.II Aerodynamics/Turbo-machinery Parameters' Selection . . . 18
II Methodology . . . 20
II.I Neural Networks Overview . . . 20
II.II LSTM RNN Forward Propagation Equations . . . 22
II.III LSTM RNN Architectures . . . 25
II.III.1 Architecture I . . . 26
II.III.2 Architecture II . . . 26
II.III.3 Architecture III . . . 26
II.IV Forward Propagation . . . 31

4 Implementation 33
I Programming Langauge . . . 33
II Data Processing . . . 33
III Machine Specifications . . . 34
III.I Optimizing Matrices for GPU Computations . . . 34
IV Training Results . . . 36

5 Results 37
I Cost Function . . . 37
II Architecture Results . . . 37
II.I Results of Architecture I . . . 48
II.II Results of Architecture II . . . 48
II.III Results of Architecture III . . . 51

6 Conclusion 52

7 Future Work 53

A Architecture I Back Propagation 54

Bibliography 66

List of Figures

1 Biological - Artificial neurons [13] . . . 10
2 Perceptron input/output [13] . . . 11
3 Simple neural network [13] . . . 11
4 Perceptron neuron Vs. Activation neuron [13] . . . 12
5 Learning Process [13] . . . 12
6 Time Series Prediction Summary [8] . . . 15

7 Simple Neural Network . . . 19
8 LSTM cell design . . . 21
9 Level 1 LSTM cell design . . . 22
10 Level 2 LSTM cell design . . . 23
11 Architecture I . . . 27
12 Architecture II . . . 28
13 Architecture III . . . 30

14 Cost function plot for the three architectures predicting vibration in 1 future sec. . . . 38
15 Cost function plot for the three architectures predicting vibration in 5 future sec. . . . 39

16 Cost function plot for the three architectures predicting vibration in 10 future sec. . . . 40
17 Cost function plot for the three architectures predicting vibration in 20 future sec. . . . 41
18 Architecture I predicting vibration for one flight. . . . 43
19 Architecture III predicting vibration for one flight. . . . 44
20 Plotted results for Architecture I for the for the three scenarios. . . . 45
21 Plotted results for Architecture II for the for the three scenarios. . . . 46
22 Plotted results for Architecture III for the for the three scenarios. . . . 47
23 Plotted results for the three architectures predicting one second in the future. . . . 49
24 Plotted results for Architecture I & III predicting one second in the future. . . . 50

List of Tables

1 Summary of advantages and challenges of classical and ANNs based time series prediction methods [48] . . . 16

2 Architectures Weights-Matrices Dimensions . . . 29
3 Architectures Weights Matrices' Total Elements . . . 29

4 Run Time (hours) . . . 34
5 Training Results . . . 36

6 Testing Process Mean Squared Error . . . 42
7 Testing Process Mean Absolute Error . . . 42

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor Dr. Travis Desell of the Department of Computer Science, School of Aerospace Studies at the University of North Dakota. My advisor was always available whenever I ran into a trouble spot or had a question about my research or writing. He consistently directed this paper to be my own work, but steered and advised me in the right direction whenever he thought I needed it. Ultimately, this effort could not see the light if not for his strong support, kind patience, and teaching.

I would also like to thank my respectable advisory committee members for their valuable comments and support: Dr. Wen-Chen Hu and Dr. Marina Kim of the Department of Computer Science, School of Aerospace Studies at the University of North Dakota and Mr. James Higgins of the Aviation Department, School of Aerospace Studies at the University of North Dakota. Special thanks to Dr. Hu for the influence he had on me through his classes in which I enjoyed his teaching very much.

In addition, I would like to thank Dr. Yen Lee Loh of the Physics Department, School of Art and Science at the University of North Dakota for his kind effort revising the work done in deriving the neural network's backpropagation gradients. I can not also forget to thank Ms. Run Li of the Chemistry Department, School of Art and Science at the University of North Dakota for her support and effort on this side of the project.

Last but not least, I want to express my sincere appreciation to my life-partner for the endless support and tolerance I enjoyed in her company. Thanks are extended to my family, especially my dearest brother who never failed me. This work would not have even started if not for him.

DEDICATION

To my parents, how can I possibly thank you enough for the gift of life!
To my partner, thank you for your care and for making life easier.
To my brother, thank you for always being there.
To Wesam Elshamy, thank you my friend.

ABSTRACT

This thesis examines building viable Recurrent Neural Networks (RNN) using Long Short Term Memory (LSTM) neurons to predict aircraft engine vibrations. The different networks are trained on a large database of flight data records obtained from an airline containing flights that suffered from excessive vibration. RNNs can provide a more generalizable and robust method for prediction over analytical calculations of engine vibration, as analytical calculations must be solved iteratively based on specific empirical engine parameters, and this database contains multiple types of engines. Further, LSTM RNNs provide a "memory" of the contribution of previous time series data which can further improve predictions of future vibration values. LSTM RNNs were used over traditional RNNs, as those suffer from vanishing/exploding gradients when trained with back propagation. The study managed to predict vibration values for 1, 5, 10, and 20 seconds in the future, with 2.84% 3.3%, 5.51% and 10.19% mean absolute error, respectively. These neural networks provide a promising means for the future development of warning systems so that suitable actions can be taken before the occurrence of excess vibration to avoid unfavorable situations during flight.

NOTATIONS

NOMENCLATURE

ANN(s) Artificiale Neural Network(s)

AR Autoregressive Model

ARIMA Autoregressive Integrated Moving Model

ARMA Autoregressive Moving Model

ART Architecture

CPU Central Processing Unit

ESN Echo State Networks

FDR Flight Data Recorder

FFNN FeedForward Neural Network

GPU Graphics Processing Unit

LAH List Appreviations Here

LSTM Long Short Term Memory

MA Moving Average Model

MAE Mean Absolute Error

MSE Mean Squared Error

O/P Output

RNN Recurrent Neural Networks

SARIMA Seasonal ARIMA

SVM Support Vector Machines

VAM Vector Autoregressive Model

vib Vibration

CHAPTER 1

INTRODUCTION

Aircraft Engine vibration is a critical aspect of the aviation industry, and accurate predictions of excessive engine vibration have the potential to save time, effort, money as well as human lives in the aviation industry. An aircraft engine, as turbo-machinery, should normally vibrate as it has many dynamic parts. However, it is not supposed to exceed resonance limits so not to destroy the engine [1].

A. V. Srinivasan [1] describes vibrations generated from engine blades' fluttering. Engine blades are the engine rotating components that have the largest dimensions among other components. When rotating at high speeds, they will withstand high centrifugal forces that would logically give the highest contribution to engine vibrations.

Engine vibrations are not that simple to calculate or predict analytically because of the fact that various parameters contribute to their occurrence. This fact is always a problem for aviation performance monitors, especially as engines vary in design, size, operation conditions, service life span, the aircraft they are mounted on, and many other parameters. Most of these parameters' contributions can be translated in some key parameters measured and recorded on the flight data recorder. Nonetheless, vibrations are likely to be a result of a mixture of these contributions, making it very hard to predict the real cause behind the excess in vibrations.

This thesis presents a means to make these predictions viable in the aviation industry within a reasonable time window. The problem is approached using LSTM RNNs, which have seen widespread recent use with strong results in image [2], speech [3] and, language prediction [4]. LTSM RNNs were chosen for this work in particular due to their generalizability and predictive power due to having a memory for the contribution of the previous time series data to predict the future values of vibration. This study provides another dimension for the use of this promising type of recurrent neural network.

Chapter 2 presents related work in the fields of LSTM RNN's and vibration detection. Chapter 3 covers the approaches taken to design out neural network architectures. Performance results and limitations of the algorithms are described in Chapter 5. Finally, Chapter 6 concludes with future work and a discussion of the next steps to improving the results, enhancing the algorithms, and proposing solutions to utilize advanced computing alternatives. As an appendix, an effort for driving back propagation process's gradients for Architecture I is introduced in Appendix A.

CHAPTER 2

RELATED WORK

Aircraft Engine Vibration

According to A. V. Srinivasan [1]: *"The most common types of vibration problems that concern the designer of jet engines include (a) resonant vibration occurring at an integral order, i.e. multiple of rotation speed, and (b) flutter, an aeroelastic instability occurring generally as a nonintegral order vibration, having the potential to escalate, unless checked by any means available to the operator, into larger and larger stresses resulting in serious damage to the machine. The associated failures of engine blades are referred to as high cycle fatigue failures"*. The *means available to the operator* in practical aviation operations are mainly: *i)* maintenance engine checks scheduled in maintenance programs based on engine reliability observations, and *ii)* engine vibration monitoring for forecasting the excess vibration occurrence based on statistical and analytical methods which consider empirical factors of safety.

Some effort has been done using neural networks to classify engine abnormalities without doing analytical computation, *e.g.*, Alexandre Nairac *et al.* [5] worked on this aspect to detect abnormalities in engine vibrations based on recorded data. To achieve that, the paper used two modules. One of the modules uses the overall shape of the vibration curve to detects unusual vibration signature. The second one reports sudden unexpected transition in the signature curves. Their approach to detect defects is not to introduce examples

of faulty engines to the neural network. Rather, only examples of healthy engines are introduced to the neural networks in the training phase. This approach was taken to overcome the lack of existence of adequate faulty engine data, enough for training. In this context, the paper introduces the term 'normality' to describe the behavior of normal engines and 'abnormality' to describe the behavior of faulty engines. Using statistical models, the faulty engines detection would be described as 'novelty' detection based on the deviation from the data distribution. The best results the paper achieved was the prediction of faulty engines with 84% successful classifications.

David A. Clifton *et al.* [6] presented work for predicting abnormalities in engine vibration based on statistical analysis of vibration signatures. The paper presents two modes of prediction. One is ground-based (off-line), where prediction is done by run-by-run analysis to predict abnormalities based on previous engine runs. The success in this approach was predicting abnormalities two flights ahead. The other mode is a flight based-mode (online) in which detection is done either by sending reduced data to the ground-base or onboard the aircraft. The paper mentions that they could successfully predict 2.5 hours in the future. However, this prediction is done after half an hour of flight data collection, which might be a critical time as well, as excess vibration may occur during this data collection time. The paper did not mention how much data was required to have a sound prediction.

Time Series Prediction

From a statistical point of view, the main goal of prediction is to provide vital information for decision makers, economists, planners optimizers, industrialists and critical systems operators. There are two sides for prediction: the qualitative

side and the quantitative side. The qualitative side utilizes methods known as the judgmental or subjective prediction methods which covers methods relaying on intuition, judgement or opinions of some kind of a referee as customers, consumers, experts and/or supporting information. Qualitative methods are considered in cases when past data is not available. On the other hand quantitative methods include univariate and multivariate methods. Though, for many study cases related to different scientific and real life problems, the time series data are available on several dependent variables, in such cases multivariate prediction methods are used [7]. This section will provide a brief discussion about some of the available prediction models.

Statistical Prediction Models

Some of the common time series prediction statistical models are discussed here. Linear statistical methods have been influencing prediction efforts for a long time. These methods include the autoregressive (AR) model, the moving average (MA) model, and hybrid models that derive from them such as ARMA (autoregressive moving average), ARIMA (autoregressive integrated moving average), and SARIMA (seasonal ARIMA) [8].

II.I.1 Autoregressive (AR) Model

In this model, there is a linear dependence between the output variable and its own value in previous time steps and a certain error. The model can formally be defined as: a process $\{z_t\}$ is an autoregressive process of order n at time t denoted AR(n) if z_t can be formally represented by:

$$z_t = \alpha_1 z_{t-1} + \alpha_2 z_{t-2} + \ldots + \alpha_n z_{t-n} + \varepsilon_t \tag{1}$$

where ε_t is the error with mean zero and fixed finite variance σ_Z.

A vector autoregressive (VAR) model can be used when there is more than one dependent variables. Formally this can be represented by:

$$z_t = \sum_{l=1}^{n} A_l z_{t-l} + \varepsilon_t \tag{2}$$

where z_t, z_{t-l}, and ε_t are a vector with number of elements equal to the number of dependent variables. A_l is a square matrix of dimensions equal to the number of independent variables squared [7].

II.I.2 Moving Average (MA) Model

If ε_t is random with mean zero and fixed finite variance σ_Z, then process $\{z_t\}$ is a moving average process of order k denoted MA(k) and represented by [7, 9]:

$$z_t = \varepsilon_t + \beta_1 \varepsilon_{t-1} + \beta_2 \varepsilon_{t-2} + \ldots + \beta_k \varepsilon_{t-k} \tag{3}$$

MA is also the process where the next sample depends on the weighted sum of the past or present inputs of an exogenous time series $\{y_t\}$ of N-dimensions formally described by:

$$z_t = \beta_0 y_t + \beta_1 y_{t-1} + \ldots + \beta_k y_{t-k} + \varepsilon_t \tag{4}$$

For more than one independent variable, MA model is represented by:

$$z_t = \sum_{j=0}^{k} B_j y_{t-j} + \varepsilon_t \tag{5}$$

where y_t is an exogenous N-dimension time series and B_k are M-by-N matrices of parameters and M is the number of independent observed variables.

II.I.3 ARMA (Autoregressive Moving Average) Model

This model is widely used because it benefits from the advantages of both auto-regressive AR(n) and the moving average MA(k) models. An ARMA(n, k) model of order (n, k) is formally defined by:

$$z_t = \alpha_1 z_{t-1} + \ldots + \alpha_n z_{t-n} + \varepsilon_t + \beta_1 \varepsilon_{t-1} + \ldots + \beta_k \varepsilon_{t-k} \tag{6}$$

where z_t is the original series and ε_t is a series of random errors which are assumed to followed the normal probability distribution. For more than one independent variable, ARMA model is called vector auto-regressive moving average (VARMA) which is represented by:

$$z_t = \sum_{l=1}^{n} A_l z_{t-l} + \varepsilon_t + \sum_{j=0}^{k} B_j y_{t-j} + \varepsilon_t \tag{7}$$

II.I.4 ARIMA (Autoregressive Integrated Moving Average) Model

AR, MA, and ARMA are used in stationary time series analysis [10]. A time series is defined as stationary when the mean of the series and the covariance among its variables do not change over time and do not follow any trend [11]. In real life, the majority of the time series is non-stationary. To fit stationary models, it is absolutely necessary to remove the non-stationary sources of variation [7]. One of the proposed solutions for this was presented by Box and Jenkins [9] when their work introduced the ARIMA model which uses differencing process to transform the non-stationary data into a stationary one [7, 11]. The general form of the ARIMA (n, h, k) process is formally described as:

$$z'_t = \nabla^k z_t = \alpha_1 z'_{t-1} + \ldots + \alpha_n z'_{t-n} + \varepsilon_t + \beta_1 \varepsilon_{t-1} + \ldots + \beta_k \varepsilon_{t-k} \tag{8}$$

II.I.5 SARIMA (Seasonal ARIMA) Model

SARIMA is an extension to the ARIMA model [7]. It is used when the data is presented with a periodic characteristic that must be known in advance [9].

Artificial Neural Network (ANN) Prediction Models

Artificial neural networks are part of the machine learning models used in time series prediction. Neural networks are an imitation for human neural cells as shown in Figure 1, where neurons (nodes/perceptrons) takes input data and perform simple operations then selectively pass the results onto other neurons [12]. In the very beginning, McCulloch and Pitts developed the first artificial neuron model in 1943 [8]. However, artificial neural networks emerged in the 1950's with a simple simulation to the biological neuron and the building cells were called perceptrons. The perceptrons took several binary inputs and produced single binary output. To compute the output, real numbers called weights are used expressing the importance of the respective inputs to the output as shown in Figure 2. The perceptron's output, 0 or 1, is determined by whether the weighted sum

$$\sum_{j=1} w_j \cdot x_j$$

is less than or greater than some threshold value. Layers of perceptrons are used where simple decision are made in early layers and then more complex and abstract decisions could be taken in later layers as shown in Figure 3. Thus,

perceptrons can be looked at as a method for weighing evidence to make decisions. To use neural networks in a learning algorithm it should be trained on a set of data by forward propagating inputs to let it try to compute the target value. Then feedback the error some how to modify the weights. This is done in a loop until the neural network can predict the target values by its own. However, The problem with perceptrons is that a small change in one of the weights can completely flip the output (output is binary). Thus, a new type of neurons are introduced: "Activation Neurons". The advantage of this type of neurons is that a small changes in their weights cause only a small change in their output. The difference between a perceptron neuron and an activation neuron is shown in Figure 4. There are several algorithms which can be used to set the weights of a neural network to represent a desired function. One of the more popular methods is by supervised learning. This involves training the neural network using a subset of correct outputs for specific inputs. The neural network then extrapolates means to classify (predict) vectors of input signals that it had not yet been presented with. An error function (cost/loss), E, is used to measure the discrepancy between the output of the neural network and the targeted value. This function has a global minima at a point at which the target value and network output are nearly equal. Different sets of input and known output are chosen for each training iteration. This causes the state of the weights of the network to converge along the average of each error function associated with each desired output. This process is shown in Figure 5 [13].

Over time, many variations and type of ANNs were developed. Differences between these variations and types could be in the architectures, the activation function, the learning algorithms, etc, depending on the proposed problem. One of those ANNs is the feed-forward neural networks (FFNNs) or multilayer perceptron's (MLP) Figure 3 presents the basic neural networks architecture [14]. In addition, there are also other more advanced ANNs architectures as recurrent

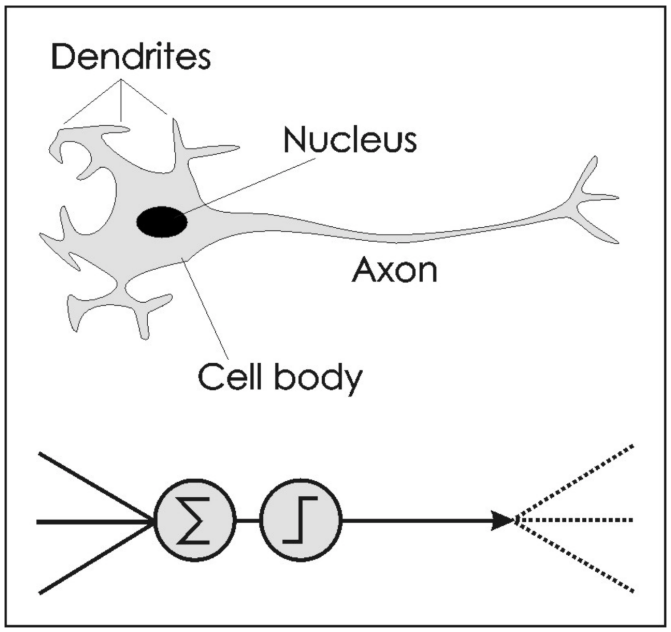


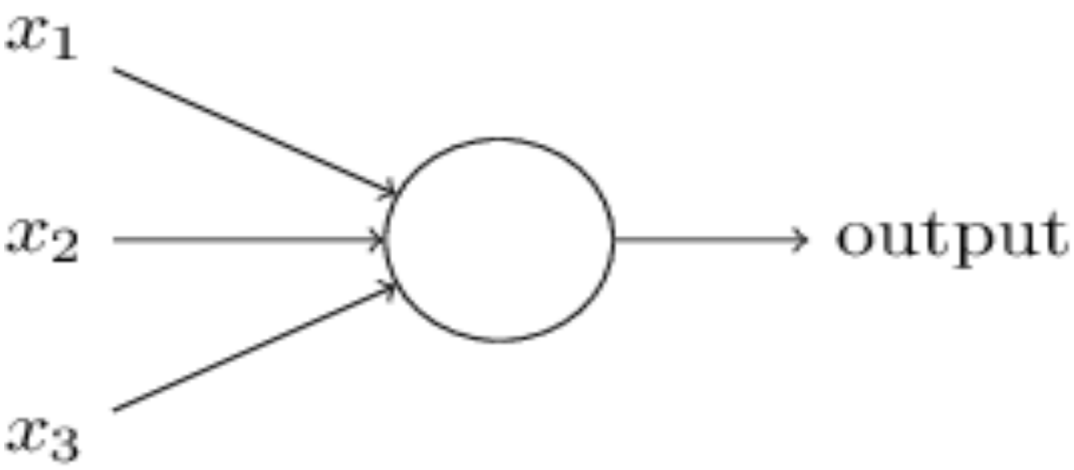
Figure 1: Biological - Artificial neurons [13]

neural networks (RNNs) that include simple recurrent neural networks as well as more complicated models as long short term memory network (LSTM) [14–16] and echo state networks (ESN) [17]. Other ANNs architectures are radial basis function (RBF) [18], cascading neural networks [19]. There are however other machine learning techniques such as support vector machines (SVM) [20].

II.II.1 RNN for Predicting Flight Parameters

Having an advantage over standard FFNNs, RNNs can deal with sequential input data, using their internal memory to process sequences of inputs and deep learn from them. This is done by feedback connections or by looping between neurons and thus making them capable of predicting more complex data [21].

An inspiring work was done on predicting flight parameters [22, 23]. The research used recurrent neural networks and applied an ant-colony optimization (ACO) algorithm [24–26], an optimization technique used in the beginning on



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Figure 2: Perceptron input/output [13]

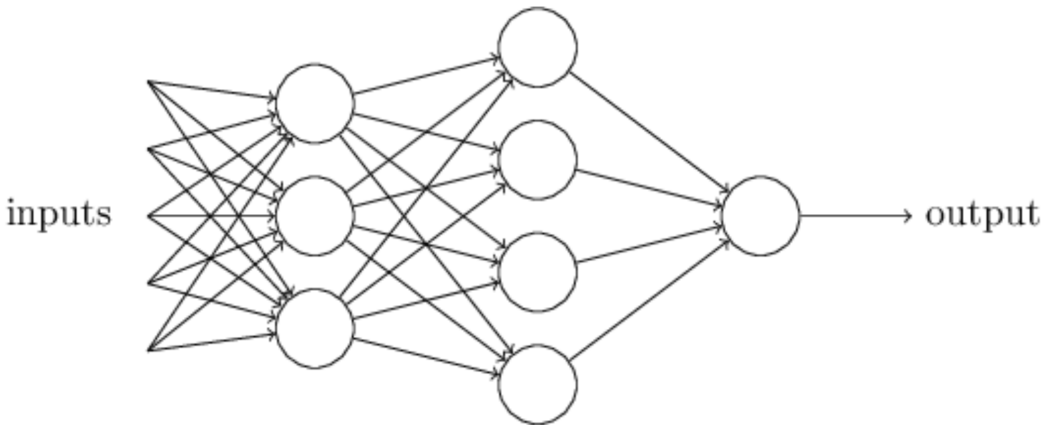


Figure 3: Simple neural network [13]

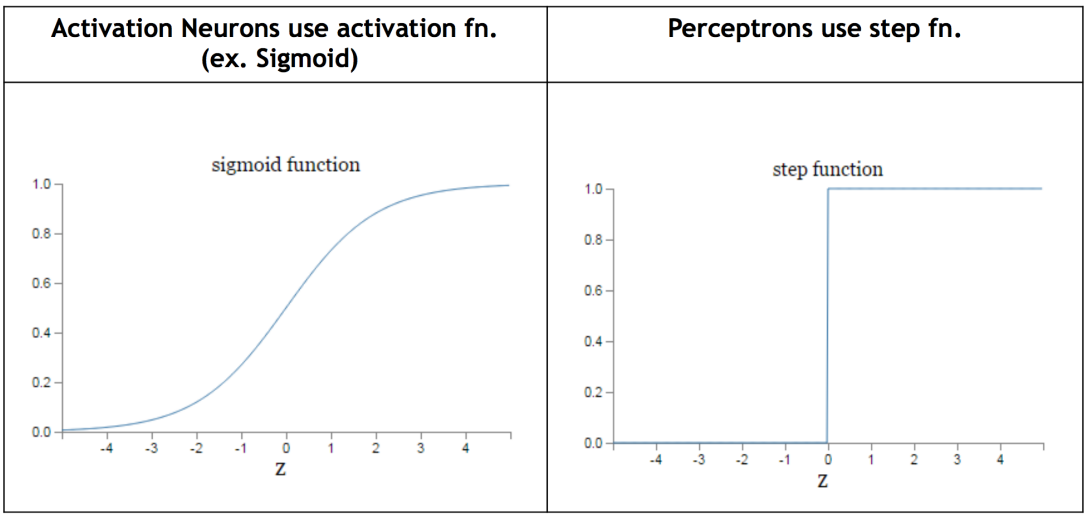


Figure 4: Perceptron neuron Vs. Activation neuron [13]

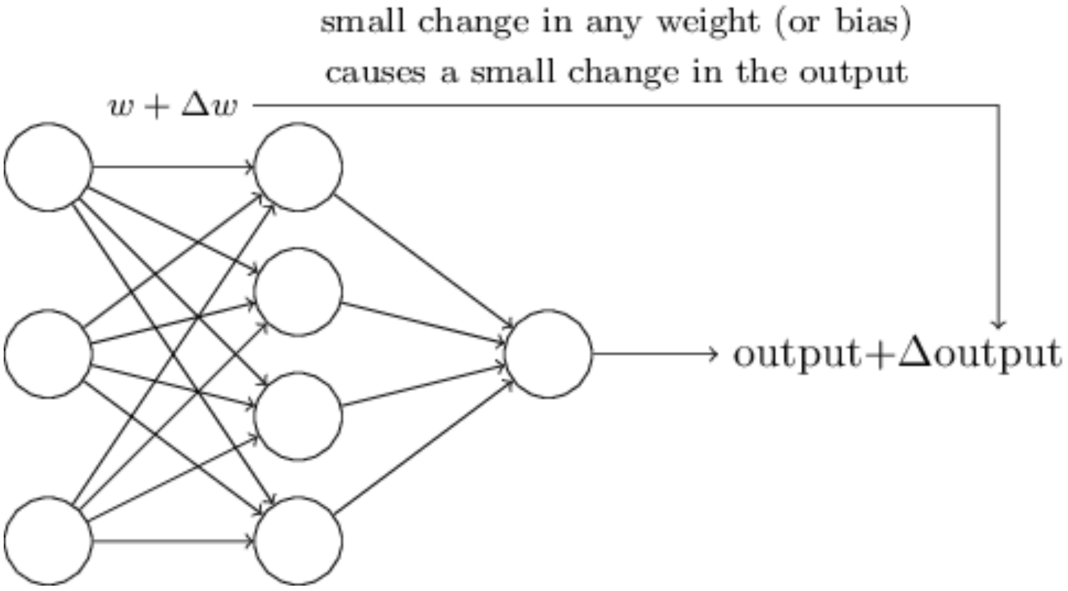


Figure 5: Learning Process [13]

discrete problems, mainly on the Traveling Salesman Problem [27]. Later it was used in continuous optimization problems [28–33], including training neural networks [34–37] T. Desell *et al.* used the ACO to give the neural network the ability to evolve to a near optimum structure. The neural network predicted airspeed, altitude and, pitch with a 63%, 97% and 120% improvement respectively, over previously best published results [38].

II.II.2 LSTM RNN

LSTM RNNs were first introduced by S. Hochrieter & J. Schmidhuber [39]. The paper introduced a solution for this problem: *" Learning to store information over extended period of time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow*". It was a solution for the exploding/vanishing gradients in backpropagtion (S. Hochrieter in 1991) to modify the weights of the network. This study paved the way for many interesting projects.

Later on, J. Schmidhuber *et al.* [16] emphasized about the forget gate in the LSTM RNNs in another publication. The paper mentioned that *"We identify a weakness of LSTM networks processing continual input streams that are not a priori segmented into subsequences with explicitly marked ends at which the network's internal state could be reset. Without resets, the state may grow indefinitely and eventually cause the network to break down. Our remedy is a novel, adaptive "forget gate" that enables an LSTM cell to learn to reset itself at appropriate times, thus releasing internal resources. We review illustrative benchmark problems on which standard LSTM outperforms other RNN algorithms. All algorithms (including LSTM) fail to solve continual versions of*

these problems. LSTM with forget gates, however, easily solves them, and in an elegant way."

However, Felix A. Gers *et al.* [40] suggest that *"LSTM RNNs does not carry over to certain simpler time series prediction tasks solvable by time window approaches"*. The paper suggests to use LSTM when *"simpler traditional approaches fails"*.

LSTM RNNs have been used with strong performance in image recognition [2], audio visual emotion recognition [3], music composition [41] and other areas.

Regarding time series prediction, for example, LSTM RNNs have been used for stock market forecasting [42] and forex market forecasting [43]. Also forecasting wind speeds [39, 44] for wind energy mills, and even predicting diagnoses for patients based on health records [45].

Hybrid Forecasting Models

After ANNs were widely used for time series predictions, a new hybrid models were introduced. Hybrid models represented a mixture of machine learning and standard statistical models. In the literature, different combination of methods have been introduced to overcome the limitation of a specific model. The idea behind these kind of models is to build a model that take advantage of the strength of different models to capture different aspects in the data. One famous example of hybrid model is the ARIMA and ANNs mixture. ARIMA itself combines three different processes including an AR function regressed on past values of the process, MA function regressed on a purely random process, and it also deals with the non-stationary linear components. On the other side, the ANN takes care of the non-linear components of the data [46, 47].

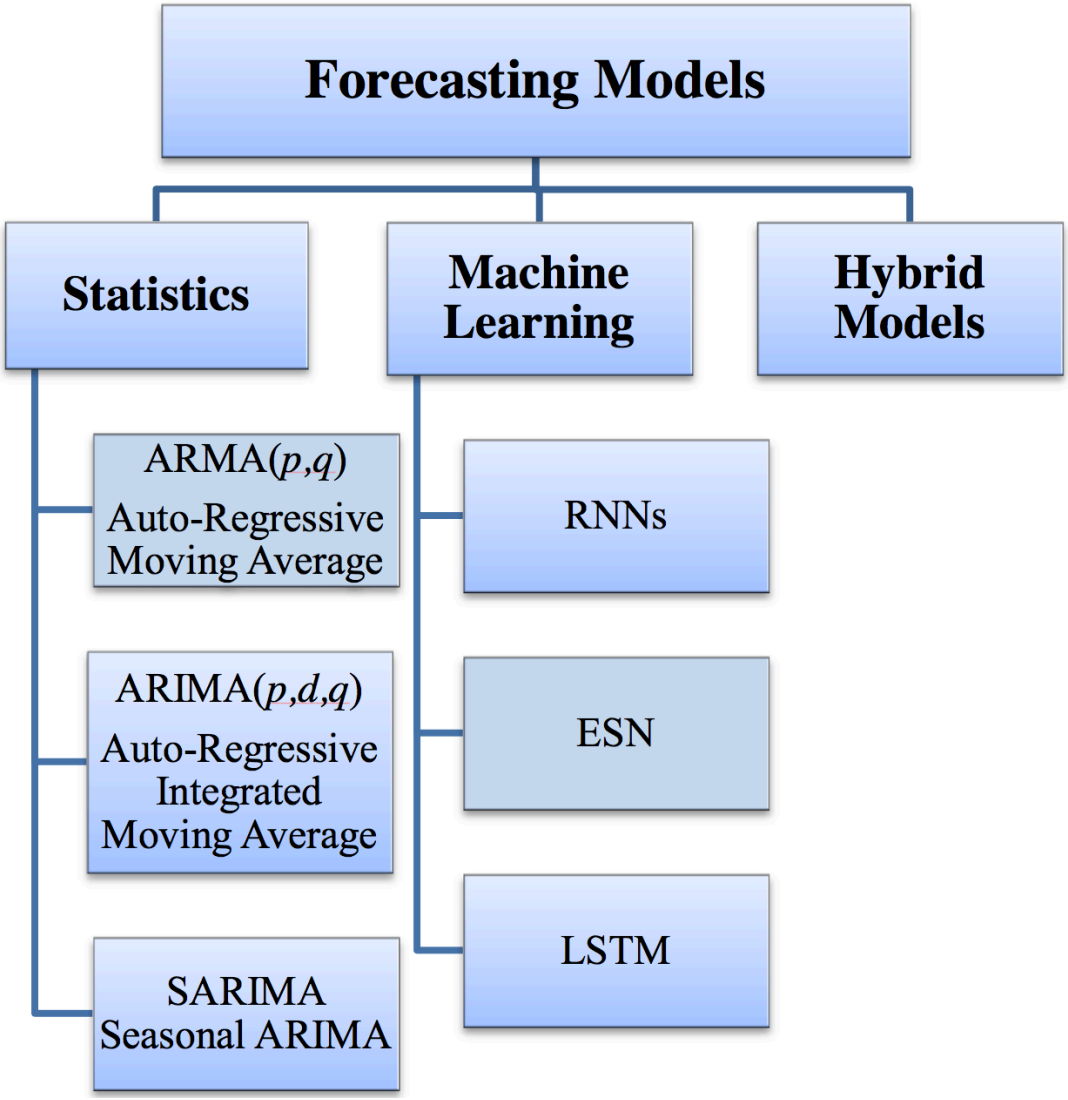


Figure 6: Time Series Prediction Summary [8]

Summary

Each of the discussed models have its own importance and strength point(s). Statistical models have provided good efficiencies for linear low order systems. On the other hand, ANNs have shown good performance with non-linear systems. Hybrid models offer the benefits of both previous systems. Figure 6 [8] shows this summary. A comparison between the standard statistical models and ANNs is presented in Table 1 [48].

Table 1: Summary of advantages and challenges of classical and ANNs based time series prediction methods [48]

Time Series Prediction Method	**Challenges**	**Error**
Standard Statistical Models	• Can be computationally efficient for low order models. • Convergence guaranteed. • Minimizes mean square error by design.	• Assumes linear, stationary processes. • Can be computationally expensive for higher order models.
ANNs	• Not model dependent. • Not dependent on linear, stationary processes. • Can be computationally efficient for feed forward process. • Capable of learning long term dependencies (LSTM). • Can detect all possible, complex nonlinear relationships between input and outputs.	• Selection of free parameters usually calculated empirically. • Not guaranteed to converge to optimal solution. • Can be computationally expensive (training process). • Neural networks have a "black box" nature. Therefore, errors within the complex network are difficult to target.

CHAPTER 3

METHODOLOGY

Experimental Data

The data used consists of 76 different parameters recorded on the aircraft Flight Data Recorder (FDR) as well as the vibration parameter. During the data processing phase of the project, two efforts were done to identify the parameters that most contributed to the engine vibration.

Data Correlation

Primarily, cross-correlation analysis [49] was exercised to find the potential parameters that highly contribute to vibration. Every parameter from each flight was cross-correlated to vibration then plotted to pick the highest correlated parameters. Cross correlation was calculated using the following Equation:

$$Cross\_Correlation = \sum_{a=-\infty}^{\infty} x[a] \cdot vib[a] \tag{9}$$

Highest correlation was determined by calculating the area under the plotted curve for the normalized data. Top correlated parameters to vibration were:

1. Right InBoard Spoiler

2. Right OutBoard Spoiler
3. Left InBoard Spoiler
4. Left OutBoard Spoiler
5. Static Air Temperature
6. Pitch 2
7. Pitch
8. Slat Configuration
9. Main Landing Gear Lock Down Sensore
10. Flap Configuration

A simple neural network was built as shown in Figure 7 to predict vibration given other parameters within the same second. However, the results were not good and there was much noise in the predictions. This imposed a question about the quality of the chosen parameters using this method and, another way of parameter-selection was sought. The potential cause for such misleading cross-correlation chosen parameters might be that some flight configuration parameters like spoilers/slats/flaps positions, pitch angle and main-landing-gear position do not change but few times during the flight. Consequently, this might be translated in high correlation with the vibration.

Aerodynamics/Turbo-machinery Parameters' Selection

A subset of the FDR parameters were chosen based on the likelihood of their contribution to the vibration based on aerodynamics/turbo-machinery

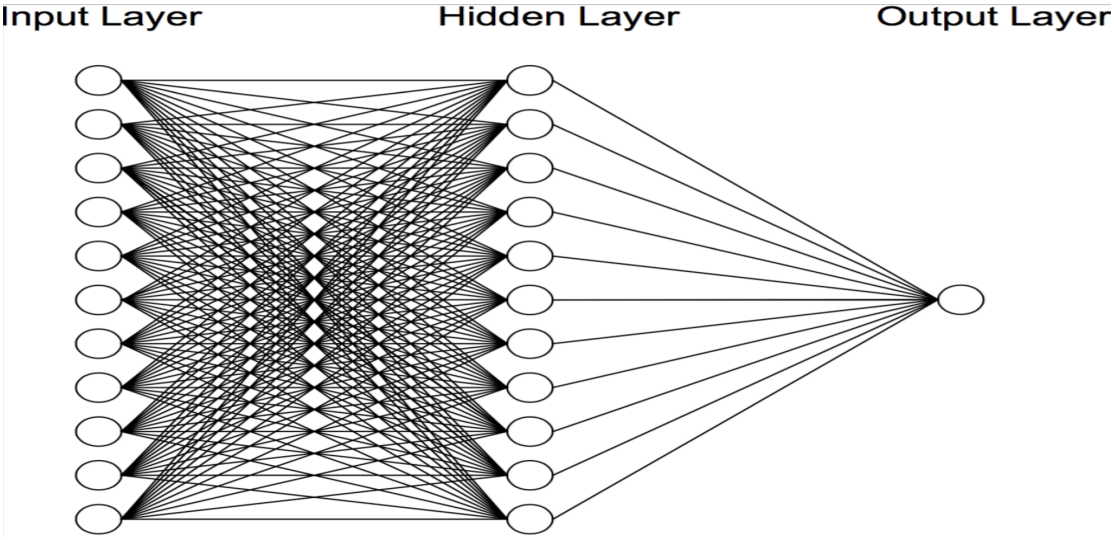


Figure 7: Simple Neural Network

background. Again, the simple neural network with a structure about the same of the one shown in Figure 7 was applied and this time results were encouraging enough to take it to the next level of prediction; predicting vibration in future.

Some parameters, such as Inlet Guide Vans Configuration, Fuel Flow, Spoilers Configuration (this was preliminary considered because of the special position of the engine mount), High Pressure Valve Configuration and Static Air Temperature were excluded because it was found that they generated noise more than positively contributing in the vibration prediction.

The final chosen parameters were:

1. Altitude
2. Angle of Attack
3. Bleed Pressure
4. Turbine Inlet Temperature

5. Mach Number
6. Primary Rotor/Shaft Rotation Speed
7. Secondary Rotor/Shaft Rotation Speed
8. Engine Oil pressure
9. Engine Oil Quantity
10. Engine Oil Temperature
11. Aircraft Roll
12. Total Air Temperature
13. Wind Direction
14. Wind Speed
15. Engine Vibration

Methodology

Neural Networks Overview

Three LSTM RNN architectures were designed to predict engine vibration 5 seconds, 10 seconds, and 20 seconds in the future. Each of the 15 selected FDR parameters is represented by a node in the inputs of the neural network and an additional node is used for a bais. Each neural network in the three designs consists of LSTM cells that receive both an initial input and the output of the previous cell, as inputs. Each cell has three gates to control the flow of information through the cell and accordingly, the output of the cell. Each cell

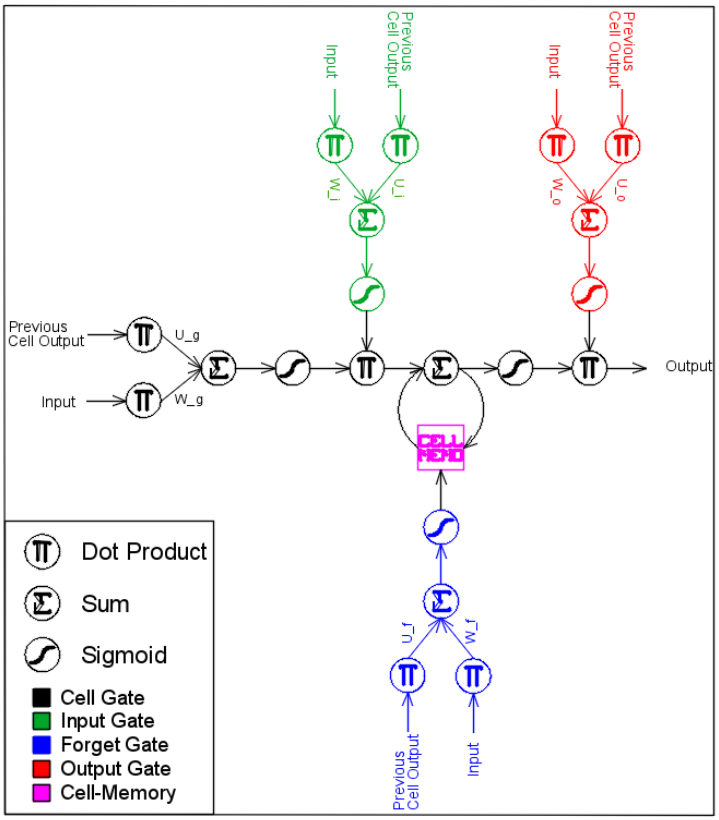


Figure 8: LSTM cell design

also has a cell-memory which is the core of the LSTM RNN design. The cell-memory allows the flow of information from the previous states into current predictions.

The gates that control the flow are shown in Figure 8. They are: *i)* the *i*nput gate, which controls how much information will flow from the inputs of the cell, *ii)* the *forget gate*, which controls how much information will flow from the cell-memory, and *iii)* the *output gate*, which controls how much information will flow out of the cell. This design allows the network to learn not only about the target values, but also about how to tune its controls to reach the target values.

All the utilized architectures have a common LSTM cell design shown in Figure 8. However, there are two variations of this common design used in the utilized architectures, shown in Figures 9 and 10, with the difference being the

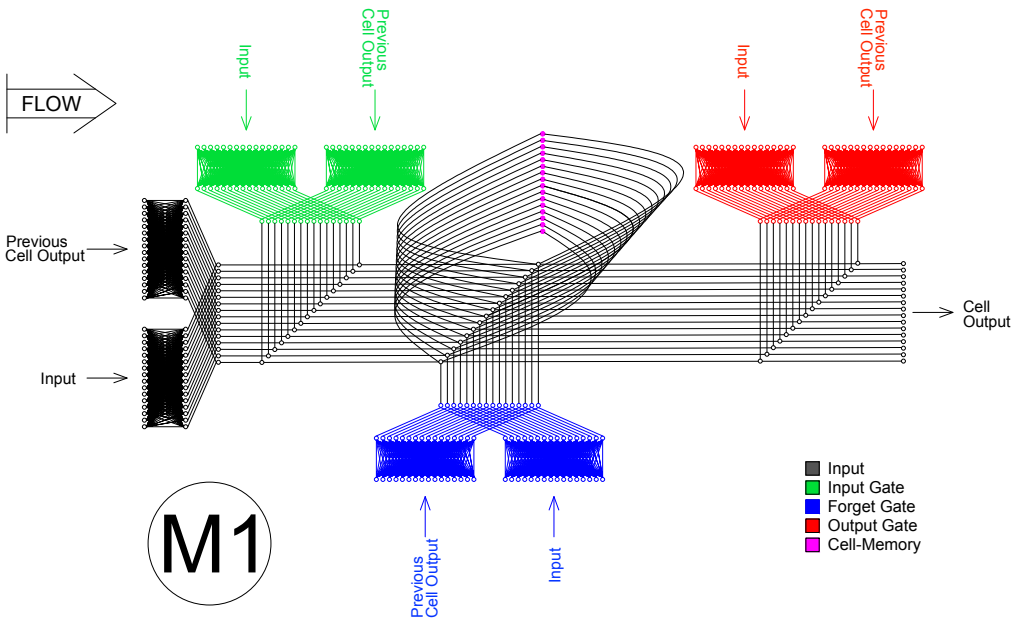


Figure 9: Level 1 LSTM cell design

number of inputs from the previous. Cells that take initial inputs from more input nodes are denoted by *'M1'* cells. As input nodes are needed to be reduced through the neural network, the design of the cell will be different and it is denoted by *'M2'* cells.

LSTM RNN Forward Propagation Equations

The equations used in the forward propagation through the neural network are:

$$i_t = Sigmoid(w_i \bullet x_t + u_i \bullet a_{t-1} + bais_i) \tag{10}$$

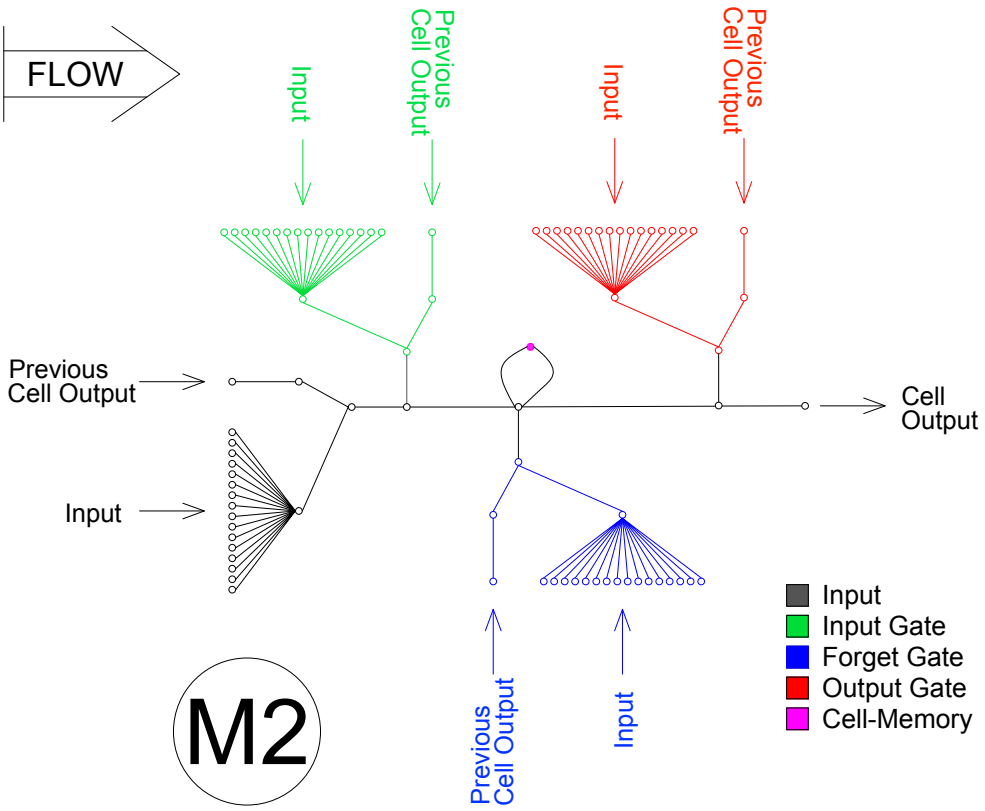


Figure 10: Level 2 LSTM cell design

$$f_t = Sigmoid(w_f \bullet x_t + u_f \bullet a_{t-1} + bais_f) \tag{11}$$

$$o_t = Sigmoid(w_o \bullet x_t + u_o \bullet a_{t-1} + bais_o) \tag{12}$$

$$g_t = Sigmoid(w_g \bullet x_t + u_g \bullet a_{t-1} + bais_g) \tag{13}$$

$$c_t = f_t \bullet c_{t-1} + i_t \bullet g_t \tag{14}$$

$$a_t = o_t \bullet Sigmoid(c_t) \tag{15}$$

where (see Figure 8):

i_t: input-gate output

f_t: forget-gate output

o_t: output-gate output

g_t: input's sigmoid

c_t: cell-memory output

w_i: weights associated with input and input-gate

u_i: weights associated with previous output and input-gate

w_f: weights associated with input and forget-gate

u_f: weights associated with previous output and forget-gate

w_o: weights associated with input and output-gate

u_o: weights associated with previous output and the output-gate

w_g: weights associated with the cell input

u_g: weights associated with previous output and the cell input

and the formula of the sigmoid function is:

$$Sigmoid(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{16}$$

LSTM RNN Architectures

The three architectures are as follows, with the dimensions of the weights of these architectures shown in Table 2 and the total number of weights shown in Table 3:

II.III.1 Architecture I

As shown in Figure 11, this architecture takes inputs from ten time series (the current time instant and the past nine). It feeds the second level of the neural network with its output. The output of the first level of the neural network is considered the first hidden layer. The second level of the neural network then reduces the number of nodes fed to it from 16 nodes (15 input nodes + bais) per cell to only one node per cell. The output of the second level of the neural network is considered the second hidden layer. Finally, the out of the second level of the neural network would be only 10 nodes, a node from each cell. These nodes are fed to a final neuron in the third level to compute the output of the whole network.

II.III.2 Architecture II

As shown in Figure 12, this architecture is almost the same as the previous one except that it does not have the third level. Instead, the output of the second level is averaged to compute the output of the whole network.

II.III.3 Architecture III

Figure 13 presents a deeper neural network architecture. In this design, the neural network takes inputs from twenty time series (the current time instant and the past nineteen). It feeds the second level of the neural network with its output. Second level does the same procedure as first level giving a chance for more abstract decision making. The output of the second level of the neural

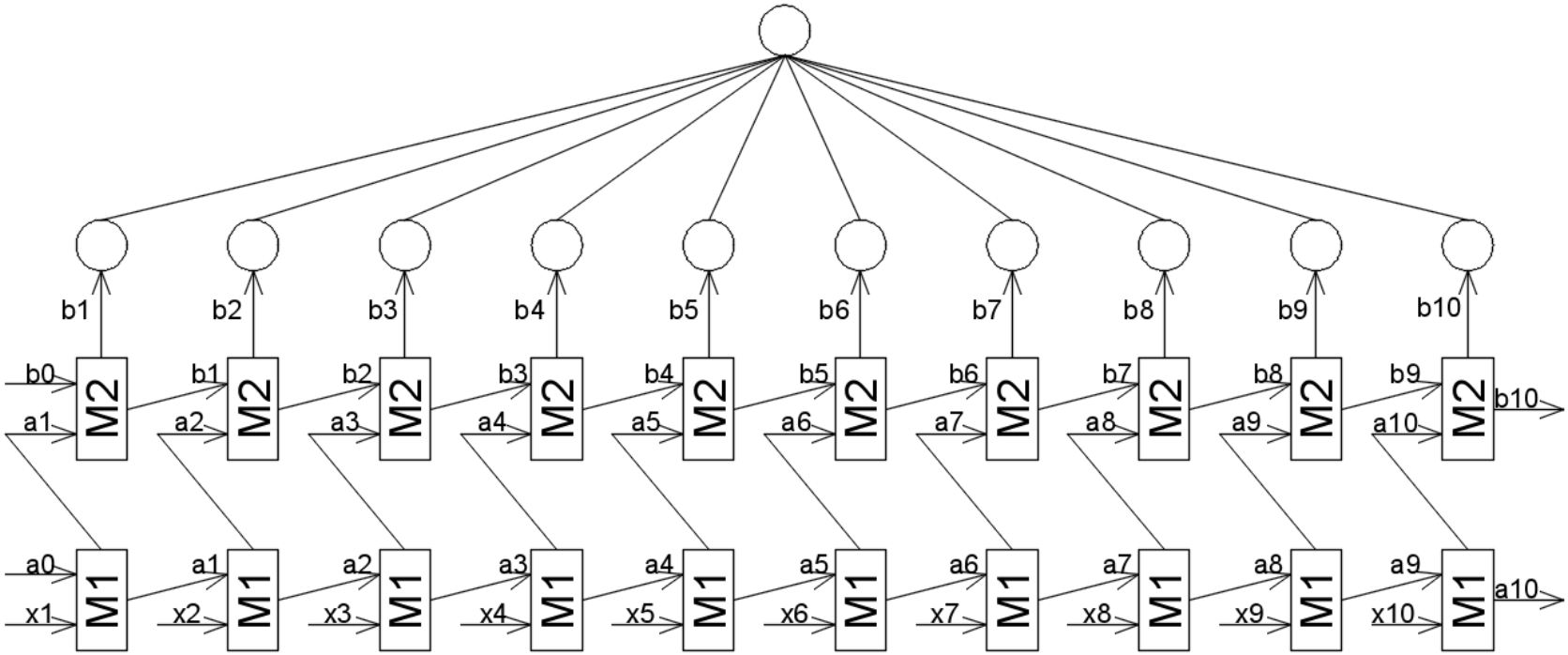


Figure 11: Architecture I

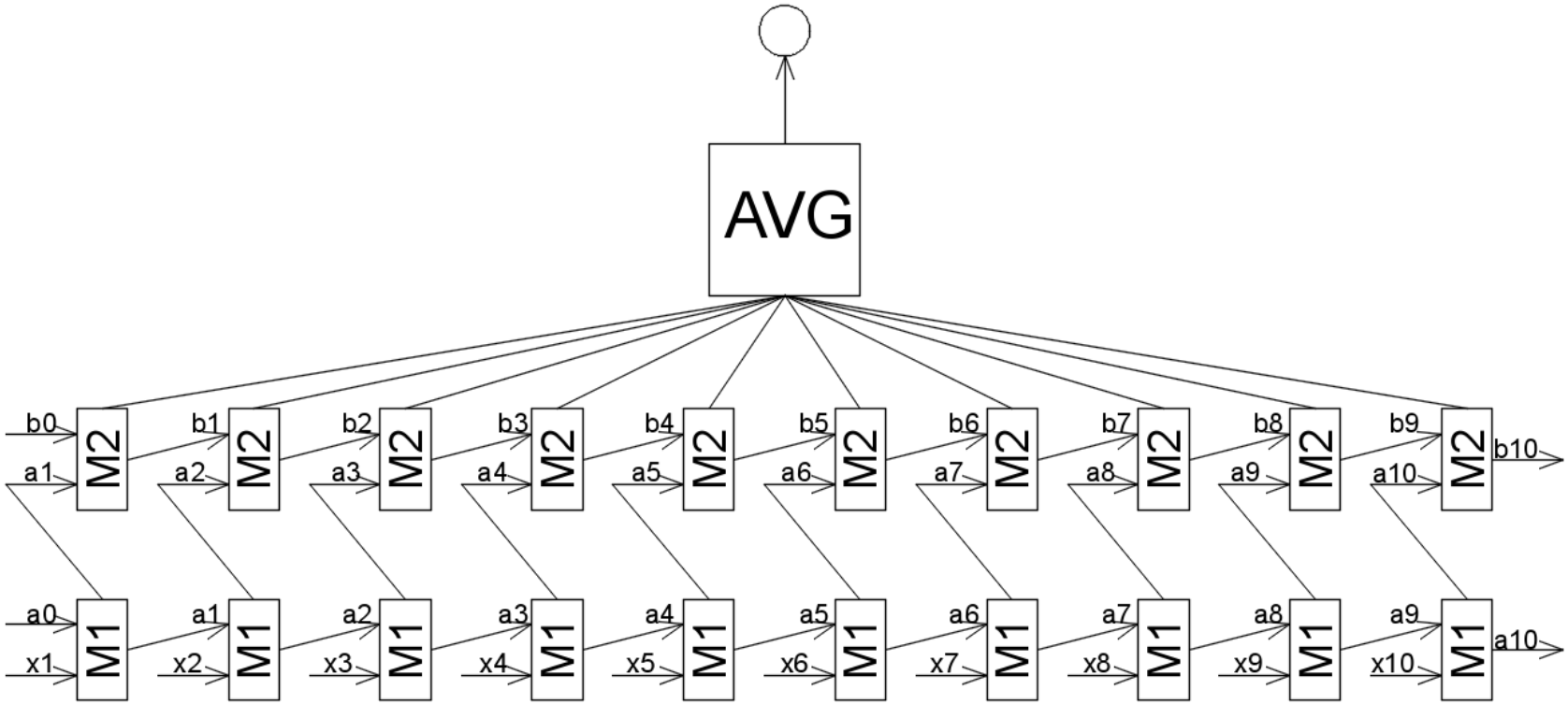


Figure 12: Architecture II

Table 2: Architectures Weights-Matrices Dimensions

Architecture I

	w_i	u_i	w_f	u_f	w_o	u_o	w_g	u_g
Level 1	16×16	16×16	16×16	16×16	16×16	16×16	16×16	16×16
Level 2	16×1	1×1	16×1	1×1	16×1	1×1	16×1	1×1
Level 3	16×1							

Architecture II

	w_i	u_i	w_f	u_f	w_o	u_o	w_g	u_g
Level 1	16×16	16×16	16×16	16×16	16×16	16×16	16×16	16×16
Level 2	16×1	1×1	16×1	1×1	16×1	1×1	16×1	1×1

Architecture III

	w_i	u_i	w_f	u_f	w_o	u_o	w_g	u_g
Level 1	16×16	16×16	16×16	16×16	16×16	16×16	16×16	16×16
Level 2	16×16	16×16	16×16	16×16	16×16	16×16	16×16	16×16
Level 3	16×1	1×1	16×1	1×1	16×1	1×1	16×1	1×1
Level 4	16×1							

Table 3: Architectures Weights Matrices' Total Elements

Architecture I	**Architecture II**	**Architecture III**
21,170	21,160	83,290

network is considered the first hidden layer and the output of the second level is considered the second hidden layer. The third level of the neural network then reduces the number of nodes fed to it from 16 nodes (15 input nodes + bais) per cell to only one node per cell. The output of the third level of the neural network is considered the third hidden layer. Finally, the output of the third level of the neural network is twenty nodes, a node from each cell. These nodes are fed to a final neuron in the fourth level to compute the output of the whole network.

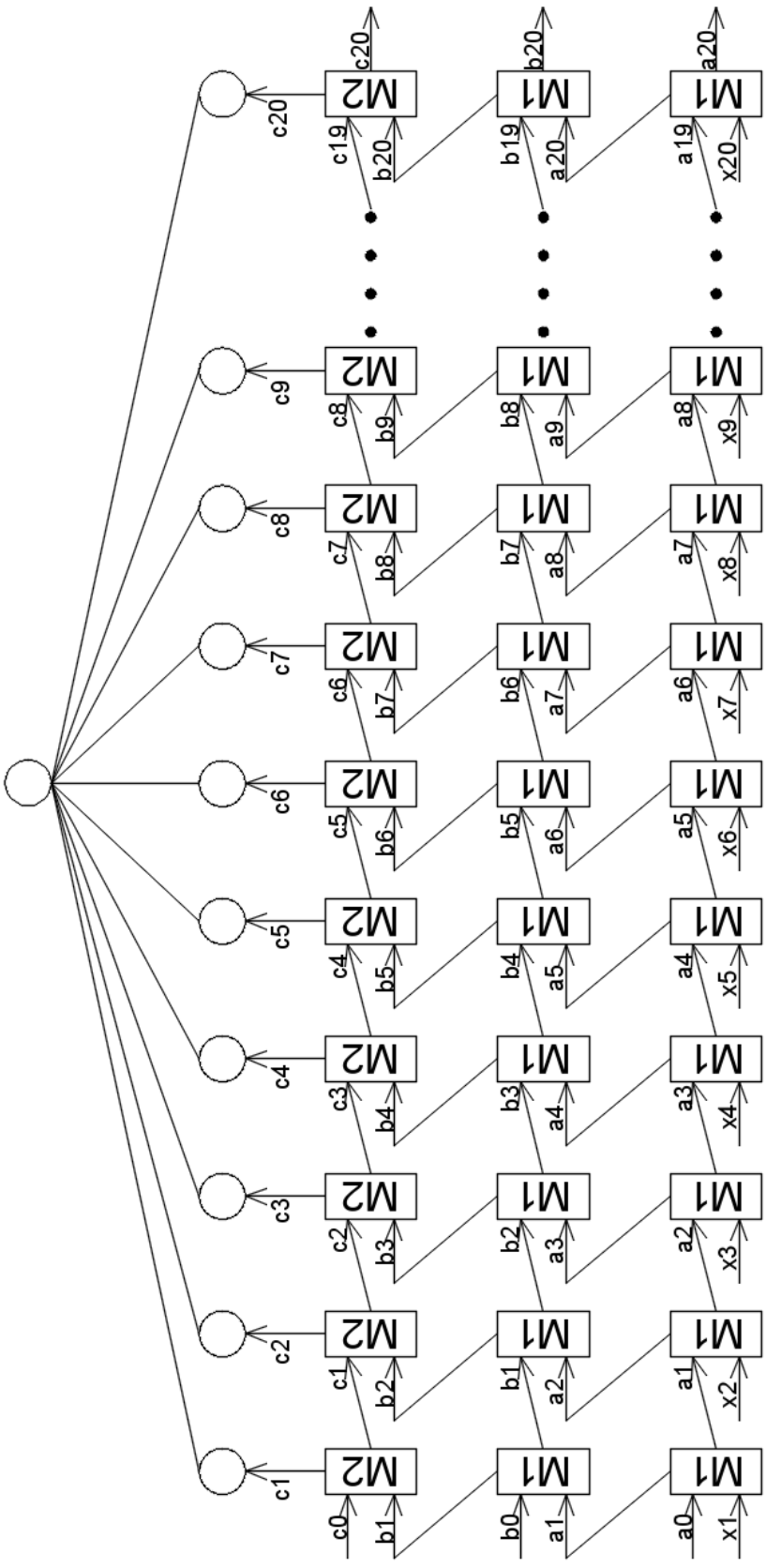


Figure 13: Architecture III

Forward Propagation

The following is a general description for the forward propagation path. This example uses Architecture I as an example but similar steps are taken in the other architectures with minor changes apparent in their diagrams. With Figure 11 presenting an overview of the structure of the whole network and considering Figure 9 as an overview of the structure of the cells in *Level 1* and in *Level 2* – the input at each iteration consists of 10 seconds of time series data of the 15 input parameters and 1 bais (*Input* in Figure 9) in one vector (x_t in Figure 11) and the output of the previous cell (*Previous Cell Output* in Figure 9) in another vector (a_{t-1} in Figure 11). Each second of time series input is fed to the corresponding cell (*i*.e., the first seconds' 15 parameters and 1 bais are fed to first cell, the second seconds' 15 parameters and 1 bais are fed to second cell, ...) into the *cell gate* (shown in black color), *input gate* (shown in green color), *forget gate* (shown in blue color) and the *output gate* (shown in red color). If the gates (*input gate*, *forget gate* and, *output gate*) are seen as valves that control how much of the data flow through it, the outputs of these gates (i_t, f_t and, o_t) are considered as how much these valves are opened or closed.

First, at the *cell gate*, x_t is dot multiplied by its weights matrix w_g and a_{t-1} is dot multiplied by its weights matrix u_g. The output vectors are summed and an activation function is applied to it as in Equation 13. The output is called g_t.

Second, at the *input gate*, x_t is dot multiplied by its weights matrix w_i and a_{t-1} is dot multiplied by its weights matrix u_i. The output vectors are summed and an activation function is applied to it as in Equation 10. The output is called i_t.

Third, at the *forget gate*, x_t is dot multiplied by its weights matrix w_f and a_{t-1} is dot multiplied by its weights matrix u_f. The output vectors are summed and an activation function is applied to it as in Equation 11. It controls how much of the *cell memory* Figure 11 (saved from previous time step) should pass. The output is called f_t.

Fourth, at the *output gate*, x_t is dot multiplied by its weights matrix w_o and a_{t-1} is dot multiplied by its weights matrix u_o. The output vectors are summed and an activation function is applied to it as in Equation 12. The output is called o_t.

Fifth, the contribution of the cell input *Input* g_t and *cell memory* c_{t-1} is decided in Equation 14 by dot multiplying them by f_t and i_t respectively. The output of this step is the new *cell memory* c_t.

Sixth, cell output is also regulated by the output gate (valve). This is done by applying the sigmoid function to the *cell memory* c_t and dot multiplying it by o_t as shown in Equation 15. The output of this step is the final output of the cell at the current time step a_t. a_t is fed to the next cell in the same level and also fed to the cell in the above level as an *Input* a_t.

The same procedure is applied at *Level 2* but with different weight vectors and different dimensions. Weights at *Level 2* have smaller dimensions to reduce their input dementions from vectors with 16 dimensions to vectors with one dimension. The output from *Level 2* a one dimensional vector from each cell of the 10 cells in *Level 2*. These vectors are fed as one 10 dimensional vector to a simple neuron shown in Figure 11 at *Level 3* to be dot multiplied by a weight vector to reduce the vector to a single scalar value: the final output of the network at the time step.

CHAPTER 4

IMPLEMENTATION

Programming Langauge

Python's Theano Library [50] was used to implement the neural networks. It has four major advantages: *i)* it will compile the most, if not all, of functions coded using it to C and CUDA giving fast performance, *ii)* it will perform the weights updates for back propagation with minimal overhead, *iii)* Theano can compute the gradients of the error (cost function output) with respect to the weights saving significant effort and time needed to manually derive the gradients, coding and debugging them, and finally, *iv)* it can utilize GPU's for further increased performance.

Data Processing

The flight data parameters used were normalized between 0 and 1. The sigmoid function is used as an activation function over all the gates and inputs/outputs. The ArcTan activation function was tested on the data, however it gave distorted results and sigmoid function provided significantly better performance.

Table 4: Run Time (hours)

	05 sec	**10 sec**	**20 sec**
Architecture I	9	8.98	8.85
Architecture II	8.44	8.41	8.4
Architecture III	21.6	19.7	18.5

Machine Specifications

Each of the examined architectures runs on a hyperthreaded 3.5 GHz core and is considered capable of real-time processing. Results were collected using a Mac Pro with 12 logical cores, with each different architecture being trained for 575 epochs. Run times for training are shown in Table 4. Some unexpected variance might be realized in these run-times, due to CPU interruptions which may have occurred over the course of the experiments. In general, the first two architectures took similar amounts of time (approximately 8.5-9 hours) for each time prediction (5, 10 and 20 seconds), and the third took a bit more than twice as long, at approximately 20 hours for each time prediction.

Optimizing Matrices for GPU Computations

The neural networks weight matrices for a LSTM cell is repeated at a given time step at a given layer. Thus, the computational cost would increase if the output if these gates would be computed separately, one gate at a time, as data input/output would consume CPU cycles. This case is also obvious if a GPU is utilized for high performance computing as the cost of sending data forward and backward between the CPU (host) and GPU (device). For that, the input of a cell at a given layer is dot multiplied by a matrix that holds all of the gates weights concatenated one after the other. Then, the outputs; g Equation 13, i Equation 10, f Equation 11 and, o Equation 12, can be extracted from the dot

product output matrix. Equation 17 is an example of combining (concatenating) the weights matrices for the LSTM cells' gates of layer one in Architecture I. By this, all weights are transferred between the CPU and the GPU as one data structure, which would theoretically boost the performance.

These measures were followed when applied GPU using the Theano library to manage the GPU threads, blocks and grids as well as the data transfer between the CPU and GPU. However, the performance was not better compared to the pure CPU version. For Architecture I as an example, for one iteration through the network during the learning process, it took the GPU version more than twenty minutes while it took slightly more than two minutes for the pure CPU version.

A further effort was made to overcome the data transfer penalty between the CPU and the GPU. The whole input data set was sent to the GPU as one data structure to avoid the data transfer through the iterations at every time series in the data and to perform those iterations on the GPU. Unfortunately, this also did not help with the performance.

Ultimately, a conclusion was reached that the subject matrices are not large enough to overcome the data transfer overhead.

Table 5: Training Results

	Prediction Error			
	1 seconds	**5 seconds**	**10 seconds**	**20 seconds**
Architecture I	0.000154	0.000398	0.000972	0.001843
Architecture II	0.001239	0.001516	0.001962	0.002870
Architecture III	0.000133	0.000409	0.000979	0.001717

$$\begin{bmatrix} w_{g_{1,1}} & w_{g_{1,2}} & w_{g_{1,3}} & \dots & w_{g_{1,16}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{g_{16,1}} & w_{g_{16,2}} & w_{g_{16,3}} & \dots & w_{g_{16,16}} \\ \hdashline w_{i_{1,1}} & w_{i_{1,2}} & w_{i_{1,3}} & \dots & w_{i_{1,16}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{i_{16,1}} & w_{i_{16,2}} & w_{i_{16,3}} & \dots & w_{i_{16,16}} \\ \hdashline w_{f_{1,1}} & w_{f_{1,2}} & w_{f_{1,3}} & \dots & w_{f_{1,16}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{f_{16,1}} & w_{f_{16,2}} & w_{f_{16,3}} & \dots & w_{f_{16,16}} \\ \hdashline w_{o_{1,1}} & w_{o_{1,2}} & w_{g_{1,3}} & \dots & w_{o_{1,16}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{o_{16,1}} & w_{o_{16,2}} & w_{o_{16,3}} & \dots & w_{o_{16,16}} \end{bmatrix} \odot \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ \vdots \\ x_{16} \end{bmatrix} = \begin{bmatrix} out_{g_1} \\ \vdots \\ out_{g_{16}} \\ \hdashline out_{i_1} \\ \vdots \\ out_{i_{16}} \\ \hdashline out_{f_1} \\ \vdots \\ out_{f_{16}} \\ \hdashline out_{o_1} \\ \vdots \\ out_{o_{16}} \end{bmatrix} \tag{17}$$

Training Results

Training process results are shown in Table 5. These results are directly proportional to the testing results as will be shown in the results chapter. The errors shown are mean squared error.

CHAPTER 5

RESULTS

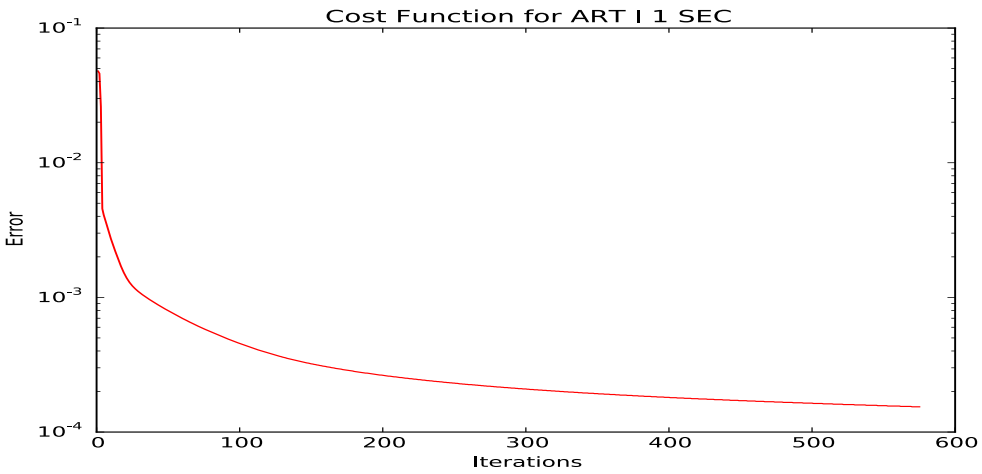
The neural networks were run against flights that suffered from the excessive vibration in a training phase. They were then run against different set of flights, which also suffered from the same problem, in a testing phase. There were 28 flights in the training set, with a total of 41,431 seconds of data. There were 29 flights in the testing set, with a total of 38,126 seconds of data. The networks were allowed to train for 575 epochs to learn and for the cost function output curve to flatten.

Cost Function

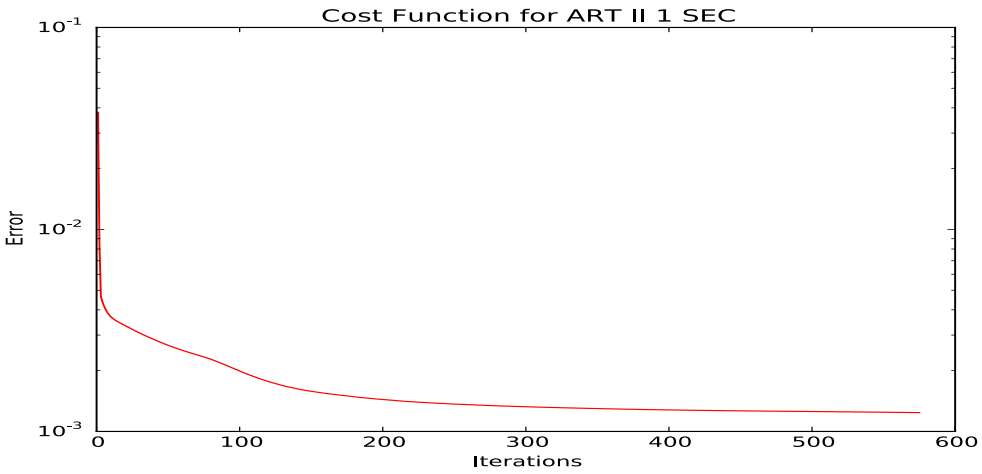
Mean squared error was used to train the neural networks as it provides a smoother optimization surface for backpropagation. The cost function output for predicting 1 sec, 5 sec, 10 sec and, 20 sec can be seen in Figures 14, 15, 16 and, 17 respectively. The Figure is a logarithmic plot for the three architectures, for predicting vibrations 10 seconds in the future.

Architecture Results

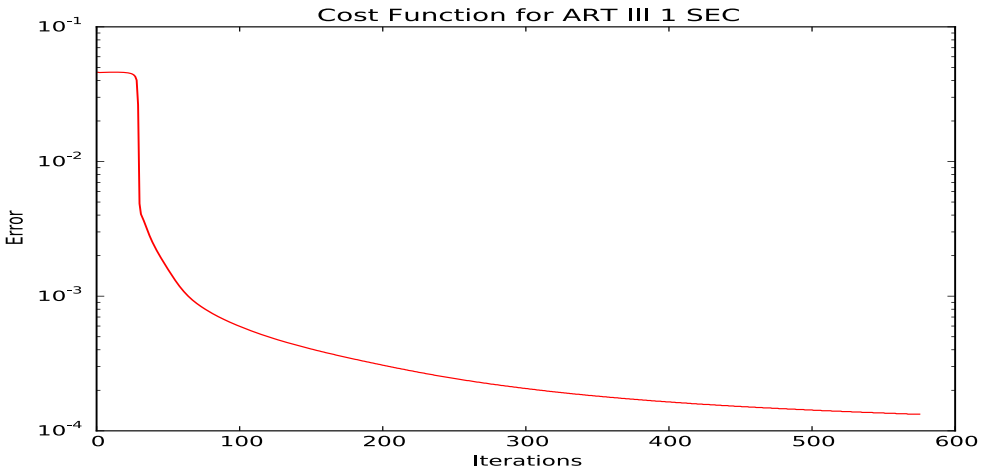
Mean Squared Error (MSE) (shown in Equation 18) was used as an error measure to train the three architectures, which resulted in values shown in



(a) ART I Cost Plot @ 1 SEC

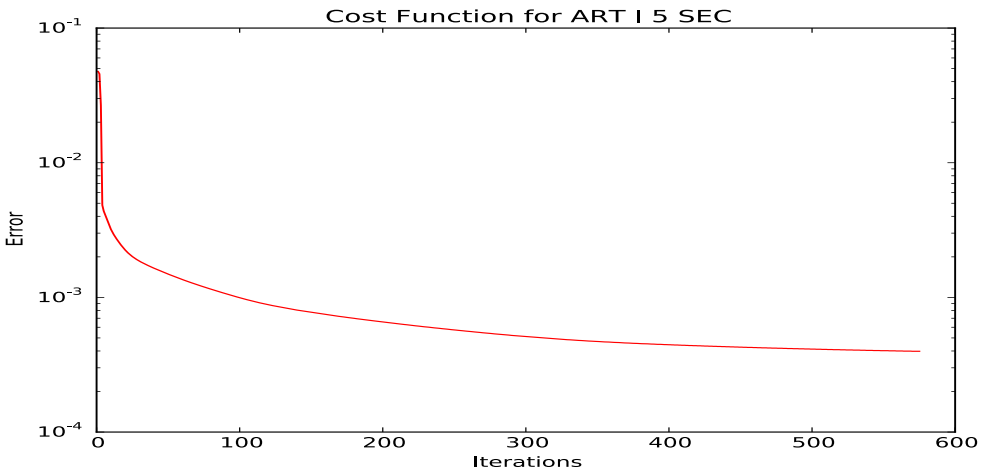


(b) ART II Cost Plot @ 1 SEC

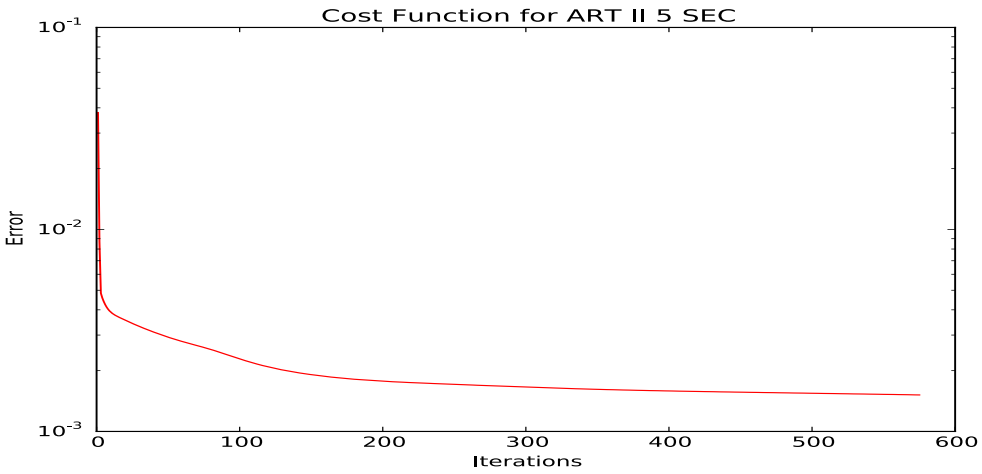


(c) ART III Cost Plot @ 1 SEC

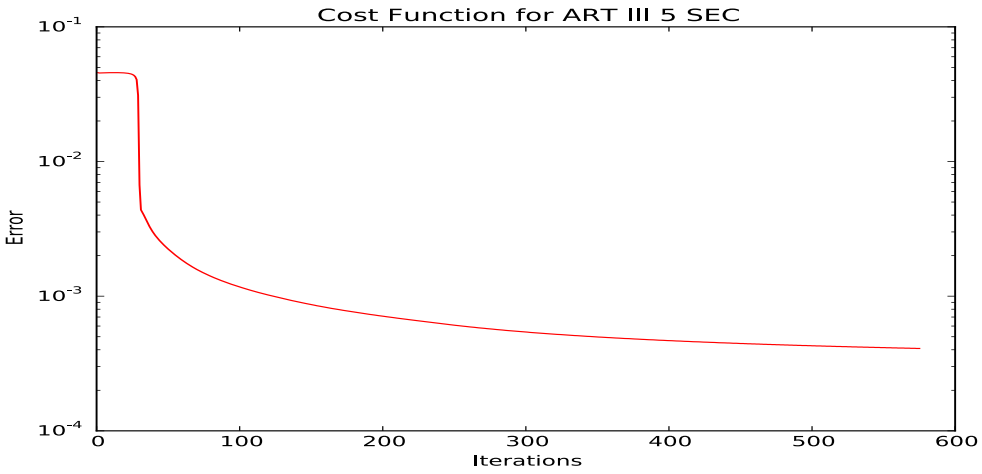
Figure 14: Cost function plot for the three architectures predicting vibration in 1 future sec.



(a) ART I Cost Plot @ 5 SEC

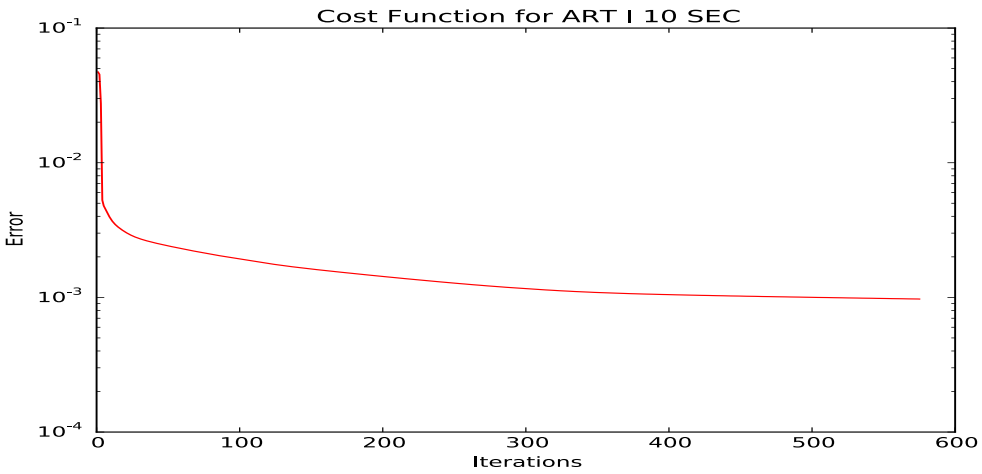


(b) ART II Cost Plot @ 5 SEC

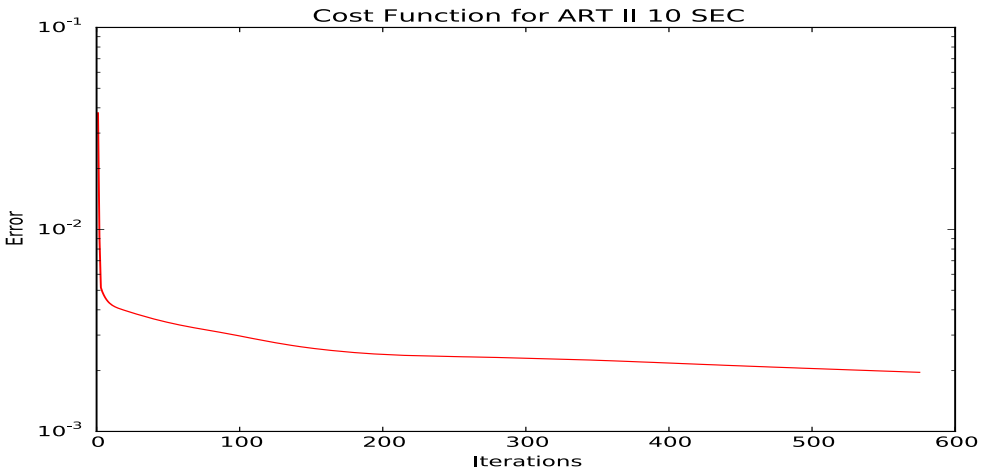


(c) ART III Cost Plot @ 5 SEC

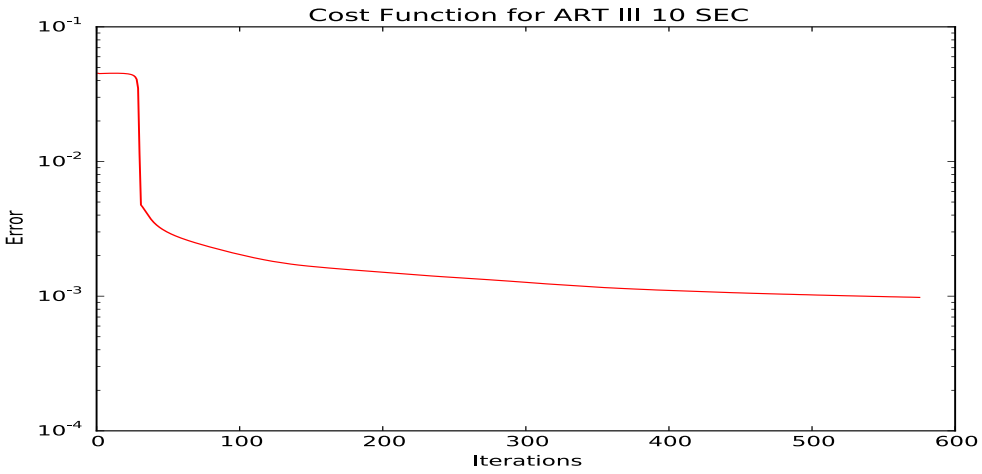
Figure 15: Cost function plot for the three architectures predicting vibration in 5 future sec.



(a) ART I Cost Plot @ 10 SEC

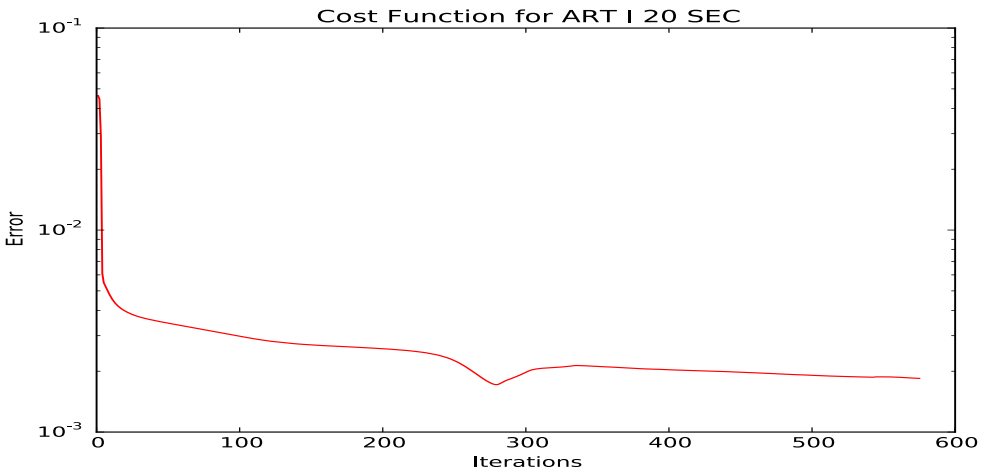


(b) ART II Cost Plot @ 10 SEC

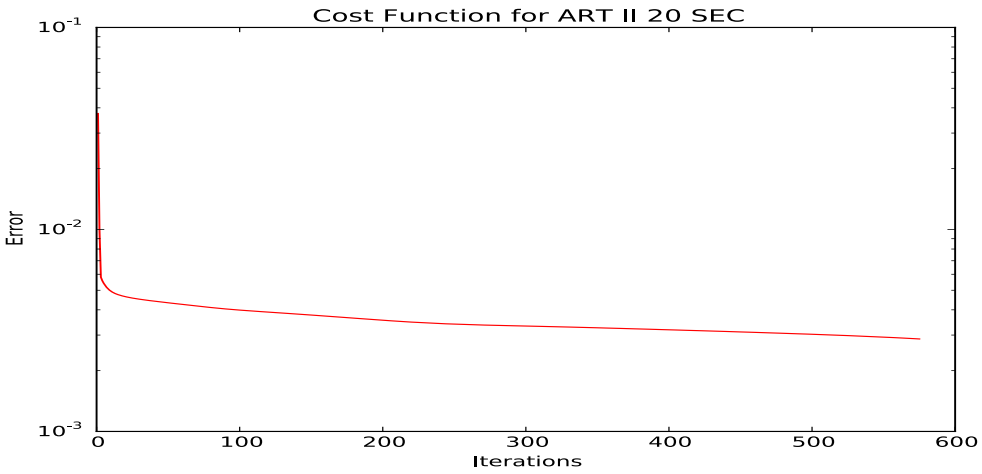


(c) ART III Cost Plot @ 10 SEC

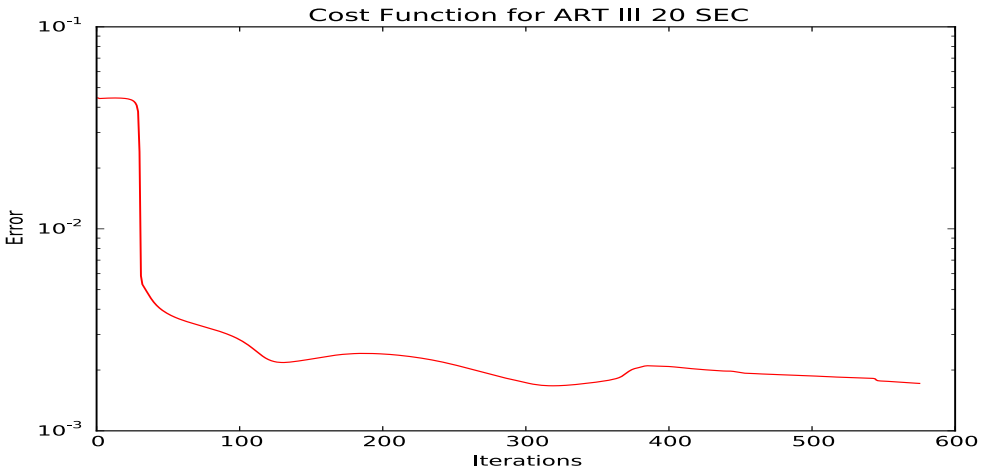
Figure 16: Cost function plot for the three architectures predicting vibration in 10 future sec.



(a) ART I Cost Plot @ 20 SEC



(b) ART II Cost Plot @ 20 SEC



(c) ART III Cost Plot @ 20 SEC

Figure 17: Cost function plot for the three architectures predicting vibration in 20 future sec.

Table 6: Testing Process Mean Squared Error

	Prediction Error			
	1 seconds	**5 seconds**	**10 seconds**	**20 seconds**
Architecture I	0.000792	0.001165	0.002926	0.010427
Architecture II	0.010311	0.009708	0.009056	0.012560
Architecture III	0.000838	0.002386	0.004780	0.041417

Table 7: Testing Process Mean Absolute Error

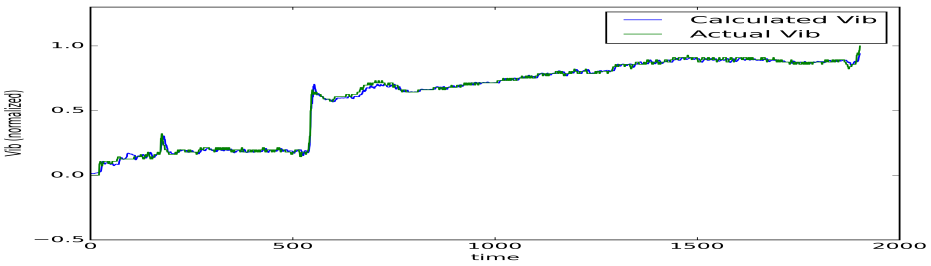
	Prediction Error			
	1 seconds	**5 seconds**	**10 seconds**	**20 seconds**
Architecture I	0.028407	0.033048	0.055124	0.101991
Architecture II	0.098357	0.097588	0.096054	0.112320
Architecture III	0.027621	0.048056	0.070360	0.202609

Table 6. Mean Absolute Error (MAE) (shown in Equation 19) is used as a final measure of accuracy for the three architectures, with results shown in Table 7. As the parameters were normalized between 0 and 1, the MAE is also the percentage error.

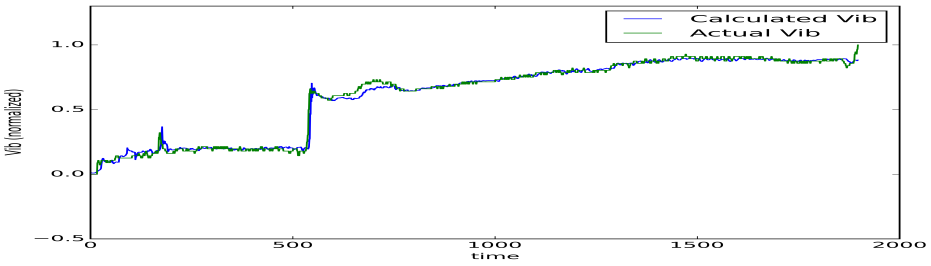
$$Error = \frac{0.5 \times \sum(Actual\ Vib - Predicted\ Vib)^2}{Testing\ Seconds} \tag{18}$$

$$Error = \frac{\sum[ABS(Actual\ Vib - Predicted\ Vib)]}{Testing\ Seconds} \tag{19}$$

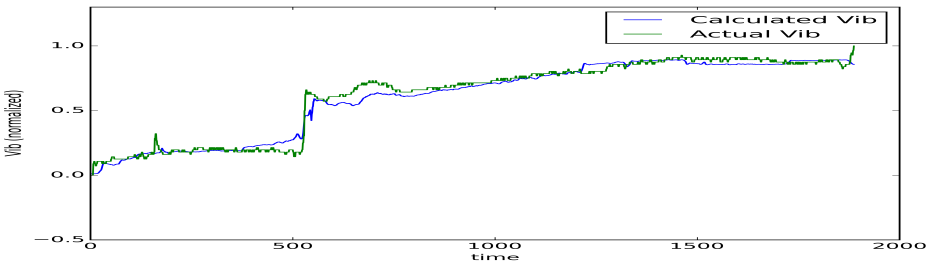
Figures 20, Figures 21, and Figures 22 present the predictions for all the test flights condensed on the same plot. Time shown on the x-axis is the total time for all the test flights. Each flight ends when the vibration reaches the max critical value (normalized to 1) and then the next flight in the test set beings. Figure 19 provides an uncompressed example of Architecture I predicting vibration 5, 10 and, 20 seconds in the future over a single flight from the testing data.



(a) ART I predicting vibration 5 seconds in the future for one flight.

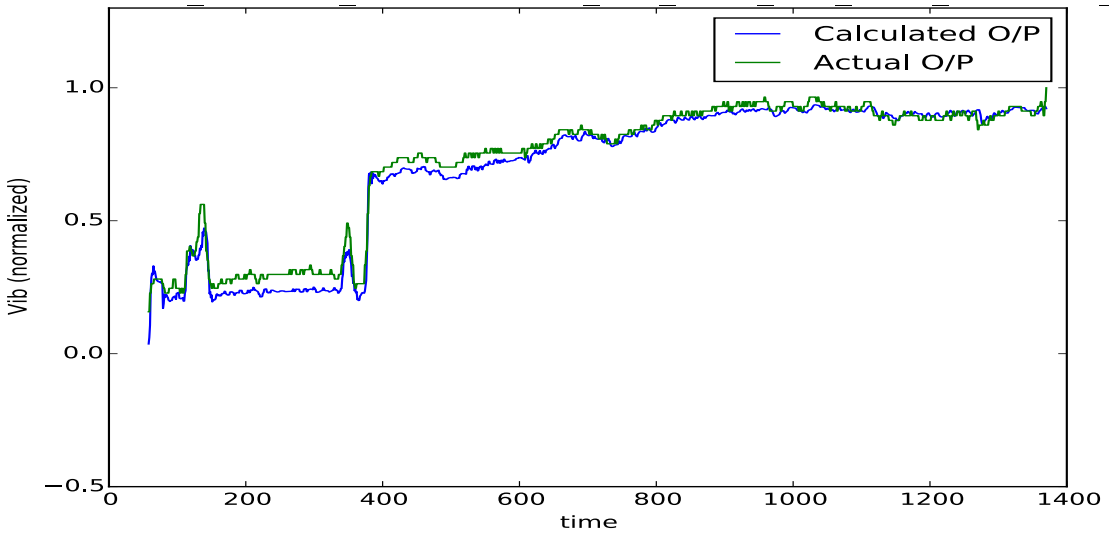


(b) ART I predicting vibration 10 seconds in the future for one flight.

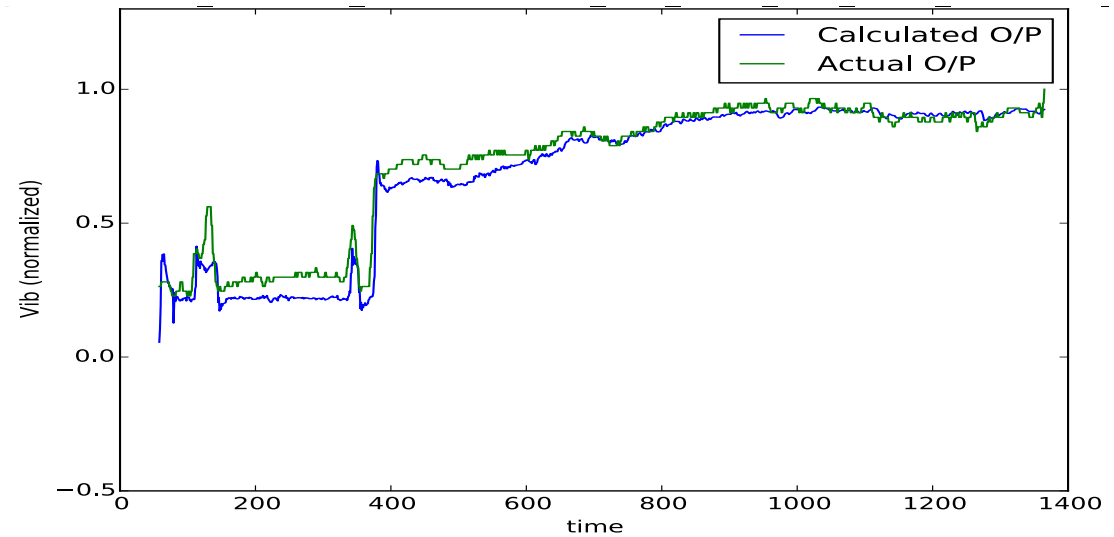


(c) ART I predicting vibration 20 seconds in the future for one flight.

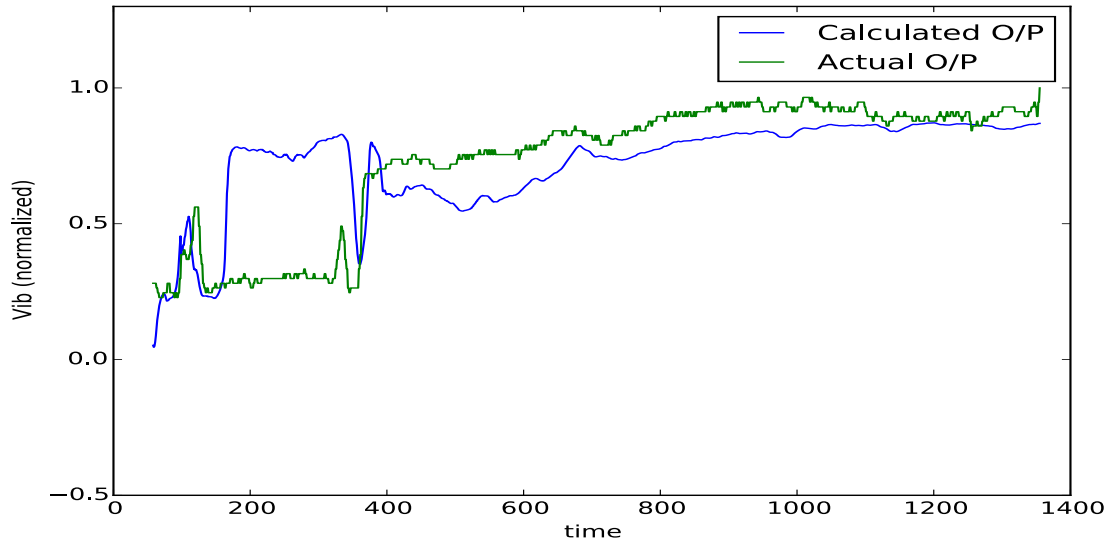
Figure 18: Architecture I predicting vibration for one flight.



(a) ART III predicting vibration 5 seconds in the future for one flight.

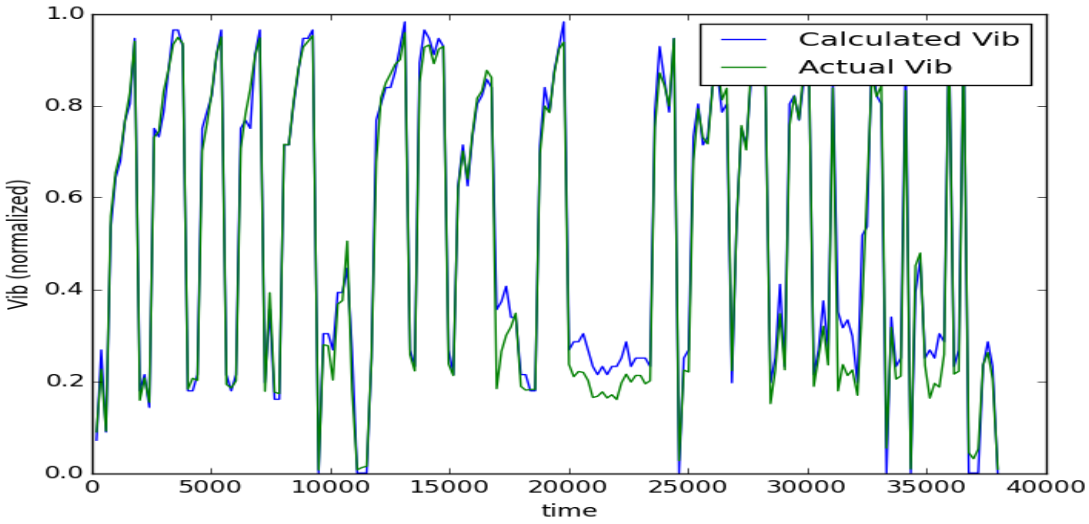


(b) ART III predicting vibration 10 seconds in the future for one flight.

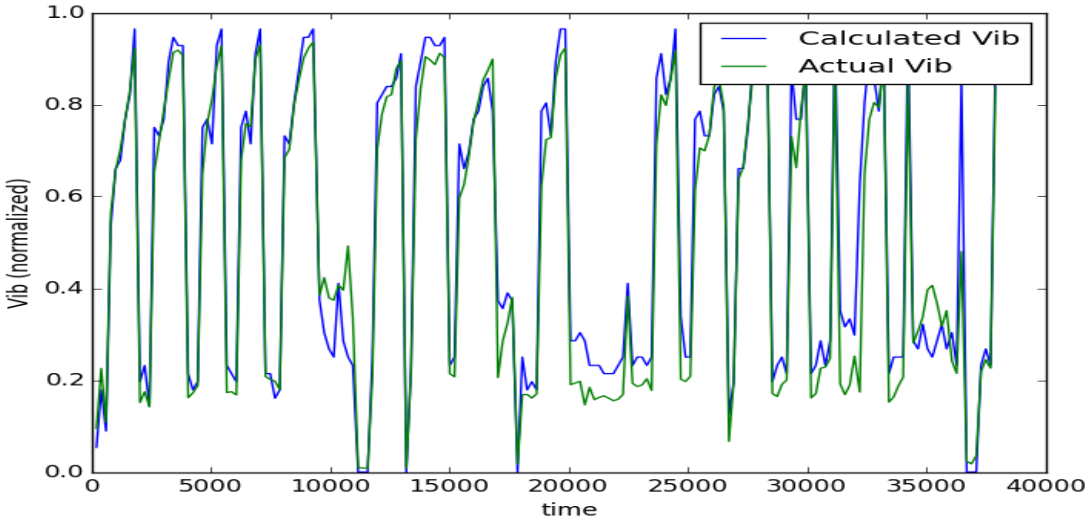


(c) ART III predicting vibration 20 seconds in the future for one flight.

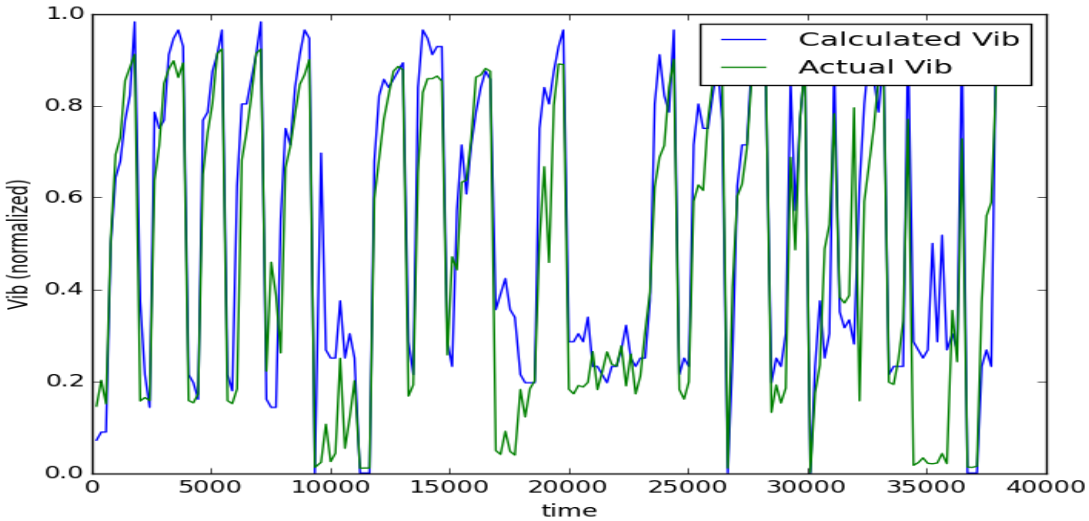
Figure 19: Architecture III predicting vibration for one flight.



(a) ART I Results Plot @ 05 SEC

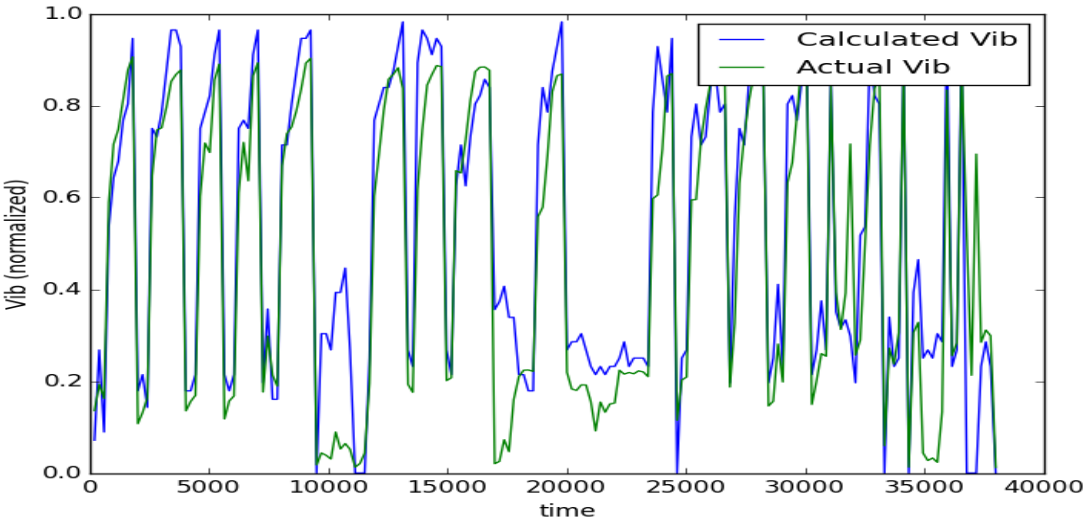


(b) ART I Results Plot @ 10 SEC

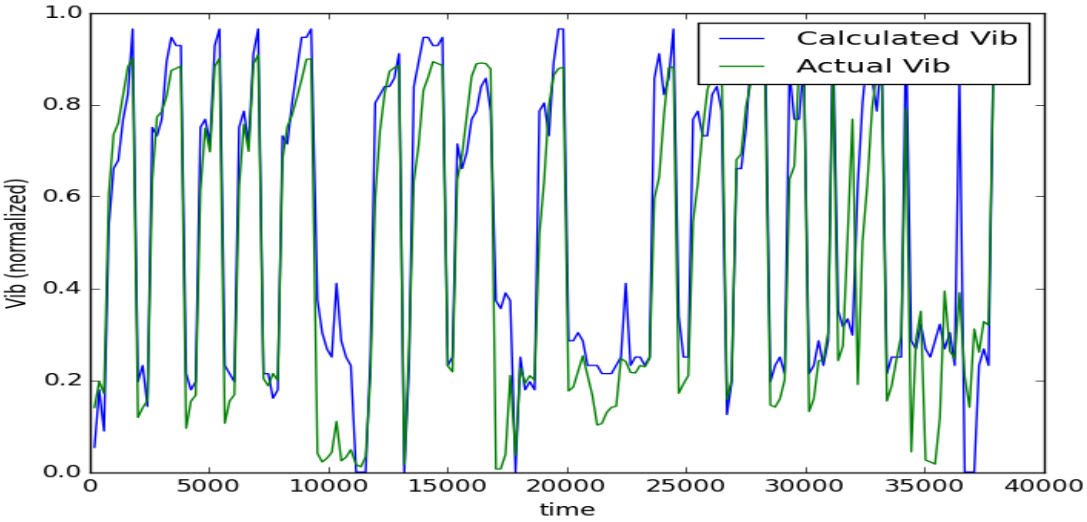


(c) ART I Results Plot @ 20 SEC

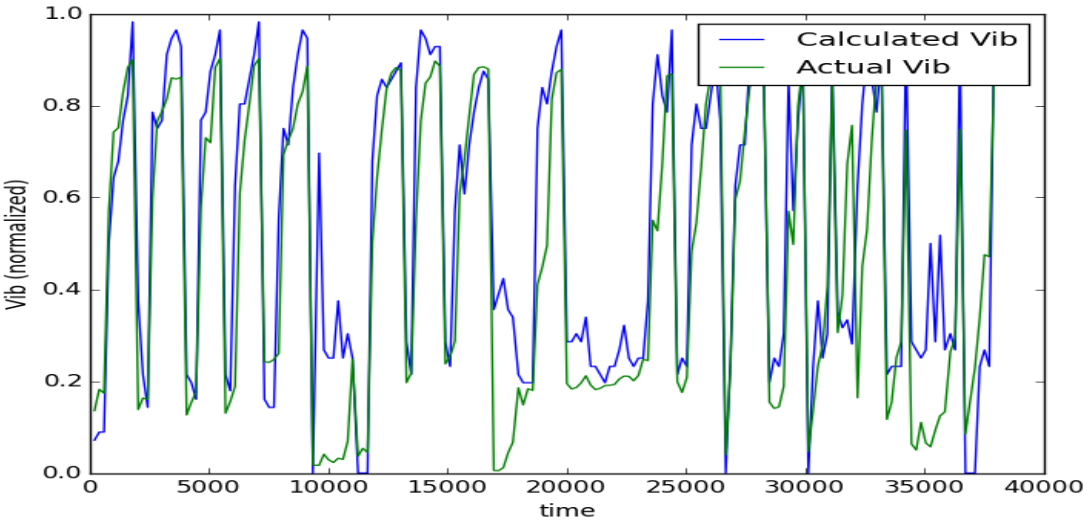
Figure 20: Plotted results for Architecture I for the for the three scenarios.



(a) ART II Results Plot @ 05 SEC

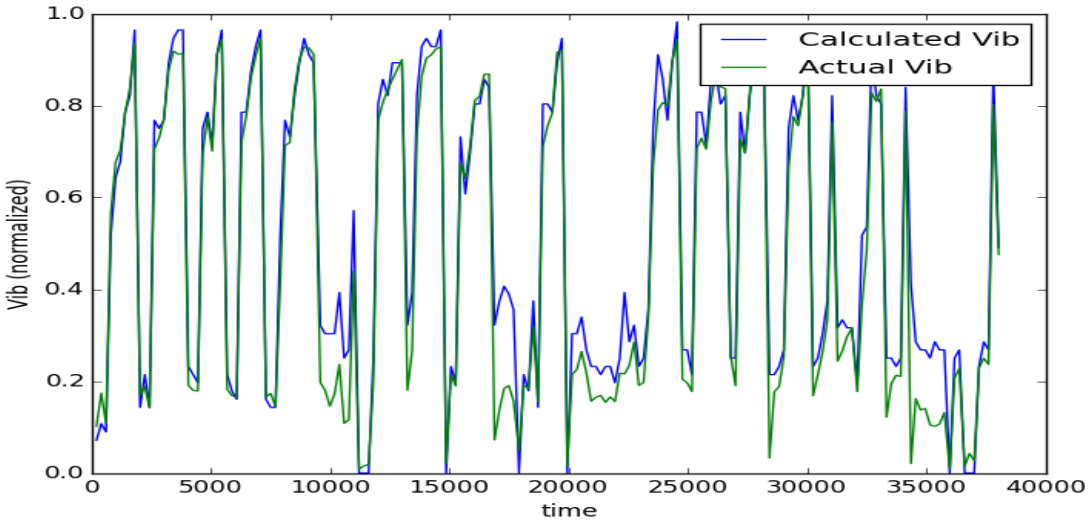


(b) ART II Results Plot @ 10 SEC

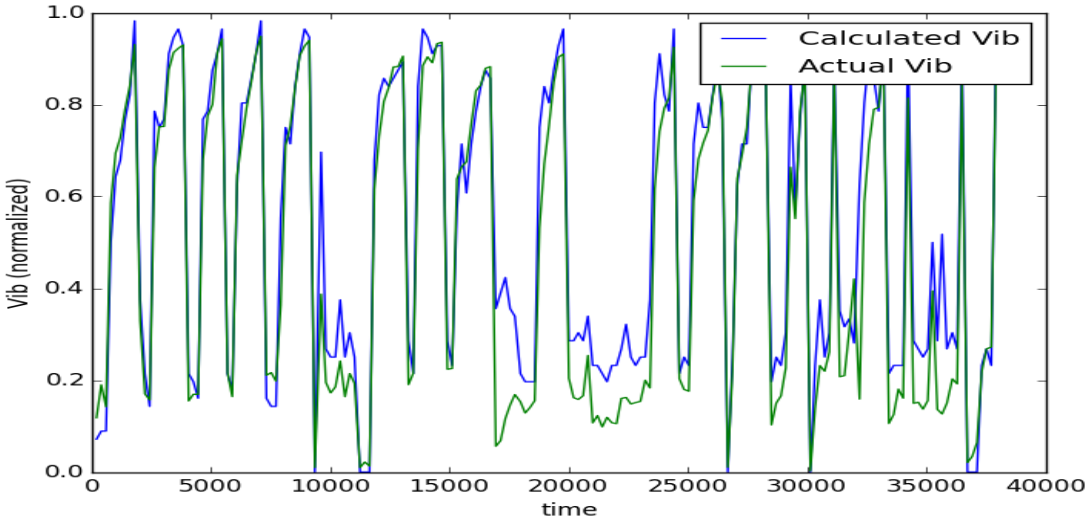


(c) ART II Results Plot @ 20 SEC

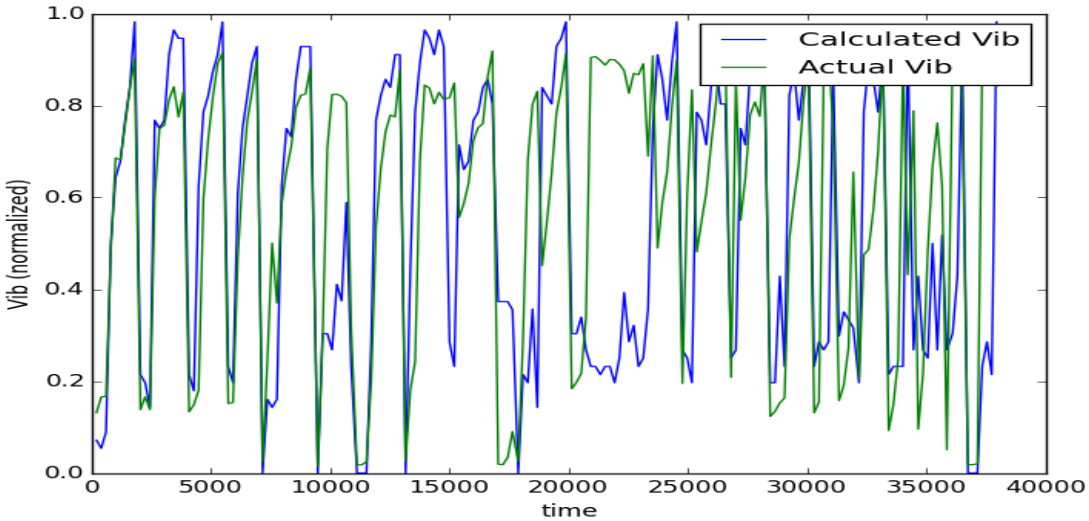
Figure 21: Plotted results for Architecture II for the for the three scenarios.



(a) ART III Results Plot @ 05 SEC



(b) ART III Results Plot @ 10 SEC



(c) ART III Results Plot @ 20 SEC

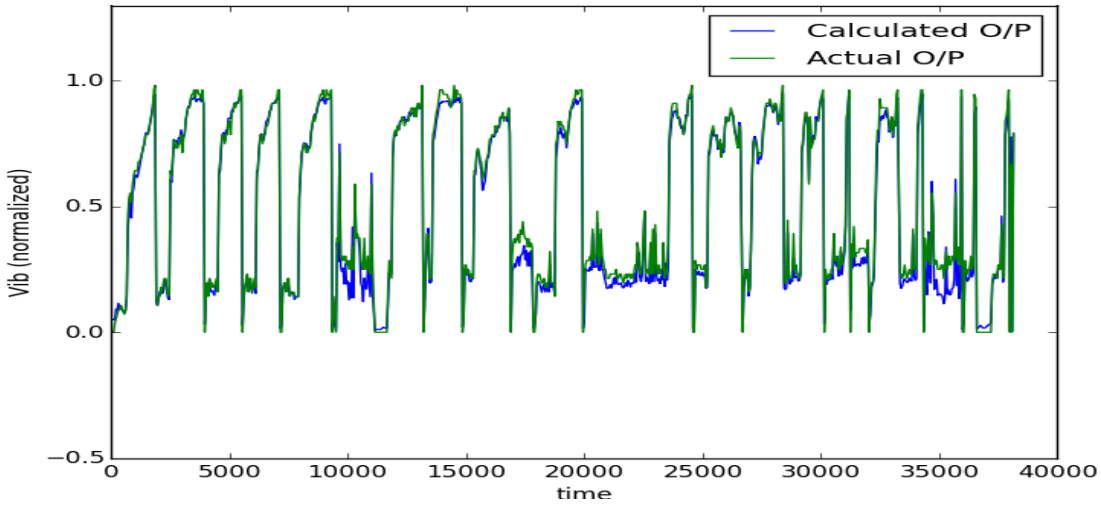
Figure 22: Plotted results for Architecture III for the for the three scenarios.

Results of Architecture I

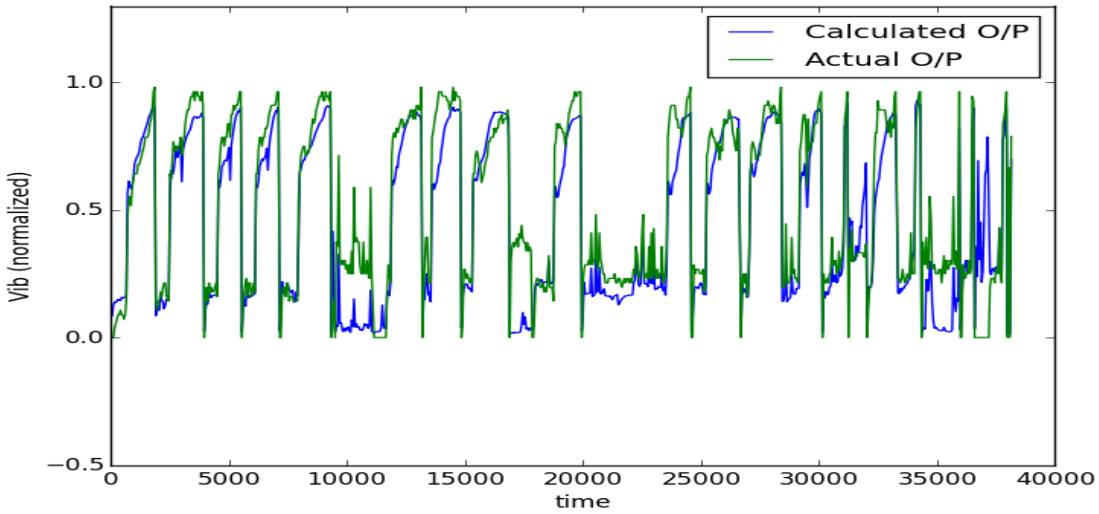
The results of this architecture, shown in Table 6, came out to be the best results regarding the overall accuracy of the vibration prediction. While there is more misalignment between the actual and calculated vibration values as predictions are made further in the future, as shown in Figure 20, this is to be expected. Also, it can be seen that the prediction of higher peaks is more accurate than the lower peaks prediction as if the neural network is tending to learn more about the max critical vibration value, which is favorable for this project. To test this further, this architecture was trained and tested on the same data set but for predicting vibration just one second in future. As expected, the results showed improvement in mean absolute error over all the test flights by about 0.5% compared to the results of the same architecture predicting five seconds in the future. A plot of the test data prediction for this experiment is shown in Figure 23a. Also, for comparison, a plot for the same flights plotted in Figure 19 is shown in Figure 24a.

Results of Architecture II

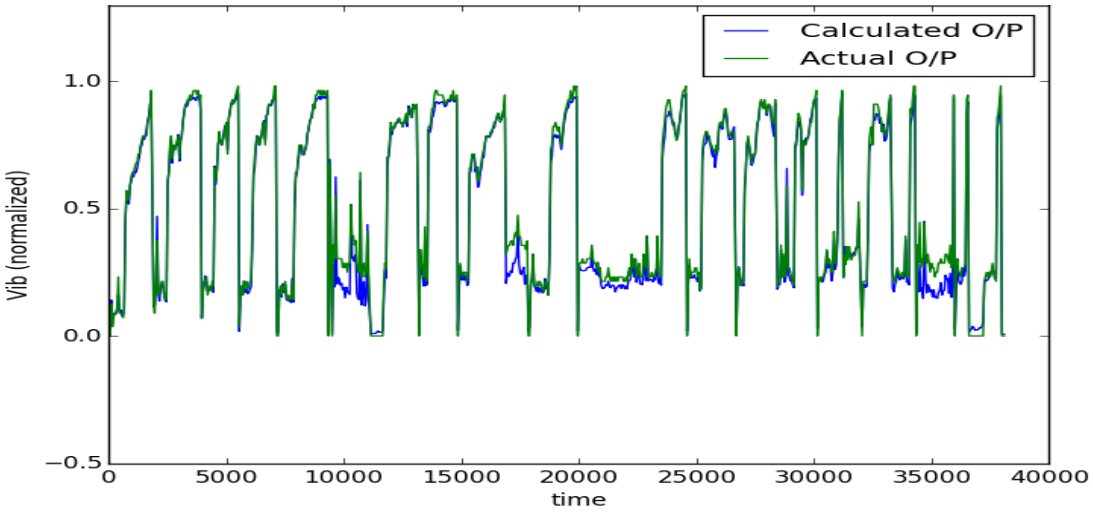
The results of this architecture in Table 6 came out to be the least successful in vibration prediction. While it managed to predict much of the vibration, its performance was weak at the peaks (either low or high) compared to the other architectures, as shown in Figure 21. It is also worth mentioning that somehow the lower peaks were better at some positions on the curve of this architecture, compared to to the other architectures.



(a) Architecture I predicting vibration 1 second in the future.

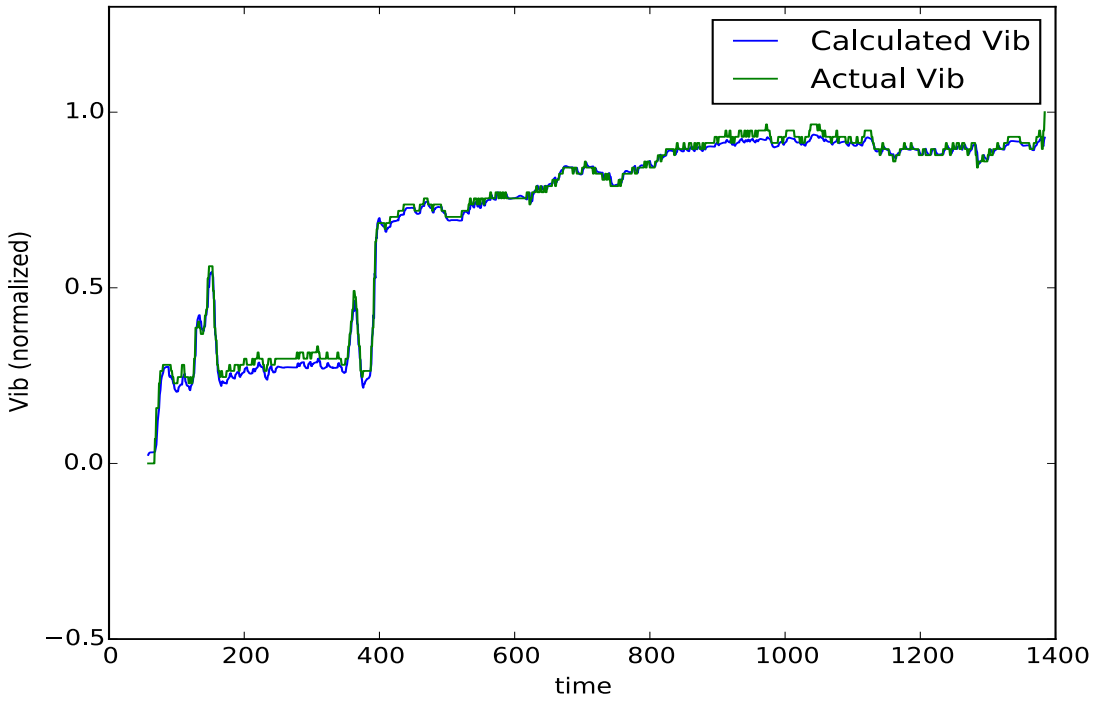


(b) Architecture II predicting vibration 1 second in the future.

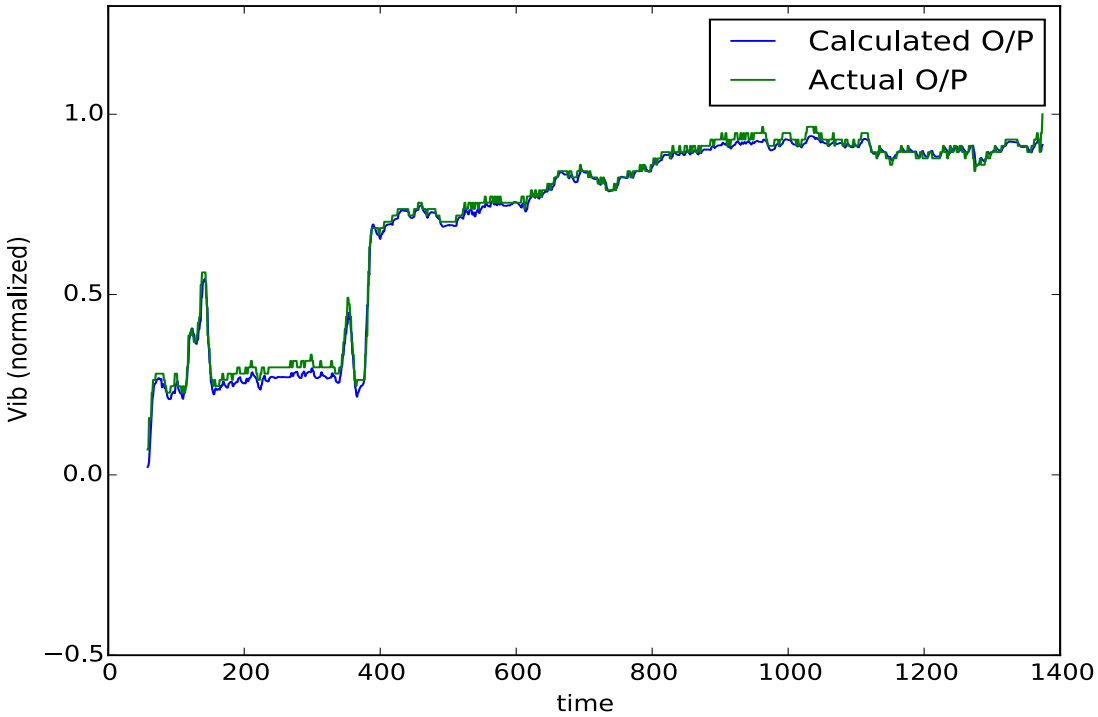


(c) Architecture III predicting vibration 1 second in the future.

Figure 23: Plotted results for the three architectures predicting one second in the future.



(a) Architecture I predicting vibration 1 seconds in the future for one flight.



(b) Architecture III predicting vibration 1 second in the future for one flight.

Figure 24: Plotted results for Architecture I & III predicting one second in the future.

Results of Architecture III

Although it was the most computationally expensive and had a chance for deeper learning, its results were not as good as expected, as shown in Figure 22. The results of this architecture in Table 6 show that the prediction accuracy for this architecture was less than the more simple Architecture I. As this came counter to the predictions for deeper learning, this opens door for investigating about the deeper learning for this problem; this LSTM RNN was one layer deeper and also had 20 seconds memory from the past which was not available for the other two LSTM RNNs used. It is also realized that the overall error in Table 6 for the prediction at 20 future seconds came relatively high. Looking at Figure 22c between time 10,000-15,0000, 20,000-25,000 and 35,000-40,000, it can be seen that the calculated curve got very much higher than the actual vibration curve. This strange behaviour is unique as it can be seen that the calculated vibration would rarely exceed the actual vibration for all the curves plotted for all the architectures at all scenarios, and it would be for relatively small value if occurred. Ideally, This network could potentially gain further improvement if trained for more epochs over the other simpler architectures since it is deeper. This was tried, giving the neural network about double the number of training epochs. However, a significant improvement in the prediction was not achieved. Nonetheless, it was realized that the plots of the cost function of this architecture was not smooth while trained to predict for 20 seconds in the future. This is thought of as a result of under-training. Initially, the training epochs were fixed at 575 for all the architecture as a standard for performance comparison. Further, the performance of this architecture (the mean absolute error) was slightly better than the other architectures when predicting for 1 second in the future. This result supports the believe that this architecture can perform better if given the chance to train for more epochs.

CHAPTER 6

CONCLUSION

This paper presents early work for utilizing long short term memory (LSTM) recurrent neural networks (RNNs) of different types to predict engine vibrations and other critical aviation parameters. The results obtained from this study are very encouraging, given the accuracy of the predictions rather far in the future – 2.84% error for 1 second predictions, 3.3% error for 5 seconds predictions, 5.51% error for 10 seconds predictions, and 10.19% error for 20 seconds predictions. This work opens up many avenues for future work, such as fine tuning the neural network designs and their hyper parameters, changing the design of the layers and/or combine different types of RNNs to further refine the results. Selecting flight parameters also had a great influence on the results. This work could be extended by further investigating the flight parameters and their contributions to the prediction process. This could be achieved by either statistical means or going deeper in the analytical and empirical theories and equations to provide a deeper understanding of the relations between parameters, and thus, more precise future predictions.

Overall, this work provides promising initial work in engine vibration prediction that could be integrated into future warning systems so that pilots can act to prevent excessive vibration events before unfavorable situations happen during flight. Nonetheless, with the availability of an appropriate data set, this work can be slightly modified to train the neural network to learn about engine abnormalities and predict problems before aircraft actually take off.

CHAPTER 7

FUTURE WORK

The use of genetic algorithms is considered as an extension for the work done in this paper to optimize the neural network. Ant Colony Optimization (ACO) will be used to let the LSTM neural network evolve to an optimized structure. Good results obtained from the work done on optimizing RNNs using ACO by T. Desell *et al.* [22] provide a good motive to try the concept on LSTM RNNs. Though LSTM RNNs might be more complicated structure wise, the concept should remain the same. This initial work concentrated primarily on the basic structure of the LSTM RNN's cell *i*.e. M1, M2 shown in Figures 9 and 10 for Architecture I. It would be interesting if a better prediction was achieved after optimizing the research's best architecture.

On the other hand, Architecture III, which has a deeper structure than the others, should be a good candidate for ACO. Although it had the worst prediction in this work susceptibly due to insufficient training iteration, ACO can optimize the connections between the nodes so we take advantage of the deeper learning capability of this architecture in even reasonable number of iterations. Ultimately, ACO can be used on yet deeper architectures to explore deep learning further.

Since the use of GPU did not work for this problem because of the size of the matrices, MPI can be used to run the ACO's ants-picked-neural-networks in parallel to reduce the run-time of the optimization's iterations.

CHAPTER A

ARCHITECTURE I BACK PROPAGATION

LEVEL1:

$$i_{1_t} = Sigmoid(W_i \odot x_t + U_i \odot a_{t-1}) \tag{20}$$

$$f_{1_t} = Sigmoid(W_f \odot x_t + U_f \odot a_{t-1}) \tag{21}$$

$$o_{1_t} = Sigmoid(W_o \odot x_t + U_o \odot a_{t-1}) \tag{22}$$

$$g_{1_t} = Sigmoid(W_g \odot x_t + U_g \odot a_{t-1}) \tag{23}$$

$$m_{1_t} = f_t \odot m_{1_{t-1}} + i1_t \odot g_{1_t} \Rightarrow Sigmoid(m_{1_t}) = mo_{1_t} \tag{24}$$

$$a_t = o_{1_t} \odot mo_{1_t} \tag{25}$$

LEVEL2:

$$i_{2_t} = Sigmoid(T_i \odot a_t + V_i \odot b_{t-1}) \tag{26}$$

$$f_{2_t} = Sigmoid(T_f \odot a_t + V_f \odot b_{t-1}) \tag{27}$$

$$o_{2_t} = Sigmoid(T_o \odot a_t + V_o \odot b_{t-1}) \tag{28}$$

$$g_{2_t} = Sigmoid(T_g \odot a_t + V_g \odot b_{t-1}) \tag{29}$$

$$m_{2_t} = f_t \odot m_{2_{t-1}} + i1_t \odot g_{2_t} \Rightarrow Sigmoid(m_{2_t}) = mo_{2_t} \tag{30}$$

$$b_t = o_{2_t} \odot mo_{2_t} \tag{31}$$

LEVEL3:

$$i_{3_T} = Sigmoid(Y_i \odot b_T + Z_i \odot c_{T-1}) \tag{32}$$

$$f_{3_T} = Sigmoid(Y_f \odot b_T + Z_f \odot c_{T-1}) \tag{33}$$

$$o_{3_T} = Sigmoid(Y_o \odot b_T + Z_o \odot c_{T-1}) \tag{34}$$

$$g_{3_T} = Sigmoid(Y_g \odot b_T + Z_g \odot c_{T-1}) \tag{35}$$

$$m_{3_T} = f_t \odot m_{3_{T-1}} + i1_t \odot g_{3_T} \Rightarrow Sigmoid(m_{3_T}) = mo_{3_T} \tag{36}$$

$$c_t = o_{3_T} \odot mo_{3_T} \tag{37}$$

$$E = \frac{1}{2}(Target - c)^2 \tag{38}$$

$$\delta_1 = \frac{\partial E}{\partial d} = -(Target - c) \tag{39}$$

$$\begin{aligned} \frac{\partial E}{\partial Y_o} &= \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial o_{3_{net}}} \cdot \frac{\partial o_{3_{out}}}{\partial o_{3_{net}}} \cdot \frac{\partial o_{3_{net}}}{\partial Y_o} \\ &= \delta_1 \cdot mo_3 \cdot (1 - o_3) o_3 \cdot b_T \end{aligned} \tag{40}$$

$$\begin{aligned}\frac{\partial E}{\partial Z_o} &= \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial o_{3_{net}}} \cdot \frac{\partial o_{3_{out}}}{\partial o_{3_{net}}} \cdot \frac{\partial o_{3_{net}}}{\partial Z_o} \\ &= \delta_1 \cdot mo_3 \cdot (1 - o_3) o_3 \cdot c_{T-1}\end{aligned} \tag{41}$$

$$\begin{aligned}\frac{\partial E}{\partial Y_f} &= \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial mo_3} \cdot \frac{\partial mo_{3_{out}}}{\partial mo_{3_{net}}} \cdot \frac{\partial mo_{3_net}}{\partial f_3} \cdot \frac{\partial f_{3_{out}}}{\partial f_{3_{net}}} \cdot \frac{\partial f_{3_{net}}}{\partial Y_f} \\ &= \delta_1 \cdot o_3 \cdot (1 - mo_3) mo_3 \cdot m_3 \cdot (1 - f_3) f_3 \cdot b_T\end{aligned} \tag{42}$$

$$\begin{aligned}\frac{\partial E}{\partial Z_f} &= \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial mo_3} \cdot \frac{\partial mo_{3_{out}}}{\partial mo_{3_{net}}} \cdot \frac{\partial mo_{3_{net}}}{\partial f_3} \cdot \frac{\partial f_{3_{out}}}{\partial f_{3_{net}}} \cdot \frac{\partial f_{3_{net}}}{\partial Z_f} \\ &= \delta_1 \cdot o_3 \cdot (1 - mo_3) mo_3 \cdot m_3 \cdot (1 - f_3) f_3 \cdot c_{T-1}\end{aligned} \tag{43}$$

$$\begin{aligned}\frac{\partial E}{\partial Y_i} &= \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial mo_3} \cdot \frac{\partial mo_{3_{out}}}{\partial mo_{3_{net}}} \cdot \frac{\partial mo_{3_{net}}}{\partial i_3} \cdot \frac{\partial i_{3_{out}}}{\partial i_{3_{net}}} \cdot \frac{\partial i_{3_{net}}}{\partial Y_i} \\ &= \delta_1 \cdot o_3 \cdot (1 - mo_3) mo_3 \cdot g_3 \cdot (1 - i_3) i_3 \cdot b_T\end{aligned} \tag{44}$$

$$\begin{aligned}\frac{\partial E}{\partial Z_i} &= \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial mo_3} \cdot \frac{\partial mo_{3_{out}}}{\partial mo_{3_{net}}} \cdot \frac{\partial mo_{3_{net}}}{\partial i_3} \cdot \frac{\partial i_{3_{out}}}{\partial i_{3_{net}}} \cdot \frac{\partial i_{3_{net}}}{\partial Z_i} \\ &= \delta_1 \cdot o_3 \cdot (1 - mo_3) mo_3 \cdot g_3 \cdot (1 - i_3) i_3 \cdot c_{T-1}\end{aligned} \tag{45}$$

$$\begin{aligned}\frac{\partial E}{\partial Y_i} &= \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial mo_3} \cdot \frac{\partial mo_{3_{out}}}{\partial mo_{3_{net}}} \cdot \frac{\partial mo_{3_{net}}}{\partial g_3} \cdot \frac{\partial g_{3_{out}}}{\partial g_{3_{net}}} \cdot \frac{\partial g_{3_{net}}}{\partial Y_g} \\ &= \delta_1 \cdot o_3 \cdot (1 - mo_3) mo_3 \cdot i_3 \cdot (1 - g_3) g_3 \cdot b_T\end{aligned} \tag{46}$$

$$\begin{aligned}\frac{\partial E}{\partial Z_i} &= \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial mo_3} \cdot \frac{\partial mo_{3_{out}}}{\partial mo_{3_{net}}} \cdot \frac{\partial mo_{3_{net}}}{\partial i_3} \cdot \frac{\partial i_{3_{out}}}{\partial i_{3_{net}}} \cdot \frac{\partial i_{3_{net}}}{\partial Z_i} \\ &= \delta_1 \cdot o_3 \cdot (1 - mo_3) mo_3 \cdot i_3 \cdot (1 - g_3) g_3 \cdot c_{T-1}\end{aligned} \tag{47}$$

$$\frac{\partial E}{\partial T_o} = \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial b_T} \cdot \frac{\partial b_T}{\partial T_o} \tag{48}$$

$$\frac{\partial c}{\partial b_T} = \frac{\partial o_3}{\partial b_T} \cdot mo_3 + o_3 \cdot \frac{\partial mo_3}{\partial b_T} \tag{49}$$

$$\frac{\partial o_3}{\partial b_T} = \frac{\partial o_{3_{out}}}{\partial o_{3_{net}}} \cdot \frac{\partial o_{3_{net}}}{\partial b_T} \tag{50}$$

$$\frac{\partial mo_3}{\partial b_T} = \frac{\partial mo_{3_{out}}}{\partial mo_{3_{net}}} \cdot \frac{\partial mo_{3_{net}}}{\partial b_T} \tag{51}$$

$$\frac{\partial o_3}{\partial b_T} = (1 - o_3) o_3 \cdot Y_o \tag{52}$$

$$\frac{\partial mo_3}{\partial b_T} = (1 - mo_3) mo_3 \cdot \frac{\partial mo_{3_{net}}}{\partial b_T} \tag{53}$$

$$\frac{\partial mo_3}{\partial b_T} = \frac{\partial f_3}{\partial b_T} \odot m3_{T-1} + \left[\frac{\partial i_3}{\partial b_T} \odot g_3 + i_3 \odot \frac{\partial g_3}{\partial b_T}\right] \tag{54}$$

$$\begin{aligned}\frac{\partial f_3}{\partial b_T} &= \frac{\partial f_{3_{out}}}{\partial f_{3_{net}}} \cdot \frac{\partial f_{3_{net}}}{\partial b_T} \\ &= (1 - f_3) f_3 \cdot Y_f\end{aligned} \tag{55}$$

$$\begin{aligned}\frac{\partial i_3}{\partial b_T} &= \frac{\partial i_{3_{out}}}{\partial i_{3_{net}}} \cdot \frac{\partial i_{3_{net}}}{\partial b_T} \\ &= (1 - i_3) i_3 \cdot Y_i\end{aligned} \tag{56}$$

$$\begin{aligned}\frac{\partial g_3}{\partial b_T} &= \frac{\partial g_{3_{out}}}{\partial g_{3_{net}}} \cdot \frac{\partial g_{3_{net}}}{\partial b_T} \\ &= (1 - g_3) g_3 \cdot Y_g\end{aligned} \tag{57}$$

$$\boxed{\begin{aligned}\frac{\partial c}{\partial b_T} = (1 - o_3) o_3 \cdot Y_o \cdot mo_3 + o_3 \cdot (1 - mo_3) mo_3 \cdot \\ [(1 - f_3) f_3 \cdot Y_f \odot c_{3_{T-1}} \\ +(1 - i_3) i_3 \cdot Y_i \odot g_3 \\ +i_3 \odot (1 - g_3) \cdot g_3 \cdot Y_g]\end{aligned}} \tag{58}$$

Similarly get the following:

$$\frac{\partial E}{\partial T_i} = \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial b_T} \cdot \frac{\partial b_T}{\partial T_i} \tag{59}$$

$$\frac{\partial E}{\partial T_f} = \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial b_T} \cdot \frac{\partial b_T}{\partial T_f} \tag{60}$$

$$\frac{\partial E}{\partial T_g} = \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial b_T} \cdot \frac{\partial b_T}{\partial T_g} \tag{61}$$

$$\frac{\partial E}{\partial V_o} = \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial b_T} \cdot \frac{\partial b_T}{\partial V_o} \tag{62}$$

$$\frac{\partial E}{\partial V_i} = \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial b_T} \cdot \frac{\partial b_T}{\partial V_i} \tag{63}$$

$$\frac{\partial E}{\partial V_f} = \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial b_T} \cdot \frac{\partial b_T}{\partial V_f} \tag{64}$$

$$\frac{\partial E}{\partial V_g} = \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial b_T} \cdot \frac{\partial b_T}{\partial V_g} \tag{65}$$

$$\frac{\partial b}{\partial T_o} = \sum_{k=0}^{t} \Big(\prod_{j=k+1}^{t} \frac{\partial b_j}{\partial b_{j-1}} \Big) \cdot \frac{\partial b_k}{\partial T_o} \tag{66}$$

$$\frac{\partial b}{\partial b_{\alpha-1}} = \frac{\partial o_{2_\alpha}}{\partial b_{\alpha-1}} \cdot mo_{2_\alpha} + o_{2_\alpha} \cdot (1 - mo_2) mo_2 \cdot \frac{\partial mo_2}{\partial b_{\alpha-1}} \tag{67}$$

$$\frac{\partial o_{2_\alpha}}{\partial b_{\alpha-1}} = (1 - o_{2_\alpha}) o_{2_\alpha} \cdot V_o \tag{68}$$

$$\frac{\partial mo_2}{\partial b_{\alpha-1}} = \frac{\partial f_{2_\alpha}}{\partial b_{\alpha-1}} \cdot m_{2_{T-1}} + \frac{\partial i_{2_\alpha}}{\partial b_{\alpha-1}} \odot g_{2_\alpha} + i_{2_\alpha} \odot \frac{\partial g_{2_\alpha}}{\partial b_{\alpha-1}} \tag{69}$$

$$\frac{\partial i_{2_\alpha}}{\partial b_{\alpha-1}} = (1 - i_{2_\alpha}) i_{2_\alpha} \cdot V_i \tag{70}$$

$$\frac{\partial f_{2_\alpha}}{\partial f_{\alpha-1}} = (1 - f_{2_\alpha}) f_{2_\alpha} \cdot V_f \tag{71}$$

$$\frac{\partial g_{2_\alpha}}{\partial b_{\alpha-1}} = (1 - g_{2_\alpha}) g_{2_\alpha} \cdot V_g \tag{72}$$

$$\boxed{\begin{aligned}\frac{\partial b_\alpha}{\partial b_{\alpha-1}} = (1 - o_{2_\alpha}) o_{2_\alpha} \cdot V_o \cdot mo_{2_\alpha} + o_{2_\alpha} \cdot (1 - mo_{2_\alpha}) mo_{2_\alpha} \cdot \\ [(1 - f_{2_\alpha}) f_{2_\alpha} \cdot Y_f \odot c_{2_{\alpha_{T-1}}} \\ +(1 - i_{2_\alpha}) i_{2_\alpha} \cdot Y_i \odot g_{2_\alpha} \\ +i_{2_\alpha} \odot (1 - g_{2_\alpha}) \cdot g_{2_\alpha} \cdot Y_g]\end{aligned}} \tag{73}$$

$$\frac{\partial b_\beta}{\partial T_o} = \frac{\partial b_\beta}{\partial o_{2_\beta}} \cdot \frac{\partial o_{2_{\beta_{out}}}}{\partial o_{2_{\beta_{net}}}} \cdot \frac{\partial o_{2_{\beta_{net}}}}{\partial T_o} \tag{74}$$

$$\boxed{\frac{\partial b_\beta}{\partial T_o} = mo_{2_\beta} \cdot (1 - o_{2_\beta}) o_{2_\beta} \cdot a_\beta} \tag{75}$$

$$boxed\frac{\partial b_\beta}{\partial V_o} = mo_{2_\beta} \cdot (1 - o_{2_\beta}) o_{2_\beta} \cdot b_{\beta-1} \tag{76}$$

$$\boxed{\frac{\partial b_\beta}{\partial T_f} = o_{2_\beta} \cdot (1 - mo_{2_\beta}) mo_{2_\beta} \cdot m_{2_\beta} \cdot (1 - f_{2_\beta}) f_{2_\beta} \cdot a_\beta} \tag{77}$$

$$\boxed{\frac{\partial b_\beta}{\partial V_f} = o_{2_\beta} \cdot (1 - mo_{2_\beta}) mo_{2_\beta} \cdot m_{2_\beta} \cdot (1 - f_{2_\beta}) f_{2_\beta} \cdot b_{\beta-1}} \tag{78}$$

$$\boxed{\frac{\partial b_\beta}{\partial T_i} = o_{2_\beta} \cdot (1 - mo_{2_\beta}) mo_{2_\beta} \cdot g_{2_\beta} \cdot (1 - i_{2_\beta}) i_{2_\beta} \cdot a_\beta} \tag{79}$$

$$\boxed{\frac{\partial b_\beta}{\partial V_i} = o_{2_\beta} \cdot (1 - mo_{2_\beta}) mo_{2_\beta} \cdot g_{2_\beta} \cdot (1 - i_{2_\beta}) i_{2_\beta} \cdot b_{\beta-1}} \tag{80}$$

$$\boxed{\frac{\partial b_\beta}{\partial T_g} = o_{2_\beta} \cdot (1 - mo_{2_\beta}) mo_{2_\beta} \cdot i_{2_\beta} \cdot (1 - g_{2_\beta}) g_{2_\beta} \cdot a_\beta} \tag{81}$$

$$\boxed{\frac{\partial b_\beta}{\partial V_g} = o_{2_\beta} \cdot (1 - mo_{2_\beta}) mo_{2_\beta} \cdot i_{2_\beta} \cdot (1 - g_{2_\beta}) g_{2_\beta} \cdot b_{\beta-1}} \tag{82}$$

$$\frac{\partial E}{\partial W_o} = \frac{\partial E}{\partial c} \cdot \frac{\partial c}{\partial b_T} \cdot \frac{\partial b_T}{\partial a_t} \cdot \frac{\partial a_t}{\partial W_o} \tag{83}$$

$$\boxed{\begin{aligned} \frac{\partial b}{\partial a_\gamma} = (1 - o_{2_\gamma}) o_{2_\gamma} \cdot T_o \cdot mo_{2_\gamma} + mo_{2_\gamma} \cdot (1 - o_{2_\gamma}) o_{2_\gamma} \cdot \\ [(1 - f_{2_\gamma}) f_{2_\gamma} \cdot T_f \odot c_{2_{\gamma_{T-1}}} \\ + (1 - i_{2_\gamma}) i_{2_\gamma} \cdot T_i \odot g_{2_\gamma} \\ + i_{2_\gamma} \odot (1 - g_{2_\gamma}) \cdot g_{2_\gamma} \cdot T_g] \end{aligned}} \tag{84}$$

$$\frac{\partial a}{\partial W_o} = \sum_{k=0}^{t} \Big(\prod_{j=k+1}^{t} \frac{\partial a_j}{\partial b_{j-1}} \Big) \cdot \frac{\partial a_k}{\partial W_o} \tag{85}$$

$$\boxed{\begin{aligned}\frac{\partial a}{\partial a_{\alpha-1}} = (1-o_{1_\alpha})o_{2_\alpha}\cdot U_o \cdot mo_{1_\alpha} + mo_{1_\alpha}\cdot(1-o_{1_\alpha})o_{1_\alpha}\cdot \\ [(1-f_{1_\alpha})f_{1_\alpha}\cdot U_f \odot c_{1_{\alpha_{T-1}}} \\ +(1-i_{1_\alpha})i_{1_\alpha}\cdot U_i \odot g_{1_\alpha} \\ +i_{1_\alpha}\odot(1-g_{1_\alpha})\cdot g_{1_\alpha}\cdot U_g]\end{aligned}} \tag{86}$$

$$\frac{\partial a_\beta}{\partial W_o} = mo_{1_\beta}\cdot(1-o_{1_\beta})o_{1_\beta}\cdot x_\beta \tag{87}$$

$$\frac{\partial a_\beta}{\partial U_o} = mo_{1_\beta}\cdot(1-o_{1_\beta})o_{1_\beta}\cdot a_{\beta-1} \tag{88}$$

$$\boxed{\frac{\partial a_\beta}{\partial W_f} = o_{1_\beta}\cdot(1-mo_{1_\beta})mo_{1_\beta}\cdot m_{1_\beta}\cdot(1-f_{1_\beta})f_{1_\beta}\cdot x_\beta} \tag{89}$$

$$\boxed{\frac{\partial a_\beta}{\partial U_f} = o_{1_\beta}\cdot(1-mo_{1_\beta})mo_{1_\beta}\cdot m_{1_\beta}\cdot(1-f_{1_\beta})f_{1_\beta}\cdot a_{\beta-1}} \tag{90}$$

$$\boxed{\frac{\partial a_\beta}{\partial W_i} = o_{1_\beta}\cdot(1-mo_{1_\beta})mo_{1_\beta}\cdot g_{1_\beta}\cdot(1-i_{1_\beta})i_{1_\beta}\cdot x_\beta} \tag{91}$$

$$\boxed{\frac{\partial a_\beta}{\partial U_i} = o_{1_\beta}\cdot(1-mo_{1_\beta})mo_{1_\beta}\cdot g_{1_\beta}\cdot(1-i_{1_\beta})i_{1_\beta}\cdot a_{\beta-1}} \tag{92}$$

$$\boxed{\frac{\partial a_\beta}{\partial W_g} = o_{1_\beta}\cdot(1-mo_{1_\beta})mo_{1_\beta}\cdot i_{1_\beta}\cdot(1-g_{1_\beta})g_{1_\beta}\cdot x_\beta} \tag{93}$$

$$\boxed{\frac{\partial a_\beta}{\partial U_g} = o_{1_\beta} \cdot (1 - mo_{1_\beta}) mo_{1_\beta} \cdot i_{1_\beta} \cdot (1 - g_{1_\beta}) g_{1_\beta} \cdot a_{\beta-1}} \tag{94}$$

where (see Figure 8):

t*: previous second*

T*: previous iteration*

i*: input-gate output*

f*: forget-gate output*

o*: output-gate output*

g*: input's sigmoid*

m*: cell-memory output*

w_i*: weights associated with input and input-gate*

u_i*: weights associated with previous output and input-gate*

w_f*: weights associated with input and forget-gate*

u_f*: weights associated with previous output and forget-gate*

w_o: *weights associated with input and output-gate*

u_o: *weights associated with previous output and the output-gate*

w_g: *weights associated with the cell input*

u_g: *weights associated with previous output and the cell input*

BIBLIOGRAPHY

[1] A. V. Srinivasan, "FLUTTER AND RESONANT VIBRATION CHARACTERISTICS OF ENGINE BLADES," 1997. [Online]. Available: http://www.energy.kth.se/compedu/webcompedu/WebHelp

[2] Donahue, Jeffrey and Anne Hendricks, Lisa and Guadarrama, Sergio and Rohrbach, Marcus and Venugopalan, Subhashini and Saenko, Kate and Darrell, Trevor, "Long-term recurrent convolutional networks for visual recognition and description," June 2015.

[3] Linlin Chao et al, "Audio visual emotion recognition with temporal alignment and perception attention," Mar 2016.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in Advances in neural information processing systems*, 2014, pp. 3104–3112.*

[5] A. Nairac, N. Townsend, R. Carr, S. King, P. Cowley, and L. Tarassenko, "A system for the analysis of jet engine vibration data," Integrated Computer-Aided Engineering*, vol. 6, no. 1, pp. 53–66, 1999.*

[6] D. A. Clifton, P. R. Bannister, and L. Tarassenko, "A framework for novelty detection in jet engine vibration data," in Key engineering materials*, vol. 347. Trans Tech Publ, 2007, pp. 305–310.*

[7] C. Chatfield, The analysis of time series: an introduction*. CRC press, 2016.*

[8] *N. A. Boukary, "A comparison of time series forecasting learning algorithms on the task of predicting event timing," Ph.D. dissertation, Royal Military College of Canada, 2016.*

[9] *P. GE, "Box., and g. m, jenkins.,"time series analysis, forecasting and control"," ed: San Francisco, CA: Holden Day, 1970.*

[10] *X. Zhang, Y. Liu, M. Yang, T. Zhang, A. A. Young, and X. Li, "Comparative study of four time series methods in forecasting typhoid fever incidence in china,"* PloS one*, vol. 8, no. 5, p. e63116, 2013.*

[11] *I. Moghram and S. Rahman, "Analysis and evaluation of five short-term load forecasting techniques,"* IEEE Transactions on power systems*, vol. 4, no. 4, pp. 1484–1491, 1989.*

[12] *D. Leverington, "A basic introduction to feedforward backpropagation neural networks,"* Neural Network Basics*, 2012.*

[13] *M. Nielsen, "Neural networks and deep learning." [Online]. Available: http://neuralnetworksanddeeplearning.com*

[14] *K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators,"* Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.*

[15] *S. Hochreiter and J. Schmidhuber, "Long short-term memory,"* Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.*

[16] *Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins, "Learning to Forget: Continual Prediction with LSTM," October 2000.*

[17] *H. Jaeger, "Echo state network,"* Scholarpedia*, vol. 2, no. 9, p. 2330, 2007.*

[18] *D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," DTIC Document, Tech. Rep., 1988.*

[19] *V. N. Ghate and S. V. Dudul, "Cascade neural-network-based fault classifier for three-phase induction motor,"* IEEE Transactions on Industrial Electronics*, vol. 58, no. 5, pp. 1555–1563, 2011.*

[20] *B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in* Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.*

[21] *F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks,"* Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.*

[22] *Travis Desell, Sophine Clachar, James Higgins, and Brandon Wild, "Evolving Deep Recurrent Neural Networks Using Ant Colony Optimization," 2015.*

[23] *——, "Evolving Neural Network Weights for Time-Series Prediction of General Aviation Flight Data," 2014.*

[24] *C. Blum and X. Li., "Swarm intelligence in optimization," 2008.*

[25] *M.DorigoandM.Birattari., "Antcolonyoptimization.In Encyclopedia of Machine Learning," p. 36–39, 2010.*

[26] *M.DorigoandT.Stützle, "Ant colony optimization: overview and recent advances. In Hand book of metaheuristics," p. 227–263, 2010.*

[27] *M. Dorigo and L. M. Gambardella, "Ant colonies for the travelling sales man problem," 1997.*

[28] *K. Socha, "Aco for continuous and mixed-variable optimization. in ant colony optimization and swarm intelligence,"* Future Generation Computer Systems*, pp. 25—-36, 2004.*

[29] *K. Sochaand & M. Dorigo, "Bottom hole pressure estimation using evolved neural networks by real coded ant colony optimization and genetic*

algorithm," Journal of Petroleum Science and Engineering, 77(3):375–385, *2011.*

[30] K. Socha, "Ant colony optimisation for continuous and mixed-variable domains," 2009.

[31] G. Bilchev and I. C. Parmee, "The ant colony metaphor for searching continuous design spaces. In Evolutionary Computing," pp. 25—-39, 1995.

[32] N. Monmarché and G. Venturini and M. Slimane, "On how pachycondyla apicalis ants suggest a new search algorithm," Future Generation Computer Systems, *vol. 16, pp. 937–946, 2000.*

[33] J. Dréo and P. Siarry, "A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions," in International Workshop on Ant Algorithms. *Springer, 2002, pp. 216–221.*

[34] R. Ashena and J. Moghadasi, "Ant colony optimization for continuous domains," European journal of operational research, 185(3):1155–1173, *2008.*

[35] C. Blum and K. Socha, "Training feed-forward neural networks with ant colony optimization: An application to pattern classification," in Fifth International Conference on Hybrid Intelligent Systems (HIS'05). *IEEE, 2005, pp. 6–pp.*

[36] J.-B. Li and Y.-K. Chung, "A novel back-propagation neural network training algorithm designed by an ant colony optimization," 2005.

[37] M. Unal, M. Onat, and A. Bal, "Cellular neural network training by ant colony optimization algorithm," 2010.

[38] R. K. Sivagaminathan and S. Ramakrishnan, "A hybrid approach for feature subset selection using neural networks and ant colony optimization," 2007.

[39]

[40] F. A. Gers, D. Eck, and J. Schmidhuber, Applying LSTM to Time Series Predictable through Time-Window Approaches. *Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 669–676. [Online]. Available: http://dx.doi.org/10.1007/3-540-44668-0_93*

[41] D. Eck and J. Schmidhuber, "A first look at music composition using lstm recurrent neural networks," Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, vol. 103, 2002.*

[42] L. Di Persio and O. Honchar, "Artificial neural networks approach to the forecast of stock market price movements."

[43] N. Maknickienė and A. Maknickas, "Application of neural network for forecasting of exchange rates and forex trading," in The 7th international scientific conference" Business and Management*, 2012, pp. 10–11.*

[44] M. Felder, A. Kaifel, and A. Graves, "Wind power prediction using mixture density recurrent neural networks," in Poster Presentation gehalten auf der European Wind Energy Conference*, 2010.*

[45] E. Choi, M. T. Bahadori, and J. Sun, "Doctor ai: Predicting clinical events via recurrent neural networks," arXiv preprint arXiv:1511.05942*, 2015.*

[46] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," Neurocomputing*, vol. 50, pp. 159–175, 2003.*

[47] J. ürgen Schmidhuber, D. Wierstra, and F. Gomez, "Evolino: Hybrid neuroevolution/optimal linear search for sequence learning."

[48] N. I. Sapankevych and R. Sankar, "Time series prediction using support vector machines: a survey," IEEE Computational Intelligence Magazine*, vol. 4, no. 2, pp. 24–38, 2009.*

[49] *J. P. Lewis, "Fast Normalized Cross-Correlation," 1995.*

[50] *Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions,"* arXiv e-prints, *vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688*