Using LSTM Recurrent Neural Networks to Predict Excess Vibration Events in Aircraft Engines

AbdElRahman ElSaid*, Brandon Wild†, James Higgins†, Travis Desell*
Department of Computer Science*
Department of Aviation†
University of North Dakota
Grand Forks, North Dakota 58202
Email: abdelrahman.elsaid@und.edu, bwild@aero.und.edu, jhiggins@aero.und.edu, tdesell@cs.und.edu

***Abstract*—This paper examines building viable Recurrent Neural Networks (RNN) using Long Short Term Memory (LSTM) neurons to predict aircraft engine vibrations. The model is 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 5, 10 and 20 seconds in the future, with 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.**

I. 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 turbomachinery, 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].

Reference [1] is discussing vibrations generated from engine baldes' fluttering. Engine blades are the engine rotating components that have the largest dimensions among other components. While their rotation 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 that 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 paper 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.

II. Related Work

A. Aircraft Engine Vibration

According to Reference [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 non-integral 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 had 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.

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 runs ahead. The other mode is an 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 had future 2.5 hours successful prediction. 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.

B. *LSTM RNN*

LSTM RNN was first introduced by S. Hochrieter & J. Schmidhuber [7]. 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 to modify the weights of the network. This study paved the way for many interesting projects. LSTM RNN's have been used with strong performance in image recognition [2], audio visual emotion recognition [3], music composition [8] and other areas.

III. Experimental Data

The data used consists of 76 different parameters recorded on the aircraft Flight Data Recorder (FDR) as well as the vibration parameter. A subset of these parameters were chosen based on the likelihood of their contribution to the vibration based on aerodynamics/turbo-machinary background. 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 finally 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

IV. Methodology

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

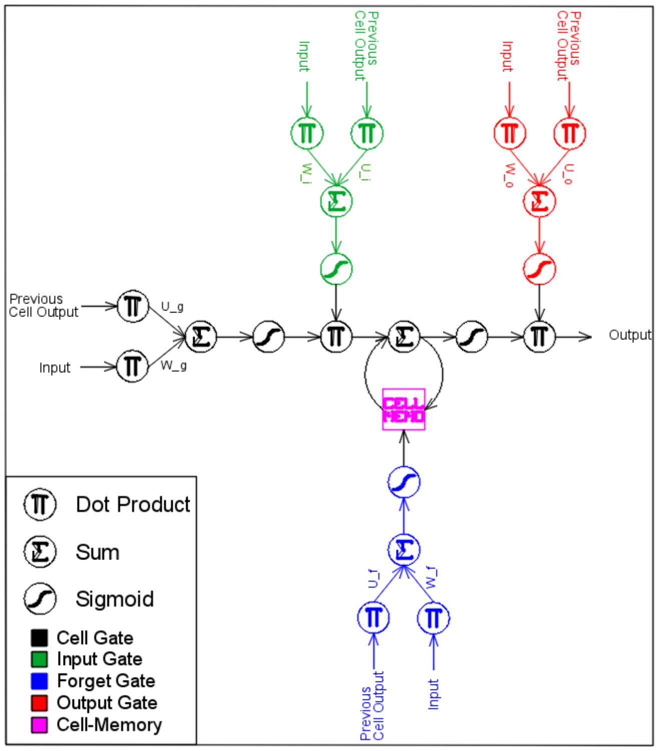


Fig. 1. LSTM cell design

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 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 1. 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 1. However, there are two variations of this common design used in the utilized architectures, shown in Figures 2 and 3, with the difference being the 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.

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

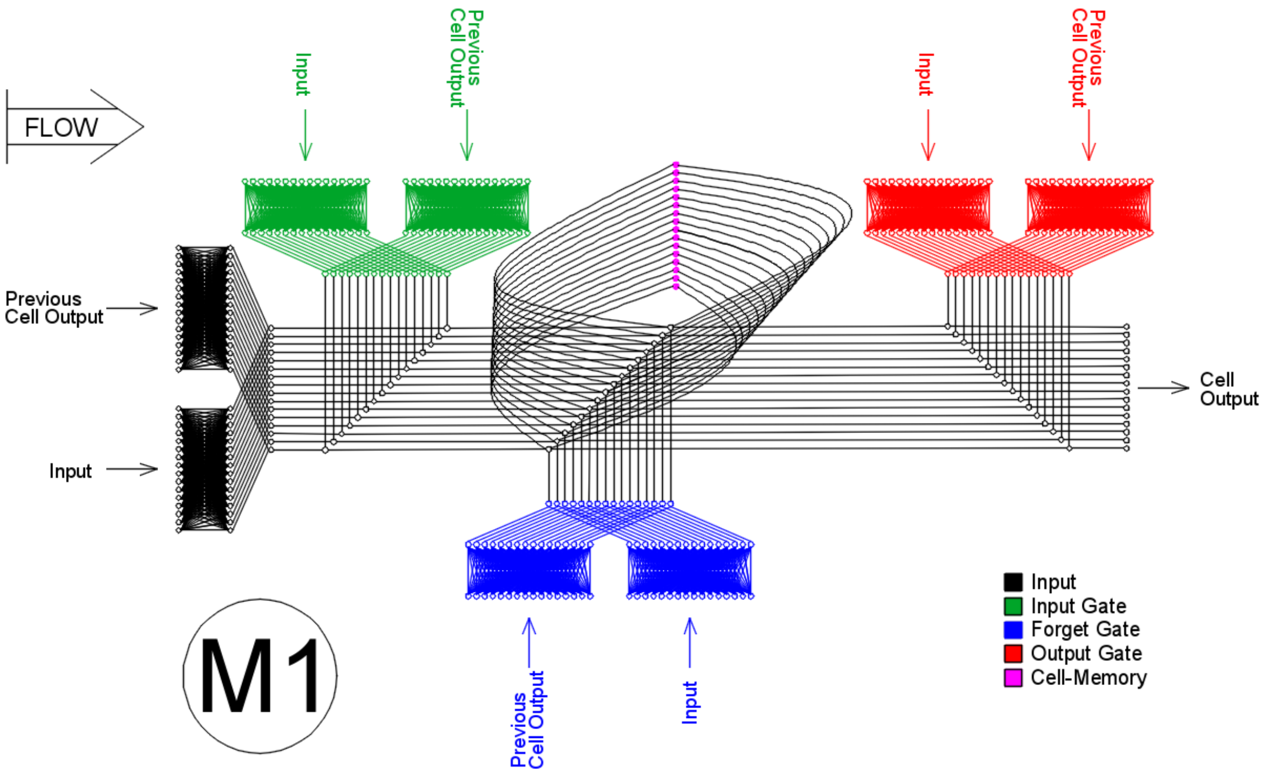


Fig. 2. Level 1 LSTM cell design

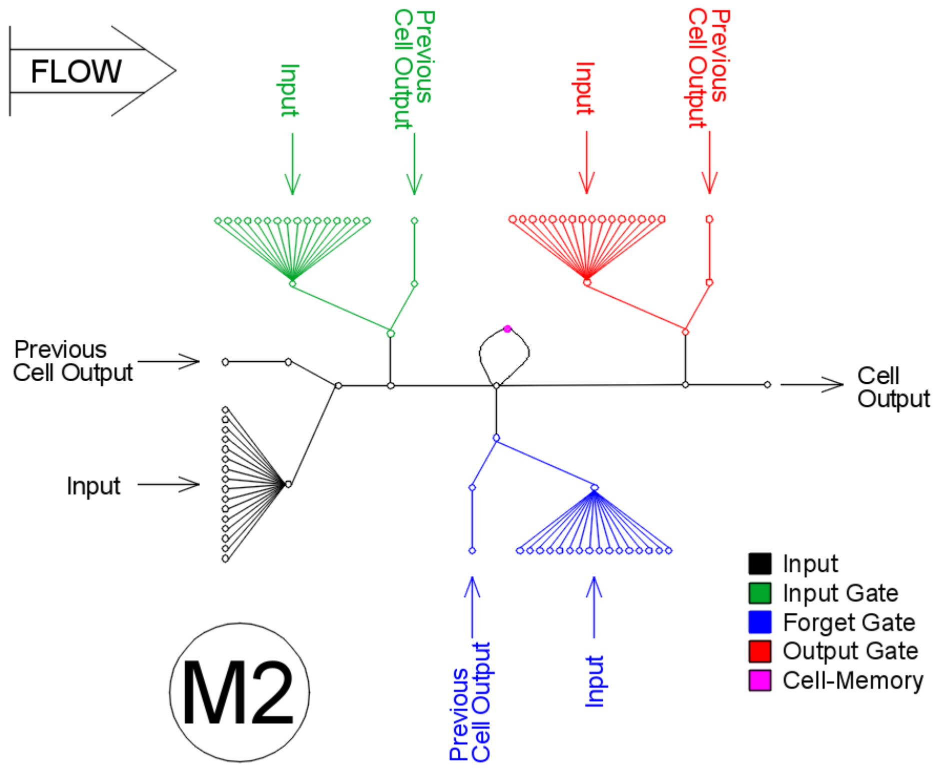


Fig. 3. Level 2 LSTM cell design

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

$$f_t = Sigmoid(w_f \bullet x_t + u_f \bullet a_{t-1} + bais_f) \quad (2)$$

$$o_t = Sigmoid(w_o \bullet x_t + u_o \bullet a_{t-1} + bais_o) \quad (3)$$

$$g_t = Sigmoid(w_g \bullet x_t + u_g \bullet a_{t-1} + bais_g) \quad (4)$$

$$c_t = f_t \bullet c_{t-1} + i_t \bullet g_t \quad (5)$$

$$a_t = o_t \bullet Sigmoid(c_t) \quad (6)$$

where (see Figure 1):

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

V. LSTM RNN Architectures

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

A. Architecture I

As shown in Figure 4, 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.

TABLE I
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 II
Architectures Weights Matrices' Total Elements

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

B. Architecture II

As shown in Figure 5, 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.

C. Architecture III

Figure 6 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 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.

D. 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 4

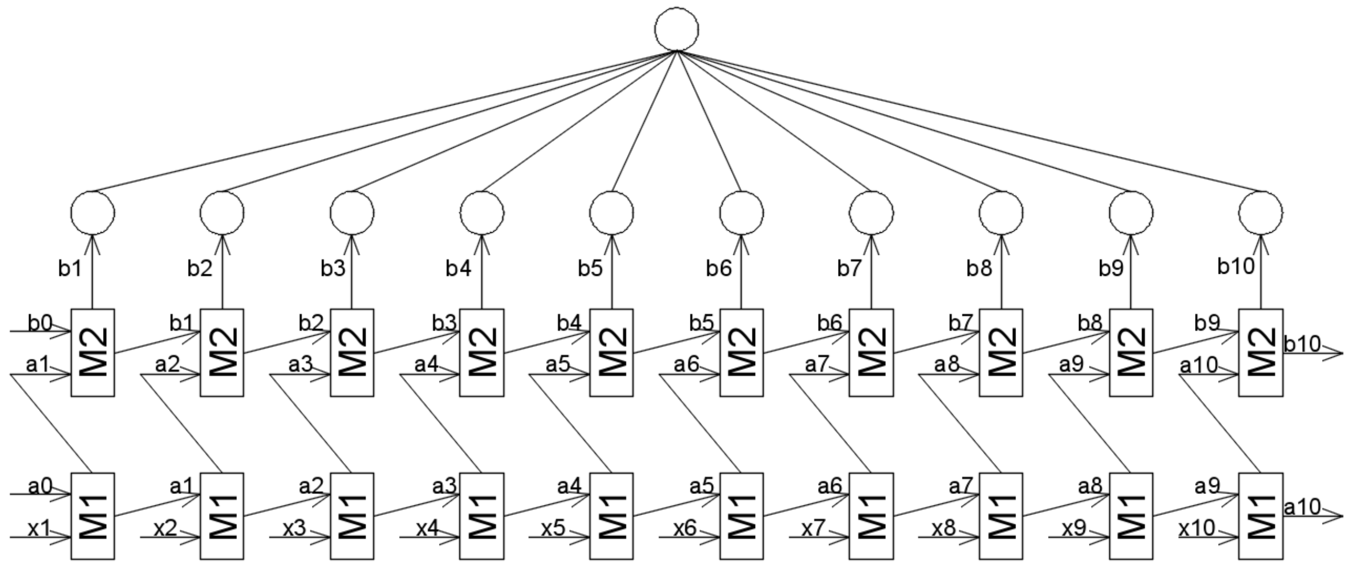


Fig. 4. Architecture I

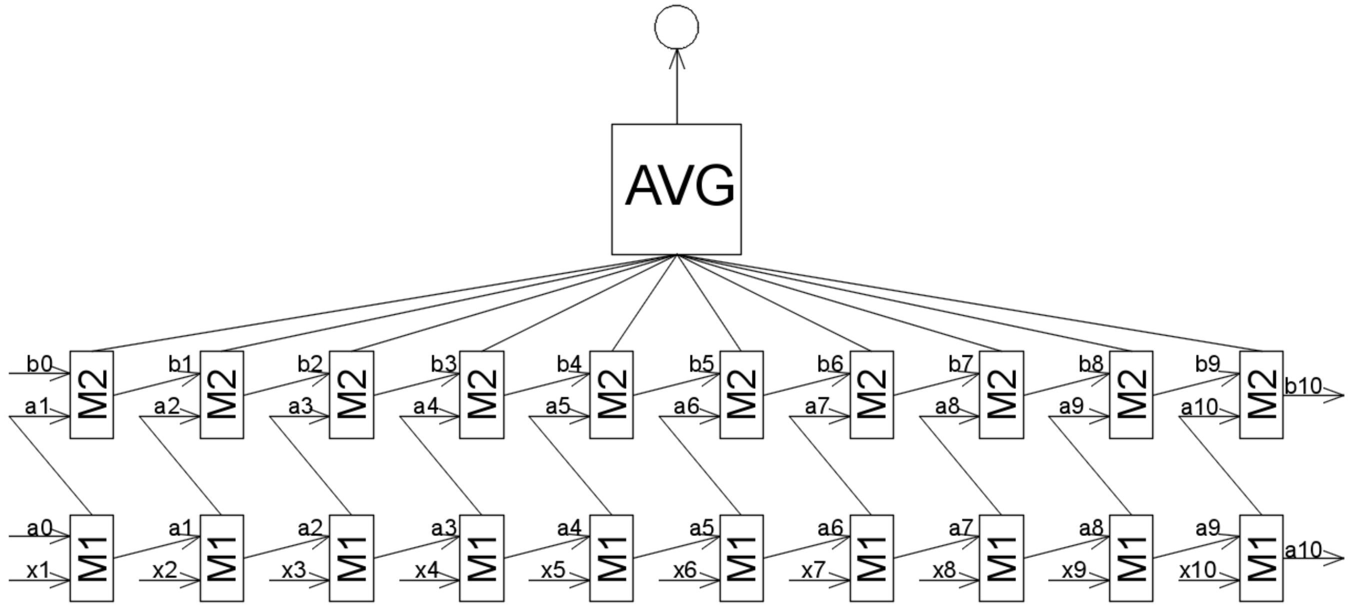


Fig. 5. Architecture II

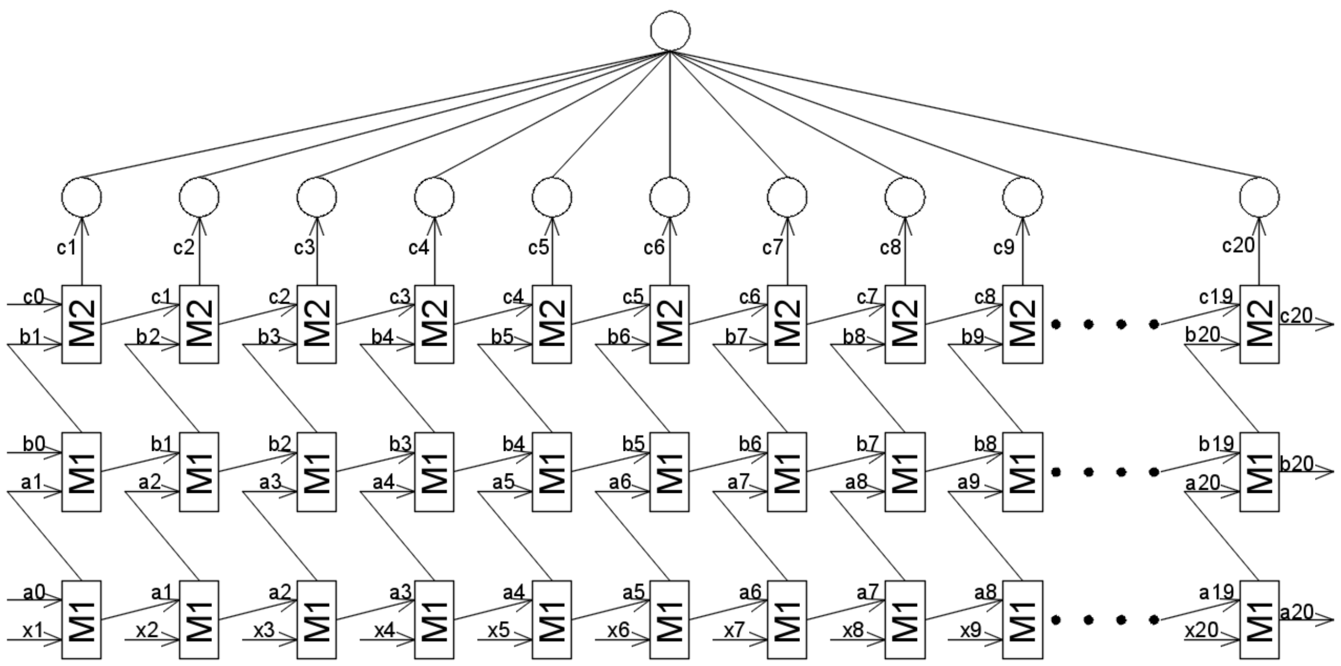


Fig. 6. Architecture III

presenting an overview of the structure of the whole network and considering Figure 2 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 2) in one vector (x_t in Figure 4) and the output of the previous cell (*Previous Cell Output* in Figure 2) in another vector (a_{t-1} in Figure 4). 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 4. 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 1. 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 2. It controls how much of the *cell memory* Figure 4 (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 3. 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 5 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 6. 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 4 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

TABLE III
TRAINING RESULTS

	Error at 5 seconds	Error at 10 seconds	Error at 20 seconds
Architecture I	0.000398	0.000972	0.001843
Architecture II	0.001516	0.001962	0.002870
Architecture III	0.000409	0.000979	0.001717

at the time step.

VI. IMPLEMENTATION

A. Programming Langauge

Python's Theano Library [9] 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.

B. 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.

C. 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 IV. 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.

VII. 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 57 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 flaten.

TABLE IV
RUN TIME (HOURS)

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

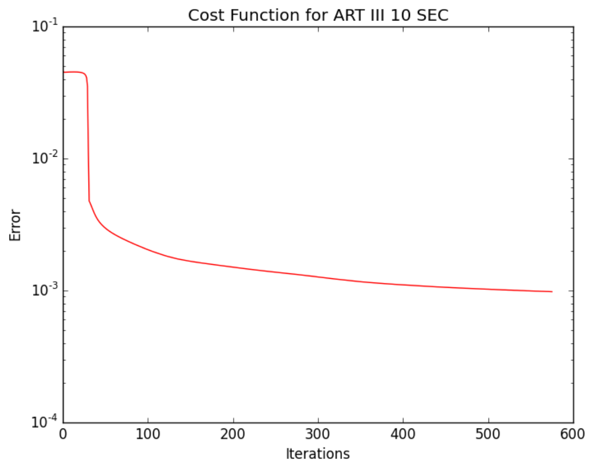


Fig. 7. Cost function plot for ART III predicting vibration in 10 future sec

A. Cost Function

Mean squared error was used to train the neural networks as it provides a smoother optimization surface for backpropagation. A sample of the cost function output can be seen in Figure 7. The Figure is a logarithmic plot for architecture III, for predicting vibrations 10 seconds in the future.

B. Architecture Results

Mean Squared Error (MSE) (shown in Equation 7) was used as an error measure to train the three architectures, which resulted in values shown in Table V. Mean Absolute Error (MAE) (shown in Equation 8) is used as a final measure of accuracy for the three architectures, with results shown in Table VI. As the parameters were normalized between 0 and 1, the MAE is also the percentage error.

$$Error = \frac{0.5 \times \sum(Actual\,Vib - Predicted\,Vib)^2}{Testing\,Seconds} \tag{7}$$

$$Error = \frac{\sum[ABS(Actual\,Vib - Predicted\,Vib)]}{Testing\,Seconds} \tag{8}$$

Figures 9, Figures 10, and Figures 11 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 8 provides an uncompressed example of Architecture

TABLE V
TESTING RESULTS

	Error at 5 seconds	Error at 10 seconds	Error at 20 seconds
Architecture I	0.001165	0.002926	0.010427
Architecture II	0.009708	0.009056	0.012560
Architecture III	0.002386	0.004780	0.041417

TABLE VI
NEW TESTING RESULTS

	Error at 5 seconds	Error at 10 seconds	Error at 20 seconds
Architecture I	0.033048	0.055124	0.101991
Architecture II	0.097588	0.096054	0.112320
Architecture III	0.048056	0.070360	0.202609

I predicting vibration 10 seconds in the future over a single flight from the testing data.

1) Results of Architecture I: The results of this architecture, shown in Table V, 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 9, 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.

2) Results of Architecture II: The results of this architecture in Table V 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 10. 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.

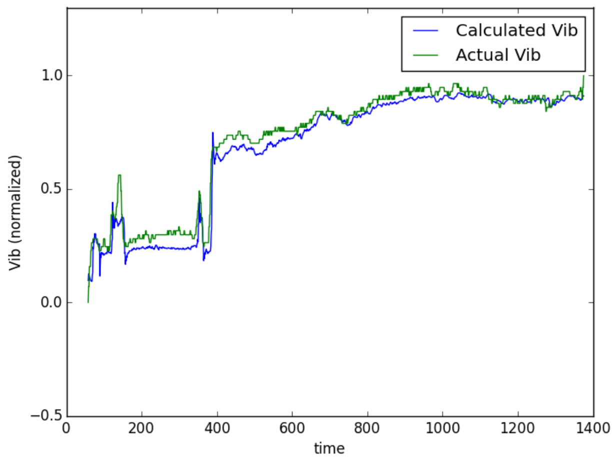
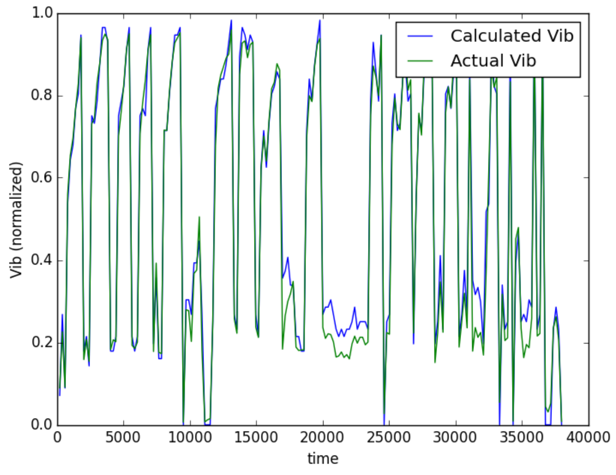
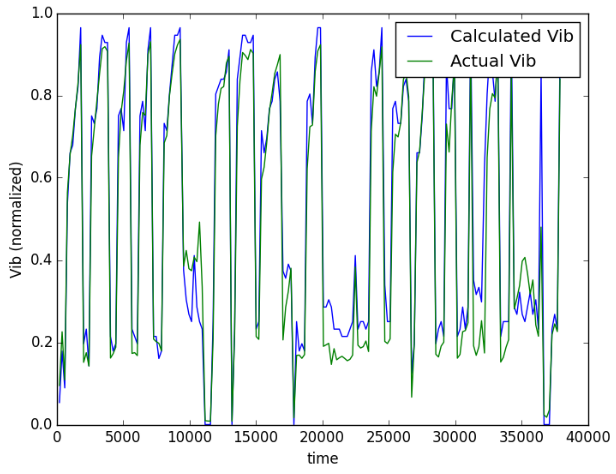


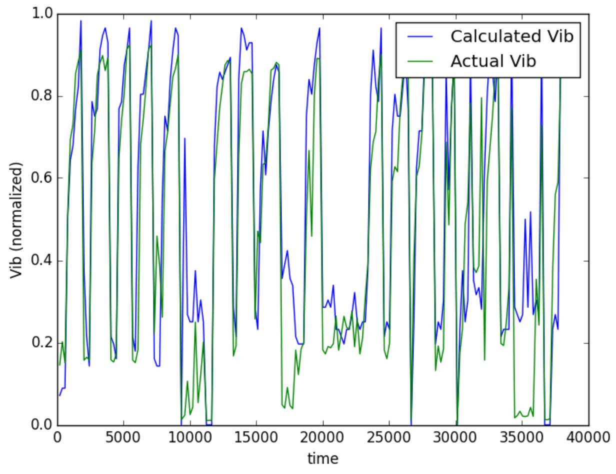
Fig. 8. Architecture I predicting vibration 10 seconds in the future.



(a) ART I Results Plot @ 05 SEC

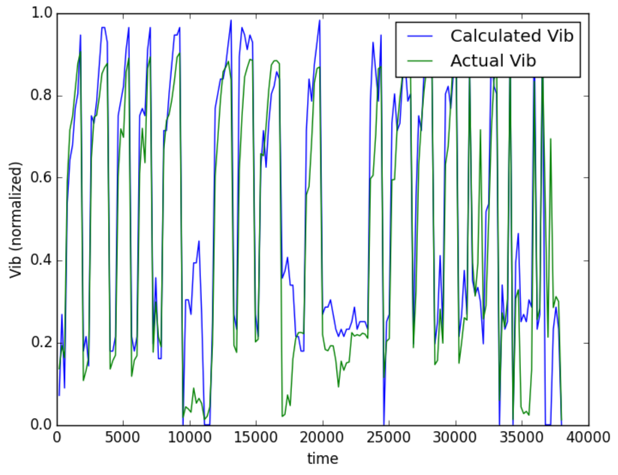


(b) ART I Results Plot @ 10 SEC

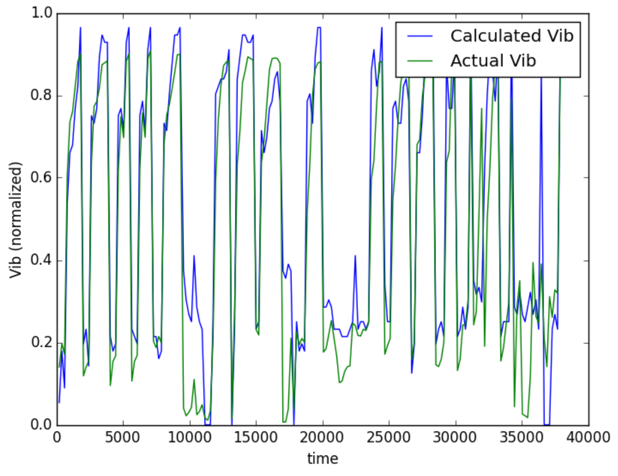


(c) ART I Results Plot @ 20 SEC

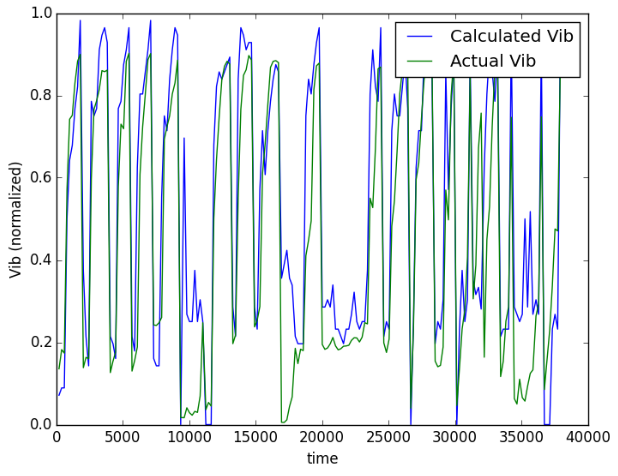
Fig. 9. Plotted results for Architecture I for the for the three scenarios.



(a) ART II Results Plot @ 05 SEC

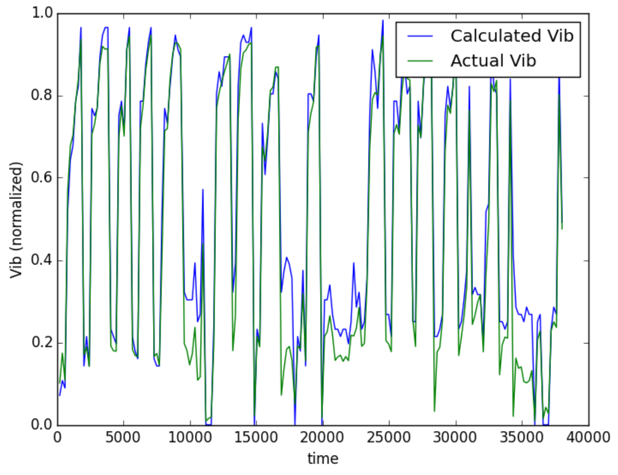


(b) ART II Results Plot @ 10 SEC

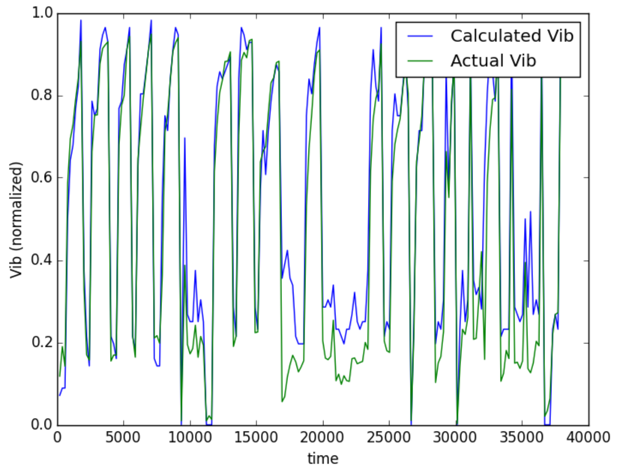


(c) ART II Results Plot @ 20 SEC

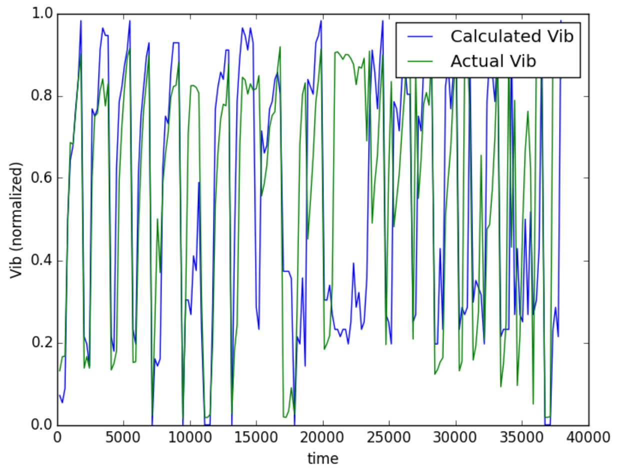
Fig. 10. Plotted results for Architecture II for the for the three scenarios.



(a) ART III Results Plot @ 05 SEC



(b) ART III Results Plot @ 10 SEC



(c) ART III Results Plot @ 20 SEC

Fig. 11. Plotted results for Architecture I for the for the three scenarios.

3) 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 11. The results of this architecture in Table V 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 layers deeper and also had 20 seconds memory from the past which was not available for the other two LSTM RNN's used. It is also realized that the overall error in Table V for the prediction at 20 future seconds came relatively high. Looking at Figure 11c 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. This network could potentially gain further improvement if trained for more epochs over the other simpler architectures.

VIII. Conclusions and Future Work

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 futuer – 3.3% error for 5 second predictions, 5.51% error for 10 second predictions, and 10.19% error for 20 second 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.

Also the use of accelerator cards such as GPUs could be used in this research to further improve training times, and allow the neural networks to be trained longer (which could potentially improve the performance of Architecture III). This can save time if well implemented; as data weights vectors and matrices for all gates (inputs, input gates, forget gates, and output gates) can be grouped together in one matrix/vector saved in one global memory variable to be transfered as on group to the GPU. This would reduce the penalty of data transfer between the CPU and GPU. Similar measures can be followed for processing the data for the several files instead of doing one data file (FDR reading) at a time. Subsequently, processing the data for several future vibration predictions (ex. at 5 sec, 10 sec, 20 sec, ...) could be performed together at the same time, reducing data transfer between CPU and GPU.

Overall, this work provides promising inital 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.

References

[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] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, "Sequence to Sequence Learning with Neural Networks," Dec 2014.

[5] A. NAIRAC et al, "A System for the Analysis of Jet Engine Vibration Data," 1999.

[6] David A. Clifton et al, "A Framework for Novelty Detection in Jet Engine Vibration Data," 2007.

[7] S. Hochrieter & J. Schmidhuber, "Long Short Term Memory."

[8] D. Eck & J. Schmidhuber, "A First Look at Music Composition using LSTM Recurrent Neural Network."

[9] 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