Degradation Estimation of High Energy Steam Piping Using Hybrid Recurrent Neural Networks

by **Johannes Lodewikus van Niekerk**

Submitted in partial fulfilment of the requirements for the degree

Master of Engineering (Mechanical Engineering)

in the

Department of Mechanical and Aeronautical Engineering

Faculty of Engineering, Built Environment and Information Technology

University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

January 2018

Abstract

Degradation Estimation of High Energy Steam Piping Using Hybrid Recurrent Neural Networks

by

Johannes Lodewikus van Niekerk

Supervisors:	Prof. PS Heyns and Dr. M Hindley
University:	University of Pretoria
Degree:	Master of Engineering (Mechanical Engineering)
Keywords:	LSTM, GRU, Recurrent Neural Network, Condition Based Maintenance, High Energy Steam Pipework, Creep Degradation Estimation, Power Plant, Machine Learning, Pipe Elevation Survey.

This dissertation is a study on estimating degradation of high energy steam pipework using modern machine learning techniques. High energy piping systems are very complex to simulate due to the many variables that could influence the useful life of a component. In this research a hybrid recurrent neural network is created that consists of a combined recurrent neural network and a feed forward neural network. The machine learning model is trained on historical data that has been captured over a six-year time period and is applied to a test dataset to see if any usable patterns exist within the training data.

In this research the following variables of the piping system components are used as input to the machine learning model: the operating temperature and pressure time sequence, the distance to the closest anchor point, the distances to neighbouring supports as well as their elevation survey readings and the last known creep damage of the component. The model is created in Python using the Tensorflow library. Two types of recurrent neural networks (RNN) are tested, gated recurrent unit (GRU) and long short term memory (LSTM). The standard gradient descent (GD) algorithm, as well as adaptive gradient descent (ADAGRAD) and adaptive movement estimation (ADAM) are tested. The model was able to predict the classification of a component with an accuracy of up to 91% on the training dataset and 56% on the test data set, which is considered to be high given the complexity of the problem.

The model is successful in recognising patterns within the data and offers an automated way to parse large data sets that consist of a temporal and static data mixture. This offers an approach to make an objective decision on similar complex data driven problems and its application is not constrained to this single problem. The methods applied in this research is expected to perform even better on problems where the frequency of data collection is higher than what is used in this research.

Acknowledgements

The author wishes to express sincere appreciation to the following:

Bilfinger Africa, Mr. Erick van Zyl and Mr. Bhavesh Naran for making the elevation survey data available.

Mr. Ruan van Tonder and Mr. Charl van Tonder for performing the elevation surveys used in this research.

Ms. Jeanine Roelofse for providing assistance with the metallurgical information and assisting with analysis of the results.

Mr. Marthinus Bezuidenhout for managing and providing the applicable outage data.

Dr. Jannie Pretorius for setting up the computation servers.

Mr. Eddie Piater for assisting with software installations and licence management.

Mr. Carel Potgieter and Mr. Manny De Sousa for assistance with data collection software configuration.

Mr. Christiaan Erasmus for overall technical guidance.

The Eskom Power Plant Engineering Institute (EPPEI) for funding this research.

Contents

List of abbreviations

ADAGRAD	Adaptive gradient descent
ADAM	Adaptive movement estimation
ANN	Artificial neural network
API	Application programming interface
CPU	Central processing unit
EP1	Elevation Point 1
EP2	Elevation Point 2
EP3	Elevation Point 3
FFNN	Feed forward neural network
GD	Gradient descent
GPU	Graphics processing unit
GRU	Gated recurrent unit
HP	High pressure
ID	Internal diameter
IPDSS	Intelligent predictive decision support system
LSTM	Long short term memory
NDT	Non destructive testing
NHPP	Non homogeneous poison process
ReLU	Rectified linear unit
RNN	Recurrent neural network
RMS	Root mean square
SIF	Stress intensification factor
TPU	Tensorflow processing unit
WT	Wall thickness

List of symbols

Notation style:

y	Scalar
$\boldsymbol{y}$	Batch of scalars
$\vec{y}$	Vector
$\vec{\boldsymbol{y}}$	Batch of vectors
$\vec{Y}$	Matrix
$\vec{\boldsymbol{Y}}$	Batch of matrices
$\vec{y}^T$	Transposed vector
$\vec{W}_n$	Matrix number n
$\vec{W}_{ij}$	Matrix with row index j and column index j
$\vec{W}_{n,ij}$	The value in matrix number n, row index i and column index j
$\vec{W}_{ij}(t+1)$	The value of matrix with row index i column index j at time step (t+1)

Roman symbols:

a, b, c, d, e, T_a, t_a	Material specific constants
$\vec{a}$	First time sequence vector
$\vec{b}$	Second time sequence vector
$\vec{a}_n$	Neural network hidden layer output vector for layer n
b_n	Constant bias parameter value for layer n
$\vec{b}_n$	Neural network bias parameter vector for layer n
b_f	LSTM forget state bias
b_c	LSTM candidate state bias
b_i	LSTM input state bias
b_o	LSTM output state bias
$\tilde{C}_t$	LSTM vector of candidate cell state values at time t
C_t	LSTM vector of the new cell state
E	Loss or error measurements
f_t	LSTM forget state
$\vec{G_{ij}}$	Matrix with size [i,j], where each element contains the sum of the squares of the past gradients w.r.t. parameter [i,j] up to time step t
$\tilde{h}_t$	GRU candidate hidden state values at time t
h_t	GRU n-dimensional hidden state at time t
ID	Internal diameter
i_t	LSTM input state
K	Number of training examples or the size of the data batch
M	Number of output classes
n	Number of pairs of sequences
o_t	LSTM output state
P	Internal pressure
$P(\sigma)$	Creep rupture parameter

r	Temperature exponent
r_t	GRU reset gate vector at time t
SIF	Stress intensification factor
T	Temperature
$\vec{T}$	Refers to a vector of any input or output features of the model
t	Predicted creep rupture time in hours or time-step in the case of neural networks
$\vec{W}_n$	Neural network weight matrix for layer n
$\vec{W}_{ij}$	Weight value in weight matrix $\vec{W}_n$, connecting input node i to output node j
W_c	LSTM candidate state weights
W_f	LSTM forget state weights
W_i	LSTM input state weights
W_o	LSTM output state weights
wt	Pipe wall thickness
$\vec{x}$	Neural network input vector
$\boldsymbol{y}$	Actual (measured) labels for the data batch
$\boldsymbol{y'}$	Predicted labels for the data batch
y'	Neural network final output/predicted label
z_t	GRU update gate vector at time t
$\vec{z}_n$	Neural network hidden layer input vector for layer n

Greek symbols:

ε	Instantaneous strain
ε_r	Strain at rupture
ϵ	Learning rate
$\vec{\epsilon}$	Modified learning rate for individual parameters
η	Smoothing parameter that ensures division by zero is not possible
λ	Material creep ductility parameter
σ	Material stress

Chapter 1

Introduction

In this chapter the background to the problem as well as the problem statement is defined. The project benefits are also summarized in this chapter. A literature review is conducted on previous work that has been done using machine learning techniques in the asset management space. The literature review also describes how a feed forward neural network as well as a LSTM recurrent neural network works. Different optimisation algorithms are investigated together with different methods of determining the loss fo a model. The damage models that are typically used on these systems to model remaining creep life, are described and a sensitivity analysis is performed on this model to get an idea of the difficulty of the problem at hand.

1.1 Background

The high pressure-steam pipework in a coal-fired power plant experiences fluctuating temperatures and pressure conditions during operation. Replacement of these components is done on a preventative maintenance basis. The condition is monitored throughout its life and replaced once significant creep damage or cracks are observed. Creep damage is observed using surface replica micrographs. Fatigue damage and external loading conditions e.g. pipe support effort are seldom monitored online. Premature replacement of these components does occur. Using traditional non-destructive testing (NDT), it is difficult to determine how much of the damage is done to a component due to pure high-temperature creep and how much is due to fatigue and system loading interaction.

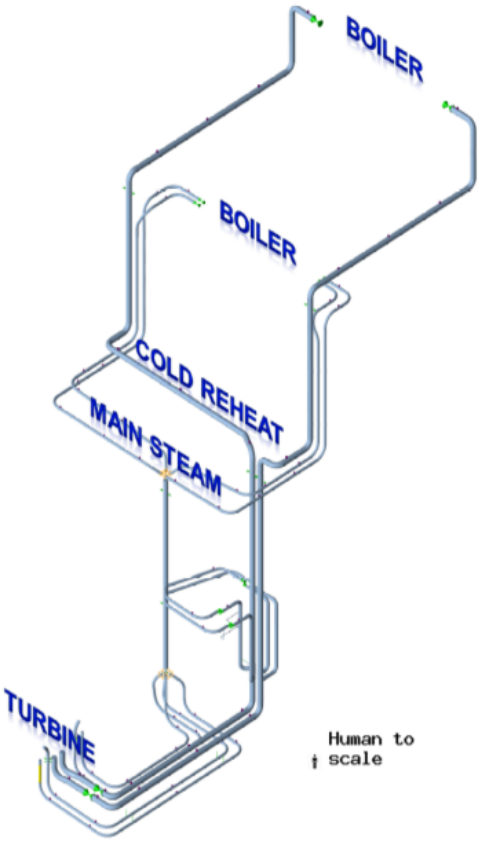
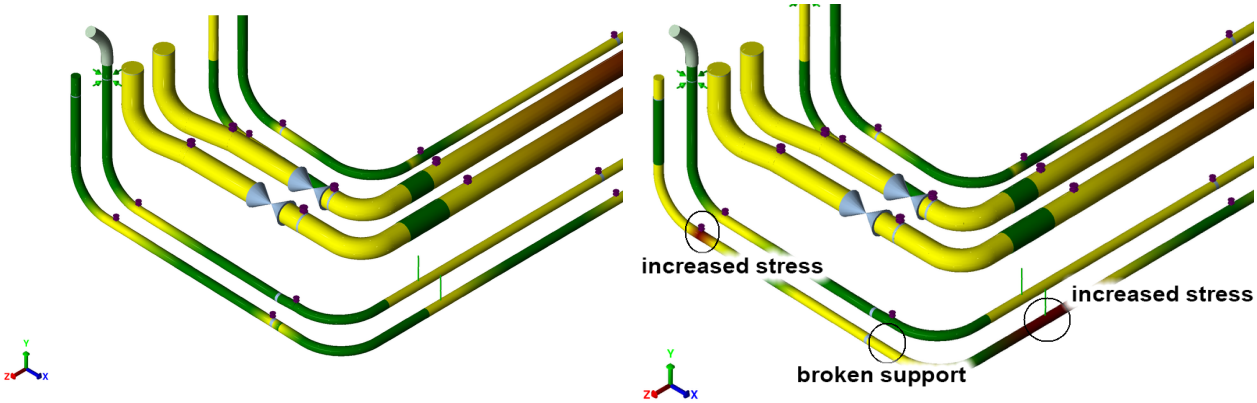


Figure 1.1: Main Steam And Cold Reheat Piping Systems

The steam piping systems in Figure 1.1 are supported from spring hangers, guides and rigid supports that are installed at strategically placed locations. The piping support system is designed to minimise the pipe stresses during operating conditions. Elevation surveys are used to see if the pipework is moving as per its original design, when the piping conditions are changed from atmospheric conditions to operating conditions.

Pipe supports that are not working within their intended design envelope usually cause increased stress on the piping system and directly influences the useful life of the pipework systems that are operating in the finite life temperature range.

Figure 1.2 illustrates how a piping system is typically supported. The two images illustrate the rise in stress when one of the supports is removed. Note that the stress does not only rise in the vicinity of where the support had been. The stress rises significantly at neighboring supports as these supports need to carry the additional load that is no longer supported by the broken support. Obtaining the exact magnitude of the stress is not of interest in this study. The aim is to see if any patterns can be recognized from historical data and to see if premature failures could be linked to the operating conditions (pressure and temperature) as well as supports that are operating outside of its design intent.



(a) Pipe stress with optimal support

(b) Pipe stress with a broken support

Figure 1.2: Influence of broken supports on pipe stress

Replacing components at the end of its usable life is a very tedious process requiring a lot of time and resources. Predicting when components will need to be replaced is notoriously difficult in these systems. Currently there exists no reliable method that can be employed to predict the end of life of a component operating in these systems. Predictions are traditionally made by experienced system engineers who base their inspection and replacement plans on previous inspection results, basic creep life model estimations and past experiences of the system engineer with these systems.

1.2 Problem Statement

High-pressure steam piping is a critical system and replacing the pipework requires long outages, ultimately leading to extended periods of power load loss. It is desirable that a method for determining and forecasting the level of damage in the high energy steam pipework is to be established. Improved remnant life estimates of high-energy piping systems will enhance lifecycle management of piping systems and pipe-replacement interventions can be planned accordingly. There exist very little failure data with these systems since they are normally replaced well before failure occurs as piping components may fail in a catastrophic manner.

Predicting the damage in a high-energy piping component is exceptionally difficult to predict even by the most experienced system engineers. Due to the high number of variables present in piping systems, it is very difficult to make an accurate prediction as to when components will need to be inspected. These variables include: material strength scatter, welding techniques, post weld heat treatment, the geometry of the piping systems, fatigue, creep, corrosion, erosion, vibration, support effort, stress and temperature during operation, thermal expansion stress, etc.

Various online monitoring tools exist that consider multiple aspects of operation (i.e. temperature, pressure, fluid chemistry, crack monitoring, displacement, vibration, corrosion, acoustics, etc.). Despite the effort that has been invested, there exists no standard or unified theory to bring all these methodologies together. Software that calculates the probability of failure of cracked components at high temperature has been developed and applied with good success (Deschanels, Escaravage, Thiry, Le Mat Hamata and Colantoni, 2006). This works well if the cracks are picked up in time with NDT. Due to the high safety risk that is involved with high energy pipework,

defects such as cracks propagating with time are not tolerated and such components are repaired or replaced before returning to service.

The hypothesis of this research is that: *a prediction of remaining life of a piping system can be made by means of pattern recognition on historical experiences.* Machine learning techniques shall be employed to determine if this is a viable method for damage prediction.

A method is needed to parse both the temporal data (temperature and pressure sequences) and static data (collected during outages) and give an objective prediction of the expected damage of a piping component before the component is inspected. The prediction needs to be data driven and based on historical experiences.

1.3 Project Benefits

The cost of loss of revenue during downtime and the cost of inspection and possible replacement of components is very high. The aim of the project is to minimize downtime of plants operating with high steam temperatures and optimizing downtime opportunities. It is not uncommon for pressure part components to operate outside of its intended design envelope due to unforeseen circumstances. This is as a result of pressure part components that are subjected to increased temperature ramp rates, extended operation life, temperatures and pressures outside of design parameters, pipe support malfunction etc. Using the techniques described in this report it would potentially be possible to issue early warnings to the plant operator and would encourage healthy plant operation.

Increasing the ability to accurately predict the damage within a component before inspection occurs will ensure:

- Increase plant operating safety
- Maximize useful life of plant
- Increased availability and reliability
- Minimising operating and maintenance cost
- Managing rate of damage during operation
- Optimizing maintenance opportunities and resources
- Ordering of critical spares in time
- Reducing the probability of expensive repairs

1.4 Contributions of the research

A system can be defined where the inputs of the system are defined as everything that physically defines a component, as well as anything that could give information about external influences on the component that could cause the component to deteriorate. The output of such a system is the life fraction consumed of the component. Traditionally one would have to account for each of the damage mechanisms that could apply to any one components. This would mean that one would need to have a deep understanding of the physics that drives the degradation of the component and

one would need to have a complex mathematical model that accounts for all types of degradation that occurs within a piping component. In the case of a high energy steam piping component, this would mean that one would need to know what damage has been done to a component due to creep, the damage to the component due to fatigue and additionally one would need to know how the two damage mechanisms interact with one another. This requires complex physics based models that becomes difficult to generate when there is allot of damage mechanism working in on a single component. When using the methods described in this research, the physics that happens is regarded to be a black box where you know only the inputs and the outputs of a system. In this research a hybrid network is created that consists of a RNN and FFNN, which will enable the model to parse both temporal and static data at the same time which was historically a very difficult thing to do. If a FFNN was to be used the importance of the static data would be diminished due to the large number of data points that is contained in the temporal sequences. IF a RNN was to be used the Model will be extremely computationally inefficient and specialized computers with vast amounts of memory would be required. As it would be forced to parse the static data with each entry in the temporal sequence.

1.5 Literature Review

1.5.1 Traditional Methods for Life Estimation

There are various traditional methods for modelling statistical mortality rates of components. In the asset management framework traditional methods generally fall under renewable systems modelling or repairable systems modelling.

Although these methods have been applied in the past on high-energy steam piping components, the accuracy of the data fit is often inaccurate and not useful on safety critical components.

Renewable Systems These systems are modelled by a probability density function. This function is approximated by fitting a distribution such as the Weibull distribution through the failure data.

The shortcomings of this method are that it assumes all components are operating under the same conditions and all components are constructed identically. This also assumes that all components are operated from new. Thus, if a component is replaced it will perform as new. This is not accurate in the case of high energy steam pipework due to the fact that when a section of pipework is replaced it is "as good as old" and not "as good as new".

Repairable Systems The non homogeneous Poisson process (NHPP) addresses the issue of "as good as old" replacement as shown in (Coetzee, 1998). However, it still assumes that all components are similar and are operating under similar constant operating conditions.

This method is normally used to optimise maintenance cost of components. Safety is the primary priority on high energy piping systems and not cost. Although cost can be linked to safety risks,

it is not an appropriate method to model the problem at hand.

There are endless statistical models that could be employed, however due to the number of variables in the system and differences between piping systems, the models become very complex and much system specific customisation is required on the model.

1.5.2 Previous work

An intelligent predictive decision support system (IPDSS) model, based on the recurrent neural network (RNN) approach, was developed and tested and run for the critical equipment of a power plant. The results showed that the IPDSS model provided reliable fault diagnosis and strong predictive power for the trend of equipment deterioration (Yam, Tse, Li and Tu, 2001). The predictive decision support system was tested on a planetary gear system of a coal mill, using a normal RNN (6-2-1) construction, where five of the inputs are the last five known measured conditions and the last input is the previous output of the model. The RNN has two hidden nodes and one output node. The model was able to fit to the training data very accurately and the model was able to predict the next conditions very accurately with an RMS-error of 4.834×10^{-5} on a test set of size 45. This enables the user to carry out a condition based maintenance, before a certain condition actually exists in the system.

Creep rupture of austenitic stainless steels have been modelled using neural networks with good success enabling the use of different data sources and employing a neural network as a regression model (Sourmail et al., 2002). The neural network offers a very versatile regression model that can work with a variety of inputs and outputs while being able to generalize well, even with data that contains a large amount of noise. This is part of the reason why neural networks are so popular with data scientists.

A comparative study of elevated temperature creep–fatigue life prediction procedures has shown that the back-propagation neural network technique, when based upon a statistically designed training set, has the potential for achieving superior creep–fatigue life cycle predictions when compared to the modified Coffin–Manson, linear life fraction and hysteresis energy methods (Venkatesh and Rack, 1999).

In other research a fusion network has been applied to vibration data from a rotary device. The model was trained to classify whether the signals are healthy or faulty. The fusion network sends the data from the train feature space to four different neural networks that have different configurations and activation functions. Outputs of the four networks are combined to statistically determine the class of each of the input feature sets. It was shown that the fusion network performs better than any of the individual neural networks used in the fusion network (Ebrahimi et al., 2016).

The artificial neural network (ANN) approach for remaining useful life prediction is a multilayer feed forward network, utilizing both failed and suspension condition monitoring histories, that have been developed for remaining useful life predictions. If a component is taken out of service before failure occurs it is regarded as a suspension history. The model uses age and condition monitoring data as inputs and the life percentage as output. In the case of suspension histories the optimal

predicted life is determined and acts as the training label for the ANN. The research shows that there is a significant decrease in prediction error if the suspension history is included in training the model, instead of using only failure history (Tian et al., 2010).

In summary, FFNNs and RNNs should theoretically be suited for the type of problem described in this dissertation. Neural networks have been used successfully in both prognostic and diagnostic applications. Previous research indicate that there is value in component suspension history when used to train a predictive model and that fusion networks/hybrid networks can be employed to increase the accuracy of predictions.

1.5.3 Feed Forward Neural Network

In the 1940s McCulloch and Pitts did pioneering work in the field. Rosenblatt's perception convergence theorem came about in the 1960s. There was a period of almost 20 years where there was little research done in the field partly due to Minsky and Papert's work showing the limitations of a simple perceptron. Since the 1980s there has been various discoveries in the field that has renewed interest with researchers in the field (Mao and Jain, 1996).

A FFNN neural network consists of an input layers, hidden layers and output layers. where all connections are forward feeding and no recurrence or backward feeding is applied. The number of input, hidden and output layers can be customized to fit a specific problem. A simple FFNN that consists of the following is depicted in Figure 1.3:
-One input layer, with two input nodes and one biased parameter
-One hidden layer, with three hidden nodes and one biased parameter
-One output layer, with two output nodes.

The feed forward neural network is a powerful pattern recognition tool as explained by (Duda et al., 2001). There exist many permutations of the feed forward neural network. The feed forward neural network is the basis upon which most artificial neural networks function. The hidden layers as well as the output layers have a non-zero bias node as an input that is connected to all nodes in the layer.

The activation functions can be any one of the functions described in subsection 1.5.5.

Assuming the activation function that is used for forward propagation is tanh, then for a system with I input layers, N hidden layers and M output layers.

$$\vec{z_1} = \vec{x}\vec{W}_1 + \vec{b_1} \tag{1.1}$$

$$\vec{a_1} = \tanh(\vec{z_1}) \tag{1.2}$$

$$\vec{z_2} = \vec{a_1}\vec{W}_2 + \vec{b_2} \tag{1.3}$$

$$\vec{y'} = \text{softmax}_c(\vec{z_2}) \tag{1.4}$$

where $\vec{W}_1 \in R^{I\times N}, \vec{b_1} \in R^N, \vec{W}_2 \in R^{N\times M}$ and $\vec{b_2} \in R^M$. The values of $\vec{b_n}$ are changed during training while b_1 and b_2 are constant values typically chosen as 1.

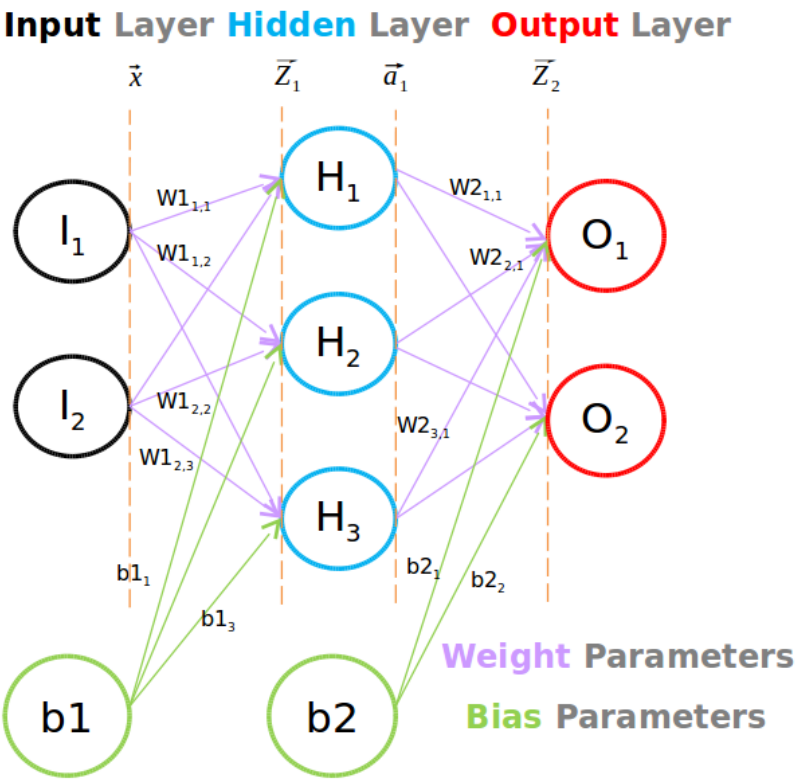


Figure 1.3: Feed forward neural network

1.5.4 Loss Measures

The loss is a measure of the discrepancy between model predictions and actual values. This is also referred to as the error measurement.

For the classification case the neural network is set up to solve classification problems where the result of a given set of input values can be classified in a single output bin, in this case (Figure 1.3) there will be M number of bins.

The predicted output of the neural network is returned as a probability vector that is defined using a *softmax* function. The *softmax* function scales the output vector so that the combined probability of all the output classes is always equal to one. Given a set of input parameters, the classification bin with the highest probability will be taken as the positive class.

There are many ways to determine the loss, however in this research the two loss measures that are used is the cross entropy loss and the root mean squared loss.

Cross Entropy Loss:

The cross entropy loss is also known as the negative log likelihood. The cross entropy loss sums over all the training examples as well as the possible output classes. This is used as an error measure to determine how accurately the predicted values of a data batch ($\vec{y}'$) correlates to actual output values ($\vec{y}$) of the data batch. This loss measure works particularly well with the classification

problems as it sums the error over all classes (De Boer et al., 2005).

$$E = -\frac{1}{K}\sum_{n \epsilon K}\sum_{i \epsilon M} \vec{\boldsymbol{y}}_{n,i} \log(\vec{\boldsymbol{y}}'_{n,i}) \tag{1.5}$$

where K is the size of the data batch. M is the number of output classes and E is the loss or error measurement.

Root Mean Squared Loss:

The RMS loss is an arbitrary error measure that can be used to determine the fitness of a neural network. The value of this error is always positive. This error measure works well with the regression problems where the output is a scalar value.

$$E = \sqrt{\frac{\sum_{n \epsilon K}(\boldsymbol{y}_n - \boldsymbol{y}'_n)^2}{K}} \tag{1.6}$$

1.5.5 Activation Functions

The activation function transforms the inputs of the hidden layer into outputs of the hidden layer. Common choices for activation functions are tanh, the sigmoid function or ReLUs, as shown in Figure 1.4.

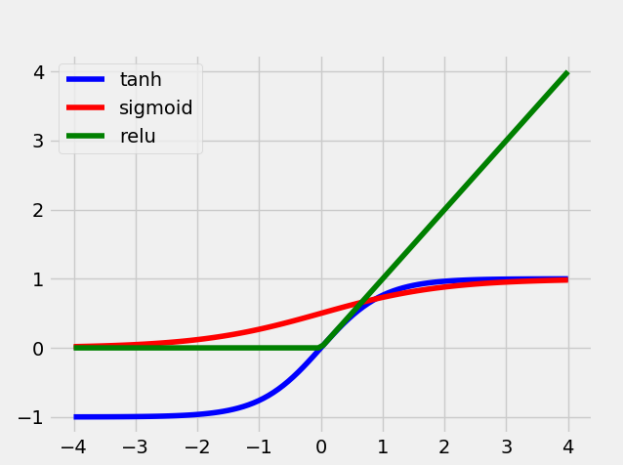


Figure 1.4: Activation function

The logistical sigmoid forces all output values to be between 0 and 1, but has shown empirically that it can easily get *stuck* during training. This is more prone to happen when very large negative numbers are converted by the sigmoid. This causes the outputs to closely approach zero.

The tanh, performs quite well in many scenarios. A property of these functions is that the derivative of $\tanh x$ is $1 - \tanh^2 x$. This is useful because it allows the computation of $\tanh x$ once and re-use

its value later on to get the derivative. This means fewer function evaluations needs to be performed by the model and decreases the training time of the model.

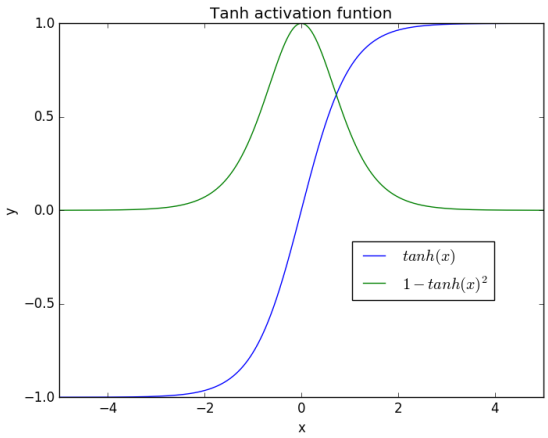


Figure 1.5: $Tanh(x)$ activation function with derivative

The neural network needs to output probabilities in order to classify and output into a certain class. The activation function for the output layer will be the *softmax* with respect to the number of output classes, which converts raw scores to probabilities. The *softmax* function is a normalized exponential function that "squashes" a K-dimensional vector of arbitrary real values to a K-dimensional vector of real values in the range [0,1] that add up to 1.

1.5.6 Back-Propagation

Back-propagation based algorithms are the most widely used algorithms for supervised learning with multi-layered feed forward networks. The basic idea of the back propagation learning algorithm is the repeated application of the chain rule to compute the influence of changes in the parameter values of the network on the value of the error-function E (Mishra and Savarkar, 2012).

$$\frac{\delta E}{\delta \vec{W}_{ij}} = \frac{\delta E}{\delta \vec{a}_i} \frac{\delta \vec{a}_i}{\delta \vec{z}_i} \frac{\delta \vec{z}_i}{\delta \vec{W}_{ij}} \tag{1.7}$$

where $\vec{W}_{ij}$ is the weight between nodes i and j in the network. $\vec{a}_i$ is the output of node i and $\vec{z}_i$ is the weighted sum of the inputs of node i.

Gradient Decent

The steepest descent algorithm is explained by Jacobs (1988). The loss or error where the loss is a measure of how accurately the model's predictions describe the training label values and is calculated as in section 1.5.4.

The gradient descent needs the gradients (Jacobian vector), of the loss function with respect to parameters of the machine learning model, as an input. Gradient descent refers to the algorithm that decreases function value of the error (loss) by evaluating the gradient of the error with respect

to each of the weights in the neural network. Each of the weights is adjusted proportionally to the specified learning rate (ϵ) in the direction of descending gradient. This should theoretically result in a decreased overall error as long as the *learning rate* is not too large, so that the minimum error value is overshot. If the learning rate is too small the model may take too long to converge and could easily get stuck in a local minimum error point.

Thus in order to make sensible changes to the model parameters the following partial derivatives need to be calculated:

$$\frac{\delta E}{\delta \vec{W}_1}; \quad \frac{\delta E}{\delta \vec{b}_1}; \quad \frac{\delta E}{\delta \vec{W}_2} \text{ and } \quad \frac{\delta E}{\delta \vec{b}_2}$$

Jacobs (1988) illustrates that the partial derivatives of the error function are:

$$\frac{\delta E}{\delta \vec{W}_2} = \vec{a}_1^T \vec{\delta}_3 \tag{1.8}$$

$$\frac{\delta E}{\delta \vec{b}_2} = \vec{\delta}_3 \tag{1.9}$$

$$\frac{\delta E}{\delta \vec{W}_1} = \vec{x}^T \vec{\delta}_2 \tag{1.10}$$

$$\frac{\delta E}{\delta \vec{b}_1} = \vec{\delta}_2 \tag{1.11}$$

where:

$$\vec{\delta}_3 = \vec{y}' - \vec{y} \tag{1.12}$$

$$\vec{\delta}_2 = (1 - \tanh^2(\vec{z}_1))\vec{\delta}_3 \vec{W}_2^T \tag{1.13}$$

Note that $\vec{\delta}_2$ is a function of the activation function being used.

Back-Propagation Weight Updates:

Gradient descent is performed by adjusting the weights after each training run of the network. Equation 1.14 and Equation 1.15 are used to update each of the elements in the weight matrix. Due to the linearity of the updating operation, this can be done with pure matrix operations which are efficient and probably the reason why this algorithm is one of the most widely used algorithms.

$$\vec{W}_{ij}(t+1) = \vec{W}_{ij}(t) - \epsilon \frac{\delta E}{\delta \vec{W}_{ij}}(t) \tag{1.14}$$

The bias parameter vector is separately updated using equation 1.15:

$$\vec{b}_i(t+1) = \vec{b}_i(t) - \epsilon \frac{\delta E}{\delta \vec{b}_i}(t) \tag{1.15}$$

where ϵ is a constant learning rate.

Normal back propagation follows the route of steepest descent to find the minimum of the error function. There exist problems where following the steepest descent is not the shortest path to

minimise the error function. This can be due to the fact that learning rate does not change when the direction of gradient descent changes, or when in the vicinity of a local minimum of the error surface. Using this method the local minima points are often "overshot" and this leads to a high number of epochs needed to optimise the neural network.

When the learning rate (ϵ) is too high the neural network may become unstable and an increase in the classification error can be observed with each additional epoch.

Adaptive Gradient Descent (ADAGRAD):

An adaptive sub-gradient method is proposed in (Duchi et al., 2011). ADAGRAD is well-suited for sparse data as it adapts the learning rate vector to individual parameters. ADAGRAD performs larger updates for parameters that occur infrequently and smaller updates for parameters that occur frequently. Normal gradient descent uses the same learning rate for all parameters and is kept constant for every iteration. ADAGRAD uses a different learning rate for each of the parameters in the neural network and changes with every time step.

The calculation of the gradient stays the same. The only difference is in the way that the weights are updated. The weight updates are as per Equation 1.16.

Back-Propagation Weight Updates:

$$\vec{W}_{ij}(t+1) = \vec{W}_{ij}(t) - \frac{\vec{\epsilon}}{\sqrt{\vec{G}_{ij}(t) + \eta}} \odot \frac{\delta E}{\delta \vec{W}_{ij}}(t) \tag{1.16}$$

The bias parameter is separately updated using equation 1.17:

$$\vec{b}_i(t+1) = \vec{b}_i(t) - \frac{\vec{\epsilon}}{\sqrt{\vec{G}_{ij}(t) + \eta}} \odot \frac{\delta E}{\delta \vec{b}_i}(t) \tag{1.17}$$

where $\vec{\epsilon}$ is the modified learning rate for every parameter, the learning rate is based on past gradients and is updated with every time step. η is the smoothing parameter that ensures division by zero is not possible, usually in the order of 10^{-9}. $\vec{G}_{ij}$ is a matrix with size [i,j]. Each element contains the sum of the squares of the past gradient with respect to parameter [i,j] up to time step t.

Adam-Optimisation Algorithm

In this research Adam-optimisation algorithm is used since a hybrid model is used where the RNN network receives data much more frequently than the FFNN, this enables different learning rates for the various parameters of the hybrid network. This should theoretical combat the problem where one of the networks in the system over-fits to the data before the other network has been trained properly. Adam-optimisation is similar to ADAGRAD, however Adam-optimisation is theoretically less prone to get stuck in local minima points due to the addition of the first and second moment terms.

Adam is different to the classical stochastic gradient decent in that a learning rate is maintained for each network parameter (weight). Each learning rate is separately adapted as learning unfolds.

Back-Propagation Parameter Updates: The parameters of the model is updated using the following equations.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1.18}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{1.19}$$

where, m_t is the estimate of the first moment(the mean) and v_t is the estimate of the second moment (the uncentered variance) of the gradients. g_t is the derivatives calculated using Equation 1.7.

The bias corrected first and second moment estimates are calculated using:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{1.20}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{1.21}$$

The model parameter, θ, which could be the the weights $\vec{W}$ or the bias parameter weights $\vec{b}$ is updated using:

$$\theta_{t+1} = \theta_t - \frac{\vec{\eta}}{\sqrt{\hat{v}_t + \epsilon}} \odot \hat{m}_t \tag{1.22}$$

The following hyper-parameters were used in this research: η =0.001; $\beta_1 = 0.9$; $\beta_2 = 0.999$ and ϵ = 1e-08.

1.5.7 Recurrent Neural Network

An RNN is a network based on the principle of the feed forward network adapted for application on sequential data. The difference is that the output of step t is used as the input to step $t + 1$. Neural networks are based on the working of a biological brain. The biological brain does not start fresh every second. Concepts are formed over time. Human thoughts have persistence, thus each new word one reads will have an impact on the idea that is formed in a biological brain. Traditional neural networks have a shortcoming in that it lacks the ability to work with sequential/temporal data. Normal RNNs solve this problem, however the RNN cannot distinguish between information that is important to the result and those that is not. The RNN treats all input data as equally important to the output.

The cell (A) takes an input x_t and outputs a value h_t and allows information contained at time t to persist to time $t + 1$. The unrolled network is shown in Figure 1.6.

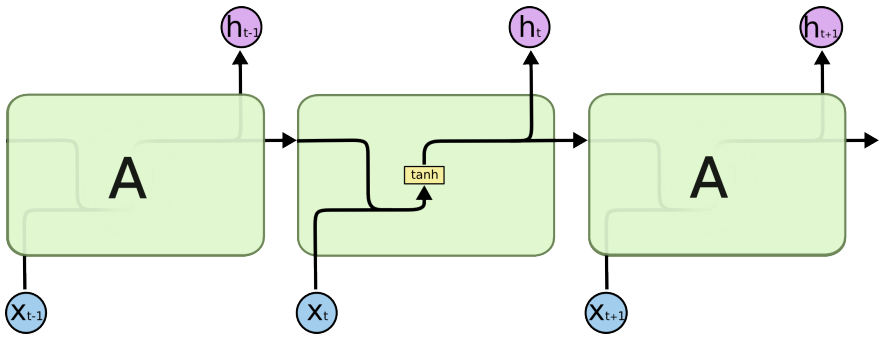


Figure 1.6: RNN-unrolled (Olah, 2015)

RNNs have proven to be ideal for sequential data tasks such as speech and handwriting recognition and generation (Chung et al., 2015). The RRNs however did prove to be very difficult to train due to the vanishing gradient problem and the exploding gradient problem. The exploding gradient problem occurs when very large adjustments are made to the weights of the neural network. This is easily solved by using gradient clipping where the gradient/weight adjustments are capped at a predetermined value. The vanishing gradient problem occurs because the activation function is applied multiple times as time progresses. As an example: an input at x_{t-100} will have passed through the activation function at least 100 times, where x_{t-1} will have passed through the activation function only once. In general the activation function is a *squashing* function hence this diminishes the contribution of x_{t-100}.

One of the appealing features of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous data to give context to the data that is read at the current time step. This is useful in analysing temperature and pressure data.

1.5.8 LSTM Networks

Bengio, Simard, Frasconi and Member (1994) investigated the problems with the RNN, and discovered that the importance of the current input on the RNN output quickly diminishes within a few steps.

The LSTM solves this problem as it enables input data at a given time step to be *remembered* and used to inform the output results many steps later. The LSTM was proposed by (Hochreiter and Schmidhuber, 1997).

For high energy steam piping operating in the creep regime this is especially important. The rate of increase in pressure and temperature is very important as it will have an influence on the damage that manifests in the material. A thermal excursion that occurred a long time ago might have a big influence on the damage of the material. There therefore exists a need for the model to have a type of memory.

LSTM Architecture

The long short term memory network (LSTM) is a variation on RNN, that gives the ability to learn long term dependencies. In order for a system to have a memory it must be able to write the data into the memory. It must be able to read the data back and it also must be able to forget data that is not important.

Instead of making binary decisions on whether to read, write or forget it makes a probabilistic decision using a sigmoid. The sigmoid is a continuous function that is differentiable and it is possible to do back-propagation across this activation function.

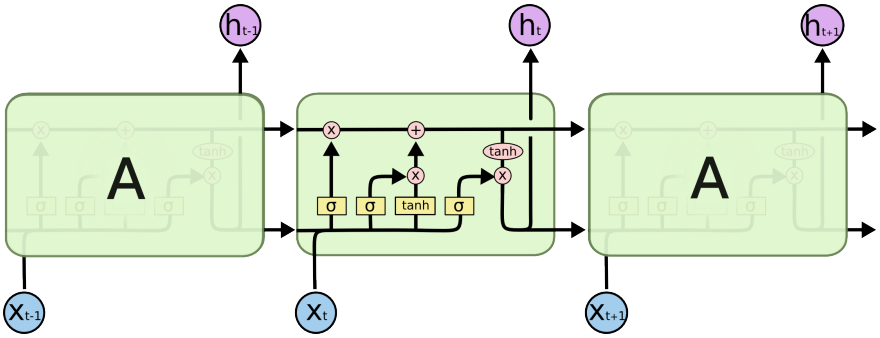
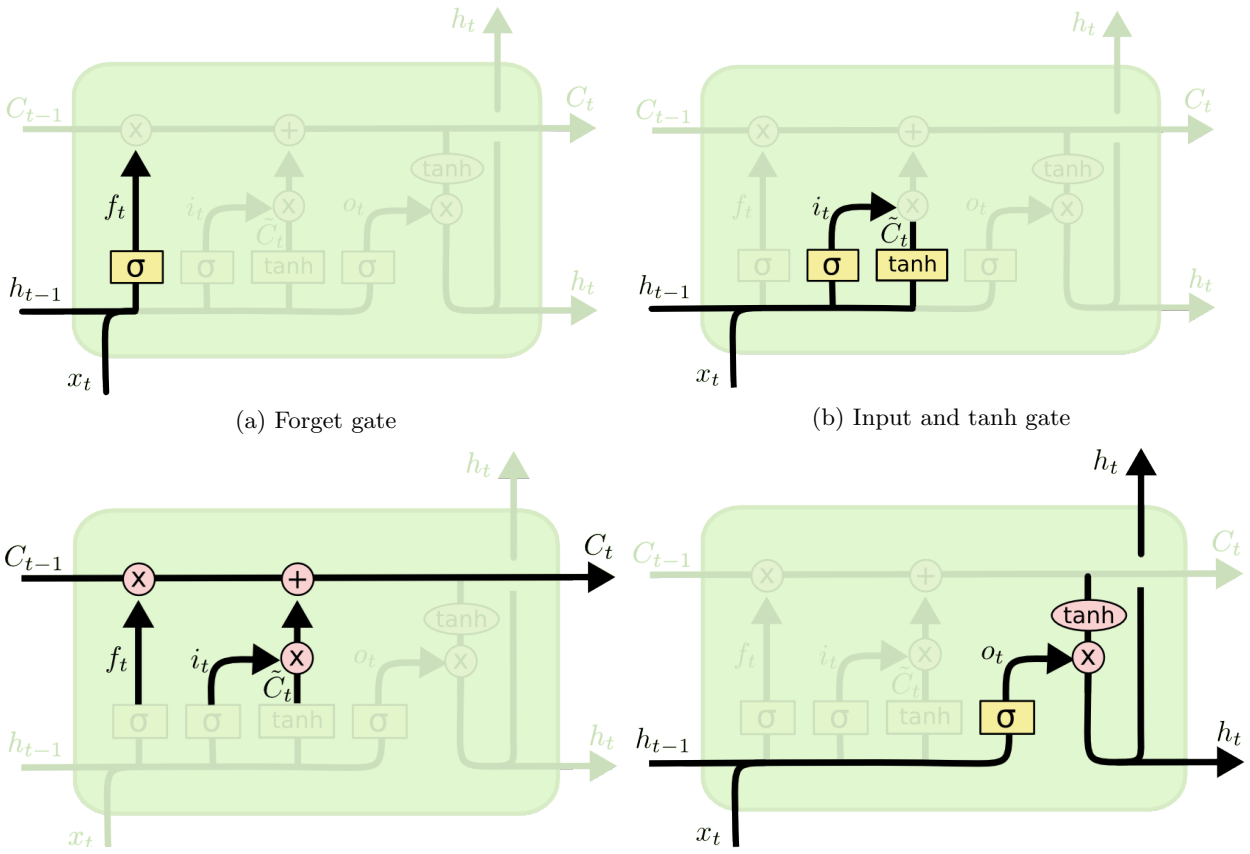


Figure 1.7: LSTM - 3 time steps (Olah, 2015)

In Figure 1.7 each line represents a vector. The LSTM cell takes the input vector (x_t) and concatenates it with the output of the previous step (h_{t-1}). Each of the yellow blocks represents a feed forward neural network as explained in section 1.5.3, three of which use a sigmoid activation function and one that uses a hyperbolic tan activation function. The pink circles indicate pointwise vector operations, vector multiplication and addition. What makes the LSTM unique is that in addition to the output of the previous step the cell state (C_{t-1}) is propagated forward. This allows information to persist many iterations later.

Figure 1.8 shows the individual step operations that happen within a LSTM cell.



(a) Forget gate

(b) Input and tanh gate

(c) Cell state update

(d) Output gate

Figure 1.8: LSTM cell updates (Olah, 2015)

The formulas used in the LSTM cell as shown in Figure 1.8:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1.23}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{1.24}$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{1.25}$$

$$C_t = f_t \times C_{t-1} + i_t \odot \tilde{C}_t \tag{1.26}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{1.27}$$

$$h_t = o_t \odot \tanh(C - t) \tag{1.28}$$

Figure 1.8a shows the forget gate. The sigmoid layer decides how important each element of the input vectors (x_t and h_{t-1}) are to keep. The sigmoid outputs a value between 0 and 1, where 1 represents a completely "keep" and 0 completely "forget".

Figure 1.8b shows two layers. A sigmoid layer that decides which values will be updated next and a hyperbolic tan layer that creates a vector of new candidate cell state values ($\tilde{C}_t$). A multiplication of the two will create an update step to the cell state.

Figure 1.8c shows how the cell state is updated using the outputs of the two previous steps.

Figure 1.8d shows the output gate. The sigmoid layer decides what parts of the cell state is more important to output. The hyperbolic tan layer scales the cell state between -1 and 1. An output is

generated by doing point-wise multiplication of the two layers.

The LSTM actually contains four feed forward neural networks within a single cell. This means that the computational power needed is much more than that of a normal single layer feed forward network.

Gated Recurrent Unit

The Gated Recurrent Unit or GRU introduced by (Cho et al., 2014). This is a simplified version of the LSTM that combines the input gate and the forget gate. The gated algorithm has gained preference from researchers, partly because the Gated Recurrent Unit trains in less time and generally can be trained using less data.

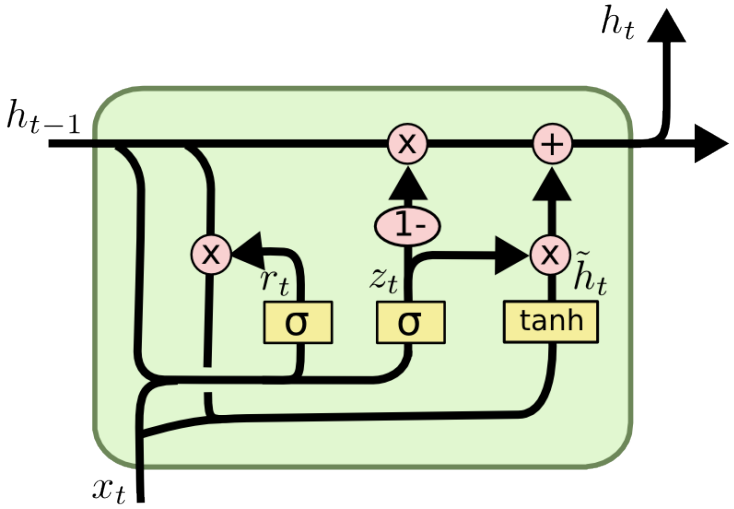


Figure 1.9: GRU cell updates (Olah, 2015)

The formulas used in the GRU cell as shown in Figure 1.9:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{1.29}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{1.30}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t]) \tag{1.31}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{1.32}$$

Theoretically the LSTM should perform better with larger datasets as it has the complexity of two extra gates. Yao et al. (2015) have developed an extension of LSTM to use a depth gate to connect memory cells of adjacent layers. This has proven to outperform GRU and LSTM cells on machine translation and language modelling tasks. This algorithm is not investigated in the current research but will potentially form part of further research.

LSTM and GRU Back propagation:

Due to the increased complexity of the LSTM and GRU cell over the normal FFNN cell, the calculation of the gradient descent algorithm becomes much more involved. The complete derivation of back propagation algorithms is explained in Hochreiter and Schmidhuber (1997). An open source application program interface that can do automatic differentiation, Tensorflow, will be used to do these computations.

1.5.9 Datasets

Data used in this dissertation consist of both temporal and static data. The temporal data consists of the operating temperatures and pressures that the piping system is operated under and is recorded in real time. The static data is measured during major outage events, such as elevation survey data, metallurgical survey data and all attributes of the component that are assumed to stay constant such as geometry and code calculated stress.

Temperature and pressure sequences are used as an input ($\vec{x}$) to the RNN part of the hybrid model because it is long temporal sequences. The output of the RNN is concatenated with the statical input data, which is the elevation survey data, previous metallographic survey results and the component features, to form the input ($\vec{\text{dense input}}$) of the FFNN. The metallurgical survey results will be used as the label ($\vec{y}$) during training.

Machine learning models need a labelled data set to train a supervised learning algorithm. The labels can either be classification labels or regression labels. Classification labels mean that n classification bins are created. One hot encoding is used to specify the correct class. Regression labels mean that the labels/model-outputs are usually in the real value domain, in this dissertation the output shall be a single real value, the metallographic survey result (creep voids per square millimetre).

The only measured parameters that are used in this dissertation shall be temperature, pressure, elevation survey data, metallographic survey results and component constant features that defines physical component such as pipe diameter. This means that there exists plenty of uncertainties with respect to external variables the hat are not measured. The material properties, external loading factors well as manufacturing defects play a very influential role in the life consumption of a component. The dataset is expected to have a considerable amount of *noise*, with large variance between components.

Training, Validation and Test sets:

The data used for performing benchmarking on machine learning algorithms are split into three parts. The first set is used to train the model. The validation set is used as a pseudo test set in order to evaluate the quality of the machine learning model during training. This is done to ensure that the optimisation algorithm is not over fitting the model to the training data. A smaller training error does not always mean a better fit. Early stopping means that a neural network is

trained until a minimum error on the validation set is reached during training.

The training data consists of both the training set and validation set. The validation set is solely used to measure the performance of the model during training on unseen data. This will also give an indication of when the model is over-fitting to the training data. The test data is used to test the model on *unseen* data. In this research the validation and test datasets shall be merged into a single dataset. This means that more data can be used as training data and only one additional data set is needed to test the model performance. The model will not be stopped by the stopping criteria as it would be when employed in operation, but would be left to run until a saturation state is reached. This will indicate whether the model over-fits to the training data if not stopped in time while the performance of the model on a test dataset is still evaluated with each iteration of model training.

Scaling, Normalizing and Standardizing The Datasets:

Real and integer values are rescaled in the range 0 to 1 or -1 to 1. Normalizing and standardizing the data generally makes the gradient descent algorithms more stable. When dealing with multidimensional problems where the scales of the input vectors differ greatly, instabilities could be caused in the gradient descent based algorithms. Where the standard deviation of $\vec{x}$ is small but the mean value of $\vec{x}$ is large it could also cause problems with gradient descent based algorithms (Bishop, 2006).

Standardization of data ensures that the data distribution has a mean of zero and a standard deviations of unity. This particular scaling has the advantage that temperatures above the mean operating temperature (T_μ), which will have the largest contribution to the creep damage, will be transposed into a positive value. Temperatures below T_μ will be transformed into a negative value.

$$\vec{x} = \frac{\vec{T} - T_\mu}{T_\sigma} \tag{1.33}$$

Normalization is a rescaling of the data from the original range so that all values are within the range of 0 to 1. This is also known as min-max scaling.

$$\vec{x} = \frac{\vec{T} - T_{min}}{T_{max} - T_{min}} \tag{1.34}$$

In this report the temperature time signals are used as the input data. The temperature can vary between 0 and 600 $^\circ C$. The temperature data is scaled to ensure that all values will be between 0 and 1.

$$\begin{aligned} \vec{x} &= \frac{\vec{T} - T_{min}}{T_{max} - T_{min}} \\ &= \frac{\vec{T} - 0}{600 - 0} \end{aligned} \tag{1.35}$$

There are many different ways to standardize and normalize the data. In this research the min-max scaling is used throughout, the reason for this is that the means and standard deviations of values between the three different datasets could vary significantly. This could cause the predictions on the test or validation sets to be biased to the training dataset mean.

1.5.10 Learning Curves and Stopping Criteria

The weights are usually initialised using random variables, meaning the initial error is expected to be very high. If a gradient descent back propagation is used, the error is expected to decrease monotonically until it reaches an asymptote that is dependent on the Bayes error (lowest possible prediction error), the amount of training data and the number of weights in the neural network (Duda, Hart and Stork, 2001).

The average errors on the test and validation sets are expected to be higher. While the general trends of errors on these data sets should be downward, the errors can increase and oscillate due to the fact that the neural network can learn the noise in the training data set and it could over-fit to the training data set.

In general, the back-propagation algorithm does not always have a clear point of convergence. When the error surface of the model predictions is considered, there might exist a global minimum point with multiple local minimum points. The training data set can be split into two sets. Say the same machine learning model is applied to the two data sets, when the global minimum of the two data sets are within a certain range of one another and further gradient descent is not possible, there is a high probability that the model has converged to a global minimum. Any further training will make the model biased to the training data and will be over-fitted. The validation set can be used in a stopping criteria in both batch and stochastic protocols. In practice gradient descent learning of the training dataset is stopped or the model parameters at this stage is saved and used as the trained model. In this research the model is only stopped after a maximum epoch limit has been reached. This means the model will continue to train so that we can observe if the model actually over-fits to the training data. When the validation dataset is too small this could cause the model to under or over-train and might cause the model to be bias to the validation dataset. In this research a relatively large validation set is used to ensure that this does not happen.

Figure 1.10 shows the errors of the three data sets during training. In this case the training will stop after 5 epochs. This does not ensure a minimum test error will be obtained. The test set is a blind set and can not be used in the training sequence. The validation set can not be reused as a test set as this data is considered to be tainted and the model will be bias to the training and validation set data.

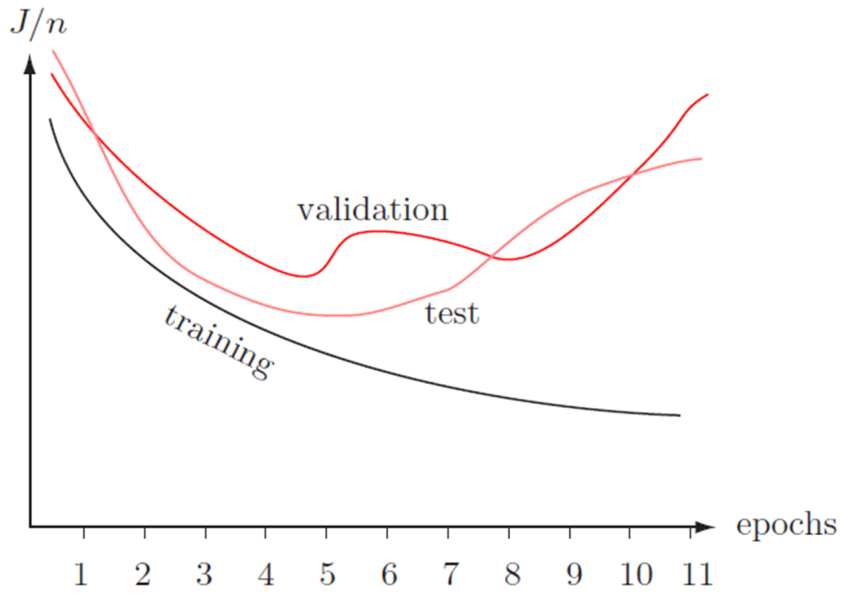


Figure 1.10: Error surfaces (Duda et al., 2001)

Stopping Criteria

Stopping criteria that can be used during training are:

1. The back-propagation algorithm has converged when the Euclidean norm of the gradient vector reaches a threshold that is sufficiently small. The Euclidean norm of a vector E is defined as $\|E\| = \sqrt{E_1^2 + E_2^2 ... E_n^2}$

2. The back-propagation algorithm have converged when the absolute rate of change in error per step/epoch is sufficiently small. This is true when there is no more significant change in error with each additional training step/epoch.

The stopping criteria that are used in this report are only to set a minimum and maximum epoch limit. The reason for this is that during the initial stages of training the obtained errors can be unpredictable and an assumption is made that the optimum solution would not be found within the first few specified runs. Maximum epoch limits are set so that the model does not take excessively long to run. Valuable information can be gained by examining the training curves and looking at what happens after the stopping criteria has been satisfied.

Pseudo code for setting the stopping criteria:

```
    while True:
      training sequence(): # call the training sequence
      if ((epochs > epoch_min) and (abs(E(t)-E(t-1))< threshold)) or (
epochs >= epoch_max):
        break     # stop the training
```

1.5.11 Damage Models

Creep damage in a material is dependent on the material properties, temperature, stress and exposure time. Equation 1.36 describes a well known damage model (Cane, 1982):

$$\frac{t}{t_r} = \left[1 - \left(1 - \frac{\varepsilon}{\varepsilon_r}\right)^{\lambda}\right] \tag{1.36}$$

where t is the operating time, t_r is the operating time at rupture, ε is the strain and λ is the material creep ductility parameter.

In practice the creep damage of a material is measured by counting the number of voids per square millimetre that has formed on the surface of the material. This is done by polishing the material to a mirror finish and then etching the material to show the material grain structure. The number of creep voids per square millimetre is subsequently counted, under a microscope, that has formed on the surface of the material. Creep can be described by three different function categories:

Primary Creep Stage

Numerous small sub-microscopic creep voids with diameters less than 0.1μ initiate as early as 10% of the creep rupture life of a material. Some of these voids become visible under an optical microscope at a magnification of 400 times, when they have grown to a size of $1 \pm 0,5\mu$. The latter will happen typically for low alloy steels where the strain rate is less than $10^{-6}/h$ at life fractions consumed between 0.2 and 0.5 (Van Zyl, Von dem Bongart, Bezuidenhout, Doubell, Havinga, Pegler, Newby and Smit, 2005).

During this stage the metallographic replicas will not yield a lot of information regarding the life fraction used of a material.

Secondary Creep Stage

During this stage the contribution of sub-microscopic voids are insignificant with respect to the strain due to the small volume fraction that it makes out of the material.

Damage models are based on metallographic micro void damage. During outages the metal surfaces

of the high energy piping systems are replicated using a metallographic replica. This is inspected under a microscope with a magnification of 400 times. Voids are counted and quantified as voids per square millimetre. Models are specific to the type of weldment, base material and geometry.

Where the creep strain rate is less than $10^{-6}/h$ the assumption is made that the strain rate is directly proportional to the void formation rate (Van Zyl et al., 2005).

$$\frac{d\epsilon}{dt} = k\frac{dN}{dt} \tag{1.37}$$

The fraction of life used can be written as:

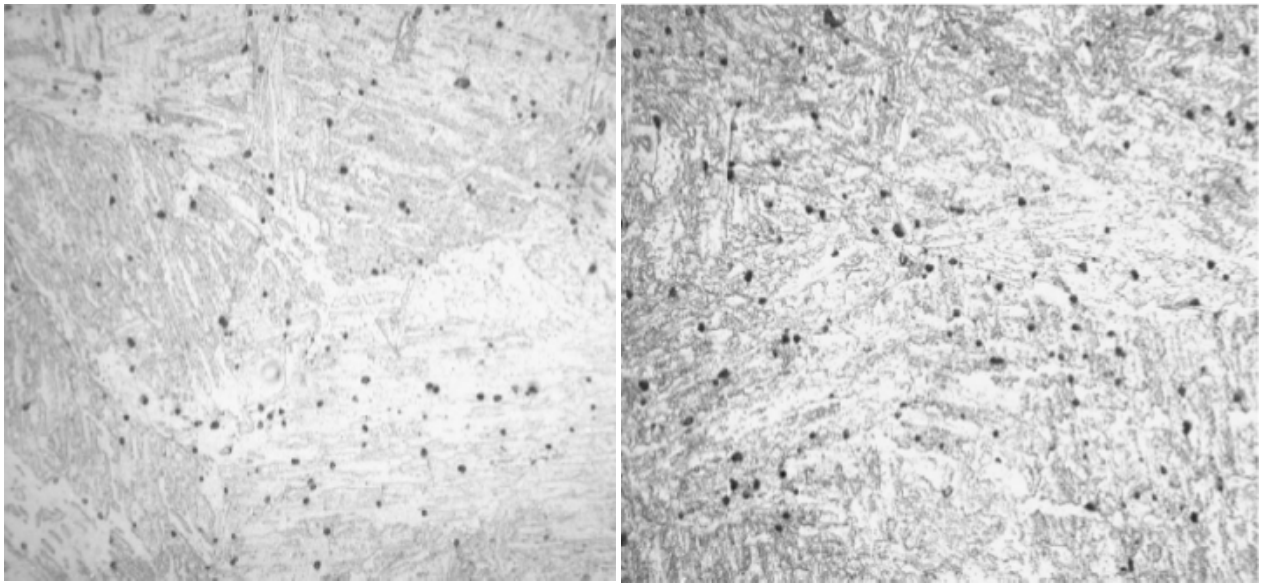
$$\frac{t}{t_r} = \left[1 - \left(1 - \frac{N}{N_f}\right)^{\lambda}\right] \tag{1.38}$$

where N is the number of voids per square millimetre. In the model the number of voids per square millimetre at failure (N_f) is limited to 1000 voids/mm^2 as components are typically not operated beyond this limit. In practice this could evolve to a few thousand voids/mm^2 before micro crack formation and crack development occur. The material creep ductility parameter ($\lambda = 5$) is used in this dissertation as this is a conservative value that has been determined from experience gained in operating multiple units that used the same material over more than 30 years and conducting post exposure tests on failed components.

Using this approach does have some calculation inaccuracies as the value of λ for low fraction life might differ from that of a higher fraction life. It is important to note that there is a direct relation between the number of voids measured (N) and the remaining life fraction of a component $\frac{t}{t_f}$. It would be more convenient to train a machine learning model on the number of voids measured (N), and use the model to predict the number of voids in a component. If the number of voids is known then it is straight forward to calculate the remaining life of the component.

Equation 1.38 is a useful way to monitor the life fraction used of a component throughout its life, by taking metallographic replicas during outages. The life fraction used ($\frac{t}{t_r}$) can be used as the regression or classification label $\vec{y}$ to train a machine learning model. In this dissertation the number of voids shall be used as the label $\vec{y}$. This should make the model less complex and besides normalizing the data, no post processing will be required for the number of voids measured during the outages.

Figure 1.11 shows the material surface under a microscope using 400 times magnification, one taken two years after the other on the same component. The same material grains are not inspected with every outage due to the fact that some material is ground away before each inspection, thus the location of the grain-boundaries and voids will change. Visually one can see that there is an increase in the number and vividness of the voids in the material after two years of operation. During outages the metallographic replicas that could be likened to a fingerprint of the grain-boundaries and voids are sent to a lab where the number of voids per square millimetre are counted for each replica slide. This is recorded and will be used to label the condition of the components in this dissertation.



(a) Creep void damage in bend 2003 (b) Creep void damage in bend reinspected in 2005

Figure 1.11: Creep void degradation

Tertiary Creep Stage

During this stage the number of voids visible at 400 times magnification hardly increase but their size increases rapidly. A material in the tertiary creep state may have low residual creep ductility with resultant constraint cavity growth. For high energy piping systems it is critical to detect creep damage before this stage as strain rates and creep propagation in this region are difficult to predict. The risk of failure is high in this region if proper condition monitoring is not enforced. Components are typically replaced before this stage is reached.

1.5.12 ISO-Mean Life Estimation

The ISO 6303 procedure can be used to approximate the useful life of a component operating at a specified temperature and pressure. In the application of this procedure the following assumptions are made:

- Effects of fatigue damage are negligible.
- No wall loss, erosion or corrosion is present.
- No external forces or moments are imposed on the component.

Equation 1.39 is used to calculate the stress in a straight pipe section. Equation 1.41 is used to calculate the mean creep life of a component, operating at a constant pressure and temperature

$$\sigma_{straight} = \frac{P \times ID}{2 \times wt} \tag{1.39}$$

$$\sigma = \sigma_{straight} \times SIF \tag{1.40}$$

where P is the internal pressure, ID the internal diameter, wt the pipe wall thickness and SIF is the stress intensification factor. SIF = 1, for a straight pipe section.

The life of a component can be determined using the ISO 6303 procedure as described in (ECCC, 2005)

$$P(\sigma) = a + b(\log\sigma) + c(\log\sigma)^2 + d(\log\sigma)^3 + e(\log\sigma)^4 = \frac{\log t - \log t_a}{(T - T_a)^r} \tag{1.41}$$

$$t = 10^{(a+b(\log\sigma)+c(\log\sigma)^2+d(\log\sigma)^3+e(\log\sigma)^4)(T-T_a)^r} t_a \tag{1.42}$$

where t is the predicted creep rupture time in hours, $P(\sigma)$ is the creep rupture parameter and T is the absolute temperature in $[K]$. a, b, c, d, e, T_a and t_a are material specific constants. σ is the stress in $[\frac{N}{mm^2}]$ and r is the temperature exponent.

Table 1.1: Creep life properties used for X20CrMoNiV11-1 (12CrMoV)

a	-0.67429707
b	1.444463833
c	-1.1950249
d	0.43632984
e	-0.0599978034
t_a	13.36
T_a	600
r	1

Plotting the ISO-mean life with properties as specified in Table 1.1, with respect to temperature and pressure yields Figure 1.12. This illustrates that there is an exponential gain in life if the operating temperature or pressure is reduced. This also illustrates how sensitive the life of a component is to the operating temperature.

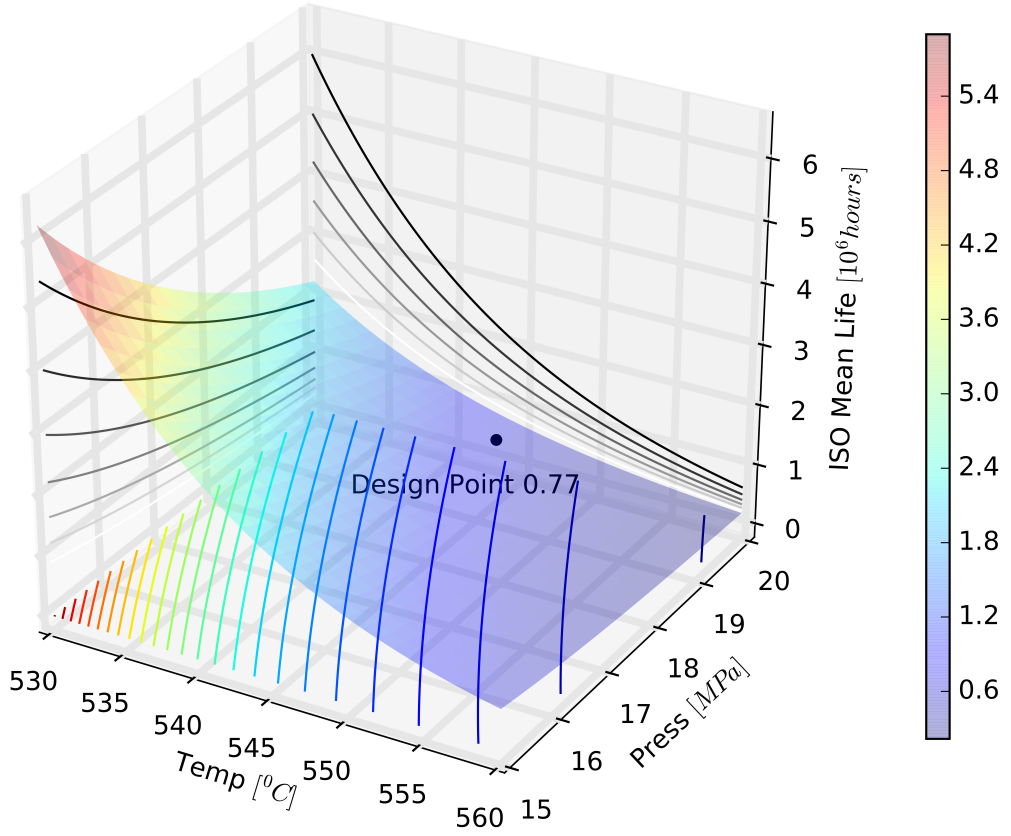


Figure 1.12: ISO mean life estimate [hrs], as function of temperature and pressure

Creep Life Sensitivities

There are a few factors that influence the creep life of a component, the most important are the temperature, stress, and material creep strength. Figure 1.13 illustrates the sensitivity of the expected creep life in hours, by taking uniformly distributed samples within the specified ranges. The illustration is for the case of a straight pipe under internal pressure at a specified constant temperature.

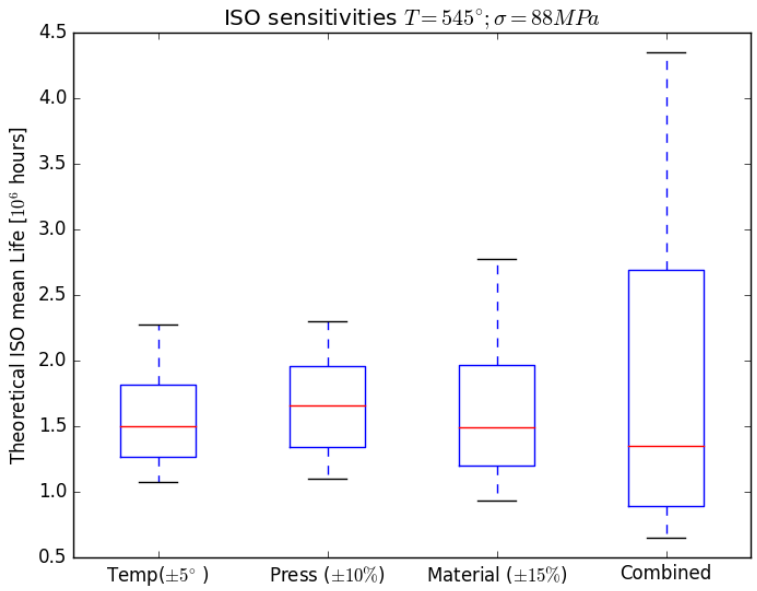


Figure 1.13: ISO-mean creep life sensitivity

Looking at the combined effects of uncertainties in temperature, stress and material properties it is clearly extremely difficult to accurately predict the creep life of a component operating at the specified temperatures and pressures. This means that if two identical components are operated where the temperature differs by $\pm 5°C$, pressure differs by $\pm 10\%$ and the material strength differs by $\pm 15\%$ there will be a big difference in expected life of the component. The difference in component life could range from 700 thousand hours to 4.4 million hours.

1.6 Scope of Research

The research focuses on the main steam piping systems at a coal-fired power station. The aim is to link historical operating and site survey elevation data to pipe degradation. It is envisaged that the feasibility of the machine learning approach could first be investigated in the context of simplified numerical models, and that these models may in future be further developed to a fully functional data driven asset management tool.

A preliminary study is done in Appendix A, to determine if the results of an existing creep life estimation model can be recreated using machine learning. The study is done on randomly generated temperature time series. The results of a creep life estimation model ($\vec{y}$) are used to train a machine learning model. The accuracy of the machine learning model is then tested on a set of new unseen temperature time series data. This is a test of how well the machine learning model mimics the outputs of a creep life estimation model, by only looking at input temperature signals and corresponding calculated life fraction consumed examples, without having any prior knowledge of the life estimation model. The preliminary research is for a straight steam pipe under varying internal temperature and pressure. The damage mechanism is creep and the influence of system loading, fatigue and other damage mechanisms is assumed to be negligible for the preliminary investigation.

The preliminary investigation is done to see if the proposed method can reproduce the results of industry standard methods. Using the ISO mean life estimate method to generate training data one can eliminate uncertainties caused by features that are not available or practicable to measure in a real world scenario for every single component. These features could be material chemical composition, material heat treatment, weld geometry, weld strength, weld material composition, welding defects, post weld heat treatment, external loadings, boiler steam quality etc. By doing this one can have a good idea if there is a problem with the method or if there is too many uncertainties in the data, once a real world dataset is used.

Historical operational (i.e. pressure and temperature), site elevation survey and creep replica micrograph data is gathered and systemically captured in a common database. The data is captured for five coal fired power generating units for a duration of six years. Data quality is assessed and the data shall be *cleaned* in order to make it usable in computer model. This means that all values that has been recorded while the recording instrumentation was broken will have to be padded with a sensible value or would need to be ignored. In the case where data was manually recorded during outages into reports and spreadsheet, the values will need to be transformed into data frames where data. For data such as comments and text based information that cannot be parsed by the model, will be either ignored or encoded in a manner that can be understood by the machine learning model. Using machine learning approaches, data driven models that link historical operational and measured elevation data to the micrograph data (i.e. creep damage) are developed. These models are used for data-driven predictions and decision making. Specific attention is focused on the identification of pipework components that are most susceptible to failure based on historical operational conditions. A hybrid recurrent neural network is constructed that is able to take a mixture of temporal as well as static data and trained on historical data captured of a piping system from a coal-fired power station. Finally, model is tested to see how well it performs on predicting the damage in components when given a new unseen dataset

1.7 Layout of the Document

In this dissertation a machine learning model is developed that will work with the data that is available that has been collected from an operating power plant.

In chapter 2 the development process of the Hybrid RNN model that has been developed using Tensorflow is documented. The layout of the machine learning model is described in this chapter as well as defining the inputs and outputs of the model. This chapter also describes the computational graph that is created in Tensorflow to successfully train the model. The choice of loss function and optimisation algorithms is discussed in this chapter.

In chapter 3 the process that is followed to collect, clean and structure the data is explained. The datasets are segregated into training testing and validation sets. A brief description of the stress analysis that is used as one of the inputs to the model is documented in this chapter. Special requirements, to make computation more effective, such as how the data is stored in Tensorflow records files are discussed.

The results that are obtained by running the same datasets with different model parameters are compared with one another in chapter 4

Appendix A documents an initial investigation where a RNN model is trained on the results of an ISO-mean life curve. The input temperature signals used are generated randomly and the corresponding life fraction consumed is calculated using the ISO-mean life curve. This investigation was done to ensure that when the real data is used that the RNN will be able to recognise patterns in temperature sequences as well as contribute to the hybrid model.

Chapter 2

Machine Learning Models

In this chapter the software packages used are discussed, as well as the special requirements. The hybrid RNN machine learning model layout is described. The loss functions used as well as optimisation algorithms used are also described. An arbitrary method for calculating the accuracy of the model is discussed in this section.

2.1 Prestudy

A pre-study has been performed and is documented in Appendix A. In this pre-study an RNN is trained on a creep damage model. This is done to see how well the model would perform if the model input ($\vec{x}$) consisted purely of a temporal dataset, with life fraction consumed as an output ($\vec{y}$). Artificial noise is added to the data to see if the model is able to "ignore" the noise in the data and recognise the underlying patterns in the data.

The model was able to achieve a training accuracy of up to 93% with a corresponding validation accuracy of 90%. The lessons learned from this pre-study is that LSTMs generally perform better than GRUs on large datasets. GRUs tend to train in less time, due to it being less complex. The pre-study also indicates that it is better to train the problem as a regression problem rather than a classification problem. The reason is that no information is lost when converting the output label ($\vec{y}$) from a real value to a one hot encoding vector. GD and ADAGRAD optimisation algorithms do not converge to the same solutions and ADAGRAD is seemingly more suited to optimize the given model.

The pre-study indicates that RNNs are well suited for dealing with data that has a lot of noise imposed on it and is very successful at "ignoring" the effects of noise on the model predictions. The labels ($\vec{y}$) in this pre-study is a calculated percentage of life fraction consumed. The labels ($\vec{y}$) used in the main study is the number of creep voids per square millimetre.

2.2 Tensorflow

RNNs notoriously have very long training times and are computationally very expensive to train. GPU processing for machine learning application only became practicable in recent years and is mostly the reason for the recent popularity gain of RNNs with data scientists.

Tensorflow is an open source application program interface (API). Tensorflow takes variables and store them in predefined multi-dimensional matrices called tensors. A graph is automatically generated that specifies the "flow of tensors" (i.e. addition and multiplication of tensors). This optimizes the number of parallel operations that can be performed. Graphical Processing Unit (GPU) computation is used as thousands of parallel operations can be performed. This reduces the computational time significantly.

Tensorflow has preprogrammed neural network cells such as LSTM and GRU cells. The API has multiple preprogrammed optimisation algorithms such as gradient descent (GD), adaptive gradient decent (ADAGRAD) and adaptive movement estimation (ADAM), that can perform automatic differentiation by applying the chain-rule recursively.

2.3 Tensorflow Records Files

The raw data used in this study is in Microsoft Excel format. To enable easy data manipulations the data is saved into multiple Python data frames.

The different data frames created is shown in Table 2.1.

Table 2.1: Data frames

Data frame	Description
Tags	A Python script reads all the pressure and temperature data that is collected in monthly excel files into one consolidated data frame for all the units. The data frame is saved in a Tags.pickle file that makes later access easier.
Outage Lists	The outage date lists are saved into a data frame that contains the outage dates and file locations of elevation surveys and outage reports.
Replica Data	A Python script iterates through each of the outages listed in the Outage Lists data frame, and reads the creep damage from corresponding outage report into a replica data, data frame and is saved in a Replica.pickle.
Opp Data	The applicable operational data for each outage is extracted from the Tags dataframe. Saved into Opp data.pickle.

2.4 Tensorflow Graph

Figure 2.1 shows the Tensorflow graph that is constructed to perform the computations. There are three different input producers that read the train, test and validation data respectively from the Tensorflow record files created in section 2.3. The record files are parsed into tensors. Tensors are multidimensional variables, that have a predetermined shape. The Tensorflow graph describes

the flow of tensors i.e. the sequence in which computations will take place. Once the graph is constructed and sequence of computations is established, Tensorflow can perform automatic differentiation using the chain rule which is used during back propagation.

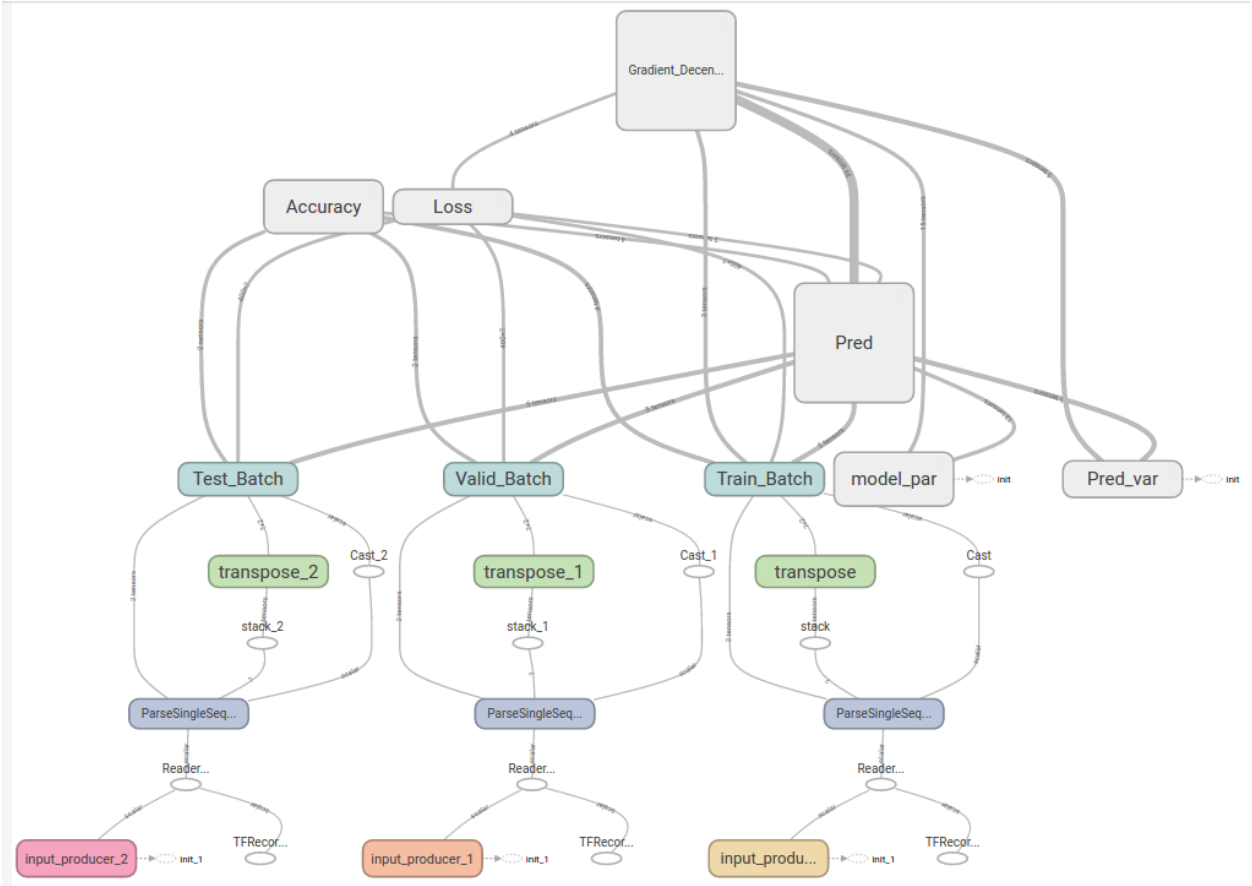


Figure 2.1: Tensorflow graph

For each three dataset four tensors are created [x, y, seqlen, components]

x	-	The temperature and pressure input sequence. A two dimensional float tensor of shape [sequence length, number of features].
y	-	The label, in this case the maximum number of voids experienced by a component, the tensor contains a single float value.
seqlen	-	The length of the sequence is saved in a tensor that contains a single value.
components	-	The discrete information related to each of the components including elevation survey readings, last known maximum number of voids.

Tensorflow batches are created ("batch size = n"). A batch is a collection of multiple inputs. A batch of size n will contain the data for n components. Data are fed into the model as batches to enable parallel computation capabilities.

2.4.1 Training Sequence

Figure 2.2 shows the training part of the Tensorflow graph. It is important to note that the gradient decent algorithm is only dependent on the inputs of the training batch, hence the model is only trained using the training data. Firstly the "Train Batch" is send to the model ("Pred"). The

model makes a prediction on each of the entries in the training batch. The "Loss" is calculated by comparing the predicted labels to the actual labels. The "Loss", which is an error measure, is sent to the gradient descent algorithm that calculates the gradients as per subsection 1.5.6. The gradient-descent algorithm determines the updates that need to be made to the "model parameters" and the model is updated accordingly. The model parameters are saved in order for it to be recalled during the testing and validation sequences.

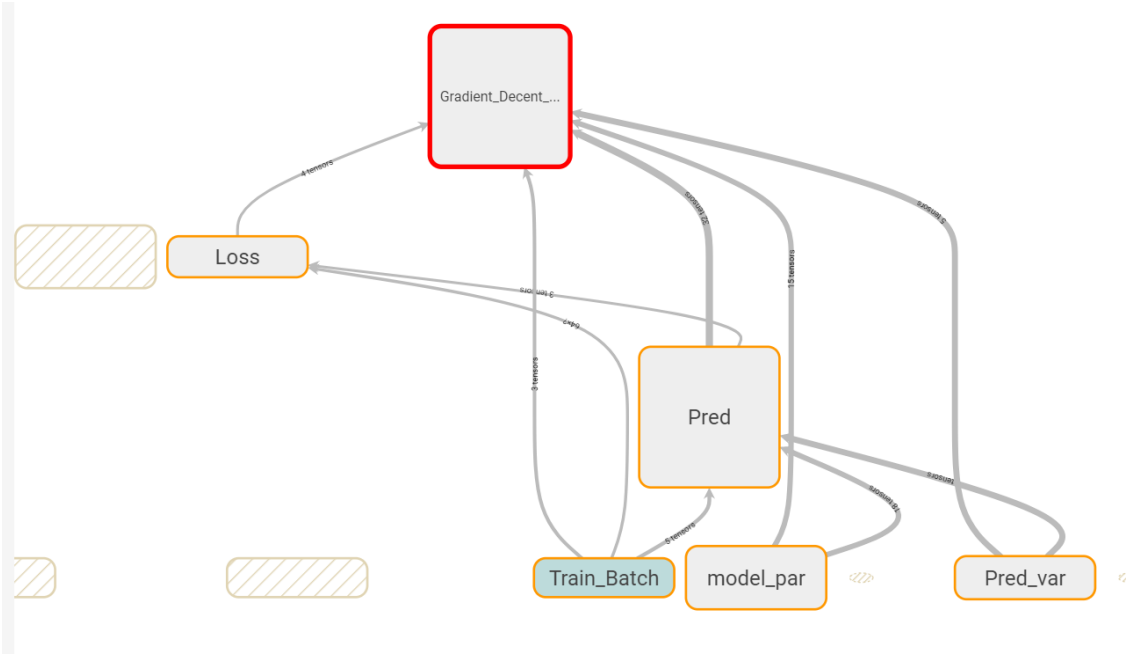


Figure 2.2: Tensorflow training section

2.5 The Model Layout

The model layout is shown in Figure 2.3. The sequential data is sent through a LSTM model to compress the information contained in the temperature and pressure sequence to a single value.

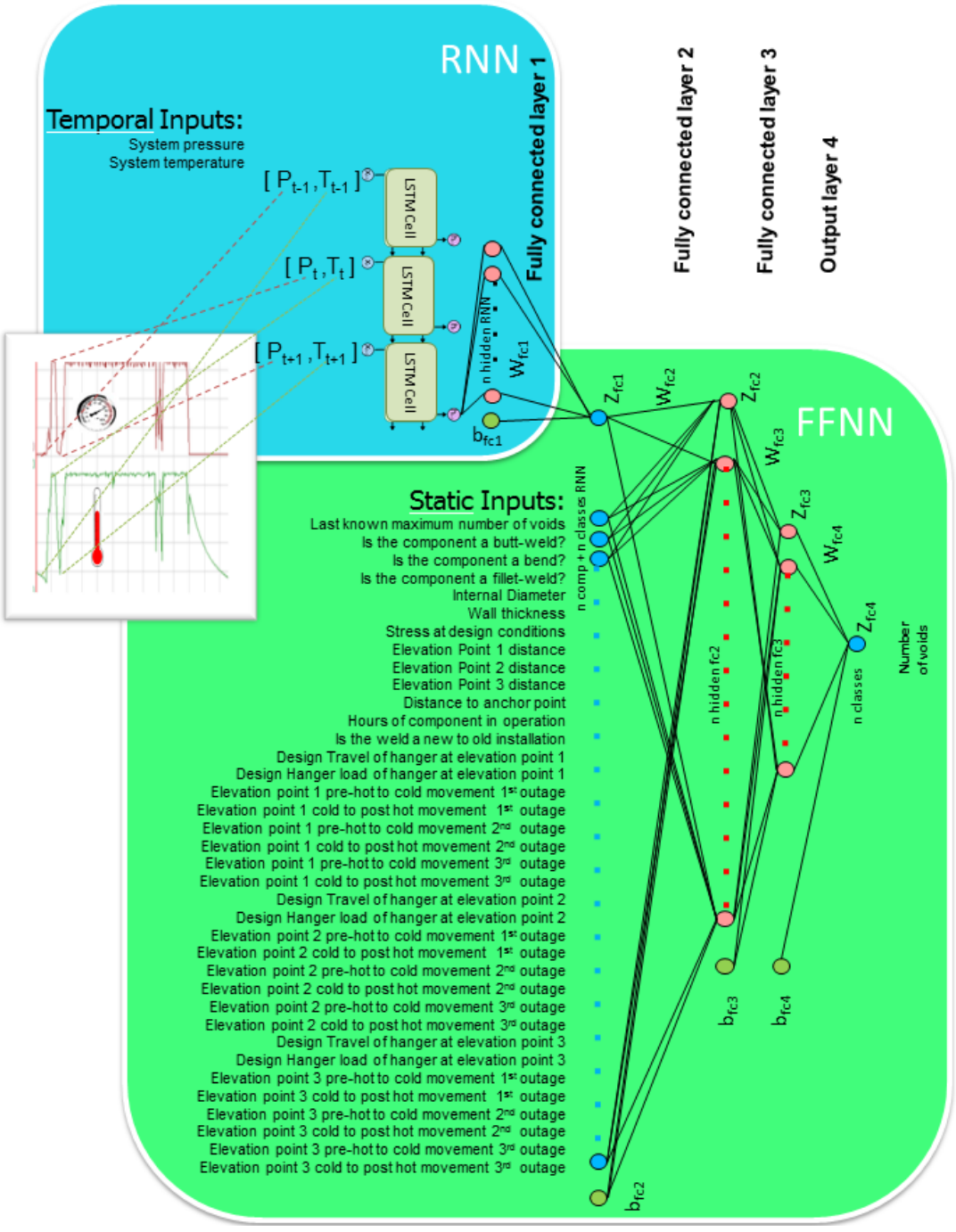


Figure 2.3: Model layout

2.6 RNN Cell

In Appendix A both long short term memory (LSTM) and gated recurrent unit (GRU) cells are employed. It shows that GRU cell does have significant advantages in computational speed, however the complexity of the LSTM usually yields better results. In this case LSTM cell is used to parse the sequential data.

2.7 The Hybrid Recurrent Neural Network Forward Pass

The network architecture needs to be decided before the network can be optimized. Note that in this research the network architecture are loosely based on what previous research has shown to work and the experience of the author with these networks. Increasing the number of nodes and layer of the model causes a complexity increases, this in turn makes the optimisations more difficult and means a larger training set is often required. When the number of layers are increased the model is able to group extracted features/node values together. This means that the model should in theory be able to identify more complex patterns in the data. If the number nodes are increased and the layers are not enough or the layers are increased but there are too few nodes in a layer one will find little to no improvement in model performance. In this research the model architecture are tweaked manually to find a good solution, using optimisation techniques to optimise the hyper-parameters is not practicable as it would require multiple optimisation runs where a single run could take days to complete.

An example of the hyper parameters that define the architecture is:

n hidden RNN	= 32	(RNN cell output size/ number of output hidden nodes)
n classes RNN	= 1	(Number of outputs of the RNN Network)
n comp	= 37	(Number of static component features)
n hidden fc2	= 256	(Number of hidden nodes in fully connected layer 2)
n hidden fc3	= 32	(Number of hidden nodes in fully connected layer 3)
n classes	= 1	(Number of output classes, this is 1 in case of regression)

The feed-forward pass function takes the batch x, batch component, and batch sequence length values as an input.

The built-in dynamic RNN function is used, which enables the use of varying length tensors in the same batch. The dynamic RNN uses the batch x values and the sequence length. The specified cell is an LSTM cell and the outputs of the RNN are calculated as described in section 1.5.8.

The RNN gives an output in the shape: [batch size, max sequence len, n hidden RNN]

The RNN output saves the output vector for each step of the RNN. However only the last output is of interest. The index of the last output of each of sequence in the batch needs to be saved.

$$\text{index} = \text{range}(0, \text{batch size}) \times \text{maximum sequence length} + (\text{seqlen } -1) \tag{2.1}$$

The output is then transformed in the shape[batch size × maximum sequence length, n hidden RNN]. Only the rows with the index numbers are saved in the "index" variable, the rest are discarded. Each line in the output matrix contains the last RNN outputs of shape [n hidden RNN], of an entry of the batch. This means the output is in the shape, [batch size, n hidden RNN].

The final RNN output is calculated for each batch using:

$$\vec{z}_{fc1} = ReLU(\vec{output})\vec{W}_{fc1} + \vec{b}_{fc1} \tag{2.2}$$

where ReLU is an activation function as described in subsection 1.5.5. The RNN Output is concatenated with the static inputs (batch comp) to form a dense input to the next fully connected layer of shape [n classes RNN + n comp].

$$\vec{z}_{fc2} = \tanh(\vec{\text{dense input}})\vec{W}_{fc2} + \vec{b}_{fc2} \tag{2.3}$$

This is sent through more feed forward layers in the same fashion

$$\vec{z}_{fc3} = ReLU(\vec{z}_{fc2})\vec{W}_{fc3} + \vec{b}_{fc3} \tag{2.4}$$

The final layer is sent through a sigmoid to ensure the result lies between 0 and 1.

$$\vec{z}_{fc4} = sigmoid(ReLU(\vec{z}_{fc3})\vec{W}_{fc4} + \vec{b}_{fc4}) \tag{2.5}$$

2.8 Loss Function

In the case where the problem is set up as a single output regression problem, the cross entropy loss (section 1.5.4) is not beneficial over using the root mean squared loss (section 1.5.4)

The loss is calculated per batch hence the loss is:

$$E = \sqrt{\frac{\sum_i (\vec{\boldsymbol{y}}_i - \vec{\boldsymbol{y}}'_i)^2}{K}} \tag{2.6}$$

where y_i is the actual life fraction used for each of the components in the data batch. y'_i is the predicted life fraction used for each of the components in the data batch and K is the size of the data batch.

2.9 Optimisation Algorithm

There exist multiple optimisation algorithms that could potentially be used to optimize the parameters of the model to ensure the loss is as small as possible. Gradient descent based algorithms are often favoured over stochastic algorithms, in machine learning problems where computational power is expensive.

The reason for this is that gradient descent algorithms are relatively easy to apply and the algorithm hyper-parameter optimisation can be done manually with minimal repeated runs. Using gradient descent based algorithms offers repeatable solutions where stochastic methods may yield different results with each run.

Learning stochastic recurrent networks is clearly difficult as explained by Bayer and Osendorfer (2015). Training neural networks with genetic algorithm have been proven to be very successful (Koehn, 1994), however multiple models need to be created and multiple generations of different models need to be generated and evaluated, making it very computationally intensive.

In Appendix A GD and ADAGRAD optimisation algorithms are used. It shows that ADAGRAD outperforms the GD algorithm for the given data set.

There exists multiple gradient descent algorithms such as resilient back-propagation RPROP (Igel and Hüsken, 2000) and delta-bar-delta (Sutton, 1992). In this research ADAGRAD and ADAM are used due to the fact that we a hybrid model is used where the LSTM network receives data much more frequently than the feed forward neural network, this enables different learning rates for the various parameters of the hybrid network. This should theoretically combat the problem where one of the networks in the system over-fits to the data before the other network has been trained properly.

Due to the fact that a hybrid model is used, it makes the computation of the gradients very cumbersome, and making changes to the model layout would mean that the gradients need to be recalculated. Tensorflow does not compute the gradients using numerical methods but uses automatic differentiation. By constructing a Tensorflow graph of the model, Tensorflow knows what the sequence of elementary arithmetic operations will be. Tensorflow automatically applies the chain rule repeatedly to these operations. Hence, it does not make use of symbolic differentiation or numerical differentiation. Using automatic differentiation has the advantage over numerical differentiation that the answer is exact and fewer function evaluations are needed to determine the gradient.

2.10 Calculating the Accuracy

The model has been set up as a regression problem and the accuracy of the model is an arbitrary value. Four arbitrary classes are created based on the maximum void count of a component. Traditionally the components are placed in a class based on the hours of remaining life, however this is dependent on the component type and component material as well as the location of the void count. To simplify one would just want to categorize the components based on the number of voids as this does have a relation to the remaining life of a component.

The assumption is made that sufficient preventative maintenance action can be taken if each of the components can be categorised correctly into the correct quartile based on remaining life.

The components are considered to be completely creep exhausted once the void-count reaches 1000 voids per square millimetre.

Hence, four classes are created:

Table 2.2: Classification labels for accuracy calculation

Class	**Description**	$\vec{y}$ **Encoding**
Class1	Void Count 750+	[1,0,0,0]
Class2	Void Count 500 to 750	[0,1,0,0]
Class3	Void Count 250 to 500	[0,0,1,0]
Class4	Void Count 0 to 250	[0,0,0,1]

$$Accuracy = \frac{\sum_{i=1}^{K} \delta(Class(\vec{y}_i'), Class(\vec{y}_i))}{K} \tag{2.7}$$

The δ function is 1 if the constituents of the function variables are equal to one another. $Class(\vec{y}')$ is the predicted class, $Class(\vec{y})$ is the actual class and K is the batch size.

Chapter 3

Machine Learning Model Application

In this chapter application of the model on real world data is described. The different data sources are described in this chapter as well as the data handling, such as grouping the data into training, testing and validation sets. The automated data reading methods used to read the data from the data sources are briefly described. Once all the data has been collected the data is normalized and then converted into Tensorflow record files, which will be used by the machine learning model. Finally the Tensorflow records are used to train, validate and test the machine learning model described in chapter 2. In chapter 4 the results obtained from the machine learning model are discussed.

3.1 Generating the Data Set

The dataset consists of data collected from a coal-fired power station, from 2011 to 2017, for the main steam piping systems of five of the operating units. The data consists of the temperature and pressure that the piping system experienced during operation, elevation survey data, metallographic inspection results, pipe stress analysis results as well as constant component properties.

3.1.1 Temperature and Pressure Data

The temperature and pressure data is provided in five minute intervals. Four temperature measurements are provided, one at each of the boiler outlet locations. The pressure signal is measured at one location in the common line of the main steam piping system. Even during so called stable operating conditions the temperatures does not stay constant and the temperature at each of the four boiler outlets may differ significantly due to various operational deviations. The boiler is often shut down for scheduled and unscheduled outages. Boiler trips do also occur when out of normal operating conditions occur. Due to the age of the plant these events do occur often and can be seen in Figure 3.1 when all the temperature and pressure signals drops close to atmospheric conditions. This is not to be confused with noise in the data.

The temperature and pressure data is downloaded monthly and saved in comma delimited files

(*.csv). A Python script is used to read all the data contained in the files into a data frame. Table 3.1 illustrates the format in which the data is captured, the table is an extract of data sequences that are very long.

Table 3.1: Pressure and Temperature Extract

Time	**RA14T003**	**RA13T003**	**RA12T003**	**RA11T003**	**RA20P006**
2011/01/01 00:05	542.125	534.359	573.773	541.392	16.63
2011/01/01 00:10	542.125	538.462	553.26	537.289	16.514
2011/01/01 00:15	542.125	538.462	553.26	533.187	16.41
2011/01/01 00:20	538.022	538.462	561.465	537.289	16.52
2011/01/01 00:25	538.022	538.462	565.568	537.289	16.52
2011/01/01 00:30	538.022	538.462	565.568	537.289	16.41
2011/01/01 00:35	538.022	538.462	565.568	537.289	16.52
2011/01/01 00:40	538.022	538.462	557.363	537.289	16.624
2011/01/01 00:45	538.022	538.462	557.363	537.289	16.514
2011/01/01 00:50	538.022	538.462	557.363	537.289	16.514
2011/01/01 00:55	538.022	538.462	561.465	537.289	16.404
...	...	...	...	...	...

Figure 3.1 illustrates the data that has been captured for the fifth operating unit. The data consists of four temperature signals and one pressure signal.

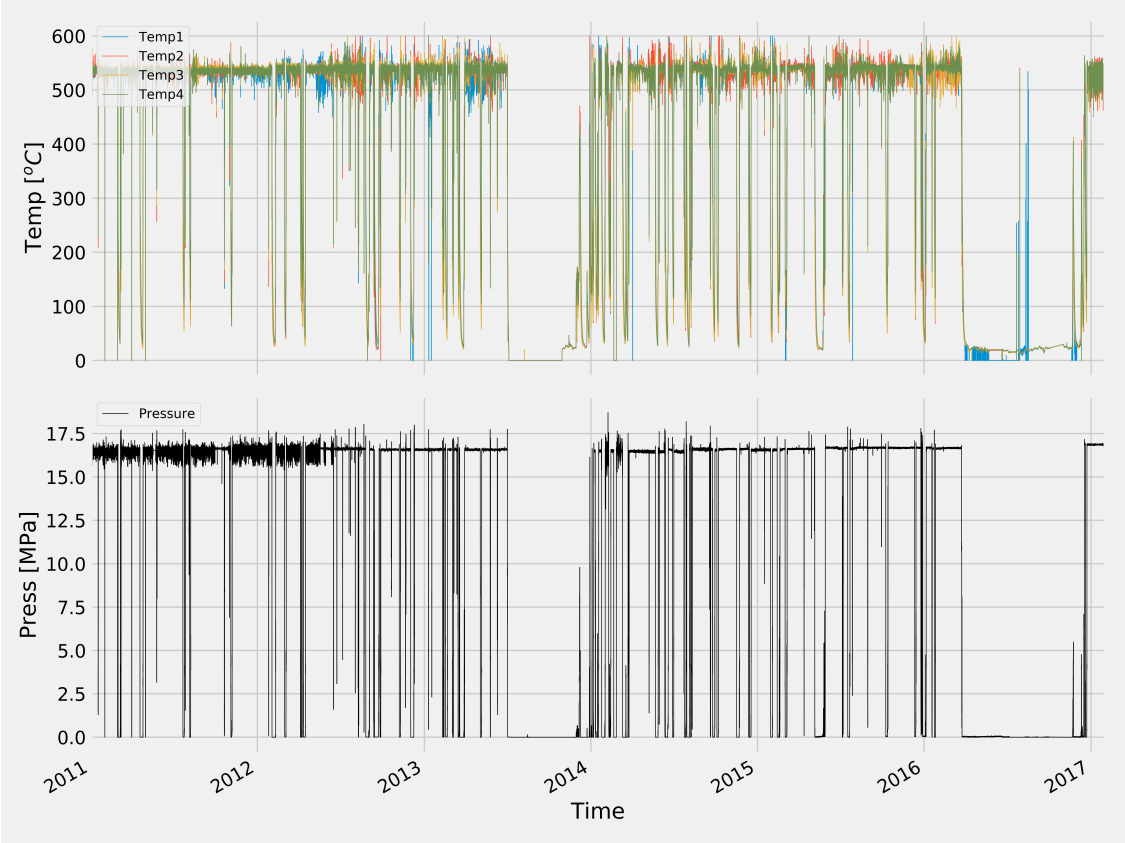


Figure 3.1: Unit 5 pressure and temperature data

During operation, some instrumentation might fail or give false readings. Incorrect readings are ignored from the dataset. It is known that the pressure and temperatures in a boiler operating under normal conditions has a high positive correlation. There should also be a near perfect positive

correlation between the four temperatures and the pressure measurement of each unit.

A correlation matrix is constructed for each temperature and pressure dataset. The correlation coefficient is calculated using:

$$r_{ab} = \frac{n\Sigma\vec{a}_i\vec{b}_i - \Sigma\vec{a}_i\Sigma\vec{b}_i}{\sqrt{[n\Sigma\vec{a}_i^2 - (\Sigma\vec{a}_i)^2][n\Sigma\vec{b}_i^2 - (\Sigma\vec{b}_i)^2]}} \tag{3.1}$$

where n is the number of pairs of sequences, $\vec{a}$ is the first time sequence vector and $\vec{b}$ is the second time sequence vector.

Figure 3.2 shows the correlation between all the temperature and pressure signals collected. Signals 0 to 4 belongs to unit one and 5 to 9 belongs to unit two and so forth. It is clear that there is a strong correlation between each one of the units. It is easy to see the signals within each of the units that do not correlate well, such as signal numbers 0, 3 and 21. Typical reasons why one signal might not correlate well with the rest is that the instrumentation might be damaged, slightly detached from the pipe or off calibration. Operating abnormalities such as tube leaks, stuck spray-water valves or during on-line boiler cleaning events may cause an actual difference in steam temperature between the different boiler outlet locations.

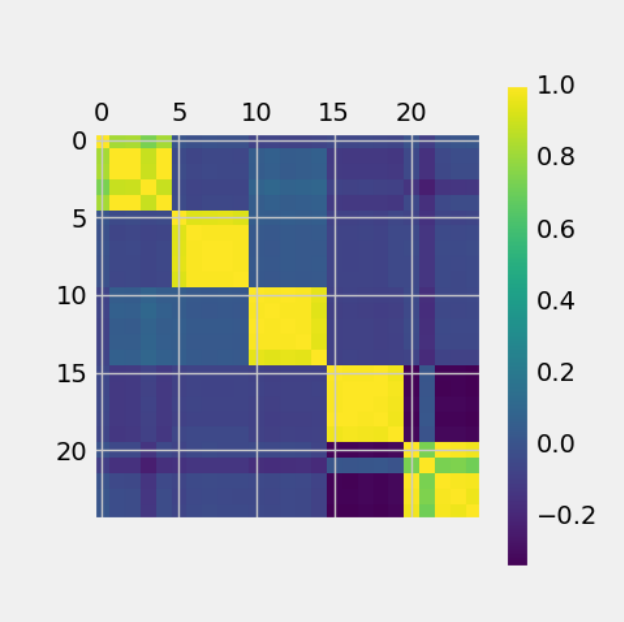


Figure 3.2: Overall correlation matrix

Individual correlation matrices are done per outage time frame, shown in Figure 3.3. In this case there exists a high correlation between the four different temperature signals collected. None of the temperature signals shall be ignored by the machine learning model and the effective operating temperature shall be the average of the four temperature signals for each time step.

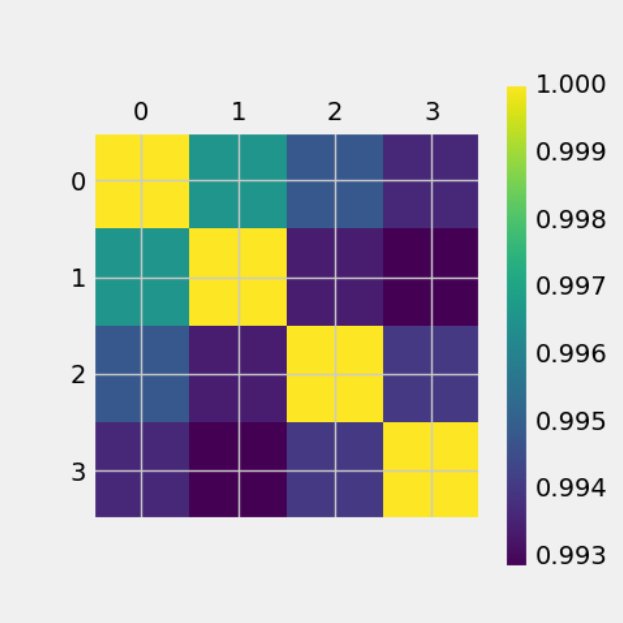


Figure 3.3: Temperature correlation matrix unit 1 2012-01 to 2013-02

Only one pressure signal is recorded because the pressure signal is already processed from three different pressure signals into a single pressure, before it is stored to the power station distributed control system's archive. A two out of three system is used where the two temperatures that are closest to one another are used to calculate an average pressure which is used as the operating pressure of the piping system. All the temperature and pressure data is from 2011 to 2017 read automatically from various excel files and is stored in a Python based data frame.

3.1.2 Elevation Survey Data

The elevation survey data is a measurement of the pipe-movements that is done before an outage while the unit is on load (pre-hot) during the outage while the unit is cold and after the outage while the unit is hot (post-hot). Throughout the unit there are fixed datum points (Figure 3.4a) on the surrounding steel and concrete structures. An automatic level is used to determine the elevation of different points along the pipe with respect to the datum points (Figure 3.4b). A measuring rod is pushed through a hole in the insulation in order to measure the elevation of the bottom of the pipe section. The errors in measurement between two datum points can be as high as 2mm. Differential expansion between the boiler and the surrounding structure could account for further inaccuracies that are difficult to quantify.

(a) Datum points 64 meter level

(b) Pipe elevation survey measurements

Figure 3.4: Elevation survey

Table 3.2 shows an extract from typical elevation survey readings. The elevation is measured as the height above ground level. Each elevation point is measured relative to a known fixed elevation point usually on one of the eight main boiler support columns, or a concrete surface. These fixed elevation points usually are less prone to thermal movements, than other structural members of the boiler house. The elevations are measured at each support location as well as discrete measuring points between supports marked with ".1" (see Table 3.2).

The main purpose of the elevation surveys are to determine if there are any counter slopes that may prevent condensate to drain towards the drain points. The hypothesis however is that the elevation survey also gives some insight into the operating performance of the supports. If a support is operating outside of its design conditions, increased system loading on surrounding components would be expected. This will increase the rate of creep void formation in those components.

Equation 1.38 shows that the number of voids is directly proportional to the remaining life of a component. As seen in Equation 1.41, the component remaining life is a function of the stress and temperature under which the component operates. This means the life fraction used and number of voids is a function of the temperature and the stress under which a component is operated. Thus one can argue that the rate of void formation should be a direct function of stress and temperature under which a component is operated. The stress that a component experiences is a function of the pressure, temperature and external forces applied to the system. The only variable external forces working in on the system is the forces applied by the pipe supports. This can be seen in the pipe stress analysis described in subsection 3.1.4 and Figure 1.2. When one of the supports is broken the stress of the piping system is effected up to the closest three pipe support locations. Components located further are also influenced, but the change in pipe stress is regarded to be negligible.

In other words a components' useful life is directly dependent on the stress under which the component is operated as shown in subsection 1.5.12. According to the beam stress theory, the bending moment applied to a component is directly dependent on the displacement of the support as well as the distance of the component to the support. The bending moment applied to a component has a direct influence on the maximum stress intensity experienced by a component. The support locations with respect ot each of the components are known and used as a static input to the model. The change of displacement at the support location is also measured when the system is changed

from cold to hot operating state and is also used as a static input to the model.

An assumption is made that the rate of void formation is a function of the difference of expected and actual movement of a support between hot and cold conditions as well as the distance of a component to the support and the load carrying capacity of the supports.

Table 3.2: Elevation survey unit 5 extract

SUPPORT	**Hot 1 (Post) May 1999**	**Hot 2 (Pre) Dec 2000**	**Cold 1 Mar 2001**	**...**	**Hot 9 (Post) Apr 2010**	**Hot 10 (Pre) Mar 2013**	**Cold 6 Nov 2013**	**Hot 11 (Post) Jan 2014**
RA-11								
.1	65.364	65.382	65.48	...	65.352	65.352	65.457	65.371
MS2	65.369	65.392	65.475	...	65.330	65.325	65.427	65.351
.1				...	65.314	65.308	65.406	65.355
MS6	65.380	65.390	65.444	...	65.274	65.268	65.356	65.314
.1	65.374	65.400	65.446	...	65.263	65.256	65.343	65.303
MS10	58.010	58.010	58.099	...	57.877	57.863	57.996	57.918
MS14	46.169	46.176	46.263	...	46.103	46.100	46.227	46.129
MS20	46.109	46.114	46.126	...	46.079	46.079	46.083	46.068
MS22	45.932	46.031	45.964	...	46.036	46.051	45.979	46.037
...	...	...	...	...	...	...	...	...

When a component is operating outside of its design intent it is not carrying the appropriate load or travelling the appropriate distance, when the boiler operating conditions are changed from zero % load to 100% load and vice versa. If a support does not operate within design intent or there is an imbalance in the piping system, the supports will not move as designed when the piping system changes from a cold to hot condition. These imbalances cause system loading that increases the stress in surrounding components.

The model takes the design travel as an input ass well as the actual travel. The travel is calculated as the hot elevation reading minus the cold elevation reading. Thus, for each outage there are two travel measurements. One from pre-hot to cold and one from cold to post-hot. Elevation survey measurement of the past three outages are assumed to be significant and used as an input to the model, because elevation data that dates further back is not available for all units.

In some cases the unit is shut down earlier that planned due to unforeseen circumstances. In these cases there is no time to perform a pre-hot elevation survey and the last known post-hot elevations are assumed to be still valid.

In order to quantify the deviation in load applied to the pipe at the support location, it is important to know the design load schedule shown in Table 3.3.

Table 3.3: Load schedule extract

Support Number	**Design Travel [mm]**	**Hanger Load [kN]**
MS2	-104	14.5
MS6	-94	13.4
MS10	-135	60.4
MS14	-135	19.2
MS20	-29	19
MS22	68	22
...	...	...

3.1.3 Metallographic Inspection Results

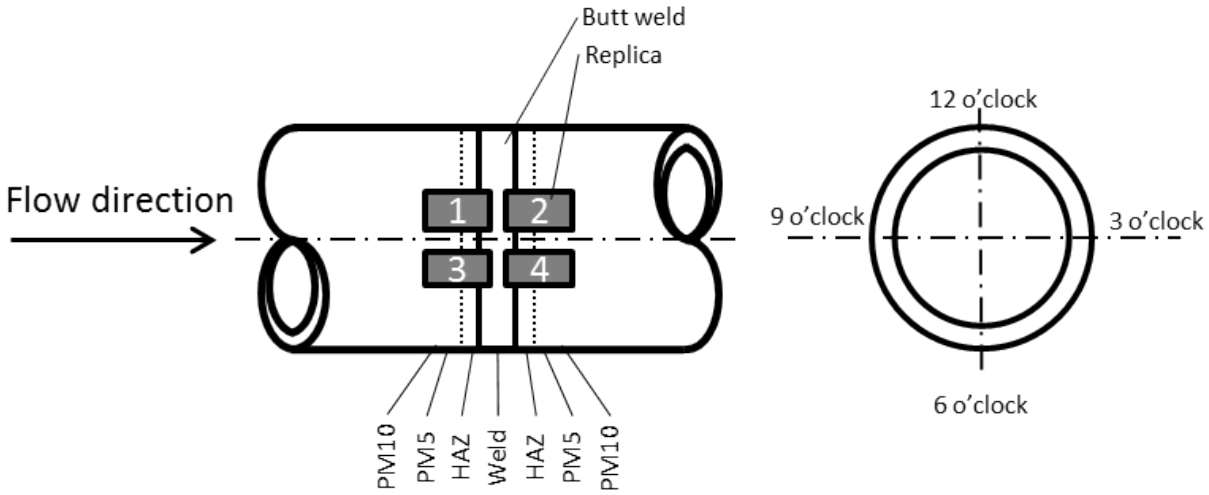
The metallographic inspection results give an indication of the number of creep voids per square millimetre. This has a direct correlation to the life fraction consumed and remaining life of a component as shown in Equation 1.38.

Welds are usually inspected at four locations around the circumference of the weld. At each location multiple replica strips are used to replicate the ground and polished metal surfaces as shown in Figure 3.5a. Figure 3.5b shows the location of replicas for fillet welds.

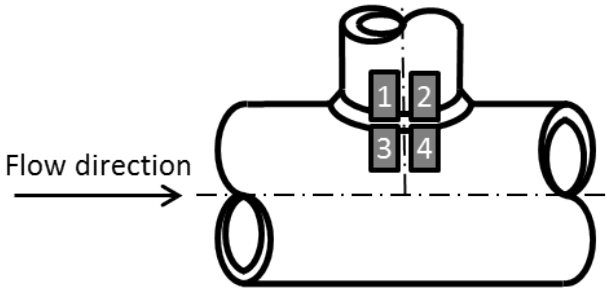
The metal chemical etchant is used to reveal the micro-structure of the metal. The metal is subsequently cleaned and an acetate film is placed on the metal surface to replicate the micro-structure of the metal.

The replica strips are sent to a lab where the number of creep voids are counted manually through a microscope.

Replica strips are placed across the weld so that the following locations can be measured. The form in which the results are captured is shown in Table 3.4.



(a) Straight pipe replica location



(b) T-piece replica locations

Figure 3.5: Replication locations

Around the circumference of the pipe the void count at each of the following locations are recorded:

- PM10 -Parent material 10mm upstream
- PM5 -Parent material 5mm upstream.
- HAZ -Heat affected zone upstream.
- Weld -Weld material
- HAZ -Heat affected zone downstream.
- PM5 -Parent material 5mm downstream.
- PM10 -Parent material 10mm downstream

Table 3.4: Outage report replica results extract

Comp	**No**	**PM 10**	**PM 5**	**HAZ**	**WELD**	**HAZ**	**PM 5**	**PM10**	**Min WT (mm)**	**Depth (mm)**
Butt weld	RA11.Blr exit		-	-	-	<20	<20		29.88	0.72
			<20	<50	-	-	-		29.44	0.89
			<20	<50	-	-	-		28.53	0.93
			<20	<20	-	-	-		29.88	0.71
Butt weld	RA11.1011		-	-	-	-	-		32.16	0.33
			-	-	-	-	-		33.19	0.71
			<50	<50	-	<50	<50		33.84	0.69
			-	-	-	<20	<20		31.88	0.5
Butt weld	RA11.1012		<50	160	-	200	100	60	30.5	0.7
			<20	70	-	250	140		29.9	0.6
			<20	60	-	120	<50	100	30	0.8
			<50	70	-	160	140		30	0.6
...	...	..	...	...	...	...	...	...	...	...

3.1.4 Pipe Stress Analysis

A pipe stress analysis was done for steady state design conditions of the pipework. The analysis was done using Caesar II software, the model get updated as modifications are made to the main steam systems.

The model includes the pipe support effort, however the assumption is made that the model operates with optimum support effort. The model does account for the insulation weight. Both the main steam and cold reheat systems are modelled as they are interconnected with the HP bypass lines.

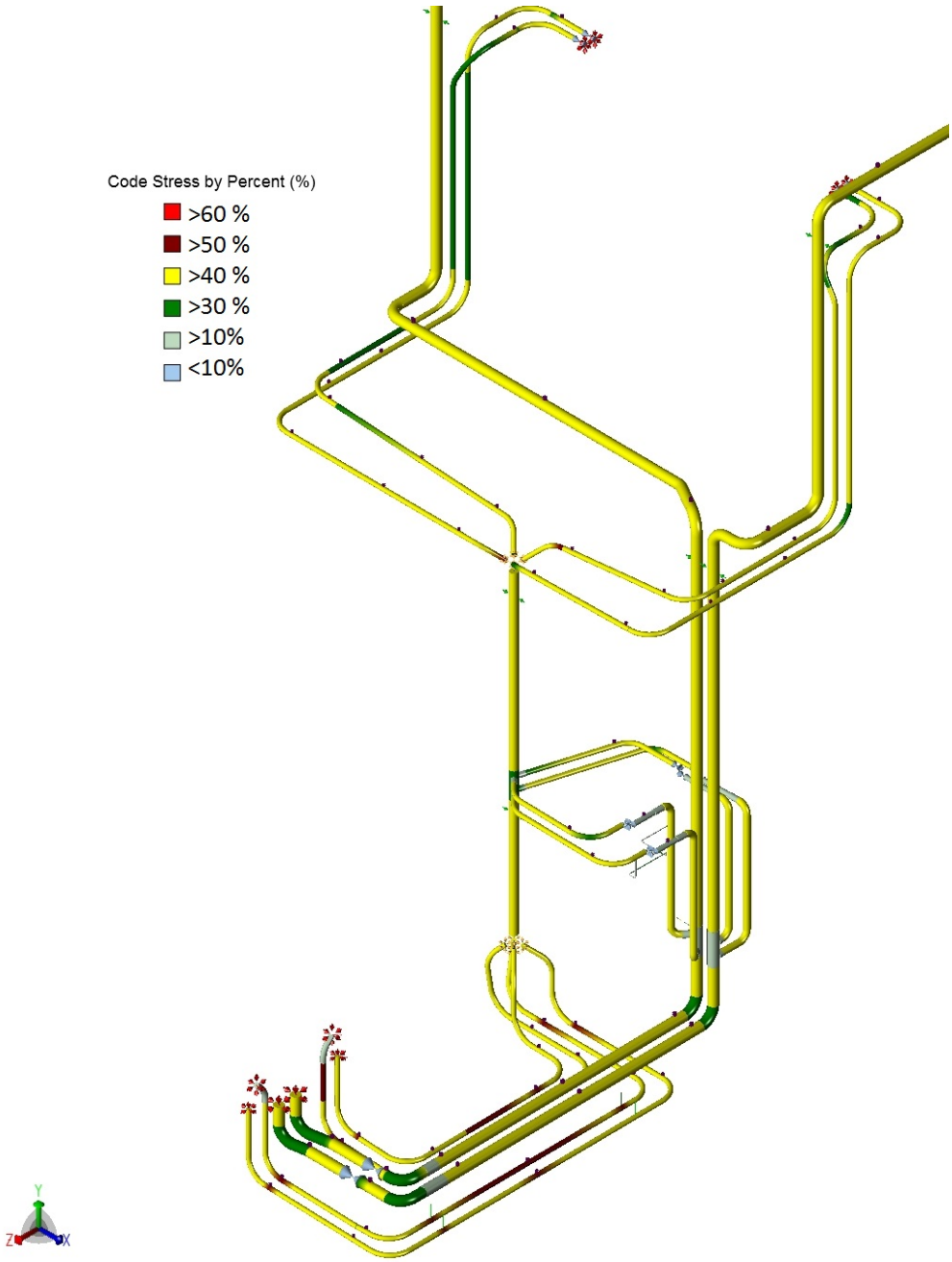


Figure 3.6: Main steam stress

The software calculates a code stress according to the EN 13480:2012 code requirements. The code stress for each component under steady state design conditions is used as a training input feature to the model. The actual stress value is not what is of importance in this analysis. A stress analysis is done to determine the stress that a component experiences relative to the stress experienced by the other components. Components that operate under higher stress will tend to show accelerated creep void formation. The operating load case below considers:

W - Self weight
D1 - Displacements at the turbine legs and boiler exits (as they are not fixed)
T1 - Design Temperature 545 [$°C$]
P1 - Design Pressure 16.9 [MPa]
H - Hanger support efforts as per hanger load schedule
F1 - External loadings to simulate valve weights and forged pieces weights.

Table 3.5: Caesar II pipe stress results operating case (W+D1+T1+P1+H+F1)

Name -Node	**Axial Stress kPa**	**Bending Stress kPa**	**Torsion Stress kPa**	**Hoop Stress kPa**	**Max Stress Intensity kPa**	**Code Stress kPa**	**Piping Code**
3.101.RA22 - 10	30324.7	15127.2	-320.8	69287.3	99580.9	55230.1	EN-13480
3.102.RA22 - 20	30297.3	15245.6	320.8	69287.3	99580.9	55348.5	EN-13480
3.102.RA22 - 20	30297.3	15245.6	-320.8	69287.3	99580.9	55348.5	EN-13480
3.103.RA22 - 30	30261.3	15403.8	320.8	69287.3	99580.9	55506.6	EN-13480
3.103.RA22 - 30	29259.3	14345.4	-298.8	67237.1	97562.2	53438	EN-13480
	...	...	...	...	...	...	...

The model used for the simulation is the model that is used by the system engineers to see the impact of changes to the system on the stresses of the components and to ensure code compliance. The model is not specific to a single unit, but is a generic model for all units.

3.1.5 Defining the Components

In this study components are defined as circumferential butt welds, fillet welds or bends. Past experience indicate that these are the areas where excessive damage are most likely to occur. Figure 3.7 illustrates the component numbers for leg RA11 of the main steam piping system.

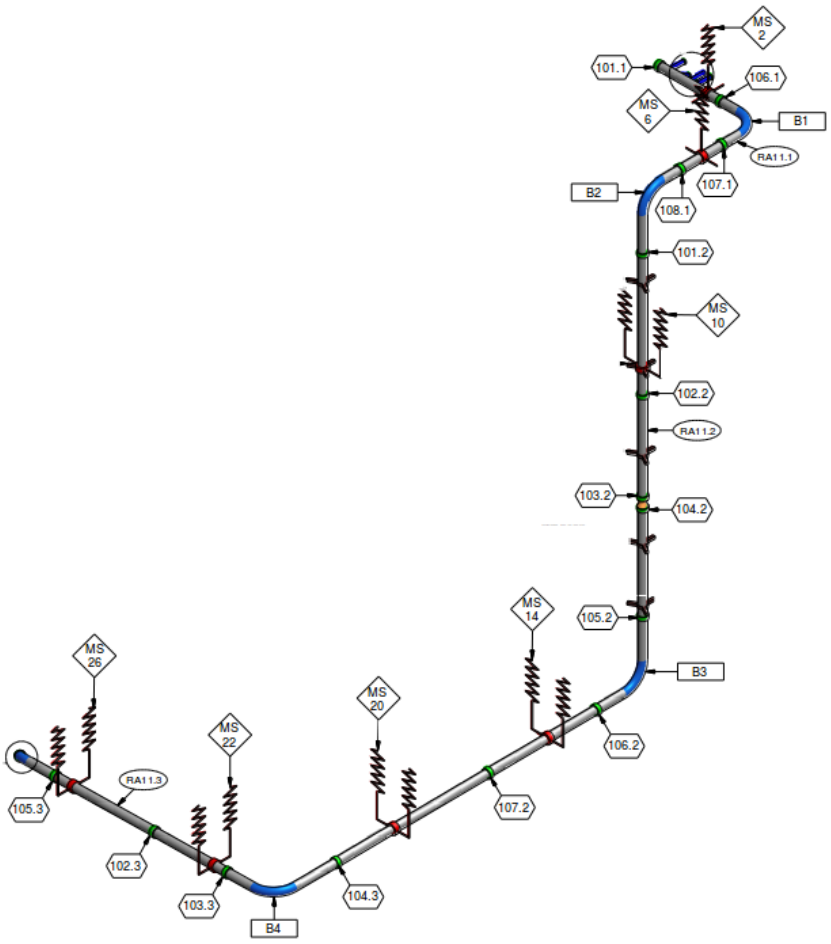


Figure 3.7: RA 11 Main Steam Components

A Python dictionary was created that contains all the information of a component that is assumed constant throughout its operating life.

Some information such as text (string value), cannot be fed into a mathematical machine learning model. For instance, the component type is described by a string value "Buttweld". In order to discriminate between component types a one hot encoding vector was generated. In this case there exist three different component types thus an encoding vector could be [1, 0, 0] which indicates that the component is of the first type. Thus, only one of the values in the vector is 1, the rest are 0.

The Python dictionary captures the following information for each of the components in the piping system:

Component Name	-	The unique name of the component that corresponds to similar components on other operating units
Component Type	-	One hot encoding (buttweld, filletweld or bend)
ID	-	Internal Pipe Diameter
WT	-	Pipe wall thickness
Stress	-	Code stress under design conditions
EP1	-	Elevation Point 1 (the closest support)
EP1 dist	-	Horizontal distance along pipe to Elevation Point 1
EP2	-	Elevation Point 2 (the second closest support)
EP2 dist	-	Horizontal distance along pipe to Elevation Point 2
EP3	-	Elevation Point 3 (the third closest support)
EP3 dist	-	Horizontal distance along pipe to Elevation Point 3
Dist Anchor	-	Distance to an anchor point
Repl HRS	-	The year in which the component was installed
OTN	-	This denotes if the weld was new material to old material or new material to new material at the time of installation

3.1.6 Defining the Outages

An outage dictionary is created to describe the time that a unit was started up after an outage as well as the date that the unit was shut down for an outage. The dictionary describes the location of the outage report files as well as the location in the outage report that contains the replica results. The outage dictionary also describes the location where elevation survey data must be read from.

Table 3.6: Outage dictionary extract

Outage num	**Unit num**	**Opp Start**	**Opp End**	**Outage report**	**Replica Sheet**	**Elevation Sheet**
1	1	2011-05-01	2011-09-01	20120731-DUV-U1-HP-Outage Report October 2011.xls	ATT 11 - MS Replica results	TRPT002723 DUVHA1HP.xlsx
2	1	2012-01-01	2013-02-26	20130325-DUV-U1-HP-Outage Report February 2013.xls	ATT 10 - MS Replica Res	TRPT002723 DUVHA1HP.xlsx
3	1	2013-03-06	2013-09-29	20140116-DUV-U1-HP-Outage Report October 2013.xls	ATT 10 - MS Replica Results	TRPT002723 DUVHA1HP.xlsx
4	1	2013-09-29	2014-07-05	20140812-DUV-U1-HP-Outage Report July 2014.xls	ATT 10 - MS Replica Results	TRPT002723 DUVHA1HP.xlsx
...	...	...	...	...	...	...

3.1.7 Grouping Datasets

For this dissertation the data for all the components in a recorded outage is kept together. This outage data set is then grouped into one of the three data groups [train, valid, test]. The probability that the data for any given outage belongs to a certain group is selected to be the following:

$$\begin{aligned} \boldsymbol{P}(train) &= 0.4 \\ \boldsymbol{P}(valid) &= 0.3 \\ \boldsymbol{P}(test) &= 0.3 \end{aligned}$$

A uniform random number generator is used to generate a number, the data for the specific outage is grouped in a data group based on the number that was generated.

In order to compare the difference in model performance after certain changes have been made, the grouping is kept the same for all the runs. The datasets are placed in a group per outage and not per component. Firstly, the data batches that the model will receive in future will be per outage and not a mixture of components from different outages. Secondly, if the components of one outage is distributed between the train, test and valid groups, the model will be trained on the temperature and pressure sequences that will appear again in the test and validation datasets. This is not ideal as the model could over-fit to those sequences.

There are 15 outages in total and they are numbered as in Table 3.6. During each outage multiple components may be inspected thus each outage contains a set of components. Each outage is grouped in one of the three data groups.

The following grouping shown in Table 3.7 was obtained for the first run. For all other runs this same grouping was forced to ensure the runs could be compared directly with one another.

Table 3.7: Outage Grouping by Outage Number

	Train	**Test**	**Valid**
	3	1	4
	7	2	6
	12	5	8
	14	10	9
	15	11	13
Total number of components	627	322	369

The spread in data is ideal, as we would typically use the largest part of the data to train the model, while the test and validation is still large enough to make an objective decision on the performance of the model.

3.1.8 Automated Data Reading

The script iterates through the inputs in the outage dictionary. The operational temperature and pressure data signals are read from the data frame that contains all the pressure and temperature data. This is saved in an operational data pickle (Opp data.pickle).

The replica data is read from the corresponding outage report. The components that have been

Table 3.8: Opp Pickle extract

Date Time Index	**Common Line Pressure [MPa]**	**Average Temp [° C]**
12/02/06 00:00	16.667	543.66325
12/02/06 01:00	16.654	544.24925
12/02/06 02:00	16.667	544.83525
12/02/06 03:00	16.661	544.359
12/02/06 04:00	16.673	546.0075
12/02/06 05:00	16.685	547.5825
12/02/06 06:00	16.679	549.5605
12/02/06 07:00	16.673	544.79875
...	...	...

inspected during the outage is kept and other components are discarded. A data frame is generated that contains a list of all components inspected during this outage. The replica results of each of the components are saved in the data frame as well as the last known replica results of the components (Replica.pickle).

The properties of each of these components are read from the component database. Elevation survey results for the closest three supports of each component are read from the elevation data-frames. The component data-frame is saved in a component pickle (Component.pickle).

Table 3.9: Component pickle extract

Component Name	RA11.Blr.exit	RA11.101.1	RA11.106.1	RA11.107.1	RA11.108.1
Current Max	50	2300	1100	2200	1700
Previous Max	80	70	60	80	140
Buttweld	1	1	1	1	1
Bend	0	0	0	0	0
Fillet	0	0	0	0	0
ID	250	250	250	250	250
WT	31	31	31	31	31
SIF	69382	69382	65329	39910	51347
EP1 dist	2587	2000	-510	-4460	-6960
EP2 dist	8097	7337	4907	270	-1230
EP3 dist	10494	9734	7307	3357	1857
Dist Anchor	0	-760	-2510	-5472	-7972
Repl Hrs	368	44	44	368	368
OTN	0	1	1	0	0
Design Travel 1	-104	-104	-104	-104	-104
Hanger Load 1	14.5	14.5	14.5	14.5	14.5
Hot 1.1.1	-0.093	-0.093	-0.093	-0.093	-0.093
Hot 1.1.2	-0.092	-0.092	-0.092	-0.092	-0.092
Hot 1.2.1	-0.1	-0.1	-0.1	-0.1	-0.1
Hot 1.2.2	-0.101	-0.101	-0.101	-0.101	-0.101
Hot 1.3.1	-0.092	-0.092	-0.092	-0.092	-0.092
Hot 1.3.2	-0.095	-0.095	-0.095	-0.095	-0.095
Design Travel 2	-94	-94	-94	-94	-94
Hanger Load 2	13.4	13.4	13.4	13.4	13.4
Hot 2.1.1	-0.086	-0.086	-0.086	-0.086	-0.086
Hot 2.1.2	-0.078	-0.078	-0.078	-0.078	-0.078
Hot 2.2.1	-0.125	-0.125	-0.125	-0.125	-0.125
Hot 2.2.2	-0.104	-0.104	-0.104	-0.104	-0.104
Hot 2.3.1	-0.098	-0.098	-0.098	-0.098	-0.098
Hot 2.3.2	-0.081	-0.081	-0.081	-0.081	-0.081
Design Travel 3	-135	-135	-135	-135	-135
Hanger Load 3	60.4	60.4	60.4	60.4	60.4
Hot 3.1.1	-0.111	-0.111	-0.111	-0.111	-0.111
Hot 3.1.2	-0.109	-0.109	-0.109	-0.109	-0.109
Hot 3.2.1	-0.175	-0.175	-0.175	-0.175	-0.175
Hot 3.2.2	-0.143	-0.143	-0.143	-0.143	-0.143
Hot 3.3.1	-0.136	-0.136	-0.136	-0.136	-0.136
Hot 3.3.2	-0.127	-0.127	-0.127	-0.127	-0.127

The automated data reading programs create two pickle files for each recorded outage, that will be used to train the model.

1. Opp data.pickle

2. Component.pickle

The Opp-data pickle creates the temporal data for the outage. The assumption is made that

all components in the main steam piping system had been subjected to the same pressures and temperatures during operation from the previous outage to the current outage. The piping system is well insulated with less than $5°C$ temperature loss from system inlet to outlet. There are also no valves or components in the main steam system that cause a considerable pressure loss.

The component pickle contains the static data of the components that have been inspected during the current outage. Static data is data that is assumed to stay constant throughout the duration of the operation from the previous outage to the current outage, such as pipe diameter and wall thickness. The movement of the pipe supports is not known, however there is a static record of the pipe movement from cold to hot, before and after each major outage. Thus the assumption is made that the pipes were at the measured elevation for the entire duration of pipe operation from the previous outage to the current outage. In the Hybrid RNN all these inputs are inserted on the same level, hence there is no discrimination as to which inputs are more important than other. During the training phase inputs that causes allot of confusion or that has little or no influence on the model output will be trained out. This means that the weight parameters associated with a given input shall typically tend to zero.

The inputs that are used in this research are chosen based on experience that has been gained while managing the assets. It is well known that the main damage mechanism in the main steam systems is high temperature creep. It is also known that there is an influence on the creep degradation due to thermal fatigue. All information that is available that can give information about the component as well as the conditions that is was operated under while in operation is used as an input to the model. During the operation the stress that the components is operated under is not measured, however the operating temperatures and pressures are an indirect measurement as these two factors play a very large roll in the stress that a component experiences. The previous metallurgical results are used as an input as it gives information about the starting condition of the component. The temperature and stress cycles that the piping systems is operated under has a direct influence on the creep and fatigue degradation that component experiences. This is why the temperature and pressure sequences are used as an input as there is valuable information contained in these sequences that we hope to extract using the machine learning model, without doing in depth physics based computations.

The elevation survey results contains information about the external forces that is applied to the piping system. The external forces applied to the system has a direct influence on the stresses experienced by the surrounding components. The stress of a component under design conditions is used as an input because this gives an indication of the stress that a component should experience under normal conditions. Component that is operating under a higher stress tends to have accelerated creep damage.

3.2 Serializing the Data into Protocol Buffers

After the data has been read into the respective pickle files as described in section 3.1.8, the data is read from the pickles, normalized and then saved into Tensorflow record files.

3.2.1 Normalizing

The data is normalized using min-max scaling described in section 1.5.9. If a data point T lies between T_{min} and T_{max} the normalized value x will be in the range [0,1]. This makes the neural network computationally more stable, meaning that features such as stress in the 10^5 [kPa] range can be used in the same model as the elevation survey features in the 10^{-3} [m] range.

$$x_i = \frac{T_i - T_{min}}{T_{max} - T_{min}} \tag{3.2}$$

The following T_{min} and T_{max} values are used to normalize the training data (Feature $\in [T_{min}, T_{max}]$):

Temperature	∈ [0,600]	Pressure	∈ [0,17]
Buttweld	∈ [0,1]	Bend	∈ [0,1]
Filletweld	∈ [0,1]	Pipe ID	∈ [0,500]
Pipe Stress	∈ [20000,170000]	Maximum Voids	∈ [0,1000]
EP1	∈ [-19554, 5802]	EP2	∈ [-9832, 10917]
EP3	∈ [-3824,14317]	Distance to Anchor	∈ [-31714, 27568]
Months in operation	∈ [0,369]	Old to new	∈ [0,1]
Support design travel	∈ [-154,168]	Support design load	∈ [0,156.8]
Support movement	∈ [-0.154,0.168]		

Converting Data-frames to Tensor-flow Record Files

Tensorflow has the ability to serialize the protocol buffer to a string and then writes the string to a Tensorflow record file. The Tensorflow record files contain the different features of an entry as fields, the entry can be either a sequence or a single value. Using this approach has multiple advantages, such as batching the data of multiple components together is easy. The main reason for using this is that computations are sped up significantly as the data is already in the format that the Tensorflow graph is expecting to receive. During training, time is not waisted in reading data from files and then serializing the data.

A Python script loads the respective datasets (train, test and validation) one at a time. The script cycles through each of the outages in the dataset and loads the pickles that have been created for the given outage into the respective *opp data* data-frame and *component* data frames.

The script then cycles through each component in the component data frame of the current outage, and loads the corresponding pressure data temperature data, sequence length of the operational data, component data and label.

Each of the five information sets are serialized and appended to the TFrecords file. Each entry to the TFrecords file is indexed by a feature, in this case the features are called: 'Press', 'Temp','Seqlen', 'Comp' and 'Label'.

During the Tensorflow session the serialized protocol buffers are loaded using the built in TFrecords reader function. The feature indices ('press', 'temp', etc.) are used to get the specific data from the protocol buffer. The data is then parsed into tensors, which is the data-type used by Tensorflow to

do matrix computations.

3.3 Random Seeding the Model Parameters

The parameters of the network such as the weights are randomly chosen, before the network is optimized to ensure that different optimisation techniques can be objectively compared to one another. The same random seed was used for every run. This means that as long as the network layout is the same, the random initial weights of the network will be the same for all the cases.

3.4 Running the Model

A Python script is used to transfer the data from TFrecords files in batches of a predefined size. The inputs are gathered as tensors and used to train the model that has been developed in chapter 2.

The model runs the complete training dataset *num epochs* times trough the optimisation sequence. The model processes one batch at a time where a batch contains the specified *batch size* number of components.

After each training batch sent to the optimisation sequence, the updated model is validated with a validation batch. Thus for each training epoch the batch loss and batch accuracy for each of the training, validation data sets are recorded and plotted as documented in chapter 4.

Chapter 4

Results

In this chapter the results obtained from the implementation of the machine learning model described in chapter 2, using the data described in chapter 3 are discussed. Multiple runs are done to see how the change of model hyper parameters influences the accuracy of the model. The accuracy and loss of the model is evaluated to see which model layout and hyper parameter set performs best. The influences of the data sample frequency, optimisation algorithm and the RNN-cell type (LSTM or GRU) are investigated. Due to the long training times required to train such a complex model on a large dataset the number of runs were limited. The initial hyper-parameters of the model was selected based on the results obtained in the preliminary study that had been performed. With each run one hyper-perameter such as the number of nodes in a given layer or the optimisation algorithms used is changes, this will enabable one to see how the change of a single hyper-perameter influences the performance of the model.

4.1 RNN Hybrid Network Results

The Hybrid RNN network design as described in chapter 2 is used with the input data collected as described in chapter 3. The results listed in this section has been run on a windows based operating system using Python 3.5 with the Tensorflow 1.2 library. The following hardware setup was used to give the reported results:

Table 4.1: Hardware Setup

CPU	**GPU**	**System Memory**	**HardDrive**
Intel Core i5 @ 3.30GHz	NVIDIA GeForce GTX 1060 6GB, 1280 CUDA Cores	20GB @ 1600MHz	SATA Magnetic Drive

Some of the model parameters were kept constant for all the runs in order to keep consistency between different runs as well as keeping the degree of complexity of the problem lower. The model was trained using the following fixed hyper-parameters, where hyper-parameters refer to parameters that are set manually and are not changed during optimisation of the model:

Table 4.2: Constant Model Hyper Parameters

Sub-sample method	**Learning rate**	**n classes RNN**	**Loss**	**Component features**
max	0.001	1	RMS	37

The max sub sample method means that each sample value extracted from the original sequence is the maximum value experienced since the previous sample was taken. In other words, if the sub sampling rate is one sample per hour, then the sample value will be the maximum value experienced in each hour for the entire sequence.

With each run some hyper parameters were changed to see how the machine learning algorithm performs. The hyper parameters for each of the runs are:

Table 4.3: Model hyper-parameters for different runs

Run	**Sample rate**	**Num Epochs**	**Batch Size**	**RNN hidden nodes**	**layer 2 hidden nodes**	**layer 3 hidden nodes**	**RNN cell Type**	**Optimisation Algorithm**
1	1 hour	1457	64	32	64	0	LSTM	ADAGRAD
2	1 hour	2478	64	64	64	0	LSTM	ADAGRAD
3	1 Day	3513	64	64	64	0	LSTM	ADAGRAD
4	1 Day	1787	512	32	64	0	LSTM	ADAGRAD
5	1 Day	1787	512	64	256	0	LSTM	ADAGRAD
6	10 min	2041	16	32	256	0	LSTM	ADAGRAD
7	1 hour	1712	128	32	256	0	LSTM	ADAGRAD
8	1 hour	1712	128	32	256	0	GRU	ADAGRAD
9	1 hour	2190	128	32	256	0	LSTM	GD
10	1 hour	2133	128	32	256	32	LSTM	ADAGRAD
11 [(1)]	1 hour	1640	128	32	256	0	LSTM	ADAGRAD

elevation survey data padded with zero values[(1)]

A total of 11 runs were completed. The loss diagrams off all 11 runs are shown in Figure 4.1. Significant smoothing is applied to see the general trend of the loss as some training curves are irregular.

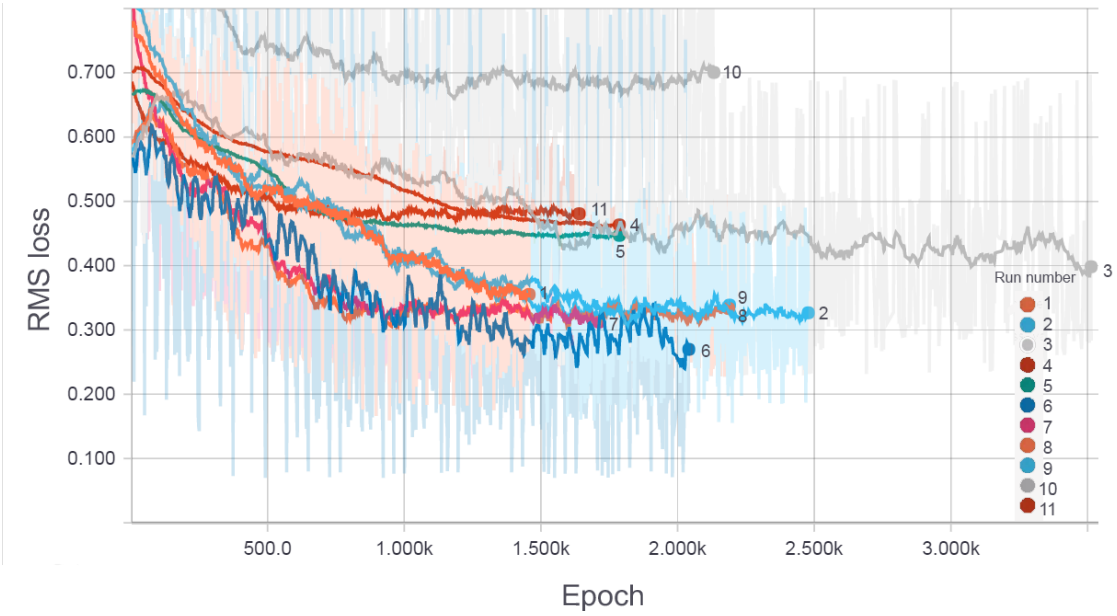


Figure 4.1: Training Loss

The accuracy of the model (as defined in Equation 2.7) is expected to increase as the loss decreases. The accuracy diagrams of the corresponding 11 runs are shown in Figure 4.2.

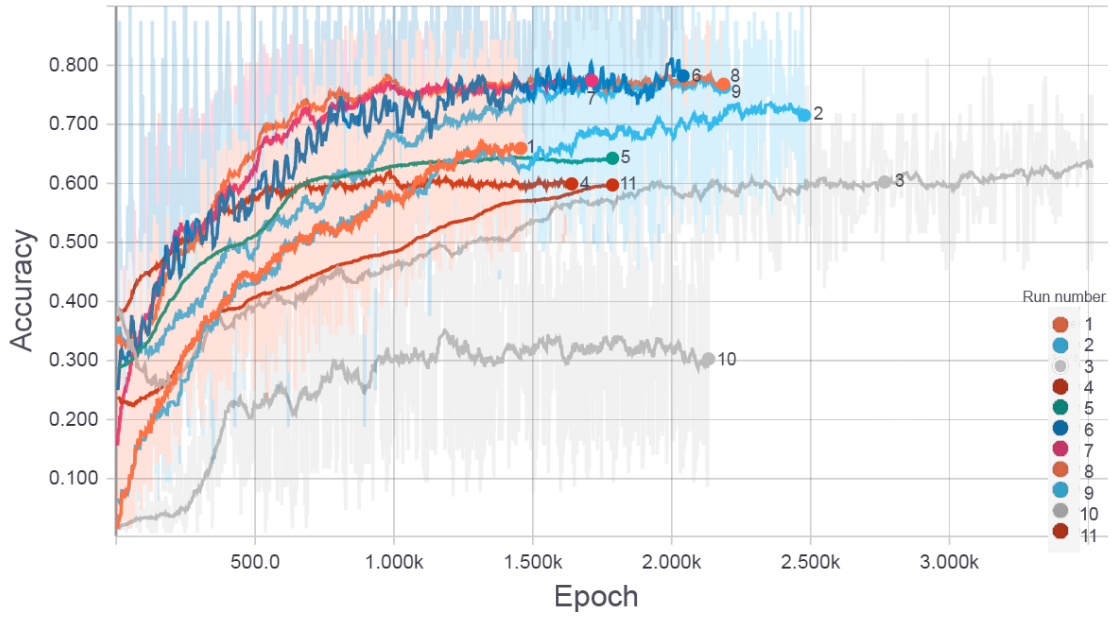


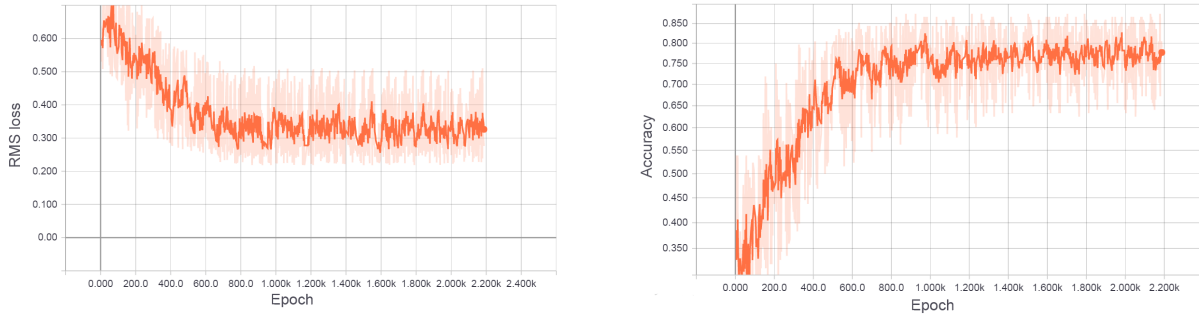
Figure 4.2: Training Accuracy

The results for each of the training runs are shown in Table 4.4

Table 4.4: Results

Run number	**Loss_train**	**Loss_valid**	**Acc_train**	**Acc_valid**	**Running time [hours]**
1	0.377	0.600	67.5%	49.9%	14.4
2	0.331	0.491	71.0%	52.9%	24.1
3	0.400	0.430	63.3%	56.2%	1.7
4	0.456	0.447	60.0%	52.1%	3.0
5	0.448	0.386	64.7%	64.3%	2.4
6	0.240	0.413	77.4%	52.9%	103.4
7	0.273	0.493	77.2%	59.9%	21.0
8	0.264	0.473	79.0%	62.5%	18.9
9	0.340	0.597	76.3%	58.2%	25.1
10	0.647	0.641	36.5%	32.0%	25.0
11	0.481	0.561	52.7%	49.4%	20.2

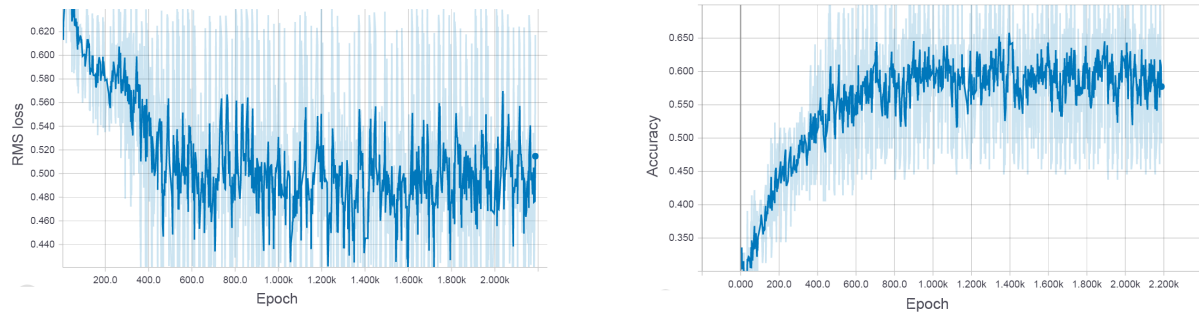
For each run the train, test and validation datasets were evaluated to see how the model performs. The loss and accuracy curves for run number 8 is shown in Figure 4.3 to Figure 4.4



(a) Training Loss

(b) Training Accuracy

Figure 4.3: Training Curves



(a) Validation loss

(b) Validation Accuracy

Figure 4.4: Validation Curves

The training loss curve initially decreases and finally steadies out after about 800 epochs. It is clear that the machine learning algorithm is able to recognise patterns in the data and accuracy is improving as the epochs are increasing.

When the temporal dataset length gets too long the GPU (Graphics Processing Unit) memory is too small and the program runs into an *out of memory* error. To get around this problem the batch size can be reduced. This causes an increase in training time as fewer parallel computations

can be done by the GPU. When the length of the temporal datasets is decreased, the batch size can be increased meaning that Tensorflow can make more effective use of the parallel computation capabilities of the GPU.

There does not seem to be a noticeable difference in performance between the GRU and LSTM cell, as is the case with the investigation done in Appendix A. The difference is that there are only five temporal data-sets in the training set, compared to the 100 temporal datasets used in Appendix A and the temporal set is also longer. This means that the model cannot extract the optimal amount of information from the given temporal datasets, this should improve if a larger training dataset is used. Reducing the sampling rate has a minor influence on the accuracy of the model but has a major influence on the required training time.

4.2 Learning Rate

The learning rate (ϵ) is kept constant for all runs at 0.001 as this has been found to provide a stable decrease in loss during training when the batch size is large. When the learning rate is increased the decrease in loss is not stable and the optimisation algorithms struggle to minimise the loss because the changes to the model parameters are not subtle enough. When the learning rate is too small, the changes in the parameters are not large enough to have a significant influence in the model output. This means that the learning rate needs to be chosen carefully for each problem. If computational time and computational power had not been a limiting factor multiple runs could be done to completely optimize the parameters that define the model such as learning rate, activation functions used and the model layout. However, even with modern day computational power this is still not feasible and a lot of research is done with respect to hyper-parameter optimisation techniques.

4.3 Network Layout

The network layout is constrained by the computational power that is available. While conducting the research with results tabulated in Table 4.4, it was found that a RNN with output size of 32 and a fully connected feed forward network with a (38-256-1) configuration is computationally stable enough to run on a 6gb GPU. If the network size is increased the model would not run on the computer set up that was used as it would overfill the GPU memory when a sample of the temporal sequence sample rate was every 10 minutes. When increasing the network size, additional steps should be taken to ensure the GPU memory is not overfilled. It was also found that when the model complexity is increased there is no noticeable increase in prediction accuracy.

4.4 Optimisation Algorithm

Adaptive gradient descent performs better than normal gradient descent and is able to converge to a mature model in a shorter time. This is partially because a hybrid model is used and having an

adaptive gradient descent is a large advantage.

In general both the gradient descent and adaptive gradient descent algorithms struggle to over-fit to the training data. The ADAM-optimiser is able to fit to the training data with a very low loss, this is a good sign that the model is able to pick up patterns within the data. The validation error is higher than that of the training error. This is mostly because the training dataset is not large enough or the information contained in the input data is not enough to make an accurate prediction.

4.5 Training Time

The training time is largely influenced by the sample rate of the temporal data. There is an exponential increase in training time when the temporal sequence sub-sampling rate is less than ten minutes per sample. However, there not not significant decrease in training loss as seen in Figure 4.5

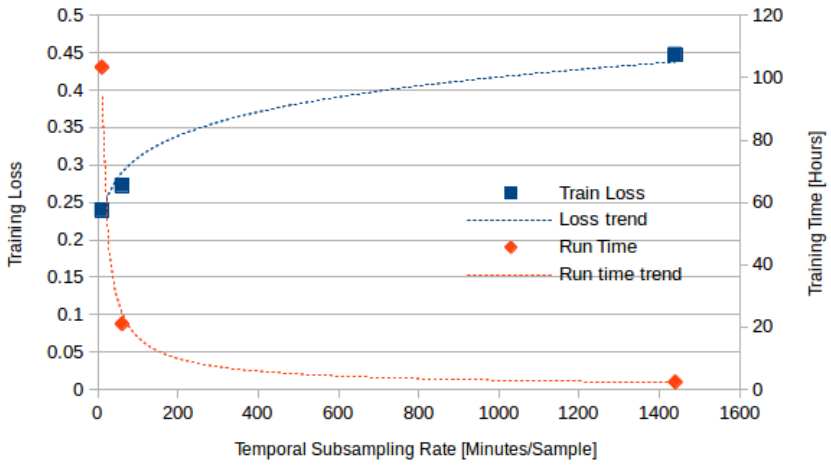


Figure 4.5: Temporal Sub-sampling rate influence

The Tensorflow package has the ability to do some the computations in parallel. Running the computation on a TPU or Tensorflow processing unit will decrease the required computation time. A normal GPU could be used to carry out the required computations. Figure 4.6 shows the percentage of the operations that are compatible with TPUs. This model has an overall compatibility of 75 %, meaning that there is room for improvement. This however does not fall within the scope of this research.

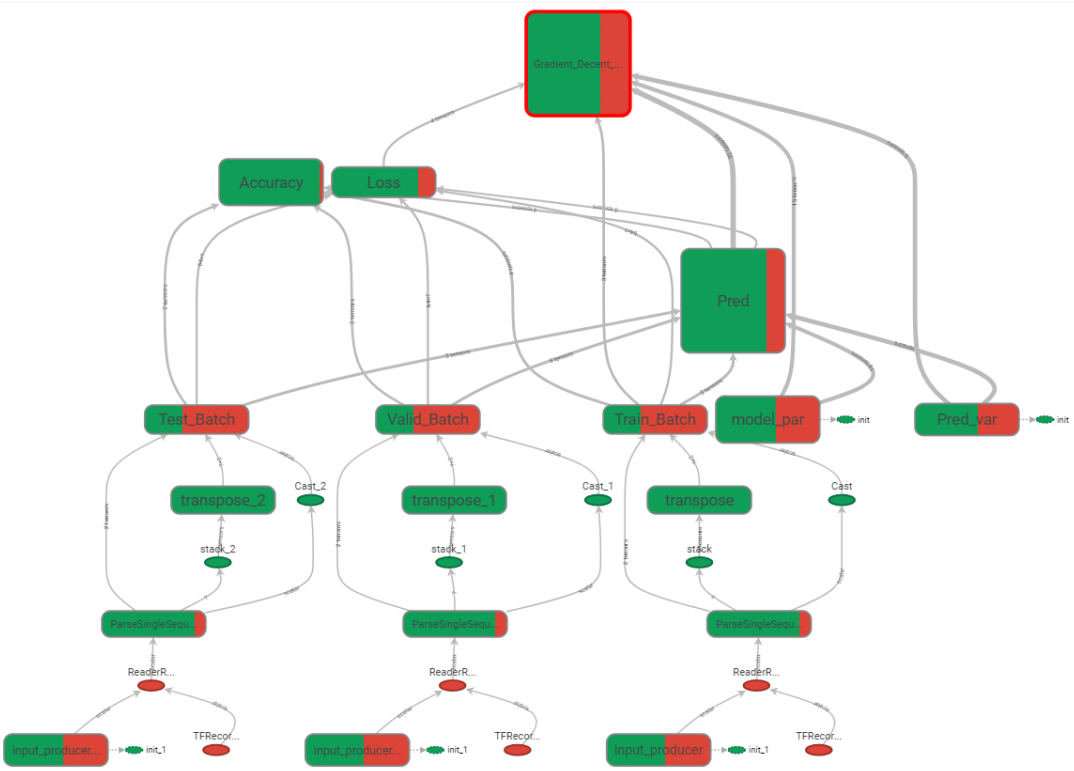


Figure 4.6: TPU Compatibility

4.6 LSTM cell vs GRU cell

In this research most of the runs are performed using the LSTM cell as the results of the preliminary indicated that the LSTM cell performed better, and is a more well known cell, that has been proven to work well with different types of datasets. The results obtain in this research indicates that this is not the case that LSTM outperforms the GRU. The GRU cell reaches a saturated state during training earlier than the LSTM-cell. This is a state where there are no more upward or downward trends in the loss of any datasets. This means that the model is not over-fitting to the training data, however it is as there are fewer parameters that need to be optimised. Unlike the findings in Appendix A, the LSTM-cell does not outperform the GRU-cell. It is evident that the extra complexity offered by the LSTM-cell does not help when working with such long time sequences and small sample size.

4.7 Elevation Survey Impact

The impact of adding the elevation survey data is evaluated by running the model with the elevation survey data as an input and then repeating the model where all the elevation survey data input is 0. This ensures that all the model parameters are the same at the start of the optimisation and the optimisation difficulty is the same.

The results show that the model struggles to ignore the padded inputs. However, removing the inputs all together, the overall accuracy of the model seems to be around 3% lower. This indicates

that in the case where we have this data, including the data from the elevation survey does add value to the model, however this is marginal.

It can be noted that the model with reduced inputs trains significantly faster due to the reduction in model complexity and converges to a solution much earlier.

4.8 Best Runs

Due to the high number of class 4 elements in the training data there exists a local minimum point where the model always predicts a low damage class four elements. This means that the model does not really fit to the training data but is able to achieve a low loss with good accuracy by just predicting class 4 components every time. This is not ideal for the given data set because we are more interested in components with high damage in order to perform the required maintenance activities on those components. Normal GD and ADAGRAD optimisation seems to struggle with this. Using the built in ADAM-optimizer function in Tensorflow alleviates this problem as the model is able to fit the training data very well.

It was found that using this approach the model is able to obtain a very high training accuracy of 94.1%. This means it is very good at classifying a set of input data that it has seen before. However when it is provided a new set of input data, it only achieves a classification accuracy of 50.1%. This could mean that the training dataset is not large enough, or there is insufficient information contained in the input data or that there is a very low correlation between the measured inputs and the damage of a component. Another run was performed increasing the size of the training dataset, from 627 components to 948, which increased the validation accuracy to 56.1%

When a different random training set is used a validation accuracy of 64% can be achieved. This however cannot be benchmarked against the other runs because in real world application you would not have the opportunity to choose a good training set that will yield better validation accuracies.

The runs performed has the same hyper parameters as previous runs with changes as noted in Table 4.5

Table 4.5: Best Runs Hyper Parameters

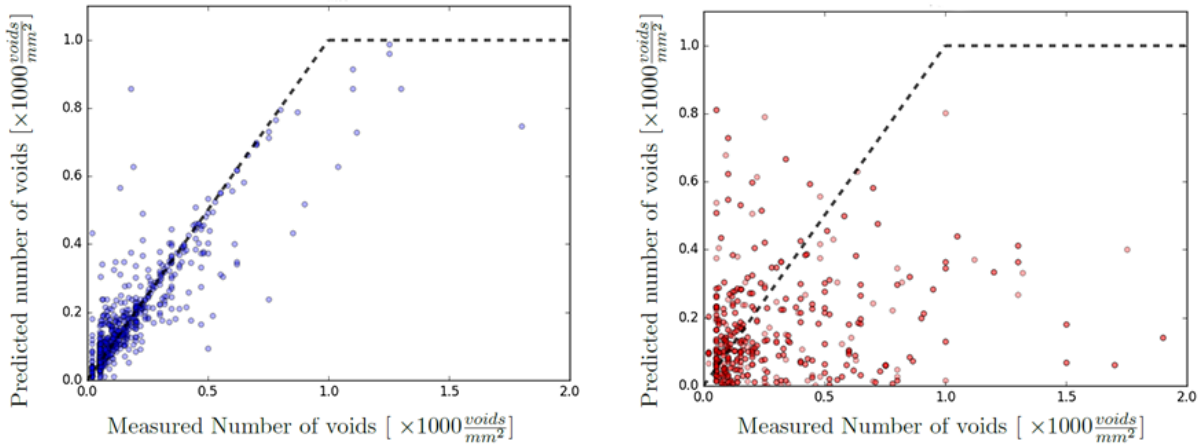
Run Number	**Sample Rate**	**Number Epochs**	**Batch Size**	**RNN Hidden Nodes**	**Layer_2 Hidden Nodes**	**Layer_3 Hidden Nodes**	**RNN Cell**	**Optimization Algorithm**
13	6H	1540	625	1	125	64	LSTM	ADAM
14	6H	1540	500	1	125	64	GRU	ADAM

The results for the two runs are listed in Table 4.6

Table 4.6: Best Runs Results

Run number	Loss train	Loss valid	Accuracy train	Accuracy valid	Running time
13	0.132	0.407	94.1%	50.1%	5.3
14	0.099	0.403	90.8%	56.1%	5.1

Figure 4.7 shows the predicted number of voids compared to the actual measured number of voids. The dotted line indicates an ideal line where the predicted number of voids is exactly the same as the measured number of voids. From this figure it is clear that given the complexity of the problem the model fits the training data very well. However, when the model is given a new validation dataset the fit is not as good. It would seem that the model has over-fitted to the training data which is not the case, because Figure 4.11 shows a monotonic decrease on loss on the training dataset. It means that the model is getting better at predicting the damage of the components. An over-fit or bias towards a training dataset would cause the validation loss to dramatically increase which is not the case.



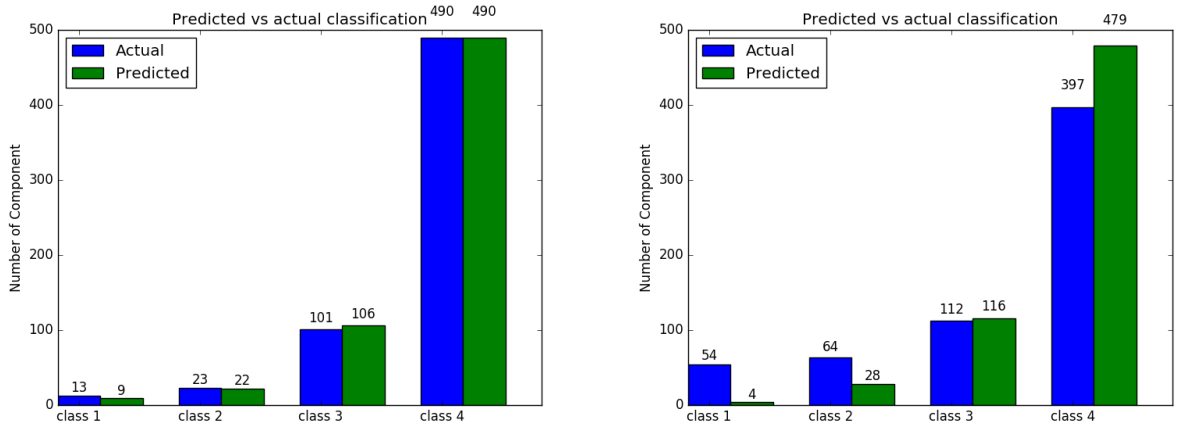
(a) Training data fit

(b) Validation data fit

Figure 4.7: Validation Curves

Figure 4.8 compares the number of components in each of the classes for the model predictions and the actual measured values. Note that even though the model does not have a very accurate fit to the validation data, the distribution of the number of components in each of the classes for the validation set is surprisingly accurate.

The confusion matrix (Figure 4.9) highlights the areas where the model has misclassified the most components in the training data set. This shows that the model misclassifies a lot of class four components as class three and vice versa. It also shows that the model classifies some class one components as class two, but for the given batch none of the class two components are misclassified as a class one. The model seems to predict the damage slightly lower than the actual damage on average.



(a) Training data classes

(b) Validation data classes

Figure 4.8: Measured vs Predicted Classification

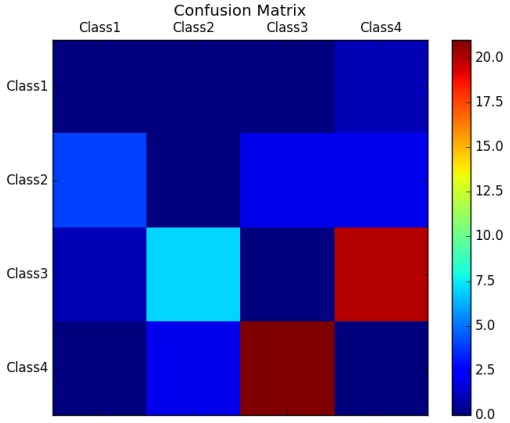
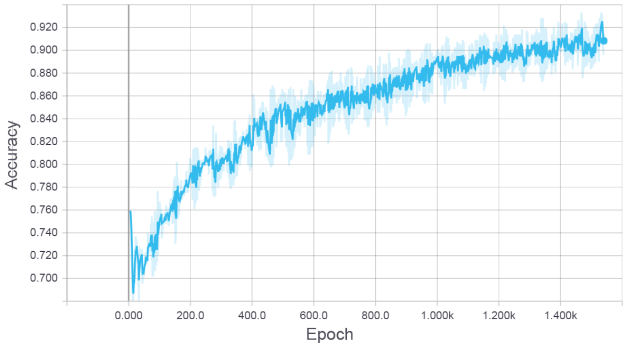
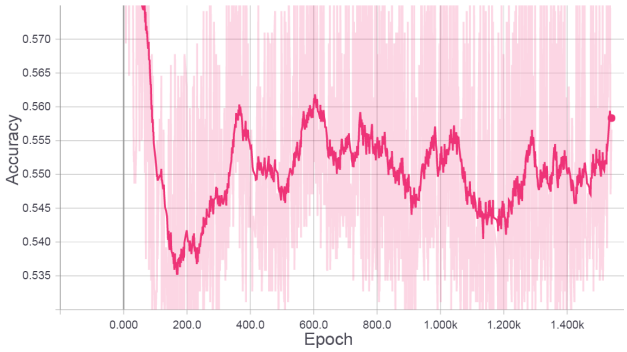


Figure 4.9: Training Set Confusion Matrix

Figure 4.10 and Figure 4.11 illustrate the training curves for the training and validation set. It is clear that the training accuracy trends upward while the validation accuracy does not follow any trend. The loss curves shows a clear downward trend in both the training and validation set. This means that there are some patterns contained in the training data that can be used to improve predictions about the component damage of a different dataset. The fact that the validation error decreases is a very good sign, and makes these methods very promising. Keep in mind that the accuracy is an arbitrary measure and a decreasing accuracy does not necessarily mean that the model becomes worse at making predictions.

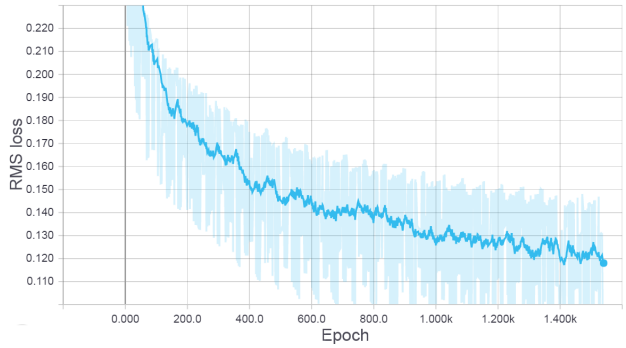


(a) Best Run Training Accuracy

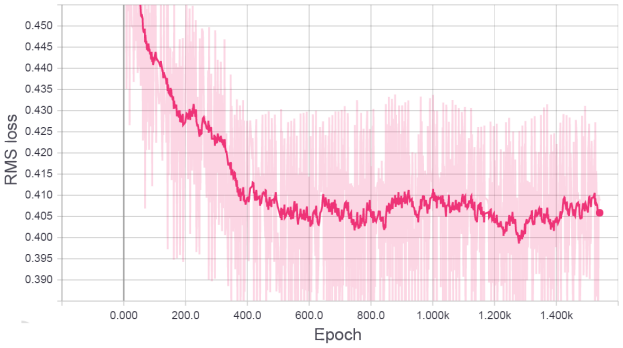


(b) Best Run Validation Accuracy

Figure 4.10: Best Run Accuracy



(a) Best Run Training Loss



(b) Best Run Validation Loss

Figure 4.11: Best Run Loss

Chapter 5

Conclusion

Historically temporal/sequential data sequences such as temperature-time data sequences, are notoriously difficult to analyse and perform pattern recognition. The LSTM cell based Recurrent Neural Network has been proven to be able to perform pattern recognition successfully on these sequences. The flexibility that the hybrid-RNN model offers, makes it ideal for application to real world data.

The hybrid RNN is able to recognize patterns within the data, by using the temperature and pressure signals as an input, as well as previous damage and elevation survey data. The model is able to predict damage in a component with relatively high accuracy. Despite the accuracies that have been achieved the model does struggle to predict component that goes from low damage to high damage classes within a short period of operating time.

GD, ADAGRAD and ADAM back-propagation algorithms can be used to optimize the parameters of the machine learning model. The use of ADAM is clearly superior to the other algorithms as the algorithm is more stable and convergence is reached earlier. This is especially true in the case of the hybrid recurrent neural network model.

With the randomly generated data, a training accuracy of 90.6% was obtained on a training set and 56.1% was obtained in a validation set. On a different training dataset validation accuracies as high as 64% have been achieved.

It was noted that adding additional information to the model such as elevation survey information does improve the model accuracy by around 3%. It was noted there is a significant reduction in training time due to the decrease in model complexity. The user is advised to take this into consideration to determine if the extra effort used to collect the data is worth the increase in model prediction accuracy.

In order to improve the accuracy of the model more frequent training samples need to be taken, thus if the piping support movement as well as the creep void formation could be tracked in real time it would potentially increase the model accuracy significantly.

At this stage a large amount of computational power is needed to parse all the information that

is available within a fleet of operating units. It has been shown that when the size of the training data is increased from 627 to 948 components there is a 6% increase in validation accuracy. Thus it can be expected that the model should get better at predictions as the size of the dataset grows.

Although the model is not able to predict all the damage of components with high levels of accuracy, this does prove to be a very promising method of parsing large and complex data sets. The model will not be able to identify all the class one components in a system. It can however list components that are at high risk, based on the current data provided and previous experiences.

The model has proven to be a promising tool that could be used to create inspection plans. Even if the classification accuracy is only 56.1% this would be a major advantage over guessing where damage components might be. Randomly guessing the classification of a component should theoretically yield a classification accuracy of 25% given a uniform distribution throughout the classes, which is not the case, meaning that an even lower accuracy is to be expected when randomly guessing the classification of a component. The model has also been proven to be able to predict the distribution of components within each of the classes. This would also give the user a good idea of the damage within the system and the overall risk profile of a given system. This tool can be used to double check that all inspections plans do include the possible high risk "class 1 and class 2" components. Even if the location of the high risk components are not known, having an accurate estimation of the distribution of components in the damage classes will assists with ordering spare parts in time because a lot of components are interchangeable.

The hybrid-RNN is a very promising tool that can be applied to similar data driven problems. The problem in this research is a very difficult one as shown with the sensitivity analysis in section 1.5.12, and is expected to do even better with other problems. The method used in this research is expected do well with problems where plenty of historical data is available and where creating a physics based model is too complex. The hybrid-RNN is scalable and configurable to work with most types of data driven problems.

5.1 Further Research and Recommendations

Further research will potentially include looking at the networks proposed by Yao et al. (2015) as it adds further complexity to the LSTM cell by adding a depth gate that connects the memory cells of adjacent layers. Depth-Gated Recurrent Neural Networks have proven to increase accuracy of language modelling and could potentially increase the accuracy of the current model.

The recent success with convolutional neural nets has proven to be a promising pattern recognition tool (Deng, Hinton and Kingsbury, 2013). The convolutional neural net has outperformed all other machine learning models on multiple platforms such as speech recognition, image recognition as well as playing the game of *GO*, which is regarded by some to be one of the most challenging pattern recognition tasks attempted.

In order to address the out of memory problem when working with long temporal datasets in a RNN, the temporal data should be split into smaller mini batches, and the state of the LSTM cell must be saved, and reloaded if the next mini batch continues from the previous mini-batch. This

would potentially have a massive impact on the computational time and would remove the current limit on the length of the temporal sequences that can be loaded into the GPU memory.

Methods should be researched on how to increase the TPU compatibility of the Tensorflow graphs. This should aid in decreased training times of the models.

Further research should include optimizing the model hyper parameters, i.e. the number of hidden nodes, the number of hidden layers used, the optimisation algorithms used, etc. This could be done generating a latin hypercube sample space of the model hyper parameters and doing multiple runs to see which model configuration yields the best results (Stein, 2012). This would however require a lot of computation time.

If more data is available the model could be trained specifically on components with high damage. This would ensure that the model will specialise in identifying components with high damage. Due to the fact that components with high damage is in the minority, the influence of the these components on the model parameters is small compared to components with low damage. This is not ideal because for the given data set it is much more important to identify components with high damage than it is to identify components with low damage.

It is recommended that a data capturing standard be created for all future data collection where these methods are to be applied. Standardizing the way in which data is stored will make automated data reading easier. When data is saved in tabular formats such as Microsoft Excel format, one should also refrain from entering multiple values in a single cell, which will make data reading much easier.

To truly benchmark the model, it is recommended that the model predictions are compared to that of an experienced system engineer and compared with one another.

References

Bayer, J. and Osendorfer, C. (2015), 'Learning Stochastic Recurrent Networks', *ICLR Confenerence* pp. 1–9.

Bengio, Y., Simard, P., Frasconi, P. and Member, S. (1994), 'Learning Long-Term Dependencies with Gradient Descent is Difficult', **5**(2).

Bishop, C. M. (2006), *Pattern Recognition and Machine Learning*, Vol. 4, Springer, 2006, Singapore.

Cane, B. J. (1982), 'Remaining creep life estimation by strain assessment on plant', *International Journal of Pressure Vessels and Piping* **10**(1), 11–30.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y. (2014), 'Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation', *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* pp. 1724–1734.

Chung, J., Kastner, K., Dinh, L., Goel, K., Couville, A. and Bengio, Y. (2015), 'A Recurrent Latent Variable Model for Sequential Data', *Nips* pp. 1–9.

Coetzee, J. L. (1998), *Maintenance*, Maintenance Publishers, Hatfield.

De Boer, P. T., Kroese, D. P., Mannor, S. and Rubinstein, R. Y. (2005), 'A Tutorial on the Cross-Entropy Method', *Annals of Operations Research* **134**, 19–67.

Deng, L., Hinton, G. E. and Kingsbury, B. (2013), 'New types of deep neural network learning for speech recognition and related applications: An overview', *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* pp. 8599–8603.

Deschanels, H., Escaravage, C., Thiry, J. M., Le Mat Hamata, N. and Colantoni, D. (2006), 'Assessment of industrial components in high temperature plant using the "ALIAS-HIDA" - A case study', *Engineering Failure Analysis* **13**(5), 767–779.

Duchi, J., Hazan, E. and Singer, Y. (2011), 'Adaptive Subgradient Methods for Online Learning and Stochastic Optimization', *Journal of Machine Learning Research* **12**, 2121–2159.

Duda, R. O., Hart, P. E. and Stork, D. G. (2001), *Pattern Classification*, edition 2 edn, John Willey and Sons, New York.

Ebrahimi, O. H., Esmaeili, M., Oskouei, A., Mirhadizadehd, S. and Tse, P. (2016), 'Defect detection of helical gears based on time-frequency analysis and using multi-layer fusion network Defect detection', *Nondestructive Testing and Evaluation* (November), 1–18.

ECCC (2005), 'ECCC Data Sheets 2005', *ETD* **47**(2).

Hochreiter, S. and Schmidhuber, J. (1997), 'Long Short-Term Memory', *Neural Computation* **9**(8), 1735–1780.

Igel, C. and Hüsken, M. (2000), 'Improving the Rprop learning algorithm', *Proceedings of the Second International Symposium on Neural Computation* pp. 115–121.

Jacobs, R. A. (1988), 'Increased rates of convergence through learning rate adaptation', *Neural Networks* **1**(4), 295–307.

Koehn, P. (1994), 'Combining Genetic Algorithms and Neural Networks : The Encoding Problem', *The University of Tennessee,* (December), 1–67.

Mao, J. and Jain, A. K. (1996), 'Why artificial neural networks?', *Communications* **29**, 31–44.

Mishra, S. and Savarkar, S. (2012), 'Image Compression Using Neural Network', *Proceedings of International Conference and Workshop on Emerging Trends in Technology (ICWET)* **3**(2), 18–21.

Olah, C. (2015), 'Understanding LSTM Networks'.
URL: *http://colah.github.io*

Sourmail, T., Bhadeshia, H. K. D. H. and Mackay, D. J. C. (2002), 'Neural network model of creep strength of austenitic stainless steels', *Material Science and Technology* **18**(June), 655–663.

Stein, M. (2012), 'Large Sample Properties of Simulations Using Latin Hypercube Sampling', *Technometrics* **29**(January 2014), 37–41.

Sutton, R. S. (1992), 'Adapting Bias by Gradient Descent: An Incremental Version of Delta-Bar-Delta', *Proceedings of the Tenth National Conference on Artificial Intelligence* pp. 171–176.

Tian, Z., Wong, L. and Safaei, N. (2010), 'A neural network approach for remaining useful life prediction utilizing both failure and suspension histories', *Mechanical Systems and Signal Processing* **24**(5), 1542–1555.

Van Zyl, F. H., Von dem Bongart, G., Bezuidenhout, M. E. J., Doubell, P., Havinga, F. C., Pegler, D. A. H., Newby, M. and Smit, W. (2005), 'Life Assessment and Creep Damage Monitoring of High Temperature Pressure Components in South Africa's Power Plant', *ECCC Creep Conference* (September), 934–945.

Venkatesh, V. and Rack, H. (1999), 'A neural network approach to elevated temperature creep–fatigue life prediction', *International Journal of Fatigue* **21**(3), 225–234.

Yam, R., Tse, P., Li, L. and Tu, P. (2001), 'Intelligent Predictive Decision Support System for Condition-Based Maintenance', *Advanced Manufacturing Technology* (17), 383–391.

Yao, K., Cohn, T., Vylomova, K., Duh, K. and Dyer, C. (2015), 'Depth-Gated LSTM', *Microsoft Research* pp. 1–5.

Appendix A

RNN trained on a creep damage model

In this section an RNN is trained on a creep damage model. This is done to see how well the model would perform if the problem consisted purely of a temporal dataset, with life fraction consumed as an output and all other external factors that could influence the remaining life of a component ignored. Artificial noise is added to the data to see if the model is able to "ignore" the noise in the data and recognise the underlying patterns in the data.

In order to train a model a training set of data is needed. A training dataset consists of an input vector ($\vec{x}$) with a corresponding function value or label ($\vec{y}$). In the preliminary models the function value ($\vec{y}$) may be either represented by a real scalar value or an *one-hot* encoding vector.

A.1 Generating the Data Sets

Random data sets are generated following the same methods described in subsection 1.5.9. The datasets are randomly generated using a Python script. Each random dataset is labelled using Equation A.1.

With the classification problem the labels are represented by a one hot encoding vector e.g. [0, 0, 0, 1]. In this case a one hot encoding vector means that four bins are created in which every input vector can be classified. The correct bin is denoted by "1" and the other bins are denoted by the "0".

The labels $\vec{y}$ is calculated using Equation A.1.

$$\vec{y} = \%\text{ life used} = \sum_{j=1}^{N} \frac{\Delta t_j}{t} \tag{A.1}$$

where N is the length of the input vector ($\vec{x}$), Δt the time between data points in the input vector ($\vec{x}$). The components calculated creep rupture life (t) is calculated using Equation 1.41, while the input variables (T,P,σ) are assumed to be constant throughout Δt.

To ensure that the data set is more representative of a real world data set, Gaussian noise is added to the generated labels. This is done in order to create a model that will be more applicable to a real life data set where the actual damage measured on a component is likely to be in some confidence bound of the calculated damage on the same component. Gaussian noise is added using Equation A.2.

$$\vec{y}_{noise} = \vec{y} \times \mathcal{N}(\mu = 1, \sigma = 0.4) \tag{A.2}$$

where: $\vec{y}_{noise}$ is the associated life fraction label with added noise. $\vec{y}$ is the associated life fraction as calculated using Equation A.1 and $\mathcal{N}(\mu = 1, \sigma = 0.4)$ is the noise multiplication factor sampled from the distribution shown in Figure A.1.

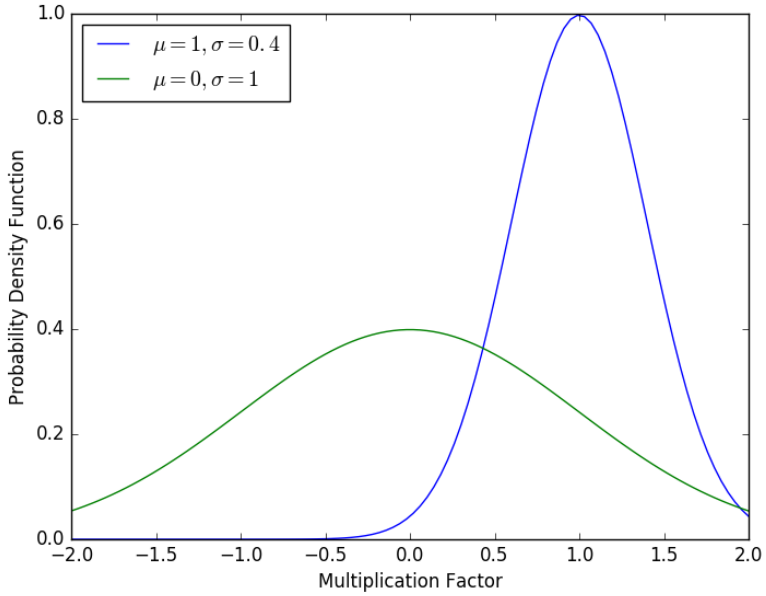


Figure A.1: Gaussian Noise Distribution

The noise distribution is chosen so that the mean of the distribution is unit. Thus if enough training sequences are generated the algorithm should be able to ignore the effects of noise and should be able to predict the most probable life fraction used quite accurately. Actual real plant data will not follow such a clean distribution. By adding Gaussian white noise to the training labels, a more realistic dataset is generated and ensures that the machine learning model is able to lessen the effect of noise in real sequential data sets on the prediction model output.

A.2 Sequence Learning Models Used to Classify Life Fraction Consumed by Temperature Sequences

The generated data sets are divided into training, test and validation set as per section 1.5.9. The model is trained using the training data set. The validation set is used to monitor the loss or accuracy of the model after each training iteration. If the validation loss starts to increase while the training loss is still decreasing, it means that the model is starting to over-fit to the training data and the training needs to stop. The test set is used as a separate untainted data set to test how the model performs once the training is completed.

Table A.1: Model parameters

Parameter	Value	Comment
$\vec{x}$	variable	Temperature signal
Δt	20000	Time at which a temperature signal stays constant
batch size	64	The number of datasets $\vec{x}$ and corresponding $\vec{y}_{noise}$ used within one epoch
n_{hidden}	64	Number of nodes in hidden layer
ϵ	0.001	Learning rate
training iterations	500000	Number of time gradient descent and weight updates are applied

A.3 Tensorflow Graph

Figure A.2 illustrates the tensor graph that is generated by Tensorflow. The model takes an input variable $\vec{x}$ the output of the model is compared to the actual label $\vec{y}$. The loss (error) is calculated and used by the specified stochastic gradient descent algorithm (ADAGRAD in this case) to adjust the model parameters. The stochastic gradient descent algorithm is used to make adjustments to the RNN cell (basic LSTM cell in this case). The model parameters is updated and the cycle is repeated for each batch of training data. Note that the thickness of connecting lines gives an indication of how many computations can be done in parallel.

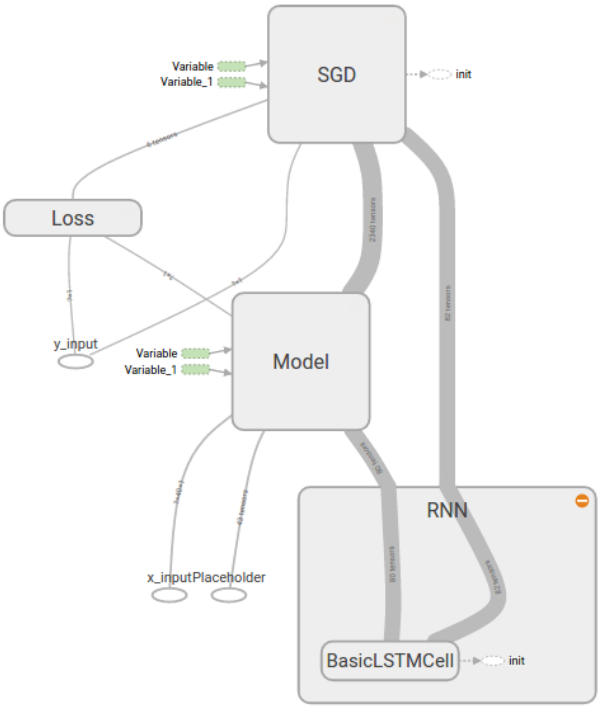


Figure A.2: Level 1 Tensor Graph for Basic LSTM Regression Model

A.3.1 Classification

If the data is treated as a classification problem the training labels ($\vec{y}_{noise}$) that are used by the machine learning model is converted to one hot encoding vectors as specified in Table 2.2. The machine learning model calculates a pseudo-probability of each of the bins being the correct label for a given input vector ($\vec{x}$). A typical predicted value ($\vec{y'}$) with a softmax classifier could be [0.05, 0.1, 0.3, 0.65]. The $argmax(\vec{y'})$ is used to determine the predicted class. In this case the one hot encoding will be [0,0,0,1]. Thus the model will predict class 4.

The generated data ($\vec{x}$) is used to train the classification model. The generated temperature signals are scaled using min-max scaling described in section 1.5.9.

Figure A.3 shows four sampled signals from the training dataset and four sample signals from the test dataset. The actual signal label ($\vec{y}$) as well as the predicted signal label without a $softmax$ and $argmax$ filters ($\vec{y'}$) is displayed.

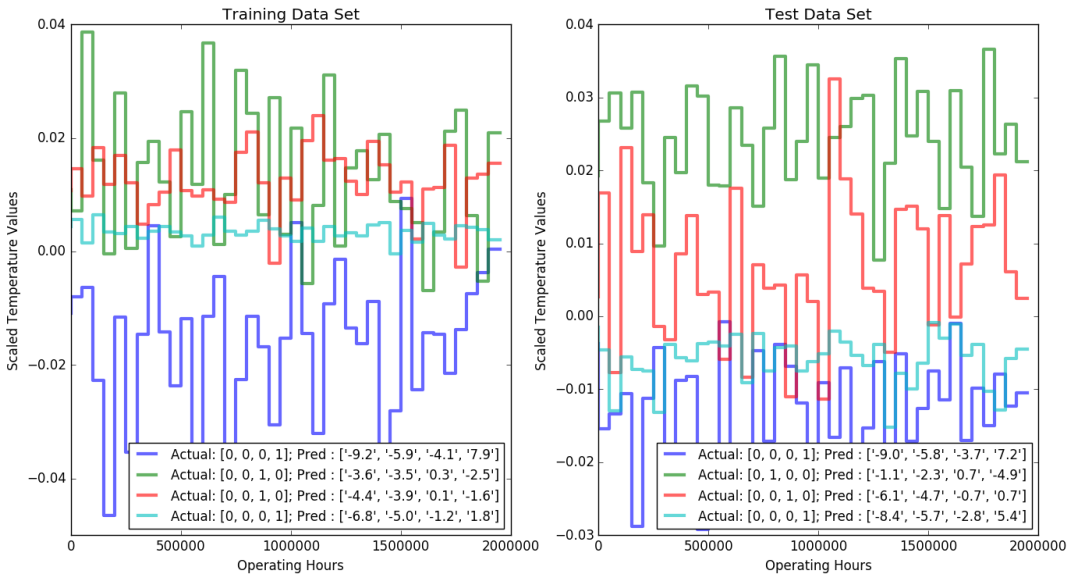


Figure A.3: Classification Temperature Signals

The prediction values of a given signal is set of four arbitrary values which could be seen as a pseudo probability distribution, where the bin with the highest positive value is the predicted classification for the given component.

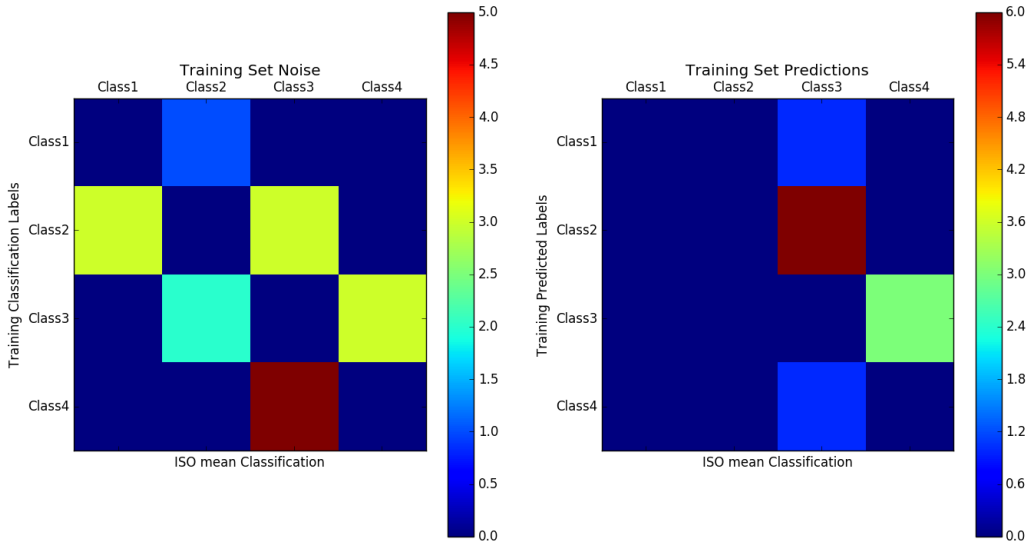


Figure A.4: Training Dataset Confusion Matrix

Figure A.4 shows the confusion matrix for the classification case on the training data set. The size of the training data set is 100 data sets .The confusion matrix shows that six classification labels were incorrectly classified to class2 instead of class3. The figure on the left gives an indication of the noise that was added to the training data set. The figure on the right gives an indication of how many of the training datasets the model will classify incorrectly.

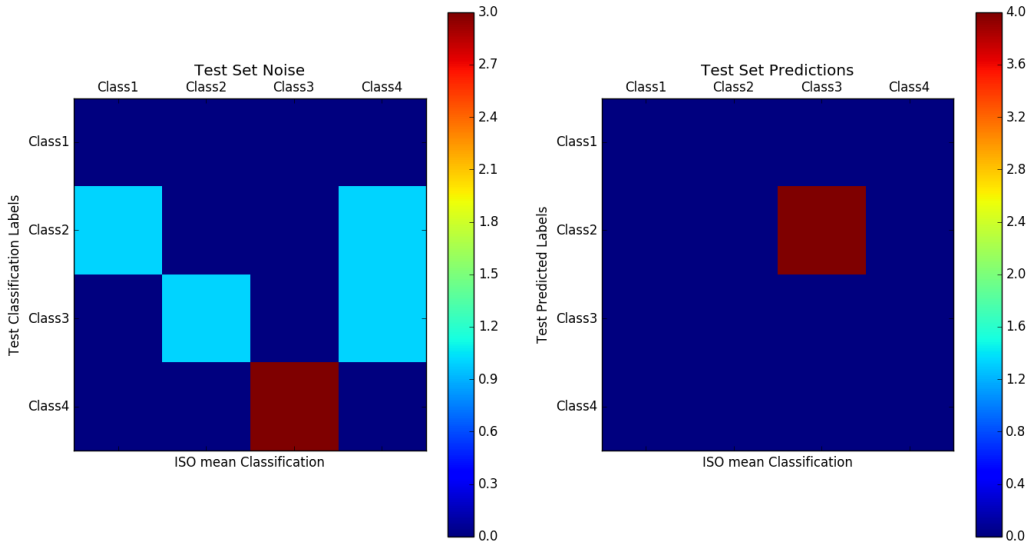


Figure A.5: Testing Dataset Confusion Matrix

Figure A.5 shows the confusion matrix for the classification case on the testing data set. The size of the training sata set is 40 data sets. The confusion matrix shows that four class 3 datasets were incorrectly classified as class 2 instead of class 3. The figure on the left gives an indication of the noise that was added to the testing data set. The figure on the right gives an indication of how many of the testing datasets the model will classify incorrectly.

A.3.2 Classification Training Curves

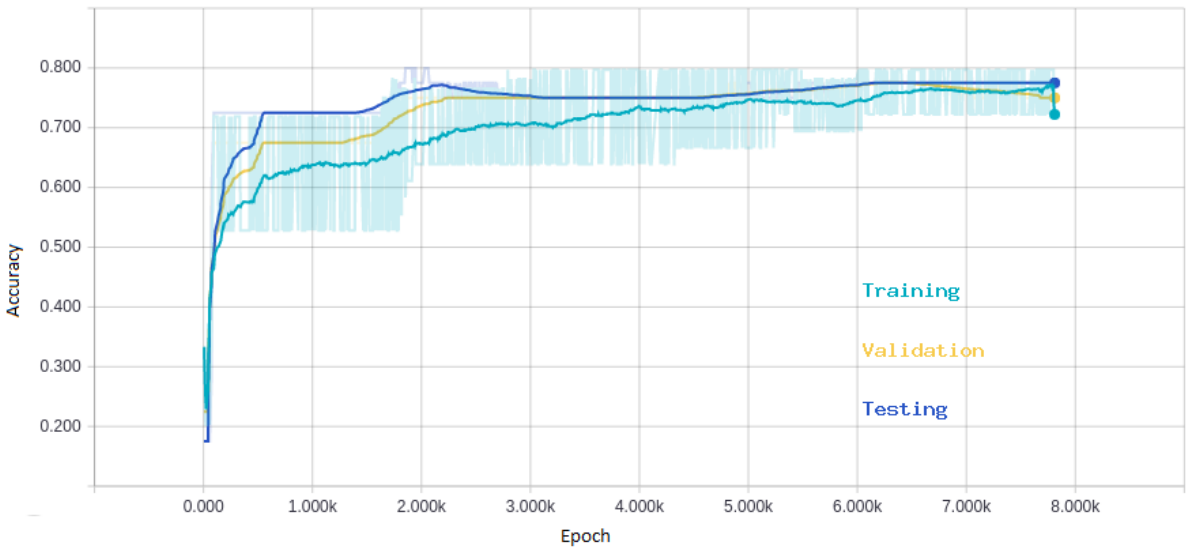


Figure A.6: Accuracy GRU Classification ADAGRAD (iter = 500k)

Figure A.6 shows the accuracy training curves for the training validation and test curve. The time axis shows the number of calculation iterations. The data is processed in batches of 64 iteration

to allow for parallel processing.Thus 500k iterations equates to 7813 calculation iterations/epochs. Smoothing is applied to the curves to better track the overall trend. The ADAGRAD algorithm very efficiently increases the accuracy of the machine learning GRU model within a relatively short period of time. The training curve shows a general upward trend until the model is stopped after 7813 epochs. The validation curve does not decrease sufficiently to indicate that the model is over-fitting to the training data. Given that the training, testing and validation accuracies is within close proximity of one another. It can be said that the machine learning algorithm is matured and little to no improvement is expected with additional training of the model.

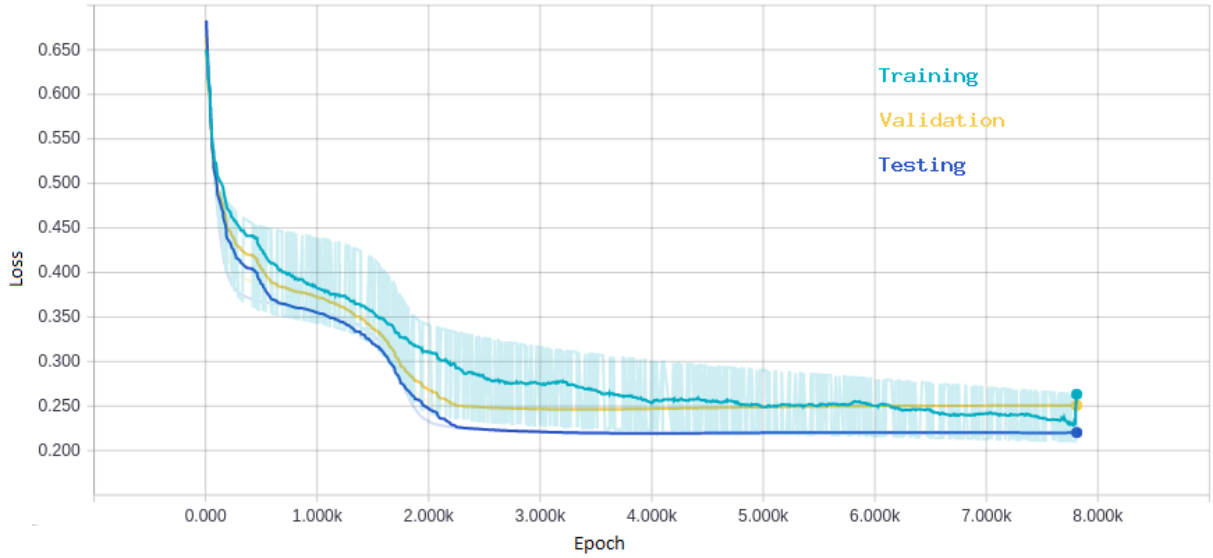


Figure A.7: Loss GRU Classification ADAGRAD (iter = 500k)

Figure A.7 shows the loss training curves. At around 6000 epochs the training curve loss decreases slightly below the validation curve indicating that further training could possibly decrease the accuracy of the machine learning model on the testing data set.

A.3.3 Regression

The classification problem very effectively classifies the noisy data. In the classification problem only four bins are used, which makes the data very coarse. This is ideal for human interface and works well to prioritise component inspection and replacement plans. Most of the optimisation algorithms that is available uses some form of gradient descent method. These methods work well with smooth continues objective functions. Approaching this problem as regression problems holds various advantages.

As with the classification problem the datasets are generated as per subsection 1.5.9, however instead of converting the life fraction used into a classification label. The data set label $\vec{y}$ is left in the real value domain.

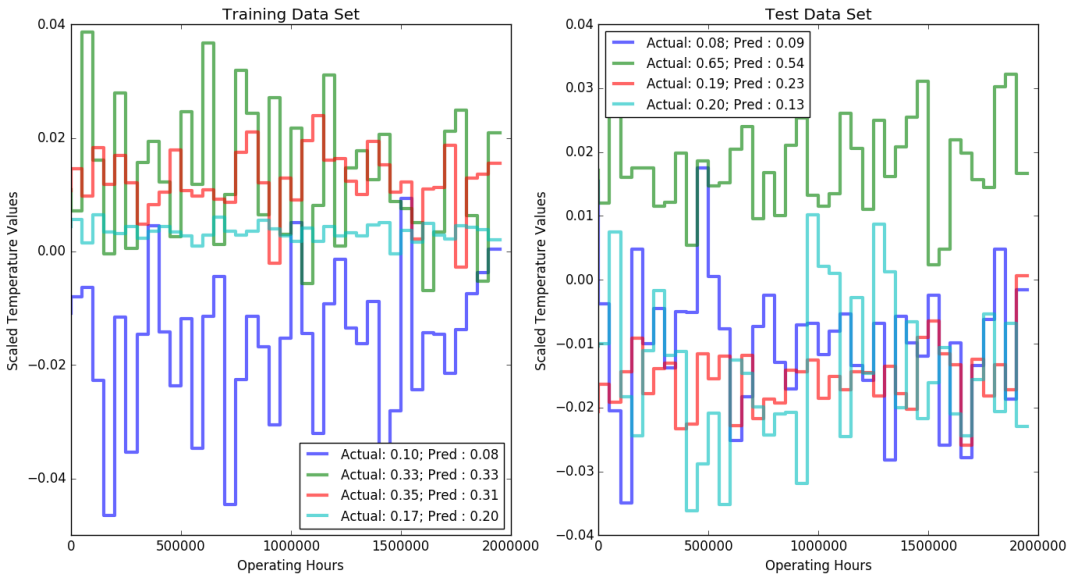


Figure A.8: Regression Signals

Figure A.8 illustrates the same scaled temperature signals as used in the classification problem. The signal labels y are real values and not *one hot encoding* vectors. Both the actual label ($\vec{y}$) and predicted label ($\vec{y'}$) are shown in the legends.

A.3.4 Regression Training Curves

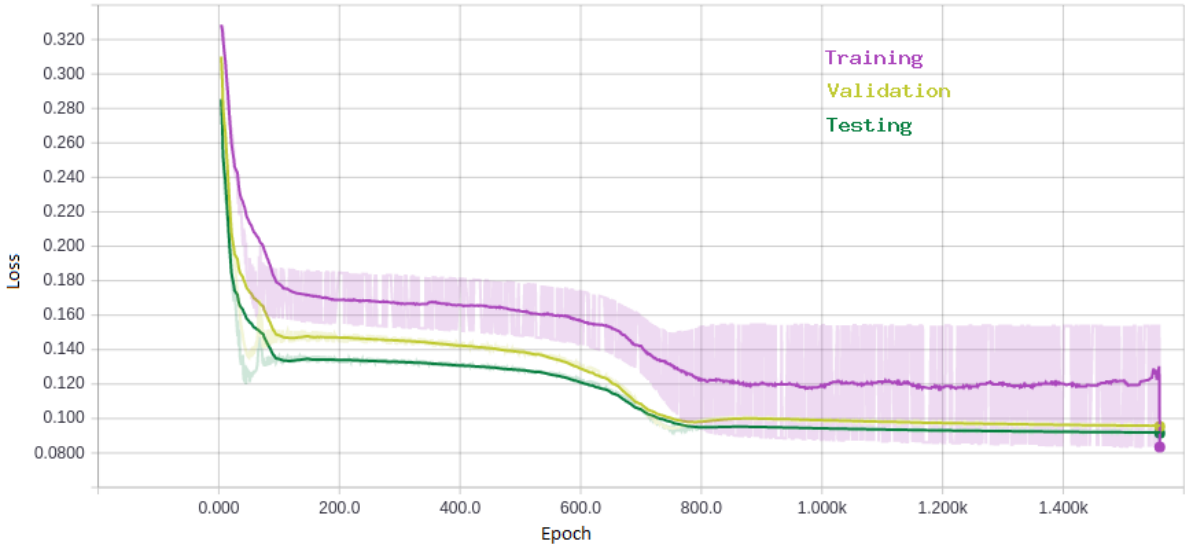


Figure A.9: Loss LSTM Regression ADAGRAD (iter = 100k)

Figure A.9 shows the Loss training curves of a LSTM cell recurrent neural network. The machine learning model converges to a solution within 900 epochs. The model is able to classify al 40 testing signals into the correct classification bin.

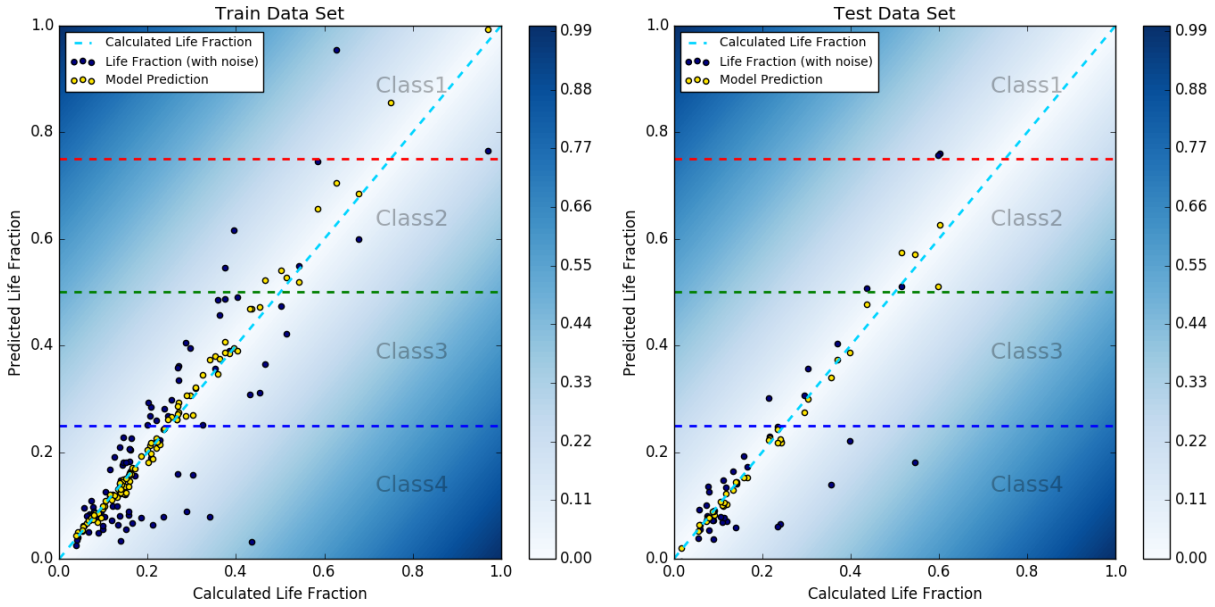


Figure A.10: Regression Accuracy

Figure A.10 illustrates the calculated life predictions with Gaussian noise ($\vec{y}_{noise}$) in blue and the model predictions ($\vec{y'}$) in yellow. One dot on the figure represents the life fraction used of one input time series. It is interesting to notice that the model prediction fits the noise free calculated life fraction ($\vec{y}$), indicate with light blue dotted line, values surprisingly accurately. Keep in mind that the model has only used the blue dots ($\vec{y}_{noise}$) to train. Also keep in mind that each blue dot represents a input temperature sequence. Thus one can say that the Machine Learning Model does not over-fit to the training data and very effectively learns to ignore noise in the data. The background contour plot is shows the RMS error of a prediction point. Areas close to the light blue dotted line ($\vec{y}$) are accurate predictions and the error/loss is low.

A.4 Results

A Python based program using the Tensorflow library was used to obtained the results reported in this chapter.

A.4.1 LSTM and GRU Comparison

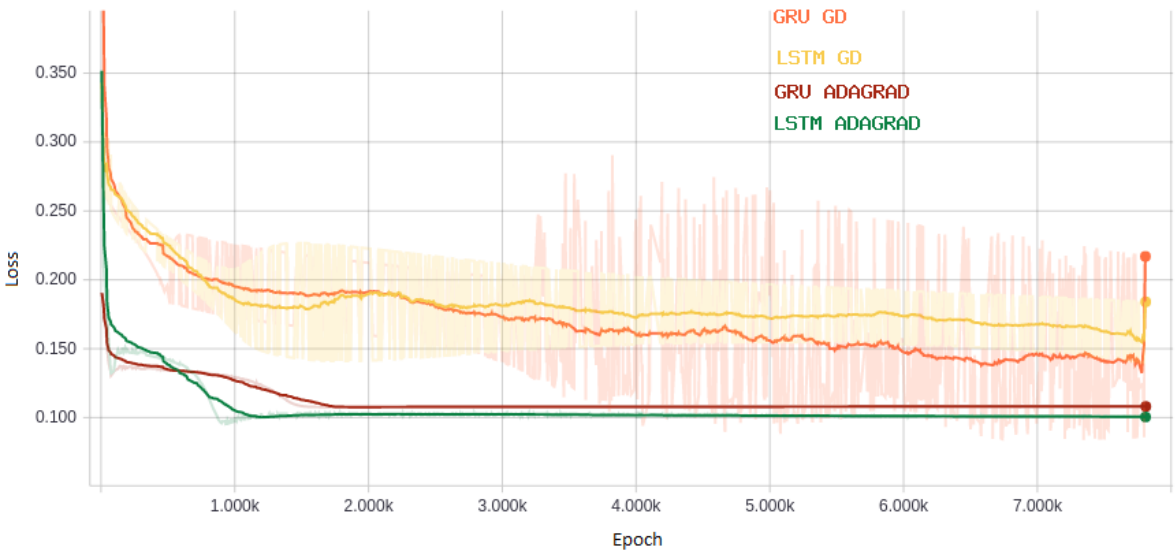


Figure A.11: Cell type and regression algorithm comparison

Figure A.11 compares the validation training curves of the LSTM cell and the GRU cell, as well as the influence of the gradient descent algorithm that is employed. It is clear that the ADAGRAD algorithm is a much more stable algorithm and takes much longer to converge than Gradient Decent algorithm. Generally the LSTM cell outperforms the GRU cell. The GRU cell in general takes less time to execute, because there is less function evaluations to perform with each iteration.

Table A.2 summarizes the results obtained with the GRU cell based model. Is shows that there is an increase in classification accuracy if the classification model is changed to a regression model. In not one of the five cases the model has over-fit to the training data as the training set predictions are no more accurate than the test or validation set predictions. Normal gradient descent does not seem to work well with the GRU cell as in both cases accuracy of the model was very low. The ADAGRAD algorithm on the other hand seems to work well with the GRU cell. The ADAGRAD algorithm also manges to score a high accuracy of a 95% correct classification rate in the case of the regression problem.

Table A.2: GRU

	CLASSIFICATION		REGRESSION		
Cost	SCEWL$^{(1)}$	SCEWL$^{(1)}$	RMS$^{(2)}$	RMS$^{(2)}$	RMS$^{(2)}$
Optimizer	GD$^{(3)}$	ADAGRAD$^{(4)}$	ADAGRAD$^{(4)}$	ADAGRAD$^{(4)}$	GD$^{(3)}$
Training iterations	500000	500000	100000	500000	500000
$\text{Loss}_{\text{training}}$	0.349	0.263	0.096	0.137	0.083
Loss_{val}	0.418	0.220	0.151	0.087	0.075
$\text{Loss}_{\text{test}}$	0.415	0.251	0.109	0.108	0.217
$\text{Accuracy}_{\text{training}}$	65.0%	90.0%	85.0%	93.0%	69.0%
$\text{Accuracy}_{\text{val}}$	62.5%	87.5%	90.0%	92.0%	67.5%
$\text{Accuracy}_{\text{test}}$	75.0%	90.0%	75.0%	95.0%	75.0%
SOLVING TIME	7m34s	7m29s	1m52s	8m57s	9m23s

SCEWL$^{(1)}$	Sigmoid cross entropy with logits
RMS$^{(2)}$	Root mean square
GD$^{(3)}$	Gradient descent
ADAGRAD$^{(4)}$	Adaptive gradient decent

Table A.3 shows the results for the LSTM cell based model. The ADAGRAD algorithm shows increase classification accuracy over the gradient descent algorithm. From the tests conducted, no conclusion can be made to weather the classification of regression case will yields better results. It is interesting to note that the most accurate testing accuracy obtained was 100% on a test set of 40 datasets. This accuracy was obtained by using ADAGRAD with a LSTM based model, with 100k training iterations. Intrestingly the accuracy goes down with more training iterations.

Table A.3: LSTM

	CLASSIFICATION		REGRESSION		
Cost	SCEWL$^{(1)}$	SCEWL$^{(1)}$	RMS$^{(2)}$	RMS$^{(2)}$	RMS$^{(2)}$
Optimizer	GD$^{(3)}$	ADAGRAD$^{(4)}$	ADAGRAD$^{(4)}$	ADAGRAD$^{(4)}$	GD$^{(3)}$
Training iterations	500000	500000	100000	500000	500000
$\text{Loss}_{\text{training}}$	0.250	0.861	0.083	0.136	0.180
Loss_{val}	0.225	0.775	0.091	0.112	0.173
$\text{Loss}_{\text{test}}$	0.301	0.700	0.095	0.100	0.176
$\text{Accuracy}_{\text{training}}$	89.0%	90.0%	93.0%	88.0%	58.0%
$\text{Accuracy}_{\text{val}}$	85.0%	87.5%	90.0%	87.5%	62.5%
$\text{Accuracy}_{\text{test}}$	90.0%	92.5%	100.0%	95.0%	62.5%
Solving Time	9m20s	9m22s	1m30s	10m05s	9m01s

SCEWL$^{(1)}$	Sigmoid cross entropy with logits
RMS$^{(2)}$	Root mean square
GD$^{(3)}$	Gradient decent
ADAGRAD$^{(4)}$	Adaptive gradient decent