DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Machinery Health Indicator Construction using Multi-objective Genetic Algorithm Optimization of a Feed-forward Neural Network based on Distance

Master Thesis in Machine Learning

JACOB NYMAN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Machinery Health Indicator Construction using Multi-objective Genetic Algorithm Optimization of a Feed-forward Neural Network based on Distance

Master Thesis in Machine Learning

JACOB NYMAN

Master's Programme, Machine Learning, 120 credits
Date: May 13, 2021

Supervisor: Arvind Kumar
Examiner: Pawel Herman
School of Electrical Engineering and Computer Science
Host company: CNet Svenska AB
Swedish title: Maskin-Hälsoindikatorkonstruktion genom Multi-objektiv Genetisk Algoritm-Optimering av ett Feed-forward Neuralt Nätverk baserat på Avstånd
Swedish subtitle: Examensarbete i Maskininlärning

Abstract

Assessment of machine health and prediction of future failures are critical for maintenance decisions. Many of the existing methods use unsupervised techniques to construct health indicators by measuring the disparity between the current state and either the healthy or the faulty states of the system. This approach can work well, but if the resulting health indicators are insufficient there is no easy way to steer the algorithm towards better ones.

In this thesis a new method for health indicator construction is investigated that aims to solve this issue. It is based on measuring distance after transforming the sensor data into a new space using a feed-forward neural network. The feed-forward neural network is trained using a multi-objective optimization algorithm, NSGA-II, to optimize criteria that are desired in a health indicator. Thereafter the constructed health indicator is passed into a gated recurrent unit for remaining useful life prediction. The approach is compared to benchmarks on the NASA Turbofan Engine Degradation Simulation dataset and in regard to the size of the neural networks, the model performs relatively well, but does not outperform the results reported by a few of the more recent methods.

The method is also investigated on a simulated dataset based on elevator weights with two independent failures. The method is able to construct a single health indicator with a desirable shape for both failures, although the latter estimates of time until failure are overestimated for the more rare failure type. On both datasets the health indicator construction method is compared with a baseline without transformation function and does in both cases outperform it in terms of the resulting remaining useful life prediction error using the gated recurrent unit. Overall, the method is shown to be flexible in generating health indicators with different characteristics and because of its properties it is adaptive to different remaining useful life prediction methods.

Keywords

Prognostics, Health Indicator Construction, Remaining Useful Life Prediction, Multi-objective Optimization, Distance

Sammanfattning

Estimering av maskinhälsa och prognos av framtida fel är kritiska steg för underhållsbeslut. Många av de befintliga metoderna använder icke-väglett (unsupervised) lärande för att konstruera hälsoindikatorer som beskriver maskinens tillstånd över tid. Detta sker genom att mäta olikheter mellan det nuvarande tillståndet och antingen de friska eller fallerande tillstånden i systemet. Det här tillvägagångssättet kan fungera väl, men om de resulterande hälsoindikatorerna är otillräckliga så finns det inget enkelt sätt att styra algoritmen mot bättre.

I det här examensarbetet undersöks en ny metod för konstruktion av hälsoindikatorer som försöker lösa det här problemet. Den är baserad på avståndsmätning efter att ha transformerat indatat till ett nytt vektorrum genom ett feed-forward neuralt nätverk. Nätverket är tränat genom en multi-objektiv optimeringsalgoritm, NSGA-II, för att optimera kriterier som är önskvärda hos en hälsoindikator. Därefter används den konstruerade hälsoindikatorn som indata till en gated recurrent unit (ett neuralt nätverk som hanterar sekventiell data) för att förutspå återstående livslängd hos systemet i fråga. Metoden jämförs med andra metoder på ett dataset från NASA som simulerar degradering hos turbofan-motorer. Med avseende på storleken på de använda neurala nätverken så är resultatet relativt bra, men överträffar inte resultaten rapporterade från några av de senaste metoderna.

Metoden testas även på ett simulerat dataset baserat på elevatorer som fraktar säd med två oberoende fel. Metoden lyckas skapa en hälsoindikator som har en önskvärd form för båda felen. Dock så överskattar den senare modellen, som använde hälsoindikatorn, återstående livslängd vid estimering av det mer ovanliga felet. På båda dataseten jämförs metoden för hälsoindikatorkonstruktion med en basmetod utan transformering, d.v.s. avståndet mäts direkt från grund-datat. I båda fallen överträffar den föreslagna metoden basmetoden i termer av förutsägelsefel av återstående livslängd genom gated recurrent unit-nätverket. På det stora hela så visar sig metoden vara flexibel i skapandet av hälsoindikatorer med olika attribut och p.g.a. metodens egenskaper är den adaptiv för olika typer av metoder som förutspår återstående livslängd.

Nyckelord

Prognostik, Hälsoindikatorkonstruktion, Återstående Livslängd, Multiobjektiv Optimering, Avstånd

Acknowledgments

I would like to thank Arvind Kumar for the helpful discussions and valuable feedback throughout the thesis. I also would like to thank Peter Rosengren and the team at CNet for the cooperation and ideas to adjust the project to work with encountered problems.

Contents

List of acronyms and abbreviations

Notation	Description
$\boldsymbol{x_{i:j}}$	The subvector of vector x starting from index i and ending at index j, $j > i$
AM	Artificial Intelligence Method
BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
DBN	Deep Belief Network
DDM	Data-driven Method
EWS	Elevator Weight Simulation
FBD-HI	Failure-based Distance for Health Indicator Construction
FNN	Feed-forward Neural Network
GA	Genetic Algorithm
GRU	Gated Recurrent Unit
HBD-HI	Healthy-based Distance for Health Indicator Construction
HI	Health Indicator
HM	Hybrid Method
HP	Hyperparameter
HPS	Health Prognostics
LSTM	Long Short-Term Memory
MOGA	Multi-objective Genetic Algorithm
MOGA-NN	Multi-objective Genetic Algorithm Neural Network
Mon$_n$	Monotonicity with step size n
NN	Neural Network
NSGA-II	Non-dominated Sorting Genetic Algorithm II
PCA	Principal Component Analysis
PDF	Probability Density Function
PHM	Prognostics and Health Management
PMM	Physical Model-based Method
RBM	Restricted Boltzmann Machine
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RUL	Remaining Useful Life
Second-mon$_n$	Second-degree Monotonicity with step size n
SMM	Statistical Model-based Method
TEDS	Turbofan Engine Degradation Simulation
Trend	Trendability
Trend$_s$	Spearman-trendability
VHI	Virtual Health Indicator

Chapter 1

Introduction

Prognostics and Health Management (PHM) is an increasingly important field as manufacturing intensity grows. A machine or system failure can cause impactful negative consequences to the business operation and in turn to the profit. A common strategy is to continuously monitor the system of interest, model its health and forecast future failures. These forecasts can then be used to perform well-timed and condition-based maintenance based on real time information. Consequently, increasing efficiency and averting costly failures.

For forecasting failures it is crucial to have a health indicator (HI) that estimates the health of the system as the degradation evolves. Usually we have one or more measurements obtained through sensors that, to a certain degree, describe the degradation state of the system. But these measurements can be noisy, nonlinear and therefore, difficult to model and forecast. For these reasons, methods have been developed to construct *virtual* HIs (VHIs) by fusing all sensor measurements into a single measurement in a supervised or unsupervised fashion. Several statistical methods and machine learning techniques have been utilized for this purpose. For instance, genetic programming [1], linear regression [2], deep belief network [3] (DBN) and restricted Boltzmann machine [4] (RBM).

Thereafter, the constructed HI is used to forecast the remaining useful life (RUL) of the system using time series techniques. In recent years deep learning have been successfully applied for this purpose using models such as recurrent neural networks (RNNs) [5]–[7] and convolutional neural networks (CNNs) [8], [9]. But there are also benefits in using more classical statistical methods, such as state space models [2], [3], due to

the way uncertainty is quantified and the effectiveness on small datasets.

1.1 Thesis aim

In many of the current methods HIs are constructed through some unsupervised or supervised approach without explicitly telling the method what is sought. For example, Hou *et al.* [4] used the reconstruction error from an RBM trained on healthy sensor data as an HI. This can work well but there is little to no control on the form of the HI. For instance, in these methods, it is not possible to include knowledge about the desired shape and properties of the HI that fits with the forecasting model or generally characterize a good HI.

The aim of this thesis is to investigate a more deliberate framework for HI construction where there is more control over the final shape of the HIs. At the core of this framework lies the multi-objective genetic algorithm (MOGA). It is used to optimize a function to transform sensor data/features/states into vectors in a new space. Thereafter, in this space, distances between the states and the *faulty states*, defined as all states right before failure in the training data, are measured. Then the HI at each time point is defined as the minimum distance from the transformed state of the time point to any of the (transformed) faulty states.

The fitness/objective functions of the optimization process are metrics related to the quality of the resulting HIs. Because the MOGA is used, the process does not only enable the user to incorporate domain knowledge and desired shape of the HI using fitness functions, but also provides a large population of solutions for the user to choose between. The function to be optimized is a neural network (NN). NNs are very powerful function approximators as proven by the universal approximation theorem [10]. They can also handle noisy data such as sensor data very well. For this reason this is the model whose parameters are to be optimized.

1.2 Research Questions

The overall aim is to validate the method, compare it to existing methods and test it in a difficult setting. The following research questions are investigated:

- How does the proposed method compare with state of the art approaches such as [4], [6], [11], [12] in terms of RUL prediction?

The first question implies a comparative evaluation of the proposed method.

- What is the effect of generating HIs by transforming the states before measuring the distance to the faulty states in terms of the resulting HIs and RUL prediction error?

The second question is about justifying the notion of using a transformation function before measuring distance to the faulty states, which is similar to justifying the use of a more complex model. Simply measuring the distance in the original space is the baseline and the transformation function should have a compelling argument to be preferred (Occam's razor).

- What are the advantages and disadvantages in using the proposed method to handle competing risks or failures with different characteristics?

Finally, the third question is about the utility of the method under different conditions. Competing risks are defined as situations where several failures could happen but only one of them can occur. This means that all failures have to be simultaneously modeled and taken into account when forming the HI and/or performing the prediction of RUL. The failure types can be seen as failures with different characteristics.

1.3 Scope and limitations

The focus of the thesis lies in the process of constructing HIs from sensor data/features and then predicting RUL using these HIs. This is done specifically in regard to machines. For this reason one assumption that is made is that the system is degrading monotonically. This implies that the system does not heal without intervention. For machines, usually the speed of the degradation can change but not the direction. The assumption is heavily relied upon in the selection/composition of the fitness functions.

In both of the explored datasets the trajectories were also run to failure, and thus, no censored data was assumed (see section 2.3 for a short discussion about censored data). Additionally, in both datasets the

training/vaidation part of the dataset consisted of 100 trajectories/time series of degradations. Consequently, the results are bounded to these types of scenarios.

Chapter 2

Background

2.1 Prognostics and Health Management

PHM is a framework that encompasses several objectives related to monitoring systems, estimating their health, predicting failures and generating maintenance strategies. There are several reasons for the growing interest in this field. The obvious one is that downtime and maintenance could be costly when a system is unexpectedly failing, especially in industries where reliability is key. But there could also be catastrophic consequences impacting human lives directly as pointed out in [13]. Thus, there are obvious motivations for the interest and development of the field.

2.2 Health Prognostics

Health prognostics (HPS) is a subcomponent of PHM and the focus of this thesis. HPS aims to describe the health of a system over time with the objective of forecasting the RUL of the system. This process can be decomposed into four steps [14]:

- data acquisition
- HI construction
- health stage division
- RUL prediction

The first stage, data acquisition, is the collection of sensor data that can measure some property related to the degradation of the system in focus over time. Without time-dependent data, dynamic health state estimation is not possible. If something happens that causes the degradation process to accelerate (or decelerate) we would not know about it.

In the second stage, HI construction, the acquired data is processed to generate HIs that can adequately describe the degradation of the system. An HI is a one-dimensional time series that describes the health of the system over time. This stage can be more or less involved depending on the complexity of the system. Thereafter the constructed HI is decomposed into health stages, where the final health stage is used for the final step, RUL prediction. This is, as the name suggests, about predicting how much time left there is before failure. In this thesis the focus will lie in the second and fourth step of the process. That is constructing HIs and then using these HIs to predict the RUL. The two steps involved in this thesis are shown in Figure 2.1.

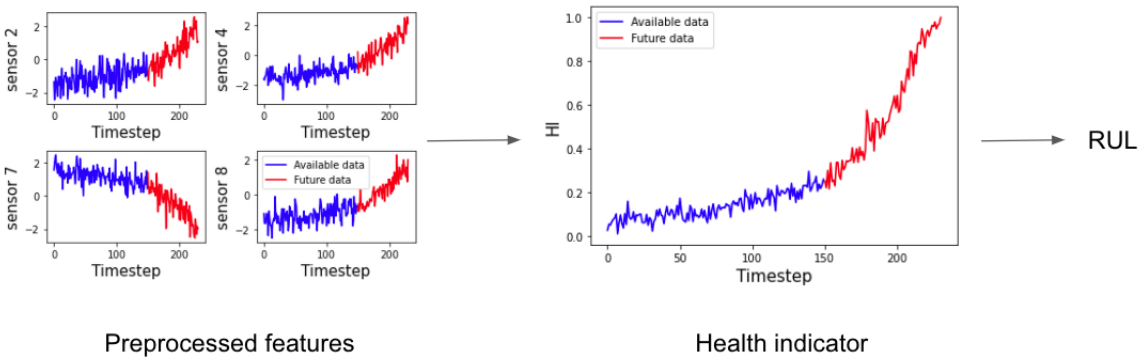


Figure 2.1 – The two steps of HPS in focus for this thesis. In blue the data available at the time of prediction and in red the unavailable future data points.

2.3 Data acquisition

To be able to create HIs and predict RUL, data is necessary. Sensors have to be installed to measure quantities related to the degradation. Examples of popular sensors for mechanical systems are accelerometers that measure vibration, microphones that measure sound pressure and infrared thermography that measures temperature [15]. Also the time of

failure and potentially what type of failure occurred (if there are multiple) have to be recorded. In scenarios where the degradation is monotonic and is run from start to failure this should be enough. But sometimes there can be intervening actions, such as maintenance operations, that cause the health of the system to be restored. These types of actions also need to be recorded otherwise assumptions of monotonic degradation are violated and modelling becomes more difficult. Even if a repairing action is performed and no failure occurs, the sensor data acquired up until this point can still be useful. This is called *censored* data, specifically right censored data [16]. Because it is known that the system has not failed yet it is also known that the time until failure will be greater than the time that has passed so far. There are many methods in the field of survival analysis [16] devoted to making use of such data.

2.4 Health indicator construction

As mentioned earlier the HI construction can be more or less involved. Sometimes there is multidimensional sensor data with various complex characteristics. This creates a non-trivial problem of constructing a VHI that can describe the degradation of the system. Various statistical or machine learning techniques have been attempted to do this. In other cases a physical HI already exists and no additional processing is required. In this thesis the usage of an NN in combination with an MOGA is investigated for the former scenario.

2.4.1 Previous work

Several methodologies have been developed for constructing VHIs. The initial step is extracting predictive features from the data. Here domain knowledge and understanding of the system at hand can be beneficial. Time domain, frequency domain and time frequency domain features are common. Thereafter the techniques vary.

Sun *et al.* [2] used a linear regression model to transform the sensor measurements into an HI. This was done by creating a dataset where the states near the end of the time series close to failure were assigned a value of zero and the states at the beginning of the time series were assigned a value of one. Then the regression model was fit. The fit model could then transform sensor measurements to an HI where the health deteriorates approximately from one to zero. This is a useful way to turn the HI

construction problem into a supervised problem. The problem with this method is that it assumes a linear transformation of the sensor data or extracted features is enough, which is not always the case. Furthermore, since it only uses states at the beginning and end a lot of information is neglected. The proposed method does not have any of these weaknesses.

Liao [1] used genetic programming to learn a function that transformed the sensor data into HIs by optimizing the monotonicity of the HIs (defined in equation 2.1). Genetic programming is a specialized genetic algorithm (GA) that uses trees to construct a series of mathematical operators applied to the initial features. Thus, the model could handle nonlinear data and also used all datapoints of the trajectory. This is similar to the optimization performed in this thesis but in the proposed method instead of single-objective optimization, multi-objective optimization is used. This enables a more complete evaluation and the ability to find several solutions with different characteristics. One problem with this approach by Liao is that there is nothing restricting the HIs from failing at widely different values. It is desirable to fail at similar HI-values for RUL prediction (see section 2.6). The proposed method handles this issue desirably using distance.

Peng *et al.* [3] used a DBN to project the features into a vector space containing the most salient information of the data. The HIs were then constructed by extracting the vectors from the faulty states (states close to failure in the training data) and taking the minimum difference between the current vector and any of the vectors in the faulty state set. Hou *et al.* [4] used an RBM but here instead of using the minimum distance to the faulty states the reconstruction error after fitting the RBM to the healthy states was used as the HI. Similarly Wang *et al.* used the same concept but with autoencoders. The reconstruction error for the sensor data was then expected to increase as the state moved further away from the healthy working condition, and thus, an HI was obtained that was increasing over time. One advantage with these methods is that NNs are used, which are very powerful in extracting information from raw data and complex relationships. In both approaches though, (reconstruction loss and distance to faulty states) there is no possibility of encoding information about what is sought of the constructed HI as opposed to the proposed method.

Baraldi *et al.* [18] used multi-objective optimization with binary differential evolution to optimize criteria of interest in HIs: trendability (see equation 2.2), monotonicity (see equation 2.1) and prognosability

(see [18]). Each solution in the optimization procedure is defining a subset of the extracted features to use. These features are then fused together to form HIs using auto-associative kernel regression and reconstruction loss based on states in the training data close to failure. Finally, the HIs are evaluated based on the metrics and the solutions (subsets of features) updated using the optimization procedure. This method is similar to the method used in this thesis. Both methods use multi-objective optimization to optimize characteristics of the HIs and use discrepancies to the faulty states of the system (reconstruction loss vs. distance). The big difference lie in how the problem is defined. In [18] the optimization process is optimizing the subset of features to be input into the HI construction method, but the method itself is unaffected. Thus, for the method to be able to construct a wide variety of HIs it requires a wide variety of features. In contrast, in the proposed method a nonlinear function approximator, an NN, is optimized directly to produce useful HIs using distance. Because a function is optimized as opposed to only the selection of features, the proposed method is much more flexible.

2.4.2 Comparing properties of healthy and faulty states

The approach used by Peng *et al.* calculates distance to the faulty states after transforming the raw data, the approach used by Hou *et al.* and Wang *et al.* uses the reconstruction loss after fitting to the healthy states and the approach used by Baraldi *et al.* uses the reconstruction loss after fitting to the faulty states. What these approaches have in common is that they measure some kind of change over time based on comparing properties from either healthy or faulty states only.

This is a powerful concept since the difference between healthy/faulty states and the current state should presumably increase/decrease as the system is degrading. If we use the distance to the faulty states we also obtain a natural threshold of where the failure occurs on the HI, that is around zero (zero for the training data, but likely not exactly zero for future failures). If these distances are calculated from a transformed space using an unsupervised approach such as the DBN there is not much control of how the distance evolves over time.

In this thesis distance will be used, but instead of learning the transformation function using an unsupervised method a more direct approach is used. This is done by optimizing the transformation function so as to transform the sensor state into a vector space where the change of

distance over time between the faulty states and the current state exhibits properties that are desired in an HI. Common ones are correlation with time and monotonicity (see the next section). To do this multi-objective optimization is performed using a GA. Distance-based methods are discussed more in section 3.1.

2.5 Health indicator evaluation

To be able to construct effective HIs we need to understand their purpose and goal. The goal of HIs is to track the underlying health of the system and facilitate prediction of RUL. Thus, the ideal evaluation function for a given set of HIs is to fit some model to them and use the HI that leads to the optimal prediction of RUL. The problem is that this means that if we want to evaluate several HIs we have to fit a model to each one which can be very time consuming. Instead we would like to have some metric(s) that can quickly assess the ability of the HI to be useful for its purpose regardless of what model will be used. Many different metrics have been created for this purpose. A number of them are presented by Lei *et al.* [14]. The most common ones are *monotonicity* and correlation with time, also known as *trendability*, defined below:

$$\begin{gathered}\text{Mon}(HI) = \left|\frac{\text{No. of diff} > 0}{T-1} - \frac{\text{No. of diff} < 0}{T-1}\right| \\ \text{where diff} = HI_{1:T} - HI_{0:T-1}\end{gathered} \tag{2.1}$$

$$\text{Trend}(HI) = \left|\frac{\sum_{t=1}^{T}(HI_t - \overline{HI})(t-\bar{t})}{\sqrt{\sum_{t=1}^{T}(HI - \overline{HI})^2 \sum_{t=1}^{T}(t-\bar{t})^2}}\right| \tag{2.2}$$

where the notation $HI_{i:j}$ denotes the subvector starting from index i and ending at index j of vector/time series HI. T is the length of the HI and the overline represents the mean.

These complement each other in the sense that monotonicity evaluates the local changes and trendability the global trend. But alone these two are not perfect. Monotonicity is very sensitive to noise and trendability expects a linear degradation curve. To evaluate nonlinear degradation the *Spearman correlation coefficient* can be used between the HI and time. It is here denoted as *Spearman-trendability*. It is defined as:

$$\text{Trend}_s(HI) = \text{Trend}(\text{rank}(HI)) \tag{2.3}$$

where rank(HI) denotes the rank sequence of the HI. Thus, it is the correlation between the rank sequence of the HI and time. One major difference between correlation and Spearman correlation when the HI is linear is that Spearman correlation is more robust to outliers (in the monotonic direction) in the beginning and end of the HI. This is because it only cares about the rank and not the value.

Hybrid metrics [14] are defined as the combination of several metrics to evaluate the HI in different aspects simultaneously, often through a weighted mean. The downside of this is that a weight has to be assigned to each objective. But using multi-objective optimization, as is used here, this problem disappears and more diverse and multifaceted solutions can be found.

2.6 Health indicator modelling

The true objective of an HI is to use it to predict RUL and there are several different ways to do this. One of the more common ways is to model the time until the HI passes a known failure threshold. Then the RUL is defined as [14]:

$$\text{RUL}_k = \inf(l : HI_{l+k} \geq \gamma) \tag{2.4}$$

where RUL_k is the RUL at the kth timestep, HI_{l+k} is the HI at time $l + k$ with $l \geq 0$, inf is the limit infimum, and γ is the failure threshold. Often due to simplicity, the failure threshold is defined as a constant value even though ideally we would like to have a probability distribution. As previously mentioned, one advantage with generating HIs using the distance to the faulty states is that the final distance will be close to zero. Consequently, we obtain a natural failure threshold. Another option is to simply predict the RUL directly given the history of the HI.

The different methods can be categorized into four groups [14]:

- Physical model-based methods (PMMs)
- Statistical model-based methods (SMMs)
- Artificial Intelligence methods (AMs)
- Hybrid methods (HMs)

PMMs are methods based on mathematical models that describe the physics of the system to assess its current and future health [19]. This requires in-depth knowledge of the degradation phenomena which is rarely available for complex systems. To be able to model systems without extensive knowledge of the physical interactions, *data-driven methods* (DDMs) can be used.

As opposed to PMMs that are system specific, DDMs can be used for different systems. The downside is that more data is needed. The data-driven methods can be divided into SMMs and AMs. SMMs predict the RUL by modelling the HI using statistical models and the RUL is often predicted by estimating when the failure threshold is exceeded (as defined in equation 2.4). Because statistical techniques are used we also obtain a probability density function (PDF) of the predicted RUL which can be advantageous for maintenance decisions. *Kalman filter* [20] and *particle filter* [21] are techniques that often can be applied to adaptively fit the SMM to the current HI trajectory. Thus, the method can be resilient to individual differences between systems of the same type. Examples of models are Wiener process models, gamma process models, the double exponential model and proportional hazard models.

AMs are methods using machine learning techniques to model the degradation pattern and have become more popular in recent years. For example RNNs [5]–[7] and CNNs [8], [9]. With these methods the RUL can be predicted using a failure threshold for the HI but more commonly by directly predicting the RUL. In some cases the step of constructing HIs is skipped entirely and instead of first constructing an HI the RUL is predicted immediately from the raw sensor data, e.g. [5], [8]. Thus, the pipeline becomes a black box. While having an end-to-end pipeline can be desirable in some scenarios it also makes the whole process less transparent and understandable. An HI can be useful for diagnosis, maintenance decisions and also reassurance of how the predictions are made. Additionally, multiple models of different types can be applied to the same data.

Overall, the advantage with AMs is that they can perform very well on difficult and nonlinear problems given enough data. When predicting the mode this is often the more accurate approach which can be seen on benchmarks, for example the dataset presented in section 4.5.1 where all top solutions used an AM. The downside is that these methods require a lot of data, more than statistical methods, otherwise there is an increased risk of overfitting. Also there is no simple way to provide confidence

intervals. Since each type of method has its pros and cons, there have been research on how to combine them to incorporate the advantages of each individual method to generate an HM.

The method used in this thesis is an AM, but since the previous step constructs a number of diverse HIs the last step can easily be changed to an SMM. This could potentially vary from case to case but during experimentation using AMs for the RUL prediction was shown to achieve the best results when point estimates were of interest. The used AM for the HI modelling is a gated recurrent unit (GRU), an RNN, and is explained in section 3.6.1.

Chapter 3

Theory

3.1 Categorizing distance-based HIs

There are two main ways to construct HIs using distance, either the distance from the healthy states is measured or the distance to the faulty states is measured. If the faulty states are used the distance will decrease towards zero. If the healthy states are used the distance will increase towards some arbitrary distance, likely varying for each trajectory (see Figure 3.1).

Reconstruction loss will exhibit similar properties as using distance but it does not necessarily follow the same mathematical axioms and is thus left out of the discussion. Distance is defined as follows. Given a set X, a function d, $d : X \times X \longrightarrow \mathbb{R}$, is a distance function if the following axioms hold:

1. $d(x, y) \geq 0$
2. $d(x, y) = 0 \iff x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, y) \leq d(x, z) + d(z, y)$

A failure-based distance method for HI construction (FBD-HI), is here defined as a method using the minimum distance to any of the faulty states in the training data to generate HIs. A faulty state is defined as the

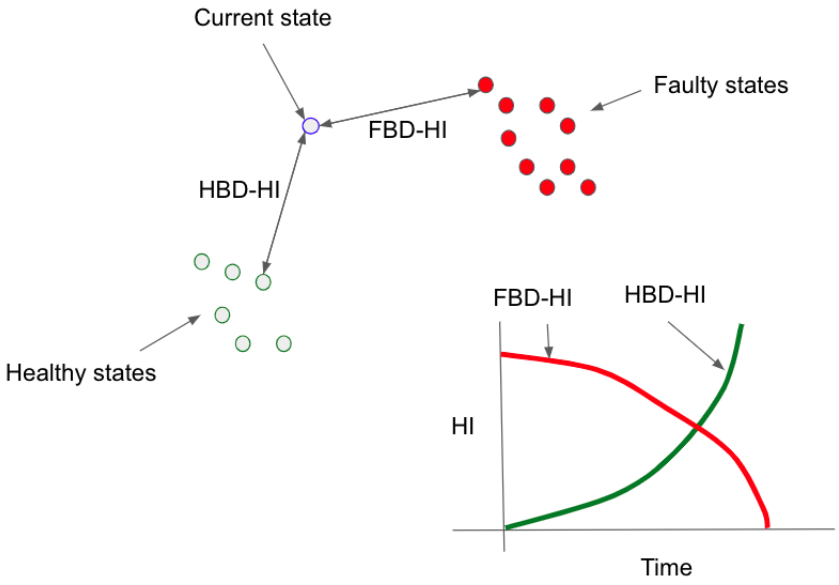


Figure 3.1 – The difference between healthy-based and faulty-based HI construction. In the top-left the state space is visualized (in reality it will often be in a higher dimensional space). In the bottom-right the corresponding HI for each type is shown. Note that this is not real data but a schematic representation.

last state before failure. The FBD-HI is mathematically defined below:

$$\begin{aligned} f(s_t) &= h_t \\ f(s_{train,i,T}) &= z_i \\ HI_t &= \min_i d(z_i, h_t) \end{aligned} \tag{3.1}$$

where s_t is the sensor state at time t to generate an HI for, $s_{train,i,T}$ is the sensor state at time T (last state before failure, the faulty state) of the ith trajectory in the training data and f is some transformation function.

A healthy-based distance method for HI construction (HBD-HI), is here defined as a method using the minimum distance to any of the healthy states in the training data to generate HIs:

$$\begin{aligned} f(s_t) &= h_t \\ f(s_{train,i,j}) &= z_{i,j}, j \in HS_i \\ HI_t &= \min_{i,j} d(z_{i,j}, h_t) \end{aligned} \tag{3.2}$$

where HS_i is the healthy indices of the ith trajectory. Here HS is

more loosely defined, but often the initial states of each trajectory will work.

The transformation function is a function that transforms the sensor states to some vector space where the distances exhibit properties of interest. This can be done using DBNs and autoencoders but also methods such as principal component analysis (PCA) and kernel PCA. What these methods have in common is the fact that they do not directly optimize the transformed space for HI learning, they are unsupervised. The intention behind the proposed method is to directly optimize this function.

The advantage of the FBD-HI as opposed to the HBD-HI is that an expected point of failure around zero is automatically obtained, which is very useful for RUL prediction models using failure thresholds. Because of this property the FBD-HI is the focus of this thesis.

3.2 Feed-forward neural network

A feed-forward NN (FNN) is a machine learning algorithm where the inputs/features are nonlinearly transformed using matrix transformations together with activation functions in a forward fashion, that is without loops:

$$y = f(Wx + b)$$

where y is the output, W is the matrix of weights that describes the linear transformation, b is the added bias, x is the input and f is the activation function. This operation can be, and commonly is, performed several times where the output of the previous transformation becomes the input to the next. Then each intermediary output-vector creates a *layer* and each individual output in the layer denotes a *node*. The activation functions are nonlinear in the intermediary layers (otherwise the network can be collapsed into a single linear transformation) while the last layer (the actual output) can be shaped based on the labeled data. For example for categorical data the softmax function generates a probability distribution. If we have input and output pairs, are using differentiable (with respect to the input) activation functions and a differentiable loss function that can evaluate the error between prediction and the actual output labels. Then we can train the weights for every layer to minimize the error of the loss function (and in turn learn the

input-output mapping) using backpropagation [22].

3.3 Genetic algorithm

A GA is an evolutionary algorithm (metaheuristic population-based optimization algorithm) inspired from the Darwinian theory of evolution [23]. It consists of a population of solutions called *chromosomes*, which in turn consist of parameters called *genes*. These solutions are evolved or discarded based on scores obtained through a *fitness function* (or objective function). The algorithm advances the population of chromosomes using a few key steps iteratively:

1. Initialization
2. Evaluation
3. Selection
4. Crossover
5. Mutation
6. Stop if predetermined criteria is fulfilled, otherwise go to stop 2

Initialization

In the first phase the population is initialized randomly. The most important thing here is to create a diverse population to explore the search space adequately.

Evaluation

In the evaluation phase each chromosome is assigned a score based on the fitness function, that is what we are trying to optimize.

Selection

The selection phase, inspired by natural selection, is where fit individuals produce offspring that will be part of the next generation of the population. To determine which individuals that will mate and produce offspring, different selection operators have been developed. Two common ones are *tournament selection* and *roulette wheel selection*.

In roulette wheel selection the fitness is normalized into a probability distribution where each individual's probability of selection is proportional to its fitness [24]. Then the cumulative sum of these probabilities are taken to create a series of consecutive bins proportional to the size of selection for each individual. Finally, N numbers, where N is the size of the population, are simulated uniformly between zero and one. For each number the corresponding bin it falls into (hence the similarity to a roulette wheel) is selected for the future generation.

In tournament selection k individuals, where k is the tournament size, are chosen at random from the population. Then the best individual among the k is selected as part of the new population [24]. This is performed N times where the same individual can be selected multiple times. Here k is a parameter that decides to what degree the better individuals should be selected. For example if k is equal to N only the best individual will be selected.

Crossover

After having selected individuals for the future generation, some of the individuals will crossover/mate to share successful genes. This is the crossover phase. This process represents local search or *exploitation* where we refine the chromosomes by sharing information between them. There are many different techniques to perform crossover, two common ones are single and double-point crossover. In these methods the two parent chromosomes are seen as two long vectors and are split at one (single point crossover) or two (double point crossover) points to generate two or three parts, respectively. Then the children are generated by combining parts from the parents [23], see Figure 3.2.

But at times these general crossover methods do not work because the chromosome/vector could have internal relationships that will be destroyed by simply swapping points. In this thesis GA will be applied to multi-layer NNs as chromosomes and this problem exists here. See section 3.5.2 for how this operation is performed.

Mutation

If only crossover was used the search would quickly get stuck in a local optima since no new information is ever incorporated. To mitigate this there exists another phase called mutation which represents *exploration* and leads to a global search. What this method does is

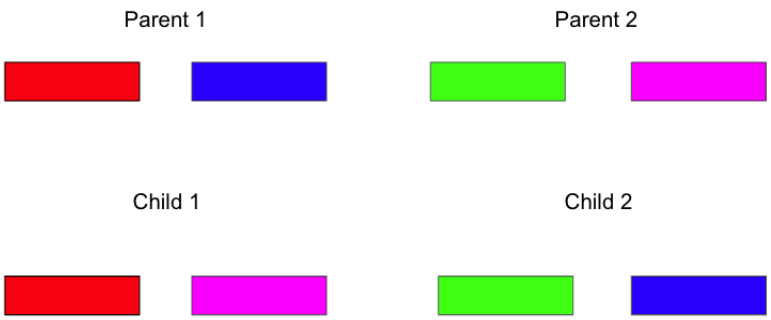


Figure 3.2 – Single-point crossover illustrated. The two parent-solutions are each split into two components, then the components are combined in a new form to produce offspring.

that it slightly alters or *mutates* a gene randomly to look outside of the current population and attain new information. This way the search can get out of local optimas. If the chromosome is represented by bits the switch of a bit could be a mutation operation or if we have a continuous representation adding Gaussian noise could be the mutation operation.

Parameters

There are two main parameters that will affect the search process: *mutation rate* and *crossover rate*. Mutation rate is the probability of mutating a gene. Crossover rate is the probability of a chromosome in the next generation to be produced by crossover as opposed to being the same as the previous generation excluding mutation. The crossover rate will often be high to exploit found information adequately and the mutation rate will be low for the method not to become a complete random search. But naturally it will vary from problem to problem. The population size also matters, where a small population would lead to less diversity but quicker iterations and vice versa for a large population.

3.4 Multi-objective optimization

In single-objective optimization one objective function is being optimized and we can only optimize it in one way, increase or decrease it (depending on if it is a minimization or maximization problem). But when we add another objective function there is usually a trade-off involved, that is

increasing one objective function will decrease the other. An example in finance is the trade-off between risk and return: if we decrease risk the return will drop and if we increase return the risk will increase. One method to handle this is to simply add fixed weights to the objectives, for example:

$$f(x) = w_1 z_1(x) + w_2 z_2(x) + ... + w_K z_K(x)$$

where z_i is the ith objective function, w_i the associated weight and K the number of objective functions. With this we have turned multiple objectives into a single objective. The problem with this is that determining the weights is difficult, the objective function could have completely different ranges and even if they do not, usually there is no known perfect trade-off.

To solve this issue there is an approach to multi-objective optimization where instead of attempting to find a single solution to the problem an entire set of solutions are searched. The set that is sought is the so called *Pareto optimal set* [25]. To explain what this set represents, *domination* in regard to multi-objective optimization is explained.

If one solution dominates another, in regard to the optimization problem at hand, this means that we can claim the solution is objectively better than the other. This fact can only be stated (given that we have no known weight for each objective) if the solution is at least as good in every objective where in at least one of the objectives it is also strictly better. Here "at least as good" is less than or equal for a minimization problem and greater than or equal for a maximization problem and "better" less than for a minimization problem and greater than for a maximization problem. Otherwise if one objective is worse than the other solution there is no way to claim it is objectively better. This can mathematically be defined by the following if all the objectives are to be minimized [25]:

$$\begin{gathered} x \text{ dominates } y,\ x \prec y, \text{ if and only if} \\ z_i(x) \leq z_i(y), \text{ for } i = 1, ..., K \text{ and} \\ z_j(x) < z_j(y) \text{ for at least one objective function } j \end{gathered}$$

Note that in some literature $x \succ y$ is instead used to convey that x dominates y, like in [25]. Now a solution is called *Pareto optimal* if there is no solution dominating it in the solution space. This means that if one objective is improved one or several other objective functions will decline. Finally, the set of all Pareto optimal solutions is known

as the Pareto optimal set/front [25]. Thus, the objective of many multi-objective optimization techniques is to approximate the Pareto front with a finite number of solutions. The challenges lie in approximating the optimal solutions as accurately as possible as well as having the solutions spread out across the Parent front.

3.5 Multi-objective genetic algorithm

GAs are very useful for multi-objective optimization (so called MOGAs) because of several reasons. They are population based, thus multiple solutions of the Pareto front can be found in a single run. The crossover can combine solutions with high scores in different objective functions to discover new areas of the Pareto front. Additionally, no modification of the objective functions is needed such as scaling [25].

3.5.1 NSGA-II

Non-dominated Sorting Genetic Algorithm II (NSGA-II) [26] is a popular MOGA. It is characterized by *elitism* and *non-dominated sorting*. Elitism specifies that the best solution(s) are kept in the population and are not part of the stochastic selection process. Non-dominated sorting describes a fitness assignment based on how dominant the solutions are by assigning them to *fronts*. The first front is the non-dominated solutions, and therefore, the current best approximation of the Pareto front. The second front is the non-dominated solutions given that we have removed all the solutions from the first front. The third is the same but with the second front also removed and so on until all solutions have been assigned a front. Another important property in multi-objective optimization as mentioned previously is the diversity of the solutions to try to cover the Parent front as widely as possible. To deal with this issue a so called *crowding distance* assignment is performed. What this does is add a score to each solution based on how close it is to other solutions in "objective value space". Whenever there is a choice between solutions, for example in the selection procedure, the solution with the smallest front-value is chosen. But for solutions within the same front, for example if we have more candidates for the Pareto front than the population size, we will select the solutions with the highest crowding distance score. The algorithm is defined in Algorithm 1.

Algorithm 1 NSGA-II [26]

```
# Both with population size N
P_0 ← initialize elitist population
Q_0 ← initialize candidate population
evaluate(P_0)
evaluate(Q_0)
while max iteration not exceeded or predefind condition not fulfilled do
    R_t = P_t ∪ Q_t
    # Sort the solutions into fronts
    F = fast-non-dominated-sort(R_t)
    P_{t+1} = ∅
    i ← 1
    # Assign crowding distance and fill P_{t+1} with the best solutions
    while |P_{t+1}| + |F_i| < N do
        crowding-distance-assignment(F_i)
        P_{t+1} ← P_{t+1} ∪ F_i
        i ← i + 1
    end while
    crowding-distance-assignment(F_i)
    # sort by crowding distance operator
    Sort(F_i, ≺_n)
    # fill the last space of P_{t+1} with the solutions with highest crowding distance
    P_{t+1} ← P_{t+1} ∪ F_i[1 : (N − |P_{t+1}|)]
    # generate candidates, the fitness used in the selection is based on front and crowding distance
    Q_{t+1} ← Selection, Crossover, Mutation, Evaluation using P_{t+1}
    t ← t + 1
end while
```

First two populations are generated of the same size, P_0 and Q_0. P will represent the elite solutions and Q the candidate solutions/offspring. Thus, P is deterministically chosen based on the best solutions found yet and Q stochastically chosen based on the standard GA procedures. Thereafter in the main loop these two populations are combined, evaluated based on fast non-dominated sort and crowding distance assignment to generate the new P. From P, Q is generated using the GA operators and then the cycle is repeated. The selection procedure is binary tournament selection. I refer the reader to the original paper

by Deb *et al.* [26] for pseudo code for the two key algorithms, fast non-dominated sorting and crowding distance assignment.

In NSGA-II constraint handling is performed in two ways. Firstly, in the non-dominated sorting a solution that has failed a constraint will be dominated by all solutions that have not. If two solutions have both failed constraints then the one with the smaller overall constraint violation will dominate the other. Secondly, in the selection procedure, solutions with constraint violation will lose against those without in the binary tournament. If they both have violations the smaller violation wins [26].

3.5.2 MOGA Neural Network

NNs have had immense success in the last decade in areas such as computer vision and natural language processing. The power of NNs lie in their abilities to extract information from raw features without feature engineering as well as the many dynamic ways one can shape network-structures. For example FNNs, CNNs, RNNs, DBNs and so on. The common way an NN is trained as previously mentioned is using one of the various versions of backpropagation. While it used to have issues such as the vanishing/exploding gradient problem, these are mostly solved using appropriate activation functions such as ReLU and using skip-connections. Still though, backpropagation has two major weaknesses. Firstly, it requires a differentiable objective function, secondly, it cannot optimize multiple objectives simultaneously by finding the Pareto optimal set. Both of these issues appear in the challenge of HI construction. When constructing an HI we want to optimize characteristics such as monotonicity but these metrics are not always differentiable or have zero gradient (e.g. monotonicity). At the same time, even if we managed to construct and use only differentiable metrics, still the issue of how to optimize multiple objectives exist.

To manage these problems, the parameters of the network will be found using MOGA, specifically NSGA-II, and with this a set of NN parameters are found with the goal of populating the Pareto front. Then we can use a more exact metric to decide between the different solutions. For instance how well a model fits the HIs constructed from the NN or how well a model predicts RUL using the HIs. The method is denoted MOGA-NN.

Applying GA to neural networks

It is not obvious how to apply the GA's operations, especially crossover, to an NN. Since the weights are correlated in sensitive ways due to the graph structure, performing a single/double-point crossover will not yield anything sensible. Instead, a specialized crossover has to be performed.

In this thesis a three-layer NN will be used, that is an input layer, hidden layer and output layer. In three or more layer networks, two networks can act similarly yet have parameters that seem dissimilar. This causes a problem when applying crossover and is often called the *permutation problem* or *competing conventions problem* [27]. One solution to this is implemented by Garcıa-Pedrajas *et al.* [27]. There the crossover is performed with respect to the hidden nodes of the NN. The idea is that each hidden node and the connections from and to it represent one building block. Altering this building block by merging it with another building block will not yield anything reasonable in the sense of crossover. Instead concatenating and/or removing hidden nodes will combine building blocks into a network where the contribution of each hidden node will be intact and useful. Define the ith hidden node in the jth NN in the population as:

$$H_i^j(x) = f(\sum_{k=1}^{K} w_{k,i}^j x_k)$$

Where K is the number of inputs. Then the zth output of the jth NN (given that it is a three-layer NN) is:

$$y_z^j(x) = F(\sum_{i=1}^{N_j} \beta_{i,z}^j H_i^j(x))$$

Where N_j is the number of hidden nodes in the jth NN. Now the zth output of the offspring due to crossover of two or more NNs can be defined as:

$$y_z(x) = F(\sum_{j=1}^{P} \sum_{i=1}^{N_j} \alpha_{i,j} \beta_{i,z}^j H_i^j(x))$$

where P is the number of parents and $\alpha_{i,j} \in \{0, 1\}$ indicates whether the ith hidden note in the jth network is present in the offspring. In [27] the authors see this now as an optimization problem to find the

optimal αs, and expand on methods to solve it such as GA and simulated annealing. Thus, the idea is to, inside the crossover operation of GA, perform another optimization procedure. This is not done in this thesis for two reasons. Firstly, in this case we are dealing with multi-objective optimization, and therefore, it is not as simple to evaluate the candidate offspring. Secondly, this sub-optimization is not very GA-like where offspring usually are generated stochastically. Instead the offspring will be generated randomly by sampling hidden nodes from the parents. Specifically, each hidden node has 50% probability of being part of the offspring and at least one hidden node from each parent will be part of the offspring. Note that in the above definition the bias was not mentioned, but in the method used in this thesis there will be bias added to each hidden node and it is treated as part of the building block.

One major advantage of the proposed method to other crossover operations is that the number of nodes in the hidden layer can evolve over time. Thus, both the weights and structure are optimized to some degree. For the mutation operation Gaussian random noise is added.

3.6 Recurrent neural networks

An RNN is a type of NN used to handle sequential data. The simple RNN contains a recurrent hidden state that is updated based on the current input and the previous hidden state. The update equations are shown below:

$$h(t) = g(W_h x(t) + U h(t-1) + b_h)$$
$$y(t) = f(W_o h(t) + b_o)$$

where $h(t)$ is the hidden state vector at time t, $x(t)$ is the input at time t, W_o, W_h and U are weight matrices, b_h and b_o biases, g is the hidden layer activation function, for example sigmoid or tanh, and f the output activation function that depends on the output. The network is trained using an extended version of gradient descent that propagates gradients backwards through time called backpropagation through time (BPTT). The problem with the simple RNN is that it has been shown to fail to capture long-term dependencies. When the gradients are propagated through a large amount of timesteps the learning process becomes unstable, leading to issues such as the vanishing gradient problem or

more rarely the exploding gradient problem [28]. For this reason several better RNNs have been developed to handle this issue.

3.6.1 Gated Recurrent Unit

A GRU [29] is an RNN with added capabilities to capture both short and long-term dependencies to resolve the issues faced with the simple RNN. It is similar to the Long short-term memory (LSTM) [30], but is simpler to compute and implement [29]. It has been shown to outperform LSTM on some datasets in both performance and convergence time [31]. The update equations look as follows:

$$
\begin{aligned}
r(t) &= \sigma(W_r x + U_r h(t-1) + b_r) \\
z(t) &= \sigma(W_z x + U_z h(t-1) + b_z) \\
\tilde{h}(t) &= tanh(Wx + U(r(t) \odot h(t-1)) + b_h) \\
h(t) &= z(t)h(t-1) + (1 - z(t))\tilde{h}(t) \\
y(t) &= f(W_o h(t) + b_o)
\end{aligned}
$$

where σ denotes the sigmoid function, $\odot$ is the element-wise multiplication, f depends on the output, W, W_r, W_z, U, U_r, U_z, W_o are matrices and b_r, b_z, b_h are bias-vectors that are learned through BPTT. The vector $r(t)$ is called the *reset* gate and decides how much of the previous hidden state to incorporate anew into the hidden state. The vector $z(t)$ is called the *update* gate and controls how much the previous hidden state and the new candidate hidden state, $\tilde{h}(t)$, should influence the new hidden state.

Chapter 4

Method

4.1 The pipeline

The process from raw data to prediction consisted of the following steps:

1. Split the data into training, validation and testing datasets.
2. Extract features.
3. Tune parameters iteratively and train a population of FBD-HI NNs using the MOGA with the training set. Visually validate that the learned functions generalize to the validation data.
4. Find the most suitable solution in the population of FNNs.
5. Fit an RUL prediction model using the generated HIs from the training data and tune hyperparameters (HPs) using the HIs from the validation data.
6. Measure the final performance of the model on the testing dataset

Thus, the HI construction model and RUL prediction model were trained separately. Step 4 and 5 were in this thesis performed in a single step, but could for other circumstances be separated. The feature-extraction process was dataset-dependent and how it was performed for each dataset is explained in section 4.5.

4.2 MOGA-NN for HI construction

An FNN (presented in section 3.2) was used to construct the HI. As previously mentioned the network had three layers. This structure was simple enough not to overfit easily to the data while also with the hidden layer being able to describe various complex transformations of the sensor data. Additionally, the constructed GA crossover operator was specialized for this structure. The approach was a type of FBD-HI. The FNN took the sensor data as input and outputted a vector representing the sensor data in a transformed space. Thereafter the minimum distance, given a selected distance metric, was measured between the given state and the faulty states. Here the last state of each trajectory was considered a faulty state. Thus, there were N, equal to the number of training data trajectories, states to compare with. In situations where N is very large, it should not be a problem clustering the states or selecting a few representatives. Since N was not very large in any of the datasets, this was not performed in this thesis.

Thus, for the proposed method, f in the FBD-HI definition in equation 3.1 was an FNN. The critical difference to other approaches was that the parameters of the FNN and partially the structure (the hidden nodes) were learnt using the MOGA NSGA-II. This was performed using the GA-operators presented in section 3.5.2, to optimize fitness functions that are desired for an HI. Each time the network was evaluated, the HI-curves were constructed for all training data trajectories and assessed using the fitness functions (see section 4.2.3). The result of the optimization procedure were a set of FNNs, each one able to construct HIs using distance.

4.2.1 Baseline

A baseline method was used to verify the value of a transformation function. It was an FBD-HI without transformation function. Thus, the sensor data for each state was used as is (aside from any preprocessing steps) before measuring the distance to the faulty states to generate HIs. Here f in the FBD-HI definition was simply defined as $f(x) = x$.

4.2.2 Distance metrics

To use distance, a distance metric/function had to be selected. For the MOGA-NN the Manhattan distance was used for both datasets. For the baseline the Manhattan distance was also used for both datasets to make the comparison fair. For other datasets other metrics could potentially work better, for example the Euclidean distance.

$$\begin{aligned} \text{Euclidean distance}, d(x,y) &= \sqrt{\sum_{i=1}^{K}(x_i - y_i)^2} \\ \text{Manhattan distance}, d(x,y) &= \sum_{i=1}^{K}|x_i - y_i| \end{aligned} \tag{4.1}$$

where K is the dimension of x and y.

4.2.3 Fitness function

As mentioned in section 2.5, there are many ways to evaluate an HI. Since no metric takes into account all factors of a good HI, it is desirable to use multiple. Hybrid metrics have the weakness of requiring an arbitrary weight to be assigned to each metric. This was solved using the MOGA. The used metrics in this thesis are defined in Table 4.1 and the idea behind each of them explained. Additionally, the way the results were aggregated from the evaluation of each trajectory to one value is specified in the rightmost column. The evaluation process of one solution was performed as follows. First the solution generated an HI-vector for each training data trajectory. Then the ith fitness function was applied to each HI-vector:

$$values_i = [f_i(\overline{HI}_1), f_i(\overline{HI}_2), ..., f_i(\overline{HI}_N)]$$

where $\overline{HI}_j$ is the jth HI-vector generated from the jth training data trajectory. Thus, for every trajectory one value was obtained. Since only one value per fitness function is needed, the values had to be aggregated. This was done using the function, agg, specified in the rightmost column:

$$value_i = agg(values_i)$$

At this point each solution had been assigned a value for each fitness function. These values then determined the domination-front and crowding distance, as specified in the NSGA-II algorithm in section 3.5.1,

which the algorithm used to approximate the Pareto front and evolve the population.

Name	Formula	Idea	Aggregation
Oscillation-penalty	$\lvert\frac{HI_{max} - HI_{min}}{sum(\lvert\text{diff}\rvert) + \epsilon}\rvert$ where diff $= HI_{1:T} - HI_{0:T-1}$	Penalizes oscillations. The percentage of change that was moving the HI. The ϵ is just a small number to avoid divide by zero.	Mean
Mon_n (monotonicity with step size n)	$\lvert\frac{\text{No. of diff}_n > 0}{T-n} - \frac{\text{No. of diff}_n < 0}{T-n}\rvert$ where $\text{diff}_n = HI_{n:T} - HI_{0:T-n}$	The noise sensitivity problem is reduced. For longer time series, the one-step difference is expected to be very small, but for several steps more significant.	Mean
Second-mon_n (second-degree monotonicity with step size n)	Monotonicity of the first difference: $\text{Mon}_n(\text{diff})$ where diff $= HI_{1:T} - HI_{0:T-1}$	If there is indication that the HI is expected to be increasing faster over time this metric would favour such solutions.	Mean
Trendability	See equation 2.2	Favours a linear trend.	Mean/min
Spearman-trendability	See equation 2.3	Favours nonlinear increasing or decreasing trends.	Mean/min

Table 4.1 – Used fitness functions explained.

For monotonicity both Mon_1 and Mon_{10} were used in the experiments. For second-degree monotonicity, Second-mon_{10} was used. Here for the two trendability-metrics the minimum was used as the aggregation function for the first dataset (described in section 4.5.1) where the trajectories were fairly consistent. Then a minimum was useful to steer the solutions to achieve a good trendability for every single trajectory. In the second dataset (described in section 4.5.2) there were two different failures with different characteristics. If the minimum was used the value of the metric would only be affected by the type of failure with least trendability. Instead, to obtain a better assessment of the trendability of all trajectories, the mean was used.

4.2.4 Normalizing the HI

After having created a model that can generate HIs from sensor data, it was useful to normalize the HIs. This made it easier to visually compare HIs and for the RUL prediction model to digest the data. The normalization was performed by first extracting the maximum and minimum value of the HIs generated from the training data. Then each HI was min-max scaled based on these values.

$$MinMaxScale(HI) = \frac{HI - HIs_{min}}{HIs_{max} - HIs_{min}} \tag{4.2}$$

Of personal preference the HI was also modified to increase with a simple change:

$$HI = \begin{cases} 1 - MinMaxScale(HI), \text{ if } HIs \text{ are decreasing over time} \\ MinMaxScale(HI), \text{ if } HIs \text{ are increasing over time} \end{cases} \tag{4.3}$$

Thus, each HI from the training data was between zero and one and increased over time (if the HI was approximately monotonic). For the testing/validation data the min and max extracted from the training data HIs were used for the normalization.

4.2.5 Tuning the MOGA-NN

There were several parameters that had to be decided: the number of nodes in the FNN, the activation function, the fitness functions, constraints on the fitness functions, the distance metric, the crossover rate, the mutation rate and the population size. Because the crossover operation of the GA could increase and decrease the number of hidden nodes, the initial hidden nodes was not very important. What was fixed on the other hand was the dimension of the transformed space where the distance was calculated. The difficult part was also that there were multiple objectives which means that it was challenging to determine which option was the best. Because of this difficulty, the many different parameters and the time it took to run a complete optimization, no exhaustive tuning was performed.

Initially a small population was run using a small number of iterations to test some of the parameters. The maximum achieved value for the fitness functions in combination with the visualization of the HIs were

used to compare the different settings. Then a few candidates were run with a larger population size and more generations. Overall, the performance of the method was not very sensitive to most parameters and for this reason not much tuning was required. This is discussed more in section 6.3.2. One of the more critical parameters that improved the RUL prediction was adding a constraint to the trendability, especially for the first dataset. This way the population of solutions were more narrowly positioned in a part of the search space where the solutions were more successful in terms of RUL prediction with the GRU.

4.3 GRU for RUL prediction

Finally, RUL was predicted using the generated HIs. In this thesis this was done using a GRU. The training data consisted of trajectories of HI and RUL pairs: $(HI_{1:T}, y_{1:T})$ where y_t was the RUL target at time t. At time t the GRU had processed the data from the initial health to the tth point $(HI_{1:t})$ and outputted a prediction $\hat{y}_t$. Thus, this model was not threshold based. The error/loss was then measured in root mean square error (RMSE), $\sqrt{\frac{1}{T}\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}$, and weights updated through BPTT. The final activation function (f in section 3.6.1) was the ReLU, $\text{ReLU}(x) = \max(0, x)$, since RUL is non-negative. The steps from features to RUL are shown in Figure 4.1.

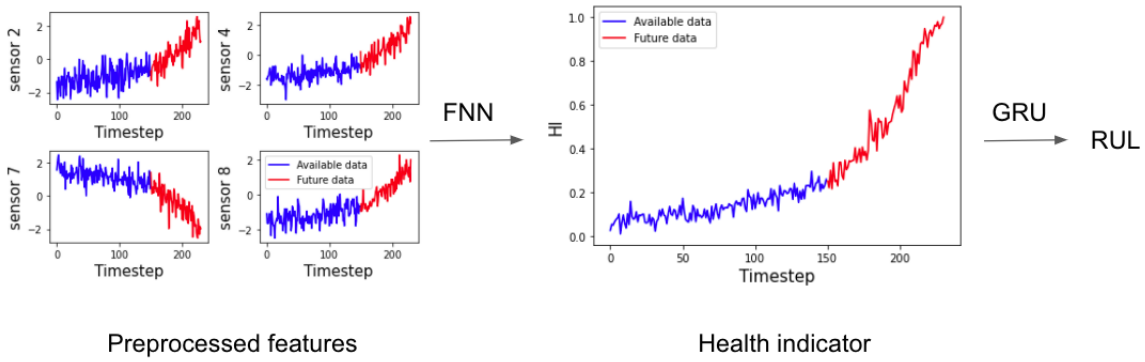


Figure 4.1 – How predictions were made after fitting and selecting an FNN from the MOGA and fitting a GRU using BPTT. Note that the FNN was not directly outputting the HI, but a vector where the minimum distance to the faulty states (also processed using the FNN) was generating the HI. Given that we are at the 150th timestep, the only data available for prediction is the data in blue. The shown data is described in section 4.5.1.

4.3.1 Piecewise linear RUL

One problem with fitting an RUL prediction model is that often in the early stages it is very difficult to forecast the RUL since the degradation is minimal while the RUL still evolves linearly. To combat this and improve model training a commonly used method is to change the RUL by transforming it into a piecewise linear target. The new RUL is the same for shorter time frames, but for longer time frames a limit is applied:

$$\widehat{\mathrm{RUL}}_t = \min(\mathrm{RUL}_t, L) \tag{4.4}$$

where L is the specified threshold. The new target RUL for $L = 100$ is shown in Figure 4.2. This approach was adopted in this thesis.

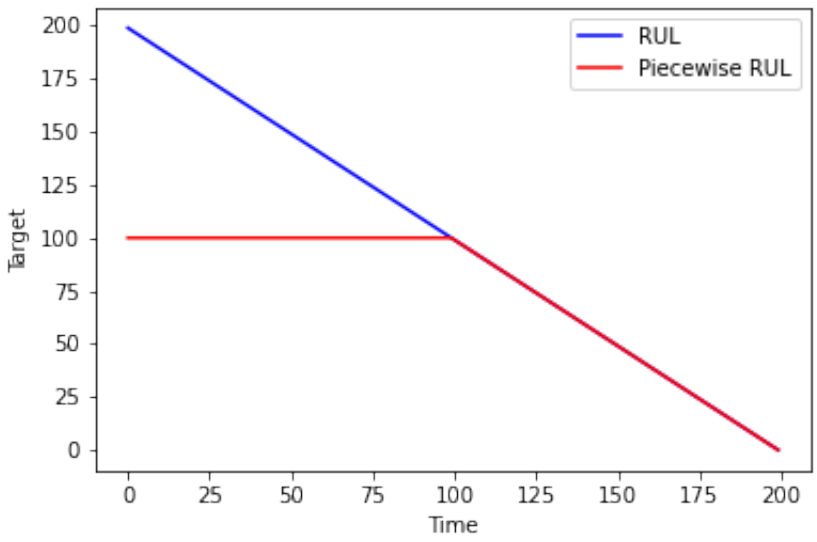


Figure 4.2 – Difference between piecewise linear target RUL and actual RUL.

4.3.2 Finding the most suitable solution and training the RUL prediction model

These steps could have been performed differently if a different model was used such as an SMM. Since a GRU was used, the model was relatively time-consuming to fit as opposed to SMMs. Also there were a few HPs that had to be adjusted which meant the required number of experiments would become large if all options had to be exhausted. At

the same time even if all solutions were fit we are dealing with a stochastic optimization procedure. To combat this the search for the optimal HPs was performed using a GA. The index of the HI constructing solution in the population was also treated as an HP. Thus, one search algorithm was used to both find the HPs for the prediction model and select the FNN for HI construction. The fixed part of the search was as follows:

- Loss function: RMSE
- Optimizer: Adam [32]
  - $\eta = 0.01$
  - $\beta_1 = 0.9$
  - $\beta_2 = 0.999$

The HPs to be tuned were:

- The solution in the population: index between one and size of approximated Pareto front
- The number of hidden nodes: between 8 and 64
- The batch size, that is the number of trajectories fed into the network in each batch: between 2 and the number of trajectories in the training data (max 80 in both of the datasets)

The fitness function/evaluation criterion was the smallest achieved validation loss. To avoid unnecessary computation, the technique known as *early stopping* was used. This meant that the optimization procedure was stopped if the validation loss had not improved in a fixed number of iterations, here 50.

In the used GA, the mutation operation and the initialization of the parameters were performed by simply sampling uniformly from the range of each parameter. The crossover of two parents was done as follows for each parameter independently:

$$w \sim U(0, 1)$$
$$child = (1 - w)p_1 + wp_2$$

where $child$ is the child-parameter and $p1$ and $p2$ are the parameter values of the first and second parent, respectively.

For the batch size and the number of hidden nodes an integer is needed and for this reason the final number was rounded to the nearest integer. The indices used for the specific FNN had no order, and thus, it would have been pointless to take a weighted average of them. Instead the weighted average of the vectors of fitness values was taken. That is the fitness values achieved from the evaluation of the training data HIs produced by the parent FNNs. To get the offspring index the FNN whose fitness values most resembled the produced vector, calculated by minimum Euclidean distance, was picked. Let the vector of fitness values for one solution be defined as:

$$v_i = [f_1(HIs_i), f_2(HIs_i), f_3(HIs_i), ...]$$

where HIs_i is the HIs produced from the training data by the ith FNN and f_j the jth fitness function. The crossover of the ith and jth FNN was then:

$$\begin{aligned} w &\sim U(0,1) \\ v &= (1-w)v_i + wv_j \\ child &= \arg\min_k d(v_k, v) \end{aligned}$$

where d is the Euclidean distance and *child* the index of the child-FNN.

For selection, binary tournament selection was used. Overall, this kind of search is less time-consuming than doing a grid search and more efficient than a random search.

The used parameters for the GA were the following:

- 5 generations
- 8 population size
- 0.9 crossover rate
- 0.2 mutation rate

It was run twice and the parameters achieving the lowest validation loss chosen.

4.4 Measuring the performance of the model on the test dataset

In the previous step an FNN for HI construction was selected and optimized HPs for the GRU found. To get a good representation of the model performance the following process was then performed:

1. Three times in a row fit the GRU with the found HPs and early stopping using the HIs constructed from the FNN
2. Pick the model that gave the lowest validation loss
3. Measure the test data RMSE

This was done 10 times and the resulting RMSEs define the model performance. This way some of the variability of the stochastic optimization process was removed while also taking into account that different fits lead to different test errors.

4.5 Datasets

To answer the research questions two datasets were explored the using the proposed method and baseline.

4.5.1 Turbofan Engine Degradation Simulation

The Turbofan Engine Degradation Simulation (TEDS) Dataset was simulated using the Commerical Modular Aero-Propulsion System Simulation developed by NASA and is a common benchmark for prognostics methods [33]. A diagram of a turbofan engine is shown in Figure 4.3.

In this thesis the first of the four datasets, FD001, was used to test and compare the proposed method with other methods. It contained 100 training trajectories and 100 test trajectories for different engines of the same type. The training data was split into 80 training trajectories and 20 validation trajectories. For the test trajectories only a partial trajectory was available together with the RUL. The dataset contained 21 sensors together with three operating settings. In FD001 there was only one operating condition and also some sensors were constant over time. Thus, the operating settings and the constant sensors were removed. Left

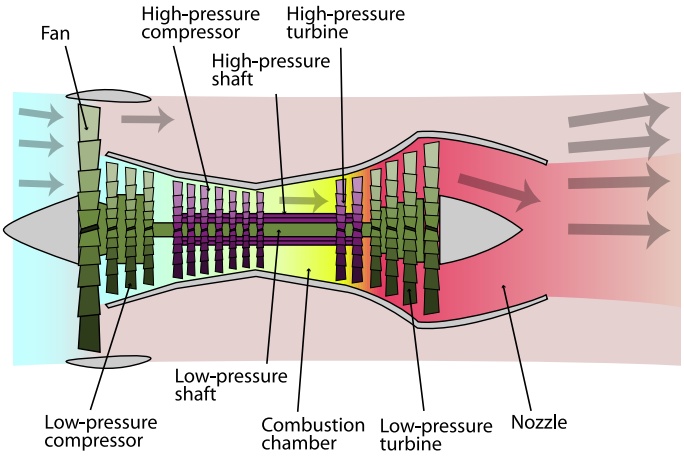


Figure 4.3 – Schematic diagram of a turbofan engine. Image source: "Turbofan operation" by K. Aainsqatsi / CC BY-SA 3.0 [34].

were 14 sensor measurements with indices 2, 3, 4, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20 and 21. Four sensors for one trajectory are shown in Figure 4.4.

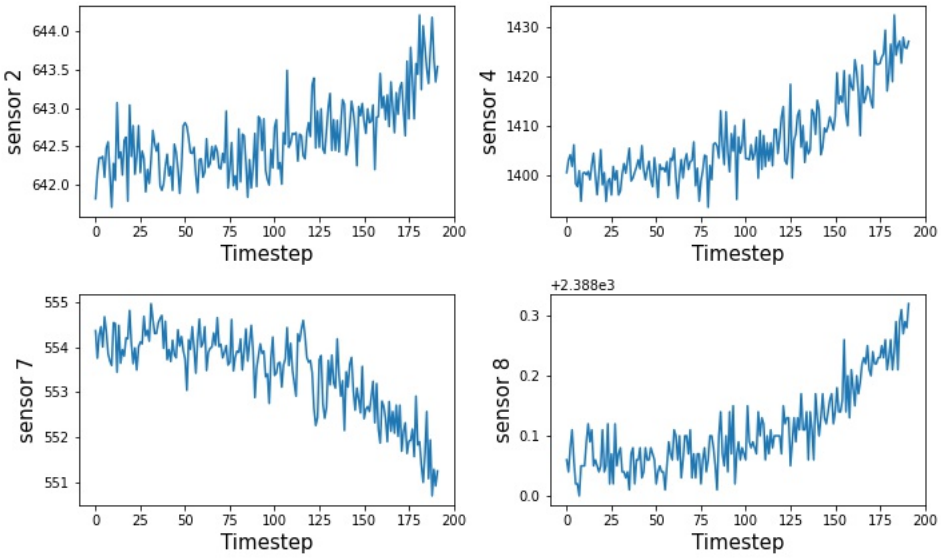


Figure 4.4 – Four TEDS dataset sensors visualized for one trajectory.

The sensor measurements were standard scaled:

$$z = \frac{x - \mu}{\sigma} \tag{4.5}$$

This is a common operation when training NNs with gradient descent

but it is also important here since the initialisation of the NN's weights and the adding of Gaussian noise as the mutation operator could be of a fixed scale. Note that for the validation and test data the mean and standard deviation extracted from the training data were used to standard scale the measurements.

Parameters

For the TEDS dataset the following parameters were used for the FBD-HI using the MOGA-NN:

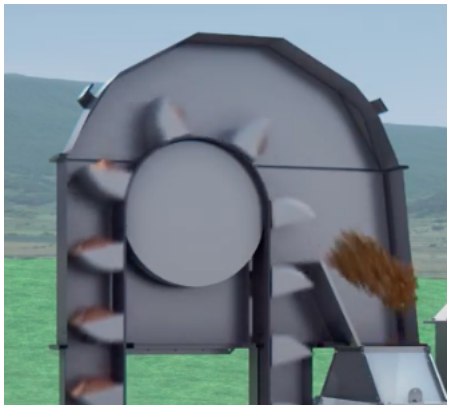
- population size of 100
- crossover rate of 0.9
- mutation rate of 0.05
- 16 hidden nodes
- 8-dimensional transformed space
- ReLU activation function
- initialization using $N(0, 1)$
- mutation using $N(0, 1)$
- Manhattan distance
- Fitness functions, aggregation type:
  - oscillation-penalty (mean)
  - trendability (min)
  - Spearman-trendability (min)
  - Mon_1 (mean)
  - Mon_{10} (mean)
  - Second-mon_{10} (mean)
- Constraints: trendability > 0.8
- Piecewise limit of 130 (same for the baseline method)

For the baseline, Manhattan distance was also used.

4.5.2 Elevator weight simulation dataset

This dataset was simulated by mimicking the weights/force exerted on an elevator inside a grain storage facility. It is here denoted the elevator weight simulation (EWS) dataset. A graphical representation of such an elevator is shown in Figure 4.5. The failure was due to either the loosening of the elevator belt that is moving buckets up and down with grain. Or other disturbances such as objects being stuck in sensitive areas causing increased tension between the motor pulling the belt and the belt. An image of the time series of weights for one batch is shown in Figure 4.6. As can be seen the weight is initially low before the grain begins to be transferred, then it increases and stays on the same level for a bit, scooping up and dropping off grain iteratively. Finally, there is no grain left in the batch and the level drops down.

(a) The elevator picks up the grain



(b) After elevating the grain it is thrown to the next part in the transportation process

Figure 4.5 – Graphical representation of elevator transporting grain.

In the simulation the idea was that as the belt loosens, the weight at the lowest point would decrease until a point where it failed. At the same time, independently, objects could get stuck in sensitive areas inside the elevator causing increased pressure on the motor. This was shown as "spikes" in the weight pattern and could cause a different error. Thus, there were two simultaneous failures that could occur. Similar to the TEDS dataset, there were 80 training trajectories, 20 validation trajectories and 100 test trajectories. This time the test trajectories were not partial, but complete.

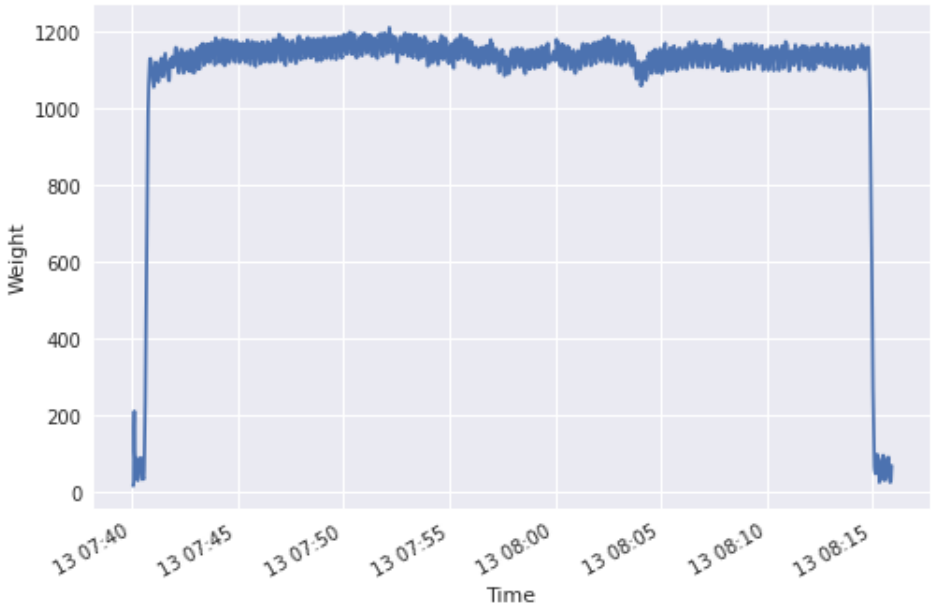


Figure 4.6 – Real elevator weights captured using a sensor.

How it was simulated

The simulation was performed as follows. Firstly, two independent degradation curves were generated using Algorithm 2.

	z	p_1	p_2	r_1	r_2	s_1	s_2
Expected failure	1.01	0.005	0.01	0.03	0.07	0	0.1
Shock failure	1.0	0.001	0.002	0.1	0.15	0.2	0.4

Table 4.2 – Parameters for the degradation curves.

The parameters for the two degradation curves/different failure types are shown in Table 4.2. The crucial difference here was that $z = 1.0$ for the shock failure. This meant that the degradation would not start (x did not change) before a shock occurred (s increased) since:

$$x_i = x_{i-1} - r(x_s - x_{i-1} + s) = \{x_s = x_{i-1}, s = 0\} = x_{i-1}$$

Also notice that the rate (r_1, r_2) and shock size (s_1, s_2) were higher for the shock failure but the probability of shock (p_1, p_2) was much smaller. Thus, the expected failure was anticipated to happen sooner or later while the shock failure was a more rare event. But when the shock degradation

Algorithm 2 Degradation simulation: $r_1, r_2, p_1, p_2, s_1, s_2, z$ are parameters. The final series of xs: $x_0, x_1, ...$ is the degradation patttern. The length varies stochastically.

$s \leftarrow 0$

$x_0 \sim U(3.5, 5)$

$q \sim U(r_1, r_2)$

$r \leftarrow \frac{5q}{x_0}$

$p \sim U(p_1, p_2)$

$x_s \leftarrow z \cdot x_0$

$i \leftarrow 1$

while $x_{i-1} > 0$ **do**

  $q \sim U(0, 1)$

  if $q < p$ **then**

    $a \sim U(s_1, s_2)$

    $s \leftarrow s + a$

  end if

  $x_i = x_{i-1} - r(x_s - x_{i-1} + s)$

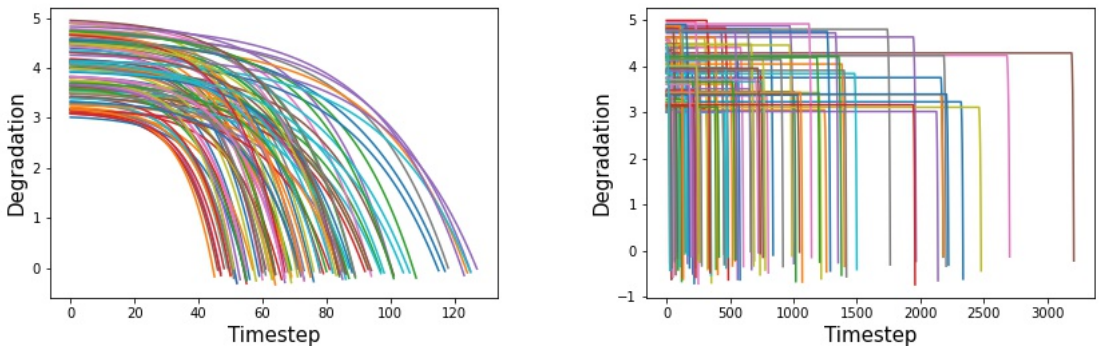
  $i \leftarrow i + 1$

end while

started it progressed quickly. See Figure 4.7 for the degradation curves of the two events. As can be seen, the variance of the time from start to failure was very high for the shock failure while the expected failure was more consistent.

The simulation was performed for both degradation curves simultaneously until one of them fell to zero or below and the simulation was stopped. About 10% of the time the shock failure caused the simulation to stop and the rest of the times the expected failure.

At each time point the two degradation values were then used to construct one batch of elevator weights. The expected degradation state affected the weight when there was no grain in the elevator. Specifically, the empty weight was two times the degradation state. The shock degradation state affected the oscillations when the elevator was



(a) Expected failure

(b) Shock failure

Figure 4.7 – The two types of simulated degradation curves.

transferring the grain by causing shocks. This was manifested in a stochastic manner in the system based on the probability $(x_s - x)/x_s$ every 7th timestep (representing the time for the elevator to turn one lap). At the same time the weights were masked with noise. The code to generate the elevator weights was added as an appendix.

Finally, components were extracted from the generated weights using simple heuristics and different features extracted from each. The pipeline is visualized in Figure 4.8. In summary, two degradation curves were generated and stopped after one of them reached the threshold. For each timestep the states generated weight data. From this weight data, parts were extracted and features calculated to create the dataset. One added difficulty was that at some points the initial or last part of the time series of weights were missing. Then all features from those components were set to zero and a dummy-encoded indicator (one if missing, zero if present) marked that the features were missing.

The extracted features are listed below:

- The initial and last part (before/after grain enters/leaves the elevator):
  - min
  - max
  - mean
  - missing indicator
- The middle part (when the system is filled with grain):

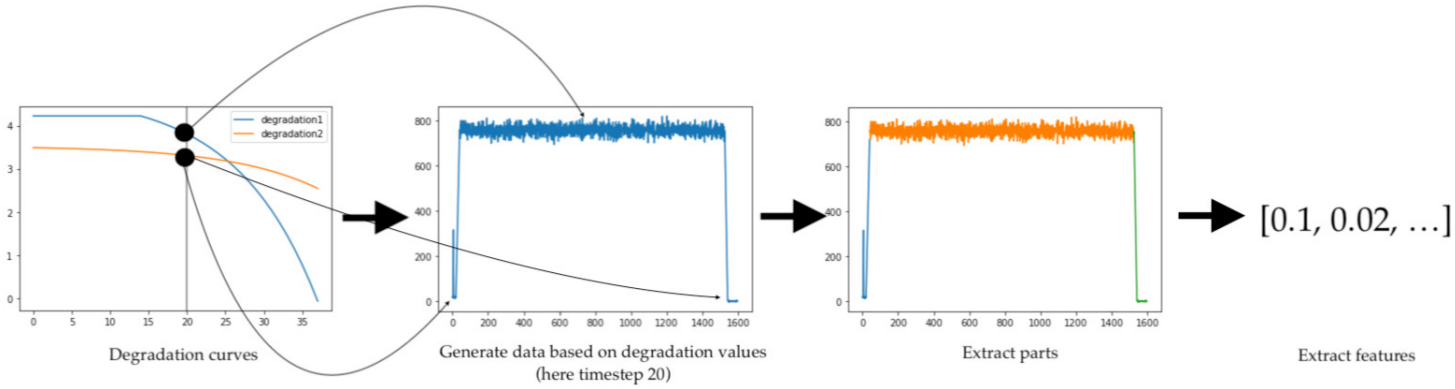


Figure 4.8 – Pipeline for the simulation.

– variance of the first difference

Thus, there were in total nine features. Four of them are visualized for one trajectory in Figure 4.9. In the top row two features related to the expected failure are shown. These were very noisy, but also go to zero sporadically. This was due to the missing data as previously mentioned. The time series in the bottom-left shows when the data was missing (one = missing, zero = not missing). At the bottom-right the single feature that was related to the shock failure is shown. It is much easier to see how the degradation evolves but there were far fewer failures of that type. Finally, the extracted features, like the TEDS dataset, were standardized using equation 4.5.

One objective with this dataset was to explore how the proposed method and baseline handle multiple errors and competing risks. In this case there were really two errors that could be monitored separately (since they were independent). Thus, practically it could be more effective to generate two HIs, one for each failure type. Here it was assumed though that all this information about the number of failures and how the features were related to each failure was unknown to investigate how the method behaved. In practice this would be useful when, for example, maintenance is done simultaneously for all components and there is no individual adaptation. Also if there is only one type of failure but it could occur in a multitude of ways with different characteristics. Competing

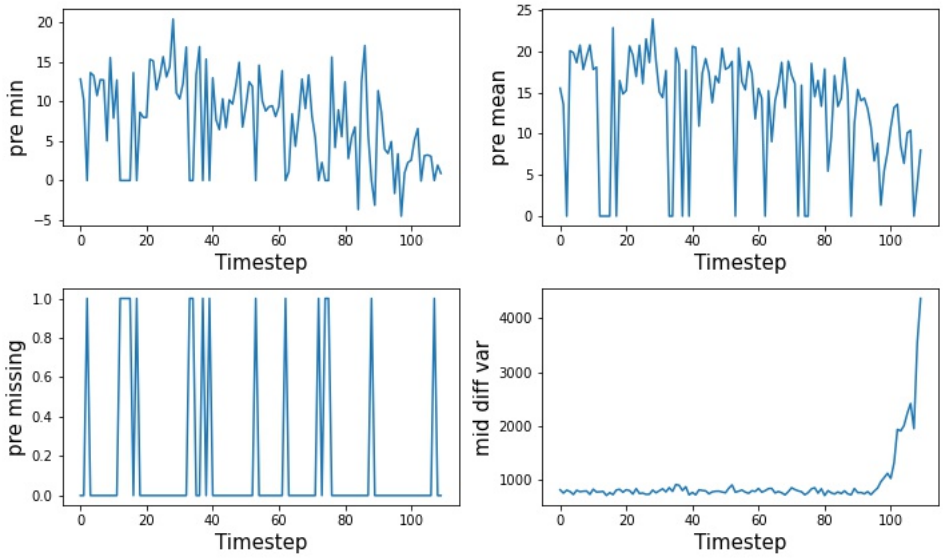


Figure 4.9 – Four EWS dataset features visualized for one trajectory. Here the failure was caused by shock.

risks are common in survival analysis and healthcare where a person can for instance be at risk of dying due to different diseases.

Parameters

For the EWS dataset the following parameters were used for the FBD-HI using the MOGA-NN:

- 500 generations
- population size of 100
- crossover rate of 0.9
- mutation rate of 0.05
- 16 hidden nodes
- 2-dimensional transformed space
- ReLU activation function
- initialization using $N(0, 1)$
- mutation using $N(0, 1)$

- Manhattan distance
- Fitness functions, aggregation type:
  - oscillation-penalty (mean)
  - trendability (mean)
  - Spearman-trendability (mean)
  - Mon_1 (mean)
  - Mon_{10} (mean)
  - Second-mon_{10} (mean)
- Constraints: trendability > 0.1
- Piecewise limit of 80 (same for the baseline method)

For the baseline, Manhattan distance was also used.

Chapter 5

Results

5.1 Turbofan Engine Degradation Simulation

5.1.1 HIs

Baseline

First the baseline approach, described in section 4.2.1, was applied to the TEDS dataset. To apply the method, only the faulty states from the training data were needed. Then the HIs were generated by taking the minimum distance from each state to any of the extracted faulty states. The results from this approach are shown in Figure 5.1. On the left are the generated validation HIs, each color represents one trajectory. In the center the states from three validation trajectories (blue, green and orange) projected onto the two first principal components are visualized. Additionally, the faulty states (from the training data) are shown with larger size in red. PCA was used to approximately visualize the state space since the real state space was 14-dimensional. On the right the HIs of the same trajectories as in the center plot are shown. These two plots show three trajectories instead of all of them to avoid the plot from becoming too cluttered. The trajectories are chosen based on the visualization of the state space so as to show trajectories that do not overlap much. In Figure 5.2 & 5.3, shown for the proposed method in the next section, the same trajectories are displayed but in a transformed space.

The results are satisfactory in the sense that the HIs appear to be approximately monotonic and consistent, and the state space seems structured. From a visual perspective, it is not immediately obvious

what could be improved with the results aside from reducing the noise. The achieved fitness values for the training data are shown in Table 5.1. These results were used as references to compare the proposed method with, and are revisited in the next section.

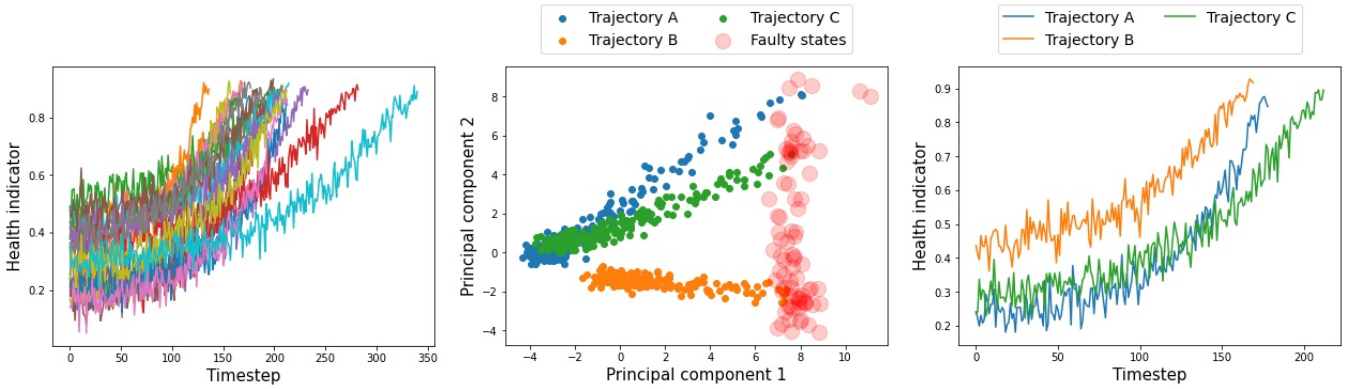


Figure 5.1 – Visualized results for the TEDS dataset using the baseline, FBD-HI without transformation function. Left column: all 20 validation HIs. Middle column: state space visualized using PCA for three validation trajectories and faulty states. Right column: three validation HIs, corresponding to the ones visualized in the middle column.

Metric / Solution	Oscillation-penalty	Trendability	Second-mon$_{10}$	Spearman-trendability	Mon$_{10}$	Mon$_1$
Baseline FBD-HI	0.090	0.82	0.0086	0.84	0.38	0.056

Table 5.1 – Fitness values obtained using the FBD-HI without transformation function (the baseline) applied to the TEDS dataset for the training data.

MOGA-NN

The proposed method, described in section 4.2, was then applied to the TEDS dataset. As before, the faulty states were needed. In addition, with this method several FNNs have been trained using the MOGA to transform the sensor data to vectors in a new state space. Then the distances between the states and faulty states in this space over time generated HIs that were optimized with respect to the fitness functions.

In the parameters, listed in section 4.5.1, there was a constraint on the trendability which improved the RUL prediction performance using the GRU. This is not necessarily optimal for other models. For this reason the results produced by the solutions resulting from the MOGA with a

smaller constraint of trendability > 0.1 are first shown in Figure 5.2. Three solutions that produced HIs from the training data that resulted in the best Mon_{10}, trendability and oscillation-penalty are shown in the same way as the baseline in Figure 5.1. As before, PCA was used to visualize the states.

As can be seen from the HIs in the left column of each row in the figure, the solutions showed a lot of diversity. For both trendability constraints ($> 0.1, > 0.8$), the maximum size of the Pareto front was achieved, equal to the population size (100). With six fitness functions this was rather expected. Thus, in total there were 100 solutions like these, but many were highly correlated.

In Figure 5.3 the more constrained results (trendability > 0.8) are shown. In the bottom-row of Figure 5.2 and Figure 5.3 the solutions that achieved the highest oscillation-penalty are shown. Comparing them, it can be seen that the HIs (right and left column) in Figure 5.3 rise more consistently in the beginning stages because of the forced trendability and consequently reduced oscillation-penalty. If the best solution has a high trendability this constraint improves the search because of the shrunken search space.

In the top row of Figure 5.3 the best solution found using the HP search is shown. It is not obvious why this solution was chosen in particular. One observation from the HIs in the left column, was how the increasing trend wore off at the end of the HI before failure. An hypothesis was that the GRU used this information. This highlights one of the strengths of the optimization through the MOGA. It generates several solutions, increasing the probability of generating one that is compatible with the model of choice and achieves a good performance. In Figure 5.4 a histogram of the number of hidden nodes in the solutions is shown. It can be seen that the solutions increased the number of hidden nodes from the starting point of 16.

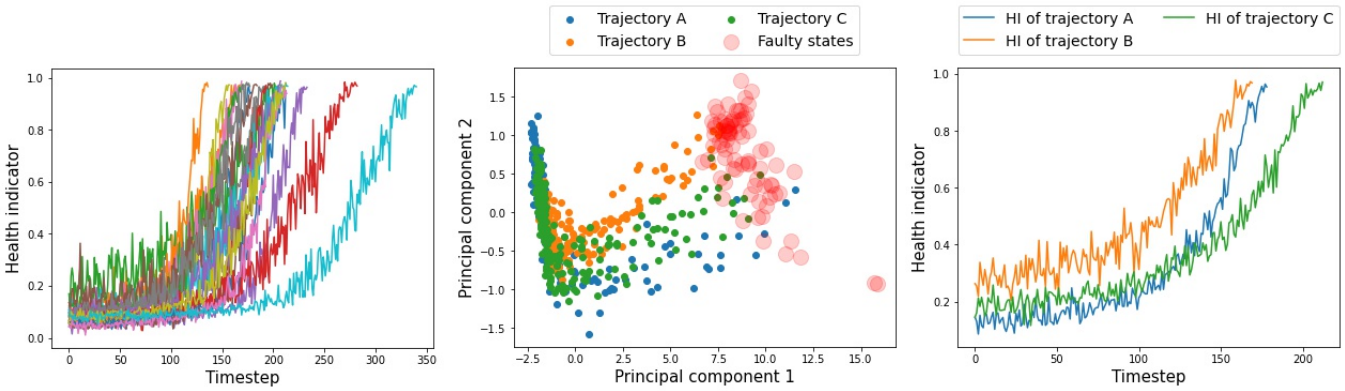
The final fitness values (of the training data HIs) of the best solution chosen by the HP search and the solutions with the highest score on any individual metric are shown in Table 5.2. By iterating through and comparing the fitness values obtained from all the MOGA-NN solutions (not only the ones shown in Table 5.2) with the fitness values obtained with the baseline, it was shown that the baseline was not dominated by any of the solutions from the MOGA-NN in regard to the training data. In turn the baseline would be part of the approximated Pareto front. The same comparison was done but with the fitness values achieved

from the validation data. In this case the baseline was dominated by 3 solutions. The fact that just a few solutions dominated the baseline for the validation data and none in the training data is not unexpected with six metrics. The NSGA-II algorithm uses crowding distance to spread the solutions out across the front and because the number of solutions are finite there is a lot space left . Comparing the state space of the baseline with the spaces generated by the MOGA-NN, one clear difference was that in the baseline each trajectory was more separated. If this is something that better facilitates RUL prediction is uncertain.

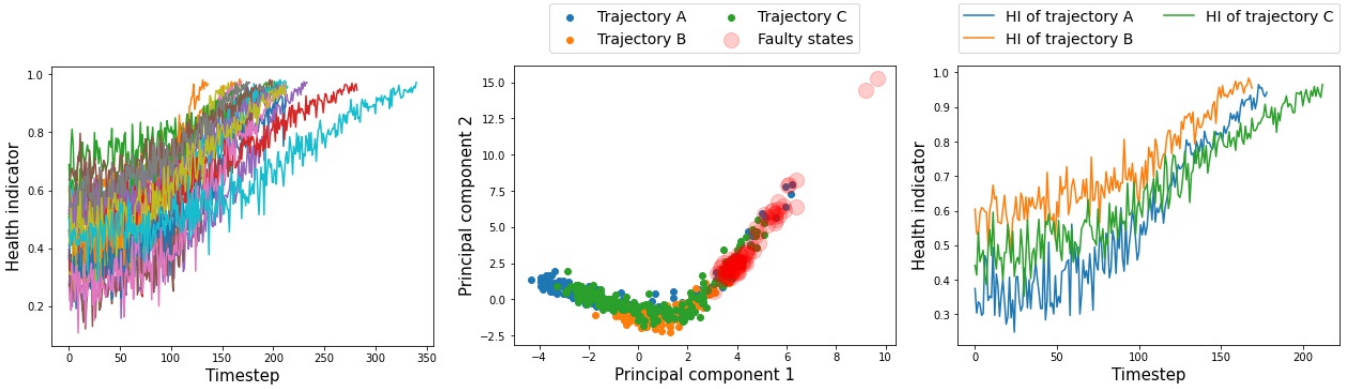
Overall, the only obvious advantage that can be seen was the fact that the MOGA generated several solutions with different characteristics. That being said, the true objective was to predict the RUL and this comparison is made in the next section. To understand what kind of metrics the final solution selected for the GRU favoured, the sum of each subset of fitness values for the training data were sorted. Then the position of the chosen solution was extracted. It was found that the chosen solution had the highest score of the combination of Spearman-trendability and oscillation-penalty.

Metric / Solution	Oscillation-penalty	Trendability	Second-mon$_{10}$	Spearman-trendability	Mon$_{10}$	Mon$_1$
Best GRU	0.10	0.82	-0.0028	0.90	0.41	0.055
Best oscillation-penalty	0.11	0.80	0.0025	0.88	0.41	0.052
Best trendability	0.069	0.91	0.0079	0.90	0.39	0.059
Best second-mon$_{10}$	0.068	0.85	0.029	0.84	0.33	0.039
Best Spearman-trendability	0.073	0.90	0.0094	0.91	0.39	0.058
Best Mon$_{10}$	0.10	0.82	0.0056	0.88	0.42	0.057
Best Mon$_1$	0.088	0.85	0.0071	0.87	0.38	0.074

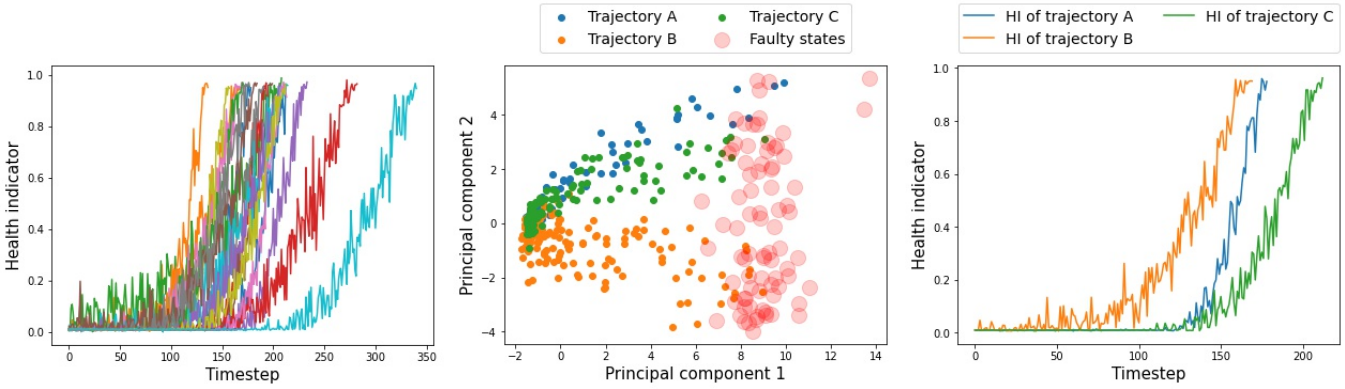
Table 5.2 – Fitness values obtained using the FBD-HI MOGA-NN applied to the TEDS dataset with a constraint of trendability > 0.8 for the training data.



(a) The solution with the best achieved Mon_{10} on the training data.

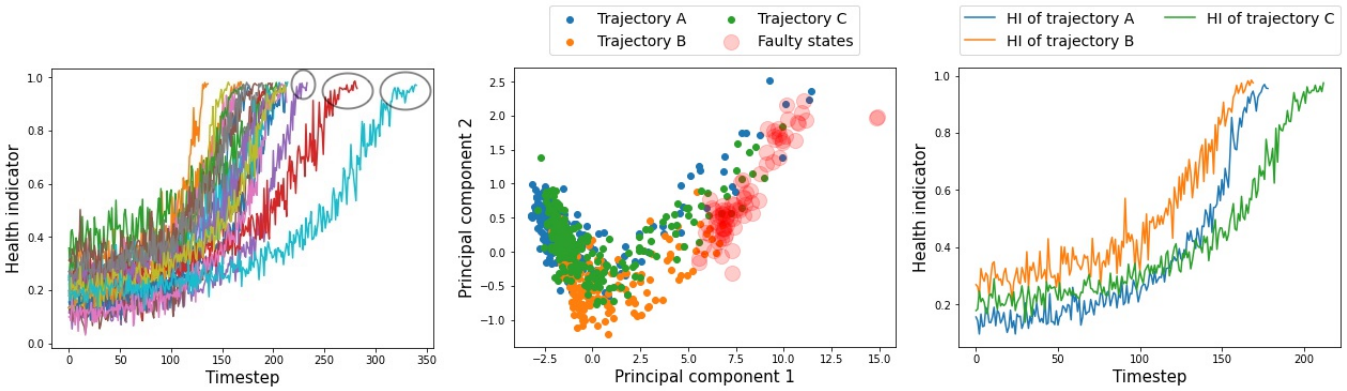


(b) The solution with the best achieved trendability on the training data.

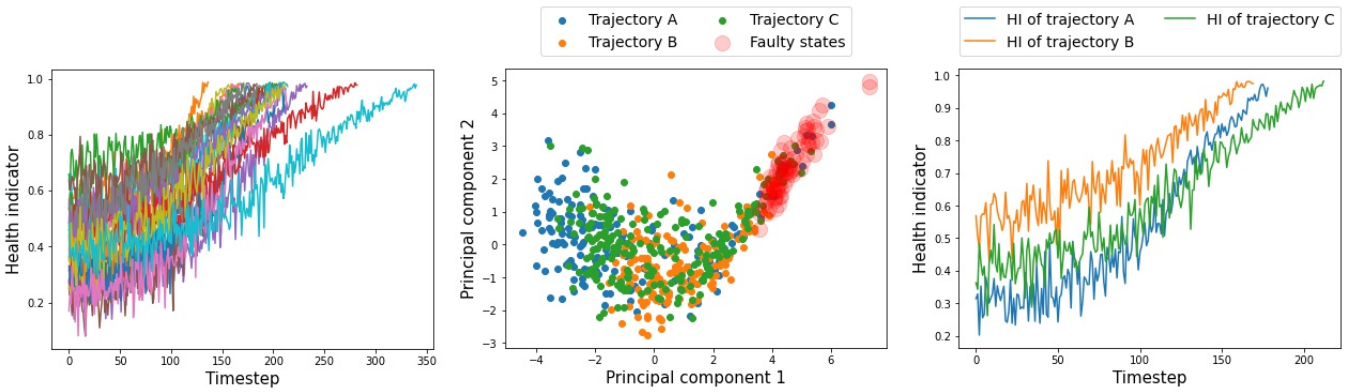


(c) The solution with the best achieved oscillation-penalty on the training data.

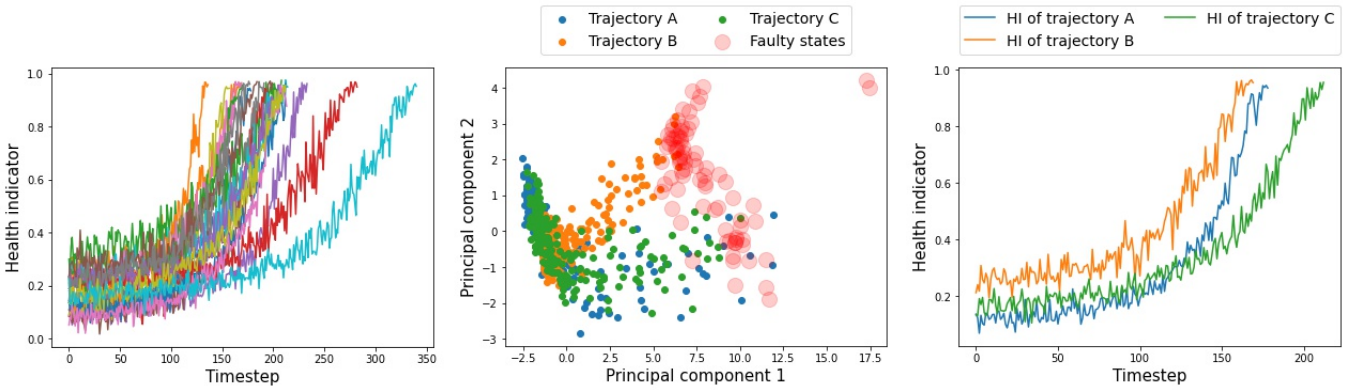
Figure 5.2 – Visualized results for the FBD-HI MOGA-NN with trendability > 0.1 applied to the TEDS dataset. Each row shows the results of one solution. Left column: all 20 validation HIs. Middle column: transformed state space visualized using PCA for three validation trajectories and faulty states. Right column: three validation HIs, corresponding to the ones visualized in the middle column.



(a) The best found solution with the HP search.



(b) The solution with the best achieved trendability on the training data.



(c) The solution with the best achieved oscillation-penalty on the training data.

Figure 5.3 – Visualized results for the FBD-HI MOGA-NN with trendability > 0.8 applied to the TEDS dataset. Each row shows the results of one solution. Left column: all 20 validation HIs. Middle column: state space visualized using PCA for three validation trajectories and faulty states. Right column: three validation HIs, corresponding to the ones visualized in the middle column. Marked in the top-left is a peculiarity of the solution selected by the HP search.

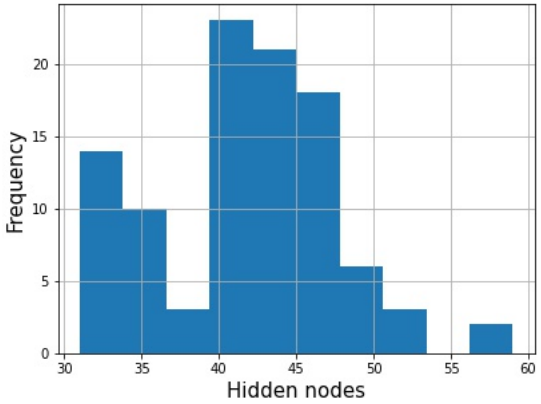


Figure 5.4 – Histogram of the number of hidden nodes in the solutions constructed using the FBD-HI MOGA-NN for the TEDS dataset with trendability > 0.8.

5.1.2 RUL Prediction

Lastly, an HP search was performed to select one solution from the population found using the MOGA and HPs for the GRU. The results from the chosen FNN are shown in the top row of Figure 5.3. The found GRU parameters were 27 hidden nodes with a batch size of 41. An HP search was also conducted for the baseline without the additional solution-parameter. The resulting HPs was 13 hidden nodes and a batch size of 62. In Figure 5.5 predictions against the target RUL are plotted for the GRU using HIs from the selected solution from the MOGA for three validation trajectories.

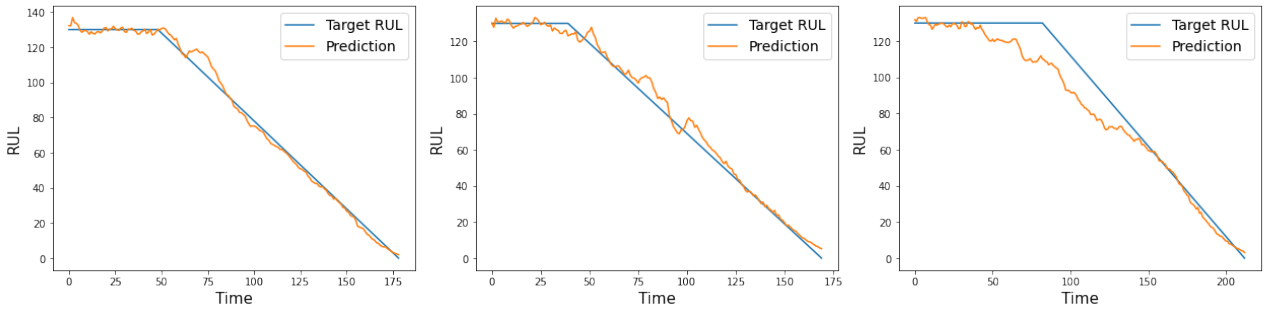


Figure 5.5 – Predictions using the FBD-HI MOGA-NN with GRU, plotted against the target RUL for three validation trajectories.

The models were then trained and tested 10 times each as explained in section 4.4. The test results in RMSE for the GRU using HIs from the proposed method and the baseline are shown in the form of boxplots in Figure 5.6. A nonparametric hypothesis test, the Wilcoxon rank-sum test, was also performed to assure that the difference between the errors was significant ($p < 0.0002$). Thus, the evidence suggested that the added NN transformation function improved RUL prediction.

The mean RMSE for the tested methods and the comparison with several benchmarks for the dataset is shown in Table 5.3. From the benchmarks it can be seen that the performance of the proposed method was competitive, but there were a couple of approaches that reported a smaller RMSE.

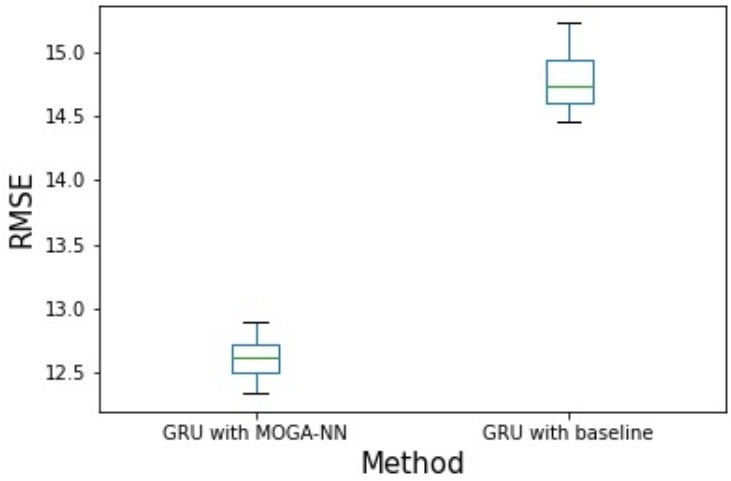


Figure 5.6 – TEDS dataset test results in RMSE. Two boxplots are shown, one for the GRU using HIs from the baseline FBD-HI, and one for the GRU using HIs from the MOGA-NN FBD-HI. Each method was tested 10 times.

Method	**RMSE**
MLP [9]	37.56
SVR [9]	20.96
RVR [9]	23.80
CNN [9]	18.45
Deep LSTM [7]	16.14
FMLP [35]	13.36
RBM-Bi-LSTM-Sim [4]	13.28
Deep-CNN [8]	12.61
Dual-task LSTM [6]	12.29
1-FCLCNN-LSTM [12]	11.17
2-S-MIC-TCNN-AM [11]	10.45
FBD-HI-MOGA-NN-GRU	**12.61**
FBD-HI-Baseline-GRU	**14.80**

Table 5.3 – Benchmarks for FD001. The second to last row shows the result of the proposed method and the last row the result of the baseline.

5.2 Elevator weight simulation

5.2.1 HIs

Baseline

The same procedure as for the TEDS dataset was performed on the EWS dataset. The results from the baseline approach are shown in Figure 5.7. The differences from Figure 5.1 (aside from the dataset) are that a moving average is here used to smooth out the noise and two trajectories of each type are shown in the center and right plot. In the right plot in red, the HIs resulting in a shock failure are shown and in blue, the HIs resulting in an expected failure are shown. As before, PCA was used to visualize the state space.

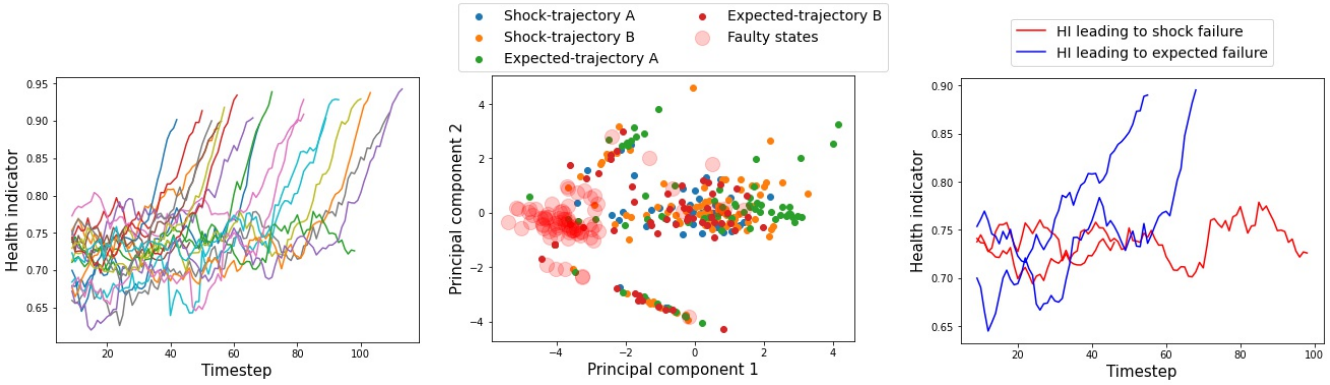


Figure 5.7 – Visualized results for the EWS dataset using the baseline, FBD-HI without transformation function. Left column: all 20 validation HIs. Middle column: state space visualized using PCA for four validation trajectories and faulty states. Right column: four validation HIs, two HIs leading to shock failure and two HIs leading to expected failure, corresponding to the ones visualized in the middle column. A moving average has been applied to every HI.

The EWS dataset had difficulties that were not present in the TEDS dataset. For one the trajectories were shorter, which means there were fewer timesteps to average out the variance. Additionally, there were missing values in the data as explained in section 4.5.2. Furthermore, there were two types of failure with different characteristics. For this reason the HIs on the left and right in Figure 5.7 appeared more "wiggly". The state space also appeared less structured. In the right plot, it can be seen that the HIs of the different types of failure differentiated. The

expected failure, while noisy, had a clear trend upwards. The shock failure on the other hand, showed no increasing trend whatsoever.

The fitness values achieved with this approach are shown in the top row of Table 5.4. Because the majority of failures were caused by the expected type, the fitness values did not tell the full story. Overall, there was clearly room for improvement in terms of handling the more rare failure.

MOGA-NN

The same procedure for the MOGA-NN was performed as for the TEDS dataset but with the EWS dataset. The size of the Pareto front was 98 solutions. In Figure 5.9 the results are visualized. Note that this time PCA was not used to visualize the states since the data was already two-dimensional. Once again a moving average was applied to every HI. Comparing the state spaces, in the middle column, with the baseline in the middle column of Figure 5.7, there are slight differences. But as the TEDS dataset, it is difficult to interpret the consequences of these differences. Instead, looking at the HIs on the right, they were with this method much more consistent. In fact it was quite difficult to distinguish shock failures from expected failures from the HIs. The histogram of the number of hidden nodes is shown in Figure 5.8. As for the TEDS dataset, the number of hidden nodes in the solutions had increased from the starting point of 16.

This time there was no subset of the fitness functions where the selected solution (found using the HP search) had the highest sum. The best subset was of trendability and Mon_1 and had the third highest sum of all solutions. This could partially have been because the HP search is stochastic and there might have been a better solution in the population. But also the weight associated with each fitness function was not necessarily uniform. It is in such situations where multi-objective optimization excels. In Table 5.4 the fitness values of a few of the MOGA-NN constructed solutions are shown. This time, even with six fitness functions, the baseline would *not* be part of the approximated Pareto front for the training data nor the validation data. In fact 56 of the found solutions were dominating the baseline in the training data, and 40 of the found solutions in the validation data. Thus, based on the fitness values and visualized HIs, the MOGA-NN solutions and added transformation function show more promise.

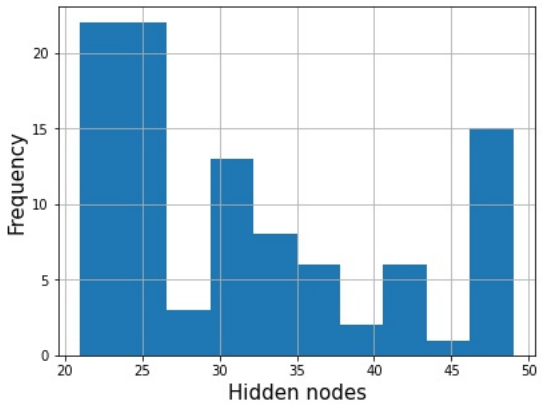
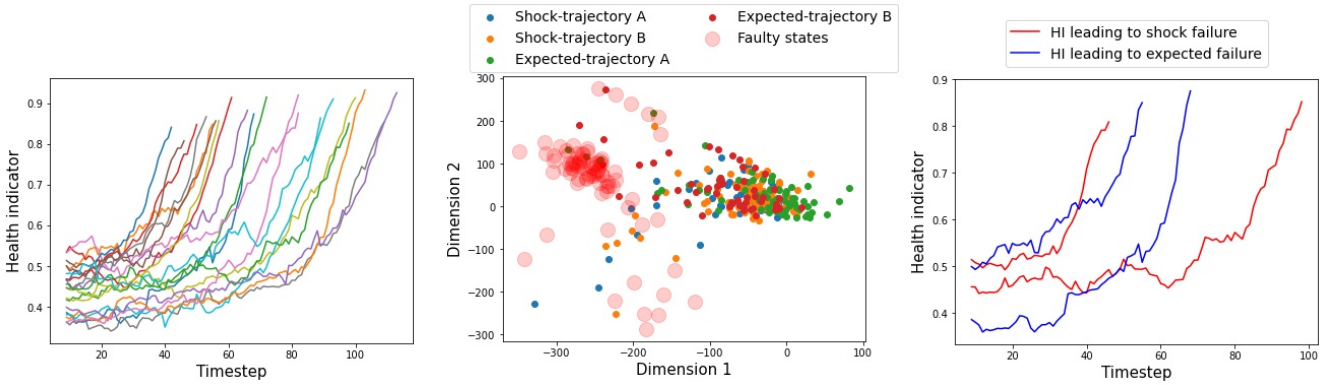


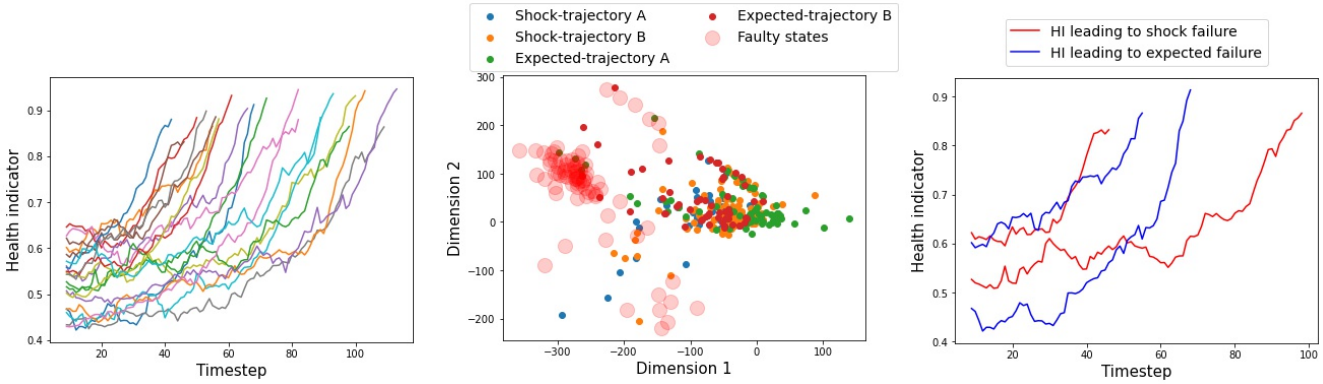
Figure 5.8 – Histogram of the number of hidden nodes in the solutions constructed using the FBD-HI MOGA-NN for the EWS dataset.

Metric / Solution	Oscillation-penalty	Trendability	Second-mon$_{10}$	Spearman-trendability	Mon$_{10}$	Mon$_1$
Baseline FBD-HI	0.091	0.57	0.036	0.54	0.23	0.049
Solutions from the MOGA-NN FBD-HI						
Best GRU	0.14	0.79	0.033	0.78	0.42	0.10
Best oscillation-penalty	0.19	0.62	0.043	0.61	0.30	0.073
Best trendability	0.13	0.80	0.038	0.79	0.43	0.078
Best Second-mon$_{10}$	0.12	0.68	0.074	0.66	0.30	0.049
Best Spearman-trendability	0.15	0.80	0.032	0.81	0.45	0.094
Best Mon$_{10}$	0.14	0.79	0.037	0.80	0.45	0.083
Best Mon$_1$	0.12	0.75	0.025	0.72	0.36	0.11

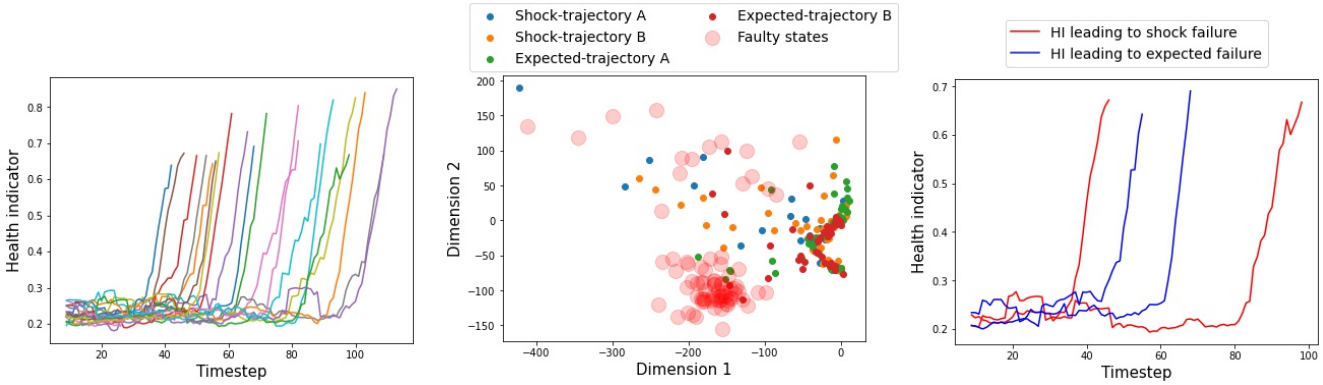
Table 5.4 – Fitness values obtained using the FBD-HI baseline and MOGA-NN applied to the EWS dataset for the training data.



(a) The best found solution with the HP search.



(b) The solution with the best achieved trendability on the training data.



(c) The solution with the best achieved oscillation-penalty on the training data.

Figure 5.9 – Visualized results for the FBD-HI MOGA-NN applied to the EWS dataset. Left column: all 20 validation HIs. Each row shows the results of one solution. Middle column: transformed state space visualized for four validation trajectories and faulty states. Right column: four validation HIs, two HIs leading to shock failure and two HIs leading to expected failure, corresponding to the ones visualized in the middle column. A moving average has been applied to every HIs.

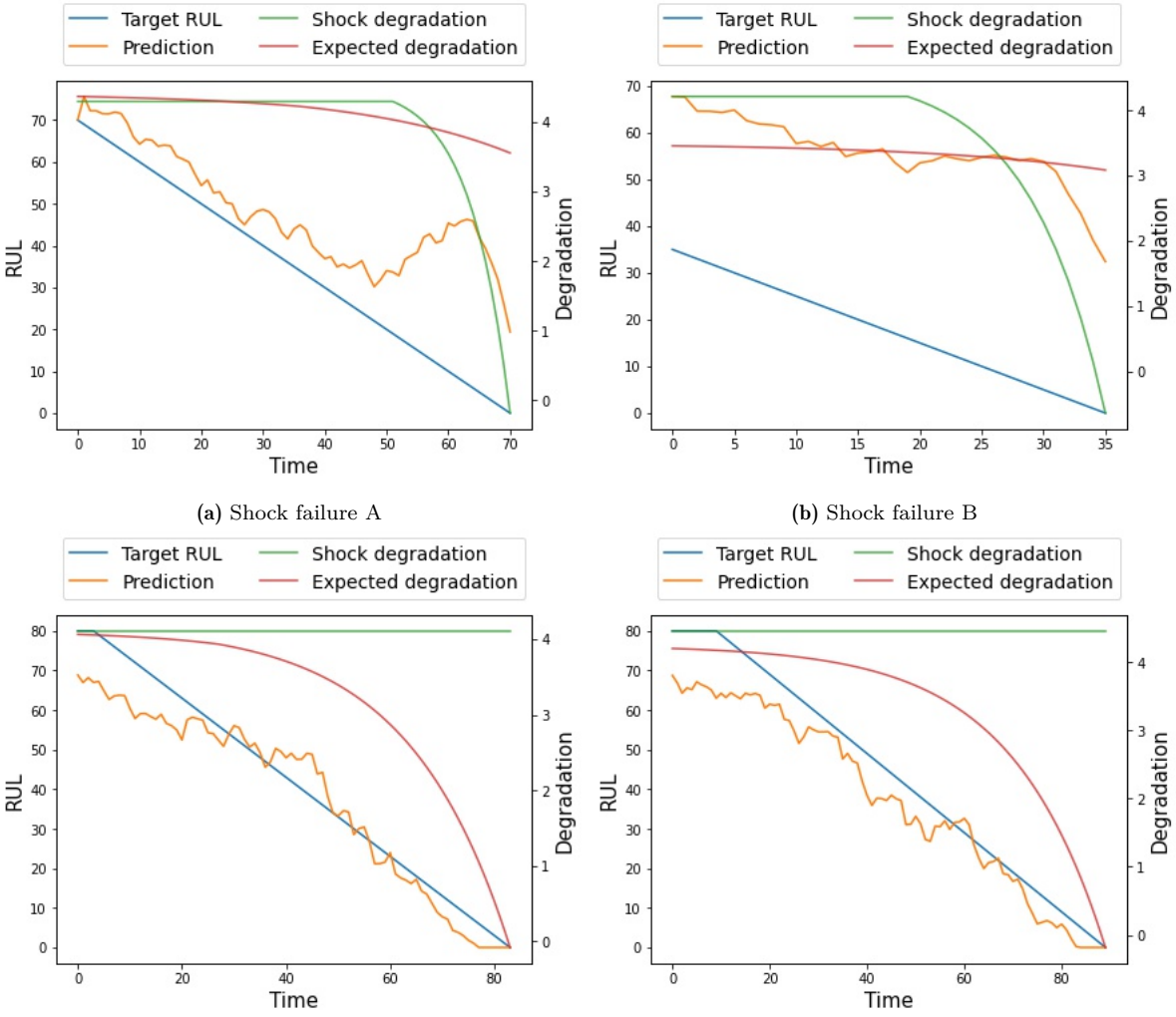
5.2.2 RUL prediction

Finally, once again an HP search was performed to find a single solution from the population and HPs for the GRU. The results generated from the chosen solution of the FBD-HI MOGA-NN are shown in the top row of Figure 5.9. The found GRU parameters using HIs from the MOGA-NN were 36 hidden nodes with a batch size of 60 and for the GRU using HIs from the baseline, 21 hidden nodes with a batch size of 43. To evaluate the results, the predictions for the GRU using the baseline HIs and the MOGA-NN generated HIs were compared for the test dataset. The baseline-HI predictions are shown in Figure 5.10 and the MOGA-NN-HI predictions are shown in Figure 5.11.

Four plots are shown, two failures caused by shock and two failures caused by expected failure. Additionally, in each graph, the actual degradation curves are displayed. The red curve is the expected degradation and an expected failure occurred when this curve went zero. The green curve is the shock degradation and the shock failure occurred when this curve went to zero.

Comparing the predictions for expected failures in the bottom row of each figure, both methods made reasonable predictions. The more visible difference was seen for the shock failure predictions in the top row of each figure. The methods were making predictions in regard to the expected degradation until the shock degradation began. Then the baseline-GRU at first, contradictingly, increased the RUL forecast. The MOGA-NN-GRU had a much more reasonable response to the progression of the shock degradation and started to decrease. Overestimation is expected before the shock degradation has begun, since then the expected failure is more likely to occur. However, even the MOGA-NN-GRU, while not as much as the baseline, overestimated the RUL after the shock degradation had started. This behaviour could have been caused by the high stochasticity, the small amount of samples (of shock failure), the quick degradation or the difficulty of modelling both errors.

The RUL prediction models were trained and tested 10 times as explained in section 4.4. The achieved mean test RMSE for the GRU using HIs from the FBD-HI MOGA-NN was 15.52 and it was lower than the GRU with HIs from the baseline FBD-HI, which was 17.14 (Wilcoxon rank-sum test, $p < 0.0002$). The results are visualized in boxplots in Figure 5.12. Thus, the evidence once again suggested that the proposed method outperformed the baseline in terms of RUL prediction. The



(a) Shock failure A

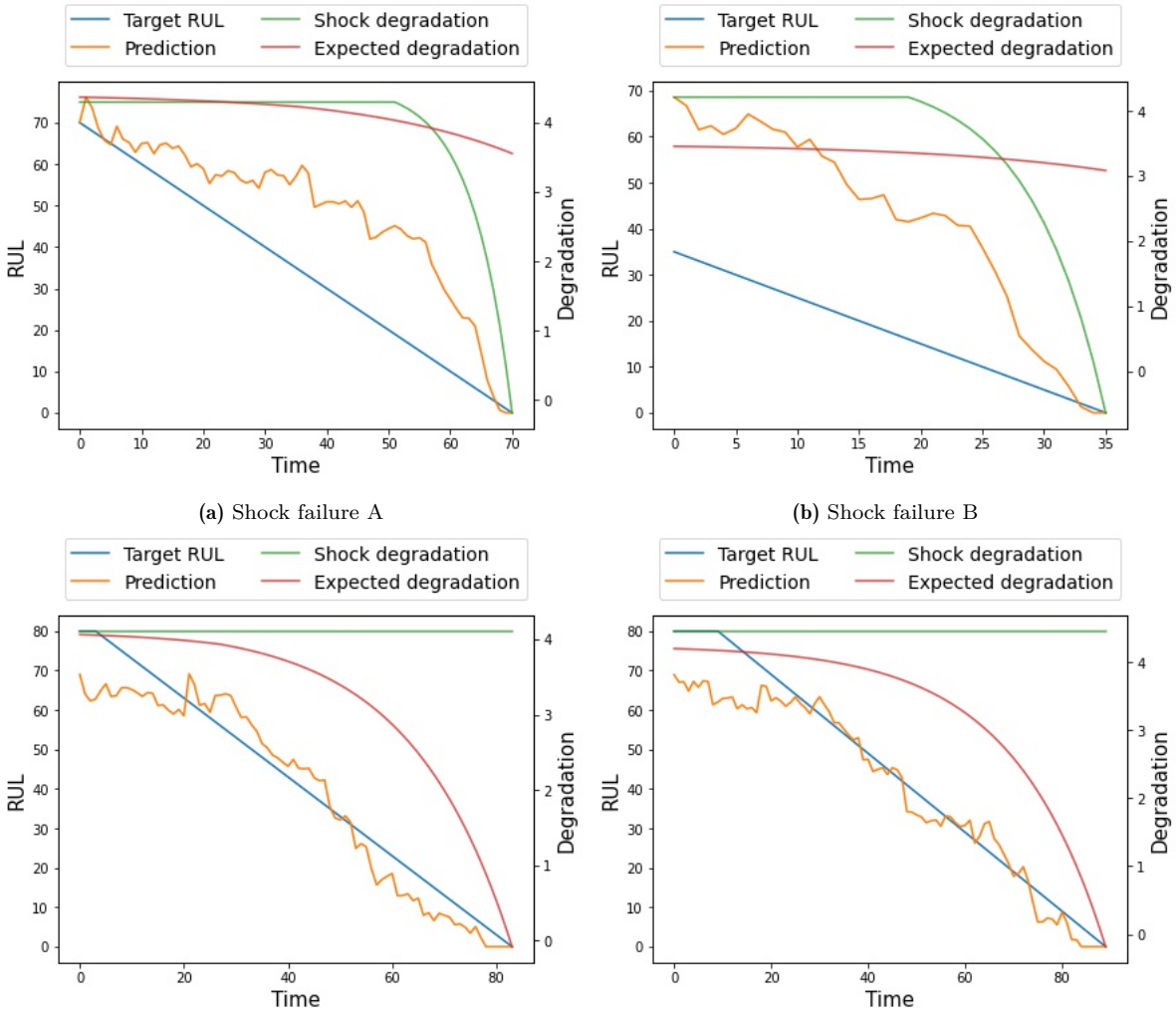
(b) Shock failure B

(c) Expected failure A

(d) Expected failure B

Figure 5.10 – Predictions using a GRU fitted with HIs from the FBD-HI without transformation function (the baseline) plotted against target RUL for four test trajectories. Additionally, in each plot the shock and expected degradation curves are shown. These are the curves that generated the data which in turn the features were extracted from. In the two plots in the top row the final failure was a shock failure and in the two plots in the bottom row the final failure was an expected failure.

number of failures of each type in each split of the data are shown in Table 5.5.



(a) Shock failure A

(b) Shock failure B

(c) Expected failure A

(d) Expected failure B

Figure 5.11 – Predictions using a GRU fitted with HIs from an FBD-HI MOGA-NN solution plotted against target RUL for four test trajectories. Additionally, in each plot the shock and expected degradation curves are shown. These are the curves that generated the data which in turn the features were extracted from. In the two plots in the top row the final failure was a shock failure and in the two plots in the bottom row the final failure was an expected failure.

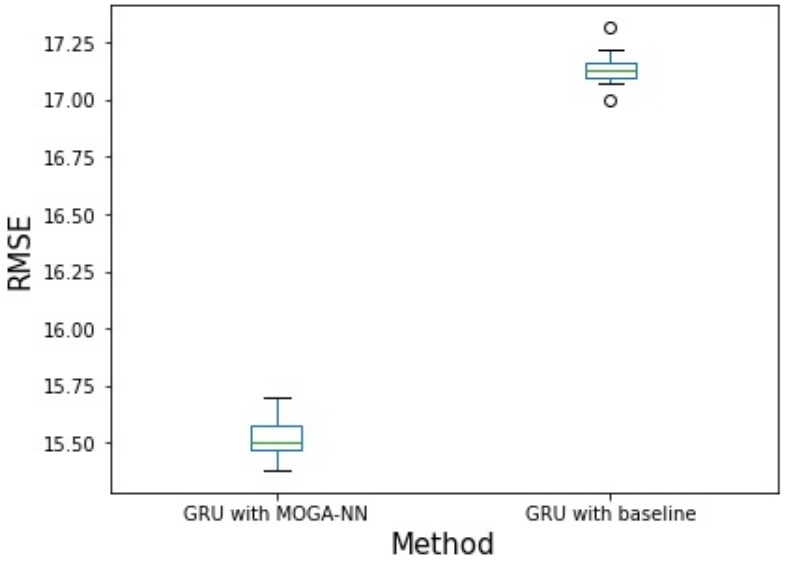


Figure 5.12 – EWS dataset test results in RMSE. Two boxplots are shown, one for the GRU using HIs from the baseline FBD-HI, and one for the GRU using HIs from the MOGA-NN FBD-HI. Each method was tested 10 times.

Set \ Failure type	Expected	Shock
Train	71	9
Validation	18	2
Test	93	7

Table 5.5 – The number of failures of each type in the different splits of the EWS dataset.

Chapter 6

Discussion

The goal with this thesis was to validate the use of a transformation function in the proposed method, compare the method with benchmarks and investigate its utility for competing risks. This was performed with two datasets, the TEDS dataset and the EWS dataset. The findings are discussed in the next section.

6.1 Summary of findings

6.1.1 Transformation function

The effect of the transformation function was analyzed by comparing the resulting RUL prediction RMSE with and without it, but also by making a qualitative judgement by visualizing the generated HIs.

In both datasets the FNN transformation function, trained using MOGA, performed significantly better in terms of the RUL prediction RMSE using the GRU than the baseline without transformation function. On the EWS dataset, from the HIs alone, the proposed method showed more promise. It visibly took into consideration both failure types (shock and expected) while the baseline struggled and also produced solutions that were dominating the baseline in terms of fitness values. The obtained solutions, especially for the TEDS dataset, were multifaceted and would likely be difficult to find without multiple-objective optimization. If it would be desired to use multiple RUL prediction models, each one can choose their most compatible HI-generator from the found set of solutions. Additionally, while not performed in this thesis, it should also be possible to incorporate domain knowledge. For instance if we knew the

degradation is exponential, a fitness function evaluating the correlation between the logarithm of the HI and time could be used. This is solely enabled by the MOGA-trained transformation function. Overall, the transformation function adds complexity and in turn flexibility, thus, enabling more difficult problems to be solved.

6.1.2 Comparison with benchmarks

The comparison with benchmarks was performed with the TEDS dataset, a popular benchmark for prognostics. Since several new methods for RUL prediction are evaluated on this dataset, it is very convenient for performance comparison. The evaluation metric was RMSE. See Table 5.3.

While a few methods reported a better result than the proposed method, one thing to notice is the simplicity of the proposed method in comparison with the other methods. For example, RBM-Bi-LSTM-Sim [4] used a two layer bidirectional LSTM, both layers with more nodes than used in the single-layer GRU in the proposed method but reported a worse performance. 1-FCLCNN-LSTM [12] had even higher complexity with three stacked LSTM layers combined with multiple CNN-layers and FNN-layers. The Dual-task LSTM [6] used a two-layer stacked LSTM and three FNN-layers, although with a small number of nodes in each. 2-S-MIC-TCNN-AM [11] used a different feature extraction method on the original features before applying a temporal convolutional network with attention mechanism. Overall, in terms of performance in relation to its simplicity, the proposed method did very well. Combining the FBD-HI MOGA-NN with other steps of the pipeline, such as a different RUL prediction model and feature extraction, would be interesting to explore in future research. Ideally, the proposed method should be compared with other methods on more datasets, but this was not performed due to lack of time.

6.1.3 Competing risks

The efficacy of the method in handling competing risks was addressed with the EWS dataset. In the simulation two independent failures could occur, which enabled the investigation of how the proposed method handles the edge case of failures with different characteristics. That is errors that are completely unrelated. Here a more qualitative analysis

was performed by visualizing the generated HIs and comparing RUL predictions to the real degradation data (which was available since the data was simulated).

By comparing the predictions to the target RUL and the degradation curves, it could be seen that the model took into account both failure types (shock and expected), although it overestimated the more rare failure. That being said, the simulated dataset was complex, exhibiting missing data, short trajectories, independent failures and the rare failure only occurring in 9 out of 80 trajectories in the training data. The way the method handles competing risks is by creating one HI for all degradations. The advantage is that there will be more training data for the RUL prediction model, since the trajectories for all failures are combined and not handled separately. For instance, in this case it is not certain that the GRU could model the more rare failure alone since it requires a lot of data. Additionally, since it generates a single HI, practically any RUL prediction model can be used. The disadvantage is that by mapping the information about all failures onto a single HI, information is lost. Thus, if the HI is unable to represent all failures adequately, the method will perform poorly. In this dataset there was some indication from the overestimation of RUL for the shock failure that the HI did not perfectly account for both failures. Overall, I believe this way of handling competing risks or failures with different characteristics will be useful when the failures have shared traits. Then the HI will be more likely to represent both failures sufficiently.

6.2 Method choices

6.2.1 MOGA-NN and MOGA-RNN

In this thesis an FNN was trained using the MOGA to generate HIs. An alternative option would be to use an RNN with the MOGA, but this alternative was not selected for a few reasons. Most importantly, the FNN can be applied to each state independently. Thus, one added performance improvement was to concatenate all states in all trajectories into a matrix and then feed this matrix into the FNN. Thereafter, the output could be split into separate HIs for each trajectory. If the RNN would have been used instead, a hidden state would have to be taken into consideration and passed between each consecutive state. The training time would likely increase by an order of magnitude. Additionally, the

GA operators would be much more complex. That being said, there are some potential advantages of using an RNN, as explained in section 6.4.

6.2.2 Backpropagation

An alternative to generating HIs separately with the MOGA-NN and then training the GRU is to train, in an end-to-end approach, the entire pipeline using backpropagation. Then the sensor data is first transformed using the FBD-HI NN to generate an HI. Thereafter, this HI is passed to the GRU for RUL prediction and finally the error is propagated backwards to update the weights. A few tests were performed with this approach and the results were similar to the results achieved using the FBD-HI MOGA-NN with GRU. In the end, the MOGA-version was chosen instead since it is not specifically designed to work with the GRU, and thus, can be combined with multiple method types.

6.2.3 Artificial points

Another idea that was investigated was to add "artificial points". This would be states that were added to the transformed space to help measure distance. They could be constructed using additional weight-vectors added to each solution, aside from the FNN-weights, that are updated using the GA-operators. Alternatively, *entity embeddings* [36] could be used with backpropagation.

Then these points could either be used as additional faulty states to measure the minimum distance to, but also using some kind of intermediate distance calculation as "checkpoints". For instance, one idea was to use the cumulative minimum distance to the points in addition to the distance to the faulty states as the HI. Thus, after the trajectories passed through these points, the cumulative minimum distance would be close to zero and only the distance to the faulty states would remain. But these additions did not cause any significant improvements over the previous method, and therefore, was abandoned. There is potentially some use of these points but further research needs to be conducted to solidify this.

6.3 Thesis evaluation

6.3.1 Ethics and sustainability

The goal of the thesis is to contribute to increased longevity and productivity for machines. Additionally, by obtaining a better understanding of when failures can happen, accidents can be avoided that could impact human lives directly. On the other hand, if the demand for machines increases due to improved effectiveness, this could contribute to an increased energy consumption and environmental consequences. This problem exists with practically all new technologies. My belief is that the solution to environmental problems should come from clean and renewable energy and not from stopping development.

In general the development of artificial intelligence and machine learning comes with many ethical challenges such as potential discriminatory bias, replacement of jobs and ill-intended use. For health prognostics, since the predictions are made in a closed system in regard to machines, no harmful discriminatory bias is involved. The human labour that is part of the process is to perform manual checks on the machines to detect the need for replacement or reparation, and the replacement/reparation itself. A successful prognostics system would reduce/eliminate the time spent on both of these tasks. The manual check could be performed by the prognostics method using data from sensors and the required reparation would be less severe since the reparation is timed before failure. The dilemma of replacing jobs is repeatedly present in technological advancements. Often the tasks replaced are tedious, like the manual checking. But in turn new jobs emerge, such as researchers creating new methods and engineers implementing the methods. Finally, the possibility of ill-intended use is practically neglectable, since it is a very specific task acting in a closed environment.

6.3.2 Strengths and weaknesses

One weakness with this work is some of the seemingly arbitrary choices made when it came to tuning and fitting the FBD-HI MOGA-NN. Because there were numerous choices of parameters for the GA, the fitness functions, the NN, the distance calculation etc. it is practically impossible to test all parameters. Additionally, the evaluation of each setting is nontrivial since multiple solutions with multiple metrics are

constructed. That being said, there is a reason why GA works well, regardless of parameters, with an NN in this type of problem specifically. Since only relative distances between the states and the faulty states are used in the creation of the HIs, multiple settings can lead to similar results. Furthermore, solutions are easy to find because of the FBD-HI approach. For instance, the baseline, which simply used the original features, generated an HI with an approximately monotonic shape on the TEDS dataset without transformation. Consequently, the method is robust to different parameters.

One of the strengths, in my opinion, is how the results were presented. The necessary information that quantifies the performance can be described concisely in the RUL prediction results. But this was accompanied by several figures describing the different HIs and the state spaces that were generated. While not quantitative evidence, this provides intuition into what the method does and potential strengths and weaknesses. Additionally, in the second dataset where the real degradations were available, a qualitative comparison could be made by visualizing the predictions for failures of different types in relation to the degradations.

Arguably, more statistical evidence could have been provided for the resulting fitness values, but this data is also inherently inexact. The evaluation metrics are only approximate measures used to quickly assess the HIs. These are thus useful to get a rough estimate of an HI-generator without selecting an RUL prediction model. But since the actual RUL predictions using the HIs are what defines the aptness of the HIs this is what was quantified statistically.

6.3.3 Contribution

The contribution of this thesis is first and foremost the method or framework, FBD-HI MOGA-NN for HI construction. This solves problems that many methods often possess, such as the inability to steer HIs to exhibit specific characteristics, the HIs failing at widely different values or not being able to handle nonlinear data. The method was compared with the state of the art. Furthermore, generating HIs by measuring distance with and without transformation function was compared, providing evidence that using such a function has a significant impact. Additionally, a dataset was generated to test the method in a difficult situation, thus, exploring a scenario rarely tested. In

an industrial setting where domain knowledge is available, adding this knowledge in the form of a fitness function could be beneficial to generate useful HIs.

A distance categorization was also made, FBD-HI and HBD-HI. In general, I believe it would be useful to partition methods into groups in this field to more easily be able to grasp differences, advantages and disadvantages between approaches.

6.4 Future research

In this thesis a new flexible framework for HI construction was proposed with the ability to construct a wide variety of HIs with competitive RUL predictions. Overall, I believe there are a lot of promising future research possibilities and opportunities to extend the method further. For example, it would interesting to explore more fitness functions. The shape of the chosen solution for the TEDS dataset using the HP search was quite unexpected. This lead me to believe that there are room to improve the fitness functions to promote such solutions.

Another possibility is to investigate the use of multiple HI-generators from the MOGA, instead of just one, to potentially improve performance. In that case the method would be more suitable for AMs, since many SMMs presume a single HI. Since many of the solutions are highly correlated, using all of them would likely not work. Thus, a selection procedure, similar to the GA used in this thesis, would be needed.

Another approach would be to change the structure of the transformation function entirely by for instance swapping the FNN for an RNN. As previously mentioned, this causes the training time to increase significantly. That being said, an hypothesis is that this could improve the performance in situations where the states are highly stochastic and at each state it is difficult to estimate the health. With the current method one could use lagged states to mitigate this problem, but RNNs are in general more flexible.

Chapter 7

Conclusion

In this thesis an MOGA-NN HI construction method based on distance was introduced and used together with a GRU to predict RUL. The MOGA-NN served as a transformation function in the FBD-HI framework to optimize fitness functions desireable in HIs. It was tested on two datasets to analyze its performance and behaviour.

On the TEDS dataset it was able to create a variety of HI-generators and achieved a strong performance in terms of RUL prediction given the small size of the NNs, but did not beat the results reported by a few methods of the state of the art. On the EWS dataset the method was able to construct a single HI with a desirable shape for two independent failures, although the RUL predictions overestimated the more rare failure to a certain degree.

Comparing the method with a baseline FBD-HI without transformation function, it was shown to outperform the baseline on both datasets in terms of RUL prediction. Additionally, on the EWS dataset in the visualization of the baseline without transformation function, there were signs of problems handling the two failures that were not seen in the visualization of the proposed method. Because the method is an FBD-HI, it also has the desirable property of a natural failure threshold which is useful for many RUL prediction models.

References

[1] L. Liao, "Discovering prognostic features using genetic programming in remaining useful life prediction", *IEEE Transactions on Industrial Electronics*, vol. 61, no. 5, pp. 2464–2472, 2013.

[2] J. Sun, H. Zuo, W. Wang, and M. G. Pecht, "Application of a state space modeling technique to system prognostics based on a health index for condition-based maintenance", *Mechanical systems and signal processing*, vol. 28, pp. 585–596, 2012.

[3] K. Peng, R. Jiao, J. Dong, and Y. Pi, "A deep belief network based health indicator construction and remaining useful life prediction using improved particle filter", *Neurocomputing*, vol. 361, pp. 19–28, 2019, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.07.075. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231219310823.

[4] M. Hou, D. Pi, and B. Li, "Similarity-based deep learning approach for remaining useful life prediction", *Measurement*, vol. 159, p. 107 788, 2020, ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2020.107788. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263224120303262.

[5] A. L. Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, and H. Zhang, "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture", *Reliability Engineering & System Safety*, vol. 183, pp. 240–251, 2019.

[6] H. Miao, B. Li, C. Sun, and J. Liu, "Joint learning of degradation assessment and rul prediction for aeroengines via dual-task deep lstm networks", *IEEE Transactions on Industrial Informatics*, vol. 15, no. 9, pp. 5023–5032, 2019.

[7] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long short-term memory network for remaining useful life estimation", in *2017 IEEE international conference on prognostics and health management (ICPHM)*, IEEE, 2017, pp. 88–95.

[8] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks", *Reliability Engineering System Safety*, vol. 172, pp. 1–11, 2018, ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2017.11.021. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832017307779.

[9] G. S. Babu, P. Zhao, and X.-L. Li, "Deep convolutional neural network based regression approach for estimation of remaining useful life", in *International conference on database systems for advanced applications*, Springer, 2016, pp. 214–228.

[10] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[11] Y. Jiang, C. Li, Z. Yang, Y. Zhao, and X. Wang, "Remaining useful life estimation combining two-step maximal information coefficient and temporal convolutional network with attention mechanism", *IEEE Access*, vol. 9, pp. 16 323–16 336, 2021.

[12] C. Peng, Y. Chen, Q. Chen, Z. Tang, L. Li, and W. Gui, "A remaining useful life prognosis of turbofan engine using temporal and spatial feature fusion", *Sensors*, vol. 21, no. 2, p. 418, 2021.

[13] K. L. Tsui, N. Chen, Q. Zhou, Y. Hai, and W. Wang, "Prognostics and health management: a review on data driven approaches", *Mathematical Problems in Engineering*, vol. 2015, 2015.

[14] Y. Lei, N. Li, L. Guo, N. Li, T. Yan, and J. Lin, "Machinery health prognostics: a systematic review from data acquisition to rul prediction", *Mechanical Systems and Signal Processing*, vol. 104, pp. 799–834, 2018.

[15] C. Murphy. (ca. 2020). Choosing the most suitable predictive maintenance sensor, [Online]. Available: https://www.analog.com/en/technical-articles/choosing-the-most-suitable-predictive-maintenance-sensor.html (visited on 03/17/2020).

[16] P. Wang, Y. Li, and C. K. Reddy, "Machine learning for survival analysis: a survey", *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[17] M. Wang, Y. Li, H. Zhao, and Y. Zhang, "Combining autoencoder with similarity measurement for aircraft engine remaining useful life estimation", in *Proceedings of the International Conference on Aerospace System Science and Engineering 2019*, Z. Jing, Ed., Singapore: Springer Singapore, 2020, pp. 197–208, ISBN: 978-981-15-1773-0.

[18] P. Baraldi, G. Bonfanti, and E. Zio, "Differential evolution-based multi-objective optimization for the definition of a health indicator for fault diagnostics and prognostics", *Mechanical Systems and Signal Processing*, vol. 102, pp. 382–400, 2018.

[19] A. Cubillo, S. Perinpanayagam, and M. Esperon-Miguez, "A review of physics-based models in prognostics: application to gears and bearings of rotating machinery", *Advances in Mechanical Engineering*, vol. 8, no. 8, p. 1 687 814 016 664 660, 2016.

[20] R. E. Kalman, "A new approach to linear filtering and prediction problems", 1960.

[21] F. Gustafsson, "Particle filter theory and practice with positioning applications", *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 7, pp. 53–82, 2010.

[22] R. Rojas, "The backpropagation algorithm", in *Neural networks*, Springer, 1996, pp. 149–182.

[23] S. Mirjalili, "Genetic algorithm", in *Evolutionary algorithms and neural networks*, Springer, 2019, pp. 43–55.

[24] N. Saini, "Review of selection methods in genetic algorithms", *International Journal of Engineering and Computer Science*, vol. 6, no. 12, pp. 22 261–22 263, 2017.

[25] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: a tutorial", *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: nsga-ii", *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[27] N. Garcıa-Pedrajas, D. Ortiz-Boyer, and C. Hervás-Martınez, "An alternative approach for neural network evolution with a genetic algorithm: crossover by combinatorial optimization", *Neural Networks*, vol. 19, no. 4, pp. 514–528, 2006.

[28] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[29] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation", *arXiv preprint arXiv:1406.1078*, 2014.

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

[31] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling", *arXiv preprint arXiv:1412.3555*, 2014.

[32] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[33] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation", in *2008 international conference on prognostics and health management*, IEEE, 2008, pp. 1–9.

[34] K. Aainsqatsi. (2008). Turbofan operation. Wikimedia Commons, [Online]. Available: https://commons.wikimedia.org/wiki/File:Turbofan_operation.svg. License: CC BY-SA 3.0.

[35] Q. Wang, S. Zheng, A. Farahat, S. Serita, and C. Gupta, "Remaining useful life estimation using functional data analysis", in *2019 ieee international conference on prognostics and health management (icphm)*, IEEE, 2019, pp. 1–8.

[36] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables", *arXiv preprint arXiv:1604.06737*, 2016.

[37] C. R. Harris *et al.*, "Array programming with NumPy", *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

Appendix A

Generating weight simulation data

The code is in Python using the package NumPy [37]:

```
import numpy as np

def add_noise(path):
    return np.random.normal(0, 3, size=path.shape[0])

def gen_pre(mean):
    path = np.ones(np.random.randint(20, 120))*mean*2
    # add noise
    path += add_noise(path)
    # add spike
    end = np.random.randint(2, 5)
    last = path.shape[0]-10-end
    spike = np.random.randint(5, max(last-5, 6))
    path[spike:spike+2] += np.random.randint(150, 300,
    ↪  size=2)
    # increase
    increase_point = np.random.randint(max(min(spike-10,
    ↪  last-1), 0), last)
    mx = np.random.normal(666, 100)
    path[increase_point:-end] +=
    ↪  np.linspace(np.random.randint(20, 30), mx,
    ↪  num=(path.shape[0]-end)-increase_point)
    path[-end:] += mx
    return path
```

```
def gen_mid(mean, shock):
    path = np.zeros(np.random.randint(500, 1500))
    # add noise
    path = np.random.normal(mean, 20, size=path.shape[0])
    # add shocks
    shocks = (np.random.uniform(size=path.shape[0]) <
    ↪  shock)*(np.random.normal(-100, 10))
    path += shocks*((np.arange(shocks.shape[0]) % 7) == 0)
    return path

def gen_end(start, mean):
    # decrease
    path = np.zeros(np.random.randint(5, 100))
    down = min(np.random.randint(10, 20), path.shape[0])
    path[:down] = np.linspace(start, mean +
    ↪  np.random.randint(20, 30), num=down)
    # add noise
    path[down:] = mean + add_noise(path[down:])
    return path

# Main function to be called
def gen(mean, shock_prob):
    pre = gen_pre(mean)
    mid = gen_mid(pre[-1], shock_prob)
    end = gen_end(mid[-1], mean)
    return np.concatenate([pre, mid, end])
```

For DIVA


```
{
"Author1": {
            "Last name": "Nyman",
            "First name": "Jacob",
            "E-mail": "jacobny@kth.se",
            "organisation": {"L1": "School of Electrical Engineering and Computer Science ",
                             }
          },
"Degree": {"Educational program": "Master's Programme, Machine Learning, 120 credits"},
"Title": {
            "Main title": "Machinery Health Indicator Construction using Multi-objective Genetic Algorithm Optimization of a Feed-forward
Neural Network based on Distance",
            "Subtitle": "Master Thesis in Machine Learning",
            "Language": "eng" },
"Alternative title": {
            "Main title": "Maskin-Hälsoindikatorkonstruktion genom Multi-objektiv Genetisk Algoritm-Optimering av ett Feed-forward
Neuralt Nätverk baserat på Avstånd",
            "Subtitle": "Examensarbete i Maskininlärning",
            "Language": "swe"
},
"Supervisor1": {
                  "Last name": "Kumar",
                  "First name": "Arvind",
                  "E-mail": "arvkumar@kth.se",
                  "organisation": {"L1": "School of Electrical Engineering and Computer Science ",
}
               },
"Examiner1": {
                  "Last name": "Herman",
                  "First name": "Pawel",
                  "E-mail": "paherman@kth.se",
                  "organisation": {"L1": "School of Electrical Engineering and Computer Science ",
}
               },
"Cooperation": { "Partner_name": "CNet Svenska AB"},
"Other information": {
"Year": "2021", "Number of pages": "ix,82"}
}
```

TRITA -EECS-EX-2021:221