A self-organizing map and a normalizing multi-layer perceptron approach to baselining in prognostics under dynamic regimes

Baptista, Marcia Lourenco; Elsa, Elsa M.; Goebel, Kai

DOI
10.1016/j.neucom.2021.05.031

Publication date
2021

Document Version
Final published version

Published in
Neurocomputing

Citation (APA)
Baptista, M. L., Elsa, E. M., & Goebel, K. (2021). A self-organizing map and a normalizing multi-layer perceptron approach to baselining in prognostics under dynamic regimes. *Neurocomputing*, *456*, 268-287. https://doi.org/10.1016/j.neucom.2021.05.031

Neurocomputing 456 (2021) 268–287

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A self-organizing map and a normalizing multi-layer perceptron approach to baselining in prognostics under dynamic regimes

Marcia Lourenco Baptista [a,*], Elsa M. P. Henriques [b], Kai Goebel [c,d]

[a] Air Transport and Operations, Faculty of Aerospace Engineering, Delft University of Technology, Delft, Netherlands
[b] LAETA, IDMEC, Instituto Superior Tecnico, University of Lisbon, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
[c] Division of Operation and Maintenance Engineering, Luleå Technical University, Luleå, Sweden
[d] Palo Alto Research Center, Palo Alto, CA 94304, United States

ARTICLE INFO

Article history:
Received 23 February 2020
Revised 29 March 2021
Accepted 11 May 2021
Available online 18 May 2021
Communicated by Zidong Wang

Keywords:
Self-organizing map
Normalizing multi-layer perceptron
Prognostics
Baselining
Turbofan sensor data

ABSTRACT

When the influence of changing operational and environmental conditions, such as temperature and external loading, is not factored out from sensor data it can be difficult to observe a clear deterioration path. This can significantly affect the task of engineering prognostics and other health management operations. To address this problem of dynamic operating regimes, it is necessary to *baseline* the data, typically by first finding the operating regimes and then normalizing the data within each regime. This paper describes a baselining solution based on neural networks. A self-organizing map is used to identify the regimes, and a multi-layer perceptron is used to normalize the sensor data according to the detected regimes. Tests are performed on public datasets from a turbofan simulator. The approach can produce similar results to classical methods without the need to specify in advance the number of regimes and the explicit computation of the statistical properties of a hold-out dataset. Importantly, the techniques can be integrated into a deep learning system to perform prognostics in a single pass.

1. Introduction

Most engineering systems go through different operating and environmental conditions during their life cycle. For example, speed [1], or load conditions [2] may change continuously during the operation of an engineering system. A system's full range of operating and environmental conditions is usually grouped into a finite set of operating regimes. An operating regime (or mode) is a subset of operating points where the system behaves similarly. For example, in a commercial airplane flight, a general classification of the flight regimes are take-off, climb, cruise, descent, and landing [3]. Importantly, changes in the operating regimes can significantly affect sensor readings obscuring the systems' degradation signature [4]. When the influence of these variations is not reduced or eliminated, one may not easily deduce the underlying damage trajectories from the different response signals.

The RUL is a random variable of interest in prognostics that describes the remaining time to failure of equipment at a given point in time. The task of RUL estimation can benefit from having a baselining step to minimize the impact of the operational and environmental conditions on data quality. Baselining is a classical pre-processing step of RUL estimation. By *baselining* the sensor data, what is meant is the factoring out of the influence of dynamic operating regimes on sensor measurements. Note that the goal of baselining is to reveal the progression of the underlying fault mechanisms –, i.e., the physical processes which cause failure. Since the exercise of RUL estimation, or in other words, of prognostics, greatly depends on being able to observe the failure mechanisms in the form of P-F curves [1] [5], baselining is of paramount importance. Disregarding this step altogether can promote poor prognostics, and in the case of purely data-driven prognostics, it may even lead to meaningless results.

Baselining can be broken down into two stages shown in Fig. 1: first, identifying to which operating regime each data point belongs; and second, fusing the data points of the different regimes. The first task is usually done by decomposing the regimes using a technique such as K-means clustering [6]. For the second task, a statistical rule is often employed to normalize the data within each regime. Data normalization (also known as feature scaling) means the standardization of the range of the data fea-

* Corresponding author.
E-mail addresses: m.lbaptista@tudelft.nl (M.L. Baptista), elsa.henriques@flad.pt, elsa.h@tecnico.ulisboa.pt (E.M. P. Henriques), kai.goebel@ltu.se (K. Goebel).

[1] A P-F curve is a common graph used to represent the degradation of equipment between Potential failure and Functional failure. The P stands for Potential failure and the F for Functional Failure.

https://doi.org/10.1016/j.neucom.2021.05.031
0925-2312/

Nomenclature

Symbols

U	Number of equipment units.
S	Number of sensor features.
R	Number of operating regimes.
u	Index of unit such that $u \in \{1, \ldots, U\}$.
s	Index of sensor feature such that $s \in \{1, \ldots, S\}$.
r	Index of operating regime such that $r \in \{1, \ldots, R\}$.
T_u	End of life of equipment u.
t	Time index such that $t \in \{1, \ldots, T_u\}$.
$\vec{x}^{us}$	Sensor feature s for unit u.
$\vec{m}_t^u$	Original collection of sensor measurements for unit u at time t.
$\vec{m}_t^{us}$	Original sensor measurement of feature s for unit u at time t.
Reg_r	Operating regime.
$f_a(.)$	First baselining function that assigns an operating regime to each sensor measurement $f_b : \mathbb{R} \rightarrow \{Reg_r\}_{r=1}^{r=R}$.
$f_b(.)$	Second baselining function that transforms the sensor measurements reducing the influence of the operating regimes $f_b : \mathbb{R} \rightarrow \mathbb{R}$.
$\vec{y}^{us}$	Baselined sensor feature s for unit u.
y_t^{us}	Baselined sensor measurement of feature s for unit u at time t.
C_r	Collection (cluster) of data points x_t^{us} classified as belonging to a certain operating regime r.
G	Number of control and environmental variables.
P	Number of input neurons in the self-organizing map.
g	Index of control/environmental variable such that $g \in \{1, \ldots, G\}$.
N_{SOM}	Total number of computational neurons of self-organizing map.
$size_x$	Number of neurons in the x-axis of self-organizing map.
$size_y$	Number of neurons in the y-axis of self-organizing map.
j	Index of computational neuron of self-organizing map.
i	Index of input neuron of self-organizing map.
$\vec{p}$	Input of the self-organizing map.
$\vec{z}$	Computational neuron of the self-organizing map.
$\vec{w}_j$	Weight vector associated to each computational neuron of self-organizing map.
n	Number of input vectors z of self-organizing map.
α	Learning rate of self-organizing map.
$h_{cj}(t)$	Neighborhood function of self-organizing map.
c	Best matching unit of self-organizing map.
σ_{SOM}	Neighborhood width of self-organizing map.
v	Coordinate of neuron in self-organizing map.
I_{SOM}	Number of iterations (epochs) of batch self-organizing map.
$\mathbb{X}$	The set of training examples of multi-layer perceptron.
N_{MLP}	The size of $\mathbb{X}$ (multi-layer perceptron).
I_{MLP}	Number of iterations (epochs) of multi-layer perceptron.
$(\vec{x}^{(n)}, \vec{y}^{(n)})$	The n-th example pair in $\mathbb{X}$ of multi-layer perceptron (supervised learning).
$\vec{x}$	The array of examples of multi-layer perceptron.
$\vec{x}^{(n)}$	The n-th example of multi-layer perceptron.
d	The dimension of a data point $\vec{x}^{(n)}$ of multi-layer perceptron.
D	The number of dimensions of a data point $\vec{x}^{(n)}$ of multi-layer perceptron.
$\vec{y}$	The array of labels of multi-layer perceptron (supervised learning).
$y^{(n)}$	The n-th label of multi-layer perceptron (supervised learning).
$\widehat{\vec{y}}$	The array of predicted labels of multi-layer perceptron.
$\widehat{y^{(n)}}$	The n-th predicted label of multi-layer perceptron.
W^{MLP}	The weight matrix of multi-layer perceptron.
$w_d^{(n)}$	The weight corresponding to the d-dimension of the n-th input of the multi-layer perceptron.
$x_d^{(n)}$	The input corresponding to the d-dimension of the n-th input of the multi-layer perceptron.
E_W	The error function of multi-layer perceptron.
φ	Activation function (multi-layer perceptron).
bias	Bias (multi-layer perceptron).
η	Learning rate (multi-layer perceptron).
net	Weighted sum of inputs (multi-layer perceptron).
γ	Best parameters of the function that the neural network tries to approximate (multi-layer perceptron).
K	Mini-batch size (multi-layer perceptron).
$v_{\min}$	Minimum value of uniform distribution used to initialize multi-layer perceptron.
$v_{\max}$	Maximum value of uniform distribution used to initialize multi-layer perceptron.
μ^r	Mean of the data of one regime.
σ^r	Standard deviation of the data of one regime.

Acronyms

C-MAPSS	Commercial Modular Aero-Propulsion System Simulation
RUL	Remaining Useful Life
SOM	Self-Organizing Map
NASA	National Aeronautics and Space Administration
MLP	Multi-Layer Perceptron
PCA	Principal Component Analysis
PHM	Prognostics and Health Management
SVM	Support Vector Machines
ARX	Autoregressive with Extra Input
SHM	Structural Health Monitoring
FFNN	Feed Forward Neural Network
BMU	Best Matching Unit
TRA	Throttle Resolver Angle
MAE	Mean Absolute Error

tures. Different formulas can be applied to perform the normalization. Regardless of the procedure, the final result should be a collection of sensor features where the operating conditions' influence is factored out, and degradation trends are more readily observable.

When the number of operating regimes is known beforehand, assigning each data point to a regime is a more straightforward task. However, in many applications, the number of operating regimes is unknown. In such cases, alternative ways need to be utilized to discover how many operating modes exist in the data. This paper proposes a modified version of the SOM that does not require setting the number of regimes. The goal of the proposed SOM is to discover and discern among the different operating regimes given a set of control and ambient variables. Based on these variables, the SOM learns the regimes and assigns the data points accordingly.

After associating each data point to a regime, it is possible to perform data normalization and factor out the influence of the

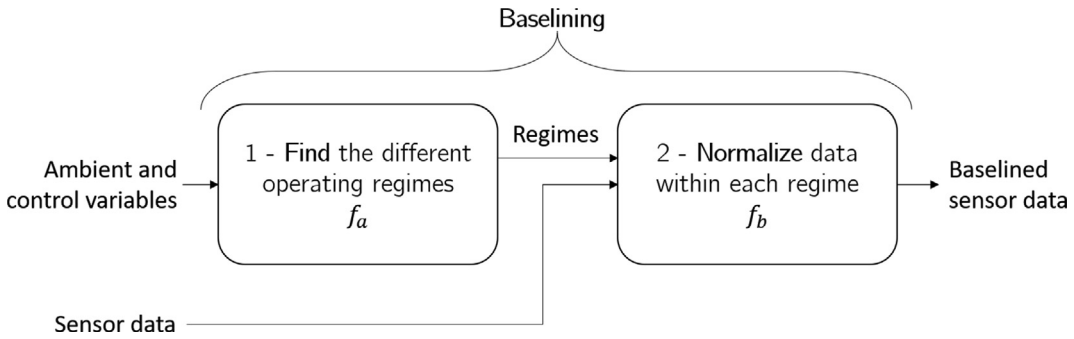


Fig. 1. The baselining of value data seeks to remove the influence of the different operating conditions on the data. Typical flow consists of first identifying regimes and then normalizing the data of different regimes to a common scale.

non-stationary operational and environmental conditions. In classical normalization approaches, such as standardization, it is necessary to have a hold-out dataset to compute each regime's mean and standard deviation before applying the z-score formula. In applications where the availability of data is scarce, ruling out data on the grounds of mean and standard deviation computation can be a significant problem. This difficulty is even more critical for data-driven prognostics in fields such as aeronautics, where the availability of large volumes of run-to-failure data can be an issue. The normalizing MLP proposed in this paper addresses this difficulty. This paper shows that the MLP can be trained on a single unit's degradation data and still perform reasonably well. Based on the SOM regimes, the MLP normalizes the data within each regime. The goal of the MLP is to preserve each regime's degradation trends while normalizing the data.

In technical terms, the contribution of the proposed approach is twofold:

- The original SOM architecture is subject to adaptations to serve the purpose of regime clustering. Of note is how the SOM is combined with a technique of connected component labeling to group the neurons and discover the clusters corresponding to the operating regimes. A mechanism is also proposed to ensure the correctness of the overall clustering method.
- The MLP, due to its unique error function and structure, allows normalizing unscaled data and the network functions without supervision.

This paper's contribution is also to show how neural networks can be used to address the baselining problem on an engineering application. The proposed approach is evaluated on two publicly available datasets generated from CMAPSS. CMAPSS is a turbofan simulation model developed by the NASA. The two datasets consist of thousands of run-to-failure trajectories. The goal of these data is to allow researchers to build, evaluate and benchmark different approaches to PHM problems.

The remainder of this paper is as follows. Section 2 reviews related work. A mathematical formulation of the problem is presented in Section 3. Section 4 describes the proposed approach. The case study is presented in Section 5. Results are shown in Section 6. Section 7 concludes the paper.

2. Related work

Implementing a PHM program involves developing methods, protocols, or infrastructure to sense, diagnose, and prognosticate health state changes of engineering equipment [7]. One of the goals of PHM is to detect at the earliest onset of failure. Here, failure can be defined [8] as the point when degradation reaches a predetermined threshold level. The process of degradation or damage of an engineering system can have multiple meanings [9]. In civil engineering [10], for example, changes to the materials or geometric properties of physical infrastructures such as bridges, towers, tunnels, dams are typically subject to analysis. In electronics, battery discharge cycles are examined [11]. In mechanical engineering, performance or operating limits are considered [9]. In general, a system is considered to have failed when it no longer meets its predefined functional requirements such as usage, operating limits, or other specifications [12].

It is easier to detect and predict failure if the deterioration process is naturally correlated with condition monitoring data [13]. However, in practical applications, operational and ambient conditions can produce changes in sensor signals and mask the damage trajectories. In such cases, the operating differences can influence the extent and the rate of change of the degradation, obscuring the fault signatures. Further aggravating the problem, the more the sensor signals are responsive to deterioration, the more sensitive they are to operational and environmental conditions [9]. Baselining the data to improve its quality and how well the data represent the degradation process is an essential step for PHM [14]. Some of the most important contributions to the field are reviewed hereafter.

Sohn et al. [15] were among the first to study the influence of ambient conditions on the non-stationary responses of large-scale structures. The authors ascertained that effective damage detection of large infrastructure should consider the variability of modal parameters. A linear four-input filter model was proposed in Ref. [15] to distinguish between structural damage and temperature changes. The model was applied with success to a case study of the Alamosa Canyon bridge in New Mexico. However, the generality of the results was limited, as the model was developed for a particular bridge considering only one external variable. Also, the model was static in that it did not take into account thermal dynamics.

Peeters and De Roeck [16] proposed an ARX model to capture thermal dynamics and perform damage detection in bridges. ARX models can represent the dynamics found in data as they relate current output to past input and output [17]. In their work, Peeters and De Roeck [16] used "healthy" data from the Z24-Bridge in Switzerland to train an ARX and simulate eigenfrequencies. When a measured eigenfrequency fell outside the estimated confidence intervals, the model detected damage. Despite the advantages of ARX models, these approaches assume a linear relationship between the output and input, which may not always be the case. Peeters and De Roeck [16] found a bi-linear relation between frequency and temperature. The differences in asphalt stiffness in colder periods induced nonlinearity. To remove this variable's

impact, Peeters and De Roeck [16] considered only data from periods where the asphalt did not influence the relationship between the bridge's natural frequencies and temperature.

Another contribution is the work of Moser and Moaveni [18] for bridge structures. The authors [18] modeled the relationship between the natural frequencies and temperature of the Dowling Hall Footbridge on Tufts University using different models, namely linear, bi-linear, quadratic, third-order, fourth-order polynomials, and an ARX model. The fourth-order regression model was able to achieve the best fit. Another case study by Moaveni and Behmanesh [19] on the Dowling Hall Footbridge also used a fourth-order regression model with positive results.

Other approaches have been proposed to eliminate the effects of environmental variables (e.g., temperature) while performing damage detection in large-scale structures. For example, Yan et al. [20] used a method based on PCA to perform damage detection on computer-simulated data and experimental data. The technique was extended in Yan et al. [21] for non-linear cases. Deraemaeker et al. [22] used modal filters to dissociate structural damage from environmental effects. The methodology was applied to a simulated numerical model of a bridge. Other authors, such as Ni et al. [23], have proposed the use of SVM to detect damage in bridges under variable operational conditions. Four machine learning algorithms, namely, an auto-associative neural network, factor analysis, Mahalanobis distance, and singular value decomposition, were compared by Figueiredo et al. [9] in their ability to deal with changing operational and environmental conditions. For a review of models concerning changes in civil structures' vibration properties due to thermal effects, please refer to Ref. [24].

Based on different assumptions and degrees of sophistication and complexity, several approaches have been proposed over the last decades to separate changes caused by structural damage from changes caused by operational and environmental conditions. The ultimate goal here is typically that of structural damage diagnostics, **and not** prognostics. Damage prognostics aims not solely to detect the current damage state but also to forecast system performance through health state assessment [25]. In both disciplines, damage diagnostics or damage prognostics, data baselining is a critical consideration.

If it is true that research on data baselining in SHM is extensive, the same is not so for other engineering fields. Specifically, in system prognostics, and as some authors such as Gebraeel et al. [26], and Heng et al. [14], and Peng et al. [27] note, the implications posed by having different operating regimes continue to be seen as a peripheral concept. This overlook often hinders the application of prognostics to real-world scenarios. Most prognostics approaches are not suited for multi-regime situations and do not adequately respond to the problem. Some works address regime-independent prognostics. Some of these contributions are reviewed below.

Gebraeel and Pan [26] proposed a prognostics stochastic framework for computing the residual life of systems and components operating under changing operational and environmental conditions. The authors used degradation signals and information related to the environmental conditions to estimate residual life distributions in real-time. The performance of the proposed framework was evaluated on a real-world case study of rotating machinery.

Wang [28] adopted K-means as the technique for clustering data of different operating regimes. Besides K-means, Wang suggested other clustering algorithms such as Gaussian mixture models [29], and fuzzy c-means [30] to the same effect. In all the previous works of Wang, and after clustering the regimes, the sample mean and standard deviation of each regime were computed and used to normalize each sensor individually using the standard rule [31]. Importantly, the author noted that this technique could only correctly preserve the original degradation patterns if each operating condition had an identical probability to occur at each cycle. The fourth CMAPSS training dataset [32] was used in these experiments.

Al-Dahidi et al. [33] applied an unsupervised ensemble clustering approach, previously proposed in Ref. [34], to identify the distinct degradation states of a homogeneous discrete-time finite-state semi-Markov model for RUL estimation. These states described the equipment behavior according to the different operating conditions. To normalize the data [33] followed the same approach of Wang [28].

A similar approach to Wang's [28] was followed by Rigamonti et al. [35] who used fuzzy c-means algorithm [30] to cluster the different operating conditions present in the second CMAPSS dataset [32]. Data normalization was performed also with the standard rule [31] considering the data ranges of each regime.

Other authors took into consideration operating regimes but adopted slightly different baselining strategies. For example, Rigamonti et al. [36] studied the case of capacitor degradation under variable operating conditions. Baselining was achieved with a relatively simple procedure. The authors proposed a degradation indicator independent of operational temperature. The indicator was defined as the ratio between the equivalent series resistance measured at a given temperature and its initial value at the same temperature. The health indicator was applied to estimate the capacitor RUL on both simulated and real data with satisfactory results.

Another significant contribution is the work of Ramasso [37] who proposed a health indicator that was computed at each point in time considering the current operating regime. The author used a clustering technique to determine the assignment of the data to the different regimes. The author showed that among the various characteristics tested, the operating conditions had the most significant impact on RUL estimation. Tests were performed on the second and fourth CMAPSS datasets.

Bektas et al. [38] also introduced the notion of operating regime into the construction of a health indicator. Specifically, the authors proposed a multiple linear regression transformation to reduce multi-regime data's dimensionality from the original scales to a common scale. The method worked by clustering the sensor readings at each operational mode and by performing dimensionality reduction to these clustered readings. By using this method, the raw values of the different time series of an asset, often inconsistent with each other, are transformed from a high-dimensional space to a space of a single wear level dimension.

Siegel and Lee [39] used K-means clustering [6] to take into account operating regimes while doing prognostics. The K-means technique was used to partition the wind speed mean values into two clusters. By clustering these data, the authors showed it was possible to focus on the samples where the anemometer was lagging and where the fault signature was more evident. The authors based their work on the empirical findings of Hale et al. [40].

More recently, Bian et al. [41] proposed a Bayesian prognostics framework for systems operating under dynamic regimes. The operational and environmental conditions were modeled as a continuous-time Markov chain. Peng et al. [27] proposed the use of parametric inverse Gaussian process models to model degradation rates with monotonic S-shapes. A case study involving a heavy-duty machine tool's spindle system was analyzed. Li et al. [42] considered the effects of time-varying operating conditions on RUL prediction by analyzing changes in degradation rates and sudden jumps in degradation rates at condition change-points separately. Their prediction method included these two factors into a state-space model. The solution was evaluated on a simulation study and a case of rolling element bearings.

Despite the importance of these works to the field, some questions regarding baselining remain to be addressed. For instance, most baselining models require some configuration, i.e., the number of clusters or some statistical properties (e.g., mean and standard deviation) of a representative dataset. In this work, neural network approaches are utilized to perform baselining. Notably, the system does not need to know any specific parameters about the problem at hand, such as the number of regimes. Also of relevance is that there is no need for a hold-out dataset to extract statistical information to complete the normalization process.

3. Problem

It is assumed the existence of U units characterized by a set of S condition monitoring features subject to R operating regimes. For a given unit u, with a duration of T_u units of time, each feature s is represented by a vector of measurements

$$\vec{m}_t^u = \left[m_1^{us}, m_2^{us}, \ldots, m_{T_u}^{us}\right] \tag{1}$$

Each measurement $\vec{m}_t^u$ corresponds to a regime Reg_r from the set of possible operating regimes

$$f_a(\vec{m}_t^{us}) \in \{Reg_r\}_{r=1}^{r=R} \tag{2}$$

The first objective of a baselining approach is to find a function $f_a(.)$ that can map the condition monitoring data to the operating regime space. Specifically, the function should classify each data sample $\vec{m}_t^{us}$ into one of the Reg_r operating regimes. The result of this first step is a collection of clusters such that each cluster of data corresponds to an operating regime

$$C_r = \left\{\vec{m}_t^u : f_a(\vec{m}_t^{us}) = Reg_r\right\} \tag{3}$$

Fig. 1 illustrates the first step of baselining as the first rectangle with rounded corners. Note that to find the regimes, the input can be either the original sensor data or some control and ambient variables. This paper investigates the utilization of a number G of ambient and control indicators.

The second goal of baselining is to combine the data of the different clusters C_r using a function $f_b(.)$. This function should be able to transform the condition monitoring data. The parameters of $f_b(.)$ are the mapping discovered in the previous baselining step and the original health data.

$$y_t^{us} = f_b\left(\vec{m}_t^{us}, f_a(\vec{m}_t^{us})\right) \in \mathbb{R} \tag{4}$$

For each unit u and feature s, the result of this second step is a vector of data

$$y_t^{us} = \left[y_1^{us}, y_2^{us}, \ldots, y_{T_u}^{us}\right] \tag{5}$$

The conceptual objective here is to factor out the influence of the different operating regimes without losing any meaningful information to prognostics. Typically (but not necessarily) this procedure is accomplished by normalizing the data according to the corresponding cluster's range. Fig. 1 summarizes the described baselining flow.

4. Theoretical background and adaptations

This paper proposes a neural network system based on a SOM and an MLP to address the baselining problem. The SOM technique is used in the first step of the baselining flow in Fig. 1. The goal here is to find a suitable function (Eq. 2) capable of partitioning (clustering) the condition monitoring data into different regimes given a set of ambient and control variables. The MLP network is used for the second step of Fig. 1 to normalize the data (Eq. 4). This network receives as input the sensor data and the regimes detected by the SOM.

SOM and MLP methods share the same theoretical background, as they are both neural networks. The idea underlying these two techniques' choice is to integrate different networks within the same neural system to have a multi-purpose architecture. The ultimate goal, to which this paper provides essential progress, is combining various neural networks in the same architecture to perform prognostics in a single iteration. The aim is to advance a model that can be an integral part of deep learning prognostics systems. Hereafter follows a description of the theoretical background of each technique and motivation for their use.

4.1. Self-organizing map

This section describes the theoretical background of the SOM technique and the adjustments and adaptations performed to fit the goal and scope of this work.

The Self-Organizing Map (SOM) was originally proposed by Kohonen in 1982 [43] to solve problems related to data visualization and abstraction. Since then, this technique has been used in a wide range of applications [44]. SOMs are particularly useful [45] to perform clustering. Clustering' methods are techniques able to "divide a set of n observations into g groups so that members of the same group are more alike than members of different groups" [46]. In this paper, the clustering capabilities of the SOM are used to group condition monitoring data into different operating regimes given an input set of ambient and control variables.

The SOM can be viewed as a two-layer neural network, with an input layer and a computational layer. The input layer is composed of P neurons, one for each ambient or control variable ($P = G$), such that the input vector is

$$\vec{p} = [p_1, \ldots, p_P] \tag{6}$$

The computational layer forms a grid of geometrically ordered neurons. Typically, this grid is 2-dimensional such that $size_x \times size_y = N_{SOM}$. Each neuron in the computational layer $\{z_1, \ldots, z_{N_{SOM}}\}$ is connected to all the source nodes $\{p_1, \ldots, p_P\}$ in the input layer. It is also assumed the existence of a weight vector $\vec{w}_j \in \mathbb{R}^P$ associated to each computational neural node $\vec{z}$ of index j. The weight vector expresses the connections established between a computational neuron and all the nodes in the input layer. There is a time dimension t, such that $\vec{p}(t)$ is the input vector selected at time t and $\vec{w}_j(t)$ is the weight vector at time t of a computational neuron $\vec{z}_j$.

There are a number of variants [47] of SOM. In this work, the batch version of SOM is used. However, and for clarity, this section starts by describing the online variant of SOM. In online SOM [47], a random vector is provided to the network at each time t and the Euclidean distance between input and weight vectors is computed as

$$\mathrm{d}_j(t) = \|\vec{p}(t) - \vec{w}_j(t)\|^2 \tag{7}$$

The BMU (c) for a given data sample $\vec{p}$ is the j that minimizes the distance function $\mathrm{d}_j(t)$. According to theBMU found for the sample $\vec{p}, c$, the weight vectors are updated at time t:

$$\vec{w}_j(t+1) = \vec{w}_j(t) + \alpha(t) h_{cj}(t)\left[\vec{p}(t) - \vec{w}_j(t)\right] \tag{8}$$

where $\alpha(t) \in]0, 1[$ is the learning rate and $h_{cj}(t)$ is the neighborhood function. The learning rate controls the magnitude of the update and it must converge to zero to ensure convergence of the training process. The neighborhood function determines the rate of change according to the BMU neuron.

In this work, the standard Gaussian neighborhood function is used such that

$$h_{cj}(t) = \frac{\exp\left(-\|v_j - v_c\|^2\right)}{\sigma_{SOM}(t)^2} \quad (9)$$

where v_j and v_c are the coordinates of neurons j and c BMU on the SOM grid. The width σ_{SOM} of the neighborhood function decreases during training.

Instead of the online version of SOM, batch SOM [47] is used in this work due to performance reasons. The batch variant updates the weight vectors only at the end of each epoch, after a single pass over the input data using the following formula

$$\vec{w}_j(t_{final}) = \frac{\sum_{t=t_0}^{t=t_{final}} h_{cj}(t)\vec{p}(t)}{h_{cj}(t)} \quad (10)$$

$$\mathrm{d}_j(t) = \|\vec{p}(t) - \vec{w}_j(t_0)\|^2 \quad (11)$$

$$c(t) \equiv \arg\min_j \mathrm{d}_j(t) \quad (12)$$

where t_0 and t_{final} are the start and end times of the current epoch, $\vec{w}_j(t_{final})$ the weight vectors at the end of the current epoch and $\vec{w}_j(t_0)$ are the weight vectors computed at the previous epoch. The winning node BMU at each time t is $c(t)$. The neighborhood function $h_{cj}(t)$ was previously described in Eq. 9.

The batch version of SOM has several advantages over the conventional online SOM. Since the algorithm works in batch, there is no dependence upon the input vectors' order so that data samples influence the outcome equally. There is also no need for a learning rate $\alpha(t)$ which reduces the probability of non-convergence [48]. The algorithm is also faster and has a more straightforward computation process since it does not include an update step at each input submission.

The learning process of SOM aims to move the weight vectors of the winning neuron and its neighbors closer to the input nodes. The result is a map that discretely represents the input data in a lower-dimensional space. However, and as noted by Vesanto and Alhoniemi [49], when the number of neurons in SOM is large, neighbor neurons might need to be further grouped. In their work, Vesanto and Alhoniemi [49] review different approaches to this problem, such as agglomerative clustering [50]. Connected components labeling is applied in this modeling approach to perform the final grouping of the neurons of SOM. Here, by "connected component" it is meant a subgraph of an undirected graph in which any two vertices are connected by a path [51].

The idea of using the connected components labeling approach with SOM was inspired by the work of Hamel and Brown [52] who used a model to connect all the neural elements that lie close together in data space. However, in this paper, the enhanced version [53] of the Hoshen-Kopelman algorithm [54] is used to group winning neurons at the end of the last epoch. The proposed solution is different as it is based on 4-connectivity, which means that a neuron is a 4-neighbor of a given neuron if the neurons share an edge in the SOM grid.

In the proposed SOM, illustrated in Fig. 2, an input vector ($\vec{p}$) corresponds to a collection of ambient or control variables obtained at a given time. The SOM learns the weights of a grid of N_{SOM} neurons in a certain number of batch I_{SOM} iterations. This training process results in a set of "active" neurons (in the computational layer). These active neurons are the winning neurons found during the learning process. One or more adjacent active neurons may represent an operating regime. Connected component labeling helps grouping neighbor neurons that belong to the same regime. Implementation is based on the Python MiniSom package [55].

4.2. Normalizing multi-layer perceptron

This section describes the theoretical background of the MLP technique and chosen architecture. It includes an explanation for each design option. It also provides a formal description of the network functionality.

The Multi-Layer Perceptron (sMLP) or FFNN is the classical neural network model. The MLP can be composed of one or more layers whose neurons are fully connected. The network is fully connected in the sense that each neuron in a layer is connected to all the neurons in the previous layer. Despite this forward dependence between successive layers, neurons (and their weights) are independent in the same layer.

The specific goal of an MLP is to approximate some function $f(.)$. Given, for example, a mapping of input $\vec{x}$ to output $\vec{y}$, the MLP attempts to find the function f such that $\vec{y} = f(\vec{x};\gamma)$, and the best parameters γ for it. In this learning process, there is an important function, the error (or loss) function (E_W), that is responsible for determining how much the predicted output ($\widehat{y^{(n)}}$) corresponds to the ground truth ($y^{(n)}$). The lower the error function, the closer the predicted value is to the actual output, and the better the approximation is (unless the model has over-fitted). Another core component of the network is the optimizer. The optimization goal is to find the set of weights, W^{MLP}, that minimizes the loss function. The optimizer works by iterating over the search space, usually moving in the direction defined by the error function's gradient. The learning rate (η) defines the extent to which the algorithm moves in every iteration (epoch).

The MLP specific implementation, here shown in Fig. 3, is not strictly an MLP, **as it works in an unsupervised way**. The network does not approximate the function f based on the labels $\vec{y}$. Indeed, the algorithm does not consider any ground truth $\vec{y}$. Rather, the error function has the goal of minimizing the square of the predictions ($\widehat{y^{(n)}}^2$). The goal here is to force the network to reduce the output signal. The square function is used to prevent the network from receiving a benefit in making the signal negative.

There are other cases, such as the one described in this paper, where the neural network's goal is not to approximate a function

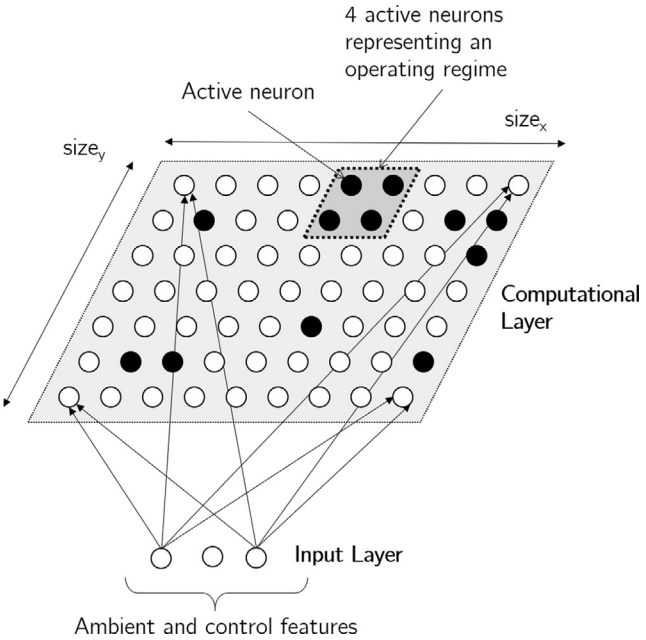


Fig. 2. A Self-Organizing Map (SOM) is proposed to cluster the different operating regimes present in the condition monitoring data.

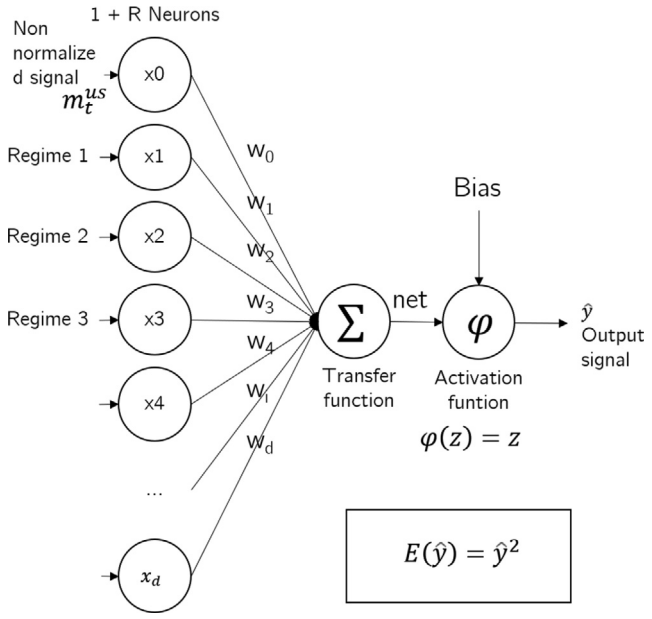


Fig. 3. A Normalizing Multi-Layer Perceptron (MLP) is proposed to normalize the sensor readings under different operating regimes.

but to optimize some network properties. For example, variational auto-encoders [56] can learn data distributions in an unsupervised way by optimizing a Gaussian prior that is solely a function of the hidden layers [57]. Sparse auto-encoders are also based on the principle of unsupervised machine learning [58]; they work by placing a sparsity constraint on the activations of the hidden layers. In the case of the MLP proposed, a restriction is placed on the output signal. This restriction allows obtaining a baseline of the input signal without having to supervise the network. The novelty of the "normalizing" MLP is the underlying idea that it is possible to optimize the network using an error function that does not depend on the difference between network input and output but only on the output. **The authors believe this is an idea of significance both to prognostics and machine learning in general.**

Different network architectures were designed, and the simplest model that was able to produce the desired results for the case study (Section 5) is shown in Fig. 3. The network has two layers: (i) input layer with $R+1$ neurons where R is the number of regimes and (ii) the output layer with one neuron. The first layer provides two important sources of information to the network at each time: i) a non normalized sensor reading ($\vec{m}_t^{us} \in \mathbb{R}$) and ii) a collection of R inputs, each for an operating regime, indicating to which operating regime the sensor reading belongs to ($Reg_r \in \{0,1\}, r \in \{1,2,\ldots,R\}$) multiplied by a sufficiently large factor, such as the maximum value of the sensor ($\max(\vec{x})$).

It is necessary to place the regime variables on the same scale as the (non-normalized) input signal. If the regime values were to be binary variables, they would be considerably smaller than the sensor reading values (when active). In such a case, the regime variables' significance would be lost as the values forwarded through the network, and these variables would have a small contribution to the error function. As a result, and due to the way backpropagation works [59], the associated weights would suffer only minor adjustments not producing the expected signal normalization within each regime. All the inputs need to be at a comparable range to give the same relevance to the data.

The simplest activation function ($\varphi(z)$), the linear one, is utilized. This is the simplest function as it does not change the input values

$$\varphi(z) = z \tag{13}$$

The output is therefore defined as

$$\widehat{y^{(n)}} = \varphi(\text{net}) + \text{bias} = \varphi\left(\sum_{d=1}^{R+1} w_d^{(n)} x_d^{(n)}\right) + \text{bias} \tag{14}$$

where $x_d^{(n)}$ is the d-th input, φ is the activation function, $w_d^{(n)}$ is the weight on the connection from the input unit $x_d^{(n)}$ to the output unit, net is the weighted sum of inputs and bias is the output bias

The optimization of the network is carried by the mini-batch gradient descent method based on the following error function

$$E_W\left(\widehat{y^{(n)}}\right) = \widehat{y^{(n)}}^2 \tag{15}$$

where E_W is the decision function, $\widehat{y^{(n)}}$ is the actual output of the last neuron

For a better understanding of how the MLP network works, it is presented a derivation of the network gradients. This derivation is used by the gradient descent method to update the weights of the network in its backward pass. In order to calculate the partial derivative of the error function with respect to an input weight $w_d^{(n)}$, the chain rule is applied twice

$$\frac{\partial E_W}{\partial w_d^{(n)}} = \frac{\partial E_W}{\partial \widehat{y^{(n)}}} \frac{\partial \widehat{y^{(n)}}}{\partial w_d^{(n)}} = \frac{\partial E_W}{\partial \widehat{y^{(n)}}} \frac{\partial \widehat{y^{(n)}}}{\partial \text{net}} \frac{\partial \text{net}}{\partial w_d^{(n)}} \tag{16}$$

Solving the last factor, only one term in the sum depends on $w_d^{(n)}$, becoming

$$\frac{\partial \text{net}}{\partial w_d^{(n)}} = \frac{\partial}{\partial w_d^{(n)}}\left(\sum_{d=1}^{R+1} w_d^{(n)} x_d^{(n)}\right) = \frac{\partial}{\partial w_d^{(n)}}\left(w_d^{(n)} x_d^{(n)}\right) = x_d^{(n)} \tag{17}$$

The derivative of the second factor is the partial derivative of the activation function which in our case is the linear function

$$\frac{\partial \varphi(\text{net}) + \text{bias}}{\partial \text{net}} = \frac{\partial \varphi(\text{net})}{\partial \text{net}} = \frac{\partial}{\partial \text{net}}(\text{net}) = 1 \tag{18}$$

The first factor can be derived as

$$\frac{\partial E_W}{\partial \widehat{y^{(n)}}} = \frac{\partial}{\partial \widehat{y^{(n)}}} \widehat{y^{(n)}}^2 = 2\widehat{y^{(n)}} \tag{19}$$

Substituting Eq. 17, Eq. 18, and Eq. 19 in Eq. 16

$$\frac{\partial E_W}{\partial w_d^{(n)}} = \frac{\partial E_W}{\partial \widehat{y^{(n)}}} \frac{\partial \widehat{y^{(n)}}}{\partial \text{net}} \frac{\partial \text{net}}{\partial w_d^{(n)}} = \frac{\partial E_W}{\partial \widehat{y^{(n)}}} \frac{\partial \widehat{y^{(n)}}}{\partial \text{net}} x_d^{(n)} = x_d^{(n)} \delta \tag{20}$$

with

$$\delta = \frac{\partial E_W}{\partial \widehat{y^{(n)}}} \frac{\partial \widehat{y^{(n)}}}{\partial \text{net}} = 2\widehat{y^{(n)}} \times 1 = 2\widehat{y^{(n)}} \tag{21}$$

Note that at the end of each mini-batch of size K, the weight adjustment is done by averaging over the gradients. With this in mind and from the previous equations, it is possible to derive the weight update equation

$$\Delta w_d^{(n)} = -\frac{1}{K}\sum_{n=1}^{K} \eta x_d^{(n)} \delta_n = -\eta \frac{2}{K} \sum_{n=1}^{K} \widehat{y^{(n)}} x_d^{(n)} \tag{22}$$

where η is a fixed constant ($\eta > 0$) that represents the network learning rate

Notably, during the first epochs, the first weight (w_0), i.e., the weight associated with the original input signal, will decrease more rapidly than the other weights. This is because the update of the other weights, associated with the regime variables, will consider several batch samples in which the term $\widehat{y^{(n)}} x_d^{(n)}$ is zero

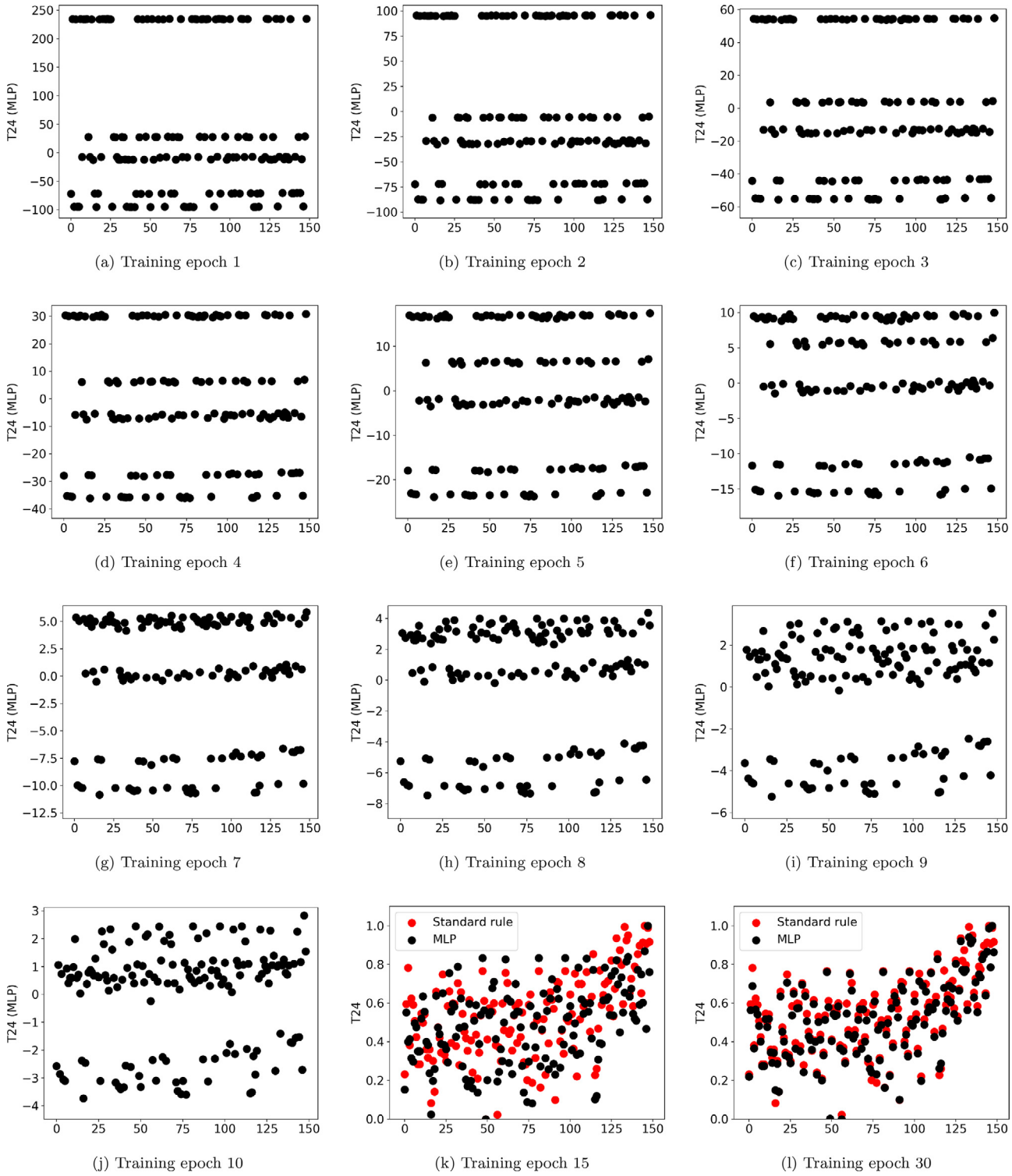


Fig. 4. Training examples with Multi-Layer Perceptron (MLP).

due to $x_d^{(n)} = 0$ (meaning the data sample does not belong to that regime). This different update velocity will cause a weight difference, with the first weight becoming smaller at a faster pace than the other weights. This process will cause the signal's overall scaling down to happen much quicker than the adjustments produced by the regime variables. When the weight difference reaches a certain amount, the regime variables' contribution to the error will be more significant. At this point, the different regime signals will be pushed towards each other so that the loss (the output signal) is reduced towards zero. After this, the weights become increasingly small, and few changes occur.

For illustrative purposes, an example is presented in Fig. 4 which shows the output signal of the network at each successive training epoch, from epoch 1 to epoch 10, 15, and 30. The last two charts also show the original signals baselined using the standard rule. From an analysis of the charts, it is possible to see that

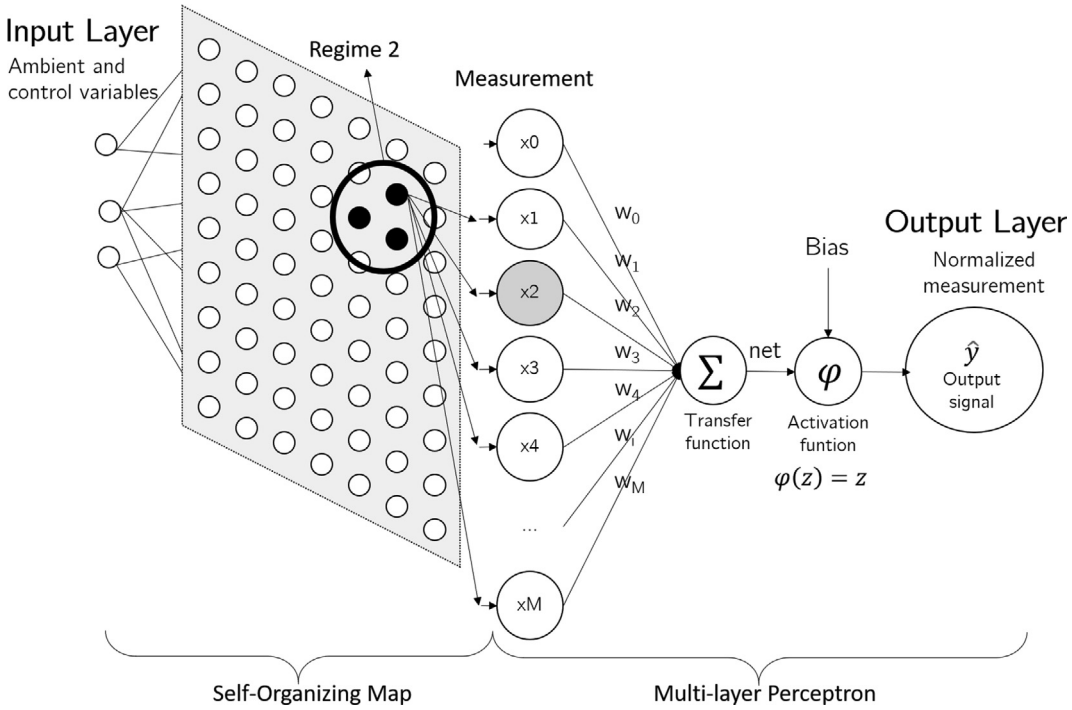


Fig. 5. Model overview. A SOM-MLP architecture is proposed to normalize sensors under different operating regimes.

Table 1
Configuration parameters of Self-Organizing Map (SOM) and Multi-Layer Perceptron (MLP).

SOM		
Size of grid	$size_x$ = $size_y$	Number of neurons per side of grid
MLP		
Epochs	I_{MLP}	Number of epochs to optimize the network
Weight min value	$v_{\min}$	Minimum value of uniform distribution used to initialize the network weights
Weight max value	$v_{\max}$	Maximum value of uniform distribution used to initialize the network weights
Learning rate	η	Learning rate of the network
Batch size	K	Batch size of network

the first epochs (Fig. 4a to d) cause a scaling down of the signals. From epochs 5 to 15 (Fig. 4e to Fig. 4k) the network collapses the fault responses resulting in a baselined signal. From epoch 15 to 30 (Fig. 4k to Fig. 4l), the network performs smaller and smaller adjustments to the weights resulting in a signal that resembles the one produced by the standard rule.

4.3. Final model

This section describes the final system and how the two network architectures previously described (Section 4.1 and 4.2 are combined.

The final model, shown in Fig. 5, is built by combining the two proposed networks, SOM and MLP. At each time t, the SOM inputs the operational variables that characterize the regime. From these features, it is possible to identify the operating regime. According to the regime to which the data sample belongs, R new input features are created. These will serve as input to the MLP. Each of these features will correspond to an operating regime and will be set either to zero or be set equal to the maximum value of the signal. The (already trained) MLP receives as additional input the sensor reading that it should normalize. The output is a normalized sensor reading. Note that the binary variables are multiplied by the sensor's maximum value to ensure the regime variables have the same weight as the sensor readings so that equal importance is given to the regime indicators and sensor readings.

The configuration parameters of the proposed system are described in Table 1. One advantage of the SOM approach is that it only needs the setting of the grid size (size = $size_x = size_y$). In Section 6.1 a sensitivity analysis is presented, and some general guidelines are discussed on how to set this parameter. The MLP has five hyper-parameters to set. It is necessary to set the number of epochs I_{MLP}. Also, the network weights are drawn from a uniform distribution between $v_{\min}$ and $v_{\max}$. The correct setting of these hyper-parameters is important to avoid the exploding gradients problem [60]. This problem occurs when large error gradients accumulate and lead to model instability. Mini-batch gradient descent is a popular variant of the algorithm of gradient descent in the field of neural networks [61]. This optimizer was selected over other alternatives, as mini-batch gradient descent uses several examples from the training dataset (i.e., $K > 1$) [61], instead of a single one, to compute the loss and update the network coefficients. The use of this optimizer allows having a minimum number of random examples from which to calculate the loss at each step (Eq. 15).

5. Case study

Complex engineering systems such as jet engines operate under different operating and environmental conditions. Such differences can significantly influence the values of condition monitoring variables such as temperature or vibration. The performance of jet

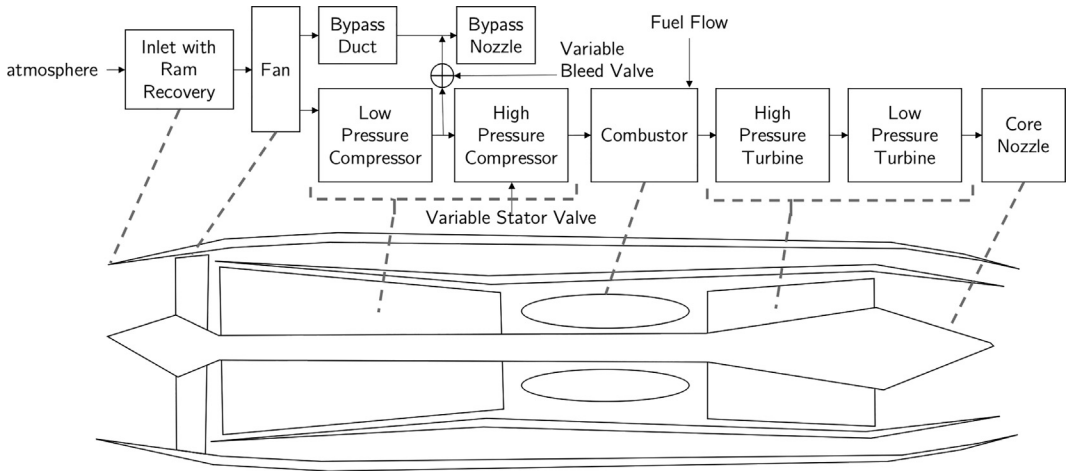


Fig. 6. C-MAPSS software modules and simplified diagram of turbofan engine.

engines operating under dynamic regimes is the case study of this paper.

The steady-state signature of a jet engine varies significantly with the different **operating conditions**. The conditions in which jet engines are operated can be determined by examining their main parameters such as the thrust and fuel consumption as well the jet velocity and flight altitude [62]. There are several distinct operating regimes [62]: take-off mode (with maximum permissible gas temperature and r.p.m.); continuous climb power mode (80–85% thrust and 95% r.p.m.); cruising power mode (80–85% thrust), idling power mode (5–7% thrust) and a holding mode. This work aims to distinguish among the operating regimes present in a turbofan engine's sensor data for posterior data normalization.

Jet engines also operate within considerably different **environmental conditions**, i.e., temperature and pressure of the intake air. These conditions also have an indirect impact on engine performance. As ambient pressure and temperature change significantly with altitude, variables such as total temperature and pressure at engine inlet change with these conditions. Air intake can affect the performance of the compressor and the combustion chamber and even result in damage.

The data used in this paper were generated using the state-of-the-art CMAPSS developed at the NASA Glenn Research Center [63]. The simulator emulates a large, high-bypass ratio turbofan engine similar to the GE90. The model is composed of several modules, as shown in Fig. 6. Six different flight conditions were simulated based on three operational conditions: altitude (0–42 K ft.), Mach number (0–0.84), and Throttle Resolver Angle (20–100). The TRA is the angular deflection of the pilot's power level, varying between 0% and 100%. The Mach number is the ratio of flow velocity to the speed of sound at the medium. Altitude relates to atmospheric conditions. Six operating conditions are only a subset of all possible operating conditions, which can be any combination of altitude, Mach number, and TRA.

The CMAPSS data consists of a collection of time series of observables at cruise snapshots produced due to variation of the High-Pressure Compressor module's flow and efficiencies from initial settings (nominal behavior) to failure values. Each degradation trajectory is characterized by series of observables (features) that include sensor and operational variables that change over time from some nominal condition to failure. The data is described in detail in [32].

In particular, Fig. 7 shows how the different operating regimes affect the CMAPSS data. The first plot (Fig. 7a shows a time series of one jet engine in CMAPSS. As depicted, no clear degradation pattern can be visualized. Applying K-means clustering to the operational variables of TRA, Mach number, and altitude allows to discriminate the different six regimes. Looking at the results of K-means and looking back at Fig. 7, it can be seen that the data of the different regimes are disposed around different values of the y-axis. For example, the values of the sixth regime are around 1500°R[2] while the values of the fifth regime are around 1300°R. Fig. 7b shows even more clearly how the data is on different levels hindering its analysis. However, when the different regimes' data are normalized, it is possible to see a clearer degradation pattern. Fig. 7c show the data after normalizing each regime time series with the standard rule [31]. A tentative degradation trajectory can be drawn on top of the new data as Fig. 7d depicts.

This paper analyzes the second and fourth CMAPSS training datasets. The second dataset comprises simulated data of 260 jet engines, and the fourth dataset 249 jet engines. These datasets are subject to six operating regimes. The first and third datasets of CMAPSS are less complex, having only one regime, which is not of interest to this study.

6. Results

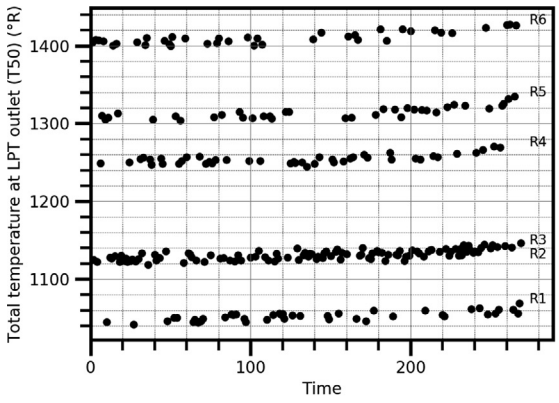
This section presents and discusses the results of the two experiments, A and B, designed to evaluate the proposed solution's baselining capabilities.

6.1. Experiment A

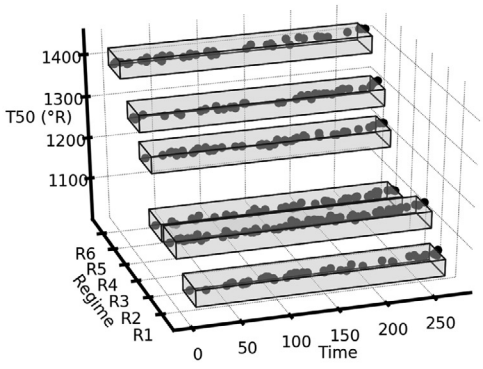
This section describes and discusses the results of experiment A, whose goal was to investigate the extent to which the proposed SOM network, previously described in Section 4.1, was able to separate, i.e., to cluster, the different regimes found in the data from the CMAPSS.

In experiment A, it was assumed that the regime classification of the sensor data produced by the K-means baseline method (with $K = 6$) was the correct one, i.e., it represented the ground-truth. In other words, it was assumed that K-means was able to partition the sensor data into six regime clusters with 100% of accuracy. The K-means approach to this case study has been validated by a significant number of works in the field of prognostics [35]. Also, several authors, such as Rigamonti et al. [35], have visually shown that the clusters generated by K-means on the CMAPSS case study are a good fit for the data.Being an unsupervised task and given the absence of ground-truth data, the classification generated by K-

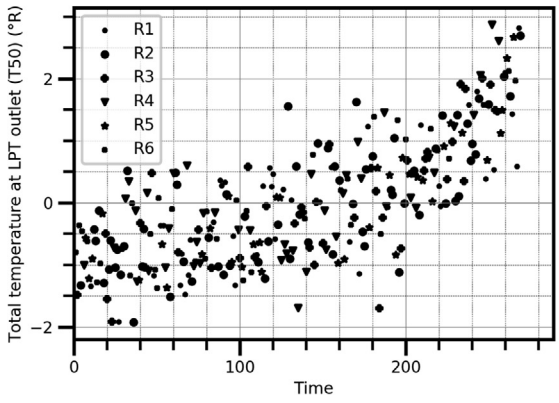
[2] °R is a temperature measurement used in aeronautics.



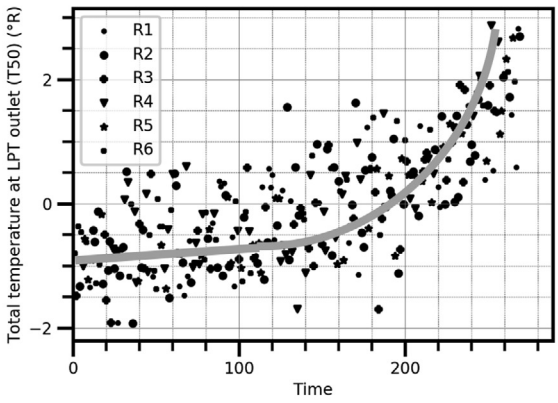
(a) T50 Sensor Readings of Unit in 2D (No Processing)



(b) T50 Sensor Readings of Unit in 3D (No Processing)



(c) T50 Sensor Readings Standardized within each Regime



(d) Tentative Trend Curve on top of Standardized T50 Sensor Readings

Fig. 7. KMeans regime detection based on different features vs KMeans regime detection based on C-MAPSS 3 operational condition variables.

means was assumed to be the best possible classification, and thus it was considered the correct one. This ensured that the comparison between the SOM method and the other methods was fair.

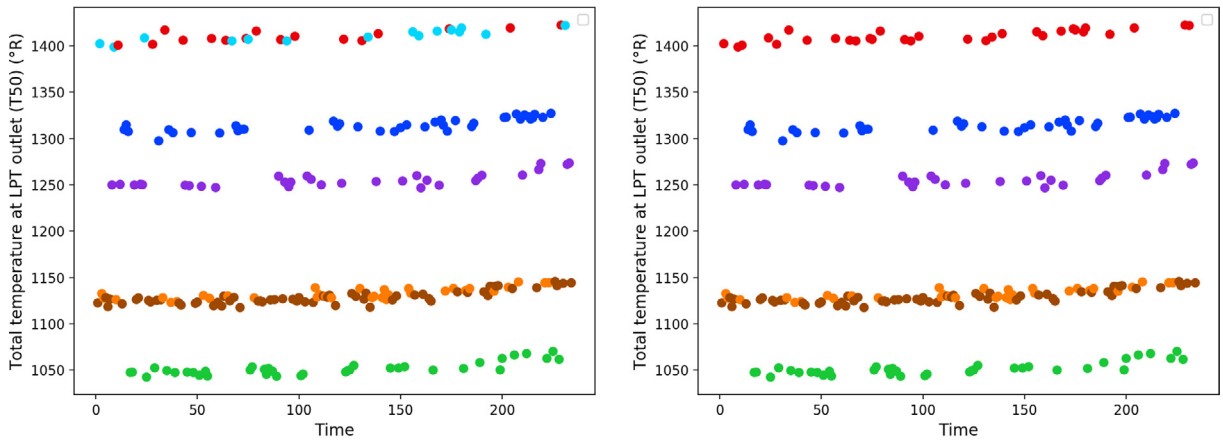
The regime separation performed by the SOM network was based on three ambient and control variables: altitude, Mach number, and TRA. The network, configured with the parameters of $size_x = size_y = 20$, clustered the condition monitoring data in two steps:

- Training stage: the three features of altitude, Mach number, and TRA were provided as training data to the network. The training procedure used data of only one randomly selected engine unit.
- Regime clustering stage: to characterize a collection of sensor measurements m_t^{ii} of an engine u at time t, the features of altitude, Mach number and TRA were utilized. By providing the values of these three features as input, the SOM predicted each unit's operating regimes at each time. The model classified the sensor readings of each unit into different regimes by iterating over all sensor data.

The SOM procedure was used on the second and fourth datasets of CMAPSS. After running the SOM model on the second CMAPSS dataset, 258 out of the 260 (engine) units were correctly clustered into the corresponding regimes, which is a success rate of 99.23%. The procedure yielded seven regimes for two units instead of the six K-means regimes. This result does not represent a problem to the overall process as the baselining of the MLP is robust to such clustering outcomes. As long as the newly found clusters are sub-clusters of a single cluster, accuracy is guaranteed. If the seventh cluster data belonged to different "ground-truth" clusters, the normalization would not consider the correct data ranges at the following MLP stage.Recall that the goal here is to have clusters representing a regime or a representative set of each regime's data. If two groups instead of one collection of data are found for one regime, that is an acceptable result. The aim is to divide the data for normalization and not necessarily to perform exact regime discovery.

On the fourth CMAPSS dataset, 247 out of the 249 (engine) units were correctly clustered.This represents a success rate of 99.20%. This rate was approximately the same success rate obtained in the second dataset, which may be explained by the fact that the data is subject to the same set of operating conditions and regimes in both scenarios. This result suggests that the difficulty of regime classification is correlated solely with the number of regimes and affecting conditions. The number of fault modes does not appear to influence the SOM outcome. Note that the fourth dataset has more fault modes than the second dataset.

The two units incorrectly clustered on the fourth CMAPSS dataset were matched with seven instead of six regimes. Fig. 8 shows how the first and second top clusters (red and light blue markers in Fig. 8a) found by SOM corresponded to a single K-means cluster (red markers in Fig. 8b) for a given engine unit.Again, this lack of



(a) Regimes found by SOM (b) Regimes found by K-means

Fig. 8. Example of regime detection by SOM and K-means for a randomly selected sensor of one engine unit. The clustered regimes are shown in different colors. In this example, the first and second top clusters found by SOM correspond to a single K-means cluster. This disparity between SOM and K-means can occur occasionally but can be corrected with several runs of SOM where the grid size varies.

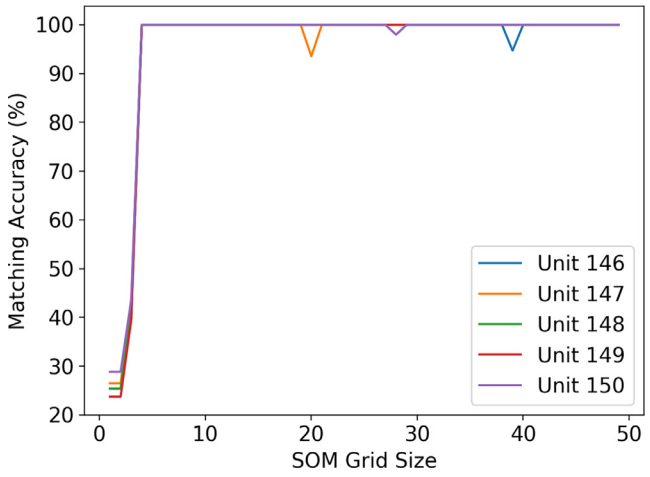


Fig. 9. Performance of SOM according to grid size.

precision of SOM does not necessarily mean a negative result as long as the SOM's additional clusters correspond to a single K-means cluster. This assumption, investigated by visual inspection, was met in all unit cases.

The only configuration parameter of the proposed SOM was the grid dimension. Accordingly, a sensitivity analysis was performed at this level. The classification accuracy of SOM was tested for different grid configurations on the two CMAPSS datasets. For simplicity, we assumed a square grid ($size_x = size_y$) and varied its size. Five engine units were tested. Fig. 9 shows the performance of SOM for different grid sizes. Performance is measured in matching accuracy (%), that is, the degree to which the classification of SOM matched that of the K-means. As can be seen, when the grid size is below five, the SOM shows difficulties in clustering the data correctly. This limitation reflects the grid being too small for the number of regimes and the neighboring neurons being merged by the connected component labeling algorithm. For large grids, the method works well most of the time except for some rare cases. For example, in Fig. 9, the method correctly determines that there are six regimes in the data for 42 of the 45 $\times 5$ cases when the grid size is above five. For these success cases, when the number of regimes is correctly computed, 100% of clustering accuracy is achieved for all measurements.

Table 2
Performance of Self-Organizing Map (SOM) measured in Mean Absolute Error (MAE) on datasets 2 and 4 ($size = 20$) for variable number of regimes.

	MAE	
Number of regimes	Dataset 2	Dataset 4
6	99.23%	99.20%
5	100.0%	100%
4	100.0%	100%
3	99.98%	99.93%
2	99.26%	99.61

These results indicate that there is a minimum grid size to make sure the SOM behaves accurately and stably. The grid size depends on the number of regimes that exist in the actual data. The more the number of regimes, the larger is the minimum size, and the contrary is also true. If we set the number of regimes as a large number, the SOM will tend to operate correctly in all circumstances, as the grid will be large enough to accommodate the different clusters. Please note that one cluster can contain several neurons since adjacent neurons are grouped into a single regime. Given this, it is necessary to make space for all clusters. As a simple rule, if the number of regimes is known, R, setting the grid size to R is sufficient as there will be $R \times R$ neurons in the grid.

Nevertheless, even when a sufficiently large grid size is defined, SOM may, on rare occasions, not achieve 100% accuracy. It may be necessary to test the SOM with different grid sizes and check whether the output is consistent in such a situation. By examining the results of distinct SOM grids, it is possible to have more confidence in the found number of regimes and the data segmentation. According to the results of the sensitivity analysis of SOM, it is preferable to use a large grid than a small one and inspect several grid outcomes instead of a single one. Based on these two strategies, there is a higher probability of clustering all units correctly.

To test the SOM ability's to accurately detect a different number of regimes, the fourth and second CMAPSS datasets were used to construct several subsets. The resulting sets had 5, 4, 3, and 2 regimes. The SOM network was then configured with $size_x = size_y = 20$ and ran on the different datasets. For all the experiments, the detection rate was above 99% as shown in Table 2. These results suggest that the SOM can have an acceptable performance compared to K-means. Again, the aim is not to outperform the optimal performance of K-means but to provide a comparable

method that had additional advantages. The investigation of cases in which the proposed method surpasses K-means' performance is out of this paper's scope, even though it is a future research goal.

The main advantage of SOM over K-means is that it does not need to know in advance the number of clusters, as long as the grid size is sufficiently large. Other algorithms such as DBSCAN [64], HDBSCAN [65] and MeanShift [66] also do not require knowing the number of clusters a priori. These methods have a good performance on C-MAPSS data achieving an accuracy of 100%, 99.91%, and 100% for DBSCAN, HDBSCAN, and MeanShift, respectively. However, conversely to the proposed method, these methods do not have a mechanism to validate the clustering results. By inspecting results across different SOM configurations, it is possible to characterize the degree of confidence in the obtained results. This result is an essential contribution in a scenario of unsupervised learning such as this one.

Another advantage of SOM is that it does not need additional parameters besides the grid size. Other clustering algorithms are typically more challenging to configure. One of the simplest clustering algorithms in configuration terms, the DBSCAN, actually requires two hyper-parameters: ε, which determines how close points should be to be part of a cluster; and *minPts*, which determines the minimum number of points to form a group.

Overall, the SOM model had a satisfactory performance on the second and fourth CMAPSS datasets. Importantly, this approach does not need the setting of additional parameters to operate correctly. Also, by observing different SOM configurations, it is possible to decide with considerable confidence if the SOM result is close to the ground truth outcome.

6.2. Experiment B

This section describes and discusses the results of experiment B, whose goal was to study the extent to which the proposed normalizing MLP network, described in Section 4.2, was able to perform the baselining of the CMAPSS data.

The standard rule's baselining method is a widely used approach in the field [35]. When standardizing a given set of data X in real-time, the mean and standard deviation parameters have to be estimated from an independent representative set $X^{holdout}$. For the standardization to make sense and not lead to misleading patterns, the set $X^{holdout}$ needs to have a mean value and a standard deviation close to the mean and standard deviation values of X. This requirement is particularly critical when it is necessary to enforce that within each operating regime subset of X the data is standardized – it has a mean of zero and a variance of 1. Ensuring this condition can make the degradation trend more transparent and obvious when the data is combined.

To ensure the fairness of the comparison between the MLP algorithm and the standard rule method, we assumed that the standardization used the exact parameters of the mean and standard deviation of the set X. The standardization used X as the hold-out set $X^{holdout}$. The outcome of this unrealistic yet "perfect" standardization was used to estimate the MAE of the predictions of the MLP. The goal here was to compare the MLP with the standard rule in an ideal scenario for the standard rule. Note that the MLP works without explicitly providing information to the algorithm about the mean or the standard deviation of the set, which is advantageous, especially when data are scarce. It is shown in this paper that the sensor data of a single piece of equipment is enough to baseline the data reasonably well.

The normalizing MLP was trained and executed to produce the desired outcome as follows:

- Training stage: the network received seven features: one feature representing a sensor reading and six binary features (multiplied by the maximum expected value of the sensor) representing the operating regime of the reading. Note that each sensor reading is related to a single regime. The six regime variables were obtained from the output of the SOM network. Two training conditions were tested: a) training with a randomly selected unit and b) training with all engine units.
- Prediction stage: the network operated on sensor data of each single engine unit. By iterating over all data points, the MLP generated baselined sensor data.

The training procedure of the MLP was tested under two different scenarios: a) with the data of only one engine unit and b) with the data of all available engine units. This testing strategy aimed at analyzing how well the network could operate with less training data. The lower the necessary volume of data, the faster the computation can be. The goal here was also to investigate the generalization of a network trained with fewer data.

After the MLP normalization of all units, results are evaluated against the reference method of standardization [31]:

$$m_t^{standardized} = \frac{m_t - \mu^r}{\sigma^r} \tag{23}$$

where $m_t^{standardized}$represents the s standardized sensor feature at time t given that m_t is the original s sensor feature at time t in the r^{th} operating condition, and μ^r and σ^r are the mean and standard deviation of the feature in the r^{th} operating condition. To evaluate the results, it was necessary to put the MLP's data onto the

Table 3
Performance of Multi-Layer Perceptron (MLP) measured in Mean Absolute Error (MAE) on datasets 2 and 4. Two training methods were tested: one using training data of a random unit and another using all data as training data.

	MAE			
	Dataset 2		Dataset 4	
	All units	Random	All units	Random
T24	2.75 ± 1.7	5.57 ± 1.1	2.57 ± 1.5	3.92 ± 1.2
T30	1.18 ± 0.7	4.78 ± 1.0s	1.3 ± 0.8	1.59 ± 0.7
T50	2.59 ± 1.2	4.57 ± 1.1	2.10 ± 1.2	2.86 ± 1.0
Nc	7.71 ± 3.1	4.53 ± 1.9	9.27 ± 4.1	8.27 ± 3.6
Ps30	1.99 ± 1.1	4.59 ± 0.9	1.54 ± 1.0	2.52 ± 0.9
phi	8.90 ± 4.0	8.41 ± 3.0	16.14 ± 5.3	17.6 ± 7.7
NRc	6.96 ± 3.1	5.39 ± 2.5	7.84 ± 3.7	7.89 ± 3.7
BPR	4.9 ± 1.2	6.2 ± 1.3	2.45 ± 1.0	2.94 ± 1.1
htBleed	1.54 ± 0.9	4.35 ± 0.8	1.60 ± 1.2	3.21 ± 0.9
W31	5.90 ± 2.7	6.18 ± 2.8	10.64 ± 3.8	11.83 ± 4.5
W32	5.82 ± 2.6	7.47 ± 2.0	10.33 ± 3.6	10.93 ± 4.0

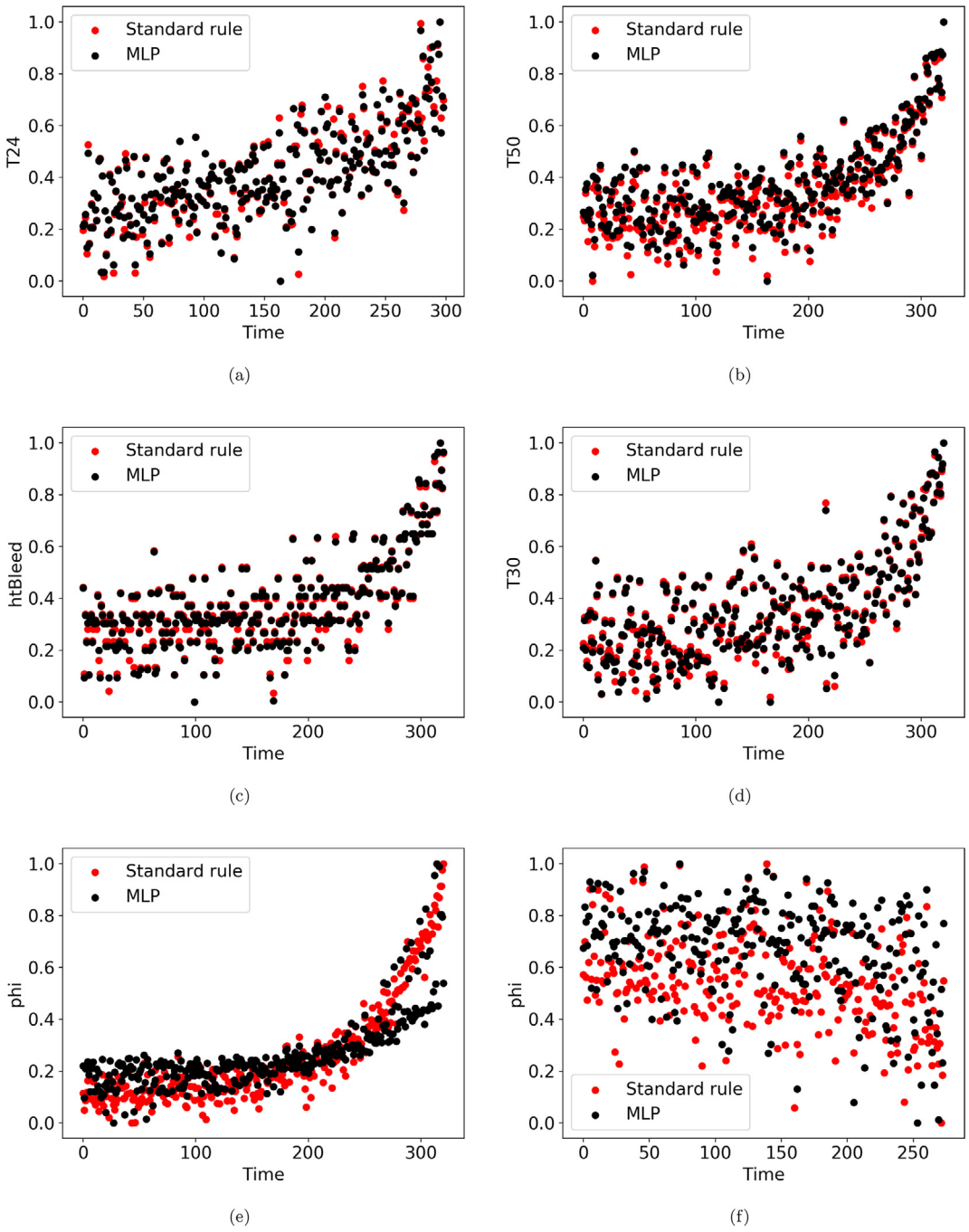


Fig. 10. Examples of normalization of different units (engines) by the Multi-Layer Perceptron (MLP) and the standard rule.

same range of the standardized data. To do this we applied mix-max normalization [31] to the output of both the standard rule (Eq. 23) and the MLP:

$$y^*(t) = \frac{x(t) - \min(y(t))}{\max(y(t)) - \min(y(t))} \tag{24}$$

where $y^*(t)$ represents the new feature at time t given that $y(t)$ is the sensor feature produced by baselining, and max and min are max and min values of the feature.

After having both signals in the range $[0, 1]$ we calculated similarity by:

$$\text{MAE}(u,s) = \frac{100}{T} \times \sum_{t=1}^{T} |y^*_{MLP}(u,s,t) - y^*_{SR}(u,s,t))| \tag{25}$$

where $\text{MAE}(u,s)$ is the mean absolute error of sensor s of unit u in percentage, $y^*_{MLP}(u,s,t)$ is the MLP output feature at time t (subject to min-max normalization) and $y^*_{SR}(u,s,t)$ is the standardized feature at time t (subject to min-max normalization). We multiply by 100 so that MAE performance is between 0 and 100. The closer the MAE is to zero the better.

Quantitative results of experiment B are presented in Table 3. The second and fourth columns show the similarity between the standardized signals and the MLP signals when the network is trained with all unit data. The third and fifth columns show the similarity between the standardized signals and the signals generated by the MLP after being trained with data from a random unit. As shown in Table 3, the results were satisfactory, especially when the MLP was trained using all the sensor data. The error was for the

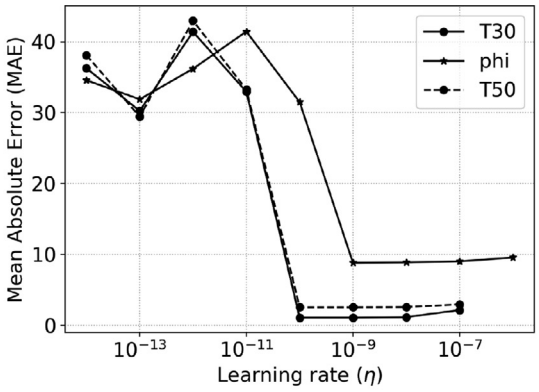
majority of the cases below the 10% threshold. These results suggest the network can approximate the optimal baseline behavior and can produce improved signals. Having a close to the optimal method in factoring out the influence of regimes can significantly impact data's quality and improve the final RUL estimations.

The MLP showed some limitations that are important to discuss. For some sensors, such as phi, the MLP exhibited higher errors than for most features (see Table 3). These results can be explained by the fact that the phi sensor can assume different shapes for different units. For example, in Fig. 10e and Fig. 10f, examples of two engine units with completely distinct shapes for the phi sensor are shown. Since the MLP tries to find a model that fits both shapes, results are not as good for phi as other, more trendable [67] sensors. One way to address this limitation would be to create several pMLP, one for each shape, and apply the most suitable model accordingly.

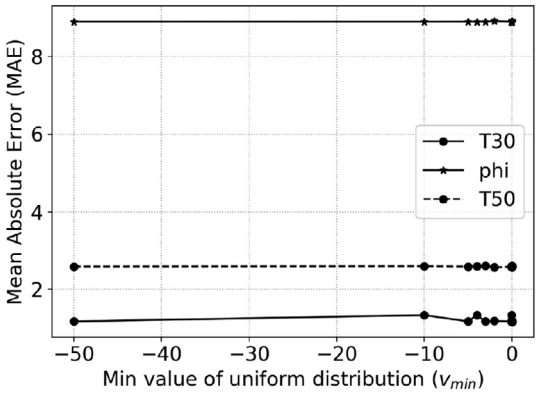
Analyzing Table 3 it can be seen that the MLP generalized well. The MAE values obtained using one random unit as train data were not very far from the values obtained with all units. This result suggests that the network has good adaptation capabilities. According to these results, the network can, after seeing few data, baseline the future data. This result is of critical significance for prognostics applications that involve few data samples.

Improving the data quality without needing to hold-out data is the ideal solution for data scarcity. This characteristic is perhaps the most advantageous property of the proposed MLP. There is no need to explicitly compute the standard deviation or mean values of the data from a large and representative hold-out dataset. In classical standardization, the designer is required to know or be able to estimate these values. This limitation prevents these classical methods from being used in situations where the equipment is critical, or the equipment is new, and few failure examples are available. In the MLP method, only a single unit is needed to train the network and produce nearly optimal results.

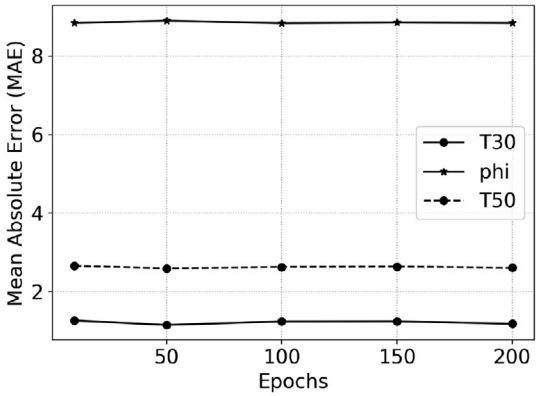
A parametric analysis of the normalizing MLP was performed to understand the effects of parameter setting on model performance (measured in MAE). The four parameters of Table 1 were considered. The analysis selected three sensor signals, the worst and best-performing features of the second dataset, respectively features phi and T30, and a randomly selected feature, T50. These three features were selected as a representative set of the sensor signals. The results of the analysis are shown in Fig. 11. From an analysis of Fig. 11b, it can be concluded that if the weight uniform initialization is within the interval $[v_{\min}, 0]$, changing the $v_{\min}$ did not produce significant changes in performance. This result suggests that the network performance is not sensible to the actual



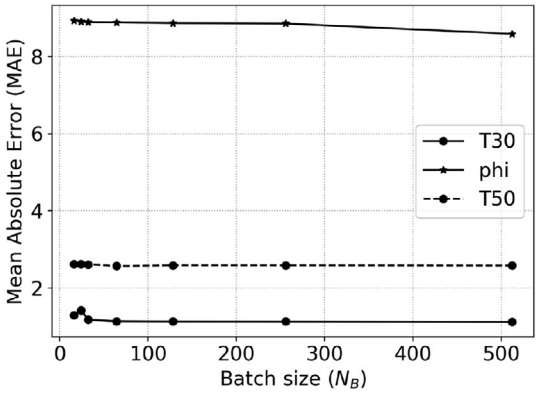
(a) MLP performance as a function of learning rate η, setting the weight uniform distribution with $v_{\max}= 0$ and $v_{\min} = -0.01$, number of epochs I_{MLP}=100, and batch size $K = 32$.



(b) MLP performance as a function of weight initialization $v_{\min}$ setting weight uniform distribution with the $v_{\max}= 0$, learning rate $\eta = 10^{-9}$, epochs I_{MLP}=100, and batch size $K = 32$



(c) MLP performance as a function of number of epochs I_{MLP}, with learning rate $\eta = 10^{-9}$, uniform distribution with $v_{\max}= 0$, $v_{\min} = -0.01$, and batch size $K = 32$



(d) MLP performance as a function of batch size K, with I_{MLP}=100, learning rate $\eta = 10^{-9}$, weight uniform distribution with $v_{\max}= 0$, $v_{\min} = -0.01$.

Fig. 11. Parametric analysis of Normalizing Multi-Layer Perceptron (MLP) on features T30, phi and T50 (dataset 2).

values of its initial weights. This finding is positive as it suggests that this hyper-parameter is straightforward to set to obtain the desired results.

Seemingly, the parameter of the number of epochs (see Fig. 11c) did not appear to affect baselining performance (measured in MAE). This result can be explained by the fact that the network was designed to prevent over-fitting. By not relying on external ground-truth data, the proposed MLP can more quickly perform its function in the initial epochs. The batch size parameter also did not produce a substantial performance change (measured in MAE).

As depicted in Fig. 11a, the learning rate (η) of the network has to be selected with caution and is one of the most influential parameters of the set of four hyper-parameters of the MLP. It determines each epoch's step size while moving toward the minimum of the error function (E_W). As shown in Fig. 11a, above a superior threshold, 10^{-7} for T24/T50 and 10^{-6} for phi, the network produced undetermined output. This behavior may be explained by the network minimizing the signal too quickly when the step size is large. Below a low threshold, 10^{-10} for T24/T50 and 10^{-9} for phi, the network had a significant decrease in performance. This behavior may be explained by the network taking too long to minimize the signal.

7. Conclusion

Systems are rarely in a steady-state operating mode. Instead, there are continuous system signatures caused by variations in the operational and environmental conditions or system dynamics. The operational and environmental variations are not descriptive of system degradation, whereas changes in system dynamics are and need to be further analyzed. Therefore, it is vital to reduce the influence of operational and environmental conditions (i.e., of the regimes) on the system's signals. In this work, we consider a "baselining" procedure composed of two steps: 1) regime identification and 2) data normalization. The product of these steps is a collection of sensors normalized within the range found in each regime.

The applications of this work are numerous. In prognostics, the science of forecasting the equipment's future health state, it is well known that the operating conditions significantly impact model performance. In fact, in the case of CMAPSS, the variations in the data produced by the different operating conditions are more dramatic than the variance introduced by the progressing faults. This fact is also true for other types of equipment and scenarios. Making operating data condition-invariant by using advanced methods is therefore of considerable utility to the general field of PHM. Also, areas such as SHM could benefit from the analysis of this work, as it is essential to factor out the influence of operating conditions before conducting proper structural damage modeling. In general, this work intends to promote a better understanding of the behavior of engineering systems.

Being able to identify the set of operating regimes of a system is essential. However, most approaches used in the literature assume that the number of regimes is known. The proposed self-organizing map provides for the automatic clustering of the operating regimes. It does not accept a parameter with the number of regimes. We evaluate the clustering quality by varying the grid size and analyzing its effect on the network's outcome.

It is important to note that the self-organizing map approach's contribution is not to outperform the standard clustering methods such as K-means or DBSCAN neither in accuracy nor in runtime. See A and B for an analysis of the runtime performance and overall time complexity. The actual contribution is the proposal of a method that can partition the regimes and that:

- Does not require knowing the number of regimes (or clusters) in advance
- Has a mechanism to detect when a data sample is being incorrectly classified
- Is a solution that can inspire integrated neural network solutions for diagnostics and prognostics

According to the regime indicator, aggregating data under different operating regimes into one composite index typically requires the standardization of the original raw data. However, in classical normalization, a suitable mathematical formula needs to be selected, and statistical parameters need to be known or estimated. For example, when using the standard rule, it is necessary to put aside data to compute the mean and standard deviations of sensor data in each operating regime. This step can be problematic in many data-driven applications in which the availability of data is essential. Excluding part of the data for the only purpose of mean/standard deviation computation is an expensive alternative. The normalizing multi-layer perceptron proposed in this paper addresses this difficulty.

A characteristic of the proposed model (self-organizing map combined with the multi-layer perceptron) is that it does not need much training data. It works in two stages. First, and for a feature (sensor) we want to normalize, we provide data of one or more units. The network is trained with this data. After the network is trained, the same information is used for prediction. The output data is normalized. Please note the importance and uniqueness of the proposed normalizing multi-layer perceptron network. Traditionally [68], data needs to be normalized before it is fed into a neural network. The model normalizes the input data according to the different normalization regimes given as input.

The contribution of the multi-layer perceptron approach is to show that it is possible to baseline sensor data in a nearly optimal way while:

- Not requiring large volumes of training data
- Being a neural network solution that works in an unsupervised manner and can receive non-normalized inputs
- Being again a solution that can inspire integrated neural network solutions for diagnostics and prognostics

This paper considers an unknown set of operating regimes that influence the overall degradation trajectories shown in the condition monitoring signals. We assume that, within each regime, the data forms a clear degradation trajectory, and therefore, the problem is to combine the data from the different regimes. If this assumption does not hold, our method is not able to effectively perform the baselining. The self-organizing map is designed to handle a vast number of operating regimes. Nevertheless, when dealing with a high number of regimes, the designer needs to set the grid size to that number, and the discovery of the regimes can be a slow process. We also assume that the operational and environmental variables are known. The multi-layer perceptron method showed some difficulty to baseline data displaying different trajectory shapes within the same fleet. Developing separate models for each trajectory shape may help achieve higher accuracy.

Future work will further analyze the properties of the model outcome. An area of improvement is reducing the level of noise present in the output of the multi-layer perceptron. Also, it would be interesting to integrate these machine learning steps into a comprehensive data-driven prognostics approach.

8. Code

The code is open-source and is available at [69,70].

CRediT authorship contribution statement

Marcia Lourenco Baptista: Conceptualization, Writing - original draft, Methodology, Software. **Elsa M. P. Henriques:** Supervision, Writing - review & editing. **Kai Goebel:** Supervision, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Computational complexity

Setting some parameters fixed, the time complexity of the approach can be shown to be linear with the size of the input. It is important however to note that, at its worst, the time complexity of the overall model can become quartic. In this work, it is assumed that the number of operating conditions is small. Hereafter follows formal proofs.

A.1. Self-organizing map

Let N_{SOM} be the number of units in the SOM, G the dimension of each input vector, n the number of sample vectors, and I_{SOM} the number of epochs of the batch version. The dimension of each input vector G is equal to the number of operating condition indicators. The batch version of the SOM has the complexity of $O(N_{SOM} * G * n * I_{SOM})$ [71]. The computational cost of the algorithm depends for the most part on the number of sample vectors. The running time is linear with the size of the input n setting the other variables as a fixed number. However, at its worst, when the number of operating conditions, number of neurons, and iterations equals the input size, the time complexity of the SOM can become proportional to $O(n^4)$.

A.2. Multi-layer perceptron feed-forward pass

The time complexity of the training and execution stages of the proposed normalizing MLP are linear in respect to the input size. At this worst, the algorithm can become quadratic.

Note: In the following calculations we assume that the time complexity of matrix multiplication for $M_{ih} * M_{hk}$ is simply $O(i * h * k)$. There are other approaches with better time complexity but typically feed-forward and back-propagation algorithms are implemented with matrices.

Since the network is composed of $R+1$ input neurons and one single output neuron, a matrix is necessary to represent the weights between the input layer and the output layer. Let W^{MLP} denote this matrix which has $R+1$ rows and h columns. Here, $R+1$ is the number of nodes in the input layer, i.e., the number of operating conditions + 1 sensor measurement, and h is the number of nodes in the output layer equaling one in the proposed multi-layer perceptron architecture. It is assumed the existence of n training examples. The input is represented by the X_{in} matrix, where there are n input samples with $i = r+1$ features each. To propagate from input layer to the output layer the following operation applies,

$$S_{hn} = W_{hi}^{MLP} X_{in} \tag{A.1}$$

This matrix multiplication has $O(n * i * h)$ time complexity. Because $h = 1, W^{MLP}$ is a vector instead of a matrix. To apply the activation function $\varphi(.)$ to S in order to complete the feed-forward step the following operation applies,

$$Z_{hn} = \varphi(S_{hn}) \tag{A.2}$$

This procedure has $O(hn)$ time complexity, because it is an element-wise operation. Note that the activation function $\varphi(.)$ is the linear function. The overall time complexity for the forward pass algorithm of the proposed network is therefore $O(n * i * h + n * h) = O(n * (i+1))$ where n is the number of samples and i is the number of operating conditions plus one. Each data sample is a collection with one sensor measurement obtained from a monitoring in addition to several binary variables indicating the measurement operating condition.

$O(n * (i+1))$ is not linear for both the variables n and i at the same time, but it is linear if one variable is set constant and the other one varies. A running time of $O(n * (i+1))$ is therefore linear with the size of the input (n) setting the number of operating conditions (i) as a fixed number. However, at its worst, when the number of operating conditions equals the input size, the time complexity of the feed-forward step is proportional to $O(n^2)$.

A.3. Multi-layer perceptron back-propagation pass

This section presents the calculations for the time complexity of the training algorithm of the proposed MLP. Typically, in back-propagation the following matrix operation is performed:

$$D_{hn}^{(out)} = Z_{hn} - Y_{hn} \tag{A.3}$$

In such approach, the matrix D is computed as the difference between the predicted output (Z) and the expected output (Y). The expected output is the result that the algorithm tries to approximate. The delta matrix D is fed backwards through the network when this classical approach is followed. The approach proposed in this paper is however different. The network aims to minimize the following equation instead of the previous one (where $\odot$ denotes element-wise multiplication):

$$D_{hn}^{(out)} = Z_{hn} \odot Z_{hn} \tag{A.4}$$

In other words, there is no concept of expected output: the network supervises itself automatically. The matrix that contains the error signals between the input layer and the output layer is computed as follows:

$$D_{hn}^{(inp)} = \varphi\prime(S_{hn}) \odot D_{hn}^{(out)} \tag{A.5}$$

The last step is to compute the weight updates:

$$\delta W^{MLP} = -\eta D_{hn}^{((inp))X_{in}^T} \tag{A.6}$$

In this last equation, the η denotes the learning rate. From Eq. A.6 it follows that the time complexity of the backward propagation algorithm is $O(n * i * h)$. The final time complexity of the algorithm involves a multiplication by the number of iterations I_{MLP} (epochs). The time complexity of the back-propagation is therefore

Table B.4
Runtime performance of the Self Organizing Map (SOM) in seconds for dataset 2 and 4. Mean values and the corresponding confidence intervals of 15 replicates are presented. The confidence intervals are set at 95%.

	Self-Organizing Map (SOM)	K-Means (Baseline)
Dataset 2	1.70s 95% CI [1.67s, 1.74s]	0.016s 95% CI [0.015s, 0.017s]
Dataset 4	1.71s 95% CI [1.70s, 1.72s]	0.017s 95% CI [0.015s, 0.018s]

Table B.5
Runtime performance of the Multi-Layer Perceptron (MLP) in seconds for 3 sensors. Mean values and the corresponding confidence intervals of 15 replicates are presented. The confidence intervals are set at 95%.

Dataset	Sensor	MLP	MLP	Standard Rule
2	phi	14.37s 95% CI [13.50s, 15.25s]	0.48s 95% CI [0.46s, 0.49s]	0.018s 95% CI [0.018s, 0.019s]
4	phi	15.55s 95% CI [15.40s, 15.70s]	0.53s 95% CI [0.55s, 0.57s]	0.019s 95% CI [0.019s, 0.020s]
2	T30	4.23s 95% CI [4.19s, 4.28s]	0.48s 95% CI [0.46s, 0.50s]	0.018s 95% CI [0.018s, 0.019s]
4	T30	4.94s 95% CI [4.77s, 5.11s]	0.54s 95% CI [0.53s, 0.55s]	0.020s 95% CI [0.019s, 0.020s]
2	T50	1.51s 95% CI [1.40s, 1.61s]	0.47s 95% CI [0.45s, 0.50s]	0.017s 95% CI [0.017s, 0.018s]
4	T50	4.94s 95% CI [4.77s, 5.11s]	0.54s 95% CI [0.53s, 0.55s]	0.020s 95% CI [0.019s, 0.020s]

$O(I_{MLP} * n * i * h)$. Setting the epochs and number of operating conditions fixed, the algorithm is linear with the size of the input (n). However, at its worst, when the number of operating conditions and epochs equals the input size, the time complexity of the feed-forward step is proportional to $O(n^3)$.

Appendix B. Runtime

In addition to calculating the time complexity, the exact running time of the algorithms was measured in a sequence of experiments. All experimental evaluations were performed using a desktop computer equipped with an Intel Core i7-10875H CPU, 32 GB memory, and a NVIDIA GeForce RTX 2060 (6 GB) GPU card.

The SOM implementation was evaluated for dataset 2 and dataset 4. Based on the operating conditions indicators (p indicators for both datasets), the SOM clustered the operating regimes in each dataset. The performance results are shown in Table B.4. It can be seen from Table B.4 that the performance of the proposed SOM solution is significantly worse than that of the baseline model, the K-means clustering. The long running time of SOM architectures is a well-known limitation [43]. The increase in running time is compensated by the fact that the proposed SOM adaptation discovers automatically the number of operating conditions, and also importantly, the proposed SOM has mechanisms in place to validate and ensure that the discovered number of clusters (i.e. number of operating conditions) is appropriate.

For simplicity, the formal evaluation of the exact running time of the MLP was performed only on 3 sensors (phi, T30 and T50). Table B.5 presents the computational performance of the MLP training and execution processes considering dataset 2 and 4 for the 3 selected sensors (phi, T30 and T50). As it is shown in Table B.5, the training time (column 3) is significantly higher than the running time of the MLP (column 4) for all three considered sensors for both datasets 2 and 4. This is an expected result for neural networks: the backward pass repeated over a given number of epochs usually takes longer than the feed-forward pass.

In order to reduce the MLP computational costs, it has been shown in Section 6 that the network can be trained on a smaller set of data. Instead of using the entire dataset, the data of a single equipment can be used to train the network. This has been shown in Section 6 to yield slightly worse but overall acceptable results (see Table 3 of the paper). In terms of computational time and in our experiments, this approach was shown to reduce by approximately ten times the training time.

It was also noticed that the more complex a sensor was, such as phi, the more epochs would be needed to train the MLP network of the sensor. This ultimately leads to higher computational costs. By complex sensor it is meant a sensor not showing monotonic behavior or trendability patterns [67]. As an example consider the third column of Table B.5. As shown in the Table, the training time of the MLP for phi, a sensor with low trendability, was consistently higher than the training time of the MPL for other sensors such as T30 or T50, which score higher in terms of trendability. This can be seen in the third column of Table B.5. To minimize this kind of effects, it is hypothesized that by improving the monotonicity and trendability of the sensors, using for example filtering and fusion techniques, the computational complexity of the proposed MLP algorithm could be lowered.

The complexity of the sensor does not seem to affect the running time of the MLP as can be seen in the fourth column of Table B.5. The different selected sensors (phi, T30 and T50) had similar running times for dataset 2. The same situation was noticed for dataset 4. This is possibly explained by the fact that once the network is trained, it is only necessary to perform a feedforward step to baseline the data. When training the network, the number of epochs constrains the process and is dependent on the sensor complexity.

In regards to execution time, from one dataset to the other there were however significant differences. This can be explained by the fact that there are more sample points in dataset 4 than in dataset 2. Since the time complexity of the feed-forward step of the MLP algorithm has been shown to be linear with respect to the sample size, this result was expected.

References

[1] M.J. Hasan, M.M. Islam, J.-M. Kim, Acoustic spectral imaging and transfer learning for reliable bearing fault diagnosis under variable speed conditions, Measurement 138 (2019) 620–631.
[2] R. Dhumale, S. Lokhande, Neural network fault diagnosis of voltage source inverter under variable load conditions at different frequencies, Measurement 91 (2016) 565–575.
[3] J.H. Travert, Flight regime and maneuver recognition for complex maneuvers..
[4] H. Sohn, Effects of environmental and operational variability on structural health monitoring, Philos. Trans. Roy. Soc. A: Math. Phys. Eng. Sci. 365 (1851) (2007) 539–560.
[5] M.L. Baptista, E.M. Henriques, K. Goebel, More effective prognostics with elbow point detection and deep learning, Mech. Syst. Sig. Process. 146 (2021) 106987.
[6] J.A. Hartigan, M.A. Wong, Algorithm AS 136: A K-means clustering algorithm, J. Roy. Stat. Soc. Ser. C (Appl. Stat.) 28 (1) (1979) 100–108.
[7] M. Pecht, Prognostics and health management of electronics, Encyclopedia of structural health monitoring..
[8] J. Sun, H. Zuo, W. Wang, M.G. Pecht, Prognostics uncertainty reduction by fusing on-line monitoring data based on a state-space-based degradation model, Mech. Syst. Sig. Process. 45 (2) (2014) 396–407.
[9] E. Figueiredo, G. Park, C.R. Farrar, K. Worden, J. Figueiras, Machine learning algorithms for damage detection under operational and environmental variability, Struct. Health Monit. 10 (6) (2011) 559–572.
[10] S.K.U. Rehman, Z. Ibrahim, S.A. Memon, M. Jameel, Nondestructive test methods for concrete bridges: A review, Constr. Build. Mater. 107 (2016) 58–86.
[11] S.M. Rezvanizaniani, Z. Liu, Y. Chen, J. Lee, Review and recent advances in battery health monitoring and prognostics technologies for electric vehicle (ev) safety and mobility, J. Power Sources 256 (2014) 110–124.
[12] D.P. Dennies, The organization of a failure investigation, Pract. Fail. Anal. 2 (3) (2002) 11–16.
[13] G. Niu, D. Anand, M. Pecht, Prognostics and health management for energetic material systems, in: 2010 Prognostics and System Health Management Conference, IEEE, 2010, pp. 1–7..
[14] A. Heng, S. Zhang, A.C. Tan, J. Mathew, Rotating machinery prognostics: State of the art, challenges and opportunities, Mech. Syst. Sig. Process. 23 (3) (2009) 724–739.
[15] H. Sohn, M. Dzwonczyk, E.G. Straser, A.S. Kiremidjian, K.H. Law, T. Meng, An experimental study of temperature effect on modal parameters of the alamosa canyon bridge, Earthquake Eng. Struct. Dyn. 28 (8) (1999) 879–897.

[16] B. Peeters, G. De Roeck, One-year monitoring of the z24-bridge: Environmental effects versus damage events, Earthquake Eng. Struct. Dyn. 30 (2) (2001) 149–171.
[17] L. Lennart, System identification: theory for the user, Upper Saddle River, NJ, PTR Prentice Hall, 1999, pp. 1–14.
[18] P. Moser, B. Moaveni, Environmental effects on the identified natural frequencies of the dowling hall footbridge, Mech. Syst. Sig. Process. 25 (7) (2011) 2336–2357.
[19] B. Moaveni, I. Behmanesh, Effects of changing ambient temperature on finite element model updating of the dowling hall footbridge, Eng. Struct. 43 (2012) 58–68.
[20] A.-M. Yan, G. Kerschen, P. De Boe, J.-C. Golinval, Structural damage diagnosis under varying environmental conditions—part i: a linear analysis, Mech. Syst. Sig. Process. 19 (4) (2005) 847–864.
[21] A.-M. Yan, G. Kerschen, P. De Boe, J.-C. Golinval, Structural damage diagnosis under varying environmental conditions—part ii: local pca for non-linear cases, Mech. Syst. Sig. Process. 19 (4) (2005) 865–880.
[22] A. Deraemaeker, E. Reynders, G. De Roeck, J. Kullaa, Vibration-based structural health monitoring using output-only measurements under changing environment, Mech. Syst. Sig. Process. 22 (1) (2008) 34–56.
[23] Y. Ni, X. Hua, K. Fan, J. Ko, Correlating modal properties with temperature using long-term monitoring data and support vector machine technique, Eng. Struct. 27 (12) (2005) 1762–1773.
[24] Y. Xia, B. Chen, S. Weng, Y.-Q. Ni, Y.-L. Xu, Temperature effect on vibration properties of civil structures: a literature review and case studies, J. Civ. Struct. Health Monit. 2 (1) (2012) 29–46.
[25] C.R. Farrar, N.A. Lieven, Damage prognosis: the future of structural health monitoring, Philos. Trans. Roy. Soc. A: Math. Phys. Eng. Sci. 365 (1851) (2007) 623–632.
[26] N. Gebraeel, J. Pan, Prognostic degradation models for computing and updating residual life distributions in a time-varying environment, IEEE Trans. Reliab. 57 (4) (2008) 539–550.
[27] W. Peng, Y.-F. Li, Y.-J. Yang, J. Mi, H.-Z. Huang, Bayesian degradation analysis with inverse gaussian process models under time-varying degradation rates, IEEE Trans. Reliab. 66 (1) (2017) 84–96.
[28] T. Wang, Trajectory similarity based prediction for remaining useful life estimation, Ph.D. thesis, University of Cincinnati, 2010..
[29] G. McLachlan, D. Peel, Mixtures of factor analyzers, in: Proceedings of the 17th International Conference on Machine Learning, 2000.
[30] J.C. Bezdek, R. Ehrlich, W. Full, FCM: The fuzzy C-means clustering algorithm, Comput. Geosci. 10 (2–3) (1984) 191–203.
[31] M. Natrella, E-Handbook of Statistical Methods, NIST/Sematech, 2010..
[32] A. Saxena, K. Goebel, D. Simon, N. Eklund, Damage propagation modeling for aircraft engine run-to-failure simulation, in: Proceedings of the International Conference on Prognostics and Health Management, IEEE, 2008, pp. 1–9.
[33] S. Al-Dahidi, F. Di Maio, P. Baraldi, E. Zio, Remaining useful life estimation in heterogeneous fleets working under variable operating conditions, Reliab. Eng. Syst. Saf. 156 (2016) 109–124.
[34] E. Zio, F. Di Maio, M. Stasi, A data-driven approach for predicting failure scenarios in nuclear systems, Ann. Nucl. Energy 37 (4) (2010) 482–491.
[35] M. Rigamonti, P. Baraldi, E. Zio, I. Roychoudhury, K. Goebel, S. Pol, Echo state network for the remaining useful life prediction of a turbofan engine, in: Proceedings of the European Conference of the Prognostics and Health Management Society, 2016, pp. 255–270.
[36] M. Rigamonti, P. Baraldi, E. Zio, D. Astigarraga, A. Galarza, Particle filter-based prognostics for an electrolytic capacitor working in variable operating conditions, IEEE Trans. Power Electron. 31 (2) (2015) 1567–1575.
[37] E. Ramasso, Investigating computational geometry for failure prognostics, Int. J. Progn. Health Manage. 5 (1) (2014) 005.
[38] O. Bektas, A. Alfudail, J.A. Jones, Reducing dimensionality of multi-regime data for failure prognostics, J. Fail. Anal. Prev. 17 (6) (2017) 1268–1275.
[39] D. Siegel, J. Lee, An auto-associative residual processing and K-means clustering approach for anemometer health assessment, Int. J. Progn. Health Manage. 2 (2011) 1–12.
[40] E. Hale, L. Fusina, M. Brower, Correction factors for NRG# 40 anemometers potentially affected by dry friction whip: Characterization, analysis, and validation, Wind Energy 15 (3) (2012) 489–502.
[41] L. Bian, N. Gebraeel, J.P. Kharoufeh, Degradation modeling for real-time estimation of residual lifetimes in dynamic environments, Iie Trans. 47 (5) (2015) 471–486.
[42] N. Li, N. Gebraeel, Y. Lei, L. Bian, X. Si, Remaining useful life prediction of machinery under time-varying operating conditions based on a two-factor state-space model, Reliab. Eng. Syst. Saf. 186 (2019) 88–100.
[43] T. Kohonen, Self-organized formation of topologically correct feature maps, Biol. Cybern. 43 (1) (1982) 59–69.
[44] D. Miljković, Brief review of self-organizing maps, in: Proceedings of the 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), IEEE, 2017, pp. 1061–1066.
[45] A. Flexer, On the use of self-organizing maps for clustering and visualization, Intell. Data Anal. 5 (5) (2001) 373–384.
[46] B.D. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, 2007.
[47] T. Kohonen, Things you haven't heard about the self-organizing map, in: Proceedings of the IEEE International Conference on Neural Networks, IEEE, 1993, pp. 1147–1156..
[48] M. Ceccarelli, A. Petrosino, R. Vaccaro, Competitive neural networks on message-passing parallel computers, Concurr. Pract. Exp. 5 (6) (1993) 449–470.
[49] J. Vesanto, E. Alhoniemi, Clustering of the self-organizing map, IEEE Trans. Neural Netw. 11 (3) (2000) 586–600.
[50] J.H. Ward Jr, Hierarchical grouping to optimize an objective function, J. Am. Stat. Assoc. 58 (301) (1963) 236–244.
[51] K.H. Rosen, K. Krithivasan, Discrete Mathematics and its Applications: with Combinatorics and Graph Theory, Tata McGraw-Hill Education, 2012.
[52] L. Hamel, C.W. Brown, Improved interpretability of the unified distance matrix with connected components, in: Proceedings of the International Conference on Data Mining (DMIN), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing, 2011, p. 1..
[53] J. Hoshen, On the application of the enhanced Hoshen-Kopelman algorithm for image analysis, Pattern Recogn. Lett. 19 (7) (1998) 575–584.
[54] J. Hoshen, R. Kopelman, Percolation and cluster distribution. cluster multiple labeling technique and critical concentration algorithm, Phys. Rev. B 14 (8) (1976) 3438.
[55] G. Vettigli, Minisom: Minimalistic and numpy based implementation of the self organizing maps, 2013..
[56] D.P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114..
[57] F.P. Casale, A. Dalca, L. Saglietti, J. Listgarten, N. Fusi, Gaussian process prior variational autoencoders, in: Advances in Neural Information Processing Systems, 2018, pp. 10369–10380..
[58] Y. Bengio et al., Learning deep architectures for AI, Foundations and Trends in Machine, Learning 2 (1) (2009) 1–127.
[59] P.J. Werbos, Backpropagation through time: What it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560.
[60] R. Pascanu, T. Mikolov, Y. Bengio, Understanding the exploding gradient problem, CoRR, abs/1211.5063 2..
[61] S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprint arXiv:1609.04747..
[62] H. Richter, Advanced Control of Turbofan Engines, Springer Science & Business Media, 2011.
[63] D.K. Frederick, J.A. DeCastro, J.S. Litt, User's Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS), Tech. rep., NASA TM-2007-215026, Glenn Research Center, Cleveland, Ohio, 2007..
[64] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of KDD, vol. 96, 1996, pp. 226–231..
[65] R.J. Campello, D. Moulavi, A. Zimek, J. Sander, Hierarchical density estimates for data clustering, visualization, and outlier detection, ACM Trans. Knowl. Discov. Data (TKDD) 10 (1) (2015) 5.
[66] Y. Cheng, Mean shift, mode seeking, and clustering, IEEE Trans. Pattern Anal. Mach. Intell. 17 (8) (1995) 790–799.
[67] J. Coble, J.W. Hines, Identifying optimal prognostic parameters from data: a genetic algorithms approach, in: Annual conference of the prognostics and health management society, vol. 27, 2009..
[68] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on Machine Learning, 2015, pp. 448–456.
[69] M.L. Baptista, Regime detection project, 2021. URLhttps://github.com/marcialbaptista/RegimeDetection [Online; accessed 28-March-2021]..
[70] M.L. Baptista, Regime normalization project, 2021.https://github.com/marcialbaptista/RegimeNormalization [Online; accessed 28-March-2021]..
[71] J. Melka, J.-J. Mariage, Efficient implementation of self-organizing map for sparse input data, IJCCI (2017) 54–63.

Marcia L. Baptista (BS and MSc. in Informatics and Computer Engineering. Instituto Superior Tecnico, Lisbon, Portugal, September 2008) holds a PhD from the Engineering Design and Advanced Manufacturing (EDAM) program under the umbrella of MIT Portugal. Her research focuses on the development of prognostics techniques for aeronautics equipment. Her research interests include datadriven modeling, prognostics, and deep learning.

Elsa Maria Pires Henriques has a doctorate in Mechanical Engineering and is associated professor at Instituto Superior Tecnico in the University of Lisbon. She is responsible for the "Engineering Design and Advanced Manufacturing (LTI/EDAM)" post-graduation. During the last fifteen years, she has participated and/or coordinated several national and European R&D projects in collaboration with different industrial sectors, from tooling to automotive and aeronautics, mainly related to manufacturing, life cycle based decisions and management of complex design processes. She has a large number of scientific and technical publications in national and international conferences and journals. She was a national delegate in the 7th Framework Programme of the EU.

Kai Goebel is a Principal Scientist in the System Sciences Lab at Palo Alto Research Center (PARC). His interest is broadly in predictive maintenance and systems health management for a broad spectrum of cyber-physical systems in the manufacturing, energy, and transportation sectors. Prior to joining PARC, Dr. Goebel worked at NASA Ames Research Center and General Electric Corporate Research & Development center. At NASA, he founded and directed the Prognostics Center of Excellence which pioneered our understanding of the fundamental aspects of prognostics. He holds 18 patents and has published more than 375 papers, including a book on Prognostics. He received his Ph.D. in Mechanical Engineering from UC Berkeley in 1990 Dr. Goebel was an adjunct professor at Rensselaer Polytechnic Institute and is now adjunct professor at Lulea Technical University. He is a member of ASME, IEEE, SAE, AAAI; co-founder of the Prognostics and Health Management Society; and associate editor of the International Journal of PHM.