TrajecNets: Online Failure Evolution Analysis in 2D Space

Nauman Shahid[1], and Anarta Ghosh[2]

United Technologies Research Center, Penrose Wharf, Penrose Business Center, Cork, Ireland
[1]*shahidn@utrc.utc.com*
[2]*ghosha@utrc.utc.com*

ABSTRACT

We propose a novel Recurrent Neural Network (RNN) based autoencoder for embedding the run-to-failure time series sensor data in a 2D feature space. The embedding, extracted from the network, is in the form of a smooth ***trajec***tory, which represents the temporal evolution of data from healthy to failure states, hence the name *TrajecNets*. The visualizable 2D trajectory can be used directly for highly intuitive and interpretable health monitoring, which can in turn be used for Remaining Useful Life (RUL) estimation task, without compromising the performance. We also propose a novel unsupervised failure prediction methodology which uses the 2D trajectories and health curve of the time series to compute evolving failure mode probabilities. Together, the visualizable 2D trajectories and the interpretable failure mode probabilities, health curve and RUL are envisaged to provide system and maintenance engineers, insight into failure dynamics. Experiments on NASA CMAPSS Turbofan benchmark dataset show promising results on degradation tracking, health monitoring, failure prediction and RUL estimation tasks.

1. INTRODUCTION

Prognostics for complex systems and components, such as those in an aircraft, has attracted significant interest of industrial and academic research community in the last few years. In the industrial field, a widely accepted definition of 'prognostics' is *the ability to predict the Remaining Useful Life (RUL) of a component after a fault has occurred* (Aizpurua & Catterson, 2015). While significant importance has been given in the academic research to improve the RUL estimates with complex algorithms, little effort is being done to explain these algorithms in terms of how or why they achieve a certain level of performance. This is in contrast to the future industrial requirement of *interpretable* safety critical PHM system (Elattar, Elminir, & Riad, 2016). Satisfying the aforementioned definition of prognostics, most PHM literature focuses on estimating RUL via *physics-based* or *data-driven* approaches. Our focus in this work is on data-driven approaches.

Data-driven approaches perform RUL prediction from the operational run-to-failure raw time series data, collected from the sensors mounted on the component or system under consideration. There are two types of data-driven approaches in the literature; direct and indirect.

The *direct* approach relies on training a neural network to learn the RUL directly from the run-to-failure time series data. Many methods which use Bidirectional Long-Short Term Memory Networks (Bi-LSTM) (A. Zhang et al., 2018), (J. Wang, Wen, Yang, & Liu, 2018), a combination of Convolutional Neural Network (CNN) and LSTM (Li, Li, & He, 2019) or a Deep Belief Network (DBN) (C. Zhang, Pin, K. Qin, & Chen Tan, 2016) have been proposed in this context. These methods have superior performance but are uninterpretable due to the direct mapping of the time series to a life estimate. Furthermore, many of the aforementioned approaches use a piecewise linear function of RUL for each time series to train the deep network. It is not possible to define this function for a broad range of systems and application scenarios.

The *indirect* approach first maps the time series data into a one-dimensional *health index HI or health curve* (ranging from 1 to 0), which decreases monotonically and proportionally to the time series degradation (Mosallam, Medjaher, & Zerhouni, 2016), (Ramasso, 2014), (Mosallam, Medjaher, & Zerhouni, 2015). This task is also known as health monitoring. Similar to the direct approaches, deep learning methods such as Restricted Boltzmann Machines, DBNs and CNNs have been used very frequently (Zhao et al., 2019), (Akintayo, Lore, Sarkar, & Sarkar, 2016), (Reddy, Venugopalan, & Giering, 2016) for health monitoring purpose. More recently RNN based approaches have become even more popular (Zhao, Wang, Yan, & Mao, 2016), (Zhao, Yan, Wang, & Mao, 2017), (Wu, Yuan, Dong, Lin, & Liu, 2018), (Ellefsen, Bjørlykhaug, Æsøy, Ushakov, & Zhang, 2019) for monitoring health degradation. In particular, (Malhotra et al., 2016) and (Gugulothu et al., 2017) propose to learn health index from the time se-

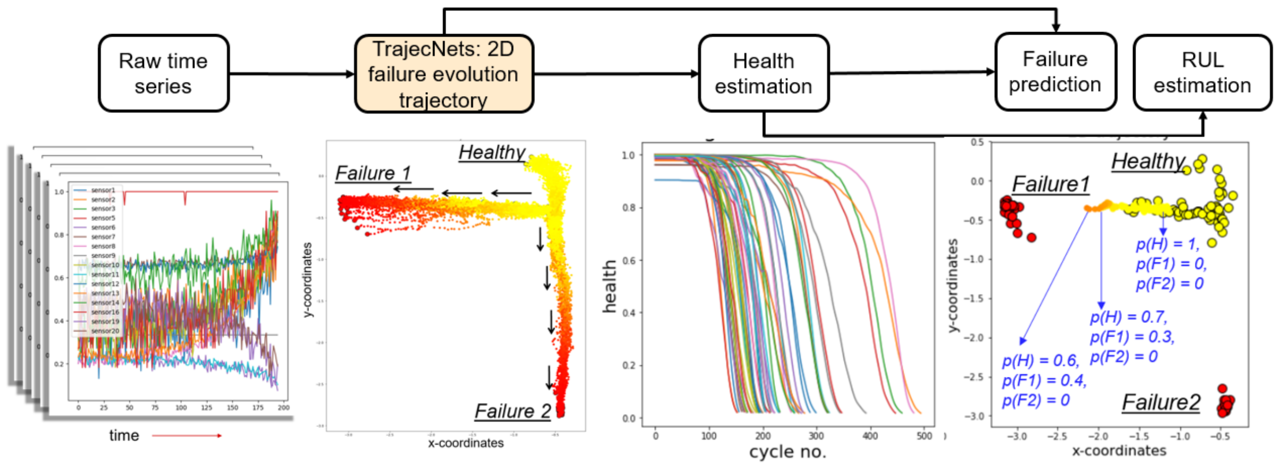

Figure 1. The proposed TrajecNets based PHM pipeline

ries reconstruction error computed with an autoencoder or distance from healthy embedding in low dimensional space. Once the health index is computed, RUL is estimated as a weighted average of RULs of matching HI curves (T. Wang, 2010) of all the time series in the training dataset.

In this paper, we are interested in the indirect approaches, which estimate RUL via health monitoring. Such approaches are easy to interpret and do not require the specification of piecewise linear function of RUL or other thresholds which cannot be estimated without apriori knowledge on the data domain.

Gaps in the state of the art health monitoring approaches: Although the one-dimensional health curve is very commonly used for health degradation monitoring and RUL estimation, it provides very limited information about the complex dynamics of the degradation, such as the connection between different failure modes. Therefore, the health curve can only be used to *quantify* the component degradation without providing any further insight into the complex failure dynamics.

Our proposal: Due to the mere quantitative nature of the health curve and as that deep neural networks are capable of extracting information richer than just one dimensional health curve, we believe that there is a need for an intermediate block in the health analysis and RUL computation pipeline. This block should be capable of summarizing the neural network model information in a low-dimensional space, which can be leveraged for maintenance decision making. Motivated by the fact that the notion of interpretability in prognostics is strongly connected to visualization (Phillips, 2012), we propose this intermediate block to be *a visualization of degradation of raw time series from healthy to failure classes, in the form of a smooth trajectory in a 2D space*. We propose to extract this trajectory from the time series data via a set of novel RNN autoencoder architectures, viz., *TrajecNets*.

1.1. Contributions

Following are the main technical contributions of our work, which is summarized in Figure 1.

1. A novel RNN autoencoder architecture, TrajecNets, which embeds the operational run-to-failure time series data in a 2D metric space in the form of a smooth trajectory, capturing the evolution of system from healthy to failure state.
2. A novel and interpretable health index computation method from 2D trajectories, which is a fusion of “local-health”, i.e., the health of the system (e.g., engines) with respect to its nascence and “global-health” - health with respect to other similar systems.
3. An unsupervised methodology to compute failure probability using the aforementioned health index.
4. A fusion of local and global health curve matching strategies for RUL estimation by using a more informative distance metric.

1.2. Overview of TrajecNets

The basic unit of TrajecNets is a stacked RNN (Figure 2), which gradually reduces the dimension to 2D (last layer) and ensures temporal smoothness of 2D time series trajectory (for details c.f., Section 3). The default architecture of TrajecNets is that of an **A**uto**E**ncoder (TrajecNets-AE), where both encoder and decoder are stacked RNNs. The autoencoder can be configured to a **Super**vised setting (TrajecNets-Super) - when the failure labels are available, or a **Siamese** Network (TrajecNets-Siamese) when the number of samples per failure class is small. Irrespective of the network configuration, the health index computation and RUL estimation methods remain same.

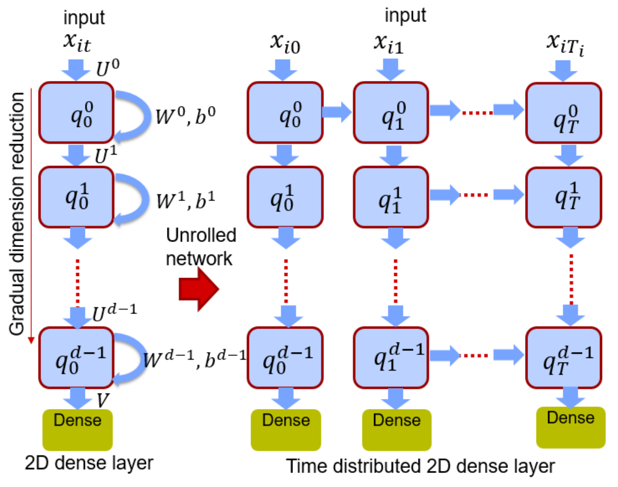

Figure 2. The base unit of TrajecNets is a stacked RNN of d layers with gradually reducing number of hidden units in each subsequent layer. We avoid skip and inter-layer connections.

1.3. Advantages of the Proposed Approach

Previous efforts to incorporate visualization into prognostics (Zhao et al., 2017), have been mostly limited to the use of 2D t-SNE (Maaten & Hinton, 2008). However, a 2D t-SNE does not define a metric space as it is trained separately on the output of a neural network (NN). For this reason it can *only* be used as a visualization tool. In our case, the 2D trajectory is learned in an end-to-end network from the raw time series. Hence, there is a one-to-one mapping between time series and the learned visualization. Thus, unlike other state-of-the-art models, our visualization space is also the inference space. This results in a major advantage of the proposed framework, i.e., the failure prediction, health curve computation and as well as RUL estimation can be done directly from the 2D visualizable trajectory. Hence, these computations become much more simpler and interpretable than the ones done on a high dimensional time series.

Interestingly, we show in Section 8.4 that even with the simplification of the health computation process, our method performs comparable to the state-of-the-art methods which use health curves for RUL estimation. This implies that the health curve computed using our method is as informative as the one computed directly from the high dimensional time series. The emphasis, therefore, is not only on the *visualization*, but also on a plethora of intuitive inference algorithms that become realizable due to the 2D mapping, without compromising the RUL estimation performance.

2. PROBLEM STATEMENT

Consider a set of multivariate time series samples $\mathbf{X}_i \in \Re^{T_i \times k}$, $\forall i \in \{0, \cdots, N-1\}$, where N is the total number of aircraft systems (e.g., engines) belonging to a fleet of aircrafts, thus the total number of samples available for training the model. Each $\mathbf{X}_i$ is a matrix that represents the sensor data collected from k sensors of the i^{th} aircraft system at multiple time instants $\{t_0, t_1, \cdots, t_{T_i}\}$, such that:

$$\mathbf{X}_i = \begin{bmatrix} x_{i0t_0} & x_{i0t_1} & \cdots & x_{i0t_{T_i}} \\ x_{i1t_0} & x_{i1t_1} & \cdots & x_{i1t_{T_i}} \\ \vdots & & \ddots & \vdots \\ x_{ikt_0} & x_{ikt_1} & \cdots & x_{ikt_{T_i}} \end{bmatrix}^\top \in \Re^{T_i \times k},$$

where x_{imt_j} is the scalar corresponding to the m^{th} sensor reading at the time stamp or cycle number t_j of the i^{th} system and T_i is the length of each $\mathbf{X}_i$. Note that T_i can vary significantly across the samples $\mathbf{X}_i$. Different aircraft systems may operate under different schedules, therefore, the timelines of a pair of aircraft systems $(i, \hat{i})$ may be completely disjoint. For this reason t_j does not refer to an actual timestamp, but to a *cycle number*. To simplify the notation we use $t_j = t$, where t is an integer representing the cycle number and T_i is the last cycle number of the system. Thus, $t \in \{0, \cdots, T_i\}$. Each row $\mathbf{x}_{it}$ in $\mathbf{X}_i$ corresponds to a snapshot of k sensors at the t^{th} flight of the i^{th} aircraft. All the important notations and acronyms used in this work are defined in Table 1 and Table 2.

To this end, we assume, without a loss of generality (Saxena, Goebel, Simon, & Eklund, 2008) that each time series $\mathbf{X}_i$ starts healthy, develops a fault at some unknown cycle t and then degrades to one of the F failures types represented by the F-dimensional categorical variable $\mathbf{y}_i \in \{0,1\}^F$. The failure class $\mathbf{y}_i$ is assumed to be unknown for each of the training time series $\mathbf{X}_i$.

Our goal in this paper is to address the following: a) Failure evolution analysis in 2D space in Section 3, b) Health index computation in Section 4, c) Failure prediction in Section 5 and d) RUL estimation in Section 6. In the following sections, we study each of these and propose the corresponding methodologies.

3. TRAJECNETS: FAILURE EVOLUTION ANALYSIS IN 2D SPACE

Our goal is to transform each $\mathbf{X}_i \in \Re^{T_i \times k}$ into a 2D trajectory $\mathbf{Z}_i \in \Re^{T_i \times 2}$, such that $\mathbf{Z}_i$ represents a smooth evolution of the time series from healthy to failure class in a 2D feature space. More formally, we want to learn a transformation $\mathbf{Z}_i = \bar{F}(\mathbf{X}_i)$. We choose to learn $\bar{F}$ via stacked-RNNs (Figure 2), i.e., a sequence of d RNNs, stacked on the top of each other. We use l to index the l^{th} RNN layer and $\mathbf{q}_t^l$ to represent the l^{th} layer hidden unit at time t.

3.1. Key characteristics of TrajecNets

There are three key characteristics of the proposed TrajecNets architecture:

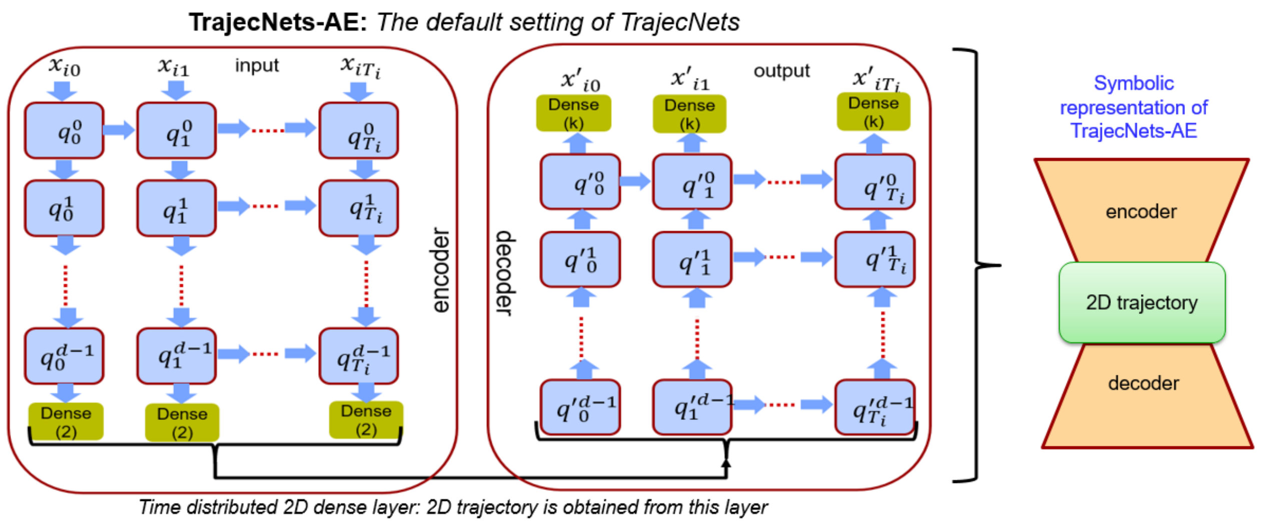
TrajecNets-AE: The default setting of TrajecNets
input
encoder
decoder
output
Dense (2)
Dense (k)
Symbolic representation of TrajecNets-AE
encoder
2D trajectory
decoder
Time distributed 2D dense layer: 2D trajectory is obtained from this layer

Figure 3. TrajecNets-AE: The default autoencoder architecture of TrajecNets.

notation	description
T_i	number of cycles of i^{th} system
k	number of sensors under monitoring
N	total number of systems / samples for training
$\mathbf{X}_i \in \Re^{T_i \times k}$	run-to-failure time series matrix of i^{th} system
$t \in \{0, \cdots, T_i\}$	integer representing the cycle number
$\mathbf{x}_{it} \in \Re^k$	t^{th} temporal sample / row of $\mathbf{X}_i$
F	total number of failure modes
$\mathbf{y}_i \in \{0,1\}^F$	categorical failure label vector for i^{th} system
$\mathbf{Z}_i \in \Re^{T_i \times 2}$	2D trajectory matrix for the time series $\mathbf{X}_i$
$\mathbf{z}_{it} \in \Re^2$	2D embedding of $\mathbf{x}_{it} \in \Re^k$
$\mathbf{z}_{i(t-w)}$	2D embedding at the $t-w$ time instant
$\mathbf{Z}_{i(t-w:t)}$	2D embedding matrix of $t-w$ time instants
F	function representation for stacked RNN
d	number of layers in stacked RNN
$\mathbf{q}_{it}^l$	t^{th} hidden unit at the l^{th} layer of stacked RNN
$\lvert\mathbf{q}_{it}^l\rvert$	size of $\mathbf{q}_{it}^l$
$\mathbf{Q}_i^l$	matrix of all temporal hidden units at l^{th} layer
$\mathbf{W}^l, \mathbf{U}^l, \mathbf{b}^l$	weights of the l^{th} RNN layer in encoder
$\mathbf{V}$	weight matrix of 2D time distributed dense layer
$\hat{\mathbf{W}}^l, \hat{\mathbf{U}}^l, \hat{\mathbf{b}}^l$	weights of the l^{th} RNN layer in decoder
$\mathbf{h}_i \in [0,1]^{T_i}$	health curve for $\mathbf{X}_i$
h_{it}	the scalar health at the t^{th} temporal sample $\mathbf{x}_{it}$
$\mathbf{h}_{Li} \in [0,1]^{T_i}$	local health curve of $\mathbf{X}_i$
$\mathbf{h}_{Gi} \in [0,1]^{T_i}$	global health curve of $\mathbf{X}_i$
$\mathbf{d}_{Li} \in \Re^{T_i}$	distances of $\mathbf{z}_{it}$ from $\mathbf{z}_{i0}$
$\mathbf{Z}_{healthy}$	set of the first points $\mathbf{z}_{i0}$ of all trajectories
$KNN(.,.)$	function to select k_{nn} nearest neighbors from $\mathbf{Z}_{healthy}$
$\mathbf{d}_{Gi} \in \Re^{T_i}$	mean distances of $\mathbf{z}_{it}$ from $KNN(\mathbf{Z}_{healthy}, \mathbf{z}_{it})$
d_{Lmini}, σ_{Li}	local normalization constants for health $\mathbf{h}_{Li}$
d_{Gmini}, σ_{Gi}	global normalization constants for health $\mathbf{h}_{Gi}$
$\hat{\mathbf{y}}_i$	failure labels estimated by Kmeans
$\mathbf{P}_i \in [0,1]^{T_i \times (F+1)}$	failure probability matrix of $\mathbf{X}_i$
$\mathbf{p}_{it} \in [0,1]^{F+1}$	failure probability vector of $\mathbf{x}_{it}$
$\mathbf{p}_{it}[j]$	j^{th} element / failure probability of $\mathbf{p}_{it}$
M	number of cycles prior to failure
$\bar{r}_i$	RUL of system i, via global health curve matching approach
$\hat{r}_i$	RUL of system i, via local health curve matching approach
r_i	fused RUL of the i^{th} system

Table 1. Table of notations. We refer to the i^{th} aircraft system or engine as i^{th} system.

Acronym	Full name
PHM	Prognostics and Health Management
2D	2 dimensional
RUL	Remaining Useful Life
CNN	Convolutional Neural Network
LSTM	Long Short Term Memory Network
Bi-LSTM	Bidirectional Long Short Term Memory Network
DBN	Deep Belief Network
RNN	Recurrent neural Network
MLP	Multilayer Perceptron
HI	Health Index
KNN	K-nearest neighbors
AE	Auto-Encoder
RBF	Radial Basis Function
DTW	Dynamic Time Warping
MSE	Mean Squared Error
MAE	Mean Average Error
FPR	False Positive Rate
FNR	False Negative Rate

Table 2. Table of Acronyms.

1. **Autoencoder structure**: The default configuration of TrajectNets is that of an autoencoder where both the encoder and decoder are stacked-RNNs. This is essential as the trajectories are extracted by reducing the dimensionality of the time series gradually along the subsequent encoder layers until the bottleneck layer, hence the name *TrajecNets*.
2. **Inter-layer ladder structure**: The number of hidden units decrease gradually in the subsequent recurrent layers, i.e., $|\mathbf{q}_t^{l-1}| > |\mathbf{q}_t^l|, \forall l \in \{0, \cdots, d-1\}$.
3. **Projection in 2D space**: The final RNN layer is followed by a time distributed dense layer which projects the output of last RNN to 2D space. The 2D degradation trajectory $\mathbf{Z}_i$ is obtained from this layer. A keen reader might argue that projecting a complex time series to a 2 dimensional space can result in the loss of information in time series. We point out that TrajecNets embed every k-

dimensional temporal sample $\mathbf{x}_{it} \in \Re^k$ of $\mathbf{X}_i \in \Re^{T_i \times k}$ into 2D space rather than the whole time series. In other words, to reconstruct a time series $\mathbf{X}_i \in \Re^{T_i \times k}$, one needs the full trajectory $\mathbf{Z}_i \in \Re^{T_i \times k}$, instead of the just the last trajectory point $\mathbf{z}_{iT_i} \in \Re^2$.

3.1.1. Comparison with other recurrent networks

Note that the proposed stacked RNN (Figure 2) is different from the state-of-the-art deep or stacked RNNs. As compared to the architectures proposed in (Hermans & Schrauwen, 2013), *TrajecNets*: a) gradually reduce the dimension (number of hidden units) along the depth of stacked RNN to provide 2D trajectories from the last layer and b) optionally avoid skip connections. As compared to the Gated Feedback RNNs (Chung, Gulcehre, Cho, & Bengio, 2015) and Recurrent Ladder Networks (Ilin et al., 2017), *TrajecNets* does not share connections between hidden units from different layers.

Gradual dimension reduction: A typical sequence-to- sequence autoencoder consists of only one RNN layer in the encoder and decoder. Thus, it embeds a k dimensional sequence directly to a lower dimensional space. The embedding is typically extracted from the last temporal hidden unit of RNN. On the contrary, our network consists of multiple stacked layers of RNN in the encoder and decoder. This particular architecture is essential for our setting because we extract the entire temporal trajectory of the time series instead of just the final temporal embedding such that the temporal degradation signature is dominant over high frequency noise in the time series. In other words, our aim is to extract globally smooth features from the time series instead of the local features (dominated by noise). State-of-the-art neural networks, such as CNNs (LeCun, Bengio, & Hinton, 2015), ensure this property with a deep network that reduces the size of spatial features gradually along the depth. In our case the depth corresponds to the various RNN layers stacked on the top of each other, with reducing number of hidden units. The smoothness in 2D space is ensured by the horizontal and vertical flow of gradients during backpropagation.

Absence of skip and intra-layer connections The purpose of skip connections in a typical deep neural network is to (He, Zhang, Ren, & Sun, 2016) (Szegedy, Ioffe, Vanhoucke, & Alemi, 2017) propagate information from earlier layers of the network to the later layers to help in the convergence. In an autoencoder, such connections between similar layers in the encoder and decoder ensure the reconstruction property. However, we avoid such connections because we did not encounter any convergence or reconstruction issues during the training of our network. Depending on k and the depth of the network, such connections might be needed.

3.2. TrajecNets-AE: The Autoencoder

The default configuration of TrajecNets is TrajecNets-AE (Figure 3), an autoencoder built from stacked RNN of Figure 2. In the encoder part, the stacked-RNN has d layers, where l^{th} layer has a recurrent weight matrix $\mathbf{W}^l$, bias $\mathbf{b}^l$ and the inter-layer weight matrix $\mathbf{U}^l$ between layers $l-1$ and l. The encoder takes as input the time series $\mathbf{X}_i \in \Re^{T_i \times k}$ and transforms it first into a $|\mathbf{q}^{d-1}|$ dimensional embedding $\mathbf{Q}_i^{d-1} \in \Re^{T_i \times |\mathbf{q}^{d-1}|}$ extracted from the last layer of stacked-RNN. The output of the t^{th} hidden unit at the l^{th} stacked-RNN layer $\mathbf{q}_{it}^l \in \mathbf{Q}_i^l$ is given as:

$$\mathbf{q}_{it}^l = \tanh\left(\mathbf{W}^l \mathbf{q}_{it-1}^l + \mathbf{U}^l \mathbf{q}_{it}^{l-1} + \mathbf{b}^l\right),$$

where $\mathbf{q}_{it}^{l-1} = \mathbf{x}_{it}$ for $l = 0$, i.e., the input layer. This embedding is then transformed into a 2D trajectory $\mathbf{Z}_i \in \Re^{T_i \times k}$ via the 2D time distributed dense layer at the bottleneck of the autoencoder. Each $\mathbf{z}_{it} \in \mathbf{Z}_i$ is given as $\mathbf{z}_{it} = \mathbf{V}\mathbf{q}_{it}^{d-1}$, where $\mathbf{V}$ is the weight matrix of the 2D time distributed dense layer.

The decoder then reconstructs the time series $\mathbf{X}_i$ from the trajectories $\mathbf{Z}_i$ by gradually increasing the dimension via d stacked RNNs, where each layer has a recurrent weight matrix $\hat{\mathbf{W}}^l$, bias $\hat{\mathbf{b}}^l$ and the inter-layer weight matrix $\hat{\mathbf{U}}^l$. The reconstructed time series $\hat{\mathbf{X}}_i$ is extracted from the time distributed dense layer of k units. The autoencoder is only trained using mean squared error between actual time series $\mathbf{X}_i$ and reconstructed time series $\hat{\mathbf{X}}_i$, as the loss function, i.e.,

$$L_{AE} = \sum_{i=0}^{N} \sum_{t=0}^{T_i} \|\mathbf{x}_{it} - \hat{\mathbf{x}}_{it}\|_2^2.$$

For a test time series $\mathbf{X}_{testi}$ of the i^{th} aircraft system, one only needs the encoder $\bar{F}$ to compute the 2D trajectory, i.e., $\mathbf{Z}_{testi} = \bar{F}(\mathbf{X}_{testi})$.

3.3. Other configurations of TrajecNets

Depending on the availability of failure class labels $\mathbf{y}_i$ and the total number of samples available for training each class, we propose two other architectures: 1) **TrajecNets-Siamese**: A siamese network which is built from the autoencoder for the case of small number of samples N and 2) **TrajecNets-Super**: A supervised TrajecNets-AE for the case when failure class labels $\mathbf{y}_i$ are available for training. Nevertheless, the most challenging case is when the failure labels $\mathbf{y}_i$ are not available or reliable. Therefore, we focus mainly on the unsupervised setting (TrajecNets-AE) in this paper and briefly discuss the other two derived architectures for the sake of completeness. We aim to present detailed results on TrajecNets-Siamese and TrajecNets-Super in our future work.

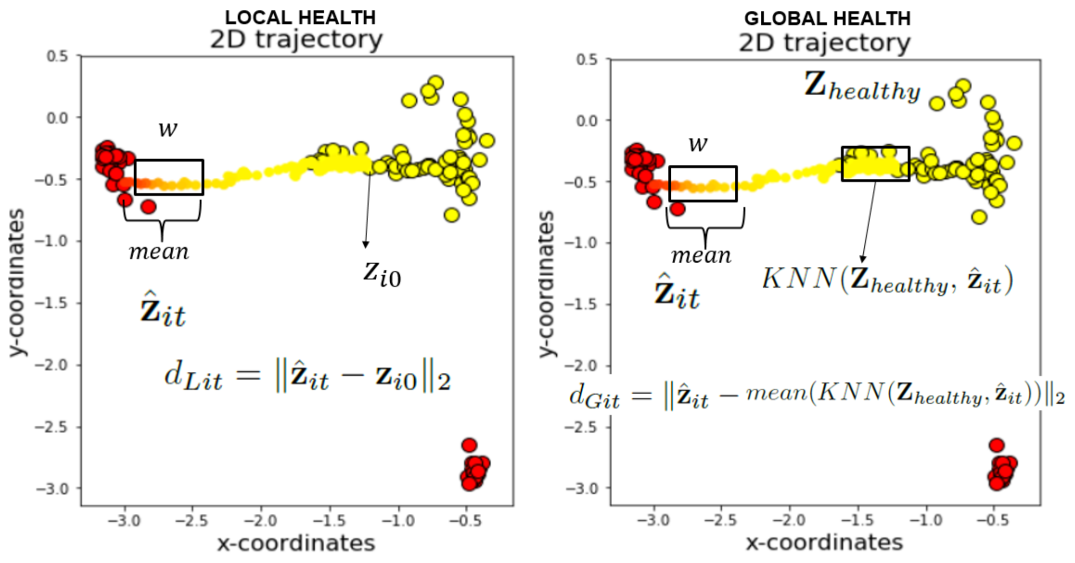

Figure 4. Procedure to compute the distances for local health on the left and global health on the right plot. The local health is computed from the streaming distances d_{Lit} of $\hat{\mathbf{z}}_{it}$ from the first trajectory point $\mathbf{z}_{i0}$. Let $\mathbf{Z}_{healthy}$ be the set of all the first trajectory points $\mathbf{z}_{i0}$ (yellow circles with black borders). The global health is computed from the streaming distances d_{Git} of $\hat{\mathbf{z}}_{it}$ from the mean of k_{nn}-nearest neighbors in the $\mathbf{Z}_{healthy}$ set.

4. HEALTH COMPUTATION: TRAJECTORIES TO HEALTH

Given the trajectories $\mathbf{Z}_i$ for time series $\mathbf{X}_i$, the next goal is to compute a health curve $\mathbf{h}_i \in [0,1]^{T_i}$, indicative of the degradation in the time series. We propose to use a fusion of "local health", which is the health of the aircraft system w.r.t its nascence and "global health" which is the health w.r.t to other systems. The global health takes into account the initial wear of the system w.r.t the most healthy ones. In both cases, the health is computed from the 2D trajectories directly, instead of the raw time series.

In an online *test* setting, it is important to make predictions for evolving time series. Thus, for every new temporal sample $\mathbf{x}_{it} \in \Re^k$ in the time series $\mathbf{X}_i$, our framework first updates the trajectory $\mathbf{z}_{it} \in \Re^2$ and then computes the scalar health $h_{it} \in [0,1]$. Ideally, the health should drop to zero when the system fails in future. This requires appropriate normalization of health index h_{it}, especially if a distance based approach is adopted. Therefore, we learn the appropriate normalization constants during the training stage.

4.1. Training Stage for Health Computation

4.1.1. Step 1: Local Health Computation

For a complete trajectory $\mathbf{Z}_i \in \Re^{T_i \times 2}$ of time series $\mathbf{X}_i \in \Re^{T_i \times k}$, the local health curve $\mathbf{h}_{Li} \in [0,1]^{T_i}$, where L in subscript represents 'local', is computed as a distance of the evolving trajectory $\mathbf{z}_{it} \in \Re^2$ from the first trajectory point $\mathbf{z}_{i0}$. In order to avoid the effect of noise in the trajectory, we use

$$\hat{\mathbf{z}}_{it} = mean([\mathbf{z}_{i(t-w)}, \mathbf{z}_{i(t-w+1)}, \cdots, \mathbf{z}_{it}]),$$

i.e., the mean of 2D embedding in a causal window of size w.

Local distances: Let

$$\mathbf{d}_{Li} = [d_{Li0}, d_{Li1}, \cdots, d_{iT_i}]$$

be a vector of Euclidean distances computed such that:

$$d_{Lit} = \|\hat{\mathbf{z}}_{it} - \mathbf{z}_{i0}\|_2,$$

i.e., $\mathbf{d}_{Li}$ is the set of distances of t^{th} trajectory point from the first one $\forall\ t \in \{0, \cdots, T_i\}$. Please refer to the left plot in Figure 4 for a pictorial view of distance calculation.

Local distances to health: The distance vector $\mathbf{d}_{Li}$ is converted into a health vector $\mathbf{h}_{Li}$ by applying a Radial Basis Function (RBF) based normalization to each scalar distance d_{Lit} in the vector $\mathbf{d}_{Li}$ as following:

$$h_{Lit} = \exp\left(\frac{-(d_{Lit} - d_{Lmini})^2}{\sigma_{Li}}\right), \quad \forall\ t \in \{0, \cdots, T_i\}, \tag{1}$$

where $d_{Lmini} = \min \mathbf{d}_{Li}$. Let $d_{Lmaxi} = \max \mathbf{d}_{Li}$, then σ_{Li} is set as:

$$\sigma_{Li} = -\frac{(d_{Lmaxi} - d_{Lmini})^2}{2}\left[\frac{1}{\log_{10}\epsilon} + \frac{1}{\log_{10}(\epsilon+\delta)}\right].$$

Normalization with the above σ_{Li} ensures that the health at the start, i.e., for the 0th time instant $h_{Li0} = 1$ and at time of failure $h_{LiT_i} \in [\epsilon, \epsilon+\delta]$, irrespective of the length of the trajectory. This is important to eliminate the effect of variable length sequences. In our experiments we use $\epsilon = \delta = 0.01$. Finally (and optionally) a moving average filter of size n is applied to all the curves $\mathbf{h}_{Li}$ to eliminate the effect of trajectory noise.

Saving the normalization parameters: It is important to point out that the normalization parameters d_{Lmini} and σ_{Li} are saved for each of the training time series $\mathbf{X}_i$ to be used later for computing the health of a test time series.

4.1.2. Step 2: Global Health Computation

Initial wear and tear in the aircraft system is inherent in nature (Saxena, Goebel, et al., 2008), which cannot be justified by "local health" only, as all health curves start from 1. In order to adjust the local health curve $\mathbf{h}_{Li}$ for this purpose, we propose a global health curve $\mathbf{h}_{Gi}$ computation for each $\mathbf{X}_i$, where G represents global. In contrast to local health $\mathbf{h}_{Li}$, the global health $\mathbf{h}_{Gi}$ is computed as a function of the distance of evolving trajectory points $\mathbf{z}_{it}$ from the mean of k_{nn}-nearest neighbor healthy trajectory windows.

Nearest neighbor indexing: More formally, let $\mathbf{z}_{i0} \in \Re^2$ be the first healthy trajectory point for each $\mathbf{X}_i$ and let $\mathbf{Z}_{healthy}$ be the set of all the $\mathbf{z}_{i0}$. First a k_{nn}-nearest neighbor indexing of the set of healthy trajectories points $\mathbf{Z}_{healthy}$ is performed.

Global distances: Then, the global distance $\mathbf{d}_{Gi}$ is computed as:

$$d_{Git} = \|\hat{\mathbf{z}}_{it} - mean(KNN(\mathbf{Z}_{healthy}, \hat{\mathbf{z}}_{it}))\|_2, \qquad (2)$$

where $KNN(.,.)$ is a function that returns the indices of the k_{nn}-nearest neighbors of $\hat{\mathbf{z}}_{it}$ from the set of healthy trajectory points $\mathbf{Z}_{healthy}$. Please refer to the right plot in Figure 4 for a pictorial view of distance calculation.

Global distances to health: Global health $\mathbf{h}_{Gi}$ is then computed by following the same normalization steps as in local health (see eq. 1). Similarly to the local health, the normalization parameters d_{Gmini} and σ_{Gi} are saved for later use in the testing stage.

4.1.3. Step 3: Fusion of Local and Global Healths

Finally, the local health $\mathbf{h}_{Li}$ and global health $\mathbf{h}_{Gi}$ are fused according to one of the following two strategies: $\mathbf{h}_i = \mathbf{h}_{Gi} \circ \mathbf{h}_{Li}$, where $\circ$ denotes the element wise (Hadamard) product or $\mathbf{h}_i = h_{Gi0}.\mathbf{h}_{Li}$, where h_{Gi0} is the first point in the health curve $\mathbf{h}_{Gi}$. The first scheme applies an adaptive wear factor computed in the form of global health $\mathbf{h}_{Gi}$ to each point in the local health curve. The second scheme assumes a fixed initial wear factor h_{Gi0}, that is the first point of global health curve and multiplies it to the entire local health curve.

4.2. Testing Stage for Health Computation

For a new partial test time series $\mathbf{X}_{testi}$, first the partial trajectory $\mathbf{Z}_{testi}$ is computed and then local and global healths are computed by following these steps:

1. Compute the local distances $\mathbf{d}_{testLi}$ using the same procedure as defined for $\mathbf{d}_{Li}$ in step 1 in Section 4.1.
2. For each of the $\hat{\mathbf{z}}_{testit} =$
$$mean([\mathbf{z}_{testi(t-w)}, \mathbf{z}_{testi(t-w+1)}, \cdots, \mathbf{z}_{testit}])$$
retrieve the k_{nn} nearest neighbors using the nearest neighbor indexing function $KNN(\mathbf{Z}_{healthy}, \hat{\mathbf{z}}_{testit})$ and then retrieve the saved normalization parameters σ_{Lj}, d_{Lminj} for these nearest neighbors, where $j \in \{0, \cdots, k_{nn}\}$.
3. Apply the RBF based normalization for each of the scalar distances in $\mathbf{d}_{testLi}$ by using
$$\sigma_{testLi} = mean(\sigma_{Lj}, \forall j \in \{0, \cdots, k_{nn}\})$$
$$d_{testLmini} = mean(d_{minj}, \forall j \in \{0, \cdots, k_{nn}\}).$$
Thus, the test normalization parameters are the mean of the normalization parameters of k_{nn}-nearest neighbors determined via the pre-computed KNN indexing.
4. Repeat the steps 1 to 3, this time for computing the global distances $\mathbf{d}_{testGi}$ and global health $\mathbf{h}_{testGi}$.
5. Fuse the local and global healths: $\mathbf{h}_{testi} = \mathbf{h}_{testGi} \circ \mathbf{h}_{testLi}$ or $\mathbf{h}_{testi} = h_{testG0i}.\mathbf{h}_{testLi}$, where $h_{testG0i}$ is the first point of global health $\mathbf{h}_{testGi}$.

5. UNSUPERVISED FAILURE PREDICTION

Irrespective of the network configuration of TrajecNets, we propose a generic and unsupervised method to compute evolving failure probabilities for the trajectories $\mathbf{Z}_i$. One can postulate that this step is redundant for Supervised TrajecNets, since the supervised networks already provide failure class probabilities as the classification output. We argue that failure class probabilities obtained from the supervised networks correspond to the whole trajectory. Our concern here is not only to extract one probability for the full trajectory, but to compute evolving probabilities for every temporal point $\mathbf{z}_{it} \in \mathbf{Z}_i$. Furthermore, the probabilities obtained as an output of the supervised model correspond to the model trained on hard labels (binary or categorical). During the test time, using this probability will result in a sudden evolution of trajectory probability from healthy to failure. Thus, it cannot be used in the field for continuous monitoring purpose. Since, the health $\mathbf{h}_i$ quantifies the evolution of the trajectory $\mathbf{Z}_i$, which in turn represents the time series degradation at every temporal point, we propose to use it to compute a failure probability.

5.1. Training Stage for Failure Prediction

Given the health curves $\mathbf{h}_i \in [0,1]^{T_i}$, number of failure classes F and the trajectories $\mathbf{Z}_i \in \Re^{T_i \times 2}$, we propose to train a Multilayer Perceptron (MLP) that can predict failure probabilities $\mathbf{p}_{it} \in [0,1]^{F+1}$ for each temporal embedding $\mathbf{z}_{it} \in \mathbf{Z}_i$. The probability vector is size $F+1$, where F is the number of

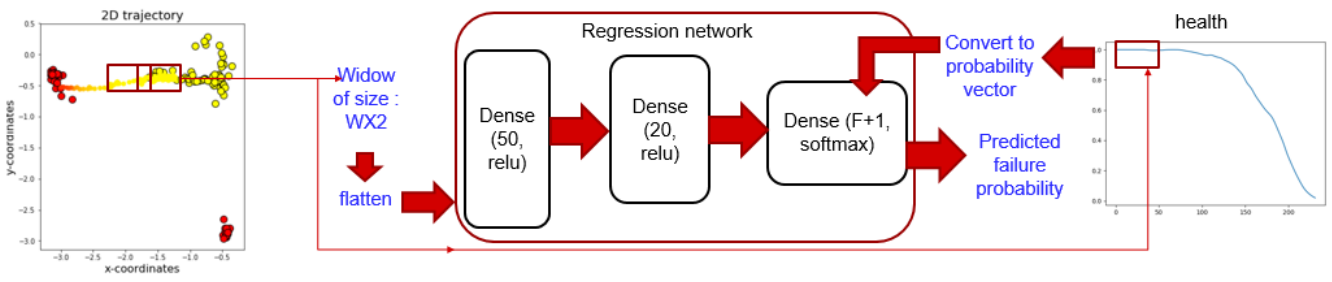

Figure 5. Training procedure of regression MLP which predicts failure class probabilities from input windowed trajectories.

failure classes and an extra class is assigned for healthy state. We formulate this problem as a regression, where the input is a partial trajectory $\mathbf{Z}_{i(t-w:t)} \in \Re^{w\times 2}$ comprising of the latest w time instants and the output is $\mathbf{p}_{it} \in [0,1]^{F+1}$. Unfortunately, the failure probabilities $\mathbf{p}_{it}$ are not known in advance for training. Therefore, we propose to extract them for each $\mathbf{Z}_i$ from the health curve $\mathbf{h}_i$.

5.1.1. Step 1: Probability Generation for Training

As in (Saxena, Goebel, et al., 2008) we assume that all time series $\mathbf{X}_i$ start from a healthy state and degrade to one of the F failure types. For every $\mathbf{Z}_i \in \Re^{T_i\times 2}$ in the training set, we first extract the last window of size w, i.e., $\mathbf{Z}_{i(t-w:t)} \in \Re^{w\times 2}$ and compute the temporal mean $\hat{\mathbf{z}}_{it} \in \Re^2$ as a representative feature of this window. Then, we perform Kmeans clustering (F clusters) on the features $\hat{\mathbf{z}}_{it}$ $\forall i \in \{0,\cdots,N-1\}$ in the training set. Let $\hat{\mathbf{y}}_i \in \{0,1\}^F$ be the Kmeans computed label for $\hat{\mathbf{z}}_{it}$. We compute the probability matrix $\mathbf{P}_i \in [0,1]^{T_i\times(F+1)}$ for the complete timeline $\{0,\cdots,T_i\}$ of the i^{th} system from the health curve $\mathbf{h}_i$ as following:

$$\begin{aligned}
&\mathbf{P}_i[0:T_i,0] = \mathbf{h}_i,\\
&\mathbf{P}_i[0:T_i,j+1] = \mathbf{1}-\mathbf{h}_i \qquad \text{if} \quad \hat{\mathbf{y}}_i[j]=1,\\
&\mathbf{P}_i[0:T_i,m+1] = 0 \qquad \forall\ m\neq j.
\end{aligned} \tag{3}$$

The above equation states that we assign $\mathbf{h}_i$ as a probability for healthy class (0^{th} class), i.e., $\mathbf{h}_i$ is placed in the 0^{th} column of $\mathbf{P}_i$. Since the categorical labels $\hat{\mathbf{y}}_i$ represent the Kmeans result, $\hat{\mathbf{y}}_i[j] = 1$ when the Kmeans algorithm assigns the j^{th} label (j^{th} failure) to $\hat{\mathbf{z}}_{it}$. Therefore, we assign $\mathbf{1}-\mathbf{h}_i$ as the probability of the j^{th} failure class. This corresponds to the $(j+1)^{st}$ column of $\mathbf{P}_i$. All other failure classes (columns of $\mathbf{P}_i$) are assigned a 0 probability. Note that eq. (3) defines a valid probability measure for each of the rows $\mathbf{p}_{it}$ of $\mathbf{P}_i$.

5.1.2. Step 2: Training the Regression MLP

Finally we train a regression Multi Layer Perceptron (MLP) with 2 hidden layers which takes as input the windowed trajectories $\mathbf{Z}_{i(t-w:t)} \in \Re^{w\times 2}$, flattened to a vector of length $2w$ and outputs a probability vector $\mathbf{p}_{it} \in [0,1]^{(F+1)}$. For training purpose, we use windowed trajectories, which are generated from the complete life-time of all training trajectories $\mathbf{Z}_i$, with a step size of 1. Thus, a sample at a given time instant is a part of many training windows $\mathbf{Z}_{i(t-w:t)}$. The total number of $(\mathbf{Z}_{i(t-w:t)}, \mathbf{p}_{it})$ pairs used for training is $\sum_i(T_i-w)$. We use relu activation in the hidden layers and a softmax activation in the output layer. The network is trained with 'mean-squared-error' loss function. A summary of this approach is shown in Figure 5.

5.2. Testing Stage for Failure Prediction

During the test stage, for a partial test sequence $\mathbf{X}_{testi}$, the most recent partial trajectory of size w, i.e., $\mathbf{Z}_{testi(t-w:t)} \in \Re^{w\times 2}$ is input into the network and the failure probabilities $\mathbf{p}_{it}$ are obtained.

When is a failure declared? *A failure of j^{th} type is declared at the t^{th} time instance when* $\max(\mathbf{p}_{it}[1:F+1]) > \mathbf{p}_{it}[0]$. In simpler words, we declare the failure of type j to be predicted M cycles before it occurs, where $M = T_i - t$ and t is the time cycle number at which the j^{th} class failure probability is the highest among all failure probabilities and it also exceeds the healthy class probability $\mathbf{p}_{it}[0]$.

5.2.1. Special case: one failure class

In the case, where there is only one failure class, i.e., all the time series start from healthy and gradually degrade to only one failure type, the failure probability is univariate and should evolve from 0 to 1, which essentially reduces the failure class prediction problem into a failure prediction only. In this case the regression MLP need not be used. In fact, training the regression MLP and then using it for failure probability prediction is equivalent to setting the failure probability as $\mathbf{p}_i = \mathbf{1}-\mathbf{h}_i$. A failure will be predicted in this case when $p_{it} > 0.5$ or $h_{it} < 0.5$. We will cover this case in detail in the experimental section of this work.

6. RUL ESTIMATION

The training health curves $\mathbf{h}_i$ computed in Section 4.1 are used for the estimation of RUL r_i for test time series $\mathbf{X}_{testi}$. More formally, the RUL r_i for a partial test time series $\mathbf{X}_{testi}$

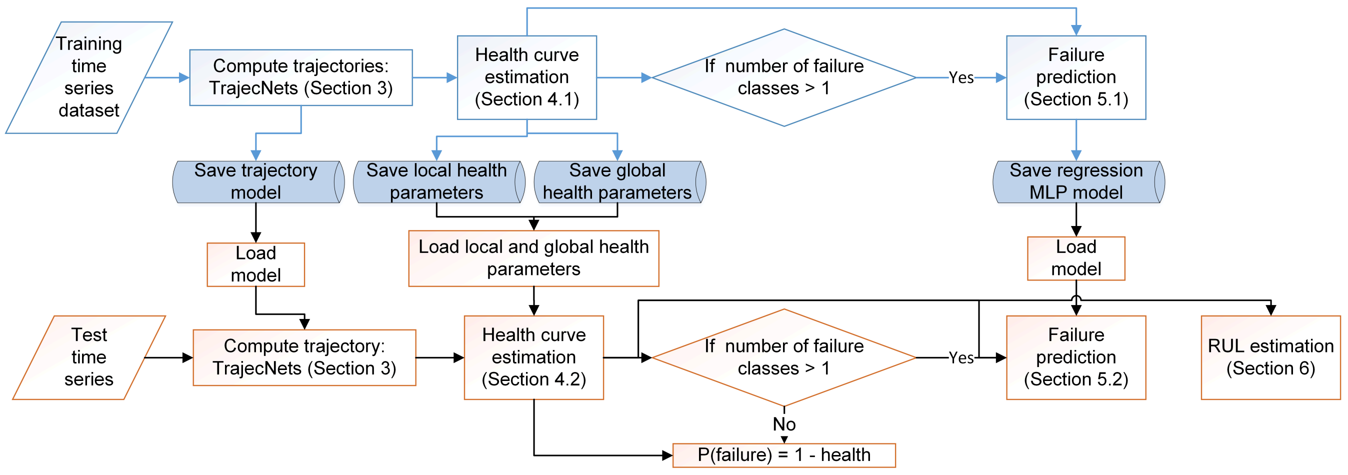
Training time series dataset
Compute trajectories: TrajecNets (Section 3)
Health curve estimation (Section 4.1)
If number of failure classes > 1
Yes
Failure prediction (Section 5.1)
Save trajectory model
Save local health parameters
Save global health parameters
Save regression MLP model
Load model
Load local and global health parameters
Load model
Test time series
Compute trajectory: TrajecNets (Section 3)
Health curve estimation (Section 4.2)
If number of failure classes > 1
Yes
Failure prediction (Section 5.2)
RUL estimation (Section 6)
No
P(failure) = 1 - health

Figure 6. A summary of the training and testing stages of our proposed framework. The upper part of the figure shows the training phase and the lower part shows the testing phase.

is defined as the number of cycles r_i the system will run before it reaches the end of its life. Our approach to computing RUL is a fusion of:

1. State-of-the-art health curve matching approach proposed in (T. Wang, 2010), where we replace the Euclidean distance metric with Dynamic Time Warping (DTW) distance.
2. Our proposed local health curve matching approach based on DTW distance.

Step 1: Global health curve matching approach: The health curve matching approach proposed in (T. Wang, 2010) is used very commonly for RUL estimation, for example in (Malhotra et al., 2016) and (Gugulothu et al., 2017). For a new test health curve $\mathbf{h}_{testi}$, this approach computes RUL as a weighted average of the RULs for all the training time series whose health curves match the test health curve under different time lags. Due to space constraints, we do not describe the details of this approach. Euclidean distance is used to compare different health curves under time lags in this approach. However, Euclidean distance is not well known to compare curves, therefore, we propose to replace this with DTW distance. Let $\hat{r}_i$ be the RUL computed using this method. We call this method as the 'global matching approach' because it takes into account the entire test health curve observed until the most recent time instant.

Step 2: Local health curve matching approach: Next, we propose to compute another RUL $\bar{r}_i$ by comparing the most recent window w of the test health curve $\mathbf{h}_{testi(t-w:t)}$ with all the possible windows of training health curves. We call this as the 'local health curve' matching approach. For this purpose, first we divide each of the health curves $\mathbf{h}_i$ in the training set into smaller curves of length w each using a sliding window method with a delay of 1 sample. For a given training health curve $\mathbf{h}_i$ of length T_i, this results in the generation of $T_i - w$ smaller health curves $\mathbf{h}_i^j$ of length w each.

Next, we use DTW distance to compare $\mathbf{h}_{testi(t-w:t)}$ with all the training windows $\mathbf{h}_i^j, \forall j \in \sum_i (T_i - w)$. All the RULs are sorted in the ascending order of their corresponding DTW distances and then the four step RUL fusion approach proposed in Section III.B of (T. Wang, Yu, Siegel, & Lee, 2008) is applied for $\bar{r}_i$ computation.

Step 3: Fusion of RULs: Finally the RULs $\hat{r}_i$ and $\bar{r}_i$ computed from the above two steps are fused as $r_i = \min(\hat{r}_i, \bar{r}_i)$.

An overall summary of Section 3, in terms of various training and testing stages of our proposed framework is presented in Figure 6.

7. EXPERIMENTAL DETAILS

We evaluate our proposed framework on the publicly available C-MAPSS Turbofan Engine dataset (Saxena, Goebel, et al., 2008) for the following tasks:

1. failure prediction
   (a) correct failure class prediction.
   (b) early prediction in terms of the number of cycles before the end-of-life *if* the algorithm predicts it.
2. RUL estimation

Since a major emphasis in our proposed framework is on the visualization of 2D trajectories, we present detailed results on the visualization of trajectories, health and evolving failure probabilities. We use Keras library (Chollet et al., 2015) with Tensorflow backend for implementing TrajecNets.

7.1. Datasets

We use simulated turbofan engine datasets FD001 and FD003 from the NASA repository (Saxena, Goebel, et al., 2008).

Each dataset comprises of separate train and test sets, where each sample has time series readings for 26 sensors. Each engine starts normally, develops a fault at some point and then degrades to exactly one failure. The location of the fault is unknown. The samples in FD001 degrade to one failure only. The samples in FD003 dataset, degrade to one of the two types of failures. All the temporal samples in each time series of both datasets belong to only one operating condition. The train samples are complete in their lifetime, i.e., from the start of the usage to the end of life, whereas the test samples are pruned some time prior to the failure. Furthermore, each engine has a different initial degree of wear which results in the initial health of each engine to be different from 1. We use MinMax normalization on each of the sensors in the datasets FD001 and FD003. We use FD00X-train, where X is the dataset number to refer to the train set of each dataset and FD00X-test to represent the test part.

7.2. Train-Test Split

For some of our experiments, as explained in Section 8.3, we will divide the training sets FD00X-train further into train and test sets. We refer to these two subsets of FD00X-train as FD00X-train-train and FD00X-train-test. More specifically, we split the 100 samples in FD00X-train datasets to 64 training FD00X-train-train and 36 test samples FD00X-train-test. It is worth mentioning here that all the proposed models for all the tasks are trained only on FD00X-train-train. A summary of all the datasets is given in Table. 3.

7.3. Architectures and Algorithm Parameters

TrajecNets-AE: We train one TrajecNets-AE on each of the FD00X-train-train datasets using a 2-layer stacked-RNN configuration with 100 and 20 hidden units, $tanh$ activations, a recurrent dropout of 0.1 with a batch size of 16, for 50 epochs. We use Adadelta with a gradient clipping norm of 5.0 as an optimizer with a learning rate of 0.01 and 'mean squared error' as a loss function. The 20 dimensional embedding is then mapped to 2D trajectory using the bottleneck time distributed dense layer of size 2 with a linear activation. The embedded trajectory is then decoded to recover the time series. As mentioned earlier, TrajecNets-AE is trained only via reconstruction loss on the time series.

Health estimation: For the health estimation method, we use a window size $w = 30$, $\epsilon = \delta = 0.1$ and $k_{nn} = 20$ for all the experiments in this work. The choice of k_{nn} was made on a hit and trial basis. However, any value between 10 and 20 was found equally suitable for health curve computation, as no significant changes were reflected in the RUL estimation performance. For the fusion of local and global health, we use the following scheme $\mathbf{h}_i = h_{G0i}.\mathbf{h}_{Li}$, which offsets the entire local health curve with an initial wear factor h_{G0i}.

Unsupervised failure prediction: Our unsupervised failure prediction relies on training a regression MLP (Section 5) with two hidden layers. We use the first dense layer of size 50 and the second dense layer of size 20, both with a 'relu' activation and a final dense layer to predict probabilities with a 'softmax' activation. We use Adadelta with a gradient clipping norm of 5.0 as an optimizer with a learning rate of 0.01. The input to this network is flattened windowed trajectories of size $2w$, where $w = 30$. The network is trained using 'mean squared error' as a loss function.

8. RESULTS

In this section we study in detail the different types of experiments for various datasets, the ground truth generation for the tasks, the evaluation metrics, experimental settings and detailed results. Throughout this section, we will follow the flow of our proposed framework in Figure 6 and present the results in the same order. A summary of experimental evaluation on the various datasets is presented in Table 4. Asterisk(*) indicates that the results are reported only for the subset of the data for which the algorithm predicts a failure. We will explain this in more detail in Section 8.3.2.

8.1. Visualization of Failure Classes & Trajectories

2D failure evolution trajectories for training time series: Our first goal is to visualize the 2D trajectories $\mathbf{Z}_i$ learned by TrajecNets-AE on FD001-train-train and FD003-train-train datasets. Figure 7 shows the 2D trajectories for all the samples in these datasets. For each of the two datasets, the 2D trajectories start from healthy state which is represented by yellow color and then transition to a failure state gradually, which is represented by red color. The color transition from yellow to red also represents the health of the time series from 1 to 0. Note that FD001 has only one failure class and FD003 has two failure classes. The distinction between healthy and various failure classes is quite clear. It is worth mentioning here that the TrajecNets-AE has no prior information on the number of failure classes, nevertheless, it successfully reveals the class structure in the datasets, due to the autoencoder architecture.

2D failure evolution trajectories for test time series: During the testing stage, we extract the 2D trajectories $\mathbf{Z}_i$ for each of the test time series in FD00X-train-test and FD00X-test. As FD00X-train-test is a subset of FD00X-train, the time series run from the beginning to the end of the life cycle. However, the ones in FD00X-test are only partial. We visualize a sample trajectory from FD001-train-test in the top-most plot of Figure 9 and another one from FD001-test in the second plot. Each of the two plots consists of 3 subplots, of which the first one shows the raw time series and the second one shows the trajectory evolving from healthy (yellow) to failure state (red), along with the failure probabilities. The yellow circles with black border represent the healthy points for each

Dataset	default train-test split	our train-test split	number of samples	number of failures	operating modes
FD001	FD001-train	FD001-train-train	64	1	1
		FD001-train-test	36	1	1
	FD001-test	-	100	1	1
FD003	FD003-train	FD003-train-train	64	2	1
		FD003-train-test	36	2	1
	FD003-test	-	100	2	1

Table 3. Details of the turbofan engine dataset used for experiments in this work.

Dataset	trajectory visualization (Section 3)	failure class prediction (training) (Section 5.1)	failure class prediction (testing) (Section 5.2)	failure prediction (cycles) (training) (Section 5.1)	failure prediction (cycles) (testing) (Section 5.2)	RUL (Section 6)
FD001-train-train	✓	✓	-	✓	-	-
FD003-train-train	✓	✓	-	✓	-	-
FD001-train-test	✓	-	✓	-	✓	-
FD003-train-test	✓	-	✓	-	✓	-
FD001-test	✓	-	-	-	*✓	✓
FD003-test	✓	-	*✓	-	*✓	✓

Table 4. A summary of experimental evaluation on the various datasets. Asterisk(*) indicates that the results are reported only for the subset of the data for which the algorithm predicts a failure (Section 5.2).

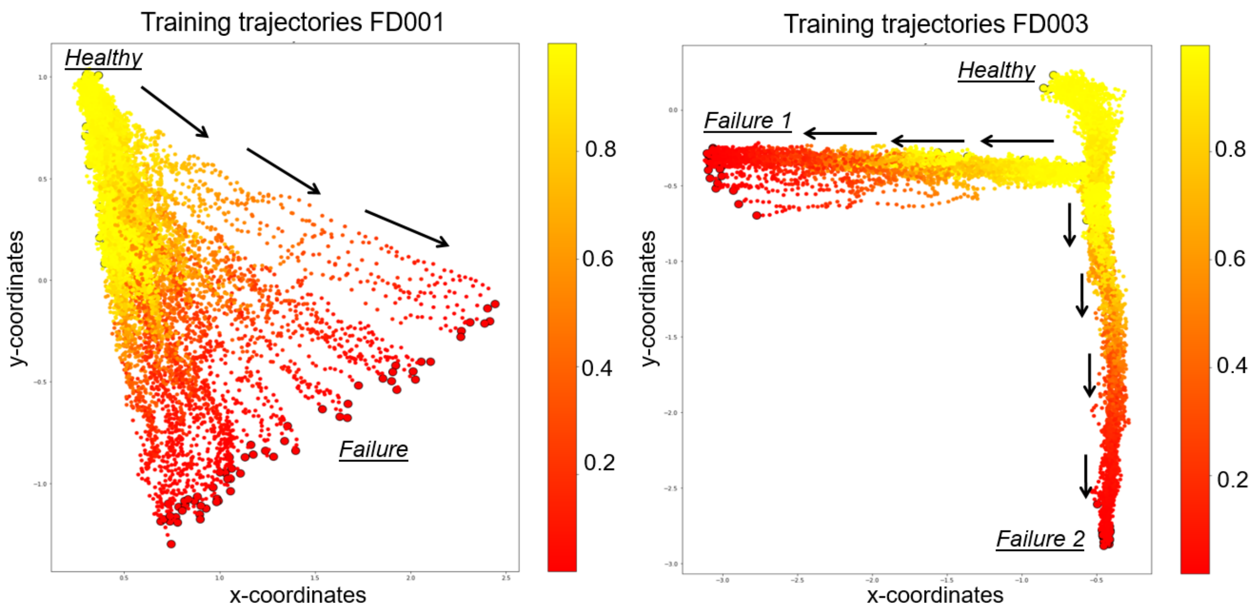

Figure 7. Training trajectories from FD001-train-train dataset on the left and FD003-train-train dataet on the right. The 2D trajectories start from healthy state which is represented by yellow color (health approximately 1) and then transition to a failure state gradually, which is represented by red color (health approximately 0). Note that FD001 has only one failure class and FD003 has two failure classes.

of the time series in the training dataset and the red points with black border show the failure state of the training time series. These points are merely the first and the last temporal embeddings extracted from the 2D trajectories of training time series. The third subplot shows the health curve. We will discuss the extraction of the health curve and the failure probabilities later in this section. Note that the time series in the second plot belongs to FD001-test, therefore its trajectory is only partial. Similar visualizations for FD003-train-test and FD003-test are shown in the bottom two plots in Figure 9. Here, in the middle subplot, instead of one failure class, it is possible to see the possible evolution of time series to two different failure classes.

8.2. Health Curve Estimation

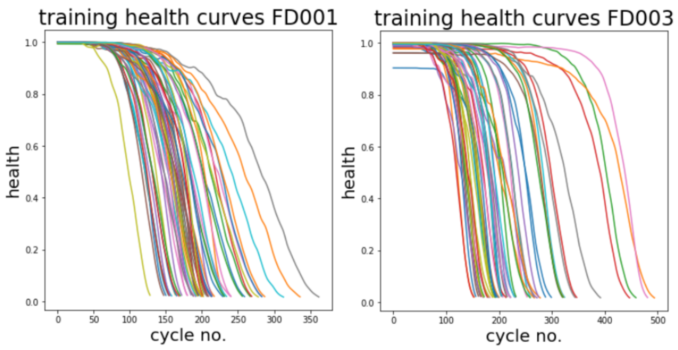

Figure 8. Health curves obtained during the training stage of health estimation for all the time series in FD001-train-train and FD003-train-train datasets.

Training health curves: The 2D failure evolution trajectories learned during the training phase are then fed into the health estimation framework (Section 4.1). Figure 8 shows the health curves obtained during the training stage of health estimation for all the time series in FD001-train-train and FD003-train-train datasets. All the health curves for FD001-train-train start from a higher value, very close to 1, and degrade gradually to a value close to 0. In contrast to that, the health curves for a few time series in FD003-train-train start from a value slightly less than 1. This is primarily due to the presence of two failure classes and variation in the degradation behavior of the time series in these two failure classes in this dataset. Due to this variation, the offset h_{G0i} computed by the global health estimation method is slightly more than that for FD001-train-train dataset. Apart from the offset, the shapes of the health curves reveal absolutely no information about the two failure classes.

Test health curves: During the testing stage, the test trajectory $\mathbf{Z}_{testi}$ is fed into the testing algorithm of the health curve estimation (Section 4.2). The health curves for two test time series from the datasets FD00X-train-test and FD00X-test are shown in the third subplot of Figure 9. The health curves for FD00X-train-test, shown in the top plots of each figure, represent the complete life cycle, whereas, those for FD00X-test, shown in the bottom plots of the two figures represent incomplete life cycles.

8.3. Failure Prediction

After the computation of health curves, the next goal is to train the regression MLP of Section 5.1 by using the health curves $\mathbf{h}_i$ and trajectories $\mathbf{Z}_i$ of the datasets FD00X-train-train as input. Then, for a test time series from the datasets FD00X-train-test or FD00X-test, we predict the failure probabilities and also extract the number of cycles M prior to the end-of-life when the failure is predicted by the algorithm, using the procedure specified in Section 5.2.

8.3.1. Experimental Setting for Training Stage

First, note that the dataset FD001-train-train has only one type of failure. This is the special case discussed in Section 5.2.1. One does not need to follow the training procedure in Section 5 for this dataset, as the univariate probability can simply be obtained from the health curve at every time instant as $p_{it} = 1 - h_{it}$. For the dataset FD003-train-train, the task is a multiclass prediction, therefore, we follow the procedure in Section 5 to train a network for failure class prediction.

8.3.2. Experimental Setting and Challenges of Testing Stage

For the time series in FD001-train-test dataset, which has only failure class, the probability of failure is set as $p_{it} = 1 - h_{it}$ for every time instant t. For the time series in FD003-train-test dataset, the multivariate failure probability vector $\mathbf{p}_{it} \in [0, 1]^{F+1}$ is obtained from the trained regression MLP of Section 5.1, where $F = 2$ is the number of failure classes.

There are several challenges in the evaluation of test results on failure class prediction:

1. Absence of ground truth failure class labels for FD003-train-test and FD003-test datasets.
2. Absence of the total number of cycles until the end-of-life of FD00X-test datasets.
3. During the test phase, the regression MLP will not predict a failure if sufficient degradation has not occurred in FD00X-test datasets.
4. The evaluation metrics for failure prediction task need to be defined.

In the following discussions, we will present our solutions for each of the above problems in detail.

Ground truth failure class label generation: In order to generate failure labels for FD003-train-test dataset, we assume that the end-of-life behavior for a type of failure is similar across the engines and differs from the signatures of other failure types. Thus, we extract a window of size $w = 30$ from the raw time series before the end of life of each engine and then compute a pairwise Dynamic Time Warping (DTW) distance matrix of size $N \times N$, where N is the total number of time series in a dataset. Then, we perform a 2D t-SNE (Maaten & Hinton, 2008) and assign failure class labels based on the Kmeans clustering on the t-SNE embedding.

The same approach cannot be used for generating labels from the actual test set (FD003-test) because of the presence of only partial time series. Hence, we do not report failure prediction accuracy on the FD003-test dataset.

Ground truth for total life time: For determining the number of cycles until the end-of-life for the datasets FD00X-test, we simply add the total length of the partial test time series to the ground truth RUL available with the data. The total life time for FD00X-train-test datasets is equal to the length T_i of the time series $\mathbf{X}_i$.

Evaluation metrics: For FD00X-train-test, the failure class labels are available, therefore, we use classification accuracy of the failure type as evaluation metric. In addition to that we also report the mean and standard deviation of the number of cycles (the metric M from Section 5.2) prior to the end-of-life at which the correct failure is predicted.

The datasets FD00X-test contain partial time series and no failure class labels. The failure prediction algorithm in Section 5.2 will only predict a failure for FD001-test if $p_{it} > 0.5$ or $h_{it} < 0.5$ for some $t \in \{0, \cdots, T_i\}$. For the FD003-test a failure will be predicted only if for some $t \in \{0, \cdots, T_i\}$, $\max(\mathbf{p}_{it}[1 : F+1]) > \mathbf{p}_{it}[0]$, i.e., when the maximum of the failure probabilities exceeds the probability of healthy state. Therefore, we report the mean and standard deviation of M for only those time series for which the failure is predicted by the algorithm.

Dataset	**total samples**	**failure predicted**	**accuracy** (%)	**cycles** (M)
FD001-train-test	36	36	100%	44 ± 14.2
FD003-train-test	36	36	100%	69 ± 9.1
FD001-test	100	31	-	49.8 ± 17.1
FD003-test	100	24	-	71± 9.4

Table 5. Results for failure prediction in terms of the accuracy and number of cycles M for different datasets. Both FD001-train-test and FD003-train-test achieve 100% accuracy in failure prediction. All the time series in FD001 and FD003 are correctly identified as progressing towards a failure (correct class) on average 44 and 69 cycles before the end-of-life. For the FD001-test and FD003-test datasets, failures were predicted in 24 and 31 out of 100 time series, on average 49.8 and 71 cycles before the end-of-life.

Quantitative results: Table 5 shows the results for failure prediction in terms of the accuracy and number of cycles M for different datasets. Both FD001-train-test and FD003-train-test achieve 100% accuracy in failure prediction. In the context of FD001, this means that all the time series were correctly identified as progressing towards a failure on average 44 cycles before the end-of-life. In the context of FD003 this implies that all the time series were also identified as progressing towards the *correct* failure class on average 69 cycles before the end-of-life.

As explained earlier, our failure prediction algorithm predicts a failure when for some $t \in \{0, \cdots, T_i\}$, $\max(\mathbf{p}_{it}[1 : F+1]) > \mathbf{p}_{it}[0]$, where $\mathbf{p}_{it}[0]$ is the probability of the system being healthy. For many of the time series in FD00X-test datasets, this condition is not met since these time series do not show any degradation. Therefore, for the FD001-test and FD003-test datasets, failures were predicted in 24 and 31 out of 100 time series, on average 49.8 and 71 cycles before the end-of-life. Our experiments show that the test time series for which the failure prediction could not be performed have a terminal health of atleast greater than 0.75. Moreover, all these time series have large RUL values. For FD001 dataset, all such time series have an RUL of atleast 52, whereas for FD003, the RUL is atleast 75. Overall, these results show that out method is capable of predicting failures accurately and sufficiently before the end-of-life of the engines.

Failure evolution probabilities: The middle subplots in Figure 9 show the evolution of failure probabilities from healthy to failure state for representative time series from FD001-train-test (top), FD001-test datasets (second row), FD003-train-test (third row) and FD003-test (bottom) datasets. Although we show failure probabilities for only three trajectory points for the clarity of the figure, computation was done for each trajectory point. From our experiments, we can easily draw the conclusion that the probabilities evolve smoothly from healthy to failure state, thus enabling continuous monitoring and decision making in the field. Moreover, in the case of FD003 datasets, one can get more elaborate information on the evolution from healthy to correct failure classes.

8.4. Remaining Useful Life Estimation

We compute the RUL for each of the test time series $\mathbf{X}_{testi}$ in FD00X-test datasets using the health curve matching method of Section 6. RULs for FD001-test are estimated by matching with the health curves from FD001-train-train and those for FD003-test by matching with curves from FD003-train-train.

Evaluation Metrics: We use the following evaluation metrics: Score (S), Accuracy (A), Mean Average Error (MAE), Mean Squared Error (MSE), MAPE1, False Positive Rate (FP) and False Negative Rate (FN), proposed in (Saxena, Celaya, et al., 2008) for evaluating the performance of our RUL estimation method. The values of τ_1 and τ_2 are set to 10 and 13 as proposed in (Saxena, Goebel, et al., 2008).

Results: Table 6 compares various performance metrics on the FD001-test dataset for the state-of-the-art *indirect data-driven* methods - which use health curve for RUL estimation, with our proposed method. The compared methods include

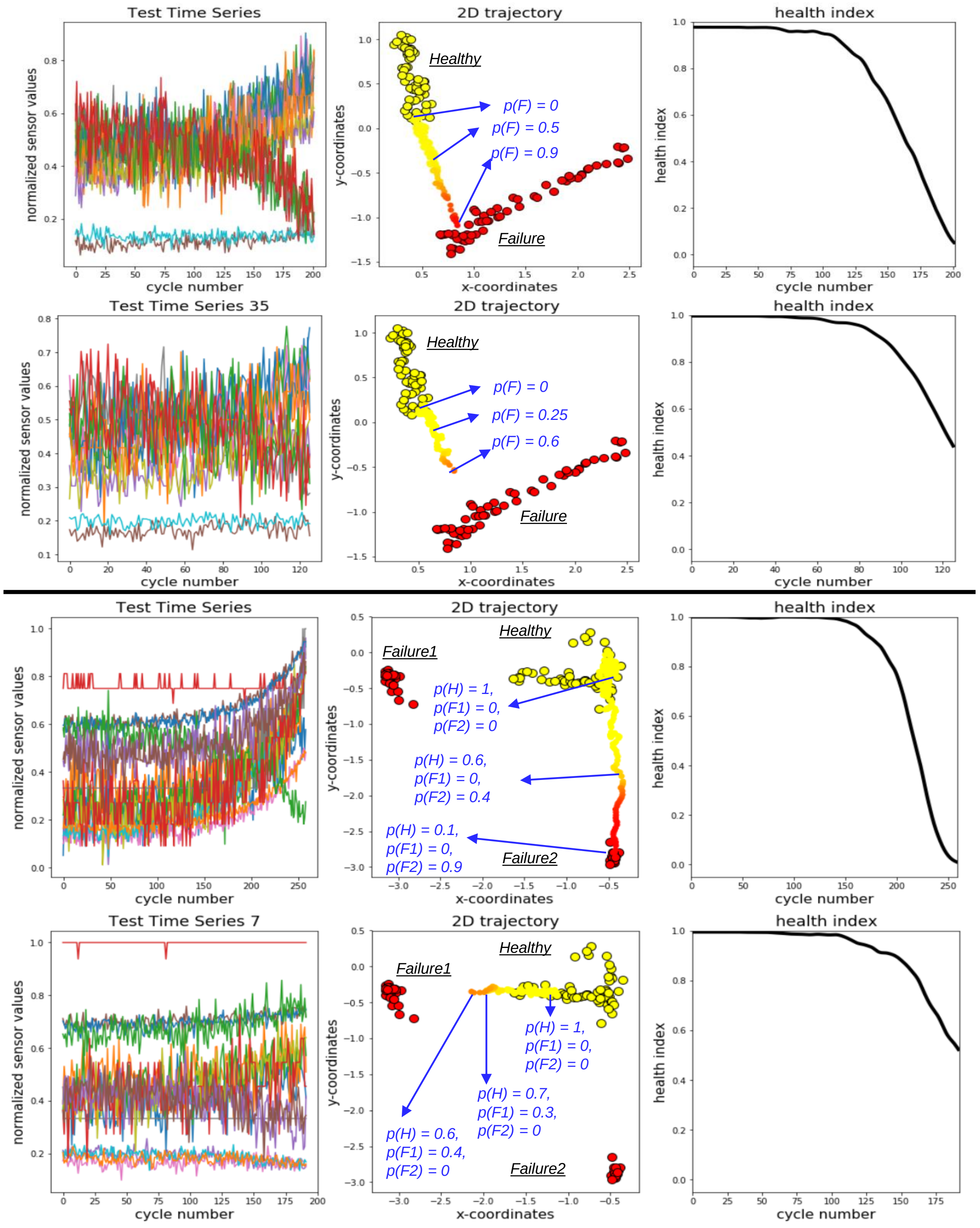

Figure 9. Sample trajectories from FD001-train-test, FD001-test, FD003-train-test and FD003-test datasets. In each plot, the first subplot shows the raw time series and the second shows the trajectory evolving from healthy (yellow) to failure state (red), along with the failure class probabilities. The yellow circles with black border represent the healthy points for each of the time series in the training dataset and the red points with black border show the failure state of the training time series. The third subplot shows the health curve.

Metric	Recon-RUL	Embed-RUL	Recon-LR1	Embed-LR1	Recon-LR2	Embed-LR2	RNN-Reg	RULCLIPPER	proposed
S	1263	810	477	219	256	232	352	**216**	360
MSE	546	456	288	**155**	164	167	219	176	207
A(%)	36	48	65	59	67	62	64	**67**	60
MAE	18	17	12	**10**	**10**	**10**	11	**10**	11
MAPE	39	39	20	19	18	19	**17**	20	21
FPR (%)	34	23	19	14	**13**	15	22	56	29
FNR (%)	30	29	16	27	20	23	24	44	**10**

Table 6. RUL estimation results on FD001-test dataset. Our method is comparable to the best performing methods in terms of MAE and better as compared to many others in terms of MSE. Our method also achieves the lowest FNR among all the other methods and outperforms the second best method (Recon-LR1) by reducing the FNR by 60%.

Recon-RUL, Embed-RUL, Recon-LR1, Embed-LR1, Recon-LR2, Embed-LR2, RNN-Reg (Gugulothu et al., 2017) and RULCLIPPER (Ramasso, 2014). The comparison with the above mentioned state-of-the-art methods is on the benchmark datasets, so these algorithms were not implemented from scratch. RUL was reported directly from the representative publication. As the test benchmark dataset is fixed, so the difference in training configurations among various algorithms, such as the amount of data used, does not impact the fairness of the experimental comparisons. Note that we do not compare our results with the *direct data-driven* approaches (A. Zhang et al., 2018), (J. Wang et al., 2018), (Li et al., 2019), (C. Zhang et al., 2016) because these methods do not involve the computation of health curve. Therefore the comparison would not be fair.

Metric	RULCLIPPER (Ramasso, 2014)	proposed
S	**317**	610
MSE	**256**	265
A(%)	**59**	55
MAE	**12**	14
MAPE	**23**	25
FPR (%)	66	**29**
FNR (%)	34	**20**

Table 7. On the FD003-test dataset, our proposed method is comparable to the RULCLIPPER method in MSE and outperforms significantly in terms of FPR and FNR. More specifically it reduces the FPR by 127% and FNR by 70%.

Table 6 shows that our method is comparable to the best performing methods in terms of MAE and better as compared to Recon-RUL, Embed-RUL, Embed-LR1 and RNN-Reg in terms of MSE. Interestingly, our method achieves the lowest FNR among all the other methods and outperforms the second best method (Recon-LR1) by reducing the FNR by 60%. This implies that our method is conservative in FNR, i.e., it does not predict unusually large RUL values. This is also evident from Figure 10, which shows the actual versus predicted RUL, sorted in the ascending order. For very short partial time series, or time series with large actual RUL values, our method tends to under-estimate the RUL, which results in an overall small FNR. When the test time series has been observed for a very small set of samples, the engine is still healthy and a reliable estimate of RUL is difficult to obtain with the health curve matching approach. In such a case an RUL prediction algorithm can easily predict unbounded values of RUL, which is avoided by our method.

Table 7 shows similar comparison for the FD003-test dataset. Our proposed method is comparable to the RULCLIPPER method in MSE and outperforms significantly in terms of FPR and FNR. More specifically it reduces the FPR by 127% and FNR by 70%.

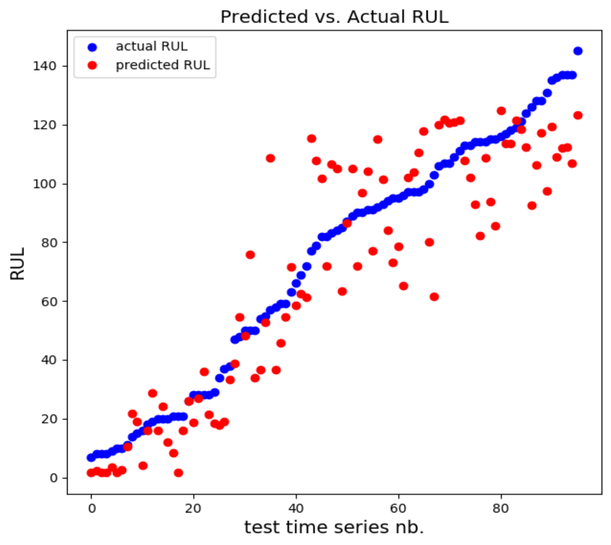

Figure 10. Actual versus predicted RUL for the FD001-test dataset, sorted in the ascending order. For very short partial time series, or time series with large actual RUL values, our method tends to under-estimate the RUL, which results in an overall small FNR.

9. DISCUSSION & FUTURE WORK

While the state-of-the-art PHM literature has focused mostly on the RUL estimation method, our work highlights the several ignored but important aspects of a PHM system, such as failure evolution analysis and failure prediction. To the best of our knowledge, our work is the first one to emphasize and incorporate visualization and interpretability into the RUL computation process, by associating a simple yet very powerful visualization to it. We show that one can obtain a performance, comparable to the state-of-the-art methods which use health curves for RUL estimation, by performing all the computations in a much simpler and interpretable manner in a 2D trajectory space. Our method, can therefore be used primarily for failure evolution analysis and visualization while achieving a reasonable RUL performance.

An important property of the proposed framework is anomaly detection. In case an unknown failure develops in the system, the trajectory in the 2D "failure space" will not move towards any of the known failure modes, but still away from the healthy mode. The proposed method is still be capable of detecting health degradation and providing useful information to maintenance personnel regarding anomalous state of the system.

Despite all the above characteristics of our framework, an important question remains unanswered: can we use the extra information provided by our framework, i.e., the 2D trajectories and failure evolution probabilities for *improving* RUL performance? Our future work will be dedicated to answer this question as well as performing rigorous experimental evaluations and comparisons of our work with the state-of-the-art in terms of RUL performance. Furthermore, we will also explore other proposed architectures, i.e., TrajecNets-super and TrajecNets-siamese in our future work.

10. CONCLUSION

We propose TrajecNets, a set of novel RNN architectures which are configured to study degradation trajectories in 2D space from healthy to failure state. The degradation trajectories are computed by TrajecNets using time series data collected by sensors installed in a system of interest. TrajecNets provides a visual, interpretable and explainable addition to the traditional PHM pipeline. It summarizes the information learned by the complex deep learning model in a 2D space and provides meaningful interpretation to the health and RUL estimation tasks. Due to the unsupervised setting the proposed methodology can automatically reveal various failure modes (classes), without the explicit requirement of failure labels during training phase. Furthermore, we propose a novel health index estimation method from 2D degradation trajectories, which consists of a global step to take into account the initial wear of the system under consideration and local health to take into account the health of the system with respect to its nascence. The computed health curve can be converted into an evolving failure probability using an unsupervised methodology. Experiments on trajectory visualization, failure class prediction, health curve and RUL estimation on the Turbofan dataset (Saxena, Goebel, et al., 2008) show the potential of our proposed framework to be used in real time critical applications.

ACKNOWLEDGMENT

The research presented in this paper is carried out under the EU H2020 framepwrok funded ReMAP project (Grant 769288-2).

REFERENCES

Aizpurua, J. I., & Catterson, V. M. (2015). Towards a methodology for design of prognostic systems. In *Annual conference of the prognostics and health management society 2015* (pp. 504–517).

Akintayo, A., Lore, K. G., Sarkar, S., & Sarkar, S. (2016). Early detection of combustion instabilities using deep convolutional selective autoencoders on hi-speed flame video. *arXiv preprint arXiv:1603.07839*.

Cadini, F., Zio, E., & Avram, D. (2009). Model-based monte carlo state estimation for condition-based component replacement. *Reliability Engineering & System Safety*, *94*(3), 752–758.

Chollet, F., et al. (2015). *Keras.* `https://keras.io`.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2015). Gated feedback recurrent neural networks. In *International conference on machine learning* (pp. 2067–2075).

Elattar, H. M., Elminir, H. K., & Riad, A. (2016). Prognostics: a literature review. *Complex & Intelligent Systems*, *2*(2), 125–154.

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

Gugulothu, N., TV, V., Malhotra, P., Vig, L., Agarwal, P., & Shroff, G. (2017). Predicting remaining useful life using time series embeddings based on recurrent neural networks. *arXiv preprint arXiv:1709.01073*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016, June). Deep residual learning for image recognition. In *2016 ieee conference on computer vision and pattern recognition (cvpr)* (p. 770-778). doi: 10.1109/CVPR.2016.90

Hermans, M., & Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. In *Advances in neural information processing systems* (pp. 190–198).

Ilin, A., Prémont-Schwarz, I., Hao, T. H., Rasmus, A., Boney, R., & Valpola, H. (2017). Recurrent ladder networks. *arXiv preprint arXiv:1707.09219*, *3*(6).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436.

Li, J., Li, X., & He, D. (2019). A directed acyclic graph network combined with cnn and lstm for remaining useful life prediction. *IEEE Access*, *7*, 75464-75475. doi: 10.1109/ACCESS.2019.2919566

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(Nov), 2579–2605.

Malhotra, P., TV, V., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., & Shroff, G. (2016). Multi-sensor prognostics using an unsupervised health index based on lstm encoder-decoder. *arXiv preprint arXiv:1608.06154*.

Mosallam, A., Medjaher, K., & Zerhouni, N. (2015). Component based data-driven prognostics for complex systems: Methodology and applications. In *2015 first international conference on reliability systems engineering (icrse)* (pp. 1–7).

Mosallam, A., Medjaher, K., & Zerhouni, N. (2016). Data-driven prognostic method based on bayesian approaches for direct remaining useful life prediction. *Journal of Intelligent Manufacturing*, *27*(5), 1037–1048.

Myötyri, E., Pulkkinen, U., & Simola, K. (2006). Application of stochastic filtering for lifetime prediction. *Reliability Engineering & System Safety*, *91*(2), 200–208.

Phillips, P. A. (2012). *Health monitoring of electrical actuators for landing gears* (Unpublished doctoral dissertation). The University of Manchester (United Kingdom).

Qiu, H., Lee, J., Lin, J., & Yu, G. (2003). Robust performance degradation assessment methods for enhanced rolling element bearing prognostics. *Advanced Engineering Informatics*, *17*(3-4), 127–140.

Ramasso, E. (2014). Investigating computational geometry for failure prognostics. *International Journal of Prognostics and Health Management*, *5*(1), 005.

Reddy, K. K., Venugopalan, V., & Giering, M. J. (2016). Applying deep learning for prognostic health monitoring of aerospace and building systems. In *1st acm sigkdd workshop on ml for phm.*

Saxena, A., Celaya, J., Balaban, E., Goebel, K., Saha, B., Saha, S., & Schwabacher, M. (2008). Metrics for evaluating performance of prognostic techniques. In *2008 international conference on prognostics and health management* (pp. 1–17).

Saxena, A., Goebel, K., Simon, D., & Eklund, N. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 international conference on prognostics and health management* (pp. 1–9).

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first aaai conference on artificial intelligence.*

Vachtsevanos, G., & Wang, P. (2001). Fault prognosis using dynamic wavelet neural networks. In *2001 ieee autotestcon proceedings. ieee systems readiness technology conference.(cat. no. 01ch37237)* (pp. 857–870).

Wang, J., Wen, G., Yang, S., & Liu, Y. (2018). Remaining useful life estimation in prognostics using deep bidirectional lstm neural network. In *2018 prognostics and system health management conference (phm-chongqing)* (pp. 1037–1042).

Wang, T. (2010). *Trajectory similarity based prediction for remaining useful life estimation* (Unpublished doctoral dissertation). University of Cincinnati.

Wang, T., Yu, J., Siegel, D., & Lee, J. (2008). A similarity-based prognostics approach for remaining useful life estimation of engineered systems. In *2008 international conference on prognostics and health management* (pp. 1–6).

Wu, Y., Yuan, M., Dong, S., Lin, L., & Liu, Y. (2018). Remaining useful life estimation of engineered systems using vanilla lstm neural networks. *Neurocomputing*, *275*, 167–179.

Zhang, A., Wang, H., Li, S., Cui, Y., Liu, Z., Yang, G., & Hu, J. (2018). Transfer learning with deep recurrent neural networks for remaining useful life estimation. *Applied Sciences*, *8*(12), 2416.

Zhang, C., Pin, L., K. Qin, A., & Chen Tan, K. (2016, 07). Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics. *IEEE Transactions on Neural Networks and Learning Systems*, *PP*, 1-13. doi: 10.1109/TNNLS.2016.2582798

Zhao, R., Wang, J., Yan, R., & Mao, K. (2016). Machine health monitoring with lstm networks. In *2016 10th international conference on sensing technology (icst)* (pp. 1–6).

Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., & Gao, R. X. (2019). Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, *115*, 213–237.

Zhao, R., Yan, R., Wang, J., & Mao, K. (2017). Learning to monitor machine health with convolutional bidirectional lstm networks. *Sensors*, *17*(2), 273.

BIOGRAPHIES

Nauman Shahid is a Senior Research Scientist at United Technologies Research Center, Ireland. He received his PhD degree in Machine Learning and Signal Processing from Ecole Polytechnique Federale de Lausanne (EPFL) Switzerland, in 2017. His current research interests include graph signal processing, low-rank matrix and tensor decompositions, time series failure prediction and PHM.

Anarta Ghosh is a Staff Research Scientist at United Technologies Research Center, Ireland. He received the BSc degree (1995) with honors in physics from Calcutta University, India. He did Master of Engineering(2000) in electrical engineering from the Indian Institute of Science, Bangalore, followed by a PhD (2007) from University of Groningen, The Netherlands, in the field of Machine Learning and Computer Vision. Prior to the present position, he has held research positions at Trinity College Dublin (Dublin, Ireland) and University of Washington (Seattle, US). His research interests are in the field of machine learning, data analytics, pattern recognition, signal/image processing, cognitive computer vision, medical image analysis and PHM.