Multi-Objective Optimization of Extreme Learning Machine for Remaining Useful Life Prediction

Hyunho Mo$^{[0000-0002-6497-2250]}$ and Giovanni Iacca$^{[0000-0001-9723-1830]}$

Department of Information Engineering and Computer Science,
University of Trento, Italy

Abstract. Given that physics-based models can be difficult to derive, data-driven models have been widely used for remaining useful life (RUL) prediction, which is a key element for predictive maintenance. In industrial applications, although the models have to be trained in a short time with limited computational resources, recent research using back propagation neural networks (BPNNs) has focused only on minimizing the RUL prediction error, without considering the time needed for training. Driven by this motivation, here we consider a simple and fast neural network, named extreme learning machine (ELM), and we optimize it for the specific case of RUL prediction. In particular, we propose to apply both single-objective and multi-objective optimization to search for the best ELM architectures in terms of a trade-off between RUL prediction error and training time, the latter being determined by the number of trainable parameters. We perform a comparative analysis on a recent benchmark dataset, the N-CMAPSS, in which we compare the proposed methods with other algorithms based on BPNNs. The results show that while the optimized ELMs perform slightly worse than the BPNNs in terms of RUL prediction error, they require a significantly shorter (up to 2 orders of magnitude) training time.

Keywords: Evolutionary Algorithm · Multi-Objective Optimization · Extreme Learning Machine · Remaining Useful Life · N-CMAPSS

1 Introduction

With the advent of Industry 4.0, the remaining useful life (RUL) prediction of industrial components has become one of the mainstream elements in predictive maintenance (PdM) research [1]. Maintenance of industrial components is in fact closely related to costs and reliability, since industry stakeholders can cut their loss by reducing any unplanned downtime. Moreover, the performance of the industrial equipment—as well as the quality of products—can be improved by offering timely maintenance before failures occur. A paradigmatic example is given by the airline industry, where it is crucial to predict the RUL of aircraft engines accurately and timely, in order to guarantee the aircraft operation safety, and make appropriate maintenance decisions. By estimating the RUL of the aircraft engines, airlines can improve maintenance schedules and indeed avoid major disasters. The RUL prediction is thus an essential requirement for minimizing maintenance cost, as well as guaranteeing safety [2].

Recently, many ML-based methods using neural networks have been introduced to handle the above task. One of the earliest approaches, discussed in [3], is to employ a multi-layer perceptron (MLP) for the RUL prediction of aircraft engines. In the same work, the authors also propose to use a convolutional neural network (CNN), that is a widely used network used especially for computer vision tasks. Instead of using traditional back propagation-neural networks (BPNNs), such as a MLP or a CNN, Yang et al. [4] propose to exploit extreme learning machines (ELM) [5], a kind of neural network that requires a much shorter training time compared to (even shallow) BPNNs. As an alternative approach, recurrent neural networks (RNNs), including long short term memory (LSTM), have been proven able to predict the RUL by directly recognizing temporal patterns of the data, instead of extracting their convolutional features [6]. More recently, deeper neural networks have been proposed to improve the RUL prediction accuracy [7]. Other deep learning (DL)-based RUL prediction methods consider a combination of an autoencoder (AE) with a CNN [8], or with a RNN [9]. An attention-based DL framework has been proposed in [10]. Another recent work [11] applied evolutionary computation to optimize deep networks tailored to the RUL prediction task. In this case the authors propose to use a genetic algorithm (GA) to optimize the architecture of a multi-head CNN-LSTM, which was handcrafted in their previous work [12].

Most of the methods proposed so far in the literature mainly focus on achieving a minimization of the RUL prediction error on a well-established benchmark called the commercial modular aero-propulsion system simulation (CMAPSS) dataset [13], which is the *de facto* standard benchmark for RUL prediction. The CMAPSS dataset has been widely used to develop and evaluate the RUL prediction models after it became publicly available on the NASA's data repository in 2008. However, one shortcoming of this dataset is that the data are solely based on MATLAB simulations, without considering real flight conditions, so that each time series is rather short (just a few hundred samples). In 2021, the NASA's data repository released the new CMAPSS (N-CMAPSS) [14] dataset, that contains data acquired under real flight conditions. One major difference with respect to the previous dataset is that each time series consists of millions of samples, thus the total size of the dataset is significantly larger. Therefore, this new realistic dataset provides a chance to develop reliable algorithms for RUL prediction in a real-world context. Moreover, while the previous dataset was small enough to allow researchers to focus only on the minimization of the RUL prediction error, without consider the training time, the N-CMAPSS dataset, due to its much larger amount of data, requires algorithms that are faster to train, without compromising the RUL prediction error.

It should be noted that, although reducing the training time (which correlates to the number of trainable parameters) is a crucial objective in industrial contexts that do not normally have access to expensive computing infrastructures, this aspect has not been considered so far in the literature, also because of the aforementioned limitations of the CMAPSS dataset.

To achieve the multiple (and conflicting) objectives of reducing the RUL prediction error while also minimizing the training time, here we propose to use an ELM and we optimize its architecture and parameters using both a single-objective optimization (SOO) and a multi-objective optimization (MOO) algorithm. For the SOO case, we use a custom GA with two different fitness function formulations (including or not a penalty on the neural network complexity): we use its results as baseline. For the MOO case, we use the well-known non-dominated sorting genetic algorithm II (NSGA-II) [15], which has been recently applied successfully also for neural architecture search [16]. The choice of using an ELM is motivated by the experimental results presented in [17], that are based on the CMAPSS dataset, where the authors show that ELM-based models can provide comparable performance to BPNNs in terms of RUL prediction error, while enabling a considerable saving in terms of training time. The main limitation of the study presented in [17], however, is that it only considers for the comparative analysis a small set of manually designed ELM architectures. On the other hand, exploring the search space of those networks with an automatic search process can potentially provide further performance improvements: our goal is to fill this gap. Moreover, to take into account a proper assessment of the training time, we perform our experiments on the N-CMAPSS dataset.

Another important advantage of our proposal is that it is based on a fully data-driven approach that is able to predict the RUL directly, by leveraging the relation between the degradation pattern of the raw sensor data and the RUL, thus without using any physics-based model of the degradation process (as recently done in [18]). This is also made possible by the large amount of data available in the N-CMAPSS dataset: in fact, these sensor data allow to model the RUL prediction problem as a regression task on a multivariate time series.

To summarize, the main contributions of this work can be identified in the following elements:

- We achieve a successful trade-off between two conflicting objectives, namely the RUL prediction error and the number of trainable parameters.
- To the best of our knowledge, this is the first use case of multi-objective neural architecture search applied to RUL prediction (and, in particular, on the N-CMAPSS dataset).

The rest of the paper is organized as follows: in Section 2, the background concepts on ELMs are introduced. The details of the proposed methods are presented in Section 3. Then, Section 4 discusses the details of our experiments and Section 5 presents the numerical results. Finally, Section 6 provides the conclusions of this work.

2 Background

An ELM is a fast training algorithm for *single-hidden layer feed-forward neural networks* (SLFNs). While BPNNs tune their parameters iteratively with (usually, time-consuming) gradient-based computations, ELMs merely use an analytically determined non-iterative solution as the output weights, together with randomly initialized input weights.

An ELM can be formally described as follows. Let N be the number of labeled samples, where each sample is a pair made of a d-dimensional input vector and the corresponding c-dimensional label, which are denoted by $\boldsymbol{x}_i$ and $\boldsymbol{t}_i$ respectively. A given set of training samples can then be written as $(\boldsymbol{x}_i, \boldsymbol{t}_i)$, $i \in [1, N]$ with $\boldsymbol{x}_i \in \mathrm{I\!R}^{\mathrm{d}}$ and $\boldsymbol{t}_i \in \mathrm{I\!R}^{\mathrm{c}}$. In our experiments on the N-CMAPSS dataset, d is the number of monitoring signals, and $c = 1$ (i.e., the label is a real number representing a RUL value).

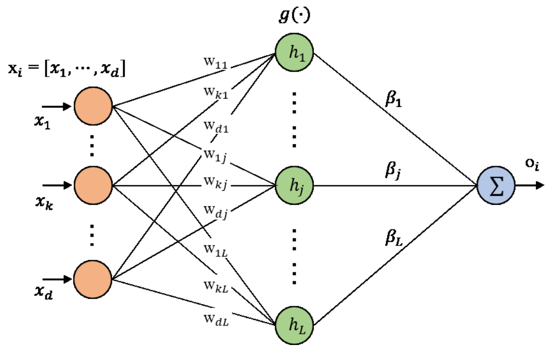

Fig. 1: Illustration of ELM with the structure of a SLFN.

The notation used for describing the ELM and the structure of a SLFN are visualized in Fig. 1. For a given input sample $\boldsymbol{x}_i$, the output of a SLFN o_i with L hidden neurons and activation function $g(\cdot)$ is defined by:

$$\mathrm{o}_i = \sum_{j=1}^{L} \beta_j g(\boldsymbol{w}_j \cdot \boldsymbol{x}_i + b_j). \tag{1}$$

where $\boldsymbol{w}_j = [w_{1j}, \ldots, w_{dj}]$ is a vector of weights on the connections between the d input neurons (which are assumed to be linear) and each j-th hidden neuron, β_j is the weight on the connection between the j-th hidden neuron and the output neuron, and b_j denotes the bias for the j-th hidden neuron. The computation for all the N equations (one for each of the N samples) can be written compactly as:

$$\boldsymbol{H} \cdot \boldsymbol{\beta} = \begin{bmatrix} g(\boldsymbol{w}_1 \cdot \boldsymbol{x}_1 + b_1) & \cdots & g(\boldsymbol{w}_L \cdot \boldsymbol{x}_1 + b_L) \\ \vdots & \ddots & \vdots \\ g(\boldsymbol{w}_1 \cdot \boldsymbol{x}_N + b_1) & \cdots & g(\boldsymbol{w}_L \cdot \boldsymbol{x}_N + b_L) \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_L \end{bmatrix} \tag{2}$$

where $\boldsymbol{H}$ is the hidden layer output matrix (of size $N \times L$), and $\boldsymbol{\beta}$ (of size L) consists of the weights of all the connections between the hidden neurons and the output neuron.

To train the SLFN defined above is equivalent to find a least square solution $\hat{\boldsymbol{\beta}}$ to the linear system $\boldsymbol{H} \cdot \boldsymbol{\beta} = \boldsymbol{T}$ where $\boldsymbol{T} = [\boldsymbol{t}_1, \cdots, \boldsymbol{t}_N]^\top$. Therefore, the mathematical formulation of the training procedure can be expressed as:

$$\|\boldsymbol{H}\hat{\boldsymbol{\beta}} - \boldsymbol{T}\| = \min_{\boldsymbol{\beta}} \|\boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{T}\| \tag{3}$$

As discussed in [5], the smallest norm least squares solution of the above equation is determined by:

$$\hat{\boldsymbol{\beta}} = \boldsymbol{H}^\dagger \boldsymbol{T} \tag{4}$$

where $\boldsymbol{H}^\dagger$ denotes the Moore-Penrose generalized inverse of the matrix $\boldsymbol{H}$. The pseudo-inverse can be calculated as $(\boldsymbol{H}^\top\boldsymbol{H})^{-1}\boldsymbol{H}^\top$. Moreover, an L2 regularization term $\alpha\boldsymbol{I}$ (with $\alpha \in \mathbb{R}$ being an arbitrarily small value) is added to prevent the inverse term from being singular (i.e., $\boldsymbol{H}^\top\boldsymbol{H}$ is replaced by $\boldsymbol{H}^\top\boldsymbol{H} + \alpha\boldsymbol{I}$). The solution to Eq. (3) is then defined by:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{H}^\top\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}^\top\boldsymbol{T}. \tag{5}$$

This last equation describes the ELM training algorithm for SLFNs[1].

As pointed out in the seminal paper on ELMs [5], the training process described by Eq. (5) is *extremely* fast (hence their name). Moreover, it has been shown that ELMs can even achieve a better generalization performance than back-propagation training algorithms [19, 20] and that, compared to BPNNs, they obtain comparable results in terms of prediction accuracy on a variety of regression tasks [17, 21]. Considering these advantages, ELMs then appear an appropriate tool for RUL prediction tasks in industrial contexts that require a fast learning process and a stable generalization performance.

On the other hand, the computational complexity of the ELM training algorithm derives from the size of the matrix $\boldsymbol{H}$: more specifically, it is $\mathcal{O}(NL^2 + L^3)$. In other words, the complexity is cubic w.r.t. the number of hidden neurons, which, in turn, is the same as the number of trainable parameters (i.e., the β values), see also the empiric characterization of the training time shown in Fig. 2. Yet, increasing the number of hidden neurons L does not always contribute to decreasing the RUL prediction error, and may lead to overfitting.

Finding the optimal value of L, as well as of the other parameters of an ELM, is therefore a crucial element for the ELM performance. However, this task that cannot be easily achieved by manual design or by empiric considerations. On the contrary, using evolutionary search to explore this parameter space and to discover optimized ELMs automatically (in terms of both the RUL prediction error and the number of trainable parameters) seems to be a more viable solution.

[1] We should note that, strictly speaking, "ELM" refers to the training algorithm only. However in the literature (as well as in the rest of this paper) "ELM" is generically used to refer to both the training algorithm and the neural network itself.

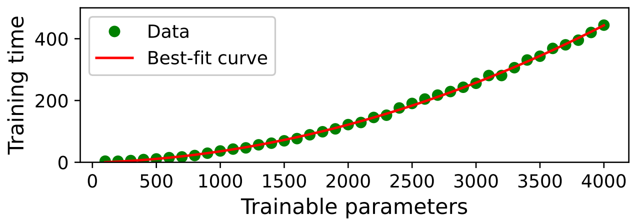

Fig. 2: Correlation between the number of trainable parameters and the training time of the ELM models. The best fit curve shows a super-quadratic dependency between the training time and the number of parameters, in line with the cubic complexity derived analytically, $\mathcal{O}(NL^2 + L^3)$.

3 Methods

We present now the details of the proposed methods: Section 3.1 describes the individual encoding, while Section 3.2 describes the SOO and MOO evolutionary algorithms.

3.1 Individual encoding

Given the baseline structure of the ELM model described in Section 2, we consider the optimization of the following integer parameters:

- number of hidden neurons with hyperbolic tangent activation (n_{tanh});
- number of hidden neurons with sigmoid activation (n_{sigm});
- L2 regularization parameter (r);

In preliminary experiments, we observed that the RUL prediction error as well as the computational complexity are largely affected by the number of hidden neurons. Furthermore, using different activation function $g(\cdot)$ for the hidden neurons also produces different results. Therefore, we encode the number of hidden neurons with two different activation functions into n_{tanh} and n_{sigm}, respectively. The remaining parameter, r, refers to the order of magnitude of the L2 regularization parameter α described in Section 2, i.e., $\alpha = 10^{-r}$.

Overall, the lower and upper bounds for each parameter considered in our experiments are set as follows: $[1, 200]$ (multiplied by a fixed value of 10) for both n_{tanh} and n_{sigm}, and $[2, 6]$ for r. These values have been chosen empirically. In particular, we use a discretization on the number of hidden nodes to reduce the search space yet allowing ELMs of up to $2,000$ tanh hidden neurons and $2,000$ sigmoid hidden neurons. This upper bound is chosen to allow to train each ELM generated during the evolutionary search within a maximum training time of 500 seconds, so that the overall runtime of the search remains affordable. Concerning r, its lower bound corresponds to $\alpha = 10^{-2}$, which we find being the largest value that does not too much affect the ELM performance; the upper bound, on the other hand, corresponds to $\alpha = 10^{-6}$, which we set as the smallest value that makes the inverse term in Eq. (4) non-singular.

3.2 Optimization algorithms

In order to optimize the ELM parameters described in Section 3.1, we consider first a SOO approach based on a GA aimed at minimizing simply the RUL prediction error. However, as we will see in Section 5, in this case the discovered ELMs tend to have a large number of trainable parameters. To avoid this, we consider a second SOO approach where the number of trainable parameters is included as a penalty factor into the fitness evaluation. Finally, we consider a MOO approach to look explicitly for the best trade-off solutions in terms of RUL prediction error and number of trainable parameters. In all the three approaches, the fitness of each individual is calculated by generating an ELM (the phenotype) associated to the corresponding genotype, i.e., a vector containing the three parameters introduced in Section 3.1. Moreover, given that the N-CMAPSS dataset consists of a training set D_{train} and a test set D_{test}, we further split D_{train} into training purpose data, E_{train}, and validation purpose data, E_{val} (i.e., $D_{train} = E_{train} \cup E_{val}$). The fitness is calculated on the latter.

Single-objective optimization First, we initialize the population by generating n_{pop} individuals at random. In the main loop of the evolutionary search, we use crossover and mutation as genetic operators. As for crossover, we implement a specialized one-point crossover: at first, the population is sorted according to the fitness of its individuals. Then we mate the individual in the $(2i)$-th position with the one in the $(2i+1)$-th position following a crossover probability p_{cx}, where $i \in [0, n_{pop}/2-1]$. This allows us not only to exploit the best individuals, but also to explore the areas of the search space that are distant from the region in which the best individuals lie. Then, we apply uniform mutation following a mutation probability p_{mut}, according to which each gene can be mutated to another value uniformly drawn from its bounds specified in Section 3.1. The expected number of mutations is determined by a p_{gene} parameter which is set to 0.4, so that we have, on average, 1.2 mutated genes out of 3. This enables us to have a relatively fast search process by having on average at least one gene mutation, while avoiding disruptive mutations. After that, we create the population for the next generation with implicit elitism, i.e., the worst parent involved in the crossover is replaced by its offspring if the latter has a better fitness. We set both p_{cx} and p_{mut} to 0.5.

We stop the above process after a fixed number of generations n_{gen}, after which the algorithm returns the best individual found during the evolutionary search based on the specific fitness. We set n_{pop} to 28 and n_{gen} to 30. We have empirically found that these settings allow enough evaluations to observe convergence on the fitness across generations.

We consider two different scenarios in terms of SOO, that we refer to respectively as SOO-ELM(1) and SOO-ELM(2). In the first case, the fitness of each individual is defined as the root mean square error (RMSE) of its phenotype on E_{val}, after training it on E_{train}. We indicate this validation RMSE with $RMSE_{val}$. As said, the limitation of this approach is that the size of the discovered ELM tends to be very large in order to decrease $RMSE_{val}$. In other words,

n_{tanh} and n_{sigm} tend to converge to their upper bounds throughout the evolutionary process. To overcome this limitation, we consider a second SOO approach in which the fitness formulation includes the number of trainable parameters of the ELM. This is obtained by calculating the fitness as $RMSE_{val} + \tau L$, where τ is a constant weight. This way, we can prevent the survival of the unnecessarily large ELMs by penalizing their fitness with τL.

Multi-objective optimization As we discussed earlier, minimizing $RMSE_{val}$ and L are conflicting objectives in the architecture search of the ELMs. While the SOO-ELM(2) approach discussed above somehow goes in the direction of compromising those two objectives, the best model still largely depends on a human decision, since it depends on how the value of τ is parametrized.

To tackle this limitation, we propose to use a MOO algorithm, NSGA-II, to search explicitly for a set of trade-off ELMs. We refer to this method as MOO-ELM. We follow the procedure of the original NSGA-II algorithm. At the beginning of the evolutionary run, a population of n_{pop} individuals is randomly initialized. In the main loop after the initial generation, an offspring population of the same size is created by using tournament selection, crossover and mutation. The tournament selection primarily checks the dominance across the individuals in the population. Then, the secondary criteria, crowding distance, is considered only once all the non-dominated individuals have been considered. Regarding the crossover and the mutation, we use the same strategies specified in the single-objective case, with the same parametrization. The combined population of the parents and the offspring is then sorted according to non-domination. The best non-dominated sets are inserted into the new population, until no more sets can be accommodated. For the next non-dominated set, which would make the new population be larger than the fixed population size n_{pop}, only the solutions largest crowding distance values are inserted in the remaining slots in the new population. When the new population is ready, its offspring population is created and the same process continues to the next generation.

Also in this case, we set n_{pop} to 28 and n_{gen} to 30. After the fixed number of generations, the evolutionary search returns a set of the trade-off solutions that have the top dominance level (i.e., the method returns a Pareto front).

4 Experimental setup

In this section, we present the details of our experimentation: first, we briefly describe the N-CMAPSS dataset in Section 4.1. In the following Section 4.2, we describe two BPNNs that are used for the comparison with our methods. Then, the computational setup and data preparation steps are outlined in Section 4.3.

4.1 Benchmark dataset

The proposed methods are evaluated on the N-CMAPSS dataset. Specifically, we only use its sub-dataset DS02, that has been developed for data-driven methods

[14]. It consists of the run-to-failure degradation trajectories of nine turbofan engines with unknown and different initial conditions. The synthetic trajectories were generated with the CMAPSS dynamic model implemented in MATLAB, but a fidelity gap between simulation and reality is mitigated by reflecting real flight conditions recorded on board of a commercial jet. Furthermore, the relation between the degradation and its operation history is considered, to extend the degradation modeling [14]. Among the nine engines, we use 6 units (u_2, u_5, u_{10}, u_{16}, u_{18} and u_{20}) for the training set D_{train}, and the remaining 3 units (u_{11}, u_{14} and u_{15}) for the test set D_{test}. In particular, the u_{14} and u_{15} relate to shorter and lower altitude flights compared to those of the training units, so that the evaluation results on the D_{test} can implicitly reflect the generalization capability of the RUL prediction model.

Table 1 describes each unit in the dataset. The total number of samples (i.e., timestamps) is 5.26M in D_{train} and 1.25M in D_{test}, with a sampling rate of 1Hz. The end-of-life time t_{EOL} points out the counted flight cycles at the end of the engine's lifespan, i.e., t_{EOL} is the same as the initial value of the labeled RUL. There are two distinctive failure modes in the dataset: the abnormal high-pressure turbine (HPT) and the low-pressure turbine (LPT). The combination of the two failure modes for a unit means that the unit is subject to a more complex failure mode than a single-failure mode.

The dataset provides condition monitoring signals that are related to the useful life of the flight engine. Following the setup used in [18], we select the same 20 signals. The multivariate time series from the 20 signals is used as an input for the ELM model, therefore the dimension of the input sample d is 20.

Table 1: Overview of each unit in the DS02 of N-CMAPSS dataset w.r.t the number of samples m_i, the end-of-life time t_{EOL} and the failure modes.

Training set (D_{train})				Test set (D_{test})			
Unit	m_i(M)	t_{EOL}	Failure Mode	Unit	m_i(M)	t_{EOL}	Failure Mode
u_2	0.85	75	HPT	u_{11}	0.66	59	HPT+LPT
u_5	1.03	89	HPT	u_{14}	0.16	76	HPT+LPT
u_{10}	0.95	82	HPT	u_{15}	0.43	67	HPT+LPT
u_{16}	0.77	63	HPT+LPT				
u_{18}	0.89	71	HPT+LPT				
u_{20}	0.77	66	HPT+LPT				

4.2 Back-propagation neural networks (BPNNs)

To compare the proposed methods to BPNNs, we consider a MLP and a CNN whose architectures were manually designed in [18]. As we discussed in Section 1, the previous works on the old CMAPSS dataset have mostly used deep networks with complex architectures. On the other hand, on the new dataset a simple feed-forward neural network (a MLP) and a 1D CNN are still used as state-of-the-art neural networks, considering the great amount of data (in the order of millions of samples) obtained from real flight conditions.

The architecture of the considered MLP has four hidden layers: the first three of them have 200 neurons each, while the last one has 50 neurons. All the hidden nodes use the ReLU activation function. Similarly to the ELM, a 20-dimensional vector for each timestamp is used as an input for the MLP, so the input layer has 20 nodes, while the output layer consists of a single node.

In contrast, the CNN requires time-windowed data as an input to apply 1D convolution in the temporal direction. Therefore, all the given time series are sliced by a time window of length 50 and stride 1. Each input for the CNN then spans 50 timestamps and has a size of 50×20. Regarding the architecture, the CNN is made up of three convolutional layers followed by a fully connected layer. The first two convolutional layers consist of 10 filters of size 10, while the last convolutional layer has only one filter of the same size. The following fully connected layer has 50 neurons. ReLU is used as activation function for all the nodes in the network.

Lastly, we should note that we follow the training setup used in [18]. In particular, stochastic gradient descent (SGD), with mini-batch size set to 1024, is used to compute the gradient, and $AMSgrad$ [22] is used as an optimization algorithm after initializing the weights with $Xavier$ initialization [23]. For the training iterations, the maximum number of epochs is set to 60 and 30 for the MLP and the CNN respectively, whereas the learning rate is set to 0.001. To handle overfitting, early stopping is considered with a patience of 5.

4.3 Computational setup and data preparation

All the neural networks used in our work are implemented in Python. In particular, TensorFlow 2.3 is used to implement the BPNNs. To implement the ELM, we use the high performance toolbox for ELM (HP-ELM)[2] that supports GPU computation. All the experiments have been conducted on the same workstation with an NVIDIA TITAN Xp GPU, so that we can have a reliable comparison between different models w.r.t the training time. Both the SOO and MOO algorithms are implemented using the DEAP library[3]. Our code is available online[4].

Since we employ the neural networks, each time series is normalized to $[-1, 1]$ by a min-max normalization. As we mentioned in Section 3.2, we split D_{train} in E_{train} and E_{val}: for that, we randomly choose 80% of the data in D_{train} and assign them to E_{train}, used for training each individual. The remaining 20% are designated as E_{val} for the fitness evaluation. For the experiments on the BPNNs, following the experimental setup used in [18], 90% of the data in D_{train} are used to train the networks, and the remaining 10% are reserved for early stopping.

5 Experimental results

The aim of our experiments is to evaluate the optimized ELMs found with the proposed methods described in Section 3.2, by comparing their results with those

[2] `https://github.com/akusok/hpelm`
[3] `https://github.com/DEAP/deap`
[4] `https://github.com/mohyunho/MOO_ELM`

obtained by the two BPNNs described in Section 4.2. The comparison is mainly based on two metrics: 1) the RMSE on D_{test}, and 2) the number of trainable parameters. To highlight the advantage of MOO-ELM, we additionally compare the training times of the methods under study, which as seen in Fig. 2 in the case of ELM is super-quadratic w.r.t. the number of trainable parameters. To improve the reliability of the results from the GA-based methods, we execute 10 independent runs with different random seeds.

Let us first analyze the convergence of MOO-ELM. For each MOO-ELM run, we collect the hypervolume (HV) [24] across 30 generations, and we normalize it to $[0, 1]$ by a min-max normalization. As shown in Fig. 3, a gradual improvement across the generations of MOO-ELM is observed by an increase in the normalized HV. The monotonic increase of the mean HV indicates that the MOO-ELM algorithm keeps finding better non-dominated solutions across the generations. In addition, the slope of the mean HV and its std. dev. indicate convergence at the end of the generations. This means that the algorithm explores the search space enough within 30 generations, even if it starts from different initial populations.

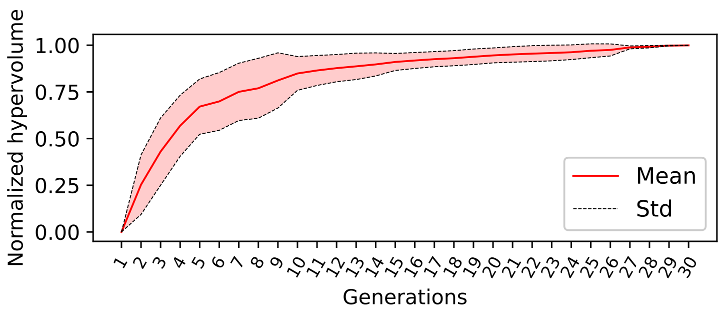

Fig. 3: Normalized hypervolume across generations (mean ± std. dev. across 10 independent runs) for the proposed MOO-ELM approach.

To compare the results obtained by the different methods under study, we aggregate the 10 independent runs in the following way: in the case of the two SOO-ELM approaches, each run returns a single solution that has the best fitness during the evolution. The aggregation is then simply the mean of the test RMSE of the 10 best individuals. On the other hand, each run of NSGA-II returns a number of solutions on the final Pareto front. In our case, we collected 417 non-dominated solutions across the 10 runs. For the sake of comparison, instead of using all of them, we select a fraction of the solutions based on their density in the fitness space, as described in Fig. 4. When we do not have any preference for a certain objective, this strategy can be used to derive a subset of the solutions which are implicitly "preferred" by the MOO algorithm. As shown in Fig. 4, we first place all the solutions from the 10 runs on the fitness space, which is discretized in equally-spaced bins. The density of the solutions can then be measured by counting the number of solutions lying in each bin. As a result, we choose the 28 solutions from the bin with the highest density and use the average of their test RMSE as the final result for MOO-ELM, shown in Table 2.

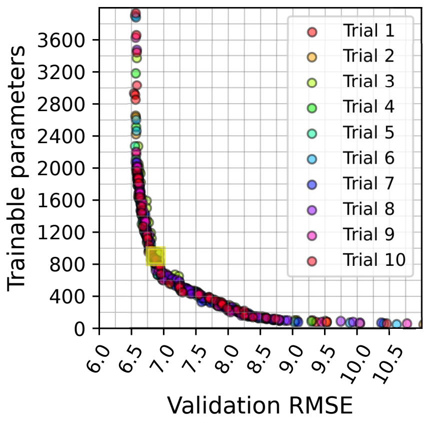

Fig. 4: Trade-off between validation RMSE and number of trainable parameters at the last generation for the 10 independent runs of the proposed MOO-ELM approach (aggregate results across runs). For further analysis, the fitness space is discretized in 20 × 20 bins. The bin highlighted in yellow corresponds to the one with the highest density of solutions.

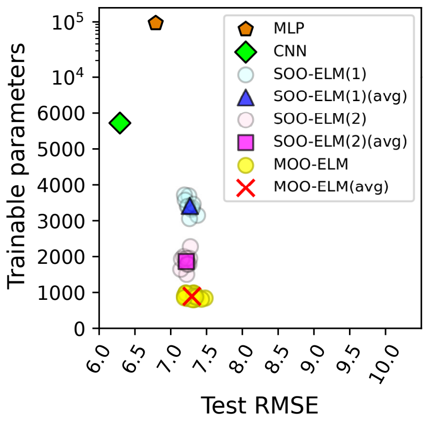

Fig. 5: Trade-off between test RMSE and number of trainable parameters for the methods considered in the experimentation. For SOO-ELM(1), SOO-ELM(2) and MOO-ELM we report the result of each of the 10 available runs, and their average. For MLP and CNN, we report only one solution related to one single run (since their computations are deterministic).

The comparative results of all the considered methods are presented in Table 2. It can be seen that MOO-ELM achieves comparable results to state-of-the-art MLP and CNN models designed by human experts [18]. As mentioned earlier, in the scope of data-driven methods, those deep networks offer indeed state-of-the-art RUL predictions (in terms of test RMSE) on the N-CMAPSS dataset. However, the MLP contains a huge amount of trainable parameter throughout four stacked layers, and training those parameters with an iterative approach requires more than 18 minutes. The CNN shows better prediction accuracy with a lower number of parameters by leveraging parameter sharing, but the training time of this DL architecture is almost twice as big as that of the MLP. Note that the test RMSE values of the BPNNs in Table 2 is different from those reported in [18], which are based on an early version of DS02 that has lower noise level on the sensor readings and a sampling rate of 0.1Hz (instead of 1Hz).

On the contrary, all the optimized ELMs have a considerably smaller number of trainable parameters, which reflects in a much shorter (up to 2 orders of magnitude) training time. The architecture discovered by SOO-ELM(1) tends to have almost the maximum available number of hidden neurons, because it

Table 2: Summary of the comparative analysis. For MLP and CNN, we report only one value related to one single run (since their computations are deterministic). For the proposed methods, SOO-ELM(1), SOO-ELM(2) and MOO-ELM, we report mean ± std. dev. obtained across 10 independent runs. Note that for the proposed methods the worst/median/best architectures are reported in Table 3. The boldface indicates the best result in each column.

Methods	Architecture	Test RMSE (on D_{test})	Trainable parameters	Training time (s)
MLP [18]	4 hidden layers	6.79	94,701	1,081
CNN [18]	3 convolutional layers	**6.29**	5,722	1,969
SOO-ELM(1)	-	7.27±0.05	3,405±202	337±49
SOO-ELM(2)	-	7.21±0.04	1,859±207	110±22
MOO-ELM	-	7.29±0.07	**898±60**	**55±10**

Table 3: Worst, median and best architectures among the best solutions found in each of the 10 runs for each of the three proposed methods, with the corresponding test RMSE and number of trainable parameters. The architectures are denoted by $[n_{tanh}{\cdot}10,\ n_{sigm}{\cdot}10]$ representing the number of tanh and sigmoid hidden neurons, respectively. The L2 regularization parameter r is almost always the same, with a value of 6 for most of the best solutions (5 for the remaining few cases). Thus, we omit its value for brevity.

Methods	Test RMSE			Trainable parameters		
	Worst	Median	Best	Worst	Median	Best
SOO-ELM(1)	[1,840, 1,310]	[1,740, 1,390]	[1,930, 1,760]	[1,930, 1,760]	[1,910, 1,490]	[1,750, 1,310]
SOO-ELM(2)	[1,780, 10]	[1,900, 50]	[1,860, 70]	[1,960, 310]	[1,860, 70]	[1,500, 10]
MOO-ELM	[740, 110]	[790, 80]	[850, 60]	[970, 20]	[840, 20]	[740, 70]

simply uses the validation RMSE as the fitness for its evolutionary search. Yet, in this case the ELM does not suffer from overfitting and its performance does not get worse even if we increase the number of hidden neurons excessively. In other words, the oversized ELM network merely can make a negligible improvement but requires unnecessarily large computational cost for training the redundant parameters. The SOO-ELM(2) approach, on the other hand penalizes those oversized ELM by introducing the penalty factor described in Section 3.2. We set the constant weight τ to 10^{-4}, small enough that the penalty does not dominate the overall fitness. We can see that SOO-ELM(2) successfully prevents the use of the redundant neurons, so that it can preserve the RUL prediction accuracy using only almost half of the neurons, w.r.t. the previous method.

Although SOO-ELM(2) uses almost a half of the neurons and requires less than two minutes of training time, a proper value of τ must be determined by empirical considerations. In contrast, MOO-ELM can overcome this problem using NSGA-II for automatically searching a set of trade-off network architectures without considering a tunable parameter such as the τ used before. Compared to SOO-ELM(2), this method achieves almost the same test RMSE but can further halve the number of trainable parameters. Moreover, it only needs, on average, less than one minute to train the best trade-off networks.

Table 4: Execution time for 30 generations of the three proposed methods (mean $\pm$ std. dev. across 10 independent runs). The boldface indicates the best result.

	SOO-ELM(1)	SOO-ELM(2)	MOO-ELM
Execution time (hours)	16.19$\pm$1.68	9.67$\pm$0.44	**6.43$\pm$0.27**

In addition, MOO-ELM has a clear advantage in terms of execution time. As shown in Table 4, on average, the evolutionary search process of MOO-ELM is much shorter than that of the two SOO-ELM approaches. For SOO-ELM(1), very large ELM models, that take long time to evaluate, tend to survive throughout the generations, because they can have better validation RMSE even though their improvement may be trivial. In contrast, those individuals hardly survive during a run of MOO-ELM, because the NSGA-II algorithm proceeds in the direction of finding better trade-off solutions.

Finally, the comparative results are visualized in Fig. 5, which easily enables to compare the performance of the different methods in terms of trade-off between the two conflicting objectives. We can observe that the CNN dominates the MLP. Among the compared algorithms, MOO-ELM obtains the best solutions in terms of number trainable parameters (on average, about 900, i.e., less than 16% compared to the CNN). Moreover, compared to the CNN, the models discovered by MOO-ELM have an approximately 97% shorter training time, while their test RMSE is on average only 16% larger.

6 Conclusions

In this paper, we applied evolutionary algorithms to search for optimized ELMs for RUL prediction tasks. We considered three methods, two single-objective based on a custom GA (SOO-ELM) and one multi-objective based on NSGA-II (MOO-ELM), and applied them to automatic design ELMs. Our goal was to achieve a trade-off between two competing objectives: the test RMSE and the number of trainable parameters. We compared our methods to state-of-the-art MLP and CNN models from the literature. The comparative evaluation was based on the experimental results on the N-CMAPSS, one of the most up-to-date benchmarks in the area of RUL prediction. The results show that MOO-ELM can search ELMs that are much smaller in size compared to ELMs obtained by optimizing only the RUL prediction error, but have similar prediction performance. Compared to the MLP and CNN, MOO-ELM performs slightly worse in terms of the test RMSE, but the number of trainable parameters is considerably smaller and the training time is significantly shorter. Hence, our work can be used as an efficient RUL prediction tool for industrial applications that require to compromise training time and prediction accuracy.

One major limitation of ELMs is that they comprise a single hidden layer network. To overcome this limitation, multiple hidden layer ELMs (MELMs) have been proposed recently [25]. In future work, we expect to obtain further improvements by applying our method to optimize the architecture of the MELMs. Another interesting future direction would be to employ recurrent extreme learning machines (RELMs) [26]. Similar to this work, we can attempt to optimize the parameters of the RELMs and their architecture in order to achieve higher accuracy while saving time during the training stage.

References

1. Zhang, W., Yang, D., Wang, H.: Data-driven methods for predictive maintenance of industrial equipment: A survey. IEEE Systems Journal **13**(3) (2019) 2213–2227
2. Zheng, C., Liu, W., Chen, B., Gao, D., Cheng, Y., Yang, Y., Zhang, X., Li, S., Huang, Z., Peng, J.: A data-driven approach for remaining useful life prediction of aircraft engines. In: International Conference on Intelligent Transportation Systems (ITSC). (2018) 184–189
3. Babu, G.S., Zhao, P., Li, X.L.: Deep convolutional neural network based regression approach for estimation of remaining useful life. In: International Conference on Database Systems for Advanced Applications (DASFAA), Springer (2016) 214–228
4. Zheng, S., Ristovski, K., Farahat, A., Gupta, C.: Long short-term memory network for remaining useful life estimation. In: International Conference on Prognostics and Health Management (ICPHM), IEEE (2017) 88–95
5. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: a new learning scheme of feedforward neural networks. In: International Joint Conference on Neural Networks (IJCNN). Volume 2., IEEE (2004) 985–990
6. Zheng, S., Ristovski, K., Farahat, A., Gupta, C.: Long short-term memory network for remaining useful life estimation. In: International Conference on Prognostics and Health Management (ICPHM), IEEE (2017) 88–95
7. Li, X., Ding, Q., Sun, J.Q.: Remaining useful life estimation in prognostics using deep convolution neural networks. Reliability Engineering & System Safety **172** (2018) 1 – 11
8. Ye, Z., Yu, J.: Health condition monitoring of machines based on long short-term memory convolutional autoencoder. Applied Soft Computing **107** (2021) 107379
9. Cheng, Y., Hu, K., Wu, J., Zhu, H., Shao, X.: Auto-encoder quasi-recurrent neural networks for remaining useful life prediction of engineering systems. IEEE/ASME Transactions on Mechatronics (2021) 1–1
10. Chen, Z., Wu, M., Zhao, R., Guretno, F., Yan, R., Li, X.: Machine remaining useful life prediction via an attention-based deep learning approach. IEEE Transactions on Industrial Electronics **68**(3) (2021) 2521–2531
11. Mo, H., Custode, L., Iacca, G.: Evolutionary neural architecture search for remaining useful life prediction. Applied Soft Computing **108** (2021) 107474
12. Mo, H., Lucca, F., Malacarne, J., Iacca, G.: Multi-head cnn-lstm with prediction error analysis for remaining useful life prediction. In: 2020 27th Conference of Open Innovations Association (FRUCT), IEEE (2020) 164–171
13. Saxena, A., Goebel, K., Simon, D., Eklund, N.: Damage propagation modeling for aircraft engine run-to-failure simulation. In: Proceedings of the International Conference on Prognostics and Health Management, IEEE (2008) 1–9

14. Arias Chao, M., Kulkarni, C., Goebel, K., Fink, O.: Aircraft engine run-to-failure dataset under real flight conditions for prognostics and diagnostics. Data **6** (2021) 5
15. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**(2) (2002) 182–197
16. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: Nsga-net: neural architecture search using multi-objective genetic algorithm. In: Genetic and Evolutionary Computation Conference (GECCO). (2019) 419–427
17. Yang, Z., Baraldi, P., Zio, E.: A comparison between extreme learning machine and artificial neural network for remaining useful life prediction. In: Prognostics and System Health Management Conference (PHM). (2016) 1–7
18. Arias Chao, M., Kulkarni, C., Goebel, K., Fink, O.: Fusing physics-based and deep learning models for prognostics. Reliability Engineering & System Safety **217** (2022) 107961
19. Huang, G.B., Zhu, Q.Y., Mao, K., Siew, C., Saratchandran, P., Sundararajan, N.: Can threshold networks be trained directly? IEEE Transactions on Circuits and Systems II: Express Briefs **53** (2006) 187 – 191
20. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing **70**(1-3) (2006) 489–501
21. Popoola, S., Misra, S., Atayero, P.A.: Outdoor path loss predictions based on extreme learning machine. Wireless Personal Communications **99** (2018)
22. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (ICLR). (2014)
23. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. Journal of Machine Learning Research - Proceedings Track **9** (2010) 249–256
24. Shang, K., Ishibuchi, H., He, L., Pang, L.M.: A survey on the hypervolume indicator in evolutionary multiobjective optimization. IEEE Transactions on Evolutionary Computation **25**(1) (2020) 1–20
25. Xiao, D., Li, B., Mao, Y.: A multiple hidden layers extreme learning machine method and its application. Mathematical Problems in Engineering **2017** (2017)
26. Ertugrul, O.F.: Forecasting electricity load by a novel recurrent extreme learning machines approach. International Journal of Electrical Power & Energy Systems **78** (2016) 429–435