Data-Driven 5G-Enabled Remaining Useful Life Estimation for Industrial IoT: a Holistic Approach - Deep Learning Models Analysis

Nicolò Longhi, Lorenzo Mario Amorosa, Sara Cavallero, *Graduate Student Member, IEEE*, and Roberto Verdone, *Senior Member, IEEE*

I. Deep learning Models for RUL estimation

In this analysis, we present the deep learning models we trained and tested for the remaining useful life (RUL) estimation. For each model, we indicate the complete list of hyper-parameters investigated, even if only a subset of all their possible combinations has been used. It is worth noting that the iterative threshold optimization algorithm described in the paper has been applied to all models. The training code implemented in TensorFlow 2.10.0 is available on Github[1] and the training data is available on Kaggle[2]

A. Logistic Regression

Logistic regression (LR) [2] is a simple statistical model designed for binary classification. It can be formulated as follows:

$$\hat{y} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n)}}, \tag{1}$$

where $\hat{y}$ is the predicted probability of the target (i.e., the probability of a sample of being anomalous), β_0 is the intercept term, β_i is the coefficients of the input feature x_i.

B. Deep Neural Network

A deep neural network (DNN) [3] is a form of artificial neural network (ANN) comprised of multiple layers of fully connected neurons, where each neuron in a layer receives inputs from all neurons in the preceding layer, and there is no recurrent connection between layers. A generic layer i of a DNN can be described as follows:

$$\mathbf{h}^{(i)} = \sigma(\mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}), \tag{2}$$

where $\mathbf{h}^{(i)}$ is the activations in the i-th layer, $\mathbf{W}^{(i)}$ is the weight matrix in the i-th layer, $\mathbf{b}^{(i)}$ is the bias vector in the i-th layer, $\sigma(\cdot)$ is the activation function (ReLU in this case). The hyper-parameters used are presented in Table I.

Parameter	Search space
Neurons per layer	[32], [64, 32], [128, 64, 32]

TABLE I: Deep neural network hyper-parameters.

C. 1D Convolutional Neural Network

A 1D convolutional neural network (1D-CNN) [4] is a type of deep learning architecture that operates on 1D data, such as time series. It uses a sliding window approach, where a fixed-size kernel moves across the input sequence $\mathbf{x}$, performing an element-wise dot product between the weights $\mathbf{w}$ of the kernel and the values of $\mathbf{x}$ within the window. The result of these dot products, known as feature maps, are then combined through an activation function $\sigma(\cdot)$ and bias term $\mathbf{b}$ to produce new features, which are eventually processed through a fully connected DNN to produce the output. Possibly, more than one kernel can be applied by increasing the filter size. The main advantages of this architecture are: 1) lower computation compared to fully connected layers, 2) good ability to learn local patterns in input sequences. An output feature $f(\mathbf{x_i})$ can be computed as follows:

$$f(\mathbf{x_i}) = \sigma(\sum_{j=0}^{k-1} \mathbf{w_j}\mathbf{x_{i+j}} + \mathbf{b}), \tag{3}$$

where $\mathbf{x_i}$ is the input sequence at index i, $\mathbf{w}$ is the kernel with k elements, and $\mathbf{b}$ is the bias vector. The hyper-parameters used are presented in Table II. Notably, for all the tested margins $m \in \{5, 10\}$, the 1D-CNN obtained the best performance using almost the same set of hyper-parameters, as shown in Table III.

Parameter	Search space
Neurons per dense layer	[32], [64, 32], [128, 64, 32]
Filter size	1, 4
Kernel size	3, 5, 7
Input sequence length	5, 10

TABLE II: 1D convolutional neural network hyper-parameters.

An earlier version of this paper was presented at the 2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC) Conference and was published in its Proceedings, DOI: 10.1109/PIMRC56721.2023.10293815 [1].

N. Longhi, L.M. Amorosa, S. Cavallero, and R. Verdone are with the Department of Electrical, Electronic and Information Engineering (DEI), "Guglielmo Marconi", University of Bologna & WiLab - National Wireless Communication Laboratory (CNIT), Italy. E-mail: {nicolo.longhi, lorenzomario.amorosa, s.cavallero, roberto.verdone}@unibo.it

The Ph.D. of Nicolò Longhi has been funded by Telecom Italia S.p.A.

[1]Code accessible at: https://github.com/Lostefra/5G_IoT_AGV_RUL_prediction.

[2]Data available at: https://www.kaggle.com/datasets/lorenzoamorosa/5g-industrial-iot-for-remaining-useful-life.

Parameter	Value
Neurons per dense layer	[64, 32]
Filter size	1
Kernel size	3
Input sequence length	5

TABLE III: 1D convolutional neural network hyper-parameters that allow to achieve best performance for margins $m \in \{5, 10\}$.

D. Autoencoder

An autoencoder (AE) [5] is a type of ANN which consists of two main components: an encoder and a decoder. The encoder maps the input data to a lower-dimensional representation, known as the latent representation or encoding. The decoder then maps the encoding back to the original data space to reconstruct the input. This model can been applied to RUL estimation by training the model on data related to automated guided vehicle (AGV)'s normal operating conditions and using the reconstruction error as a measure of abnormality for new unseen data. The idea behind this application is that data points related to normal operating conditions should have a low reconstruction error, while those related to anomalous conditions will have a high reconstruction error. An AE is typically represented as:

$$\text{Encoder:} \quad \mathbf{z} = f_{\text{enc}}(\mathbf{x}), \tag{4}$$

$$\text{Decoder:} \quad \hat{\mathbf{x}} = f_{\text{dec}}(\mathbf{z}), \tag{5}$$

where $\mathbf{x}$ is the input data, f_{enc} is the encoding function (i.e., a DNN), $\mathbf{z}$ is the encoding (latent representation), f_{dec} is the decoding function (i.e., a DNN), and $\hat{\mathbf{x}}$ is the reconstruction of the input data. The hyper-parameters used are presented in Table IV.

Parameter	Search space
Neurons per layer	[16, 8, 2, 8, 16], [64, 24, 9, 24, 64], [128, 56, 18, 56, 128]

TABLE IV: Autoencoder hyper-parameters.

E. Long Short-Term Memory

An long short term memory (LSTM) [6] model is a type of recurrent neural network (RNN) designed to handle the vanishing gradients problem in traditional RNNs and it is commonly used in tasks that require processing of sequential data. The LSTM model contains a memory cell that is able to store information for an extended period of time, and gates that control the flow of information into and out of the cell. The input gate i_t decides how much new information to add to the cell state c_t, the forget gate f_t decides how much of the previous information to discard, and the output gate o_t decides what information to output from the cell. These gates are controlled by activation functions such as the sigmoid function. In each time step, the LSTM model takes the current input x_t and hidden state h_{t-1} as input, and computes the new hidden state and output. The hidden state h_t can be thought of as a summary of the information processed so far, and the output o_t is used for prediction or other purposes. The new hidden state is a function of the current input, the previous hidden state, and the gates. An LSTM can be formulated as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i), \tag{6}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f), \tag{7}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o), \tag{8}$$

$$\tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \tag{9}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{10}$$

$$h_t = o_t \odot \tanh(c_t), \tag{11}$$

where x_t is the input vector at time step t, h_{t-1} is the hidden state vector at time step $t-1$, i_t, f_t, o_t are the input, forget, and output gates respectively, c_t is the cell state vector at time step t, $\tilde{c}_t$ is the candidate cell state vector at time step t, $\sigma(\cdot)$ is the sigmoid function, $\tanh(\cdot)$ is the hyperbolic tangent function, $\odot$ is the element-wise multiplication (Hadamard product), W_{xi}, W_{hi}, W_{xf}, W_{hf}, W_{xo}, W_{ho}, W_{xc}, W_{hc} are the weight matrices, and b_i, b_f, b_o, b_c are the bias vectors. The hyper-parameters used are presented in Table V.

F. Bidirectional Long Short-Term Memory

Bi-directional long short term memory (BiLSTM) [6] is an extension of the LSTM model. It allows the LSTM model to capture both past and future context by processing input data in two directions, forward and backward. Briefly, BiLSTM adds one more LSTM layer, which reverses the direction of information flow. The output of the BiLSTM model is then computed based on the concatenation of the hidden states of the forward and backward LSTMs. The hyper-parameters used are presented in Table V.

G. Gated Recurrent Unit

Gated recurrent unit (GRU) [6] is a type of recurrent neural network (RNN) that processes sequential data. It is designed to handle the issue of vanishing gradients in traditional RNNs by controlling the flow of information through gates, which regulate the information being passed from one time step to the next. They have become a popular alternative to LSTMs due to their simplicity and efficiency. A GRU has two gates, the update gate z_t and the reset gate r_t, that control how much of the previous hidden state h_{t-1} and the current input x_t is passed to the current hidden state h_t. The z_t decides how much of h_{t-1} should be retained, while the r_t decides how much of h_{t-1} should be forgotten. The candidate hidden state $\tilde{h}_t$ is computed using a combination of h_{t-1}, scaled by r_t and x_t. The final hidden state h_t is then computed by taking a weighted sum of $\tilde{h}_t$ and h_{t-1}, with the weights determined by the update gate. A GRU can be formulated as follows:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z), \tag{12}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r), \tag{13}$$

$$\tilde{h}_t = \tanh(W_h \cdot ([r_t \odot h_{t-1}], x_t) + b_h), \tag{14}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t, \tag{15}$$

where W_z, W_r, W_h are the weight matrices, b_z, b_r, b_h are the bias vectors, $\sigma(\cdot)$ is the sigmoid activation function, $\tanh(\cdot)$ is the hyperbolic tangent activation function, x_t is the input at time step t, h_{t-1} is the hidden state at time step $t-1$, z_t and r_t are the update and reset gates, respectively, $\tilde{h}_t$ is the

candidate hidden state, and h_t is the new hidden state. The hyper-parameters used are presented in Table V.

Parameter	Search space
Neurons per dense layer	[64], [128]
Input sequence length	5, 10

TABLE V: LSTM, BiLSTM, and GRU hyper-parameters.

H. Vanilla Transformer

Firstly introduced in [7], the vanilla transformer is a powerful deep learning model architecture that revolutionized natural language processing (NLP) tasks. This model primarily focuses on solving sequence-to-sequence problems, such as machine translation and text generation. Its key innovation lies in its attention mechanism, which allows the model to capture dependencies between different positions in the input sequence. We recommend to read [7] for a complete discussion about transformers; however, we treat the focal key points in this appendix.

The vanilla transformer consists of an encoder and a decoder, each composed of multiple layers. The encoder layer, referred to as "Enc", is represented as follows:

$$\text{Enc}(\mathbf{x}) = \text{LN}(\mathbf{x} + \text{MHA}(\mathbf{x})), \tag{16}$$

where $\text{LN}(\cdot)$ is the layer normalization, $\text{MHA}(\cdot)$ is the multi-head attention, and $\mathbf{x}$ is the input sequence. Conversely, the decoder layer, referred to as "Dec", is represented as follows:

$$\text{Dec}(\mathbf{y}, \text{Enc}(\mathbf{x})) = \text{LN}(\mathbf{y} + \text{MHA}(\mathbf{y}) + \text{MHA}(\mathbf{y}, \text{Enc}(\mathbf{x})). \tag{17}$$

The multi-head attention mechanism, which is the crucial component of transformers, is defined as:

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h)\text{W}_O. \tag{18}$$

Notably, the shorthand $\text{MHA}(x)$ can used for brevity, assuming Q, K, and V share input x. In this case, we refer to self-attention. Afterwards, each head is calculated as:

$$\text{head}_i = \text{ATT}(Q\text{W}_{Qi}, K\text{W}_{Ki}, V\text{W}_{Vi}), \tag{19}$$

and the attention function $\text{ATT}(\cdot)$ is defined as:

$$\text{ATT}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \tag{20}$$

Here, Q, K, and V denote the query, key, and value matrices, respectively. W_{Qi}, W_{Ki}, and W_{Vi} represent the learnable weight matrices for each head. d_k is the dimensionality of the key vectors. W_O is also a learnable matrix.

The layer normalization applies normalize layer-wise the input sequence, which helps in stabilizing the training process.

During training, the vanilla transformer utilizes masked self-attention in the decoder to prevent attending to future positions. This is represented as:

$$\text{MaskedATT}(Q, K, V) = \text{softmax}\left(\frac{Q(K^\top \odot M)}{\sqrt{d_k}}\right)V, \tag{21}$$

where $\text{MaskedATT}(\cdot)$ implements the masked self-attention, $\odot$ denotes element-wise multiplication, and M is a binary mask matrix with the upper triangular part set to a large negative value.

Finally, to enable the model, which lacks recurrence and convolution, to effectively utilize the order of the input sequence, positional encodings are introduced. These encodings are added to the input embeddings at the bottom of the encoder and decoder stacks. The positional encoding is represented as:

$$\text{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right), \tag{22}$$

$$\text{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right), \tag{23}$$

where pos represents the position in the sequence and i represents the dimension of the positional encoding. d is the model dimension. The hyper-parameters used are presented in Table VI.

Parameter	Search space
Number of Transformer's blocks	2, 3, 4
Heads size	128, 256
Number of heads	2, 3, 4
Neurons per dense layer	[64], [128], [128, 64]
Input sequence length	5, 10

TABLE VI: Vanilla transformer hyper-parameters.

ACKNOWLEDGMENT

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART"). Authors wish to thank BI-REX Competence Center for Industry 4.0 for the availability of the experimental environment.

REFERENCES

[1] L.M. Amorosa, N. Longhi *et al.*, "An end-to-end analysis of deep learning-based remaining useful life algorithms for satefy-critical 5G-enabled IIoT networks," in *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2023, pp. 1–6.

[2] J. Phillips, E. Cripps *et al.*, "Classifying machinery condition using oil samples and binary logistic regression," *Mechanical Systems and Signal Processing*, pp. 316–325, 2015.

[3] J.C.B. Gamboa, "Deep learning for time-series analysis," 2017.

[4] H.I. Fawaz, G. Forestier *et al.*, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, pp. 917–963, mar 2019.

[5] A. Borghesi, A. Bartolini *et al.*, "Anomaly detection using autoencoders in high performance computing systems," *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 9428–9433, jul 2019.

[6] Y. Wu, M. Yuan *et al.*, "Remaining useful life estimation of engineered systems using vanilla LSTM neural networks," *Neurocomputing*, pp. 167–179, 2018.
[7] A. Vaswani, N. Shazeer *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.