



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

CONVOLUTIONAL RECURRENT NEURAL NETWORKS FOR REMAINING USEFUL LIFE
PREDICTION IN MECHANICAL SYSTEMS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MECÁNICO

NICOLAS OYHARÇABAL ASTORGA

PROFESOR GUÍA:
ENRIQUE ANDRÉS LÓPEZ DROGUETT

MIEMBROS DE LA COMISIÓN:
VIVIANA MERUANE NARANJO
PATRICIO LONCOMILLA ZAMBRANA

SANTIAGO DE CHILE
2018

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: NICOLAS OYHARÇABAL ASTORGA
FECHA: 19/03/2018
PROF. GUÍA: ENRIQUE ANDRÉS LÓPEZ DROGUETT

CONVOLUTIONAL RECURRENT NEURAL NETWORKS FOR REMAINING USEFUL LIFE PREDICTION IN MECHANICAL SYSTEMS

La determinación de la vida útil remanente (RUL del inglés "Remaining Useful Life") de una máquina, equipo, dispositivo o elemento mecánico, es algo en lo que se ha estado trabajando en los últimos años y que es crucial para el futuro de cualquier industria que así lo requiera. El continuo monitoreo de máquinas junto a una buena predicción de la RUL permite la minimización de costos de mantención y menor exposición a fallas. Sin embargo, los datos obtenidos del monitoreo son variados, tienen ruido, poseen un carácter secuencial y no siempre guardan estricta relación con la RUL, por lo que su estimación es un problema difícil. Es por ello que en la actualidad se utilizan distintas clases de Redes Neuronales y en particular, cuando se quiere modelar problemas de carácter secuencial, se utilizan las Redes Neuronales Recurrentes o RNN (del inglés "Recurrent Neural Network") como LSTM (del inglés "Long Short Term Memory") o JANET (del inglés "Just Another NETwork"), por su capacidad para identificar de forma autónoma patrones en secuencias temporales, pero también junto a estas últimas redes, también se utilizan alternativas que incorporan la Convolución como operación para cada célula de las RNN y que se conocen como ConvRNN (del inglés "Convolutional Recurrent Neural Network"). Estas últimas redes son mejores que sus pares convolucional y recurrentes en ciertos casos que requieren procesar secuencias de imágenes, y en el caso particular de este trabajo, series de tiempo de datos de monitoreo que son suavizados por la Convolución y procesados por la Recurrencia.

El objetivo general de este trabajo es determinar la mejor opción de ConvRNN para la determinación de la RUL de un turbofan a partir de series de tiempo de la base de datos C-MAPSS. También se estudia cómo editar la base de datos para mejorar la precisión de una ConvRNN y la aplicación de la Convolución como una operación primaria en una serie de tiempo cuyos parámetros muestran el comportamiento de un turbofan. Para ello se programa una LSTM Convolucional, LSTM Convolucional Codificador-Decodificador, JANET Convolucional y JANET Convolucional Codificador-Decodificador. A partir de esto se encuentra que el modelo JANET Convolucional Codificador-Decodificador da los mejores resultados en cuanto a exactitud promedio y cantidad de parámetros necesarios (entre menos mejor pues se necesita menos memoria) para la red, siendo además capaz de asimilar la totalidad de las bases de datos C-MAPSS. Por otro lado, también se encuentra que la RUL de la base de datos puede ser modificada para datos antes de la falla. Para la programación y puesta en marcha de las diferentes redes, se utilizan los computadores del laboratorio de Integración de Confiabilidad y Mantenimiento Inteligente (ICMI) del Departamento de Ingeniería Mecánica de la Universidad de Chile.

Abstract

The determination of the Remaining Useful Life (RUL) of a machine, equipment, device or mechanical element, is a crucial issue for the future of the industry and the optimization of processes as in the case of maintenance. The continuous monitoring of machines along with a good prediction of the RUL, allows the minimization of maintenance costs and lower exposure to catastrophic faults. On the other hand, it is also known that data obtained from monitoring is varied, has a sequential nature and do not always has a strict relationship with RUL, so their estimation is difficult problem.

Nowadays in this class of problems, different kinds of Neural Networks are used. In particular when it is wanted to model problems with sequential data, Recurrent Neural Network (RNN) are preferred for its capacity to autonomously identify patterns in temporal sequences and recently, there are also alternatives that incorporate the Convolution as an operation in each cell of these Networks. Therefore these Networks in some cases are better than their convolutional and recurrent pairs, since they are capable of processing sequences of images, and in the particular case of this work, time series of monitoring data that are softened by convolution and processed by recurrence.

The general objective of this work is to obtain the best alternative based on Convolutional Recurrent Neural Network (ConvRNN) for determining the RUL of a turbofan from the time series of C-MAPSS dataset. It is also studied how to modify the database to improve the accuracy of a ConvRNN and the application of Convolution as a primary operation in a time series whose parameters show the behavior of a turbofan. For this, a Convolutional LSTM, Convolutional LSTM Encoder-Decoder, Convolutional JANET and Convolutional JANET Encoder-Decoder are programmed. Then, it is proven that the model that obtains the best results in terms of average accuracy and number of parameters necessary (less is better because less memory is needed) for the Network is the Convolutional JANET Encoder-Decoder that is also able to successfully assimilate the totality of the C-MAPSS databases. Moreover, it is also found as in other works, that the RUL from the database can be modified for data before failure.

For the start-up of Neural Networks, computers from the SRMI laboratory of the Mechanical Engineering Department of the University of Chile are used.

Dedicado a ti mamá, sin ti todo hubiese sido imposible.

Agradecimientos

En primer lugar quiero agradecer a mi mamá Elsa Astorga Sepúlveda quien siempre me ha mantenido, aconsejado, ayudado, enseñado, y los verbos no bastan para describir cuanto has hecho por mi en todo lo que llevo de vida. Sin tu ejemplo nunca hubiera logrado ninguna de mis metas que tu también me fomentaste.

También quiero agradecer Eduardo Pardo quien fue el responsable de que aprendiera matemática cuando estaba en colegio y quien siempre me ha aconsejado y estado dispuesto a ayudarme en lo que sea. Creeme, no hay día en que no me cuestione que hubiese pasado si usted no me hubiese ayudado.

Agradezco a Patricio Loncomilla, investigador del departamento de ingeniería eléctrica, quien siempre estuvo dispuesto a resolver todas mis dudas, me ayudó en diversos temas de informática que me han interesado a lo largo de mi carrera y, quien me revisó y ayudó a corregir este trabajo.

Agradezco a los profesores Enrique Lopez y Viviana Meruane quienes me presentaron los usos del Aprendizaje Profundo en ingeniería mecánica, estuvieron dispuestos a conversar de forma concisa acerca de las diversas posibilidades que esta tecnología trae en esta nuestra área y, quienes me revisaron y ayudaron a corregir este trabajo.

Agradezco a todos los profesores quienes me han aportado con sus conocimientos, ideas y formas de ver el mundo. Todo ello ha sido base para mi aprendizaje y ambiciones.

Finalmente quiero agradecer a mis amigos Cuca, Frodo, Camilo, Carlos, Joaquin, Esteban, Alvaro, Basti, Guille, Marc, Pati y todos aquellos compañeros de universidad con quienes he compartido, en las buenas y en las malas, momentos sumamente valiosos e imposibles de olvidar.

A todos muchas gracias.

Contents

PART I: Basic ideas.	1
1. Introduction	1
1.1. Motivation	1
1.2. General objectives	2
1.3. Specific objectives	2
1.4. Scope	2
1.5. Outline	2
	3
2. Related Work	4
2.1. Diagnosis and Prognosis problems	4
2.2. Use of operational data for the study of faults	5
2.2.1. Analytic Methods	5
2.2.2. Datasets and machine learning methods	6
2.2.3. Deep learning and Data-driven prognostic algorithms	6
	9
PART II: Deep Neural Networks and Database.	9
3. Fundamentals	9
3.1. Fundamentals	10
3.1.1. Tensors	10
3.1.2. Neural Network and Deep Neural Network	10
3.2. Feedforward Neural Network	14
3.2.1. Perceptron	14
3.2.2. Multilayer Perceptron	15
3.2.3. Convolutional Neural Network	16
3.3. Recurrent Neural Network	19
3.3.1. Long Short Term Memory	22
3.3.2. JANET	23
3.3.3. Encoder-Decoder Recurrent Architectures	25
3.3.4. Convolutional LSTM	26

3.3.5. Convolutional JANET	28
3.4. Training and Optimizers	30
3.4.1. Loss function	30
3.4.2. Back-propagation	31
3.4.3. Back-propagation Through-Time	31
3.4.4. Optimization	31
3.4.5. Initializators	32
	32
4. Turbofan RUL Estimation	33
4.1. Dataset's	33
4.1.1. C-MAPSS dataset's	33
4.1.2. C-MAPSS dataset evaluation formulas	35
PART III: Approach and Application.	36
5. Problem Statement	36
	38
6. Methodology and Experiments	39
6.1. Neural Networks design	39
6.1.1. Application of convolution	39
6.1.2. Number of parameters and size of datasets	41
6.2. Use of C-MAPSS database	41
6.2.1. Joining train-sets	41
6.2.2. Data preprocesing	42
6.3. Programming and start-up of a ConvRNN	45
6.4. Hiperparameters setting	45
	46
7. Results	47
7.1. Dataset Histograms	48
7.2. Training settings	50
7.3. Recurrent convolutional Neural Network models and their variations Encoder-Decoder.	50
7.4. Testing results	51
7.5. Behavior of the models in databases.	53
7.5.1. Adjustment and predictions of ConvLSTM in FD00 1, 2, 3 and 4.	54
7.5.2. Adjustment and predictions of ConvJANET in FD00 1, 2, 3 and 4.	58
7.5.3. Adjustment and predictions of ConvLSTM Encoder-Decoder in FD00 1, 2, 3 and 4.	62

7.5.4. Adjustment and predictions of ConvJANET Encoder-Decoder in FD00 1, 2, 3 and 4.	66
	69
8. Discussion	70
8.1. Problem feasibility	70
8.2. Performance of Neural Networks and related aspects	71
8.3. Models convergence	73
	74
9. Conclusion	75
Acronyms	77
Nomenclature	78
Bibliography	79
Appendix	83

List of Tables

4.1. C-MAPSS dataset's specifications.	34
6.1. C-MAPSS dataset's unions.	42
7.1. Overview of the training parameters proposed and obtained from validation. It is joined some datasets for the evaluation of the networks in the four dataset but with three datasets for training. More details in section 6.2.1 . . .	50
7.2. Models found for ConvLSTM, ConvJANET, ConvLSTM Encoder-Decoder, ConvJANET Encoder-Decoder cases.	51
7.3. ConvLSTM, ConvJANET and their varieties Encoder-Decoder evaluated in FD001.	52
7.4. ConvLSTM, ConvJANET and their varieties Encoder-Decoder evaluated in FD002.	52
7.5. ConvLSTM, ConvJANET and their varieties Encoder-Decoder evaluated in FD003.	52
7.6. ConvLSTM, ConvJANET and their varieties Encoder-Decoder evaluated in FD004.	52

List of Figures

3.1. Sigmoid activation functions.	11
3.2. Hyperbolic tangent activation function.	12
3.3. ReLU	12
3.4. ELU	13
3.5. Perceptron for classification for 5 classes.	14
3.6. Multilayer Perceptron of 2 hidden layers.(Modified from: http://blog.robfelty.com/2007/02/14/pgf-gallery ,last time consulted 06-09-2018)	15
3.7. Example of CNN structure with one convolutional layer for image classification. (Based on [LeC+98, p. 2284], own elaboration).	16
3.8. Zero padding in a sequential matrix in which kernels of size $(D_{kernel}, 1)$ are applied. (Own elaboration).	18
3.9. Recurrent network cell. Typical cyclic graph <i>(a)</i> that can be unrolled into <i>(b)</i> as an acyclic graph.(Own elaboration).	20
3.10.Recurrent Neural Network with sequence input-output one to many.(Own elaboration).	20
3.11.Recurrent Neural Network with sequence input-output many to one.(Own elaboration).	21
3.12.Recurrent Neural Network with sequence input-output many to many.(Own elaboration).	21
3.13.Flux of information in LSTM cells. Similar to RNNs, LSTMs have output and hidden states. (Own elaboration).	23
3.14.Flux of information in JANET cells. Different from RNN or LSTM cells, Janet cells don't have an output but only a hidden state. (Own elaboration).	24
3.15.Structure of an unconditional recurrent Encoder-Decoder. The inputs are processed in the encoder cells and unfolded in the decoder cells which initial state (IS) is equal to the last hidden state of the encoder. (Own elaboration).	25
3.16.Structure of a fully-connected ConvLSTM for time window input. (Own elaboration).	27
3.17.Encoder-Decoder structure of ConvLSTM for time series input. (Own elaboration).	28

4.1. Example and description of a turbofan. A model of Pratt Whitney F100, manufactured in 1970 and propeller of fighters F-15 Eagle and F-16 Fighting Falcon. (image taken from https://www.grc.nasa.gov/www/k-12/Missions/Jim/Project2_act.htm , last time consulted 06-09-2018)	34
6.1. Application of convolution between a time series and a kernel of a network that operates with convolution. The filter is only applied in the time dimension to which zero-padding is added (dotted in blue) at its start and end ends of the series. (own elaboration)	40
7.1. Histogram of FD001 dataset when its RULs have not been edited.	48
7.2. Histogram of FD002 dataset when its RULs have not been edited.	48
7.3. Histogram of FD003 dataset when its RULs have not been edited.	49
7.4. Histogram of FD004 dataset when its RULs have not been edited.	49
7.5. Accuracy of validation and training sets vs training steps for ConvLSTM in FD001.	54
7.6. RUL predicted for ConvLSTM in FD001.	54
7.7. Accuracy of validation and training sets vs training steps for ConvLSTM in FD002.	55
7.8. RUL predicted for ConvLSTM in FD002.	55
7.9. Accuracy of validation and training sets vs training steps for ConvLSTM in FD003.	56
7.10. RUL predicted for ConvLSTM in FD003.	56
7.11. Accuracy of validation and training sets vs training steps for ConvLSTM in FD004.	57
7.12. RUL predicted for ConvLSTM in FD004.	57
7.13. Accuracy of validation and training sets vs training steps for ConvJANET in FD001.	58
7.14. RUL predicted for ConvJANET in FD001.	58
7.15. Accuracy of validation and training sets vs training steps for ConvJANET in FD002.	59
7.16. RUL predicted for ConvJANET in FD002.	59
7.17. Accuracy of validation and training sets vs training steps for ConvJANET in FD003.	60
7.18. RUL predicted for ConvJANET in FD003.	60
7.19. Accuracy of validation and training sets vs training steps for ConvJANET in FD004.	61
7.20. RUL predicted for ConvJANET in FD004.	61
7.21. Accuracy of validation and training sets vs training steps for ConvLSTM Encoder-Decoder in FD001.	62
7.22. RUL predicted for ConvLSTM Encoder-Decoder in FD001.	62
7.23. Accuracy of validation and training sets vs training steps for ConvLSTM Encoder-Decoder in FD002.	63

7.24. RUL predicted for ConvLSTM Encoder-Decoder in FD002.	63
7.25. Accuracy of validation and training sets vs training steps for ConvLSTM Encoder-Decoder in FD003.	64
7.26. RUL predicted for ConvLSTM Encoder-Decoder in FD003.	64
7.27. Accuracy of validation and training sets vs training steps for ConvLSTM Encoder-Decoder in FD004.	65
7.28. RUL predicted for ConvLSTM Encoder-Decoder in FD004.	65
7.29. Accuracy of validation and training sets vs training steps for ConvJANET Encoder-Decoder in FD001.	66
7.30. RUL predicted for ConvJANET Encoder-Decoder in FD001.	66
7.31. Accuracy of validation and training sets vs training steps for ConvJANET Encoder-Decoder in FD002.	67
7.32. RUL predicted for ConvJANET Encoder-Decoder in FD002.	67
7.33. Accuracy of validation and training sets vs training steps for ConvJANET Encoder-Decoder in FD003.	68
7.34. RUL predicted for ConvJANET Encoder-Decoder in FD003.	68
7.35. Accuracy of validation and training sets vs training steps for ConvJANET Encoder-Decoder in FD004.	69
7.36. RUL predicted for ConvJANET Encoder-Decoder in FD004.	69
8.1. Fully-connected ConvJANET Encoder-Decoder that presents the minor RMSEs and Scores. The model consists of a first CNN layer of 2 size filters (15,1), then a ConvJANET Encoder from which hidden states are copied and given to the ConvJANET Decoder as initial states. Finally a fully-connected layer, with Tahn activation function, make a prediction. (Own elaboration). . .	73

1 | Introduction

1.1. Motivation

The Remaining Useful Life (RUL) of a machine is a subjective estimation of the time remaining until a machine, component or system can continue working before warranting replacement. Being a subjective measure, depends on what is expected from the operation of the machine or system's component. Is for this reason, that the quantity and complexity of the parameters (such as temperature, pressure, vibration level, etc.) that show machine's behavior, always change case by case. Some of the possible complex aspects of these parameters could be:

- Different operating conditions.
- Various failure modes.
- White noise over the data.
- Variable amounts of data available.

If a machine, component or system is reached the end of its useful life, the parameters that measure the behavior of a machine also show the effect of the failure or faults in its magnitudes. Therefore, it is necessary to ask whether the magnitudes of parameters in a given time (within useful life) can be related to a possible future failure or faults and, if a time series made from these parameters can contain enough information for the estimation of time remaining until the failure. (it is discussed these two questions in section 8.1).

From these two questions it is considered the existence of a non-linear function that is capable of processing machine's parameters for RUL estimation. An analytic function would be complex and costly to do, and it would also change case by case depending on each problem. Then, the option that can be understood as enough generalist, robust, fast and reason for this work, is to approximate a maximum value of a conditional probability, between RUL and the parameters related to a particular time cycle, with a probabilistic approach of Convolutional Recurrent Neural Network (ConvRNN).

1.2. General objectives

Determine the best option of Convolutional Recurrent Neural Network (ConvRNN) for estimate the Remaining Useful Life (RUL) of a turbofan from the time series of measured data of its sensors.

1.3. Specific objectives

- Programming and start-up of varieties of ConvRNN
- Determine how to edit the database to improve the accuracy of a ConvRNN.
- Study the application of Convolution as a primary operation in a series of time whose parameters show the behavior of a turbofan.
- Determine the architecture of ConvRNN that presents the highest precision and score.

1.4. Scope

- Programming Convolutional Long Short Term Memory (ConvLSTM) that achieves high precision.
- Programming ConvLSTM Encoder-Decoder network that achieves high precision.
- Programming Convolutional Just Another Network (ConvJANET) programming that achieves high precision.
- Programming ConvJANET Encoder-Decoder network programming that achieves high precision.

1.5. Outline

It is started with Chapter 2 showing the related work that allows to see the development of the research in Prognosis and Health Management (PHM). Then, in Chapter 3 a complete introduction to the theory of Neural Networks is shown. In Chapter 4 it is shown the C-MAPSS database, mentioning its main characteristics and justification. In Chapter 5 it is formally introduced the problem of using Neural Networks for the approximation of RULs of a machine. The methodology and experiments done in this work are described

in Chapter 6 and then the results are shown in Chapter 7. The results are discussed in Chapter 8 and finally this work concludes with Chapter 9.

2 | Related Work

In this chapter it is reviewed the data-driven algorithmic methods used for solve Diagnosis and Prognosis problems in the context of faults in mechanical systems. It is started with a bibliographic review carried out as proposed [Saxena et al., 2008], and then proceeds to the explanation of the most recent methods in which deep learning algorithms are used, making relevant observations about these works.

According to [Saxena et al., 2008], the problem of Diagnosis can be solved from an approach that does not consider the materials from which a mechanical system is made but using the data corresponding to the operation of this system. These data reflect the health state of a given component or mechanical system.

The methods for the resolution of problems of Diagnosis and Prognosis from the approach of using operational data, can be divided among those that consider the physics of the system in which it is interested, i.e, analytic methods in which a function is obtained for the extraction of characteristics and subsequent prediction of a pseudo-parameter. And by other hand, it is also had methods of unknown physical foundations but based on the statistics in which a function is determined.

2.1. Diagnosis and Prognosis problems

The problem of Diagnosis can be defined as the problem of determining whether or not a component or mechanical system is failing, while the problem of Prognosis can be understood as the problem of determining how many useful life a mechanical component or system has. Both problems are closely related possessing unique peculiarities due to the nature of the faults in mechanical systems, one of these peculiarities is that before a failure of a mechanical system, this shows parameters close to "nominal" magnitudes [Ramasso, 2014], [Goebel et al., 2007] which do not have much variability if similar operating conditions are taken into account, however at the moment of a failure occur immediately that some system parameters are different from the nominal ones. Therefore, it can be seen that particularly in these problems it is had an extensive amount of time

where data is little variable while, since the moment of failure, the data provides useful information to solve the problems of Diagnosis and Prognosis.

For the reasons mentioned above, trying to solve the problem of Prognosis before failure is a complicated process if data-driven methods are used and many times, they are not more successful than the extensive statistical methods [Rausand and Høyland, 2013] or empirical applications as described in [Neubauer, 1984].

However it is noteworthy that the complexity of Diagnosis and Prognosis problems not only lies on the physics or behavior of the systems, but also on the uncertainty of other external variables such as noise on sensors or operational conditions.

2.2. Use of operational data for the study of faults

2.2.1. Analytic Methods

The study of faults in systems and mechanical components together with development of useful algorithms, have been closely related to the use of databases of different types. The use of databases that represent historical data of a mechanical system does not focus on the initial conditions of materials and applications.

The proposal to use the operational data for the study of faults is given in the seventies by [Urban, 1973] in which is proposed the usefulness of algorithms that using operational data (found in control systems) are able to determine when a component or system fails. This work was followed by [Doel, 2002] and [Kurosaki, 2003]. [Urban, 1973] and [Doel, 2002] use a weighted-least squares approach looking for the solution -fault Diagnosis- that further reduces the error, meanwhile in [Kurosaki, 2003], the approach uses average magnitudes of pseudo-parameters so that the fixed variable that minimizes error is the one that indicates the fault.

These solutions not only take into account the behavior of the data as random variables from which it is sought to determine a state or failure, but also take into account the physical relationship between different variables and that are reflected in the thermodynamic formulas of a turbine. Also noteworthy is the fact that the solutions given by their algorithms always allude to the case that minimizes a system from physical equations, using them for the previous extraction of characteristics and subsequent resolution of an optimization problem.

The approaches given at the beginning by Urban and Doel explicitly point out the stochastic nature of the problem to be solved with notable quotes about their model as “the solution providing the greatest reduction to the residual error is used in place of the base

solution if the fault solution probability is sufficiently large”, and as it was seen in the past Chapter 2, is the way in which the training of Neural Networks is also considered.

2.2.2. Datasets and machine learning methods

The study of either Diagnosis or Prognosis for mechanical systems, with evidence also requires the creation and disposal of large databases that are not always available. That is why, in the context of the 2008 International Conference on Prognostics and Health Management, the PHM-2008 database is made available [Saxena et al., 2008] and later also the C-MAPSS database. Both are generated from the software Comercial Modular Aero-Propulsion System Simulation (C-MAPSS) in which turbofans are modeled for the generation of databases. For more details about these 2 databases see Chapter 3.

From the aforementioned databases, as [Ramasso and Saxena, 2014] say, there are more than 70 publications on different study possibilities, these studies do not necessarily deal with the problem of Prognosis but also address issues of machine learning and deep learning as supervised classification, unsupervised classification and partially supervised classification.

2.2.3. Deep learning and Data-driven prognostic algorithms

For the understanding of the usefulness of the machine learning and deep learning models on the study of faults, it can be first mentioned what these models are capable of modeling and if the problem is feasible to be modeled.

As it was seen earlier in the case of Diagnosis at the beginning of the fault study, there are often relationships between the measured parameters that are already known and given in some cases by the same equations that govern the physics of the machines to be studied. Moreover, it is also true that from these relationships the first studies were able to solve the diagnostic problem with a certain range of success. So then, it can be said that there are functions that with a certain range of success are capable of solving the Diagnosis problem. However, together with the physics given in the operating parameters of the mechanical systems, the characteristics that allow the problems of Diagnosis and Prognosis to be stochastic problems must also be taken into account, either because they are considered uncertain operation conditions, fails or the same noise over the data as mentioned above.

Therefore, it should be understood that the use of Neural Networks on prognostic problems is justified because it is not only a deterministic problem but also stochastic, that Neural Networks reproduce a conditional probability distribution [Goodfellow et al., 2016],

[Bishop, 2006], [Murphy, 2013] and that, if the problem is of Prognosis, what the Neural Network delivers is the argument that is most likely to be real. Details on the modeling of the problem of Prognosis using Neural Networks can be seen in Chapter 2 of this work.

2008 PHM Data Challenge Competition

After this small but necessary introduction to the usefulness of Neural Networks on the study of faults it can be found several models of Neural Networks applied to the databases PHM-2008 challenge and C-MAPSS.

The winner of PHM-2008 Challenge was the model proposed in [Peel, 2008]. This model considers an ensemble of Neural Networks from a Kalman filter. The Kalman filter provides a mechanism for merging predictions of multiple Neural Network models over time. The models of Neural Networks used are a Multi-Layer Perceptron (MLP) and radial basis function. This work indicates the importance of taking into account the operational settings. Finally shows that their initial predictions could tend to the mean or expected useful life of a unit. Another work is the second place of PHM-2008 Challenge proposed by [Wang et al., 2008], who uses a similarity-based approach, considering the extraction of features and a useful Health Indicator to see how much degradation there is before a failure. [Heimes, 2008] in third place, uses a recurrent Neural Network also taking into account the edition of the training database in the RUL, fixing it in a constant value but only mentions that this change improves the predictions without more details.

The 2008 PHM Data Challenge Competition is still an open competition and, together with the appearance of another set of C-MAPSS data, other models of Neural Networks and deep learning also appear to address the problem of Prognosis. In [Zheng et al., 2017] uses a Long Short Term Memory (LSTM) for the PHM-2008 challenge and C-MAPSS datasets, considering an edition of the training and test databases in the moment before the failure, fixing a constant RUL, determining the moment of failure with another Neural Network. It also indicates that the RUL estimation close to the zero RUL provides the best results. [Malhotra et al., 2016] uses an LSTM Encoder-Decoder architecture in only the first C-MAPSS dataset obtaining an unsupervised health index to determine RUL.

[Sateesh Babu et al., 2016] proposes the use of Convolutional Neural Network (CNN), typically used in images, but for Prognosis and Health Management (PHM) has the peculiarity that the application of convolutional filters is over time using padding at the start and ends of the time series. [Li et al., 2017] uses a Deep CNN in which dropout is also applied during its training, achieving the best results so far. Both works of CNN consider setting the RUL as constant in the database of training and testing.

The 2008 PHM Data Challenge Competition and C-MAPSS datasets are particularly complex because it is not specified exactly what is each parameter (sensor) given in each

column of the database to predict the RUL, therefore some physical approach to the variables is not possible. Moreover the problem to be considered can be understood as a stochastic problem [Wang et al., 2012] or, more specifically, it can be considered as a problem of statistical pattern recognition [Bishop, 2006], [Murphy, 2013] but through time, if it is seen aspects of the turbofan as 6 operating conditions, failure modes or noise, which operate changing a sequence of events.

The most successful Neural Networks models today are those that allow an modification on the RUL in the training and test data. This modification consists in that the values of the RULs greater than a particular RUL, are changed in their value by this particular RUL. This modification is justified by [Li et al., 2017] and [Ramasso, 2014], being that the last one indicates that the Health Index¹ does not show greater variation in this part of the time series. On the other hand, the convolution on series of time extract characteristics on the data with noise together with reduce the amount of parameters that the model considers, while recurrent models show good results similar those achieved by convolutional models, however recurrent models are typically used in problems of time series.

Finally, it is pertinent to ask whether perhaps it is not only that convolutional operation serves to assimilate data with noise and avoiding redundant parameters, but that they are also capable of reconstructing characteristics of the time series that the fault represents. For example, the nature of the faults allows them to be events that deliver information and do not manifest themselves for a long time, and that the application of the convolution as a linear combination over the data in the time series, may be reconstructing the information about past. This information would not be manifested in that time but in future data. So then, it can be thought that maybe in some cases (such as applying the convolution over time) the convolution can help the recurrent processes.

¹Health Index is one of assessment method on asset or equipment. The result describes the overall health condition of an asset. Additionally Health Index is a tool to manage the assets and identifier for investment needs, such as prioritizing capital investment and maintenance programs. The purpose of Health Index assessment is to measure the condition of the equipment based on various criteria related condition factors of long-term degradation that cumulatively resulted in the end of the age of the operating assets. Extracted of [Satriyadi Hernanda et al., 2014]

3 | Fundamentals

In what form is information stored, or remembered ?

How does information contained in storage, or in memory, influence recognition and behavior ?

F. Rosenblatt, The Perceptron, 1958.

To these two questions F. Rosenblatt in 1958 tries to answer with his probabilistic model for the storage of information in the brain. However, despite their antiquity, these questions still define most of what is known as Neural Network and deep learning. The first question raises a possible coding of information, while the second raises that this type of coding affects the properties of a Neural Network. This can be seen for example from the different types of Neural Network (encoders) and their usual application (their performance). A Convolutional Neuronal Network can process a feature map with great effectiveness, while a Recurrent Neural Network can model with success time series. Both types of networks have a unique dynamic (behavior) given by the equations that determine them. In this way, it can be seen that this field of studies continues in the same spirit as its pioneers who actually looked for clues about what defines how people behave.

In this chapter it is described in a general way the theory of Neural Network emphasizing the particular use of them for this work.

3.1. Fundamentals

3.1.1. Tensors

Definition 3.1 A tensor $\mathbf{T} \in \mathbf{T}_{0,n}$ of rank "n" in a "d" dimensional space is an object with the following properties:

- It has components labeled by n indices, with each index assigned values from 1 through d, and therefore having a total of d^n components;
- The components transform in a specified manner under coordinate transformations.

Extracted from the tensor definition of [B.Arffken and Weber, 2001].

Some authors [Goodfellow et al., 2016] simply consider a tensor \mathbf{T} as a n-array where T_{i_1, \dots, i_n} are its elements with i_1, \dots, i_n its indices. It is preferred to take the previous physical definition because it makes it clearer what a tensor is and what relationship it has with a scalar, a vector, a matrix or a multilinear function

Definition 3.2 A scalar $\alpha \in \mathbb{R}$ is a single number and can be considered as a tensor of rank 0.

Definition 3.3 A vector $\mathbf{v} \in \mathbb{R}^d$ is an array of scalars where \mathbf{v}_i are its elements, can be considered as a tensor of rank 1.

Definition 3.4 A matrix \mathbf{M} is a rectangular array composed for scalars defined by the number of rows and columns that it contains where \mathbf{M}_{ij} are its elements. A matrix can be considered as a tensor of rank 2.

Definition 3.5 Let A and B tensors of rank n in d dimensional space. The Hadamard product of A and B is defined by

$$(\mathbf{A} \circ \mathbf{B})_{i_1, \dots, i_n} = A_{i_1, \dots, i_n} B_{i_1, \dots, i_n} \quad (3.1)$$

3.1.2. Neural Network and Deep Neural Network

Definition 3.6 An activation function θ is defined by:

$$\theta: \Omega \subset \mathbb{R} \rightarrow \Lambda \in \mathbb{R} \quad (3.2)$$

this function in general operate element-wise on an output of a layer in a Neural Network being part of the nonlinear function between layers, allows controlling the form of the out-

put marking its threshold of the magnitude of the output. To this function it can also be added a parameter known as temperature T , $\theta(a/T)$ where a is the input of the activation function.

Definition 3.7 An activation function θ is considered saturated iff:

$$\forall z \in \mathbb{R} \mid \forall \theta(z) \in \mathbb{R} \mid \exists s \in \mathbb{R} \mid \lim_{n \rightarrow \infty} \theta(z) \leq s$$

Some popular examples are:

Logistic sigmoid activation function.

$$\theta(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

with $\sigma(z) \in (0, 1)$.

Hyperbolic tangent activation function.

$$\theta(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.4}$$

with $\tanh(z) \in (-1, 1)$.

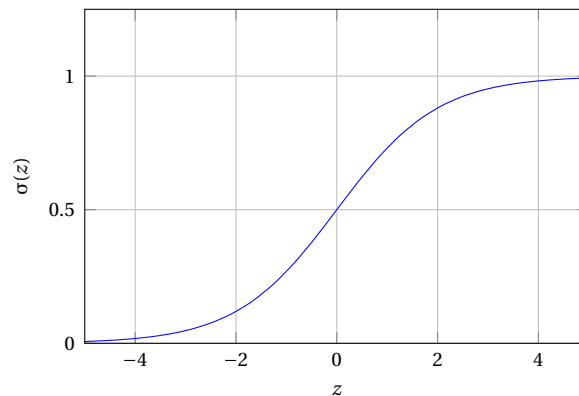


Figure 3.1: Sigmoid or logistic function. It is had $\sigma(-\infty) = 0$, $\sigma(0) = 0.5$, and $\sigma(\infty) = 1$.

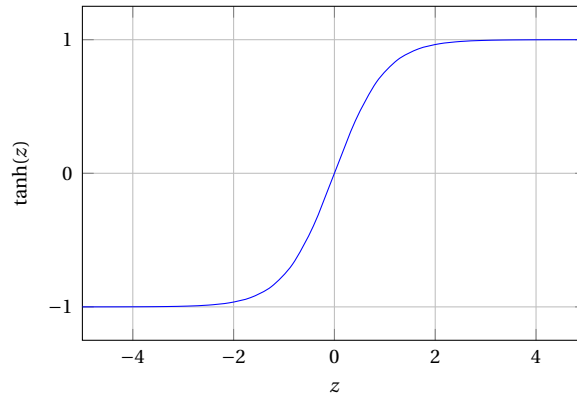


Figure 3.2: Hyperbolic tangent activation function.

Definition 3.8 An activation function θ is considered non saturated iff:

$$\forall z \in \mathbb{R} \mid \lim_{n \rightarrow \infty} f(z) \mid \rightarrow +\infty$$

Some popular examples are:

ReLU activation function.

$$\theta(z) = \begin{cases} 0 & \text{for } z < 0 \\ z & \text{for } z \geq 0 \end{cases} \quad (3.5)$$

con $\theta(z) \in (0, \infty)$.

ELU activation function.

$$\theta(z) = \begin{cases} \alpha(e^{-z} - 1) & \text{for } z < 0 \\ z & \text{for } z \geq 0 \end{cases} \quad (3.6)$$

with $\theta(z) \in (-\alpha, \infty)$ y $\alpha \in \mathbb{R}$ a certain parameter.

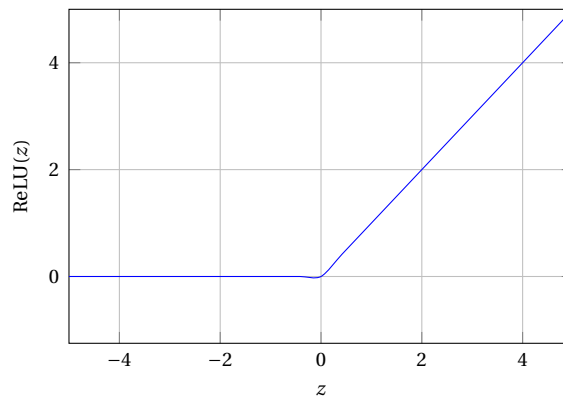


Figure 3.3: ReLU

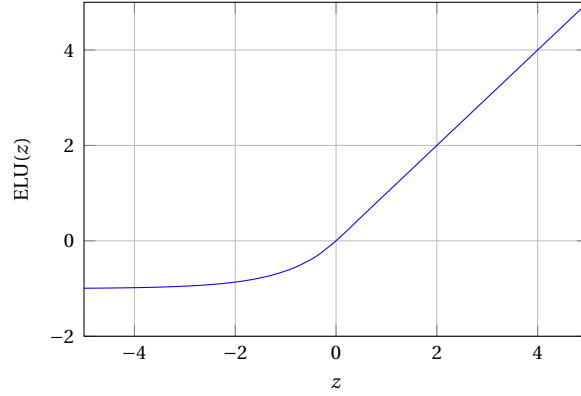


Figure 3.4: ELU

Definition 3.9 Let $\Omega \subseteq \mathbb{R}^d$ and $\Lambda \subseteq \mathbb{R}^d$. A Neural Network is a nonlinear function defined by:

$$f : \Omega \subset \mathbb{R}^d \rightarrow \Lambda \subset \mathbb{R}^d \quad (3.7)$$

whose goal is to approximate some function $f^* : \Omega' \subset \mathbb{R}^m \rightarrow \Lambda' \subset \mathbb{R}^n$. This function is usually composed for an input a_i in which an activation function $\theta(a_i)$ is applied. Then it can be established that (a_i) has the following form:

$$a_i = X^d \oslash W \quad (3.8)$$

where X^d is an input, W is a weight's matrix and \oslash is a multilinear application.

Definition 3.10 Let $\Omega \subset \mathbf{R}^m$, $\Lambda_1 \subset \mathbf{R}^{k_1}$, ..., $\Lambda_h \subset \mathbf{R}^{k_h}$, open sets and $f_1 : \Omega \rightarrow \Lambda_1, \dots, f_h : \Lambda_{h-1} \rightarrow \Lambda_h$ nonlinear functions. A Deep Neural Network is defined as:

$$f_1 \circ \dots \circ f_h : \Omega \subset \mathbf{R}^m \rightarrow \Lambda_h \subset \mathbf{R}^{k_h} \quad (3.9)$$

where $f_1 \circ \dots \circ f_h$ is a composition of functions with the same form as the previous definition that approximates another function $f^* : \Omega' \subset \mathbb{R}^m \rightarrow \Lambda' \subset \mathbb{R}^{k_h}$.

3.2. Feedforward Neural Network

A Feedforward Neural Network is a Neural Network whose directed graph is acyclical, i.e, there are no feedback connections in which outputs of the model are fed back into itself.

3.2.1. Perceptron

A Perceptron is a logistic regression (classifier) or linear regression [Bishop, 2006] whose model has a data entry $\mathbf{x} \in \mathbb{R}^m$, weights $\mathbf{w}^T \in \mathbb{R}^m$, biases $\mathbf{b} \in \mathbb{R}$ and depending of the case, a Sigmoid activation function. When the Perceptron has a Sigmoid activation function is called **Logistic Regression**. Given its activation function, the domain of its output $y \in (0, 1)$. In this way, a Perceptron is only constituted by one layer.

$$y = \sigma\left(\sum_{i=1}^m w_i \cdot x_i + b\right) \quad (3.10)$$

where y is the output, the input layer has m neurons i , the weights that unite both layers is denoted as w_i , b is the bias of a neuron and σ is the Sigmoid or Logistic function.

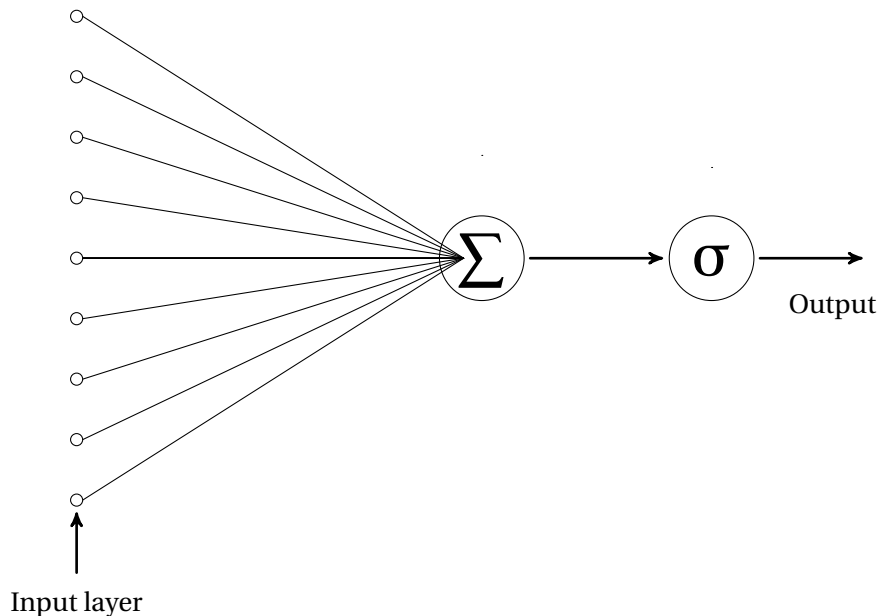


Figure 3.5: Inner structure of a Perceptron.(Own elaboration).

3.2.2. Multilayer Perceptron

A Multilayer Perceptron is a stack of layers of Neural Network, which for the first hidden layer can be written as follows :

$$y_j^1 = \sigma\left(\sum_{i=1}^m w_{ij} \cdot x_i + b_j\right) \quad (3.11)$$

meanwhile for any other k hidden layer :

$$y_j^{k+1} = \sigma\left(\sum_{i=1}^n w_{ij} \cdot y_i^k + b_j\right) \quad (3.12)$$

where the first input is $\mathbf{x} \in \mathbb{R}^m$, $y^{k+1} \in \mathbb{R}^n$ is the output of the hidden layer $k + 1$ since a layer k , the hidden layer k has n neurons i , the weights that unite both layers is denoted as w_{ij}^k , $b^k \in \mathbb{R}^m$ is the bias of a layer k and σ is the Sigmoid function.

This configuration is part of the known as Deep Learning and allows to achieve better results. The number of neurons and layers can vary according to the case addressed and as the Perceptron it is capable of addressing both regression and classification problems. It can be had a better idea from the figure 3.6:

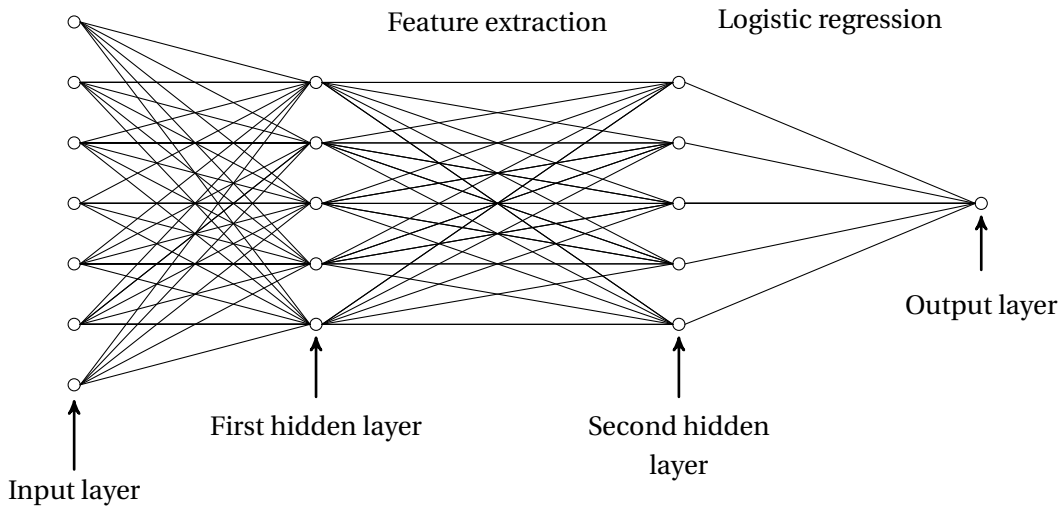


Figure 3.6: Multilayer Perceptron of 2 hidden layers.(Modified from: <http://blog.robfelty.com/2007/02/14/pgf-gallery>,last time consulted 06-09-2018)

3.2.3. Convolutional Neural Network

Convolutional Neural Network are introduced by Yann LeCun in 1990 [LeCun et al., 1990] to solve image recognition problems. Unlike the fully connected layers mentioned in before, this type of Networks does not have a total connection with the neurons of previous layer, instead uses a stack of filters of certain size that are convolved in the whole input image to generate a stack of feature maps (Figure 6.1). In this way, this kind of Neural Network can have dispersed interactions between the different points of the image, shared parameters and equivalent representations. While all redundant parameters are reduced, and with this the assimilation of noise and irrelevant information also reduced, and therefore a more generalist model is obtained without losing computational power.

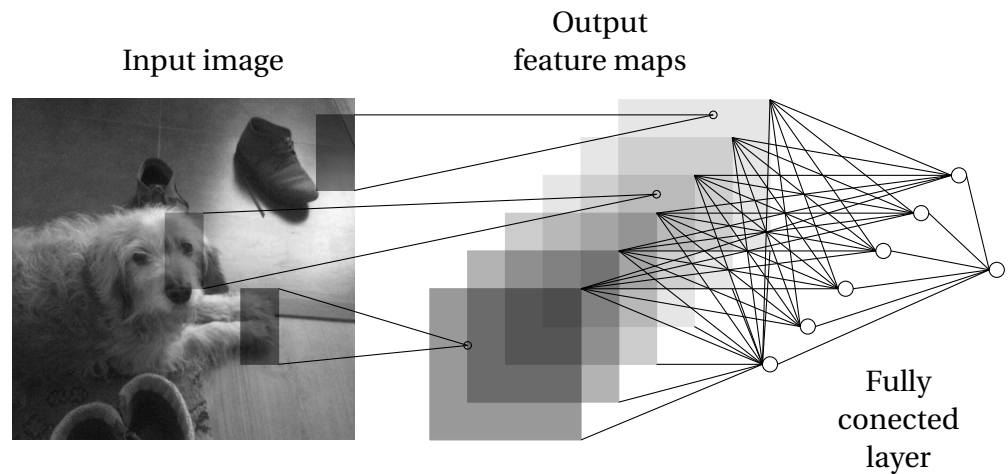


Figure 3.7: Example of CNN structure with one convolutional layer for image classification. (Based on [LeC+98, p. 2284], own elaboration).

Definition 3.11 *Discrete convolution:*

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a) \quad (3.13)$$

Definition 3.12 *Convolution in Neural Network:*

$$S(i, j) = (K * I)(i, j) = \sum_{m=1}^H \sum_{n=1}^W I(i - m, j - n)K(m, n) \quad (3.14)$$

where I is the input image of size W, H and K is the kernel or filter used.

Convolutional layer

A convolutional layer applies different filters or kernels on an input image, these filters slide along and across the image with certain steps known as **strides**. This input image is represented by a tensor in Red, Green and Blue (RGB) representation system, in a single channel in grayscale or one matrix as in the case of this work.

As a result of the application of the convolution, between a K kernel and an I image, feature maps with a size smaller than the size of the input image are obtained according to the following relationship:

$$D_{output} = D_{input} - D_{kernel} + 1 \quad (3.15)$$

where D_{output} is the dimension of feature maps (height or weight), D_{input} is the dimension of the input image (height or weight) and D_{kernel} is the kernel's dimension (height or weight).

zero-padding technique

When it is wanted to maintain the size of the image, zeros can be added at the ends, this technique is known as **zero-padding**. The number of rows of zeros added is determined by the following:

$$p = \begin{cases} k/2 & \text{for } (D_{input} - D_{kernel}) \text{ even.} \\ k/2 + 1 & \text{for } (D_{input} - D_{kernel}) \text{ odd.} \end{cases} \quad (3.16)$$

where p is the number of rows of zeros that are added at one end and k is the size of the kernel in a dimension.

Can be understood better the zero-padding technique in the figure 3.8:

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
48	32	4	54	84	10	67	99	110	
86	43	74	70	34	66	75	43	77	
99	0	15	98	45	55	53	92	88	
233	144	11	13	78	91	21	4	85	
210	187	44	77	89	25	88	65	44	
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 3.8: Zero padding in a sequential matrix in which kernels of size $(D_{kernel}, 1)$ are applied. (Own elaboration).

Fully conected layer

When it is wanted to make a prediction or classification from the application of convolutional layers, the so called Fully-connected layer is used. As its name implies, it is a layer whose neurons are completely connected to the points of the previous layer. This arrangement allows to extract other characteristics that require more information between points not necessarily linked spatially and that allows finally to make the regression or classification.

3.3. Recurrent Neural Network

A recurrent Neural Network is a network capable of processing sequential data ([Graves, 2013], [Mikolov et al., 2010]) to extract characteristics linked to a certain sequence of events that are temporally ordered (the data is ordered not necessarily according to a temporal metrical unit, although consecutively among them to achieve an output that can also be sequential). This type of Neural Network, like the Convolutional Neural Network, share parameters but, unlike the latter, the shared parameters are for the different positions of the temporal sequence.

To characterize an Recurrent Neural Network (RNN), this can be separate in 2 functions that characterize a hidden state and an output. Being that for each case it is appreciable different hidden states and outputs for the different times of the sequence (Figure 3.9).

Definition 3.13 *Let a sequential input $\mathbf{x} \in \mathbb{R}^{1 \times T}$, a Recurrent Neural Network is a cyclical graph that has H hidden units $h_j^{(t)}$ that allows to process data with a temporal order of an input $x_i^{(t)}$ in a time t for get an output $y_i^{(t)}$ in time t.*

$$h_j^{(t)} = \varphi\left(\sum_{h=1}^H w_{hj}^{h'} h_h^{(t-1)} + \sum_{i=1}^I w_{ij}^x x_i^{(t)} + b_j\right) \quad (3.17)$$

$$y_i^{(t)} = \sum_{j=1}^H w_{ji}^o h_j^{(t)} + b_j^o \quad (3.18)$$

Depending of the problem, different configurations of inputs and outputs can be considered for cells in one layer. For example, in the case of Figure 3.10 it can be seen a configuration of 3 RNN cells, of which only one receives an input meanwhile all generate outputs. In the case of Figure 3.11 a configuration is shown where all the cells receive an input although only one generates an output. While the case of Figure 3.12 2 cells receive inputs and 2 generates an outputs. In general it can be built any other kind of configuration and this depends on the particular problem to solve.

Despite being a good Neural Network to model sequential data, during its training (more details about the training process of a RNN in Section 3.4.3) it can present problems when the derivative is on the hyperbolic tangent (typical in RNN cases) whose output values are close to 1, obviously the values of the derivative approximate infinity since this point, meanwhile for values close to 5 its derivative is zero, so it does not contribute to the training process either. This is known as *Vanishing-Exploding problem* [Pascanu et al., 2013].

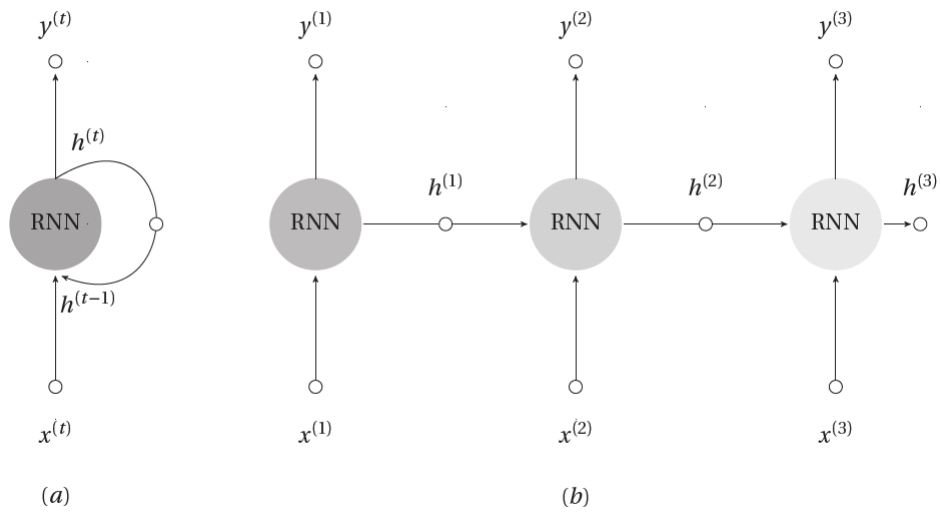


Figure 3.9: Recurrent network cell. Typical cyclic graph (a) that can be unrolled into (b) as an acyclic graph.(Own elaboration).

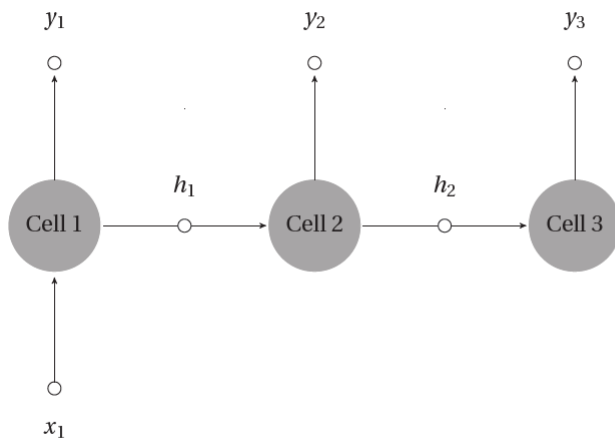


Figure 3.10: Recurrent Neural Network with sequence input-output one to many.(Own elaboration).

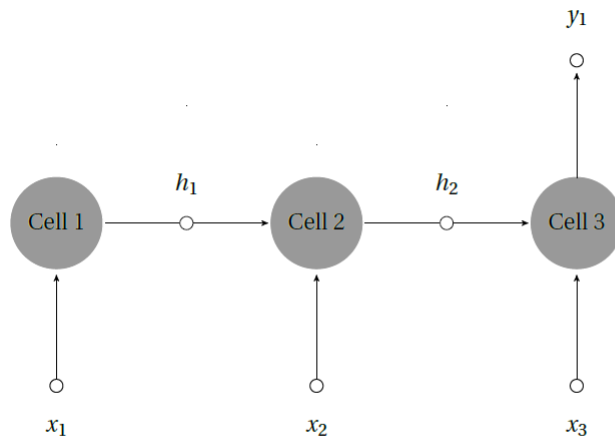


Figure 3.11: Recurrent Neural Network with sequence input-output many to one.(Own elaboration).

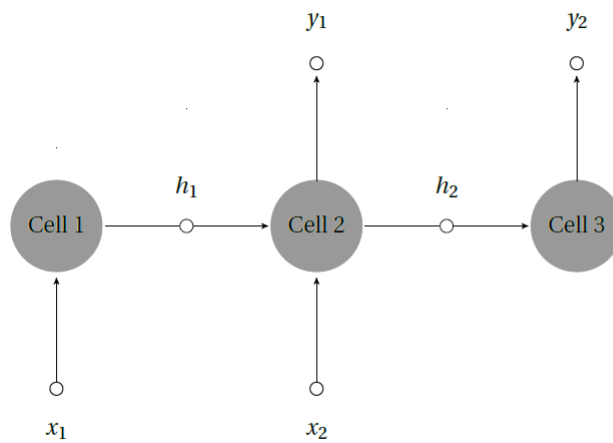


Figure 3.12: Recurrent Neural Network with sequence input-output many to many.(Own elaboration).

3.3.1. Long Short Term Memory

The Long Short Term Memory (LSTM) appear from the *vanishing-exploding problem* aforementioned usually seen in training traditional RNNs, LSTM models use some gates that allow to maintain the information of a previous state if the vanishing-exploding problem occurs. The expression long-short term refers to the fact that LSTM is a model for short-term memory that can last a long period of time. An LSTM is suitable for classifying, processing and predicting time series with unknown time and size delays between important events. The relative insensitivity to the aforementioned problem is that which gives the LSTM advantage over common RNNs.

A common LSTM unit consists of a cell with an input gate, an output gate, a forget gate and a memory cell. The memory cell is responsible for remembering the values in arbitrary time intervals. Each of the gates can be considered as a conventional artificial neuron, i.e, an activation function is used on a weighted sum of the inputs.

1. An input x_t arrives and forgets gates f_t^j , inputs gates i_t^j and a new memory cell \bar{c}_t^j are obtained:

$$i_t^j = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + V_i \cdot c_{t-1} + b_i)^j \quad (3.19)$$

$$f_t^j = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + V_f \cdot c_{t-1} + b_f)^j \quad (3.20)$$

$$\bar{c}_t^j = \tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c)^j \quad (3.21)$$

2. The new memory is created partially forgetting the previous memory c_{t-1}^j and adding the new memory \bar{c}_t^j

$$c_t^j = f_t^j \cdot c_{t-1}^j + i_t^j \cdot \bar{c}_t^j \quad (3.22)$$

3. Output gate o_t^j is computed

$$o_t^j = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + V_o \cdot c_{t-1} + b_o)^j \quad (3.23)$$

4. Hidden state h_t^j is obtained

$$h_t^j = o_t^j \cdot \tanh(c_t^j) \quad (3.24)$$

For a better understanding of the information flux in this kind of recurrent cell it can seen the next figure in which hidden states and memory cell share information between times, together with an output in each time:

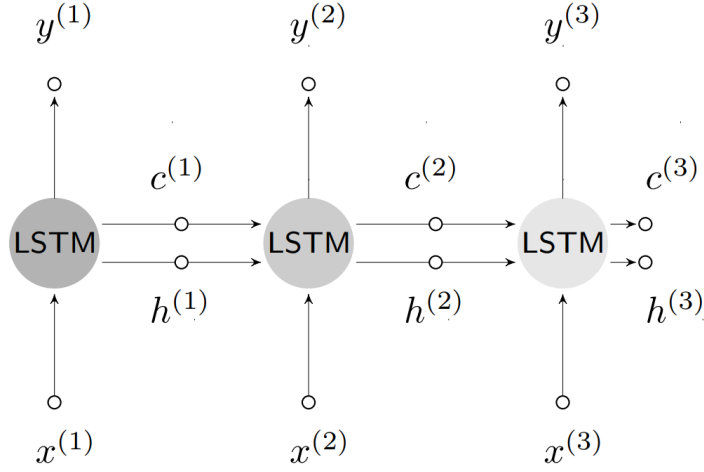


Figure 3.13: Flux of information in LSTM cells. Similar to RNNs, LSTMs have output and hidden states. (Own elaboration).

3.3.2. JANET

This kind of Networks appear answering the questions of if all the gates of an LSTM are strictly necessary, fact already refuted with the proposal of the Gated Recurrent Unit (GRU) ¹. In this way a new network called JANET [van der Westhuizen and Lasenby, 2018] that only keeps the forget gate and the memory cell obtaining a network that requires less computing power and is more general model.

Therefore the following equations that govern this type of networks.

1. An input x_t arrives and forgets gates f_t^j is obtained:

$$f_t^j = \sigma(W_f * x_t + U_f * h_{t-1} + b_f)^j \quad (3.25)$$

2. The new memory is created partially forgetting the previous memory c_{t-1}^j and adding the new memory \tilde{c}_t^j

$$c_t^j = f_t^j \odot c_{t-1}^j + (1 - f_t^j) \odot \tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c)^j \quad (3.26)$$

3. Hidden state h_t^j is equal to new memory c_t^j

$$h_t^j = c_t^j \quad (3.27)$$

¹This type of Recurrent Neural Network is not studied in this work because this work tries to evaluate the most extreme cases in all order, in this case the Networks with higher (ConvLSTM) and minor (ConvJANET) number of existing gates are evaluated, taking into account the results obtained by [van der Westhuizen and Lasenby, 2018] that indicate that a smaller amount of gates improves the results.

For a better understanding of the information flux in this kind of recurrent cell it can be seen in the next figure in which just the hidden states share information:

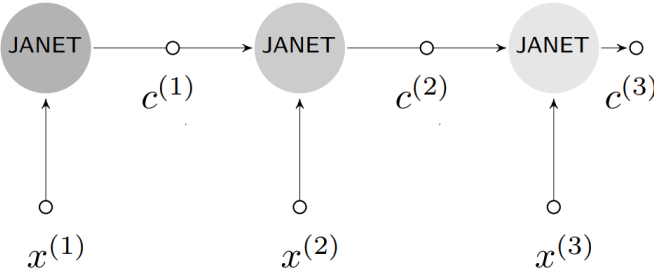


Figure 3.14: Flux of information in JANET cells. Different from RNN or LSTM cells, Janet cells don't have an output but only a hidden state. (Own elaboration).

3.3.3. Encoder-Decoder Recurrent Architectures

A recurrent encoder-decoder model was introduced independently from each other in [Chung et al., 2014] and [Sutskever et al., 2014]. Perhaps in the context of this work the application for images giving by [Srivastava et al., 2015] and [Shi et al., 2015] is interesting. The Encoder-Decoder idea could be understanding by the figure 3.15.

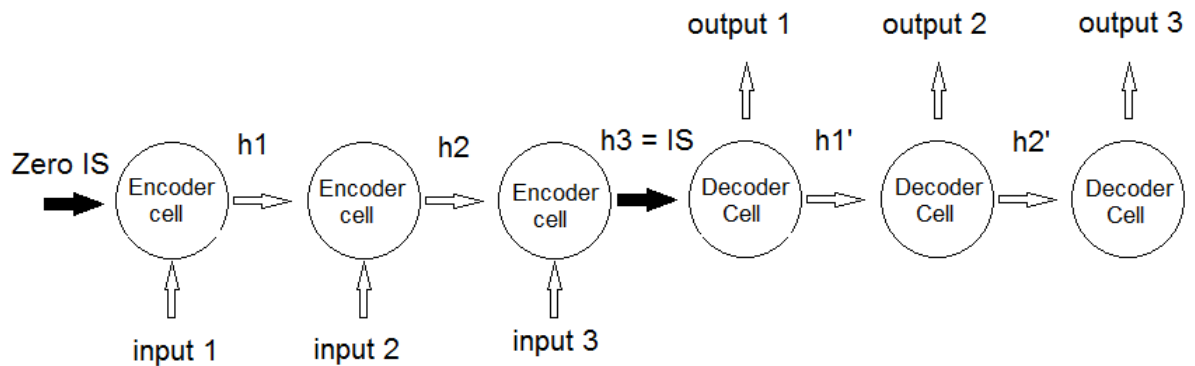


Figure 3.15: Structure of an unconditional recurrent Encoder-Decoder. The inputs are processed in the encoder cells and unfolded in the decoder cells which initial state (IS) is equal to the last hidden state of the encoder. (Own elaboration).

In the first part of the Encoder-Decoder architecture, a representation of the sequential inputs is learned in the last hidden state of the encoder and then it is used as Initial State (IS) for the first hidden state of the decoder. The decoder is capable of extract more specific features for generate an output. It is also possible to increase the number of layers both in the Encoder and in the decoder and even make them bidirectional.

In general at least 2 ways of making the decoder are known, forms given by the networks known so far. The first is to make an unconditional decoder, i.e, that the recurrent cell at a time t does not receive the output of the same cell at time $t - 1$. And the other way of making the decoder is that the recurrent cell in a time t receives the output of the same cell in time $t - 1$, which would be a conditional decoder. The use of each type of decoder

depends on the type of recurrent network and the output that is to be generated. For example, an output that in turn is also sequential probably require a conditional decoder, whereas if a non-sequential output is required in the case of photo tagging, it might be better to use an unconditional Decoder.

These forms are used mainly for the cases RNN, LSTM and GRUs, along with their convolutional counterparts but in the case of the recurrent model Just Another Network (JANET), since there is no output and the hidden state is equal to the memory cell, there is a model whose conditional only is given by its hidden state, i.e, it is strictly unconditional.

3.3.4. Convolutional LSTM

The Convolutional Long Short Term Memory (ConvLSTM) appear with the purpose that an LSTM network may be able to take into account nearby data spatially-temporally as it happens in the case of pixels in a video, i.e, **take attention**. This is then achieved by using convolutional structures in each cell with it can be built a stack of ConvLSTM (Figure 3.16). This is an encoding-forecasting structure (Figure 3.17) capable of solving problems of space-time prediction.

The big difference respect to memory selection cell of a common LSTM is that the ConvLSTM incorporate the calculation of the convolution on the inputs X_t , the states of the neuron H_t and the memory cell C_t^j . This is seen in the following equations that also show the order in which a new state is selected H_t :

1. An input X_t arrives and forgets gates f_t^j , inputs gates i_t^j and a new memory cell C_t^j are obtained:

$$i_t^j = \sigma(W_i * X_t + U_i * H_{t-1} + V_i \circ C_{t-1} + b_i)^j \quad (3.28)$$

$$f_t^j = \sigma(W_f * X_t + U_f * H_{t-1} + V_f \circ C_{t-1} + b_f)^j \quad (3.29)$$

$$\tilde{C}_t^j = \tanh(W_c * x_t + U_c * h_{t-1} + b_c)^j \quad (3.30)$$

2. The new memory cell is created partially forgetting the previous memory C_{t-1}^j and adding the new memory \tilde{C}_t^j

$$C_t^j = f_t^j \circ C_{t-1}^j + i_t^j \circ \tilde{C}_t^j \quad (3.31)$$

3. Output gate o_t^j is computed

$$o_t^j = \sigma(W_o * X_t + U_o * H_{t-1} + V_o \circ C_{t-1} + b_o)^j \quad (3.32)$$

4. Hidden state h_t^j is obtained

$$H_t^j = o_t^j \cdot \tanh(C_t^j) \quad (3.33)$$

For the case of this work, are interesting models with a fully connected layer, both in the normal case and in the Encoder-Decoder case that have a only one output or prediction. Then, it can be seen this kind of models in the next figures:

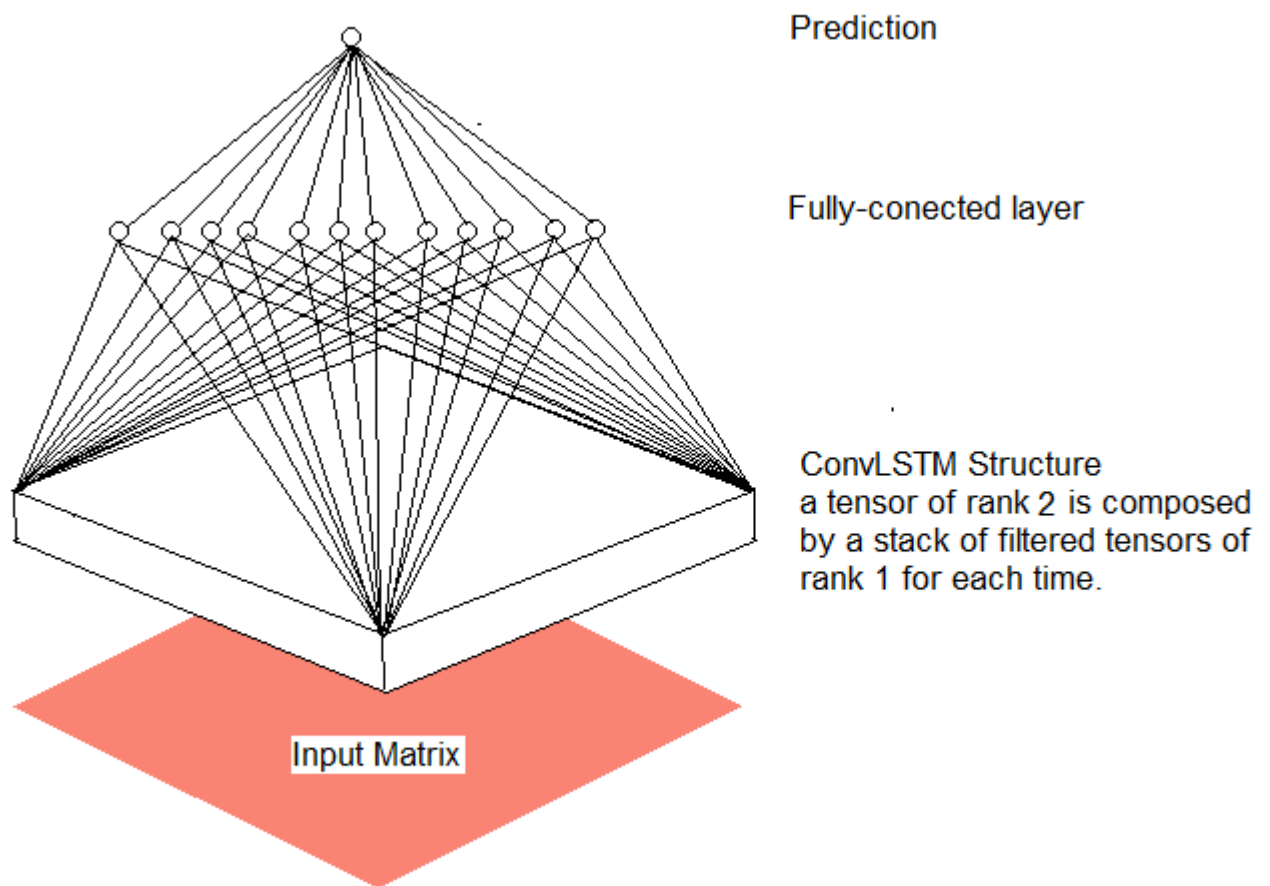


Figure 3.16: Structure of a fully-connected ConvLSTM for time window input. (Own elaboration).

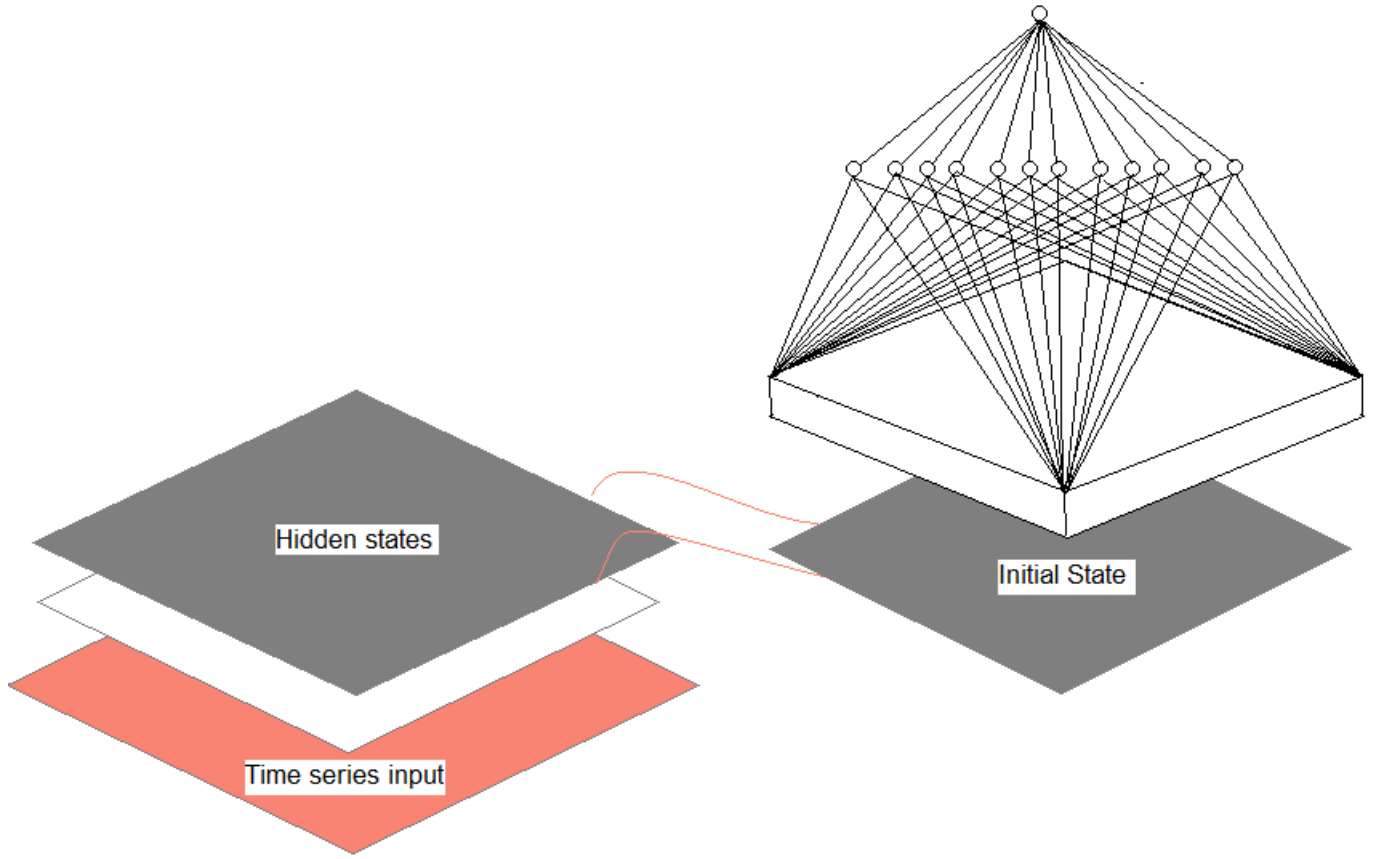


Figure 3.17: Encoder-Decoder structure of ConvLSTM for time series input. (Own elaboration).

3.3.5. Convolutional JANET

In the same way that the ConvLSTM has been proposed, the Convolutional Just Another NETWORK (ConvJANET) is proposed, which in this work is used to assimilate temporal relationships in a group of time series. The way in which a ConvJANET can operate can be seen in the following equations:

1. An input X_t arrives and forgets gates f_t^j is obtained:

$$f_t^j = \sigma(W_f * X_t + U_f * H_{t-1} + b_f)^j \quad (3.34)$$

2. The new memory is created partially forgetting the previous memory C_{t-1}^j and adding the new memory \tilde{C}_t^j

$$C_t^j = f_t^j \odot C_{t-1}^j + (1 - f_t^j) \odot \tanh(W_c * X_t + U_c * H_{t-1} + b_c)^j \quad (3.35)$$

3. The state of the hidden neuron is obtained H_t^j

$$H_t^j = C_t^j \quad (3.36)$$

As it was seen in the figures 3.16 and 3.17, a prediction can be made from a fully-connected JANET or Fully connected JANET Encoder-Decoder being basically the same application as for ConvLSTM cases..

3.4. Training and Optimizers

The training process of any Neural Network or deep Neural Network consists basically in determining the weights and biases of a network such that the model to train (or function f) can approximate a function f^* with a certain error range. Weights and biases are obtained from the maximum likelihood, i.e, the weights and biases that minimized the network error. For this purpose, an algorithm called optimizer is used meanwhile the error associated to a particular weight is obtained by the another algorithm knew as back-propagation.

3.4.1. Loss function

For problem 2.1, it is determined a function $l(\theta)$ that maximizes the likelihood. As it can be seen from this function $l(\theta)$, there are values can be considered or assumed as constants, in particular the variance ², which is linked to noise on the data. Then the error function that allows to maximize the likelihood remains as:

$$E_n = \sum_{i=1}^B (y_i - \mu(x_i))^2 \quad (3.37)$$

This function is commonly known as **Residual sum of squares**, where B is the number of examples of a batch of size B with a certain pattern n . However, in order to modify each of the weights and biases given a certain example, the error associated with a certain weight value W and bias b it must be had. Then this is accomplished using the procedure known as Back-propagation.

L1 regularization

To the aforementioned cost function, a term is added that considers the sum of the weights of the Full-connected layer. This technique is known as L1 regularization and allows the algorithms that train the networks, to better distribute the magnitudes of the weights of a particular hidden layer. So then:

$$E_n = \sum_{i=1}^B (y_i - \mu(x_i))^2 + \sum \|W\| \quad (3.38)$$

²In this work it is taken as a good approximation that the variance is constant, although as it was seen in the chapter on Related work, some authors suggest that the error is not constant in all the predictions and therefore the variation of the error does not necessarily either.

3.4.2. Back-propagation

Back-propagation is proposed by [Rumelhart et al., 1988] and have for objective give the error associated to a particular weight. This algorithm is based on the chain rule for computing the derivative in which each weight is a variable. Therefore, if a weight w_{ij} weighs an output z_i and previously there are other functions that are derivable, the following nomenclature is used:

$$\frac{\partial E_n}{\partial w_{ij}} = \delta_j \cdot z_i \quad (3.39)$$

where δ_j represents the multiplication of all derivative functions prior to w_{ij} .

3.4.3. Back-propagation Through-Time

The back-propagation through time algorithm [Werbos, 1990], like the previous one, seeks to determine errors associated to the weights and biases of a recurrent Neural Network. This is done by first unfolding the recurrence, i.e, taking the recurrence as simply a composition of k functions for k times. Then the standard back-propagation is applied. To understand the process of unfolding a bit better it can be considered the following:

$$s^t = f(s^{t-1}, \theta) \quad (3.40)$$

given a $t = 2$ the recurrent can be unfolded as:

$$s^2 = f((f(s^0, \theta))^1, \theta) \quad (3.41)$$

where s^0 is the input in time zero. In this way it can be applied back-propagation as in a feed-forward Neural Network.

3.4.4. Optimization

Gradient

From the previous back-propagation algorithms, the derivatives of the cost function that determine whether or not to increase a weight w_{ij} are determined. As is well known, $\frac{\partial E_n}{\partial w_{ij}} > 0$ implies that increasing the weight w_{ij} increase the cost E_n , while $\frac{\partial E_n}{\partial w_{ij}} < 0$ implies that increasing the weight w_{ij} decreases the cost E_n . But it must also be considered how much this weight w_{ij} is increased so that E_n is decreased, this can be done by multiplying $\frac{\partial E_n}{\partial w_{ij}}$ by a small quantity given by the hiperparameter, Learning rate (η) that denotes the amount that varies w_{ij} to decrease the cost E_n . In this way a mechanism to update (learn) a weight w_{ij} is established.

Therefore weights variation is written as,

$$\Delta w_{ij} = -\eta \frac{\partial E_n}{\partial w_{ij}} \quad (3.42)$$

Definition 3.14 *The learning process in a Neural Network consists in updating the weights w_{ij} such that the value given by the cost function is reduced. In this way the weights w_{ij} are updated in the following:*

$$w_{ij} = w_{ij} + \Delta w_{ij} \quad (3.43)$$

ADAM Optimizer

From this definition, many options have been sought [Li et al., 2009], [Duchi et al., 2011], [N. Dauphin et al., 2015] and [Kingma and Ba, 2014] that improve more and more the learning process in a Neural Network and controlling specific problems. For example, a problem can be that in each step of training there are large oscillations product of non-stationary objectives and very noisy problems.

In this work the ADAM optimizer is used because of the good results indicated by [Kingma and Ba, 2014] that, for the aforementioned gradient, adapt a learning rate.

3.4.5. Initializers

Although the continuous advance of optimizers for Neural Network and the fact that they actually converge to a minimum, some works such as [Glorot and Bengio, 2010], shows that in many times when a saturated activation function (the case of this work) and a random initialization of parameters are had, saturation can occur. Then the saturation decreases the adjustment that the network can have on the database and in consequence decreasing the accuracy of the results.

In this way, with the idea of avoiding this kind of problems, to initialize the weights on all networks it is used **Xavier initializer**[Glorot and Bengio, 2010] (also it can be used a variant of this initializer to initialize the filters of the convolutions). While in the case of initializing the biases in the recurrent part, it can be used the one known as **Chrono initializer** [Tallec and Ollivier, 2018].

4 | Turbofan RUL Estimation

4.1. Dataset's

As was mentioned in the previous chapter, the study of PHM has the need of large number of databases which is according to the number of problems studied, however these databases are not always available for all problems. That is why in 2008, in the context of the "Prognosis and Health Management Competition", a database of turbofan units with different types of faults and operating conditions was proposed [Saxena et al., 2008]. In addition, other databases with different characteristics appear and allow a broad comparison respect to aspects related with the difficulty of generating a predictor model.

In this work, the models of Neural Networks were evaluated only in the C-MAPSS database because of the variety of data that it presents and that allow to analyze the large number of models to test and discard, and consequently it allow a better understanding the generalization that has a certain model. But also, this dataset let the edition of the RUL is the test sets, issue that is discussed later.

4.1.1. C-MAPSS dataset's

C-MAPSS database is made from the software C-MAPSS ¹ created by NASA Army Research Lab [DeCastro et al., 2008]. The tool C-MAPSS is capable of simulating a turbofan (Figure 4.1) in different operating conditions and different types of failure modes, adding different noise levels in each simulation. In the software this is done through the variation of specific parameters of operating conditions, controllers or environmental specifications. In addition, it is also noteworthy that efficiency's parameters can be modified randomly in order to simulate various levels of degradation in the different components of turbofans.

¹More information about this software in <https://www.grc.nasa.gov/www/cdtb/software/mapss.html> (last time consulted 06-09-2018)

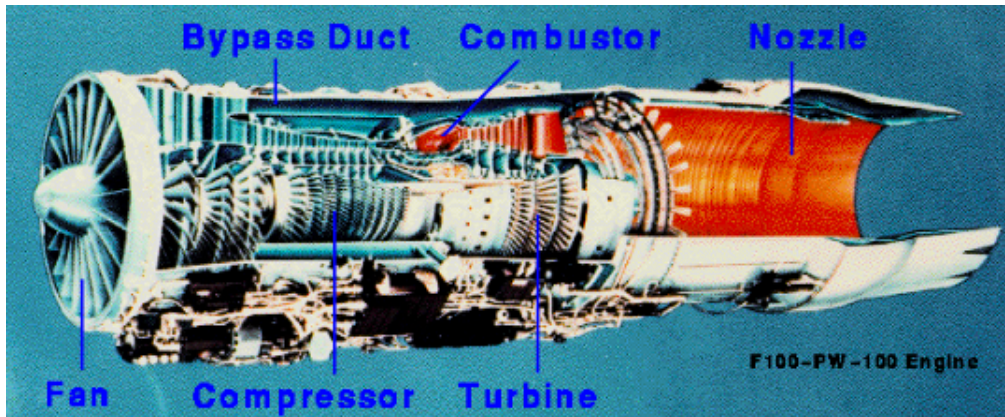


Figure 4.1: Example and description of a turbofan. A model of Pratt Whitney F100, manufactured in 1970 and propeller of fighters F-15 Eagle and F-16 Fighting Falcon. (image taken from https://www.grc.nasa.gov/www/k-12/Missions/Jim/Project2_act.htm, last time consulted 06-09-2018)

Then, four databases composed for different time series and increasing complexity are generated. In each time series the behavior of the turbofan is shown for 21 parameters of sensors of the system and also too, other 3 parameters that show turbofan's operating conditions. The number of failure modes and operating conditions are summarized in the following table:

Dataset	Operating conditions	Fault modes	train examples	test examples
FD001	1	1	100	100
FD002	1	2	259	260
FD003	6	1	100	100
FD003	6	2	248	249

Table 4.1: C-MAPSS dataset's specifications.

Each turbofan begins with different degrees of initial use and unknown manufacturing conditions although this initial use and manufacturing conditions are considered normal, i.e, it is not considered a failure condition [Saxena et al., 2008].

Therefore, turbofan's sequences shown normal or nominal behavior at the beginning of each time series and in some point begins to degrade until predefined limit in which it is considered that the turbofan can no longer be used.

In this work it can be considered the approach of leaving as constant those RUL which are greater than a certain number of cycles. This is justified because the parameters that show the behavior of the turbofan show normal operating conditions in those points, i.e, the data show a little variation that decreases the feasibility of making different accurate

predictions for each point.

In contrast, the data since the fault show a lot of information and allow obtaining the best results. Then it is considered that there is a time from the beginning of the operation such that with a 99% probability the turbofan working fine.

4.1.2. C-MAPSS dataset evaluation formulas

In order to compare different types of Neural Networks proposed by the researchers, the following formulas are proposed in [Saxena et al., 2008]:

- Root mean square difference (RMSD)

$$\sqrt{\frac{\sum_{i=1}^n (d)^2}{n}} \quad (4.1)$$

- Score

$$s = \begin{cases} \sum_{i=1}^n e^{-\left(\frac{d}{a_1}\right)} - 1 & \text{for } d < 0 \\ \sum_{i=1}^n e^{\left(\frac{d}{a_2}\right)} - 1 & \text{for } d \geq 0 \end{cases} \quad (4.2)$$

where,

s is the score achieved.

n is the number of units in the test set.

d = \hat{t}_{RUL} (RUL estimated - RUL real)

$a_1 = 10$, $a_2 = 13$, constant parameters given.

5 | Problem Statement

RUL is a subjective measure that depends on what is understood as failure and good conditions of use of a mechanical system. For example, is not the same criteria for use of an aircraft turbine that for a motorcycle engine, despite being completely different machines the truth is that aviation standards are more stricter than a common user may have for a vehicle. Then, it can be defined RUL as time difference between a given time of a machine in good conditions of use and the time in which the machine, due to its faults, can no longer operate.

Therefore, returning to the case of the problem of determining the RUL of a turbofan from a certain set of historical time series, the assigned data in each case can be seen in the following way:

$$\Omega = \{(x_i^{tj}, y_i)\}_{i=1}^N \quad (5.1)$$

where Ω is knowed as the dataset, N is the number of training examples, $x_i^{tj} \in \mathbb{R}^{T \times D}$ is a window of a time series with D features and T times, and $y_i \in \mathbb{R}$ is the RUL corresponding to a particular x_i^{tj} since the last time of itself.

Moreover, aspects such as noise, operating conditions and multiple failure modes, reduce the feasibility of creating a map that assigns a certain \tilde{y} predicted to a certain x_i^{tj} . Consequently, what it is looking for is a non-linear function that fulfills the following:

$$y_i = \mu(x_i) + \varepsilon(x_i) \quad (5.2)$$

where $\mu(x)$ is a Neural Network model with multilineal operations and particular activation functions, meanwhile $\varepsilon(x)$ is the residual error or noise that the Neural Network has and that depends of x , as can be understood from the observation of several authors as [Sateesh Babu et al., 2016], [Li et al., 2017] or [Zheng et al., 2017]. Can also assume that $\varepsilon(x)$ has a normal distribution. So then:

$$\varepsilon(x_i) \sim \mathcal{N}(\mu(x_i), \sigma^2(x_i))$$

Therefore, it can be written the model in the following way:

$$P(y | x, \theta) = \mathcal{N}(\mu(x), \sigma^2(x)) \quad (5.3)$$

where as stated before $\mu(x) = \mathbb{E}(y | x)$ is the Neural Network model and $\sigma^2(x)$ the variance of the error.

Maximum likelihood estimation

One way to look at the determination of Neural Network parameters is the maximum likelihood defined as:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \log P(\Omega | \theta) \quad (5.4)$$

In this case it is also assumed that the N training examples are independent of each other and that they are identically distributed for a large part of the cases, i.e, they are iid¹. Then it can be rewritten " $\operatorname{argmax} \log P(\Omega | \theta)$ " as " $\operatorname{argmin} -\log P(\Omega | \theta)$ ", so then:

$$l(\theta) = -\log P(\Omega | \theta) = -\sum_{i=1}^N \log P(y_i | x_i, \theta) \quad (5.5)$$

And inserting the definition of the Gaussian:

$$l(\theta) = -\sum_{i=1}^N \log \left[\left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{2\pi\sigma^2} (y_i - \mu(x_i))^2 \right) \right] \quad (5.6)$$

$$l(\theta) = \frac{-1}{2\sigma^2(x)} \sum_{i=1}^N (y_i - \mu(x_i))^2 - \frac{N}{2} \log(2\pi\sigma^2(x)) \quad (5.7)$$

Finally it can be raised the problem of predicting the RUL of a mechanical system by problem 5.1:

Problem 5.1 *Given a dataset $\Omega = \{(x_i, y_i)\}_{i=1}^N$ where x_i is a matrix of time series with maximum time T and D features, and y_i is the RUL. It is wanted to learn a mapping $x_i \in \mathbb{R}^{T \times D} \mapsto y_i \in \mathbb{R}$ such that $y_i - y_i^* = \varepsilon(x_i)$, where there are a real RUL y_i^* and an error $\varepsilon(x_i)$ which is the noise in the prediction*

It is used the dataset Ω to train the models $\mu_m(x_i)$ which are the convolutional recurrent models ConvLSTM, ConvJANET, ConvLSTM Encoder-Decoder and ConvJANET Encoder-Decoder. As it is said before, the problem of finding the parameters for a model $\mu_m(x_i)$ can be formulated as maximum likelihood problem over a space of parameters

¹In the Chapter of Results, it can be seen that these assumptions are not always necessarily true, especially when the data are supposed to be distributed identically. Even so, these assumptions help to solve the problem with a certain margin of error.

$\theta = \{w_1, \dots, w_n, b_1, \dots, b_n\}$. So then can be interpreted $y_i \in \mathbb{R}$, $x_i \in \mathbb{R}^{T \times D}$, $\varepsilon_i \in \mathbb{R}$ as random variables, and y_i^* the real RUL for x_i . Then, Ω is used to learn a prior model $P(y | x, \theta)$ and train a model $\mu(x) = \mathbb{E}(y | x)$ that learn the mapping:

$$x_i \mapsto y_i^* + \varepsilon_i = \mathit{argmax} \log P(y_i | x_i, \mu(x_i), \sigma^2(x_i)) \quad (5.8)$$

This is known as the probabilistic or Bayesian approach [Murphy, 2013], [Bishop, 2006] that let understand properly the problem of use a Neural Network for predict the RUL of a mechanical system in any time. In this way, can be observed that the problem is in principle deterministic, because it is seeked to know a certain value with a certain error that comes from not knowing with certainty what are the operative adjustments, future failure modes or even noise, i.e, aspects that affect the input data.

6 | Methodology and Experiments

In this chapter it is presented the methodology used to determine the best model of ConvRNN for the prediction of RUL in a mechanical system. In particular, the database previously described in chapter 4 (C-MAPSS) is used, to which the models to be evaluated are adapted. The search limits of hyperparameters are also specified and delimited for the number of points that the smallest training-set (**FD001**) has. From the this search, 4 models are determined that correspond to the best model found in each type of ConvRNN.

These models are trained taking the union of some related training-sets to also demonstrate the capacity of generalization of the proposed models. Finally the models are tested with each test-set of **C-MAPSS**.

6.1. Neural Networks design

It is compared the performances of 2 cell types of ConvRNN using their cells in a normal case and in **Encoder-Decoder** case. In a normal case it is basically had a layer where an input enters and an output comes out as it normally happens in any Neural Network, while in a **Encoder-Decoder** case there are 2 layer blocks. The first block is the Encoder that receives an input and a **zero initial state** that denotes the lack of information about the past. The Encoder encodes the information received in a hidden state, which is copied and given as an initial state to the second block that is capable of unfolding the previous hidden state and emitting an output.

6.1.1. Application of convolution

For the use of ConvRNN, the time window approach is taken, in which the window size considered is adjusted to the lowest temporal sequence given in an example of the training database. This configuration allows a wide generation of possibilities for learning and

also the application of filters over a sequence of time. However, the convolution in ConvRNN has only been used to extract spatial characteristics.

As it was explained in the previous chapter, ConvRNN cells are capable of extracting spatio-temporal characteristics in a data entry, but in the case of this work there are no spatial characteristics to analyze although data in different dimensions of the state of a turbofan (sensor measurements). Perhaps, a set of specific measurements of sensors can show the status of a certain component of the turbofan, but in the case of the database C-MAPSS, each measured dimension is put in random order so inferences directly from a group of these dimensions can not be made.

Therefore, if it is wanted to apply the convolution with the idea of obtaining the best results, it should be taken that the proximity between columns (measured sensors or dimensions of the turbofan's state) does not necessarily show information or characteristics to be extracted. Consequently, the convolution in this work is only applied over time as is also done in [Sateesh Babu et al., 2016] and [Li et al., 2017] as it was said earlier. In this way each part of the time series where the convolution is applied, generates a data for a certain *output time*.

It is also wanted to keep the size of the tensor that leaves the ConvRNN layer, the zero-padding technique is used (see Subsection 3.2.3). For a better understanding of the convolution application on the input and subsequent generation of a feature map it can be seen the following figure:

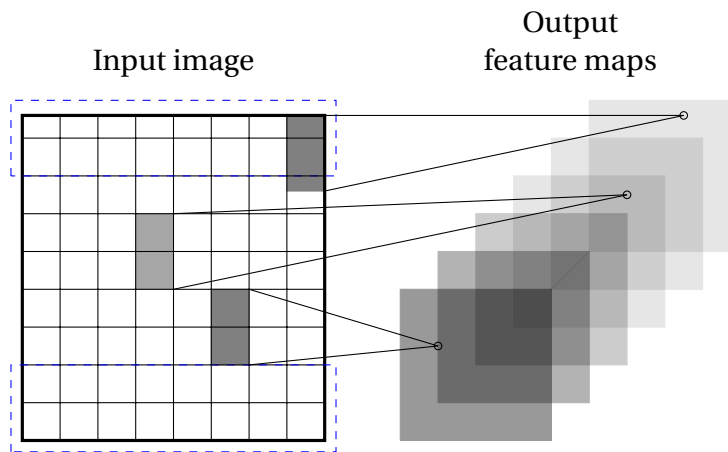


Figure 6.1: Application of convolution between a time series and a kernel of a network that operates with convolution. The filter is only applied in the time dimension to which zero-padding is added (dotted in blue) at its start and end ends of the series. (own elaboration)

The map of characteristics obtained is made from a time series that is generated with the output (ConvLSTM) or hidden state (ConvJANET) of each convolutional recurrent cell in

each time, maintaining the independence between the original dimensions of the input but obviously combining the data in time.

6.1.2. Number of parameters and size of datasets

In this work the dropout method usually used in Convolutional Neuronal Networks¹ is not used. Instead, this work seeks to justify the use of a particular model of Network that in a future work can be improved by the dropout method. Consequently, all models to be evaluated must satisfy the condition that their number of parameters should be less than the number of existing points in the database for avoid overfitting [Murphy, 2013].

6.2. Use of C-MAPSS database

6.2.1. Joining train-sets

For the use of C-MAPSS databases, must be observed that **FD001**, **FD002** and **FD003** are particular cases of **FD004** [Ramasso and Saxena, 2014]. As has been explained in Chapter 4, these databases have an increasing order of complexity, with **FD001** the simplest and **FD004** the most complex.

In this way, it can also be used the union of the simplest databases with the most complex ones with the idea of increasing the sizes of the databases for the most complex cases, obtaining better results and demonstrating the generalization capacity that Neural Networks have. However, test sets are not joined because the idea of this work is to compare the Neural Networks separately with the previous results of other works, even so, as it has to **FD004** covers the greatest amount of possibilities, it is also that its results show in a consistent way the generalization that a model has when it has been trained with the set with all the datasets. Therefore, the training datasets for each testing dataset are as follows:

¹This method allows to put a large number of neurons that finally condense as an assembly automatically [Hinton et al., 2012].

Dataset	Operating conditions	Fault modes	train examples	test examples
FD00 1	1	1	100	100
FD00 1,2,3 y 4	6	2	707	260
FD00 1 y 3	6	1	200	100
FD00 1,2,3 y 4	6	2	707	249

Table 6.1: C-MAPSS dataset’s unions.

However, despite the fact that in the cases **FD001** and **FD003** have the same amount of training examples, the truth is when the time window approach is applied to each of these examples of time series of a turbofan, the least amount of temporary windows generated, ie, examples, is in the dataset **FD001**. Then the number of parameters that networks should have can not be greater than the number of points in this dataset, in otherwise it would occur **over-fitting**.

In this way, the training-set size of **FD001** is maintained in only 100 examples of units since this is also useful to test the generalization capacity of the models in the simplest and smallest case of training-set. Moreover, the models must also be able to generalize the more complex cases such as **FD002** and **FD004** in which all the datasets are used for their training and where the biggest obstacle lies in the number of faults that the Neural Network must assimilate. It is also had that in the particular case of the dataset **FD004**, there are units with time series smaller than those of the rest, these time series are simply omitted for the training of **FD002** because in the testing of this dataset the minimum is 21 and it is simply considered that they are examples that do not contribute to that particular test.

For training of **FD003**, it is not considered **FD002** or **FD004** but **FD001**, because it is wanted to see the capacity of the models to assimilate different quantities of operating settings, and not if this particular simple case can be integrated into the training of **FD002** or **FD004**.

In conclusion, with **FD001** it bis seen the feasibility of the size of the networks when training is with a small dataset and if they are able to generalize well in that case. With **FD003** it is seen the capacity of the networks to assimilate the operating settings. Finally, the ability to integrate more than one failure mode is measured with **FD002** and **FD004** datasets as well as they are the largest of all datasets.

6.2.2. Data preprocessing

An analysis is made on the database **C-MAPSS** for the verification of the existence of data with little change and its subsequent extraction from the database so that finally the data

can be normalized using **feature scaling**.

Data selection

For the optimal working of Neural Networks, redundant data should be avoided since do not provide significant information for prediction, but that also consumes memory, increase training times and in many cases, decrease the accuracy of the model by providing this data noise and uncertainty to the training process. It should be noted that although authors of some studies point out that LSTM are capable of adapting to the noise or characteristics of each database, the truth is that noise reduction is always better.

From the above, the dimensions of the databases used in **FD001** and **FD003** are: *Cycles, Sensor Measurement 2, Sensor Measurement 3, Sensor Measurement 4, Sensor Measurement 7, Sensor Measurement 8, Sensor Measurement 9, Sensor Measurement 11, Sensor Measurement 12, Sensor Measurement 13, Sensor Measurement 14, Sensor Measurement 15, Sensor Measurement 17, Sensor Measurement 20, Sensor Measurement 21*. While in **FD002** and **FD004** cases, in addition to the previous dimensions, the 3 operating settings given are also used.

RUL edition on databases

The RULs of the databases are modified by setting them to a particular value found from the histograms of the RULs without modifying. This can be understood if it is taken into account that the data representing the behavior of the turbofan after the fault should show RULs with a frequency of constant appearance. Then it is modified the RULs that are inside the data set before the failure (data that in turn show a nominal behavior) setting them to a particular value found from the histograms of the RULs without modifying and that denotes the largest RUL with greater frequency of appearance..

Finally, It can be quickly compared if the editing of the RULs improves the predictions taking any dataset and comparing its results with respect to the case without editing.

Normalization

The data is normalized using **feature scaling**, for which the following normalization applies:

$$x' = \frac{x - x_{min}}{x - x_{max}} \quad (6.1)$$

Finally, all training datasets are divided into training set and validation set. The validation set represents 15 % of the original training set and serves to adjust the hyperparameters of the Neural Networks.

6.3. Programming and start-up of a ConvRNN

The programming of all the networks is done in **Python 2.7**, using the complementary libraries **Tensorflow gpu 1.8** [Abadi et al., 2015], **Scikit Learn** and **Numpy**. On the other hand, the code for the ConvLSTM and ConvJANET cells are elaborated from the modification of the cell code of a ConvLSTM found in the Tensorflow library², while **Chrono initializer** to initialize biases of the ConvRNN is modified from the work [Jos van der Westhuizen and Joan Lasenby, 2018] code³ so in this case the kernels bias for convolutions are initialized.

Finally, all Neural Network models are executed in a GPU **Nvidia GeForce GTX 1080**, using an Nvidia graphics card driver **CUDA Toolkit 9.0**, in the operating system Linux version 16.4 of the Smart Reliability and Maintenance Integration Laboratory (SRMI) of the Mechanical engineering Department of the faculty of physical and mathematical sciences from the University of Chile.

One extra argument per edition of RULs

In addition to the cost function presented in Section 3.4.1, it is considered an extra weight to avoid predictions greater than the number to which the RULs are fixed before failure.

Then the cost function used in the experiments is:

$$E_n = \sum_{i=1}^B (y_i - \mu(x_i))^2 + \sum \|W\| + \lambda \sum_{i=1}^B (\text{ReLU}(\mu(x_i) - \text{RUL}_{\text{edition}}))^2 \quad (6.2)$$

6.4. Hiperparameters setting

The search for models is done manually using only **FD001** in the first instance. For search, it is taken a maximum number of filters of 100, a maximum number of ConvRNN layers of 10, a maximum number of hidden neurons in the full-connected part of 2000 and a maximum number of 3 layers in the same part. In the meantime, there are tested as activation functions **ReLU**, **ELU**, **sigmoid** and **Tanh**.

The models that present a number of parameters greater than the number of points in the dataset **FD001**, are immediately discarded, while the best 10 models that present the

²https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/ConvLSTMCell

³<https://github.com/JosvanderWesthuizen/janet>

best Root Mean Square Error (RMSE), the shorter training time and fewer parameters are selected in validation.

Based in all before, the models founded are tested in the rest of the datasets , choosing 1 of each category taking into account the same standards used in **FD001**.

Finally, to achieve better generalizations, it is considered the technique of L1 regularization for the hidden layer of the Full-connected layer in the four selected models, i.e, a **ConvLSTM** model, a **ConvJANET** model, a **ConvLSTM Encoder-Decoder** model and a **ConvJANET Encoder-Decoder** model, that are tested 20 times getting 20 RMSE, 20 Scores, 20 training times, a graph showing the network setting on the dataset and another graph comparing RMSE over time in the cross validation and in the training-set.

This evaluation of 20 times is done to reduce the variance found in the process of training a Neural Network and thus make a good comparison with the rest of Neural Networks, whether it is considered only those of this work, or if it is considered other previously published and discussed in the Chapter Related Work.

7 | Results

This chapter shows the results obtained from the evaluation of the proposed Neural Networks in 4 datasets (**FD001**, **FD002**, **FD003**, **FD004**) from their training using **ADAM Optimizer** and evaluation in 3 datasets (**{FD001}**, **{FD001, FD003}**, **{FD001, FD002, FD003, FD004}**), as mentioned in the previous chapter. In this case it is looked for models that have a number of parameters less than the training set **FD001** to avoid overfitting in that case (the smallest training set of all) besides the search of the models with the minimum amount of parameters such that they achieve good results.

A manual search of most hyperparameters is done in the training set **FD001**, selecting from the validation in the same training set. Other more specific hyperparameters of each data set, such as steps, batch size or learning rate, are found from the validation in each particular set.

7.1. Dataset Histograms

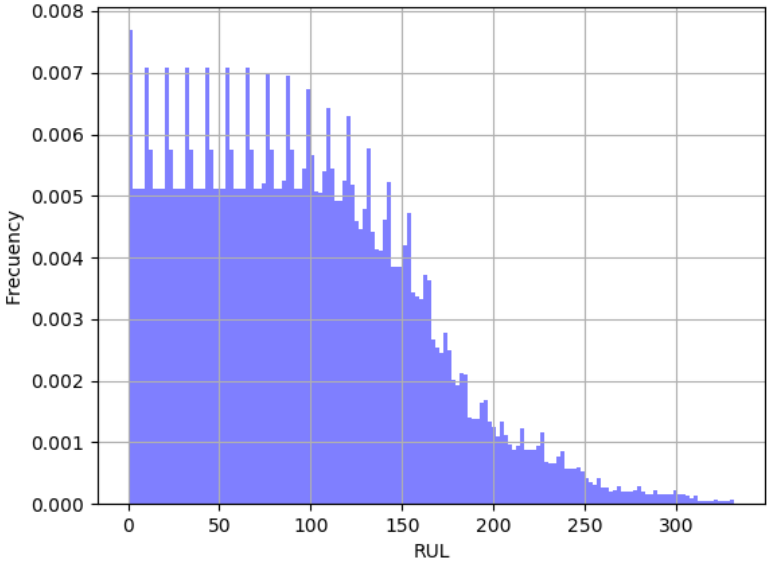


Figure 7.1: Histogram of FD001 dataset when its RULs have not been edited.

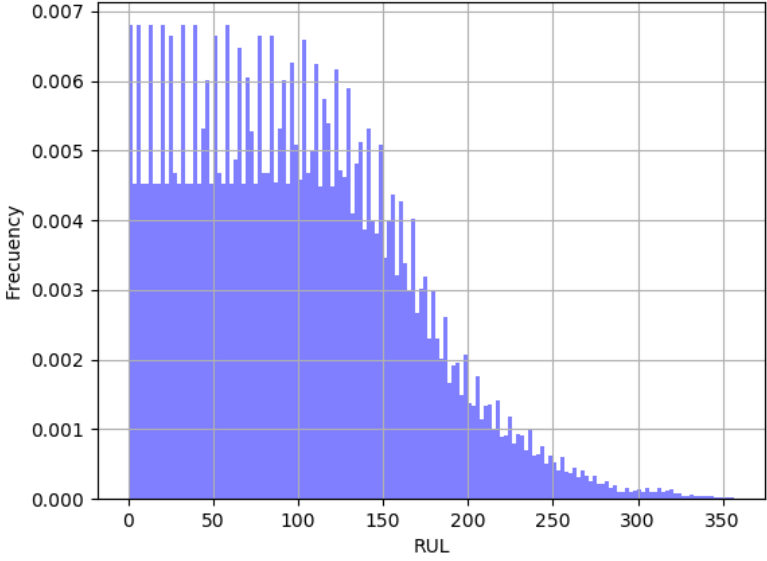


Figure 7.2: Histogram of FD002 dataset when its RULs have not been edited.

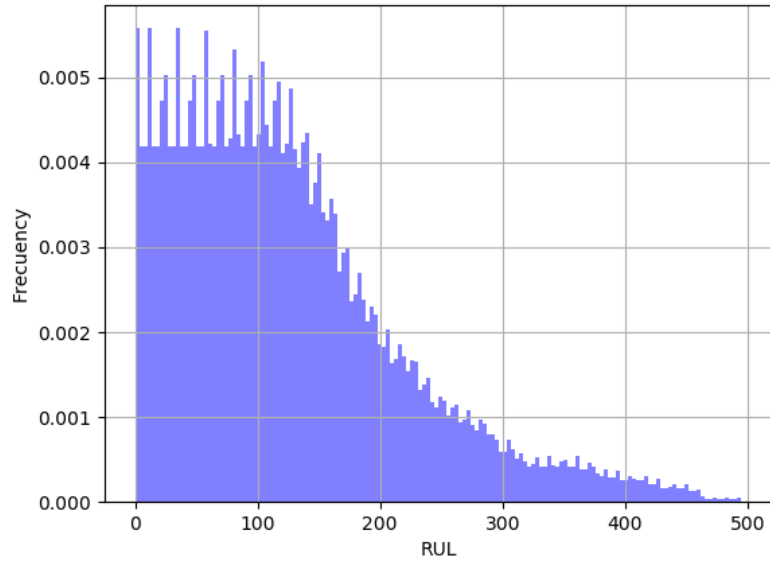


Figure 7.3: Histogram of FD003 dataset when its RULs have not been edited.

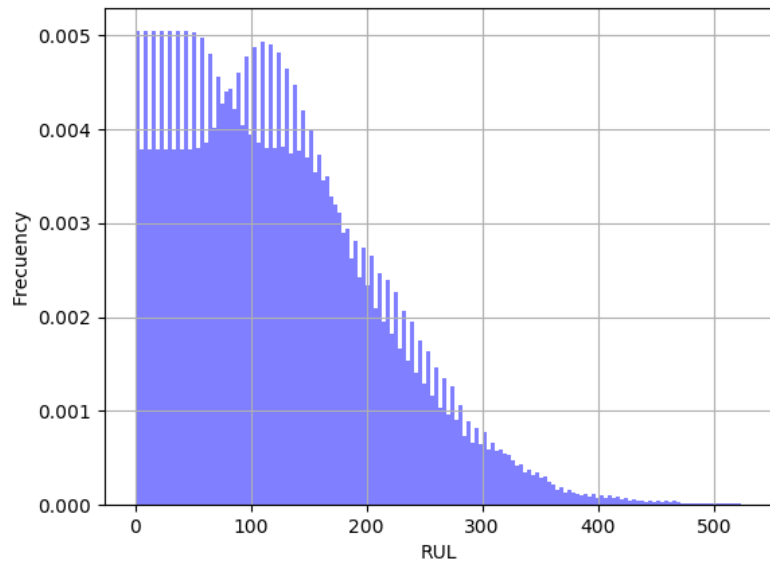


Figure 7.4: Histogram of FD004 dataset when its RULs have not been edited.

7.2. Training settings

It is found that in spite of being different datasets, the same Learning rate of 0.001 can be used, there being no significant difference with respect to a lower Learning rate. A smaller Learning rate increases the number of steps necessary for the network to be trained and evaluated. Batch size of 1024 is used for all cases.

The number of examples used in **FD002** and **FD004** are close to triple and double what they would be if only their respective datasets are used.

Moreover, it can be compared 2 parallel cases such as **FD001** and **FD004**, the first being a simple and small size set (10 % of the size of **FD004**), and the second a complex and large set. In this way, it can be seen the feasibility of the models for the particular use of **Prognosis**.

	FD001	FD002	FD003	FD004
Examples for Training	15071	111738	33618	111738
Examples for validation	2659	19718	5932	19718
Window size	30x15	21x18	30x15	19x18
Batch size	1024			
Learning rate	0.001			
$\lambda(\text{withinLossfunction})$	100			
Steps	2000	30000	20000	30000

Table 7.1: Overview of the training parameters proposed and obtained from validation. It is joined some datasets for the evaluation of the networks in the four dataset but with three datasets for training. More details in section 6.2.1

7.3. Recurrent convolutional Neural Network models and their variations Encoder-Decoder.

From validation different proposals of network models are obtained, being of special interest those configurations with a small amount of parameters, i.e, less memory usage.

A particularly remarkable case is when 2 layers ConvRNN of 10 filters each are used. A filter size of (15,1) is used in the first layer, a filter size of (4, 1) and 100 hidden neurons in a Full-connected layer. It is discovered that replacing the first ConvRNN with a simple CNN and using the same network parameters (same number of filters and filter size), it can be obtained similar results to 2 ConvRNN.

Again, it is also found that 100 hidden neurons in the Full-connected layer is enough for the four models studied.

The architectures obtained for the four cases studied are summarized in Table 7.3, where boxes show information separated by a comma where the first digit indicates the number of layers and the second the number of neurons or convolutional filters.

Architecture	CNN	ConvLSTM	ConvJANET	Full-connected
Deep ConvLSTM	1,10	1,10	-	1,100
Deep ConvJANET	1,10	-	1,10	1,100
Deep ConvLSTM E-D	1,2	2,2	-	1,100
Deep ConvJANET E-D	1,2	-	2,2	1,100
Activation function				
Tanh	No	(only in cell)	(only in cell)	Yes
Temperature	-	3	3	3

Table 7.2: Models found for ConvLSTM, ConvJANET, ConvLSTM Encoder-Decoder, ConvJANET Encoder-Decoder cases.

Each of these models is trained using each of the training sets mentioned in Table 7.2, thus giving 16 evaluation possibilities. The measurement parameters used to evaluate each model in each dataset are: **RMSE**, **score** and **training time**.

7.4. Testing results

All the models are trained and tested 20 times in order to decrease the variance given by the stochastic training process (batch components chosen randomly) because despite the process itself, also in each training case the dataset is selected from a randomly 85% of the training database for training, while 15% for validation. Therefore, the training data domain is always different.

Below are the results of the four models when are tested 20 times in **FD001**, **FD002**, **FD003** and **FD004**. Then a total of 16 cases of **RMSE**, **score** and **training time** are evaluated 20 times.

Model	FD001		
	RMSE	Score	Training time
ConvJANET	12.84 ± 0.37	321.25 ± 26.9	18.28 +- 0.14
ConvJANET E-D	12.67 +- 0.27	262.71 +- 18.93	40.02 ± 0.25
ConvLSTM	12.92 +- 0.20	336.64 +- 20.93	18.72 ± 0.07
ConvLSTM E-D	12.84 ± 0.23	285.43 ± 19.37	50.73 +- 1.23

Table 7.3: ConvLSTM, ConvJANET and their varieties Encoder-Decoder evaluated in FD001.

Model	FD002		
	RMSE	Score	Training time
ConvJANET	16.66 ± 0.6	1542.49 ± 238.6	267.42 +- 0.98
ConvJANET E-D	16.19 +- 0.23	1401.95 +- 251.11	527.43 ± 0.52
ConvLSTM	17.59 +- 0.55	1645.9 +- 198.28	277.33 ± 1.6
ConvLSTM E-D	16.33 ± 0.38	1532.9 ± 243.1	659.83 +- 0.69

Table 7.4: ConvLSTM, ConvJANET and their varieties Encoder-Decoder evaluated in FD002.

Model	FD003		
	RMSE	Score	Training time
ConvJANET	11.79 +- 0.48	246.97 +- 27.12	189.49 +- 0.81
ConvJANET E-D	12.8 +- 0.45	333.79 ± 60.79	395.84 ± 0.34
ConvLSTM	12.39 ± 0.45	288.99 ± 54.33	190.27 ± 0.94
ConvLSTM E-D	12.54 ± 0.33	298.64 +- 39.16	502.36 +- 0.72

Table 7.5: ConvLSTM, ConvJANET and their varieties Encoder-Decoder evaluated in FD003.

Model	FD004		
	RMSE	Score	Training time
ConvJANET	19.55 ± 0.3	2259.53 +- 185.71	255.40 +- 0.51
ConvJANET E-D	19.15 ± 0.28	2282.23 ± 226.58	490.95 ± 0.63
ConvLSTM	20.75 +- 0.81	2513.57 +- 287.81	266.11 ± 1.51
ConvLSTM E-D	19.53 ± 0.23	2316.28 ± 180.84	615.03 +- 0.65

Table 7.6: ConvLSTM, ConvJANET and their varieties Encoder-Decoder evaluated in FD004.

7.5. Behavior of the models in databases.

To see that the networks are able to fit each particular dataset is to measure the results of **RMSE** every 200 steps in the training using a batch of training and another batch of validation.

The prediction of RULs made by the models with respect to the real RULs are also shown. Each case represents a turbofan unit whose results are sorted in descending order for easier understanding. As according to [Saxena et al., 2008] the models can be considered as equivalents in the same dataset and it can be interpreted the results of the predictions ordered in a decreasing way as the predictions that the models would have during the life of a turbofan .

As explained in Section 3.4.4 and has been repeated previously, it is used **ADAM Optimizer** for training, so it is not necessary to evaluate multiple Learning rates as this technique allows the automatic adjustment of the Learning rate in every step.

However, when it is used Neural Networks with few parameters together with regularizing all the models for sparsity in the Full-connected layer weight matrix, it is expected that the evaluated networks present great generality in their results and with this the difference of RMSE among evaluated data in the training set and validation, low.

Finally, since a temperature of 3 is used in all **Tanh** activation functions (including those found in ConvRNN cells), it is expected to obtain more scattered and less conservative results that in this particular case is useful. It is wanted that the models can deal with the uncertainty given by the operating setting together with the failure modes. The randomness of these processes prevents that there is an exact prediction of any model, then it can not be conservative regarding the possibilities of RUL to predict.

7.5.1. Adjustment and predictions of ConvLSTM in FD00 1, 2, 3 and 4.

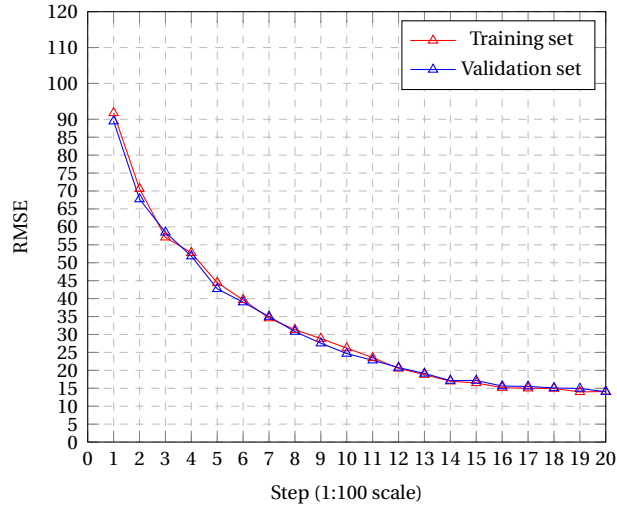


Figure 7.5: Accuracy of validation and training sets vs training steps for ConvLSTM in FD001.

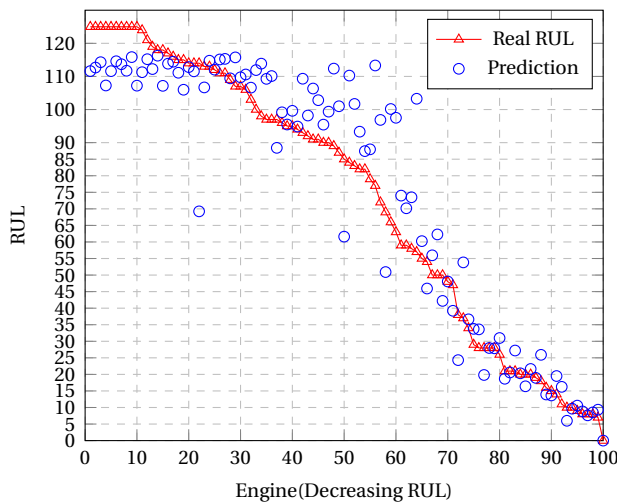


Figure 7.6: RUL predicted for ConvLSTM in FD001.

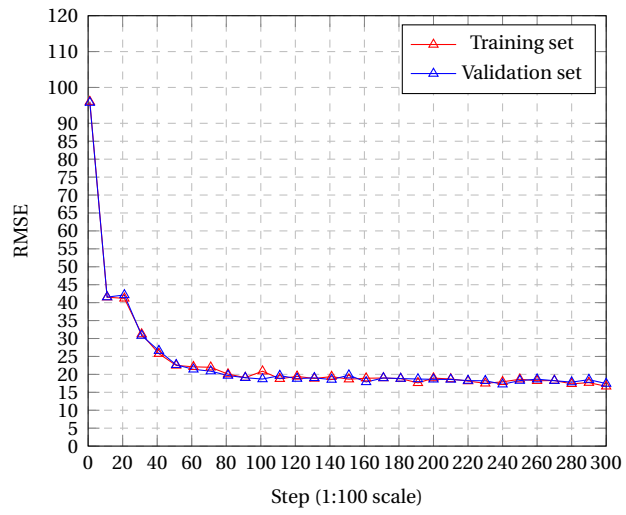


Figure 7.7: Accuracy of validation and training sets vs training steps for ConvLSTM in FD002.

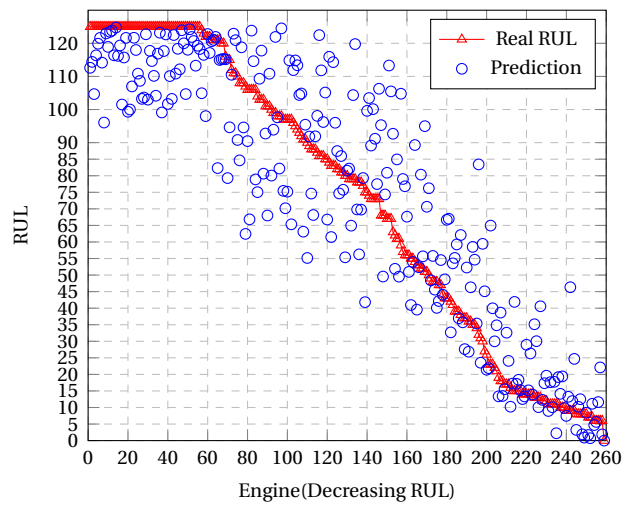


Figure 7.8: RUL predicted for ConvLSTM in FD002.

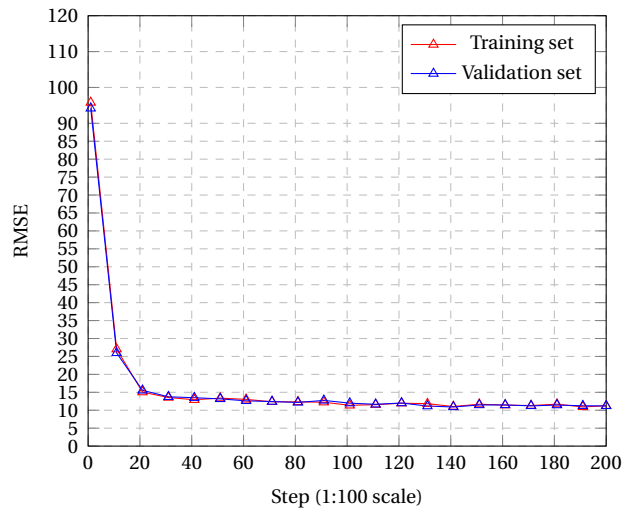


Figure 7.9: Accuracy of validation and training sets vs training steps for ConvLSTM in FD003.

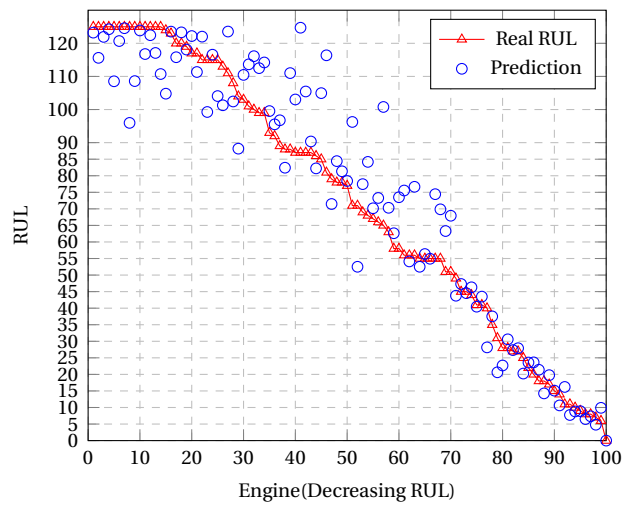


Figure 7.10: RUL predicted for ConvLSTM in FD003.

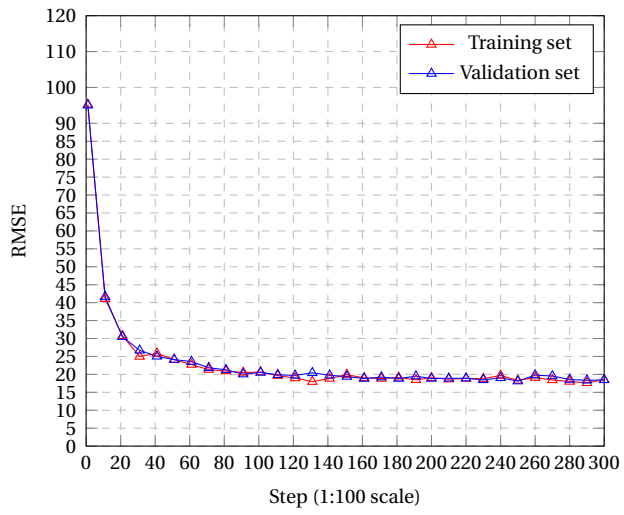


Figure 7.11: Accuracy of validation and training sets vs training steps for ConvLSTM in FD004.

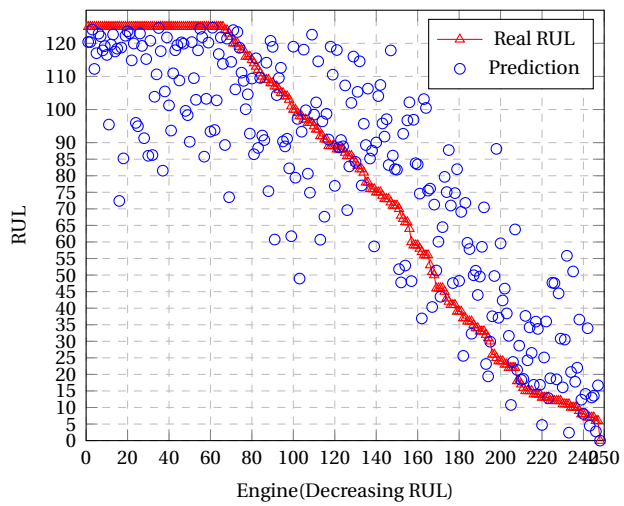


Figure 7.12: RUL predicted for ConvLSTM in FD004.

7.5.2. Adjustment and predictions of ConvJANET in FD00 1, 2, 3 and 4.

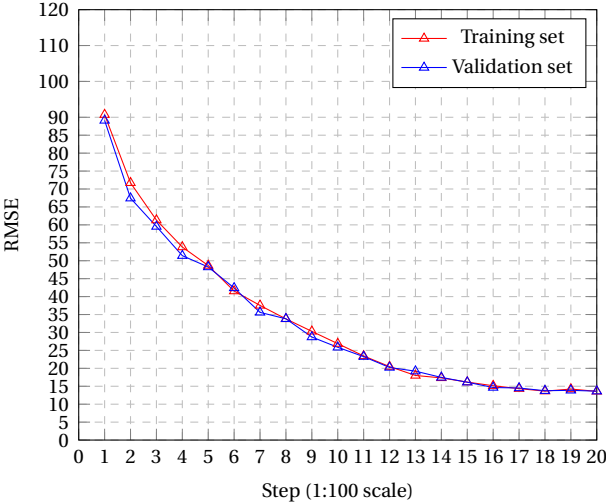


Figure 7.13: Accuracy of validation and training sets vs training steps for ConvJANET in FD001.

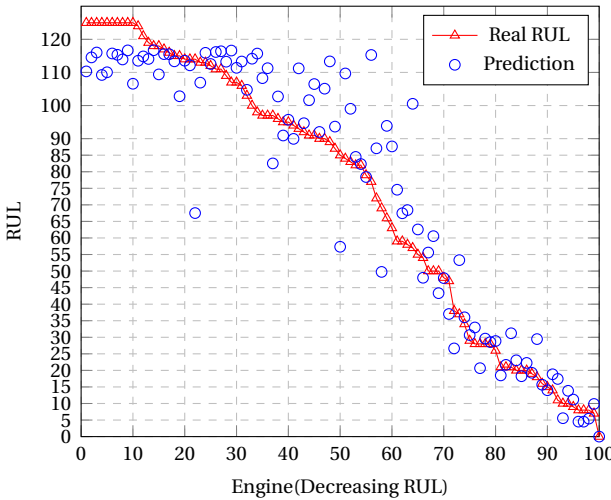


Figure 7.14: RUL predicted for ConvJANET in FD001.

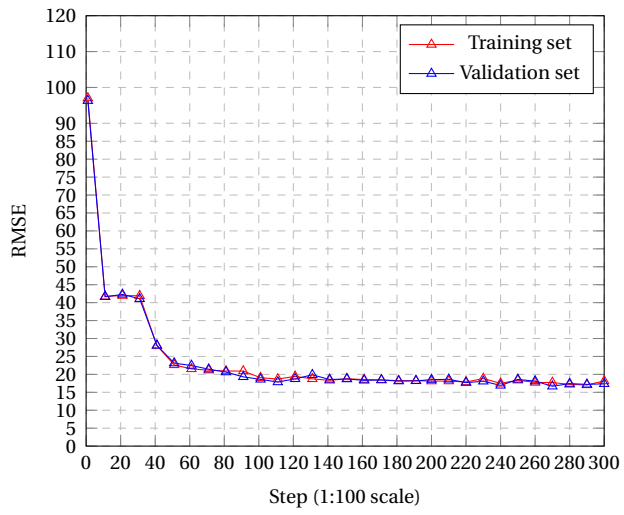


Figure 7.15: Accuracy of validation and training sets vs training steps for ConvJANET in FD002.

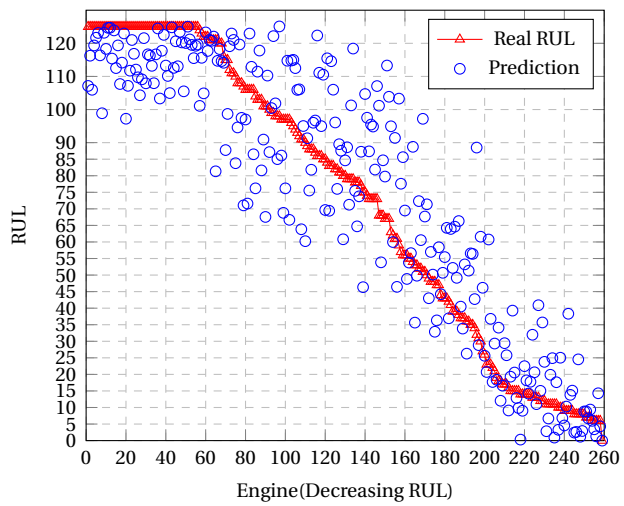


Figure 7.16: RUL predicted for ConvJANET in FD002.

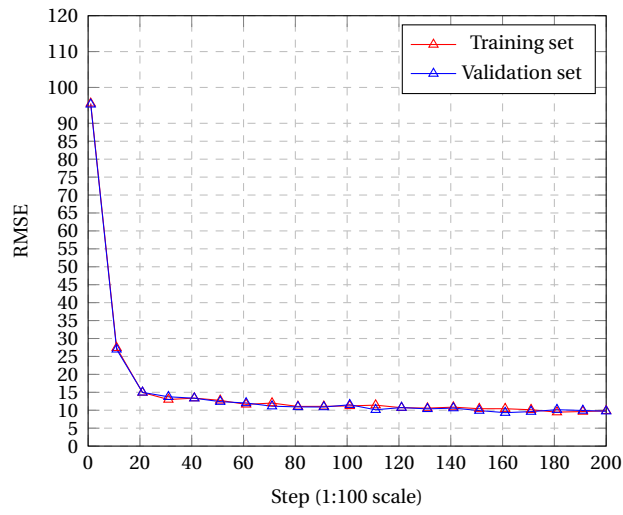


Figure 7.17: Accuracy of validation and training sets vs training steps for ConvJANET in FD003.

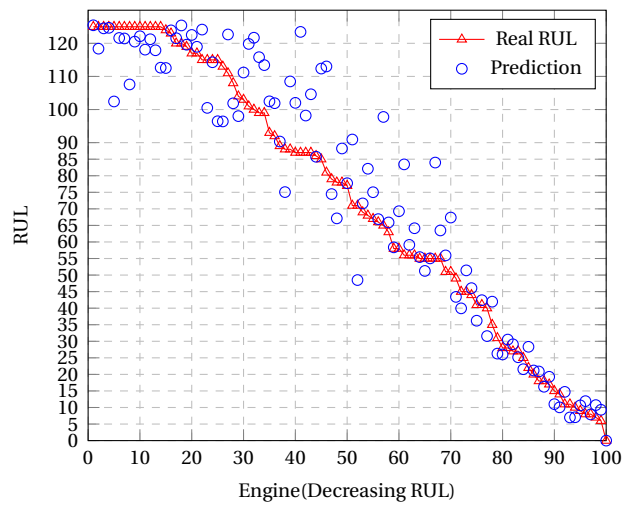


Figure 7.18: RUL predicted for ConvJANET in FD003.

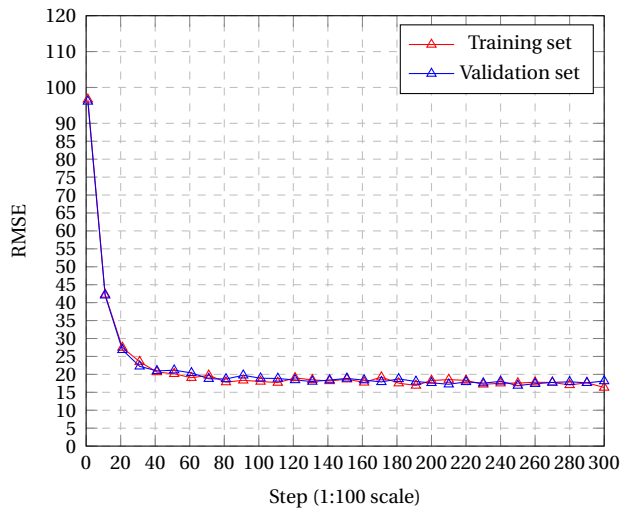


Figure 7.19: Accuracy of validation and training sets vs training steps for ConvJANET in FD004.

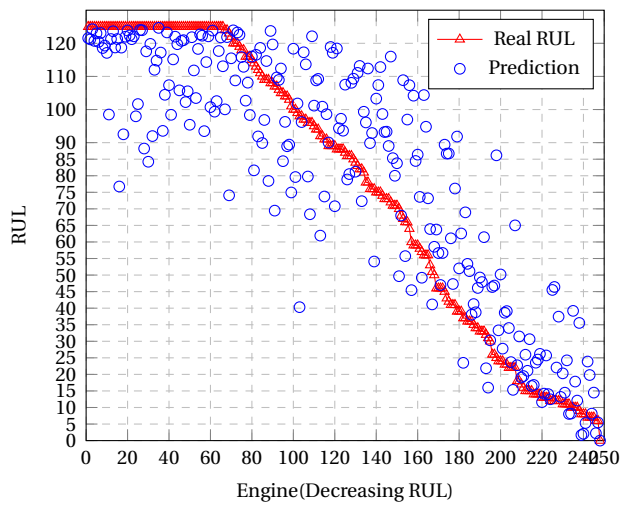


Figure 7.20: RUL predicted for ConvJANET in FD004.

7.5.3. Adjustment and predictions of ConvLSTM Encoder-Decoder in FD00 1, 2, 3 and 4.

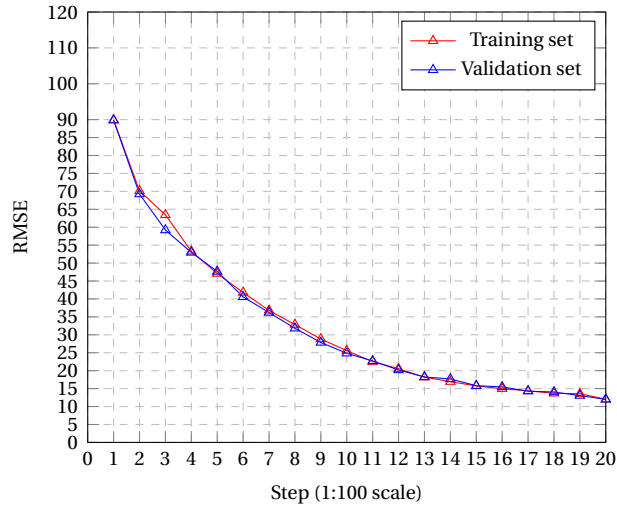


Figure 7.21: Accuracy of validation and training sets vs training steps for ConvLSTM Encoder-Decoder in FD001.

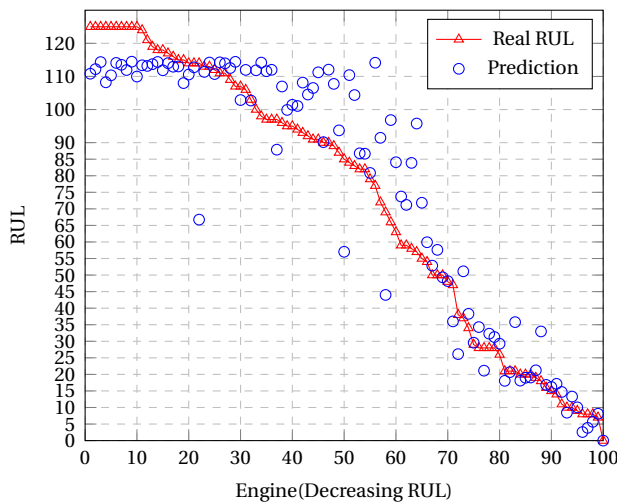


Figure 7.22: RUL predicted for ConvLSTM Encoder-Decoder in FD001.

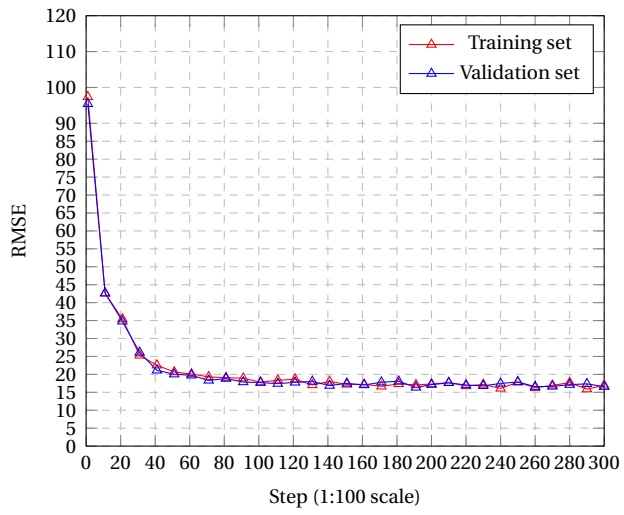


Figure 7.23: Accuracy of validation and training sets vs training steps for ConvLSTM Encoder-Decoder in FD002.

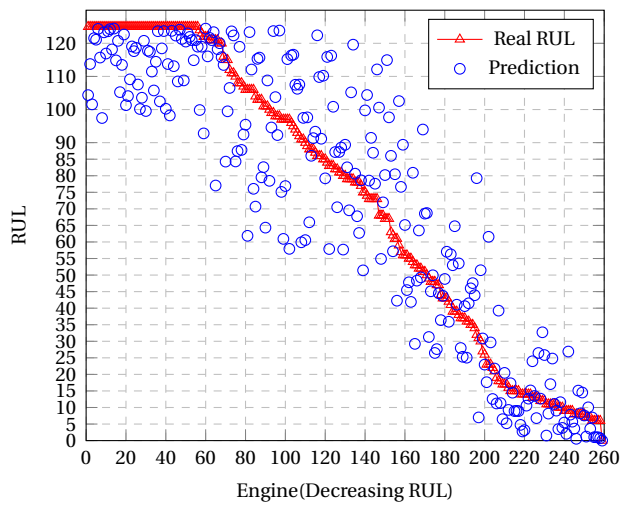


Figure 7.24: RUL predicted for ConvLSTM Encoder-Decoder in FD002.

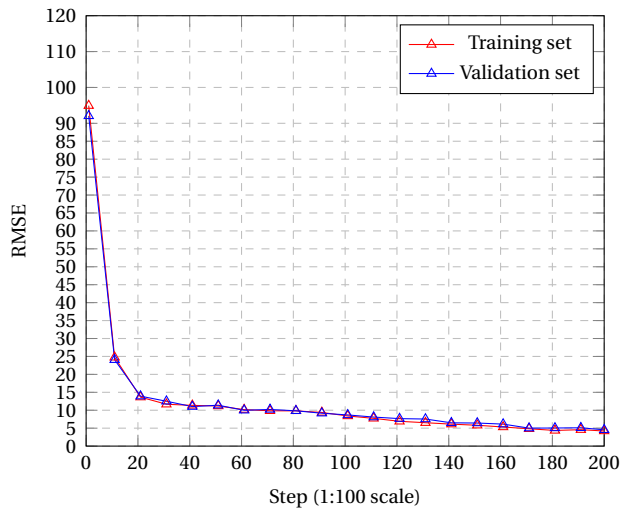


Figure 7.25: Accuracy of validation and training sets vs training steps for ConvLSTM Encoder-Decoder in FD003.

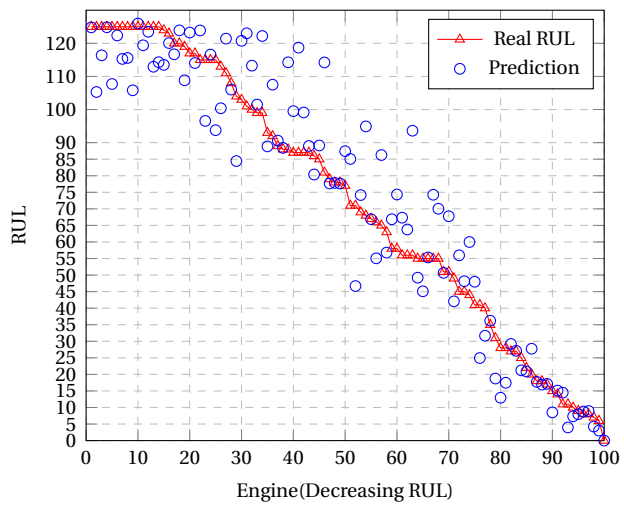


Figure 7.26: RUL predicted for ConvLSTM Encoder-Decoder in FD003.

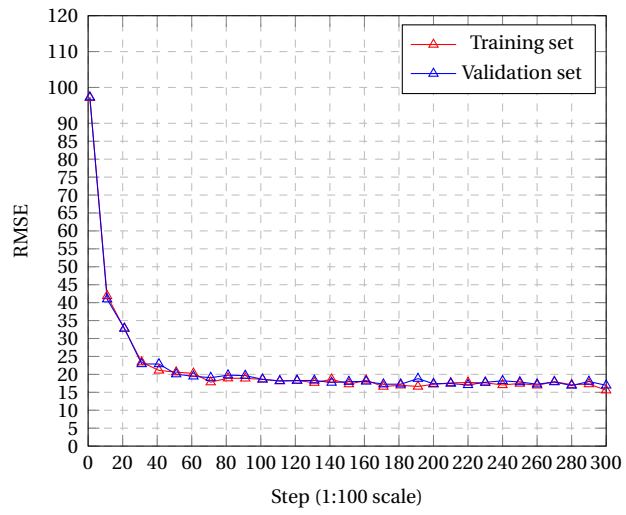


Figure 7.27: Accuracy of validation and training sets vs training steps for ConvLSTM Encoder-Decoder in FD004.

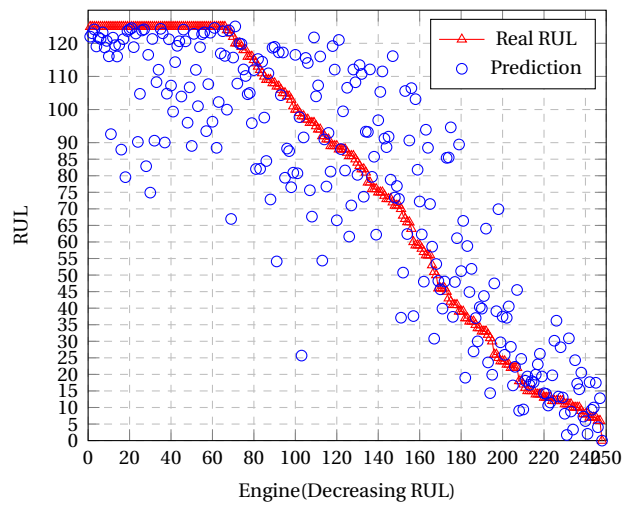


Figure 7.28: RUL predicted for ConvLSTM Encoder-Decoder in FD004.

7.5.4. Adjustment and predictions of ConvJANET Encoder-Decoder in FD00 1, 2, 3 and 4.

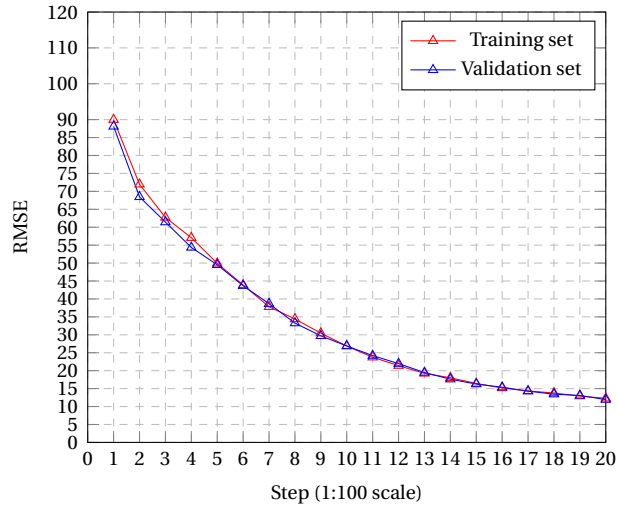


Figure 7.29: Accuracy of validation and training sets vs training steps for ConvJANET Encoder-Decoder in FD001.

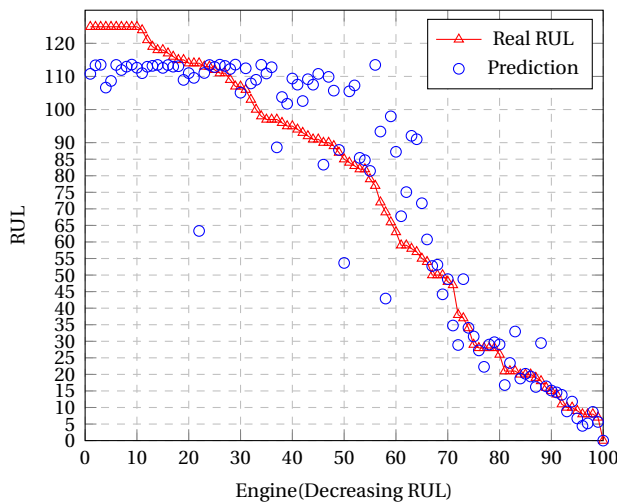


Figure 7.30: RUL predicted for ConvJANET Encoder-Decoder in FD001.

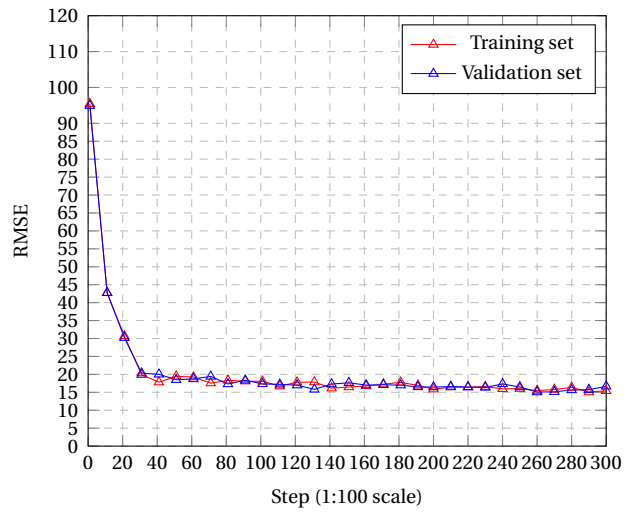


Figure 7.31: Accuracy of validation and training sets vs training steps for ConvJANET Encoder-Decoder in FD002.

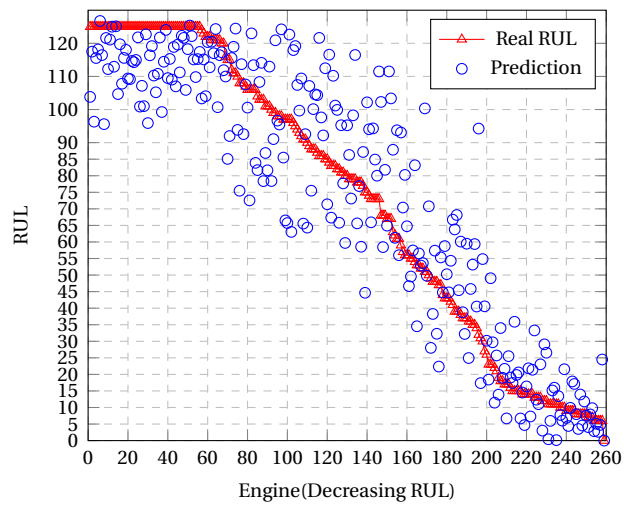


Figure 7.32: RUL predicted for ConvJANET Encoder-Decoder in FD002.

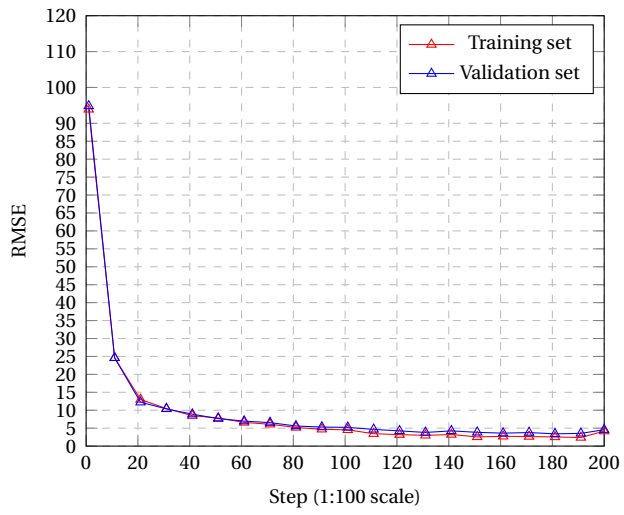


Figure 7.33: Accuracy of validation and training sets vs training steps for ConvJANET Encoder-Decoder in FD003.

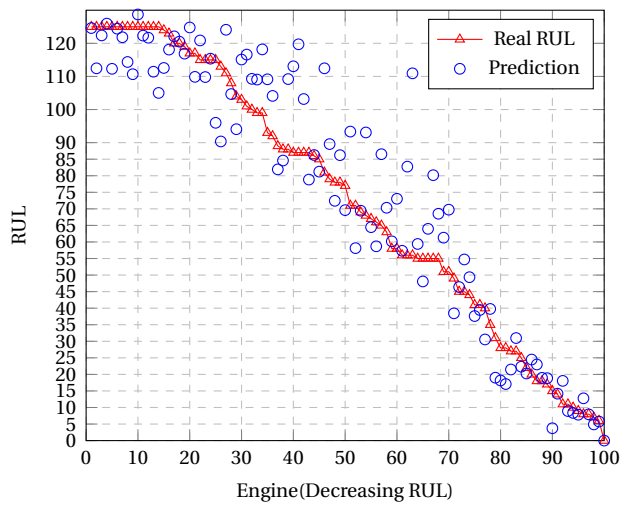


Figure 7.34: RUL predicted for ConvJANET Encoder-Decoder in FD003.

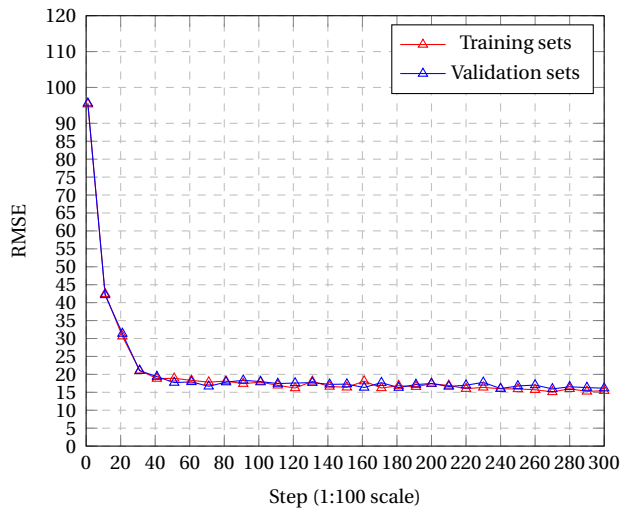


Figure 7.35: Accuracy of validation and training sets vs training steps for ConvJANET Encoder-Decoder in FD004.

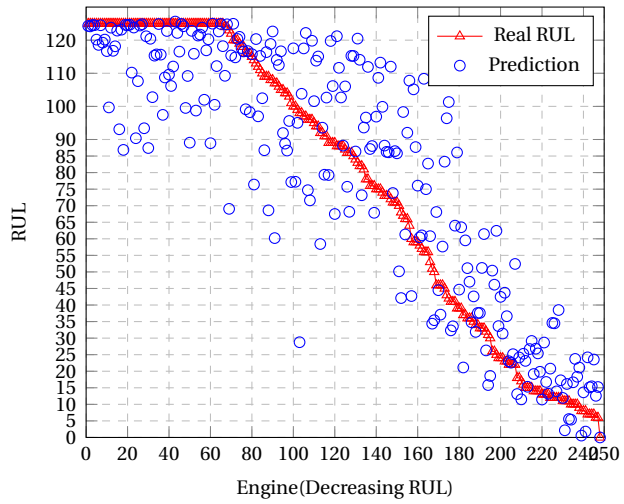


Figure 7.36: RUL predicted for ConvJANET Encoder-Decoder in FD004.

8 | Discussion

For this work it is tried to use a probabilistic approach about the use of Neural Networks for the RULs prediction in a mechanical system with the idea of trying to obtain the best Neural Network model and having a better understanding about the Neural Network model that presents the best results, ie, to understand that the best Neural Network model represents a function that approximates the maximum value of a conditional probability distribution in which a RULs is associated to an array or window of a vector of data in a time interval.

As the problem is presented in Chapter 5, the problem to be solved can be considered deterministic since it is wanted to determine a RULs with uncertainty (error σ) associated with the variable to be determined. Therefore, it is proposed **Deep Convolutional Recurrent Neural Networks** that given a Ω database can be adjusted such that its maximum likelihood is maximized. According to the latter, it is used as a cost function to minimize **Residual sum of squares**.

8.1. Problem feasibility

As it was seen in chapter 2, 3 authors [Kurosaki, 2003], [Urban, 1973] and [Doel, 2002] propose the use of operational data for the Diagnosis of faults, while [Goebel et al., 2007] proposes to solve the problem of Prognosis also using the operational data of a machine. However, [Ramasso, 2014] and [Goebel et al., 2007] also pointed out that trying to solve a problem of Prognosis before the failure can be a complicated problem since the data should be shown not too far away of the nominal behavior.

Therefore, in this work RULs greater than 125 are fixed to that same value, this value can be interpreted as the RUL when the reliability is greater than 90%. This value is obtained by simple inspection of the histograms (Figures 7.1, 7.2, 7.3 and 7.4) of the training data, in which it can be seen that in general the values lower than 125 have a similar frequency (they always happen), which together with knowing that the distribution that minimizes entropy is that where the frequency is the same for each event [Murphy, 2013], makes

believe that the data relative to the RULs between 0 and 125 show a lower entropy than those greater than 125 since the frequencies of occurrence of a RULs between 0 and 125 is closer to a constant and are associated with the values of the operational data after a fault (more information).

The proposed value to fix RULs, is a simple approximation also obtained in other works [Sateesh Babu et al., 2016] and [Li et al., 2017], but as it is proposed here, the value of the RULs when the reliability is greater than 90% serves as a more general method to improve the results of Prognosis in other possible mechanical systems using precisely the reliability values to fix the values of RULs to predict.

Then, for first instance, the performance of the networks is evaluated in 2 cases. The first case is to not fix RULs greater than 125, while the second RULs edited and fixed to 125. The performances are compared and note that the difference of RULs in some cases are greater than 20.

A possible explanation of this fact is that the proposed models are not able to differentiate between the data given before the moment of failure, since these data is similar and close to the nominal ones. Therefore, the Networks could tend to give the values that are probably the most repeated in the dataset within an interval.

In any case, establishing a constant value for the RULs of the data before the failure could help the predictions since a constant value is being associated to a group of values that also vary little . Particularly in the context of PHM, this constant value evidently should be the lowest value of RULs before the failure and could be determined in future instances from the reliability given in mechanical system.

8.2. Performance of Neural Networks and related aspects

From Table 7.2 it can seen that the same Learning rate is used for all cases of database and Neural Networks, obtaining the same results as when it is used a lower Learning rate but better than when it is used a higher Learning rate. In addition, the Learning rate used, 0.001, allows to use a smaller number of steps than a lower Learning rate, decreasing the training times and finally evaluation in this work.

Probably, this apparent insensibility of the Learning rate between the values 0.001 and 0.0001, can be due to the same mechanism of in which Learning rate varies in each step, the value may is 0.001 a good start value.

Again, it is interesting to note that a ConvRNN layer can be replaced by a CNN as can be seen from the models exposed in Table 7.3. Probably in this part of the extraction of characteristics, convolution has more weight than recurrence.

Moreover, product of the convolution use in the second layer, i.e, in the ConvRNN layer, a linear combination is obtained that includes, for the half of the series of time, all times of the window. Then, it can be believed that this allows a greater amount of information for recurrence, which also allows a better extraction of temporal characteristics. It is should also remember [Saxena et al., 2008] that during failures, the efficiency of the component that is failing presents a random and decreasing character over time, in such a way that, although there is a clear trend in time, the truth is that at each point of time there is not clearly defined values for the operational data. Then, the application of convolution can be understood as a way of smoothing the dispersion between the data of different times.

From the results previously explained in Tables 7.4, 7.4, 7.4 and 7.4, it can be seen that the Network that generally obtains better results regardless of whether it is used normally or as **Encoder-Decoder** architecture, is the ConvJANET type. If it is measured **RMSE**, **Score** or **Training time**, ConvJANET is the one that gets the best results. This probably has to do with the smaller number of parameters that it uses and that decrease the entropy of the model because in general it should have a smaller amount of redundant parameters. However, in the same line, it is also possible to see a good performance of **Encoder-Decoder** models that in all cases, with exception of **FD003**, achieve the best results regarding **RMSE** and **Score**, then it can be thought that they obtain better results because these models are able to process the entered data with greater property. Therefore, using a smaller number of parameters, the **Encoder-Decoder** models are capable of even obtaining better results than normal cases that have a greater number of parameters. Moreover, it is also notable that the **Encoder-Decoder** models take a longer time to train, this possibly has to do with the fact that their number of parameters is also lower than in normal cases ¹.

In this way, it can be thought that the **Encoder-Decoder** models are more efficient than their normal pairs.

In the case of testing in the dataset **FD003**, it can be seen that the best and worst value are given in the types **ConvJANET**, of which the normal case stands out as the best with a **RMSE** of 11.79 ± 0.48 , while the model **Encoder-Decoder** is the worst case with a **RMSE** of 12.8 ± 0.45 . It can be believed that in this particular case the number of convolutional filters takes precedence over the processing that a **Decoder** can give, and the lack of parameters in the model **ConvJANET Encoder-Decoder** may be the cause of this discrepancy. In spite of this, it is also observed that in 3 of 4 cases the model **ConvJANET Encoder-Decoder** presents the best results in terms of **RMSE** and in 2 of 4 with respect to **Score**.

So, it can be thought that the model **ConvJANET Encoder-Decoder** maybe is the best proposed model since in most cases it presents the best results both when evaluating **RMSE** and evaluating **Score**. It can also be thought that this model can be improved by

¹In general, note that this occurs when the models with a low number of parameters in large databases are made.

increasing the number of convolutional filters along with using the regularization technique **Dropout**.

Below in the Figure 8.1 it can be seen the possible best proposed model. In it can be noticed a first CNN layer with 2 size kernels (15,1), then a second layer, which is the Encoder, ConvJANET with 2 size kernels (15,1), of which hidden states are copied and given as Initial State to the Decoder with 2 size kernels (15,1) for the prediction to be made in a Full-connected layer with 100 hidden neurons and Tahn as activation function.

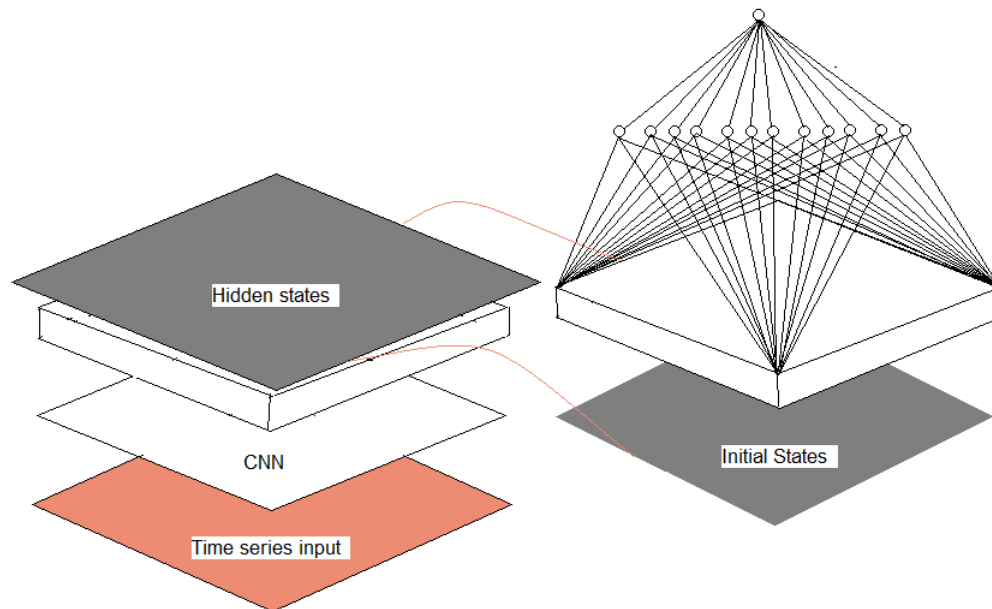


Figure 8.1: Fully-connected ConvJANET Encoder-Decoder that presents the minor RMSEs and Scores. The model consists of a first CNN layer of 2 size filters (15,1), then a ConvJANET Encoder from which hidden states are copied and given to the ConvJANET Decoder as initial states. Finally a fully-connected layer, with Tahn activation function, make a prediction. (Own elaboration).

8.3. Models convergence

In Section 7.5, figures are shown regarding the convergence of the models during their training as when they are tested. In general, it can be seen that all the models evaluated converge well, since **Encoder-Decoder** cases are the ones that converge in the best way. If it is seen Figures 7.3, 7.11, 7.19 and 7.27, it can be noticed that when **Encoder-Decoder** models are trained and tested in the dataset **FD002**, they have a less erratic convergence than the normal models. In all figures of Accuracy of validation and training set vs train-

ing steps, it can be observed a little difference between the RMSEs given for the training batch and the validation batch. It can be explained by the fact that the Networks are of few parameters and a regularizer is also used for the Full-connected layer. Then, all models are able to generalize very well as well as the fact that the batch size is 1024.

When it is verified the dispersion of the predictions made by the models respect to the real RULs, it can be noted that models tested in **FD001** dataset tend to establish a RULs less than 125, and close to 115. This type of dispersion in this particular set is similar to the one seen in the results of [Zheng et al., 2017].

While in the rest of datasets, it can be seen in general a dispersion that shows networks that are not very conservative and with a random error because there is no clear prediction direction, i.e., the prediction can be larger or smaller without a clear preference. Different is the case of the predictions made for values of 125. In them it can be seen a clear predilection of the models by values equal to or less than 125, an issue that can be explained by the extra argument that it is given to the Loss function in Equation 6.2.

Finally, one can observe a large dispersion of the predictions in relation to real RULs, noting that it decreases when approaching zero. This last result, is also shared by other works mentioned in section 2.2.3, and that probably has to do with the decrease in uncertainty given by the data in a shorter period of time to forecast but also implies that the error variance is different of a constant value.

9 | Conclusion

The study of the Prognosis problem in mechanical systems is an interesting problem nowadays because a good Prognosis allows to lower costs, to diminish the risks of catastrophic failure and to grant an extra time of use if is necessary in components or systems that already present failure or failures.

In this work a probabilistic approach is taken to raise the problem of Prognosis in a mechanical system by **Deep Convolutional Recurrent Neural Networks**. From this, 4 models of **ConvRNN** are studied on the 4 databases **C-MAPSS**. These 4 databases have different levels of complexity, while those of minor complexity are subsets of the most complex datasets.

Therefore, first the 4 models are trained and tested with the data of the simplest case (1 failure mode and 1 operating setting). Then, for the case of medium complexity, it is joined the 2 simplest databases to train the models and then they are tested in the second simplest case (1 failure mode and 6 operating settings). Finally, all the databases are combined to train the models and then they are tested in the test sets of the 2 most complex cases (both with 2 failure modes, one with 1 operating setting and another with 6 operating settings). It is noteworthy that the simplest database is remained intact as it represents the smaller database that allows to establish the least amount of parameters.

So, in this way all models are tested in an easy, medium and difficult cases, the latter being extensively tested.

It is found that modifying the RUL greater than 125 and adjusting them to this same value, helps Neural Networks to better assimilate databases. In fact, that corresponds with assigning a constant and convenient value to data entries that represent nominal operational values of a mechanical system. On the other hand, the frequencies of occurrence of RUL between 0 and 125, tend to a distribution that minimizes the entropy of the information. This fact explains why the models in general assimilate well this part of the database understood as that from the moment of failure.

Analyzing the results obtained from the metrics in the test sets, it is seen that in general **ConvJANET** types are those that present a slightly best **RMSE**, **Score** and **training time**.

While in general the type varieties **Encoder-Decoder** are the most efficient and slightly accurate because using 40% of the parameters given in the normal cases, they are able to achieve similar results.

Moreover, if more than 1 **ConvRNN** layer are used, it is possible to replace the first layer with exactly the same parameters for the convolutional kernels and not affect the results, which has to do with the ability of the Convolution to smooth data with high noise or uncertainty, and that in this case it takes greater relevance in the first layer.

Finally, it is possible to train and test the 4 models successfully and it is determined that the best model found is **ConvJANET Encoder-Decoder** because of its a slightly best performance in the metrics **RMSE** and **Score** , and efficiency of their parameters. But also too, it is suggested that this model could be improved by increasing the number of convolutional kernels and hidden neurons of the Full-connected part and using the technique of **Dropout**.

Acronyms

C-MAPSS Comercial Modular Aero-Propulsion System Simulation. 6, 39, 40

CNN Convolutional Neural Network. 7, 50, 71

ConvJANET Convolutional Just Another NETwork. 2, 28, 37, 72

ConvLSTM Convolutional Long Short Term Memory. 2, 26, 37

ConvRNN Convolutional Recurrent Neural Network. iii, 1, 2

JANET Just Another NETwork. 26

LSTM Long Short Term Memory. 7, 22, 26, 43

MLP Multi-Layer Perceptron. 7

PHM Prognosis and Health Management. 7, 33, 71

RGB Red, Green and Blue. 17

RMSE Root Mean Square Error. 46, 53, 76

RNN Recurrent Neural Network. iii, 19, 26

RUL Remaining Useful Life. iii, 1, 2, 7, 8, 34, 36–38, 75

SRMI Smart Reliability and Maintenance Integration Laboratory. 45

Nomenclature

α	Scalar
\mathbf{v}	Vector
\mathbf{M}	Matrix
T	Tensor
σ	Sigmoid activation function
\tanh	Hyperbolic Tangent activation function
\otimes	Multilinear application
η	Learning rate
δ_j	Functions derived by Back-propagation before a weighted input.
c_t^j	Memory cell
f_t^j	Forget gate
h_t^j	Hidden state
i_t^j	Input gate
o_t^j	Output
z_i	Input up to which the Back-propagation algorithm arrives.

Bibliography

- [Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467.
- [B.Arffen and Weber, 2001] B.Arffen, G. and Weber, H. J. (2001). *Mathematical Methods for Physicists, Chapter 4, Tensor Analysis*. St Louis, Missouri, U.S.A.: Academic Pr.
- [Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag Berlin, Heidelberg. <http://www.deeplearningbook.org>.
- [Chung et al., 2014] Chung, J., Çağlar Gülçehre, Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.
- [DeCastro et al., 2008] DeCastro, J., Litt, J., and K. Frederick, D. (2008). A modular aeropropulsion system simulation of a large commercial aircraft engine.
- [Doel, 2002] Doel, D. L. (2002). Interpretation of weighted-least-squares gas path analysis results. 2:53–63.
- [Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.
- [Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterton, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.
- [Goebel et al., 2007] Goebel, K., Qiu, H., Eklund, N., and Yan, W. (2007). Modeling propagation of gas path damage. In *2007 IEEE Aerospace Conference*, pages 1–8.

- [Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- [Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850.
- [Heimes, 2008] Heimes, F. O. (2008). Recurrent neural networks for remaining useful life estimation. In *2008 International Conference on Prognostics and Health Management*, pages 1–6.
- [Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.
- [Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization.
- [Kurosaki, 2003] Kurosaki, M. (2003). Fault detection and identification in an im270 gas turbine using measurements for engine control. 1:385–393.
- [LeCun et al., 1990] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan-Kaufmann.
- [Li et al., 2017] Li, X., Ding, Q., and Sun, J.-Q. (2017). Remaining useful life estimation in prognostics using deep convolution neural networks. 172.
- [Li et al., 2009] Li, Y., Fu, Y., Li, H., and Zhang, S.-W. (2009). The improved training algorithm of back propagation neural network with self-adaptive learning rate. In *2009 International Conference on Computational Intelligence and Natural Computing*, pages 73–76. Exported from <https://app.dimensions.ai> on 2018/10/08.
- [Malhotra et al., 2016] Malhotra, P., TV, V., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., and Shroff, G. (2016). Multi-sensor prognostics using an unsupervised health index based on LSTM encoder-decoder. *CoRR*, abs/1608.06154.
- [Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In Kobayashi, T., Hirose, K., and Nakamura, S., editors, *INTERSPEECH*, pages 1045–1048. ISCA.
- [Murphy, 2013] Murphy, K. P. (2013). *Machine learning: a probabilistic perspective*. MIT Press, 1 edition.
- [N. Dauphin et al., 2015] N. Dauphin, Y., Vries, H., Chung, J., and Bengio, Y. (2015). Rmsprop and equilibrated adaptive learning rates for non-convex optimization. 35.
- [Neubauer, 1984] Neubauer (1984). Remaining-life estimation for high-temperature materials under creep load by replicas. 66.
- [Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). *On the difficulty of training recurrent neural networks*, volume 28 of *Proceedings of Machine Learning Research*. PMLR, Atlanta, Georgia, USA.

- [Peel, 2008] Peel, L. (2008). Data driven prognostics using a kalman filter ensemble of neural network models. In *2008 International Conference on Prognostics and Health Management*, pages 1–6.
- [Ramasso, 2014] Ramasso, E. (2014). Investigating computational geometry for failure prognostics in presence of imprecise health indicator : Results and comparisons on c-mapss datasets.
- [Ramasso and Saxena, 2014] Ramasso, E. and Saxena, A. (2014). Review and analysis of algorithmic approaches developed for prognostics on cmapss dataset.
- [Rausand and Høyland, 2013] Rausand, M. and Høyland, A. (2013). *System Reliability Theory*. John Wiley and sons,INC.
- [Rumelhart et al., 1988] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.
- [Sateesh Babu et al., 2016] Sateesh Babu, G., Zhao, P., and Li, X.-L. (2016). Deep convolutional neural network based regression approach for estimation of remaining useful life. In Navathe, S. B., Wu, W., Shekhar, S., Du, X., Wang, X. S., and Xiong, H., editors, *Database Systems for Advanced Applications*, pages 214–228, Cham. Springer International Publishing.
- [Satriyadi Hernanda et al., 2014] Satriyadi Hernanda, I. G. N., C Mulyana, A., Asfani, D., Yulistya Negara, I. M., and Fahmi, D. (2014). Application of health index method for transformer condition assessment. 2015.
- [Saxena et al., 2008] Saxena, A., Goebel, K., Simon, D., and Eklund, N. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. *2008 International Conference on Prognostics and Health Management*, pages 1–9.
- [Shi et al., 2015] Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and chun Woo, W. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NIPS*.
- [Srivastava et al., 2015] Srivastava, N., Mansimov, E., and Salakhutdinov, R. (2015). Unsupervised learning of video representations using lstms. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15*, pages 843–852. JMLR.org.
- [Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.
- [Tallec and Ollivier, 2018] Tallec, C. and Ollivier, Y. (2018). Can recurrent neural networks warp time? *CoRR*, abs/1804.11188.
- [Urban, 1973] Urban, L. A. (1973). Gas path analysis applied to turbine engine condition monitoring. 10:400–406.

- [van der Westhuizen and Lasenby, 2018] van der Westhuizen, J. and Lasenby, J. (2018). The unreasonable effectiveness of the forget gate. *CoRR*, abs/1804.04849.
- [Wang et al., 2012] Wang, P., Youn, B. D., and Hu, C. (2012). A generic probabilistic framework for structural health prognostics and uncertainty management. 28.
- [Wang et al., 2008] Wang, T., Yu, J., Siegel, D., and 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*, pages 1–6.
- [Werbos, 1990] Werbos, P. (1990). Backpropagation through time: what it does and how to do it. 78:1550 – 1560.
- [Zheng et al., 2017] Zheng, S., Ristovski, K., Farahat, A., and Gupta, C. (2017). Long short-term memory network for remaining useful life estimation. In *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 88–95.

Appendix

```
1 import tensorflow as tf
2 from ConvRecurrentCells import ConvRecurrentCell
3
4
5 def convrecurrent2d(channels, num_input, time_step, filter_channels, kernel_x, name,
6 kind, batch_size, x=None, initial_state=None, skip_connection=False):
7     with tf.variable_scope(name_or_scope=name, reuse=tf.AUTO_REUSE):
8         convlstm = ConvRecurrentCell(
9
10             conv_ndims=2,
11             input_shape=[time_step, num_input, channels],
12             output_channels=filter_channels,
13             kernel_shape=[kernel_x, 1],
14             use_bias=True,
15             kind=kind,
16             t_max=time_step,
17             skip_connection=skip_connection)
18         if initial_state == None:
19             hidden = convlstm.zero_state(batch_size, tf.float32)
20         else:
21             hidden = initial_state
22
23         if x == None:
24             x = tf.zeros([batch_size, time_step, num_input, filter_channels], tf.float32)
25         else:
26             x = tf.convert_to_tensor(x)
27         y_1, hidden = convlstm(x, hidden)
28     return y_1, hidden
29
30 def conv2d(x, W, b, strides=1):
31     x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
32     x = tf.nn.bias_add(x, b)
33     return x
34
35
36
37 def ConvRecurrent(x, arq, filter_channels, kind, num_inputs, time_steps, batch_size,
```

```

kernel , B_kernel ,WF,BF,Wout,Bout) :
38
39
40 conv0 = conv2d(x, kernel, B_kernel, strides=1)
41 convlstm0, init0 = convrecurrent2d(channels=filter_channels ,num_input=num_inputs,
time_step=time_steps ,filter_channels=filter_channels ,
42 kernel_x=4,name='convlstm0' ,x=conv0 ,kind=kind ,batch_size=
batch_size)
43 if arq==1:
44 conv0 = conv2d(x, kernel, B_kernel, strides=1)
45 convlstm0, init0 = convrecurrent2d(channels=filter_channels , num_input=
num_inputs, time_step=time_steps ,
46 filter_channels=filter_channels ,
47 kernel_x=4, name='convlstm0' , x=conv0, kind
=kind ,batch_size=batch_size)
48 convlstm2, _ = convrecurrent2d(channels=filter_channels , num_input=num_inputs,
time_step=time_steps ,
49 filter_channels=filter_channels ,
50 kernel_x=4, name='convlstm2' , initial_state=
init0 , kind=kind ,batch_size=batch_size)
51 fc1 = tf.reshape(convlstm2, [batch_size, filter_channels * time_steps *
num_inputs])
52 elif arq==0:
53 conv0 = conv2d(x, kernel, B_kernel, strides=1)
54 convlstm0, init0 = convrecurrent2d(channels=filter_channels , num_input=
num_inputs, time_step=time_steps ,
55 filter_channels=filter_channels ,
56 kernel_x=4, name='convlstm0' , x=conv0, kind
=kind ,batch_size=batch_size)
57 fc1 = tf.reshape(convlstm0, [batch_size, filter_channels * time_steps *
num_inputs])
58
59 fc1 = tf.add( tf.matmul(fc1 , WF) , BF)
60 fc1 = tf.tanh(fc1 /3)
61 out1 = tf.add( tf.matmul(fc1 , Wout) , Bout)
62
63 return tf.abs(out1)
64
65 def Scoring(Y_true , Y_pred) :
66 h = Y_pred - Y_true
67 g = -(h-tf.abs(h))/2.0)
68 f = (( tf.abs(h)+h)/2.0)
69 return tf.reduce_sum( tf.exp(g/13.0)-tf.ones(h.shape))+tf.reduce_sum( tf.exp(f/10.0)
-tf.ones(h.shape))
70
71
72 def RMSE(Y_true , Y_pred) :
73 return tf.sqrt( tf.reduce_sum( tf.square(Y_pred -Y_true))/len(Y_true))

1 import tensorflow as tf
2 from get_data_CMAPSS import *
3 from model import *

```

```

4 import time
5 import os
6
7
8 class model:
9     def __init__(self, X_train, Y_train,
10                 X_val, Y_val,
11                 X_test, Y_test,
12                 X_test_b, Y_test_b,
13                 model_path,
14                 learning_rate, training_epoch,
15                 batch_size_train, batch_size_test, batch_size_test_b,
16                 display_step,
17                 num_inputs, time_steps,
18                 num_hidden, num_outputs,
19                 filter_channels, arq,
20                 kind, name):
21         self._X_train = X_train
22         self._Y_train = Y_train
23         self._X_val = X_val
24         self._Y_val = Y_val
25         self._X_test = X_test
26         self._Y_test = Y_test
27         self._X_test_b = X_test_b
28         self._Y_test_b = Y_test_b
29         self._model_path = model_path
30         self._learning_rate = learning_rate
31         self._training_epoch = training_epoch
32         self._batch_size_train = batch_size_train
33         self._batch_size_test = batch_size_test
34         self._batch_size_test_b = batch_size_test_b
35         self._display_step = display_step
36         self._num_inputs = num_inputs
37         self._time_steps = time_steps
38         self._num_hidden = num_hidden
39         self._num_outputs = num_outputs
40         self._filter_channels = filter_channels
41         self._arq = arq
42         self._kind = kind
43         self._name = name
44
45     def train(self):
46         tf.reset_default_graph()
47
48         kernel = tf.get_variable("kernel_c", shape=[15, 1, 1, self._filter_channels
49 ],
50                                 initializer=tf.contrib.layers.
51                                 xavier_initializer_conv2d())
52         B_kernel = tf.get_variable("B_kernel", shape=[self._filter_channels],
53                                     initializer=tf.contrib.layers.
54                                     xavier_initializer_conv2d())

```

```

53     WF = tf.get_variable('WF', shape=[self._filter_channels * self._time_steps *
self._num_inputs, self._num_hidden],
54                             initializer=tf.contrib.layers.xavier_initializer())
55     BF = tf.get_variable('BF', shape=[self._num_hidden],
56                             initializer=tf.contrib.layers.xavier_initializer())
57
58     Wout = tf.get_variable('Wout', shape=[self._num_hidden, self._num_outputs],
59                             initializer=tf.contrib.layers.xavier_initializer())
60     Bout = tf.get_variable('Bout', shape=[self._num_outputs],
61                             initializer=tf.contrib.layers.xavier_initializer())
62
63     X = tf.placeholder("float", [self._batch_size_train, self._time_steps, self.
_num_inputs, 1])
64     Y = tf.placeholder("float", [self._batch_size_train, self._num_outputs])
65
66     pred1 = ConvRecurrent(X, self._arq, self._filter_channels, self._kind,
67                             self._num_inputs, self._time_steps, self.
_batch_size_train ,
68                             kernel, B_kernel, WF,
69                             BF, Wout, Bout)
70     # score = Scoring(Y_true=Y_test, Y_pred=pred)
71     # rmse = RMSE(Y_true=Y_test, Y_pred=pred)
72
73     loss_op = tf.reduce_sum(tf.square(Y - pred1)) + tf.reduce_sum(tf.abs(WF)) +
\
74         100 * tf.reduce_sum(tf.square(tf.nn.relu(pred1 - 125)))
75
76     optimizer = tf.train.AdamOptimizer(self._learning_rate)
77     # optimize for training
78     train_op = optimizer.minimize(loss_op)
79
80     # evaluate model
81     accuracy = tf.sqrt(tf.reduce_mean(tf.square(Y - pred1)))
82
83     # initialize variables
84     init = tf.global_variables_initializer()
85     # 'Saver' op to save and restore all the variables
86     saver = tf.train.Saver()
87     print("training...")
88
89     with tf.Session() as sess:
90         sess.run(init)
91         # saver.restore(sess, model_path)
92         start = time.time()
93
94         for step in range(1, self._training_epoch + 1):
95
96             batch_x, batch_y = Next_Batch3(self._X_train, self._Y_train, self.
_batch_size_train)
97             batch_x = batch_x.reshape((self._batch_size_train, self._time_steps,
self._num_inputs, 1))
98             batch_x_val, batch_y_val = Next_Batch3(self._X_val, self._Y_val,

```

```

self._batch_size_train)
99         batch_x_val = batch_x_val.reshape((self._batch_size_train, self.
        _time_steps, self._num_inputs, 1))
100
101         # pred = sess.run(accuracy, feed_dict={X: batch_x_val, Y:
batch_y_val, keep_prob: 1.0})
102         # Run optimization op (backprop)
103         sess.run(train_op, feed_dict={X: batch_x, Y: batch_y})
104
105         if step % self._display_step == 0 or step == 1:
106             # Calculate batch loss and accuracy
107             _, acc_train = sess.run([loss_op, accuracy], feed_dict={X:
batch_x, Y: batch_y})
108             print("\n" + "Step " + str(step))
109             #print("Training: Loss = " + \
110                   #       "{:.4 f}".format(loss_train) + ", Accuracy= " + \
111                   #       "{:.3 f}".format(acc_train))
112             _, acc_val = sess.run([loss_op, accuracy],
113                                   feed_dict={X: batch_x_val, Y:
batch_y_val})
114             #print("\n" + "Step " + str(step))
115             #print("val: Loss = " + \
116                   #       "{:.4 f}".format(loss_val) + ", Accuracy= " + \
117                   #       "{:.3 f}".format(acc_val))
118
119             if self._arq==0:
120                 acc_file1 = self._model_path + "Conv" + str(self._kind)+"/
RMSE_hist/ACC_train.txt/"
121                 acc_file2 = self._model_path + "Conv" + str(self._kind) + "/"
RMSE_hist/ACC_val.txt/"
122
123             else:
124                 acc_file1 = self._model_path + "Conv" + str(self._kind) + "/"
RMSE_hist/ACC_train.txt/"
125                 acc_file2 = self._model_path + "Conv" + str(self._kind)+"/ED
/RMSE_hist/ACC_val.txt/"
126
127                 file1 = open(acc_file1, "a+")
128                 file2 = open(acc_file2, "a+")
129                 file1.write(" {:.3 f}".format(acc_train) + "\n")
130                 file2.write(" {:.4 f}".format(acc_val) + "\n")
131                 file1.close()
132                 file2.close()
133             if step == self._training_epoch:
134                 pred = sess.run(pred1, feed_dict={X: batch_x, Y: batch_y})
135                 if self._arq == 0:
136                     predicciones = self._model_path + "Conv" + str(self._kind) +
"/ajuste.txt/"
137
138                 else:
139                     predicciones = self._model_path + "Conv" + str(self._kind) +
"/ajuste.txt/"

```



```

140
141         file = open(predicciones, "a+")
142         file.write(pred)
143         file.close()
144
145
146
147
148         print("Optimization Finished!")
149
150         stop = time.time()
151         t = stop - start
152         print('\n Tiempo total de entrenamiento = %g [s] \n' % t)
153         # Save model weights to disk
154
155         save_path = saver.save(sess, self._model_path)
156         print("Model saved in file: %s" % save_path)
157
158     def test(self):
159         tf.reset_default_graph()
160
161         kernel = tf.get_variable("kernel_c", shape=[15, 1, 1, self._filter_channels
162 ],
163                                 initializer=tf.contrib.layers.
164 xavier_initializer_conv2d())
165         B_kernel = tf.get_variable("B_kernel", shape=[self._filter_channels],
166                                 initializer=tf.contrib.layers.
167 xavier_initializer_conv2d())
168
169         WF = tf.get_variable('WF', shape=[self._filter_channels * self._time_steps *
170 self._num_inputs, self._num_hidden],
171                                 initializer=tf.contrib.layers.xavier_initializer())
172         BF = tf.get_variable('BF', shape=[self._num_hidden],
173                                 initializer=tf.contrib.layers.xavier_initializer())
174
175         Wout = tf.get_variable('Wout', shape=[self._num_hidden, self._num_outputs],
176                                 initializer=tf.contrib.layers.xavier_initializer())
177         Bout = tf.get_variable('Bout', shape=[self._num_outputs],
178                                 initializer=tf.contrib.layers.xavier_initializer())
179
180         X = tf.placeholder("float", [self._batch_size_test, self._time_steps, self.
181 _num_inputs, 1])
182         Y = tf.placeholder("float", [self._batch_size_test, self._num_outputs])
183
184         pred1 = ConvRecurrent(X, self._arq, self._filter_channels, self._kind,
185 self._num_inputs, self._time_steps, self.
186 _batch_size_test,
187                             kernel, B_kernel, WF,
188                             BF, Wout, Bout)
189
190         # rmse = RMSE(Y_true=Y_test, Y_pred=pred)

```

```

186     loss_op = tf.reduce_sum(tf.square(Y - pred1)) + tf.reduce_sum(tf.abs(WF)) +
187     \
188         100 * tf.reduce_sum(tf.square(tf.nn.relu(pred1 - 125)))
189
190     optimizer = tf.train.AdamOptimizer(self._learning_rate)
191     # optimize for training
192     train_op = optimizer.minimize(loss_op)
193
194     # evaluate model
195     accuracy = tf.sqrt(tf.reduce_mean(tf.square(Y - pred1)))
196     score = Scoring(Y_true=Y, Y_pred=pred1)
197
198     # initialize variables
199     init = tf.global_variables_initializer()
200     # 'Saver' op to save and restore all the variables
201     saver = tf.train.Saver()
202     print("testing all data ...")
203
204     with tf.Session() as sess:
205         sess.run(init)
206         saver.restore(sess, self._model_path)
207         self._X_test = self._X_test.reshape((self._batch_size_test, self.
208         _time_steps, self._num_inputs, 1))
209         #start = time.time()
210         #pred = sess.run(pred1, feed_dict={X: self._X_test, Y: self._Y_test})
211         #print("Testing all data Accuracy:", \
212         #      sess.run(accuracy, feed_dict={X: self._X_test, Y: self._Y_test}))
213         #stop = time.time()
214         #t = stop - start
215         #print('\n Tiempo total de testing = %g [s] \n' % t)
216         #print("Testing all data Score:", \
217         #      sess.run(score, feed_dict={X: self._X_test, Y: self._Y_test}))
218
219
220     def test_b(self):
221         tf.reset_default_graph()
222
223         kernel = tf.get_variable("kernel_c", shape=[15, 1, 1, self._filter_channels
224         ],
225                                 initializer=tf.contrib.layers.
226                                 xavier_initializer_conv2d())
227         B_kernel = tf.get_variable("B_kernel", shape=[self._filter_channels],
228                                 initializer=tf.contrib.layers.
229                                 xavier_initializer_conv2d())
230
231         WF = tf.get_variable('WF', shape=[self._filter_channels * self._time_steps *
232         self._num_inputs, self._num_hidden],
233                               initializer=tf.contrib.layers.xavier_initializer())
234         BF = tf.get_variable('BF', shape=[self._num_hidden],
235                               initializer=tf.contrib.layers.xavier_initializer())

```

```

232     Wout = tf.get_variable('Wout', shape=[self._num_hidden, self._num_outputs],
233                           initializer=tf.contrib.layers.xavier_initializer())
234     Bout = tf.get_variable('Bout', shape=[self._num_outputs],
235                            initializer=tf.contrib.layers.xavier_initializer())
236
237
238     X = tf.placeholder("float", [self._batch_size_test_b, self._time_steps, self
239     ._num_inputs, 1])
240     Y = tf.placeholder("float", [self._batch_size_test_b, self._num_outputs])
241
242     pred1 = ConvRecurrent(X, self._arq, self._filter_channels, self._kind,
243                          self._num_inputs, self._time_steps, self.
244                          _batch_size_test_b,
245                          kernel, B_kernel, WF,
246                          BF, Wout, Bout)
247     # score = Scoring(Y_true=Y_test, Y_pred=pred)
248     # rmse = RMSE(Y_true=Y_test, Y_pred=pred)
249
250     loss_op = tf.reduce_sum(tf.square(Y - pred1)) + tf.reduce_sum(tf.abs(WF)) +
251     \
252     100 * tf.reduce_sum(tf.square(tf.nn.relu(pred1 - 125)))
253
254     optimizer = tf.train.AdamOptimizer(self._learning_rate)
255     # optimize for training
256     train_op = optimizer.minimize(loss_op)
257
258     # evaluate model
259     accuracy = tf.sqrt(tf.reduce_mean(tf.square(Y - pred1)))
260     score = Scoring(Y_true=Y, Y_pred=pred1)
261
262     # initialize variables
263     init = tf.global_variables_initializer()
264     # 'Saver' op to save and restore all the variables
265     saver = tf.train.Saver()
266     print("testing...")
267
268     with tf.Session() as sess:
269         sess.run(init)
270         saver.restore(sess, self._model_path)
271         self._X_test_b = self._X_test_b.reshape((self._batch_size_test_b, self.
272         _time_steps, self._num_inputs, 1))
273         #start = time.time()
274         #pred = sess.run(pred1, feed_dict={X: self._X_test, Y: self._Y_test})
275         #print("Testing Accuracy:", \
276         #      sess.run(accuracy, feed_dict={X: self._X_test_b, Y: self.
277         _Y_test_b}))
278         #stop = time.time()
279         #t = stop - start
280         #print('\n Tiempo total de testing = %g [s] \n' % t)
281         #print("Testing Score:", \
282         #      sess.run(score, feed_dict={X: self._X_test_b, Y: self._Y_test_b})
283         )

```

```

278     pred = sess.run(pred1, feed_dict={X: self._X_test_b, Y: self._Y_test_b})
279     if self._arq == 0:
280         predicciones = self._model_path + "Conv" + str(self._kind) + "/"
predicciones.txt/"
281         score = self._model_path + "Conv" + str(self._kind) + "/score.txt/"
282         rmse = self._model_path + "Conv" + str(self._kind) + "/RMSE.txt/"
283
284
285     else:
286         predicciones = self._model_path + "Conv" + str(self._kind) + "/"
predicciones.txt/"
287         score = self._model_path + "Conv" + str(self._kind) + "/score.txt/"
288         rmse = self._model_path + "Conv" + str(self._kind) + "/RMSE.txt/"
289
290         file = open(predicciones, "W")
291         file1 = open(score, "a+")
292         file2 = open(rmse, "a+")
293         file1.write(sess.run(score, feed_dict={X: self._X_test_b, Y: self.
_Y_test_b}) + "\n")
294         file2.write(sess.run(accuracy, feed_dict={X: self._X_test_b, Y: self.
_Y_test_b}) + "\n")
295         file.write(sess.run(pred, feed_dict={X: self._X_test_b, Y: self.
_Y_test_b}) + "\n")
296         file.close()
297         file1.close()
298         file2.close()
299
300     def val(self):
301         tf.reset_default_graph()
302
303         kernel = tf.get_variable("kernel_c", shape=[15, 1, 1, self._filter_channels
],
304                                 initializer=tf.contrib.layers.
xavier_initializer_conv2d())
305         B_kernel = tf.get_variable("B_kernel", shape=[self._filter_channels],
306                                 initializer=tf.contrib.layers.
xavier_initializer_conv2d())
307
308         WF = tf.get_variable('WF', shape=[self._filter_channels * self._time_steps *
self._num_inputs, self._num_hidden],
309                             initializer=tf.contrib.layers.xavier_initializer())
310         BF = tf.get_variable('BF', shape=[self._num_hidden],
311                             initializer=tf.contrib.layers.xavier_initializer())
312
313         Wout = tf.get_variable('Wout', shape=[self._num_hidden, self._num_outputs],
314                                 initializer=tf.contrib.layers.xavier_initializer())
315         Bout = tf.get_variable('Bout', shape=[self._num_outputs],
316                                 initializer=tf.contrib.layers.xavier_initializer())
317
318         X = tf.placeholder("float", [2048, self._time_steps, self._num_inputs, 1])
319         Y = tf.placeholder("float", [2048, self._num_outputs])#2048 batch size val
320

```

```

321     pred1 = ConvRecurrent(X, self._arq, self._filter_channels, self._kind,
322                          self._num_inputs, self._time_steps, 2048,
323                          kernel, B_kernel, WF,
324                          BF, Wout, Bout)
325     # score = Scoring(Y_true=Y_test, Y_pred=pred)
326     # rmse = RMSE(Y_true=Y_test, Y_pred=pred)
327
328     loss_op = tf.reduce_sum(tf.square(Y - pred1)) + tf.reduce_sum(tf.abs(WF)) +
    \
329         100 * tf.reduce_sum(tf.square(tf.nn.relu(pred1 - 125)))
330
331     optimizer = tf.train.AdamOptimizer(self._learning_rate)
332     # optimize for training
333     train_op = optimizer.minimize(loss_op)
334
335     # evaluate model
336     accuracy = tf.sqrt(tf.reduce_mean(tf.square(Y - pred1)))
337
338     # initialize variables
339     init = tf.global_variables_initializer()
340     # 'Saver' op to save and restore all the variables
341     saver = tf.train.Saver()
342     print("testing...")
343
344     with tf.Session() as sess:
345         sess.run(init)
346         saver.restore(sess, self._model_path)
347         batch_x_val, batch_y_val = Next_Batch3(self._X_val, self._Y_val, 2048)
348         batch_x_val = batch_x_val.reshape((2048, self._time_steps, self.
    _num_inputs, 1))
349         start = time.time()
350         #pred = sess.run(pred1, feed_dict={X: self._X_test, Y: self._Y_test})
351         print("Testing Accuracy:", \
352             sess.run(accuracy, feed_dict={X: batch_x_val, Y: batch_y_val}))
353         stop = time.time()
354         t = stop - start
355         print('\n Tiempo total de testing val= %g [s] \n' % t)

```

```

1 import numpy as np
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
4 from sklearn.preprocessing import MinMaxScaler
5
6
7
8 def TrainImages(n_set, n_flights, filter_t, f_modes):
9     # Data should be a pandas dataframe of the original raw data, e.g. FD001.txt
10    # RUL is a pandas col with the corresponding RUL value of every time step
11    # n_flights is the number of flights found for this dataset e.g. FD001 has 100
    flights
12    images = []
13    rul = []

```

```

14 set = '/home/nicolas/Desktop/C-MAPSS/train_FD00'+str(n_set)+'.txt'
15 data = pd.read_csv(set, delim_whitespace=True, header=None)
16 vuelos = np.linspace(1, n_flights, n_flights)
17 data.columns = ['Unit Number', 'Cycles', 'Operational Setting 1', 'Operational
18 Setting 2', 'Operational Setting 3',
19 'Sensor Measurement 1', 'Sensor Measurement 2', 'Sensor
20 Measurement 3', 'Sensor Measurement 4',
21 'Sensor Measurement 5', 'Sensor Measurement 6', 'Sensor
22 Measurement 7', 'Sensor Measurement 8',
23 'Sensor Measurement 9', 'Sensor Measurement 10', 'Sensor
24 Measurement 11',
25 'Sensor Measurement 12', 'Sensor Measurement 13', 'Sensor
26 Measurement 14',
27 'Sensor Measurement 15', 'Sensor Measurement 16', 'Sensor
28 Measurement 17',
29 'Sensor Measurement 18', 'Sensor Measurement 19', 'Sensor
30 Measurement 20',
31 'Sensor Measurement 21']
32 data['RUL'] = pd.Series(np.zeros(len(data)), index=data.index)
33 for vuelo in vuelos:
34     a = data[data['Unit Number'] == vuelo]
35     tiempo = len(a)
36     RUL = np.linspace(tiempo, 0, tiempo)
37     for n in range(0, len(RUL)):
38         if RUL[n] > 125:#RUL edition for 99.99% reliability
39             RUL[n] = 125
40     data.loc[a.index, 'RUL'] = pd.Series(RUL, index=a.index)
41
42 for i in range(1, n_flights + 1): # Iterate over each flight in the dataset
43     flight = data[data['Unit Number'] == i] # Take all data entries for the
44     current flight
45     RUL = flight['RUL'].to_frame() # Extract the RUL labels contained for this
46     if f_modes:
47         X = flight.drop(
48             ['Unit Number', # 'Operational Setting 1', 'Operational Setting 2', '
49             Operational Setting 3',
50             'Sensor Measurement 1', 'Sensor Measurement 5', 'Sensor Measurement
51             6', 'Sensor Measurement 10',
52             'Sensor Measurement 16', 'Sensor Measurement 18', 'Sensor
53             Measurement 19', 'RUL'], axis=1)
54     else:
55         X = flight.drop(
56             ['Unit Number', 'Operational Setting 1', 'Operational Setting 2', '
57             Operational Setting 3',
58             'Sensor Measurement 1', 'Sensor Measurement 5', 'Sensor Measurement
59             6', 'Sensor Measurement 10',
60             'Sensor Measurement 16', 'Sensor Measurement 18', 'Sensor
61             Measurement 19', 'RUL'], axis=1)
62
63     length = len(X)
64     if filter_t > length:
65         continue

```

```

52
53     n_images = length - filter_t + 1 # Num of possible images for each flight
54     X = np.array(X)
55     RUL = np.array(RUL)
56     for j in range(n_images):
57         image = X[j:j + filter_t, :]
58         images.append(image)
59         rul.append(RUL[j + filter_t - 1])
60
61 images = np.asarray(images)
62 images = np.reshape(images, (images.shape[0], images.shape[1] * images.shape[2]))
63 rul = np.asarray(rul)
64
65
66 return images, rul
67
68 def TestImages(n_set, n_flights, filter_t, f_modes):
69     # Obtain only the last image of each flight in the dataset
70     # In this case, we do not extract the RUL, since is given in a separate file
71     images = []
72     set_images = '/home/nicolas/Desktop/C-MAPSS/test_FD00'+str(n_set)+'.txt'
73     set_rul = '/home/nicolas/Desktop/C-MAPSS/RUL_FD00'+str(n_set)+'.txt'
74     Test_data = pd.read_csv(set_images, delim_whitespace=True, header=None)
75     Test_data.columns = ['Unit Number', 'Cycles', 'Operational Setting 1', '
76         Operational Setting 2',
77         'Operational Setting 3',
78         'Sensor Measurement 1', 'Sensor Measurement 2', 'Sensor
79         Measurement 3',
80         'Sensor Measurement 4',
81         'Sensor Measurement 5', 'Sensor Measurement 6', 'Sensor
82         Measurement 7',
83         'Sensor Measurement 8',
84         'Sensor Measurement 9', 'Sensor Measurement 10', 'Sensor
85         Measurement 11',
86         'Sensor Measurement 12',
87         'Sensor Measurement 13', 'Sensor Measurement 14', 'Sensor
88         Measurement 15',
89         'Sensor Measurement 16',
90         'Sensor Measurement 17', 'Sensor Measurement 18', 'Sensor
91         Measurement 19',
92         'Sensor Measurement 20',
93         'Sensor Measurement 21']
94
95     for i in range(1, n_flights + 1):
96         flight = Test_data[Test_data['Unit Number'] == i]
97         if f_modes:
98             X = flight.drop(
99                 ['Unit Number', # 'Operational Setting 1', 'Operational Setting 2', '
100                 Operational Setting 3',
101                 'Sensor Measurement 1', 'Sensor Measurement 5', 'Sensor Measurement
102                 6', 'Sensor Measurement 10',
103                 'Sensor Measurement 16', 'Sensor Measurement 18', 'Sensor

```

```

Measurement 19'], axis=1)
96     else:
97         X = flight.drop(
98             ['Unit Number', 'Operational Setting 1', 'Operational Setting 2', '
Operational Setting 3',
99             'Sensor Measurement 1', 'Sensor Measurement 5', 'Sensor Measurement
6', 'Sensor Measurement 10',
100             'Sensor Measurement 16', 'Sensor Measurement 18', 'Sensor
Measurement 19'], axis=1)
101
102         length = len(X)
103         #if filter_t > length:
104         #    continue
105
106         X = np.array(X)
107         image = X[(length - filter_t):length, :]
108         images.append(image)
109
110     images = np.asarray(images)
111     images = np.reshape(images, (images.shape[0], images.shape[1] * images.shape[2]))
112     rul = np.array(pd.read_csv(set_rul, delim_whitespace=True, header=None)) #RUL labels
    for the test set
113
114     if n_flights == 100:
115         n_units=100
116     elif n_flights == 248:
117         n_units = 248
118     else:
119         n_units = 259
120     for i in range(n_units):
121         if rul[i] > 125:
122             rul[i] = 125
123
124     return images, rul
125
126
127 def get_data(window, f_modes=True, data1=False):
128     if f_modes:
129         X1, Y1 = TrainImages(1, 100, window, f_modes=f_modes)
130         X_test1, Y_test1 = TestImages(1, 100, window, f_modes=f_modes)
131         X2, Y2 = TrainImages(2, 259, window, f_modes=f_modes)
132         X_test2, Y_test2 = TestImages(2, 259, window, f_modes=f_modes)
133
134         X1, X_val1, Y1, Y_val1 = train_test_split(X1, Y1, test_size=0.15,
random_state=48)
135         X2, X_val2, Y2, Y_val2 = train_test_split(X2, Y2, test_size=0.15,
random_state=48)
136
137         X = np.concatenate([X1, X2], 0)
138         Y = np.concatenate([Y1, Y2], 0)
139         X_val = np.concatenate([X_val1, X_val2], 0)
140         Y_val = np.concatenate([Y_val1, Y_val2], 0)

```



```

141 X_test = np.concatenate([X_test1, X_test2], 0)
142 Y_test = np.concatenate([Y_test1, Y_test2], 0)
143
144 X3, Y3 = TrainImages(3, 100, window, f_modes=f_modes)
145 X_test3, Y_test3 = TestImages(3, 100, window, f_modes=f_modes)
146
147 X3, X_val3, Y3, Y_val3 = train_test_split(X3, Y3, test_size=0.15,
148 random_state=48)
149
150 X = np.concatenate([X, X3], 0)
151 Y = np.concatenate([Y, Y3], 0)
152 X_val = np.concatenate([X_val, X_val3], 0)
153 Y_val = np.concatenate([Y_val, Y_val3], 0)
154 X_test = np.concatenate([X_test, X_test3], 0)
155 Y_test = np.concatenate([Y_test, Y_test3], 0)
156
157 X4, Y4 = TrainImages(4, 248, window, f_modes=f_modes)
158 X_test4, Y_test4 = TestImages(4, 248, window, f_modes=f_modes)
159
160 X4, X_val4, Y4, Y_val4 = train_test_split(X4, Y4, test_size=0.15,
161 random_state=48)
162
163 X_train = np.concatenate([X, X4], 0)
164 Y_train = np.concatenate([Y, Y4], 0)
165 X_val = np.concatenate([X_val, X_val4], 0)
166 Y_val = np.concatenate([Y_val, Y_val4], 0)
167 X_test = np.concatenate([X_test, X_test4], 0)
168 Y_test = np.concatenate([Y_test, Y_test4], 0)
169
170 sc_X = MinMaxScaler(feature_range=(-1, 1))
171 X_train = sc_X.fit_transform(X_train) # Only transform the training set
172 X_val = sc_X.transform(X_val)
173 X_test = sc_X.transform(X_test)
174 X_test2 = sc_X.transform(X_test2)
175 X_test4 = sc_X.transform(X_test4)
176
177 return X_train, Y_train, X_val, Y_val, X_test, Y_test, X_test2, Y_test2,
178 X_test4, Y_test4
179
180 elif f_modes==False and datal==True:
181 X1, Y1 = TrainImages(1, 100, window, f_modes=f_modes)
182 X_test1, Y_test1 = TestImages(1, 100, window, f_modes=f_modes)
183
184 X_train1, X_val1, Y_train1, Y_val1 = train_test_split(X1, Y1, test_size
185 =0.15, random_state=48)
186
187 sc_X = MinMaxScaler(feature_range=(-1, 1))
188 X_train1 = sc_X.fit_transform(X_train1) # Only transform the training set
189 X_val1 = sc_X.transform(X_val1)
190 X_test1 = sc_X.transform(X_test1)
191
192 return X_train1, Y_train1, X_val1, Y_val1, X_test1, Y_test1

```

```

189
190
191     else:
192         X1, Y1 = TrainImages(1, 100, window,f_modes=f_modes)
193         X_test1, Y_test1 = TestImages(1, 100, window,f_modes=f_modes)
194         X3, Y3 = TrainImages(3, 100, window,f_modes=f_modes)
195         X_test3, Y_test3 = TestImages(3, 100, window,f_modes=f_modes)
196
197         X1, X_val1, Y1, Y_val1 = train_test_split(X1, Y1, test_size=0.15,
198 random_state=48)
199         X3, X_val3, Y3, Y_val3 = train_test_split(X3, Y3, test_size=0.15,
200 random_state=48)
201
202         X_train = np.concatenate([X1, X3], 0)
203         Y_train = np.concatenate([Y1, Y3], 0)
204         X_val = np.concatenate([X_val1, X_val3], 0)
205         Y_val = np.concatenate([Y_val1, Y_val3], 0)
206         X_test = np.concatenate([X_test1, X_test3], 0)
207         Y_test = np.concatenate([Y_test1, Y_test3], 0)
208
209         sc_X = MinMaxScaler(feature_range=(-1, 1))
210         X_train = sc_X.fit_transform(X_train) # Only transform the training set
211         X_val = sc_X.transform(X_val)
212         X_test = sc_X.transform(X_test)
213         X_test1 = sc_X.transform(X_test1)
214         X_test3 = sc_X.transform(X_test3)
215
216         return X_train, Y_train, X_val, Y_val, X_test, Y_test, X_test1, Y_test1,
217 X_test3, Y_test3
218
219 def Next_Batch3(X,y, batch_size):
220     batch_x=[]
221     batch_y=[]
222     for j in range(1, batch_size + 1):
223         i = np.random.randint(1, len(X))
224         image = X[i, :]
225         rul = y[i]
226         batch_x.append(image)
227         batch_y.append(rul)
228     batch_x = np.asarray(batch_x)
229     batch_y = np.asarray(batch_y)
230     return batch_x, batch_y
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894

```

```

8 model_path = "/home/nicolas/Desktop/C-MAPSS/PRUEBA_FINAL/"
9 # Training Parameters
10 learning_rate = 0.001
11 #training_epoch_pretrain = 2
12 training_epoch = 25 #2200 optimo hasta ahora
13 batch_size_train = 10
14 display_step = 10
15
16
17
18 # Network Parameters
19 num_inputs = 18 # MNIST data input (img shape: 28*28)
20 time_steps = 19 # timesteps
21 num_hidden = 100# num hidden units
22 num_outputs = 1 # Only one output
23 filter_channels = 10
24 arq = 0
25 kind = "JANET"
26 name = "Model"
27
28 modelos = {"JANET":[0,1],
29           "LSTM":[0,1]}
30
31
32
33 for j in range(3):
34
35     modelx = model(X_train , Y_train ,
36                  X_val , Y_val ,
37                  X_test , Y_test ,
38                  X_test_b , Y_test_b ,
39                  model_path ,
40                  learning_rate , training_epoch ,
41                  batch_size_train , X_test.shape[0] ,259 ,
42                  display_step ,
43                  num_inputs , time_steps ,
44                  num_hidden , num_outputs ,
45                  filter_channels , arq ,
46                  kind , name)
47     for i in range(2):
48         print ("intento" ,j , i)
49         modelx.train ()
50         modelx.test ()
51         modelx.test_b ()

```