UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDO DIONE DOS SANTOS LIMA

USING DEEP NEURAL NETWORKS FOR FAILURE PREDICTION IN HARD DISK DRIVES

FORTALEZA

2018

FERNANDO DIONE DOS SANTOS LIMA

USING DEEP NEURAL NETWORKS FOR FAILURE PREDICTION IN HARD DISK DRIVES

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Inteligência Artificial

Orientador: Prof. Dr. João Paulo Pordeus Gomes

Co-Orientador: Prof. Dr. Javam de Castro Machado

FORTALEZA

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

L698u Lima, Fernando Dione dos Santos.
Using deep neural networks for failure prediction in hard disk drives / Fernando Dione dos Santos Lima. – 2018.
78 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2018.
Orientação: Prof. Dr. João Paulo Pordeus Gomes.
Coorientação: Prof. Dr. Javam de Castro Machado.

1. HDD failure prediction. 2. Deep learning. 3. RUL estimation. I. Título.

CDD 005

FERNANDO DIONE DOS SANTOS LIMA

USING DEEP NEURAL NETWORKS FOR FAILURE PREDICTION IN HARD DISK DRIVES

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Inteligência Artificial

Aprovada em: 29/11/2018.

BANCA EXAMINADORA

Prof. Dr. João Paulo Pordeus Gomes (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Javam de Castro Machado (Co-Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. José Maria Monteiro
Universidade Federal do Ceará (UFC)

Prof. Dr. Guilherme de Alencar Barreto
Universidade Federal do Ceará (UFC)

I dedicate this work to my family. My mother, Tânia, who had left us before I joined the master's program, for supporting and believing in me throughout life. My father, Francisco, and my brothers, Ozéias and Camylla, for encouraging me along this journey.

ACKNOWLEDGEMENTS

Foremost, I would like to express my gratitude to my advisor, Prof. João Paulo Pordeus Gomes, for the continuous support of my master's research, for his motivation, enthusiasm, and knowledge sharing.

Besides my advisor, I would like to thank my co-advisor, Prof. Javam de Castro Machado, coordinator of Laboratório de Sistemas e Banco de Dados (LSBD), for his encouragement prior to my enrollment in the master's degree and also for the support throughout the development of this dissertation. Also, Prof. Roselia de Castro Machado has played a major role motivating me and ensuring that LSBD is a great place to do research and work.

In addition to that, I thank the dissertation committee, Prof. José Maria Monteiro and Prof. Guilherme Barreto, for the insightful comments, hard questions and suggestions to improve this text.

To Francisco Lucas Falcão Pereira, for the *deep* contributions and work that made this dissertation possible.

To my once girlfriend Raíssa Aquino Schatzmann, I would like to thank for all the moments you took me out from home to have a good time and relief from all the stress that writing a dissertation involves.

To all the friends whom I have had the pleasure to share moments at LSBD. You have made this whole process more enjoyable. Particularly I would like to thank Ubiratan Soares Netto, Elsa Martins, Denis Moraes Cavalcante, Davi Braga, Elvis Teixeira, Paulo Amora and Manoel Rui.

To the Universidade Federal do Ceará for showing that a university can be public, free and produce research of excellence.

To Lenovo for the funding this research.

“A pessoa que se vende recebe sempre mais do que vale.”

(Barão de Itararé)

RESUMO

Discos rígidos (HDDs) ainda são a tecnologia de armazenamento mais amplamente utilizada em sistemas de armazenamento de larga escala. Isso se deve principalmente à sua relação custo-benefício em termos de custo por volume de armazenamento. Várias pesquisas foram feitas para propor técnicas de detecção antecipada de falhas para estes dispositivos, visando aumentar a disponibilidade dos sistemas de armazenamento e evitar a perda de dados. A previsão de falhas em tais circunstâncias permitiria a redução dos custos decorrentes do tempo de indisponibilidade desses sistemas por meio de substituições antecipadas dos dispositivos, além da migração de dados para estes novos dispositivos, evitando perdas de dados. Muitas das técnicas propostas até agora realizam principalmente a detecção incipiente de falhas, não permitindo o planejamento adequado de tais tarefas de manutenção. Neste trabalho, apresentamos várias abordagens de estimativa de vida útil remanescente (RUL) para discos rígidos baseados em parâmetros SMART. Tais abordagens incluem duas diferentes modelagens para o problema. A primeira modelagem é a mais tradicional, baseada em regressão e que propicia uma fina granularidade na predição. A outra abordagem possibilita um maior controle sobre a granularidade necessária pelo operador, através de configuração prévia, e consiste na modelagem do problema como uma tarefa de classificação multiclasse, ou multinomial. No contexto do problema de classificação, exploramos também dois aspectos importantes para o problema de estimativa de RUL: a ordinalidade entre classes, e o viés preditivo para classes que indicam um tempo de vida reduzido, quando da ocorrência de classificações errôneas. Isso se dá através da aplicação de uma codificação ajustável para as classes. Todos os modelos propostos são baseados em redes neurais profundas (DNNs). Na avaliação dos modelos, um conjunto de dados proveniente de uma companhia de armazenamento de dados em nuvem, e contendo amostras de 1,697 dispositivos que falharam, foi utilizado. Experimentos mostraram que as abordagens propostas propiciam resultado satisfatórios na modelagem como problema de regressão, onde foi realizada uma análise aplicando métricas desenhadas para tarefas de prognóstico. Na modelagem como problema de classificação tradicional nosso modelo obteve resultados superiores ao modelo concorrente e, no problema de classificação assimétrica, melhora nas métricas que abordam a ordinalidade juntamente com a assimetria.

Palavras-chave: Predição de falha em HDDs. Aprendizagem Profunda. Estimativa de RUL.

ABSTRACT

Hard disk drives (HDDs) are still the most widely used storage technology employed in large-scale storage systems. This is mainly a result of its excellent cost-benefit relation in terms of cost per gigabyte. Several research efforts have been done to propose early failure detection techniques for these devices in order to improve storage systems availability and avoid data loss. Failure prediction in such circumstances would allow for the reduction of downtime costs through anticipated disk replacements, as well as the migration of data to new devices, avoiding data loss. Many of the techniques proposed so far mainly perform incipient failure detection thus not allowing for proper planning of such maintenance tasks. In this work, we present several remaining useful life (RUL) estimation approaches for hard disk drives based on SMART parameters. These approaches include two different modelings of the problem. The first is a traditional regression-based that allows for fine-grained predictions. The other approach allows for a greater control over the granularity needed by the systems operator, through a previous configuration, and consists in the modeling of the problem as a multiclass, or multinomial, classification task. In the context of the classification problem, we also explore two important aspects of the RUL estimation task: the ordinality between classes, and the predictive bias towards classes that indicate a reduced device lifetime, when an incorrect prediction takes place. All models are based on Deep Neural Networks (DNNs). For evaluating the models, a dataset produced by a cloud storage service provider, and including the complete time-series for 1,697 failing devices, was employed. Experiments showed that the proposed methods produced satisfying results in the regression-based task when assessed with metrics well-suited and designed specifically for prognostics tasks. In the modeling as a traditional classification task, our model produced superior results to the baseline model and, in the asymmetric and ordinal classification task, it outperformed baselines in metrics where both ordinality and asymmetry were taken into account.

Keywords: HDD failure prediction. Deep learning. RUL estimation.

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\mathscr{L}$	Loss function
σ	Sigmoid function
θ	Hyperbolic tangent function
$\odot$	Pointwise multiplication
$*$	Convolution operator

CONTENTS

1 INTRODUCTION

Nowadays, the reliability of data centers is a significant concern for most storage service providers and end users. Failures on storage systems may not only result in temporary data unavailability but also lead to permanent data loss (SCHROEDER; GIBSON, 2007). As can be seen in figure 1, the unavailability, or downtime of data centers, resulted in an increasing cost to these service providers. More precisely, an increase of 38% from 2010 to 2016 was noticed.

Among all electronic devices, Hard Disk Drives (HDDs) exhibit one of the highest failure rates, thus affecting the whole reliability of electronic systems (Ponemon Institute LLC, 2016). Moreover, HDDs are still the most widely used technology on large-scale storage systems given its cost/benefit relation (PINHEIRO *et al.*, 2007) (YE *et al.*, 2013). Such fact makes the capability of predicting failures of HDDs, highly desirable. As a result, several works like (LIMA *et al.*, 2017; LI *et al.*, 2014; QUEIROZ *et al.*, 2016; XU *et al.*, 2016; RINCóN *et al.*, 2017), addressed the problem of HDDs health monitoring in recent years.

Today, most HDDs are equipped with an embedded failure detection system named Self-Monitoring, Analysis and Reporting Technology (SMART). The SMART system collects a set of failure related measurements and monitor its progress over time. A failure is detected if any of such measurements exceeds a given threshold. Although SMART is a widely used method to detect failures, so far, no commercial system with failure prediction capability was

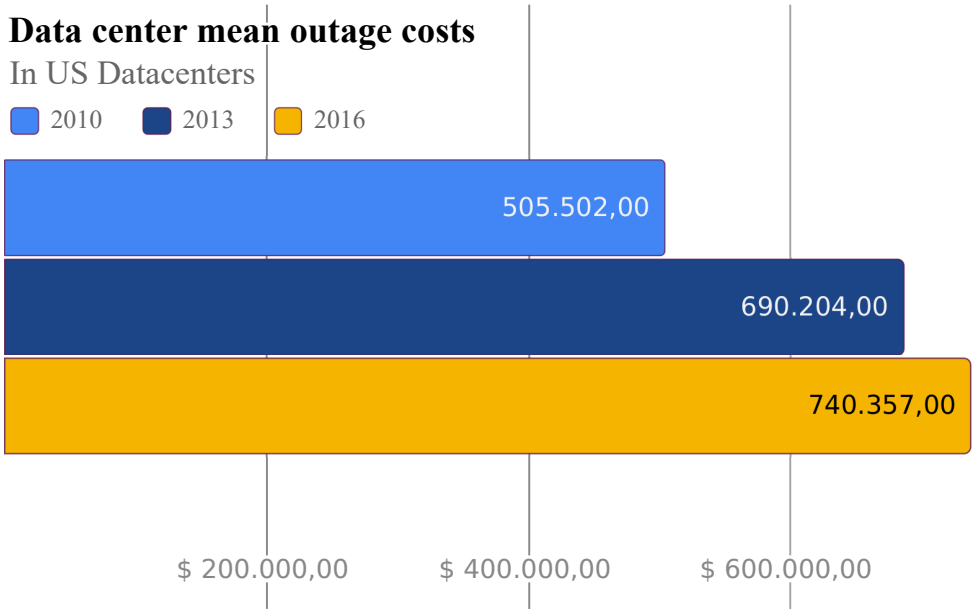

Figure 1 – Mean outage costs of data centers in the USA. Adapted from (Ponemon Institute LLC, 2016)

developed.

Works like (QUEIROZ *et al.*, 2016; WANG *et al.*, 2013; WANG *et al.*, 2014) proposed methods to solve the incipient failure prediction task. In this work, the primary objective was to detect abnormal behavior of HDDs that may indicate incipient failures. Such a problem can be formulated as a single or binary classification task. In other works like (LIMA *et al.*, 2017; CHAVES *et al.*, 2016; XU *et al.*, 2016; LIMA *et al.*, 2018), the authors propose methods to estimate the Remaining Useful Life (RUL) of HDDs. Considering the RUL estimation, two modeling approaches are commonly chosen. The most straightforward modeling approach considers the problem of estimating the RUL as a regression task and to the best of our knowledge such an approach was not explored for HDDs. Even though such formulation may provide accurate information regarding the RUL, such task is challenging. Moreover, in several applications, the decision maker can tolerate more coarse-grained information without performance compromise. Based on such an idea, several works like (CHAVES *et al.*, 2016; XU *et al.*, 2016) modeled the problems as a classification task where each class defines a degradation level of the monitored equipment. This task is usually referred to as health degree, or status, prediction (XU *et al.*, 2016).

Among all methods for health status prediction on HDDs, the most successful are based on Recurrent Neural Networks (RNN) (CHAVES *et al.*, 2016). It is straightforward to notice that the problem can be modeled as an ordinal classification task since, unlike regular classification problems, all incorrect classifications shall not result in equal penalties. For instance, consider a health status prediction problem with six degradation levels where the first state defines a healthy system and the sixth state defines a situation of imminent failure. In such a scenario, predicting a fourth-degree status for a severely degraded system (sixth level) is significantly better than classifying it in the first degradation level. Another important aspect is that errors in different directions shall also be unequally penalized. Considering the same illustrative health status prediction problem, for a system in level three, one shall penalize predictions of level two, more than predictions of level four. Such a policy can be explained by the fact that excessively early failure predictions may result in way-too-early maintenance actions while late predictions can lead to failures.

1.1 Contributions and text organization

The hypotheses defended by this dissertation are that deep neural networks are suitable models to address the problems of health status assessment and RUL prediction for HDDs. Moreover, this dissertation also defends that a more suitable encoding of health degrees, for the health status assessment problem, can improve the predictions, when taking into account particularities of the problem, such as the ordinal relationship between the health degrees.

The proposed methods were tested in a real-world public dataset collected by a cloud storage service provider comprising 1,697 HDDs. On the basis of the results, we can state that our proposals could adequately handle the particular aspects of both the health status prediction and RUL estimation problems.

In summary, the main contributions of this dissertation are:

- An LSTM based neural network architecture for performing RUL prediction for HDDs.
- A CNN based neural network architecture for performing RUL prediction for HDDs.
- A study of the impacts of RNNs state initialization tailored for improving the prediction of RUL for HDDs.
- An LSTM based neural network architecture for performing the prediction of health degrees for HDDs.
- An encoding technique for the health degrees that improves the predictions when taking into account the ordinal and asymmetric nature of the problem.

As a result of the development and evaluation of these hypotheses, the papers in table 1 were submitted and published, or are in submission process, to the scientific community.

This dissertation is organized as follows: Chapter 2 describes with more detail the theoretical foundations of this work, such as recurrent neural networks and convolutional neural networks; Chapter 3 delves in some related works that also addresses the problem of health status assessment for HDDs; In chapter 4 we explore architectures of deep neural networks that maximize their accuracy when performing RUL estimation. For this, both CNNs and RNNs, more specifically LSTM and GRU, were employed. We also evaluate the impacts of a tailored initialization of these RNNs in this task. Chapter 5 presents our LSTMs based solution for addressing the longer prediction horizon on the health degree prediction problem. It also details our custom encoding solution of health degree classes together with variants of two Deep RNNs for ordinal classification with asymmetric costs. Finally, chapter 6 summarizes the contributions presented and proposes where efforts to develop new work in the area should be directed.

Table 1 – Publications

Title	Authors	Venue
Predicting Failures in Hard Drives with LSTM Networks	**Fernando Lima**, Gabriel Amaral, Lucas Leite, Joao Pordeus, Javam Machado	BRACIS 2017
Remaining Useful Life Estimation of Hard Disk Drives based on Deep Neural Networks	**Fernando Lima**, Francisco Pereira, Lucas Leite, Joao Pordeus, Javam Machado	IJCNN 2018
Evaluation of Recurrent Neural Networks for Hard Disk Drives Failure Prediction	**Fernando Lima**, Francisco Pereira, Iago Chaves, Joao Pordeus, Javam Machado	BRACIS 2018
Asymmetric Ordinal Failure Prediction for Hard Drives with Deep Recurrent Neural Networks	**Fernando Lima**, Francisco Pereira, Iago Chaves, Joao Pordeus, Javam Machado	**To Be Defined**

2 DEEP NEURAL NETWORKS

Artificial neural networks (ANNs) were proposed as a computing model inspired by the biological brains neural networks (ROSENBLATT, 1958). Their structure consists basically in a collection of small processing units, called artificial neurons, that are intertwined by weighted connections. In this setting, the artificial neurons would represent the biological neurons, and the weighted connections, like the synapses of the biological brains, the strength of the signal between those neurons.

Such ANNs can be differentiated based on the way their connections are organized: the ones whose connections form cycles, and those with acyclic connections. The ones with acyclic connections are termed as feedforward neural networks (FNNs). This expression comes from the fact that information flows through their inner computation, the function f modeled by the neuron, being evaluated at some input x, to produce some output value y. There is no flow where the output value is fed back to the neuron. Those with cyclic connections are referred to as recurrent neural networks (RNNs). In this setting, the values produced as output by the neuron are fed back and used during the computation of subsequent outputs.

The most known and widely used form of ANN is the MLP. In this neural network, the units are arranged in layers, with connections feeding forward from neurons of one layer to the next. MLPs with a single hidden layer can be expressed as a composition of two functions, $f^{(1)}$ and $f^{(2)}$, connected in a chain such that its output, $f(x)$, for a given input x, can be written as $f(x) = f^{(2)}(f^{(1)}(x))$. The length of this chain is what defines the depth of the model, and is what originated the terminology "deep learning". For this exemplified network, x is fed to the input layer, $f^{(1)}$ is a hidden layer, and $f^{(2)}$ is the output layer. The forward computation, or pass, consists in letting input information being fed to the input layer, then propagated through the hidden layer(s), and finally to the output layer. This process is depicted in figure 2.

For each neuron in the hidden layers, their computation can be expressed as a weighted sum of their inputs, followed by the application of an activation function. This computation can be written as

$$a_h = \sum_{i=1}^{I} w_i b_i$$

$$b_h = \theta(a_h)$$

Where b is the final output of the neuron, θ is the activation function, w_i is the weight of the connection relative to the input index i and x is the input given to the neuron. An interesting aspect to notice is that, without the application of a non-linear activation function, such as hyperbolic tangent or sigmoid, even with successive computations of hidden layers, the network would still produce a linear operation, which is equivalent to a network with a single hidden layer performing a linear computation. This also means that it would only be capable of producing linear separation boundaries. Another important aspect to take into account when choosing the activation functions is whether they are differentiable, which allows it to be trained with gradient-based optimization methods.

For the output layer, a similar process is performed, but both the activation function and the number of output nodes must be chosen in accordance with the task being solved. The following equations describe the forward pass of the output layer neurons.

$$a = \sum_{i=1}^{H} w_h b_h$$

$$b = \theta(a)$$

Where b is the final output of the neuron, a is the summing of the activations from the outputs of the previous layer, b_h is the output given to the neuron by the h-th neuron in the last hidden layer and w_h is the weight of the connection of the hidden layer neuron relative to this output neuron.

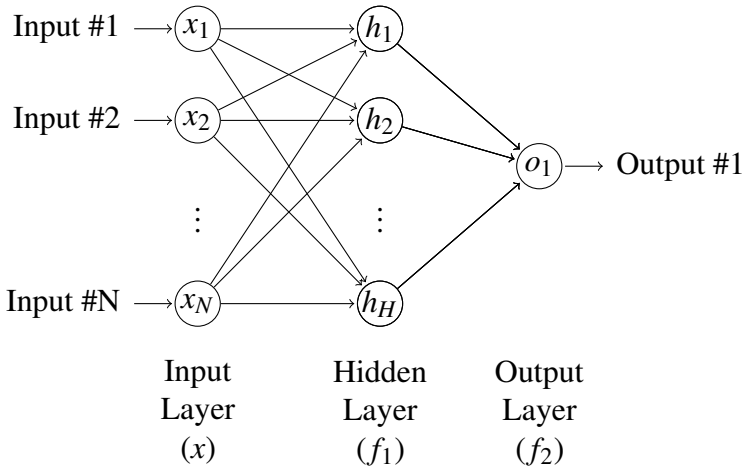

Figure 2 – A multilayer perceptron neural network with a single hidden layer and an output layer with a single output neuron.

Even though MLPs with a single hidden layer were proven to be capable of approximating any continuous function (HORNIK *et al.*, 1989), being referred to as universal function

approximators, they are more suitable for pattern classification than for sequence labeling. This comes from the fact that the outputs modeled by their function mapping depends only on current input, and not on any past or future ones. For general sequence processing, RNNs have been proven to outperform FNNs.

As mentioned before, an ANN produces its predictions in the forward pass. In order to find the weights for the connections that are better suited for a given problem, i.e. a dataset of input-output pairs, several training algorithms can be employed. In case each artificial neuron performs a computation that is differentiable and assuming that the function that measures the difference between expected output and the real outputs from the dataset, i.e. the loss function, is also differentiable, gradient descent methods can be used to find these parameters that minimize such loss functions. The steps taken in order to compute the updates in the weights based on the derivatives of the loss with respect to the parameters is known as the backward pass. To calculate such derivatives for each parameter, one can successively apply the chain rule for partial derivatives, which is a well-known technique named backpropagation (CHAUVIN; RUMELHART, 2013).

In this chapter, we focus our attention on Convolutional neural networks (CNNs), which are a class of feedforward neural networks, whose neurons perform a particular organization of connections and weight sharing mechanism, and on RNNs, more specifically Simple recurrent networks, Long short-term networks, and Gated recurrent units. All of them will be described in detail in the following sections, together with their forward and backward passes.

2.1 Recurrent Neural Networks

A recurrent neural network (RNN) is a neural network specifically designed for addressing problems involving sequential data. This kind of neural network is capable of processing really long sequences of information, even sequences whose length vary, scaling much better than other types of neural networks (e.g. MLP). They have been largely employed on several machine learning problems that can be modeled as sequential data.

Differently from feed-forward neural networks, RNNs allow for the occurrence of cycles, which in turn allows for the propagation of the output in previous steps to be used for new ones. Even though this seems to be a small change, this allows for RNNs to be capable of mapping an entire sequence information to each output of the network. In fact, what happens is that the network can store data from previous inputs in its internal state, through the recurrent

connections, which in turn is used to compute the network output. Also, analogously to the universal function approximation theorem for neural networks, an RNN with a sufficient number of hidden units can approximate any measurable sequence-to-sequence mapping to arbitrary accuracy (HAMMER, 2000).

Many varieties of RNN have been introduced in the literature, including Simple Recurrent networks (e.g. Jordan and Elman), Long short-term memory networks and Gated recurrent units. RNNs with LSTM and Gated recurrent units (GRU) have proven to be an effective tool in modeling dependencies that exist over long sequences. They are effective because they do not suffer from the classical learning problems of vanishing and exploding gradients that affect simple RNNs (SRNs) (BENGIO *et al.*, 1994). Despite the optimization issues in their learning process, SRNs have been successfully applied recently in the task of HDDs failure prediction (XU *et al.*, 2016), leading to state-of-the-art results.

2.1.1 Simple Recurrent Networks

Elman networks, also known as Simple Recurrent Networks (SRNs), were proposed by Elman (ELMAN, 1990) and essentially are back-propagation neural networks with the exception that past activations of the nodes are used in the computation of further steps. That is, differently from feedforward neural networks, they allow for the occurrence of cycles, by basically setting the processing inputs of a given time step to be both the hidden layer activations of the previous step as well as the current entry being processed from a sequence.

To perform the learning in such networks, a well-known algorithm is the back-propagation through time (BPTT) (WERBOS, 1990). In the context of BPTT, the recurrent connection of the nodes is unfolded in time, according to a desired number of steps, depending on the length of the sequences for the task at hand, performing a standard backpropagation after. The main difference between BPTT and the standard backprop is that in BPTT the errors flow from both subsequent layers connection as well as from temporal (recurrent) connections.

2.1.1.1 Forward Pass

The following equations describe the steps performed in the forward phase of SRNs for the j-th unit. Please notice that, differently from the equations for MLPs, we have written

them in matrix form, to simplify the notation.

$$a_j^{<t>} = [W b_{t-1}]_j + [U x]_j$$
$$b_j^{<t>} = \theta(a_j^{<t>})$$

Where θ is the activation function, b the hidden state vector, W the recurrent connection weights and x the input vector. The notation $[.]_j$ indicates the j-th element in the vector. The bias term is embedded inside of the input matrix. Noting that the subscript defines the entry at index t from the sequence. Also, the connections' weights are shared across all indexes.

As can be noticed, at each time step, or index $< t >$, the recurrent cell at layer h produces activation values $b_h^{<t>}$, which in turn can be used by the next layer neurons, at each time step. This is especially useful when performing tasks that require predictions for each entry in a sequence, which is precisely the kind of task tackled in the works approached in this dissertation. There are also other paradigms, such as sequence-to-sequence, commonly used for machine translation tasks, and also the many-to-one organization, which can be used for whole-sequence classification tasks. In summary, depending on the task, different network organizations. or topologies can be employed.

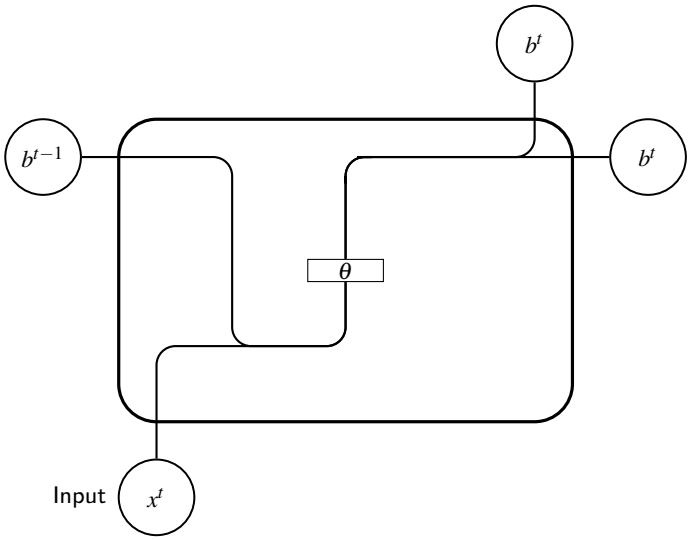

Figure 3 – Details of a SRN node from a single layer. The nodes enclosed in rectangular boxes encapsulate the weight matrix multiplication plus a bias term given as input to the corresponding function in the box. (Adapted from (OLAH, 2015).)

2.1.1.2 *Backward Pass*

Different from MLPs, the loss function on recurrent networks depends not only on the effect of the activation of a given time-step but also on the impacts of the activation on the output of the subsequent time-step. So, assuming a loss function $\mathscr{L}$, recurrent weight matrix W, input weight matrix U, and that the derivative of the loss w.r.t the output of the recurrent cell through the subsequent network layers are denoted as Δo, the partial derivative of the loss w.r.t the output of the recurrent cell j-th unit at time-step t is given by

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial b_j^{<t>}} &= [\Delta o + \frac{\partial \mathscr{L}}{\partial a_j^{<t+1>}} \frac{\partial a_j^{<t+1>}}{\partial b_j^{<t>}}]_j \\ &= [\Delta o + \frac{\partial \mathscr{L}}{\partial a_j^{<t+1>}} W]_j\end{aligned}$$

and the derivative of the loss w.r.t the activations of the recurrent cell j-th unit at time-step t is

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial a_j^{<t>}} &= [\frac{\partial \mathscr{L}}{\partial b_j^{<t>}} \frac{\partial b_j^{<t>}}{\partial a_j^{<t>}}]_j \\ &= [\frac{\partial \mathscr{L}}{\partial b_j^{<t>}} \theta'(a_j^{<t>})]_j\end{aligned}$$

Since all the weights, both recurrent and input, are shared across all time-steps of the network, we need to sum their influence over all time-steps in order to obtain the gradient of the loss w.r.t the weights. This can be expressed as

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial W} &= \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial a_j^{<t>}} \frac{\partial a_j^{<t>}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial a_j^{<t>}} b_j^{<t-1>} \\ \frac{\partial \mathscr{L}}{\partial U} &= \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial a_j^{<t>}} \frac{\partial a_j^{<t>}}{\partial U} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial a_j^{<t>}} x\end{aligned}$$

2.1.2 *RNNs with Long short-term memory*

Even though standard RNNs were a great improvement over MLPs regarding their capability to process sequences more naturally, their ability to keep contextual information (i.e. dependencies) diminishes with the increase in the length of the sequence being processed. This happens because the influence of an input on the network output either vanishes or explodes as they are computed through the recurrent connections of the network. This is also known as the vanishing or exploding gradient problem (BENGIO *et al.*, 1994).

RNNs with Long short-term memory were proposed in order to address this issue. They consist of a set of recurrently connected memory blocks, or cell states, each with a gating mechanism that controls the flow of information into the state vector of the cell. These two modifications enable the network to keep internal state information that occurred over long intervals, improving its learning capabilities.

The standard LSTM version (GRAVES; SCHMIDHUBER, 2005) makes usage of input, output and forget gates. The main idea is to control the contribution of the input data and the previous hidden state adding or removing information to the cell state. In order to understand how this gating mechanism improves the capability of the network to keep information, it is important to perceive, for example, that as long as the input gate remains inactive, the internal state of the cell will remain unchanged by new inputs, and can be made available later, by activating the output gate.

As we will see in the following chapters, although LSTMs offer advantages over standard RNNs, these are more noticeable when tackling problems with long-range dependencies.

2.1.2.1 Forward Pass

The following equations define how all these gates, memory cell, and hidden state activations are calculated for the forward pass of the j-th unit in an LSTM layer.

$$\widetilde{c}_j^{<t>} = \theta([W_c b_{<t-1>}]_j + [U_c x]_j) \quad \text{candidate state} \tag{2.1}$$

$$i_j = \sigma([W_i b_{<t-1>}]_j + [U_i x]_j) \quad \text{input gate} \tag{2.2}$$

$$f_j = \sigma([W_f b_{<t-1>}]_j + [U_f x]_j) \quad \text{forget gate} \tag{2.3}$$

$$o_j = \sigma([W_o b_{<t-1>}]_j + [U_o x]_j) \quad \text{output gate} \tag{2.4}$$

$$c_j^{<t>} = [i \odot \widetilde{c}^{<t>}]_j + [f \odot c^{<t-1>}]_j \quad \text{cell state} \tag{2.5}$$

$$b_j^{<t>} = [o \odot \theta(c^{<t>})]_j \quad \text{output} \tag{2.6}$$

$$\tag{2.7}$$

In these equations, t denotes the index being processed within a sequence x, and $\odot$ is the Hadamard product. The recurrent and input matrices are W and U, respectively, with a subscript indicating the associated gate, and b is the output of the network. Notice that the values of the gates are calculated for each new input from the sequence.

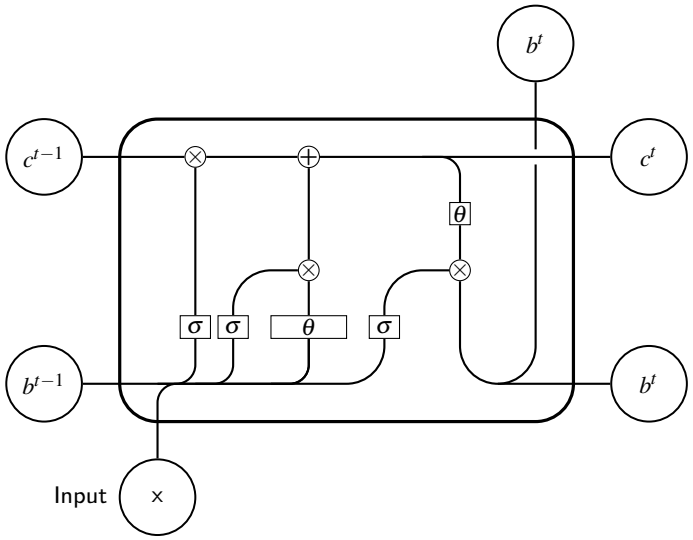

Figure 4 – Details of a LSTM memory cell. The nodes enclosed in rectangular boxes encapsulate the weight matrix multiplication plus a bias term given as input to the corresponding function in the box. Both Hadamard product and Sum operations are point-wise. (Adapted from (OLAH, 2015).)

2.1.2.2 Backward Pass

As we will compute the gradients backward, we start by calculating the gradient of the output of the cell. Assuming that the derivative of the loss w.r.t the output of the recurrent cell through the following network layers is denoted as Δo, the gradient of the loss w.r.t the output is given by

$$\frac{\partial \mathscr{L}}{\partial b_j^{<t>}} = \Delta o + \sum_{h=1}^{H} W_{j,h} \frac{\partial \mathscr{L}}{\partial b_k^{<t+1>}}$$

which basically takes into account the activations in the output (next) layer of the network plus the influence on the outputs of the next timesteps.

For the output gates we have

$$\frac{\partial \mathscr{L}}{\partial o_j} = \sigma'([W_o b_{<t-1>}]_j + [U_o x]_j) \sum_{c=1}^{C} \theta(c_j^{<t>}) \frac{\partial \mathscr{L}}{\partial b_j^{<t>}}$$

where C refers to all cell states. The gradients of the cell states is then given by

$$\frac{\partial \mathscr{L}}{\partial c_j^{<t>}} = o_t \theta\prime(c_j^{<t>}) \frac{\partial \mathscr{L}}{\partial b_j^{<t>}} + f_j^{<t+1>} \frac{\partial \mathscr{L}}{\partial c_j^{<t+1>}}$$

that can be translated as the contribution of the current cell state on the output of the current timestep plus the contribution on the cell state of the next timestep.

The forget and input gate gradients are

$$\frac{\partial \mathscr{L}}{\partial f_j} = \sigma'([W_f b_{<t-1>}]_j + [U_f x]_j) \sum_{c=1}^{C} c_j^{<t-1>} \frac{\partial \mathscr{L}}{\partial c_j^{<t>}}$$

and

$$\frac{\partial \mathscr{L}}{\partial i_j} = \sigma'([W_i b_{<t-1>}]_j + [U_i x]_j) \sum_{c=1}^{C} \widetilde{c}_j^{<t>} \frac{\partial \mathscr{L}}{\partial c_j^{<t>}}$$

The gradients of the input, forget and output gates internal parameters then become

$$\frac{\partial \mathscr{L}}{\partial W_i} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial i_j} \frac{\partial i_j}{\partial W_i} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial i_j} \sigma'([W_i b_{<t-1>}]_j + [U_i x]_j) b_j^{<t-1>}$$

$$\frac{\partial \mathscr{L}}{\partial U_i} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial i_j} \frac{\partial i_j}{\partial U_i} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial i_j} \sigma'([W_i b_{<t-1>}]_j + [U_i x]_j) x$$

$$\frac{\partial \mathscr{L}}{\partial W_f} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial f_j} \frac{\partial f_j}{\partial W_f} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial f_j} \sigma'([W_f b_{<t-1>}]_j + [U_f x]_j) b_j^{<t-1>}$$

$$\frac{\partial \mathscr{L}}{\partial U_f} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial f_j} \frac{\partial f_j}{\partial U_f} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial f_j} \sigma'([W_f b_{<t-1>}]_j + [U_f x]_j) x$$

$$\frac{\partial \mathscr{L}}{\partial W_o} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial o_j} \frac{\partial o_j}{\partial W_o} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial o_j} \sigma'([W_o b_{<t-1>}]_j + [U_o x]_j) b_j^{<t-1>}$$

$$\frac{\partial \mathscr{L}}{\partial U_o} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial o_j} \frac{\partial o_j}{\partial U_o} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial o_j} \sigma'([W_o b_{<t-1>}]_j + [U_o x]_j) x$$

Finally, the derivative of the candidate state internal parameters are

$$\frac{\partial \mathscr{L}}{\partial W_c} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial \widetilde{c}_j} \frac{\partial \widetilde{c}_j}{\partial W_c} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial \widetilde{c}_j} \theta'([W_c b_{<t-1>}]_j + [U_c x]_j) b_j^{<t-1>}$$

$$\frac{\partial \mathscr{L}}{\partial U_c} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial \widetilde{c}_j} \frac{\partial \widetilde{c}}{\partial U_c} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial \widetilde{c}_j} \theta'([W_c b_{<t-1>}]_j + [U_c x]_j) x$$

2.1.3 RNNs with Gated recurrent unit

Motivated by the LSTMs, the GRU was introduced by Cho (CHO *et al.*, 2014) as a new design of RNN cell that is capable of adaptively updating to and reading from the hidden state. The main characteristic of such unit is the need for fewer parameters for operating, leading to less computation and simpler implementations.

Instead of employing three gates, it uses only two, namely the reset and the update gates. Also, to avoid the classical learning issues that affect RNNs, rather than utilizing an

internal memory cell, it forces the hidden state to have its values bound between the previous hidden state and a candidate state (both values limited by the output of θ), calculated at the processing of each new entry.

2.1.3.1 Forward Pass

The equations below specify the calculations performed in the forward phase of the GRU cell to obtain the activation of the j-th hidden unit as well as the two gates needed for its computation.

$$r_j = \sigma([W_r b_{<t-1>}]_j + [U_r x]_j) \qquad \text{reset gate} \qquad (2.8)$$

$$z_j = \sigma([W_z b_{<t-1>}]_j + [U_z x]_j) \qquad \text{update gate} \qquad (2.9)$$

$$\widetilde{b}_j^{<t>} = \theta([W_b(r \odot b_{<t-1>})]_j + [U_b x]_j) \qquad \text{candidate state} \qquad (2.10)$$

$$b_j^{<t>} = z_j b_j^{<t-1>} + (1 - z_j)\widetilde{b}_j^{<t>} \qquad \text{output} \qquad (2.11)$$

Analogously to the LSTM equations defined above, the recurrent and input matrices are W and U, respectively, with a subscript indicating the associated gate or state. x (a vector) is an entry of a sequence, whose elements are processed, one at a time, respecting their positioning t. The notation $[.]_j$ denotes the j-th element of a vector and $b^{<t>}$ is the hidden state produced at timestep t.

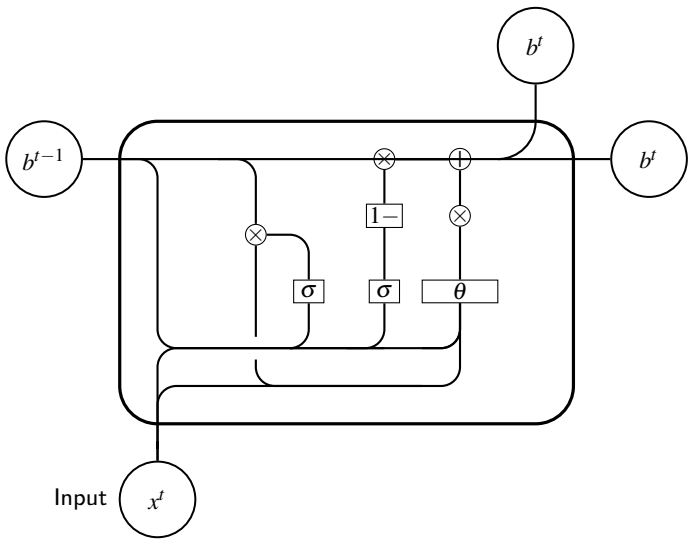

Figure 5 – Details of a GRU memory cell. The nodes enclosed in rectangular boxes encapsulate the weight matrix multiplication plus a bias term given as input to the corresponding function in the box. Both Hadamard product and Sum operations are point-wise. (Adapted from (OLAH, 2015).)

2.1.3.2 Backward Pass

Following the same approach we did for the LSTM case, let's assume that the derivative of the loss w.r.t the output obtained through the remaining layers of the network is Δo, the full derivative of the loss w.r.t the output is

$$\frac{\partial \mathscr{L}}{\partial b_j^{<t>}} = \Delta o + \sum_{h=1}^{H} W_{j,h} \frac{\partial \mathscr{L}}{\partial b_k^{<t+1>}}$$

The derivarive w.r.t the candidate state is

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial \widetilde{b}_j^{<t>}} &= \frac{\partial \mathscr{L}}{\partial b_j^{<t>}} \frac{\partial b_j^{<t>}}{\partial \widetilde{b}_j^{<t>}} \\ &= \frac{\partial \mathscr{L}}{\partial b_j^{<t>}} \frac{\partial b_j^{<t>}}{\partial \widetilde{b}_j^{<t>}} \\ &= \frac{\partial \mathscr{L}}{\partial b_j^{<t>}} (1 - z_j)\end{aligned}$$

The reset gate derivate is

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial r_j} &= \frac{\partial \mathscr{L}}{\partial \widetilde{b}_j^{<t>}} \frac{\partial \widetilde{b}_j^{<t>}}{\partial r_j} \\ &= \frac{\partial \mathscr{L}}{\partial \widetilde{b}_j^{<t>}} \theta'([W_b(r \odot b_{<t-1>})]_j + [U_b x]_j)[W_b b_{<t-1>}]_j\end{aligned}$$

The derivative of the update gate is

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial z_j^{<t>}} &= \frac{\partial \mathscr{L}}{\partial b_j^{<t>}} \frac{\partial b_j^{<t>}}{\partial z_j^{<t>}} \\ &= \frac{\partial \mathscr{L}}{\partial b_j^{<t>}} (b_j^{<t-1>} - \widetilde{b}_j^{<t>})\end{aligned}$$

The derivatives of the update and forget gates internal parameters are

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial W_r} &= \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial r_j} \frac{\partial r_j}{\partial W_r} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial r_j} \sigma'([W_r b_{<t-1>}]_j + [U_r x]_j) b_j^{<t>} \\ \frac{\partial \mathscr{L}}{\partial U_r} &= \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial r_j} \frac{\partial r_j}{\partial U_r} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial r_j} \sigma'([W_r b_{<t-1>}]_j + [U_r x]_j) x \\ \frac{\partial \mathscr{L}}{\partial W_z} &= \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial z_j} \frac{\partial z_j}{\partial W_z} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial z_j} \sigma'([W_z b_{<t-1>}]_j + [U_z x]_j) b_j^{<t>} \\ \frac{\partial \mathscr{L}}{\partial U_z} &= \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial z_j} \frac{\partial z_j}{\partial U_z} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial z_j} \sigma'([W_z b_{<t-1>}]_j + [U_z x]_j) x\end{aligned}$$

and the derivatives with respect to the candidate state internal parameters are

$$\frac{\partial \mathscr{L}}{\partial W_b} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial \widetilde{b}_j} \frac{\partial \widetilde{b}_j}{\partial W_b} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial \widetilde{b}_j} \theta'([W_b(r \odot b_{<t-1>})]_j + [U_b x]_j) b_j^{<t-1>}$$

$$\frac{\partial \mathscr{L}}{\partial U_b} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial \widetilde{b}_j} \frac{\partial \widetilde{b}}{\partial U_b} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial \widetilde{b}_j} \theta'([W_b(r \odot b_{<t-1>})]_j + [U_b x]_j) x$$

2.2 Convolutional Neural Networks

Convolutional neural networks, or CNNs, introduced by LeCun *et al.* (LECUN *et al.*, 1998), are another kind of neural network that was successfully employed in a variety of tasks. They were designed to process data whose structure are in grid-like format. Such structures can be observed in data ranging from time-series, with timesteps collected at regular intervals, and images, where the pixels form a two-dimensional grid. By utilizing convolution layers in deep neural networks, state-of-the-art results were achieved in tasks about sequential modelings, such as sentence classification (CONNEAU *et al.*, 2017) and signal processing (HERSHEY *et al.*, 2017), besides their traditional application on image classification.

CNNs work basically by reducing the number of connections, and therefore parameters, of the neurons in the hidden layer, if compared to a traditional fully-connected network. This offers great advantage such as a reduction in the memory required to store the parameters besides the efficiency in their computation. Essentially, each neuron only gets input from a local region of the input data, often referred to as sparse connectivity. Also, all neurons share the same kernel weights, or parameters also termed as filter. It is common to have multiple filters, each giving a different interpretation (filtering) over the data, also called feature maps. Usually, several convolution layers are applied sub-sequentially, with a pooling layer in-between.

The pooling layer is responsible for reducing the data dimension received as input, increasing the invariance of the network to small changes, such as a shift in a signal. Besides that, pooling also reduces the number of parameters to be learned by further layers, because of the dimensional reduction it performs in the input. The most frequently used pooling layer is the max-pooling, which takes the maximum value of the local input region given to each of its neurons. This organization of multiple layers increases the range of inputs that neurons in further layers receive, basically increasing their receptive fields relative to the initial input of the network.

2.2.1 Forward Pass

The following equation defines the computation performed by each neuron in one filter of a CNN for a single dimensional data with only one feature per entry:

$$a_i = (X * W)_i = \sum_{m=0}^{k} W_m X_{i+m} \tag{2.12}$$

$$b_i = \theta(a_i) \tag{2.13}$$

The output of each neuron of a filter map is the sum of the convolution operation, with a kernel defined by W, with size k, performed on each channel (i.e. depth or feature), in this case, only one, over the input data X. Optionally, the filtering result is passed through a non-linear activation function σ, generally is the rectifier, which is known to result in networks with sparser representations and faster training time (KRIZHEVSKY *et al.*, 2012).

2.2.2 Backward Pass

Assuming that the derivative of the loss relative to the output of the layer is given by $\frac{\partial \mathscr{L}}{\partial b_i}$, we are interested first in computing the derivative of the loss with respect to the network activations

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial a_i} &= \frac{\partial \mathscr{L}}{\partial b_i}\frac{\partial b_i}{\partial a_i} \\ &= \frac{\partial \mathscr{L}}{\partial b_i}\theta'(a_i)\end{aligned}$$

Now for the derivative of the activations with recpect to each element in the kernel W is

$$\frac{\partial a_i}{\partial W_m} = \frac{\partial}{\partial W_m}\left(\sum_{n=0}^{k} W_n X_{i+n}\right) \tag{2.14}$$

$$= \frac{\partial}{\partial W_m}(W_m X_{i+m}) \tag{2.15}$$

$$= X_{i+m} \tag{2.16}$$

from step 1 to step 2, observe that only index $n = m$ contributes to the gradient of W_m. Finally, the derivative of the loss with respect to each element in the kernel W is

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial W_m} &= \sum_{i=0}^{T-k} \frac{\partial \mathscr{L}}{\partial a_i}\frac{\partial a_i}{\partial W_m} \\ &= \sum_{i=0}^{T-k} \frac{\partial \mathscr{L}}{\partial b_i}\theta'(a_i) X_{i+m}\end{aligned}$$

Notice that the last summation occurs because of the weight sharing that is performed by the CNN layer.

3 RELATED WORK

In this chapter, we will present recent related work on the topic of HDD failure prediction.

We start by discussing in section 3.1 the work of (QUEIROZ *et al.*, 2016) that formulates the HDD failure prediction problem as an incipient failure detection task. Following, in section 3.2, we explore the work of (BOTEZATU *et al.*, 2016) that also addresses the incipient failure detection task.

Then, in section 3.3, we discuss the work of (CHAVES *et al.*, 2016) that address the prediction problem as coarse-grained regression with Bayesian networks. It bears similarity with the classification approach because it predicts the RUL quantified as trimesters. Finally, in section 3.4, we discuss the work of (XU *et al.*, 2016) that models the problem as a multiclass classification task, where the classes are the health degrees, or levels, of the devices.

3.1 A Fault Detection Method for Hard Disk Drives Based on Mixture of Gaussians and Non-parametric Statistics

Queiroz *et al.* (QUEIROZ *et al.*, 2016) proposed a failure prediction methodology that is based on the paradigm of anomaly, or incipient failure, detection. The first step in their method is to select the features that are the most relevant to discriminate a failing HDD. They do so by running the Recursive Feature Elimination (RFE) technique using the Random Forest (RF) as the basis classifier.

Once they have only the selected features at hand, they propose to construct a baseline statistical model with SMART data captured only from healthy HDDs. The idea of this model is to serve as a classifier to measure how likely a SMART sample is to have come from a non-failing HDD. This model is built by fitting a Gaussian Mixture Model (GMM). The number of Gaussians is defined through the execution of the Bayesian information criterion (BIC), which is a solution to avoid overfitting. Once they build this healthy HDD GMM model, they create dissimilarity vectors also from the healthy HDDs relative to the GMM model through the computation of log-likelihood.

In the fault detection procedure, they construct dissimilarity vectors for the HDDs being tested, analogously to the step performed for healthy HDDs. Then they traverse these newly created time series of dissimilarity vectors with a sliding window approach, for both

healthy and test samples, in order to estimate distributions of the vectors using Kernel Density Estimation (KDE), within the window. Finally, the difference between these distributions is computed through Kullback-Leibler Divergence (KLD). If such divergence exceeds a predefined threshold, then a fault is detected. Their approach is able to achieve 92.21% Failure Detection Rate (FDR) at 0% False Alarm Rate (FAR).

3.2 Predicting Disk Replacement towards Reliable Data Centers

The problem of incipient failure detection was also investigated in Botezatu *et al.* (BOTEZATU *et al.*, 2016). In this work they propose a novel data mining approach that is able to predict disk replacements based on historic disk replacement data from a cloud storage service provider. As the work discussed in section 3.1, they also employ the use of SMART sensors attributes. Their work has two main contributions, a selection of SMART attributes that are informative for detecting when a disk should be replaced; and a statistical model built from these attributes that achieve high accuracy when predicting disks that should be replaced.

As a first step, feature selection is employed to determine relevant SMART attributes that are indicative of impending disks replacements. This is done by performing a change point detection procedure, detecting change points in the attributes time series. They argue that when an attribute is informative, it must present a significant shift in its value in a moment (change point) before the device fails. This shift is represented by a change in the distribution of the attribute in such a way that the sum of the log likelihood of the distribution before and after the change point is maximized.

The time-series of the selected SMART attributes is then compacted, with the objective of producing a compact, but highly informative representation to be used in the prediction model. They argue that this is needed since a single entry in the time-series is not informative enough and also that a model built only with entries representing only the last day of the devices is not suited to anticipate such failures. Similar to the work in section 3.1, they also employ a sliding window in this compaction. With a computed time window slide width, they apply exponential smoothing, reducing the whole window to a single value, for each attribute. The time window width is defined as the median of the distribution of the time stamps when the significant change occurred.

Since the amount of samples is highly unbalanced for each class, failing and non-failing devices, and classification algorithms tend to maximize overall accuracy (favoring the

class with most samples), they propose a class balancing technique. Their algorithm basically runs a K-means clustering on the densest class, selecting the samples that are closer to the centroid as representatives of this class. In this procedure, the number of clusters is set to a number that is close to the number of samples of the other class.

To perform the detection of failing devices, they employ regularized greedy forests (RGF) classifier using the datasets produced in the previous steps. They argue that this method produces better predictions if compared to other ensemble methods, such as gradient boosted decision trees (GBDT), and other classical methods, such as SVM or logistic regression. This model achieves an accuracy of up to 98%, predicting failures with a Time In Advance (TIA) of 10-15 days.

A transfer learning technique is also explored enabling the creation of classifiers for HDD models with few data. This additional step was able to provide an enhancement of up to 50% accuracy if compared to a classifier built with only the data at hand.

3.3 BaNHFaP: A Bayesian Network Based Failure Prediction Approach for Hard Disk Drives

The work presented by Chaves *et al.* (CHAVES *et al.*, 2016) explores a Bayesian network for failure prediction of hard disk drives. To the best of our knowledge, this is the first work that addresses the problem as a failure prediction task, not only incipient failure detection, measuring the remaining number of months that the device would operate before failing. Their method comprises two main steps, namely, preprocessing and estimation of parameters. Similarly to the work in section 3.2 their method is also tested in a real-world dataset, collected by a cloud storage service provider.

The proposed method starts by applying the same procedure used by (QUEIROZ *et al.*, 2016) for feature selection, described in 3.1. Later, they perform binning on the selected SMART data using the MDLP algorithm, except for Power on Hours (POH) attribute, that quantifies the amount of time that the HDD has been operating, which is discretized using equal-width bins.

After that, the Bayesian Network parameters are estimated, which take into account POH, RUL and SMART attributes as random variables. The parameters are estimated basically by calculating the relative frequency that each node in the Bayesian network assumes one of its possible values. Once these frequencies are computed, they apply Additive (or Laplace)

Smoothing, in order to obtain a better posterior probability.

With this model they are able to achieve a smaller density of both mean and median of the quadratic errors is obtained if compared to the standard reliability-based approach used as a baseline. In addition to that, they argue that an improvement of 28.3% and 17.6% of the mean and median of the quadratic errors, respectively, can be noticed. Finally, they claim that the model starts to produce accurate predictions ahead of the baseline method, for all HDDs in the test dataset.

3.4 Health Status Assessment and Failure Prediction for Hard Drives with Recurrent Neural Networks

Recently, Chang Xu *et al.* (XU *et al.*, 2016) described a predictive model that performs proactive drive failure prediction, that contrasts with the usual passive fault tolerance technique. They argue that this approach allows the handling of failures in advance, and therefore greatly reduces impacts on both the availability and reliability of storage systems. Similarly to the other works described in this chapter, their method also makes use of SMART sensors attributes. The main contribution of this work is the claim that the HDD failure prediction problem belongs to long-range dependency. In order to solve that, they propose the use of an RNN-based solution to exploit the temporal dependencies in the time-series of SMART attributes.

Their model consists of three main steps: first, they quantify the health status of the devices based on the amount of time before it breaks; second, they train an SRN neural network to perform classification based on the health status computed previously; finally they give a time-series of SMART attributes to this trained model to produce the predictions.

The first step receives as input a scheme that discretizes a predefined time interval of the life of HDDs into bands, termed as health degrees, according to the remaining lifetime, but argues that different settings can be employed. Despite that, they perform tests only in the last month of the devices, split into six health degrees.

In the second step, they build a single layer SRN using BPTT as the training algorithm. They justify the use of BPTT, arguing that it is more efficient than global optimization methods, particularly for large-scale datasets, besides being suited for online learning. The optimization of this network is performed using Stochastic Gradient Descent (SGD).

For the inference or prediction step, they feed the network with the ordered SMART attribute sequences, together with the hidden state produced by previous entries, in order to ob-

tain a current prediction. After labeling the time series samples, a voting algorithm is employed in the outputs of the last N consecutive samples before a time point, resulting in the final estimate of the device's health level.

They compare their method with five other methods: a Hidden Markov Model, a Binary Neural Network, a Classification Tree, a multiclass Neural Network and Conditional Random Fields. Three real-world datasets, from devices of different manufacturers, are employed in the evaluation. Their best results occur in the first dataset, where the model achieves an FDR of 97.71% with a FAR of 0.06%. Overall, their method leads to results superior to all baseline methods, but the Multiclass NN, for one of the datasets, where they achieve the same FDR.

3.5 Related Work Comparison

Both the works of Queiroz *et al.* (QUEIROZ *et al.*, 2016) and Botezatu *et al.* (BOTEZATU *et al.*, 2016) deal with the task of incipient failure detection. Even though they approach the same task, a major difference regarding their assessment can be traced. The former assesses their method with a small dataset (MURRAY *et al.*, 2005), comprising only 369 hard drives, with 191 failing devices, collected from a non-uniform setting, good drive data was collected in a controlled uniform environment and the failed data comes from drives that were operated by users, whilst the later utilizes a real-world public dataset (BACKBLAZE, 2016) with 30.107 hard drives of two different brands, and two models per brand, of which 872 failed.

Note that the works of Chaves *et al.* (CHAVES *et al.*, 2016) and Chang Xu *et al.* (XU *et al.*, 2016) are the only that aim to predict the RUL of HDDs. By analyzing both works, we can state that Chang Xu *et al.* (XU *et al.*, 2016) showed the most promising results. A possible reason relies on the use of an RNN model, which is capable of directly handling the time dependency that is inherent to the problem. On the other hand, the method proposed by Chaves *et al.* (CHAVES *et al.*, 2016) only considers a snapshot of the SMART attributes to make an RUL prediction. Although the method proposed by Chang Xu *et al.* (XU *et al.*, 2016) had a remarkable performance, it is well known that RNNs suffer from the problem of gradients vanishing/exploding. This problem is more evident for tasks with long-term dependencies.

Table 2 summarizes the main characteristics of each related work and how they compare to one another, including the works presented in this dissertation.

Table 2 – Related Work Comparison.

Authors.	(QUEIROZ *et al.*, 2016)	(BOTEZATU *et al.*, 2016)	(CHAVES *et al.*, 2016)	(XU *et al.*, 2016)	**Ours**
Research Problem.	Fault detection.	Fault detection	Failure Prediction (Classification)	Failure Prediction (Classification)	Failure Prediction (Classification and Regression)
Dataset.	(MURRAY *et al.*, 2005) non-uniform	(BACKBLAZE, 2016)	(BACKBLAZE, 2016)	(ZHU *et al.*, 2013) and two others	(BACKBLAZE, 2016)
Dataset Timespan.	25 days	17 months	21 months	1 month	12 months
Main Technique.	Gaussian Mixture Model	Regularized Greedy Forest	Bayesian Network	Simple RNN	LSTMs, GRUs and CNNs

4 REMAINING USEFUL LIFE PREDICTION

In this chapter, we will approach the health assessment problem of hard disk drives failure as a regression task. More specifically, the task will consist in predicting the remaining useful life, in days, until the device fails. To the best of our knowledge, no other work approached this problem with such fine-grained modeling. A very likely reason for that is that this modeling approach leads to a task that is harder to tackle, due to the accuracy requirements imposed to perform such prediction.

We start by presenting a series of prognostics metrics, in section 4.1, that will be employed when performing the evaluation of the proposed solutions for this regression problem. Following that, in section 4.2 we describe the dataset used in the evaluation of the models proposed in this chapter.

In section 4.3, we present a solution to the problem approaching it as a standard regression task with RNNs and CNNs. In order to find adequate topologies for these networks, we apply Bayesian optimization. A custom penalization technique that gives more importance to the prediction as they approach the devices end of life is also evaluated. In order to assess the results produced by this solution, we evaluate the resulting model with well-known prognostics metrics.

Then, in section 4.4, we improve over this solution, by assessing how a set of initialization strategies for RNNs impacts on the prognostics metrics aforementioned.

Finally, in section 4.5 we provide conclusions and directions for future work taking into account the results of both approaches explored for the health assessment problem.

4.1 Prognostics Metrics

Since failure prediction methods are an important tool in supporting decision makers, this leads to an increased demand in the assessment of the quality of such predictions. The most straightforward metric that can be employed, since the prediction of Remaining Useful Life (RUL) is essentially a regression problem, is to use the average Root Mean Squared Error (RMSE). Additionally, metrics that were designed specifically to quantify the trust in such health assessment models were proposed by (SAXENA *et al.*, 2009). In this dissertation, we will use two of the metrics presented there. The first one is a modified version of Prognostic Horizon metric. The second is the α-λ Performance metric. In the following subsections, we

will explain how the last two metrics are calculated.

4.1.1 Prognostic Horizon

The Prognostic Horizon (PH) calculates the time interval between the instant that the device fails and the instant that the predictions coincide to the RUL according to a specified performance criterion. The performance criterion is defined according to a α parameter, which is an error margin around the true RUL. PH is calculated as follows:

$$PH = EoL - t \tag{4.1}$$

where EoL is the instant when a device stops working, $t = min\{i \mid (i \in T) \wedge (r^d_*(i) - \alpha \leq r^d(i) \leq r^d_*(i) + \alpha)\}$ is the instant when the predictions enter the error margin α, T is the set of trimesters of a device d, $r^d(i)$ is the RUL estimated by the proposed method at time i regarding a device d and $r^d_*(i)$ is the ground truth RUL of d in i.

Notice that the lower the instant t, the higher the score obtained by the PH. Thus, we can infer that an algorithm A yields reliable results farther in advance of an algorithm B if A gets a PH greater than B.

We are using an improved version of the PH metric that considers the consistency of the results. In this version, referred to as PH*, we consider that an instant is valid only if the prediction enters the tolerance area defined by α and stays there until the failure instant.

4.1.2 α-λ Performance

The α-λ Performance quantifies the accuracy of a prediction within a specific time instant. Similarly to PH, this metric verifies if a prediction coincides with the RUL of a device considering an error margin α in a time instant λ. The α-λ Performance is calculated as follows:

$$\alpha\text{-}\lambda = \begin{cases} yes & \text{if } (r^d_*(\lambda) - \alpha \leq r^d(\lambda) \leq r^d_*(\lambda) + \alpha) \\ no & \text{otherwise} \end{cases} \tag{4.2}$$

where α is an error margin, λ is a time instant, $r^d(\lambda)$ is the RUL estimated by the proposed method at time λ regarding a device d and $r^d_*(\lambda)$ is the ground truth RUL of d in λ.

For the purpose of better evaluating the results of the proposed method, we varied the α parameter of the α-λ Performance across the evaluated time interval. As the λ parameter increases, the α parameter is diminished, resulting in a tighter error margin when the device is closer to its EoL.

For both metrics, the parameter α was set to 45 days, meaning that a prediction misplaced by a month and a half is tolerable.

4.2 Dataset

To assess the methods proposed in this chapter, we use a dataset provided by the Backblaze Company. This dataset consists of the daily observation of 92,348 HDDs between april/2013 and december/2016 (BACKBLAZE, 2016). These observations contain information regarding the serial number, model, capacity, failure, and 90 SMART attributes of each device. According to the Backblaze Company, a device is labeled as failed if: (i) it stops working, i.e. does not turn on or does not receive commands or (ii) if SMART self-test fails for attributes 5, 187, 188, 197, or 198. Different manufacturers and device models may report different attributes and most HDD models do not report all SMART attributes. In this case, the values not reported are left blank.

To avoid a potential overfitting, we chose to perform the tests with the Seagate ST4000DM000 model whose data are most plentiful. This model has 36,555 observed disks, of which 1729 have failures. Of these, 32 were removed because their observation was interrupted without a label indicating a failure or by having submitted observations after being labeled as damaged. Thus, we used in fact for the test of the compared models a set of observations on 1,697 instances. Also, as a preprocessing step, we scaled each feature individually by its maximum absolute value, in a way that the maximum absolute value of the feature will be 1.

Due to technical limitations, we used the raw attributes returned by the feature selection process described in (BOTEZATU *et al.*, 2016). These attributes are described in Table 3.

4.3 Remaining Useful Life Prediction

In this section, we approach the prediction of the RUL of HDDs as a regression problem. The task consists in predicting the remaining lifetime of a given HDD based on the

Attribute ID	Attribute Name
SMART 1	Read Error Rate
SMART 5	Reallocated Sectors Count
SMART 7	Seek Error Rate
SMART 184	End-to-End error
SMART 187	Reported Uncorrectable Errors
SMART 188	Command Timeout
SMART 189	High Fly Writes
SMART 190	Temperature Difference
SMART 193	Load Cycle Count
SMART 194	Temperature
SMART 197	Current Pending Sector Count
SMART 198	Uncorrectable Sector Count
SMART 240	Head Flying Hours
SMART 241	Total LBAs Written
SMART 242	Total LBAs Read

Table 3 – SMART attributes used for the regression task.

current and previous SMART attributes. Based on that sequence of data, the model is inquired to give an output RUL, measured in days, for the last SMART entry.

Two deep neural network models are explored. The first model is built using LSTMs and the other one, CNNs. To find an adequate topology for these models, Bayesian optimization was applied to select both the hyperparameters and the topology information for each network type. The optimizer implementation was taken from (The GPyOpt authors,). The metric being minimized by this Bayesian optimization routine was the cost function of the validation set, which is the root mean squared error. The following sections present the architectures resulting from the optimization procedure.

4.3.1 Architecture of the LSTMs based network

For the LSTMs based DNN, the Bayesian optimization routine could vary the number of recurrent layers and the size of the hidden state vector. The candidate values to be used as the number of layers were $[1,2,3,4]$. The hidden state vector size could assume the values $[16,32,64,128]$. The resulting architecture was defined to have two recurrent layers, each with a hidden state of size 64 and a hyperbolic tangent activation function, with a fully-connected neuron in the last layer, using identity as the activation function. The topology can be seen in figure 6.

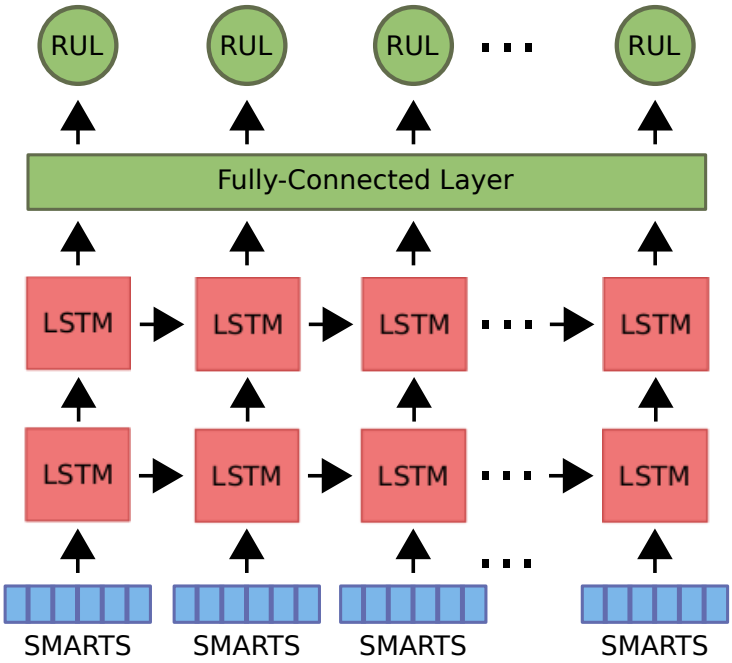

Figure 6 – Architecture of the LSTM network found through optimization.

4.3.2 Architecture of CNN found through optimization

For the optimization of the CNN DNN, 1D convolutional (Conv1D) layers were employed. The optimizer was able to vary the absence of pooling layers between the convolutional layers and the type of pooling applied (min, max or average). It was also possible to choose the activation function of the convolution units as either the identity or the rectifier. The number of Conv1D layers could be either two or three. For these layers, the optimizer could vary the number of filters (between 5 and 50), their size (between 5 and 30), the stride applied to the filters (between 2 and 20) and the type of padding to be applied to the sequences (either zero-padding or no padding). Also, a fixed fully-connected layer, with a single neuron, was set as the last layer in the network, to output the predicted RUL. A representation of the best architecture found by the optimizer can be seen in figure 7.

It is important to emphasize that in order to have a daily output from the proposed CNN architecture, for each SMART entry from an HDD time series, all sequences were amplified so that for each entry in a given sequence, a new sub-sequence was generated up to that entry. This procedure was not necessary for the RNNs, as their functioning already allows predictions at every new SMART entry.

For training the networks an Adam optimizer was used with an exponential decaying learning rate. The training was run for 100000 epochs with learning rate starting at 0.003 and finishing at 0.0001. Also, a dropout of 0.4 was used on the LSTM layers. Cross-validation

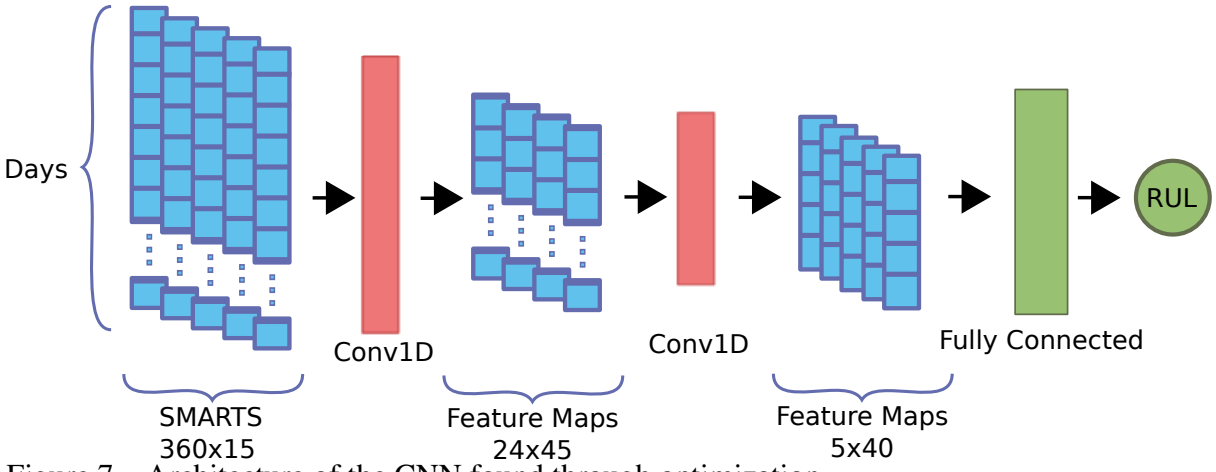

Figure 7 – Architecture of the CNN found through optimization.

	Metrics		
Models	**RMSE**	**PH***	$\alpha - \lambda$
LSTM	**56.0752**	**193.6011**	**0.4692**
CNN	64.3943	154.8583	0.3375
RNN	121.0135	0.0000	0.0780

Table 4 – Performance of the prediction models under different metrics.

was employed and the parameters were chosen based on the epoch with the smallest validation set loss.

4.3.3 Results and Discussion

The LSTM and CNN networks were compared to an Elman Recurrent Neural Network (RNN) according to the three aforementioned metrics. The RNN model was configured to have two hidden layers, each with an internal state size of 64 units and a fully-connected layer, without activation function. The same RNN configuration used in (LIMA *et al.*, 2017). The results for all three models are shown in Table 4.

As can be noticed, both deep learning models significantly outperformed the RNN in all three metrics. It is also noticeable, according to the PH metric, that the RNN could not provide consistent predictions for any of the HDDs. The zero value of the PH metric indicates that none of the predictions could consistently stay inside the RUL band defined by the α parameter for any HDD. These results may indicate that some temporal dependencies could not be adequately modeled by a standard RNN.

We can also notice that LSTM outperformed CNN on all metrics. This result is expected since LSTM models have built-in mechanisms to model time-dependent relations. To

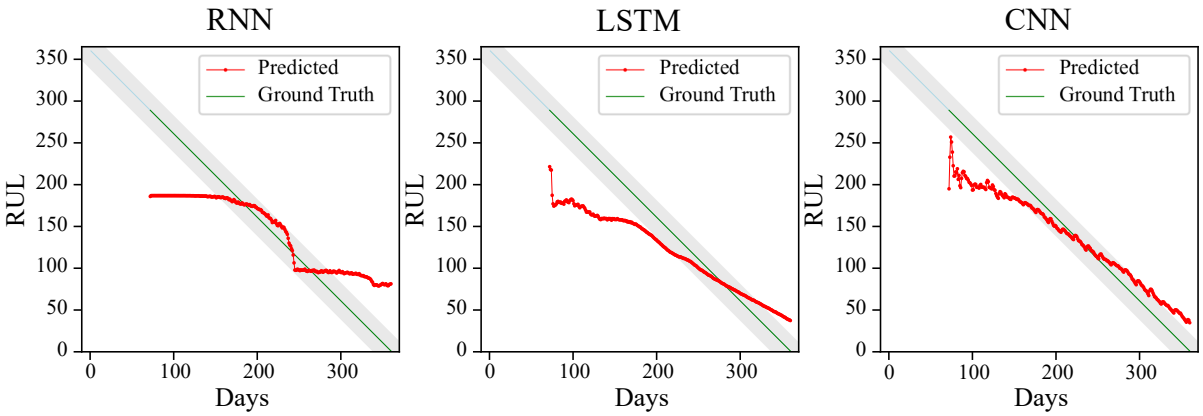

Figure 8 – Predicted and real RUL for HDD Z300ZQST.

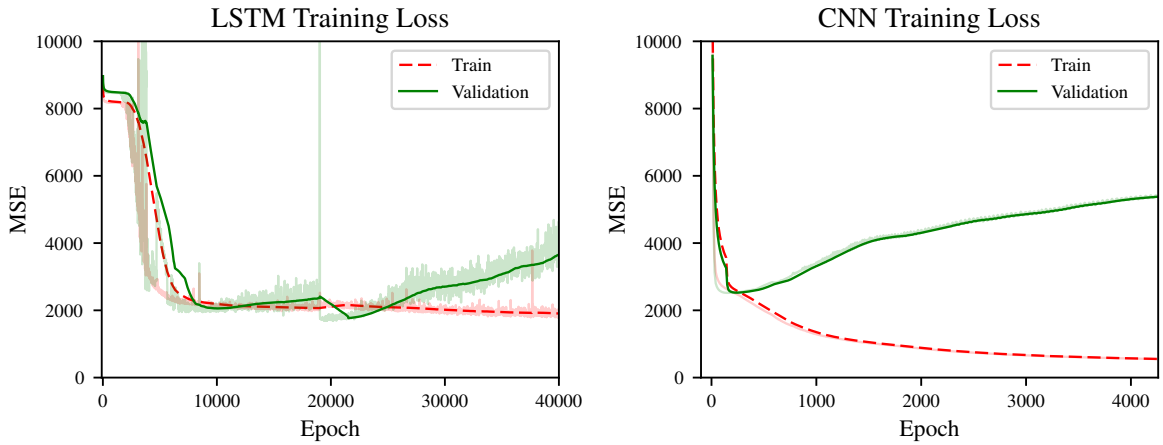

Figure 9 – Loss function along the iterations for CNN and LSTM

provide a more tangible notion of the performance gap between all models, Figure 8 presents the results for all models in HDD with serial number Z300ZQST.

The illustrative plot in Figure 8 depicts a typical behavior of most tested HDDs. One can observe that both CNN and LSTM converge to the ground truth RUL earlier than RNN. In addition to that, both models tend to stay very near to the ground truth as it approaches the failure time.

An additional comparison between CNN and LSTM can be done regarding the convergence of each model. Figure 9 shows the learning curve for both models. Considering this result, we can verify that CNN has a significantly faster convergence (fewer iterations), although LSTM achieved better results in all prognostics metrics.

One interesting aspect that can be noticed in Figure 8 and observed in many of the tested HDDs, is the presence of a prediction gap for the CNN when the RUL reaches its lowest values. This behavior is undesirable since, in a real application, a decision maker needs more accurate predictions as the failure approaches. To overcome such problem, we tested a new

	Metrics				
Models	**RMSE**	**PH***	$\alpha-\lambda$	**W-RMSE**	**W-$\alpha-\lambda$**
CNN	64.3943	**154.8583**	**0.3375**	62.5862	**0.2822**
CNN*	**61.8275**	143.9450	0.2265	**58.6983**	0.2702

Table 5 – Performance of the CNN-based prediction models under different metrics. The CNN* is the model trained with the customized loss function. W-RMSE and W-α-λ are the metrics modified to incorporate the weighting scheme.

CNN model with a modified loss function. The modified loss function aims to emphasize the notion that errors near the failure instant are more critical than errors when the RUL is high. The new loss function is given by:

$$l = \sqrt{\frac{\sum_{i=1}^{N} w_i (RUL_i - \hat{RUL}_i)^2}{\sum_{i=1}^{N} w_i}}, \tag{4.3}$$

where RUL is the real remaining useful life, $\hat{RUL}$ is the predicted remaining useful life and w are the weights of each prediction. In our setup, the weights are designed according to the real RUL as follows:

$$w_i = \frac{180}{180 + RUL}, \tag{4.4}$$

According to Eq. 4.4, predictions at a $RUL = 1$ will be weighted with a value close to 1. This value decreases as RUL increases.

To assess the impact of the modified loss function, we created two additional metrics by modifying the RMSE and the $\alpha - \lambda$ metrics. In both metrics, we included the same weighting scheme used in the loss function. As a consequence, both metrics shall weight more the error occurrences for low RUL values. The results of the standard and these additional metrics are shown in Table 5.

By observing the results on table 5, we can see that the modified loss function had the desired impact on the weighted version of RMSE. We also noticed a small variation on the modified $\alpha - \lambda$. Also, as expected, the values of the regular metrics degraded. The effect of the modified loss function illustrated in Figure 10.

As one can perceive, the prediction of the modified CNN moved towards the direction of the ground truth for low RUL values.

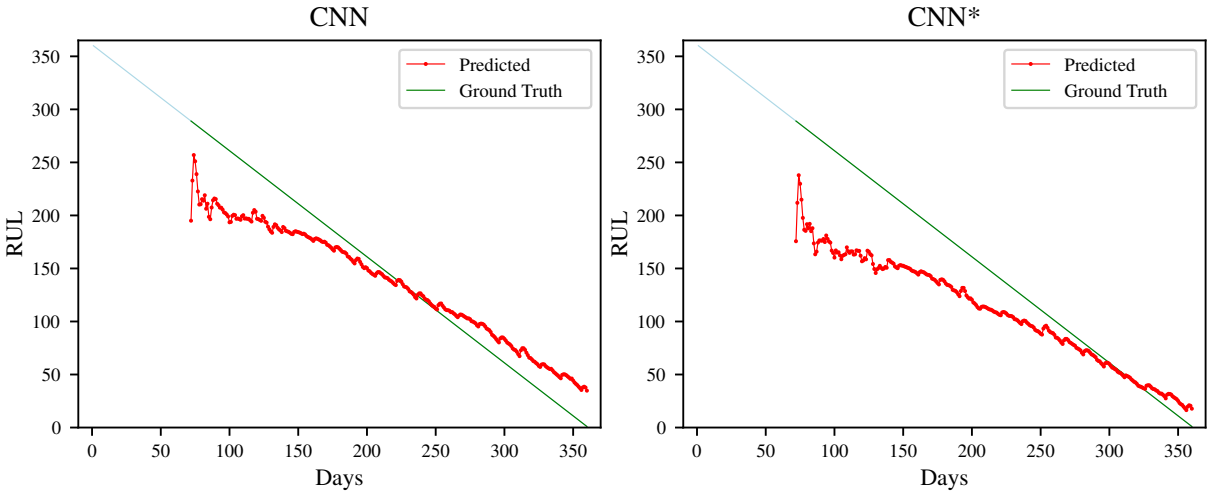

Figure 10 – Predicted and real RUL for HDD Z300ZQST with the modified loss function.

4.4 RNNs State Initialization

In this section, we evaluate the sensitivity of RNN-based models to three hidden state vectors initialization procedures in the task of HDD failure prediction, when approached as a regression problem. According to (ZIMMERMANN *et al.*, 2012) the initial state of an RNN can have a significant impact on its performance. This fact is even more noticeable in predictions where the initialization vector is used as a direct input for the RNN (i.e. the first predictions). For the subsequent predictions, this effect is diminished. Aiming to improve the performance of all RNNs, different approaches for initializing the state vectors of the networks were explored.

A usual solution for the definition of an initialization vector is to set it to be equal to zero. This is done expecting that the network is capable of reducing the impacts of a bad initialization (e.g. zero) during the training. As it is the usual initialization technique, it is the first strategy assessed.

Another possible initialization procedure is to apply noise over the values of the initial state for each sequence. This would, in turn, result in a network that is less dependant on the values of the initial state. A technique proposed by (ZIMMERMANN *et al.*, 2012) goes even further and makes the noise vary according to the error magnitude. We employ a simpler version of this technique with a fixed noise, sampled from a normal distribution with mean 0 and standard deviation 0.1 and added to an initial state of zero. This one will be termed as random initialization.

A last state initialization technique consists of making it a trainable parameter of the model. The expectation is that the model is capable of learning an initial state optimized for

the task. An obvious downside is the addition of more parameters to be learned.

4.4.1 Results and Discussion

The architecture employed in the RNNs based models was the same found for the LSTM network explored in section 4.3. All models were compared according to the three prognostics metrics, presented in section 4.1, and the number of iterations until convergence. Table 6 presents the average metric values of 3 repetitions for all RNN models. For both the PH* and the α-λ Performance metrics, the parameter α was set to 30 days, meaning that a prediction misplaced by a month is tolerable.

		Results			
Method	Initialization	RMSE	PH*	$\alpha - \lambda$	Iterations
SRNN	Zero	75.390411	78.94075145	0.2105486762	50270
	Random	88.03063965	0.2023121387	0.1209358888	67967
	Trained	66.30125216	51.19942197	0.2632570798	77700
GRU	Zero	59.77560713	158.4039017	0.3337110415	**18350**
	Random	60.30418777	120.5283237	0.3188318023	19900
	Trained	58.96779843	164.2450867	0.3087288983	24675
LSTM	Zero	57.43077586	167.2109827	0.3544073512	50270
	Random	57.07623889	167.6136802	0.3684839272	39967
	Trained	**56.32258273**	**177.3947977**	**0.4113254414**	54075

Table 6 – Performance of the prediction models combined with the initialization techniques under different metrics.

As can be observed, LSTM had the best overall performance when considering only the prognostics metrics even though the performance gap between GRU and LSTM was small. It is also noticeable that, in general, for all RNNs the use of a trained initialization resulted in better metric values. The impact of the initialization procedure is even more noticeable in the PH* metric. This result is expected since using a better initialization vector may improve long-term predictions, where the past information (previous SMART data) is scarce. The effect of the trained initialization is illustrated in Figure 11. In this figure, we present one of the LSTM predictions for HDD Z300NCXQ using zero, random and trained initialization techniques.

It is possible to verify that the trained initialization vector modifies the initial predictions by approximating them to the true RUL. Along the days the predictions of the trained and the other strategies tend to be similar since the RNN learns to compensate the effect of a bad initialization.

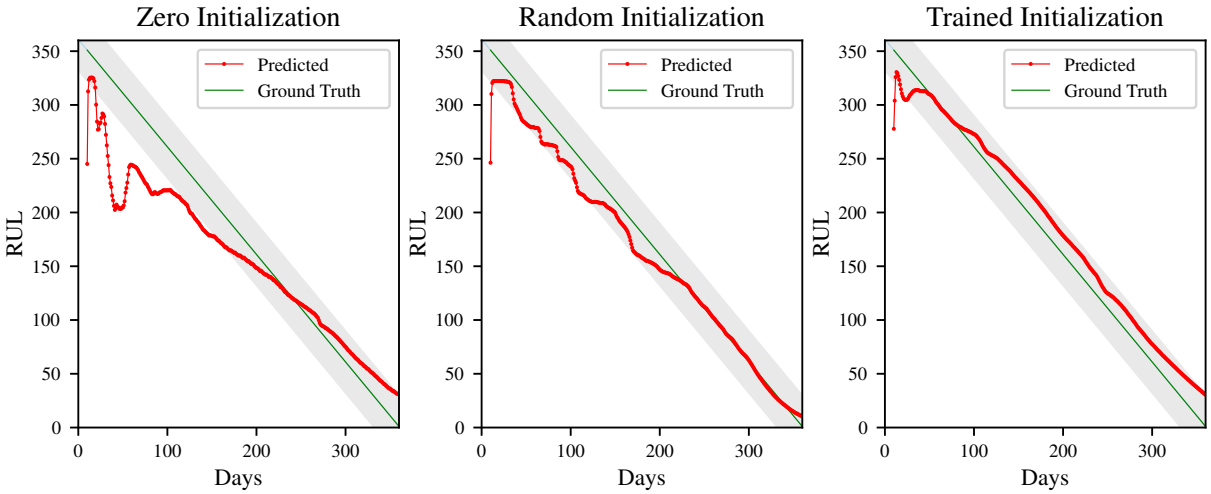

Figure 11 – Predicted and real RUL for HDD Z300NCXQ produced by the LSTM model. The gray area represents an acceptable range defined by $\alpha = 30$ for the prognostic metrics.

Concerning model convergence, GRU converged with a significantly smaller number of iterations when compared to both RNN and LSTM. Except for the GRU, we could not notice any increment in the number of iterations when analyzing the effect of different initialization procedures. This result may indicate that, although using a trained initialization strategy adds adjustable parameters to the model, such modification does not affect the computational burden of the final model.

4.5 Conclusion

In this chapter, we have approached the problem of hard drive failure prediction as a regression task, which is the most straightforward modeling of the health status assessment problem.

In section 4.3 we tested two deep learning models in the task of HDDs failure prediction. We evaluated the performance of LSTM and CNN based architectures compared to a Simple Recurrent Neural Network. The performance was assessed according to three classical prognostics metrics.

Aiming to improve the performance of the CNN model for short RUL predictions we also conducted experiments with a CNN generated with a weighted loss function. Additionally, we created two modified metrics to capture results that are more accurate in low RUL situations.

Our experiments showed that LSTM had the best overall performance, followed by CNN. Both outperformed the SRN model. Also, the modified CNN was able to improve the

predictions of CNN for low RUL situations.

In section 4.4 we assessed the impact of three initialization techniques with three different recurrent neural networks for failure prediction in Hard Disk Drives. Our results showed that both LSTM and GRU outperformed the SRN in all prognostics metrics. The LSTM model had the best overall performance. It is important to point that GRU achieved similar results when compared to LSTM but with faster convergence.

We verified that the initialization procedures had a significant impact in almost all prognostics metrics for the RNNs, being the trained initialization the best approach. Also, the number of iterations was not significantly affected by the use of an initialization procedure.

Open research opportunities in the topics covered in this chapter include the evaluation and proposal of different loss functions that can improve the prognostics metrics discussed in section 4.1. Also, additional network architectures could be explored, besides the ones defined by the parameters employed in the optimization problem discussed in section 4.3.

5 HEALTH DEGREE PREDICTION

In this chapter, we will approach the health assessment problem of hard disk drives failure as a classification task. This brings an advantage when the decision maker can tolerate more coarse-grained information without performance compromise because it lessens the accuracy requirements of the usual regression-based approach. As such, this modeling of the problem turns it into an easier task to tackle.

In section 5.1, we present a solution to the problem approaching it as a standard classification task. In this context, we compare our proposed solution to a competing one and show how it performs better in a scenario where the prediction horizon required increases.

Then, in section 5.2, we improve over this solution, taking into account the specific characteristics of the health assessment problem. These characteristics are the ordinal relationship between the classes and the inherent bias that should favor predictions that indicate a worse health degree, in case of wrong predictions.

Finally, in section 5.3 we provide conclusions and directions for future work taking into account the results of both approaches explored for the health assessment problem.

5.1 Health Degree Prediction with LSTM Networks

In this section, we propose an approach for remaining useful life prediction for HDDs in both long and short range prediction horizons through their categorization in RUL intervals. To explore the long-term temporal relations on SMART data, we propose to use LSTM networks.

The following sections are organized as follows: In section 5.1.1 we present our LSTM based solution for failure prediction of HDDs. In section 5.1.2 we evaluate our proposed solution and compare it to a baseline method.

5.1.1 Proposed Method

The proposed method consists essentially in three steps, namely, RUL Binning, Model Creation, and Failure Prediction. The first is a preprocessing step, that allows for a user of the method to adjust it to their needs. The second presents the model structure and how it is trained. The third explains how to obtain predictions from an already built method.

5.1.1.1 RUL Binning

Similar to the work of (CHAVES *et al.*, 2016) we perform a discretization of the remaining useful life attribute of the HDDs, but with several particularities. Instead of evenly spaced intervals for discretization (e.g. trimesters, months, days), we apply a custom spaced binning of the RUL, allowing user-specified configuration of the method hereby proposed. One can see that a benefit of this approach is to have a more fine-grained control of the prediction. For example, by defining a spacing between the discretization buckets that have smaller values as those buckets become closer to the end of the device's useful life, leads to a gradual increase in the information precision for such scenario.

After performing the RUL binning, we can tackle the problem as a multi-label classification task, instead of a regression problem.

5.1.1.2 Model Creation

As we are interested in the categorization of each SMART data sample from a given HDD time-series, it is straightforward to create a model using an LSTM architecture that produces an output for each input entry from a sequence. The entry, in this case, is a vector containing the SMART attributes for a given day. As we feed those entries successively to this network, it will produce an output for each entry. By taking those multiple results from the network, we can view its output as also a sequence, with the same length as the original time-series that was processed. In the proposed modeling, the input and output correspond, respectively, to the SMART data time-series of a given HDD and their respective RUL bins.

Our model employs two stacked layers of LSTM networks, with standard feed-forward connections between the two layers, and recurrent ones within the same layer. Such multi-layer LSTMs are known to more naturally capture the structure of sequences and to achieve better performance on difficult temporal tasks (HERMANS; SCHRAUWEN, 2013). A depiction of this architecture can be seen in figure 12.

Even though our experiments are performed on a dataset with a considerable amount of data, we apply Dropout (SRIVASTAVA *et al.*, 2014) to the feed-forward connections of both LSTM layers, as a preventive measure.

A fully connected standard neural network, without activation function in its neurons (i.e. linear), is later applied to the output of the last LSTM layer at each timestep (SMART

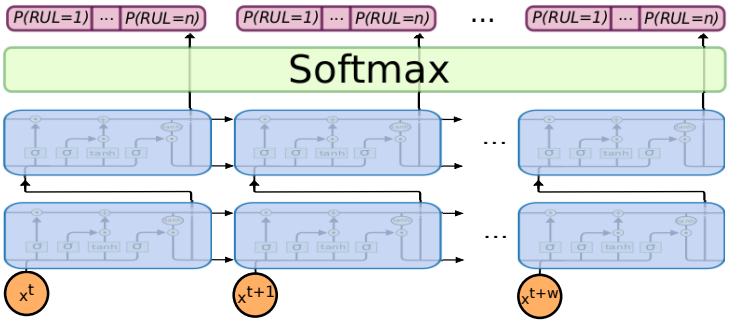

Figure 12 – Architecture of the LSTM network. The two layers of LSTMs are in purple, followed by the softmax layer

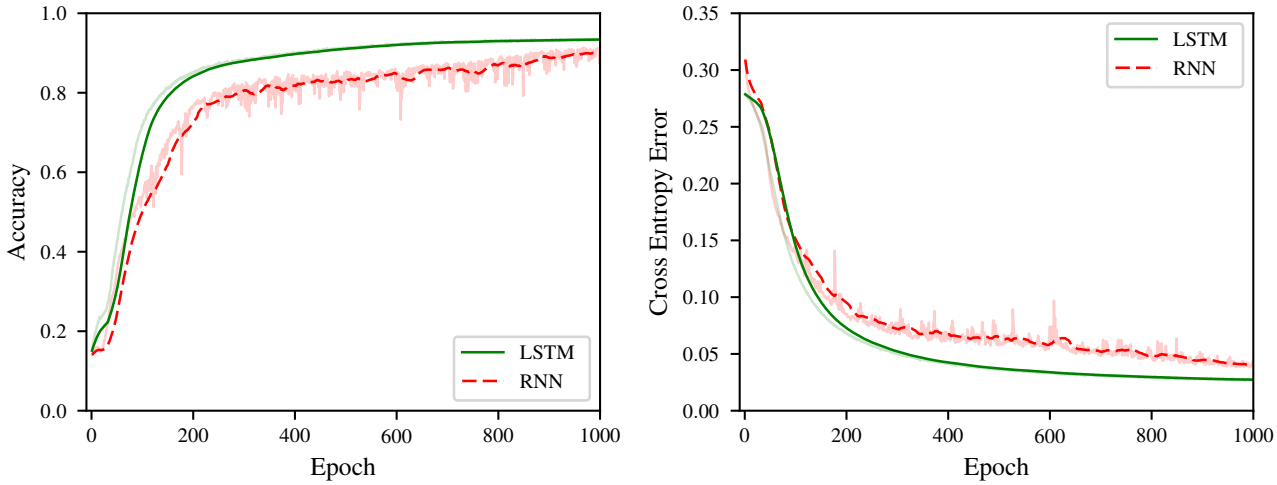

Figure 13 – Training accuracy (left) and loss (right) for 30 days prediction horizon.

data sample). This effectively maps the hidden state vector into a vector whose dimensions match the number of intervals in which the RUL was discretized in the first step of our method.

Afterward, the output (logits) of this layer is then given to a softmax activation function, which produces an estimate of the discrete probability distributions of the RUL bins.

During the training of the model, the cross-entropy objective function (i.e. negative log probability) is employed.

5.1.1.3 Failure Prediction

Finally, in order to perform the Failure Prediction step, the time-series of SMART data collected from a target HDD is given, following the chronological order of the samples, as input to the network built from the proposed architecture, described in the previous subsection. Basically, for each SMART data entry, the predicted RUL bin is the one whose probability is higher in the output of the softmax layer.

5.1.2 Experimental Results

The method proposed in this section was implemented in Python, using Pandas 0.18.1 (MCKINNEY, 2008–) and NumPy 1.12.0 libraries for data preprocessing. In addition, we used TensorFlow 1.0.1 (ABADI *et al.*, 2016) for implementing Neural Networks with equations running on GPU. In order to compare the different methods, we used metrics contained in scikit-learn 0.18.1 (PEDREGOSA *et al.*, 2011) library.

5.1.2.1 Dataset

For purposes of evaluating the method proposed, we use the data provided by Backblaze Company. This dataset is comprised of the daily observation of 92,348 HDDs during the period from 04/10/2013 to 12/31/2016. These observations contain information regarding the serial number, model, capacity, fault, and 90 SMART attributes of each device. According to the Backblaze Company, a device is labeled as faulty if it stops working (does not turn on or does not receive commands), or if SMART self-test fails for attributes 5, 187, 188, 197, or 198. Most HDD models do not report all SMART attributes. In this case, the values not reported are left blank. In addition, different manufacturers and device models may report different attributes.

To avoid a potential overfitting, we chose to perform the tests with the Seagate ST4000DM000 model whose data are most plentiful. This model has 36,555 observed disks, of which 1729 have failures. Of these, 32 were removed because their observation was interrupted without a label indicating a failure or by having submitted observations after being labeled as damaged. Thus, we used in fact for the test of the compared models a set of observations on 1,697 instances.

For this experiment, use all the SMART attributes collected by Backblaze for the chosen model. The selected attributes were: 1, 3, 4, 5, 7, 9, 10, 12, 183, 184, 187, 188, 189, 190, 191, 192, 193, 194, 197, 198, 199, 240, 241 and 242, each having its raw and normalized values included.

5.1.2.2 Performance Evaluation

To verify the performance of the LSTM model we performed short-term and long-term failure predictions. In both tests, the task is to classify each sample in one of the six RUL intervals. However, the RUL intervals are defined within one month before failure for the short-

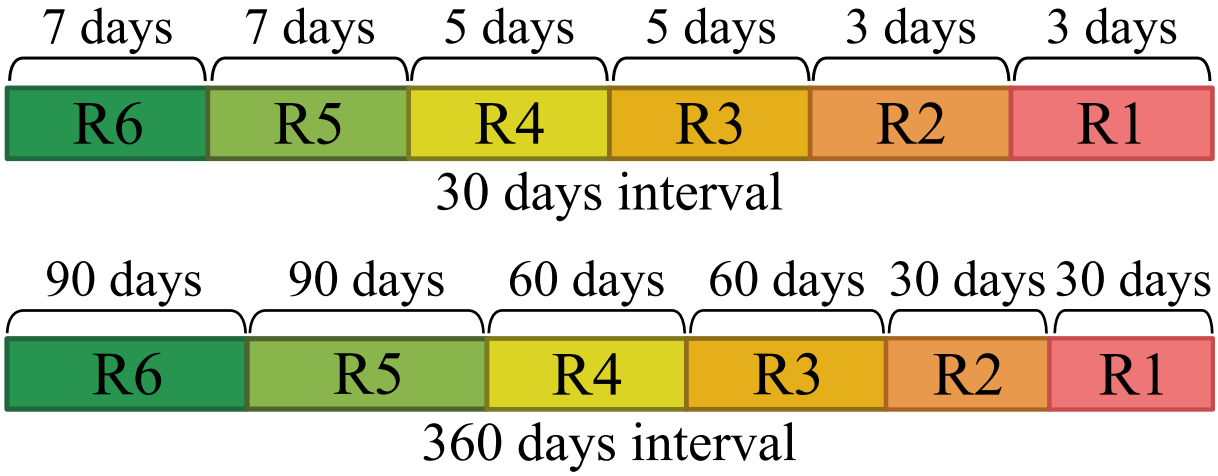

Figure 14 – Interval settings for the decreasing RUL used in the experiments.

term prediction and one year for the long-term prediction, as can be seen in Figure 14. The proposed model was compared to an Elman RNN and a Random Forest classifier. Both the Elman and LSTM recurrent networks implemented for the experiments follow the multilayer architecture described in section 5.1.1. To define the unrolling parameter of the truncated BPTT applied during the training of the networks, a grid search was performed in the intervals $[2, 30]$ and $[2, 360]$ for short and long-term prediction scenarios respectively. The parameters found for the BPTT unrolling were 6, for the RNN, in both scenarios, and, for the LSTM, 30 days for short and 360 days for long-term. The internal memory was defined to have size 10 and 64 for short and long-term predictions, respectively, in both networks. For the experiments of Random Forest classifier, we used 200 estimators (trees) with no depth limit. In addition, we use bootstrap, which optimizes the construction of the forest, reducing the dependence between its trees. The other parameters were set as the standards of the sklearn library.

Table 16 shows the performance of each method for the long and short-term prediction tasks. Since the datasets are unbalanced, we used Micro and Macro F-measures as performance indicators. The Micro-averaged F-score basically consists of calculating the F-score taking the sum of the true positives, false positives and false negatives for all classes. The Macro-averaged version simply performs an average over the individual F-scores calculated for each class. Therefore, the Micro F1 tends to bias the metric toward the most populated classes, whereas the Macro treats all classes equally (SOKOLOVA; LAPALME, 2009).

As can be noticed, for the short-term prediction task, the recurrent methods achieved similar results. However, for the long-term predictions, LSTM achieved the best results, followed by the RNN. The best performance of the recurrent methods is expected since both use historical information to perform future predictions. The significant performance gap between

Classification performance

	30 Days		360 Days	
Model	Micro F1	Macro F1	Micro F1	Macro F1
LSTM	0.9840	0.9840	0.7169	0.6861
RNN	0.9819	0.9818	0.3044	0.2547
RF	0.2513	0.1660	0.2598	0.2389

Table 7 – Performance of the classifiers under different prediction horizon settings.

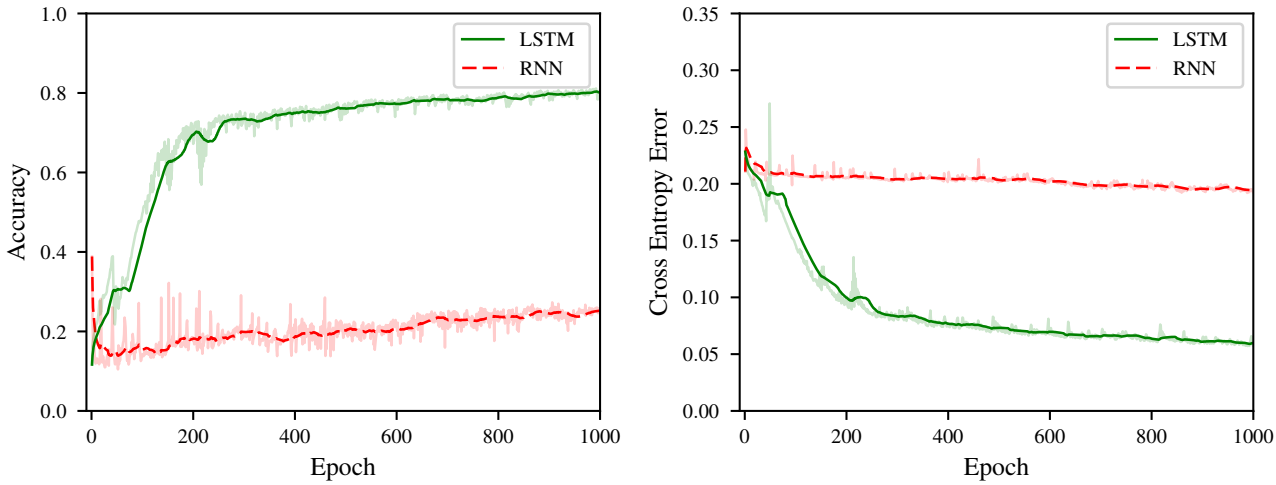

Figure 15 – Training accuracy (left) and loss (right) for 360 days prediction horizon.

		Predicted					
		R_1	R_2	R_3	R_4	R_5	R_6
Actual	R_1	**1985**	10	14	14	7	12
	R_2	14	**1953**	4	11	9	12
	R_3	7	0	**1392**	3	3	7
	R_4	2	0	0	**1392**	3	1
	R_5	0	0	0	0	**834**	3
	R_6	0	0	0	0	0	**834**

Table 8 – Confusion Matrix LSTM 30 Days.

LSTM and RNN in long-term predictions enforces our initial hypothesis that gradient vanishing/exploding problems may be observed in Elman networks.

A more detailed analysis regarding the performances of RNN and LSTM can be seen in the confusion matrices in Tables 8, 9, 10 and 11. It is interesting to perceive that for the long-term prediction (360 days), in addition to achieving the best Micro and Macro F1 scores, the LSTM model tends to produce errors that are concentrated in classes near the correct one. Since this is not observed for the RNN we can state that the LSTM classification errors can be seen as less severe than RNN errors.

		Predicted					
		R_1	**R_2**	**R_3**	**R_4**	**R_5**	**R_6**
Actual	**R_1**	**1972**	17	18	14	7	14
	R_2	12	**1953**	5	11	9	13
	R_3	3	2	**1392**	3	3	9
	R_4	0	0	2	**1391**	4	1
	R_5	0	3	0	0	**831**	3
	R_6	0	1	0	0	0	**833**

Table 9 – Confusion Matrix RNN 30 Days.

		Predicted					
		R_1	**R_2**	**R_3**	**R_4**	**R_5**	**R_6**
Actual	**R_1**	**5397**	1800	875	891	954	729
	R_2	691	**5248**	1552	906	1103	643
	R_3	752	612	**11415**	2732	1964	1543
	R_4	403	279	560	**11897**	2652	1789
	R_5	260	235	144	552	**18914**	2919
	R_6	129	17	159	149	275	**18736**

Table 10 – Confusion Matrix LSTM 360 Days.

		Predicted					
		R_1	**R_2**	**R_3**	**R_4**	**R_5**	**R_6**
Actual	**R_1**	**2935**	1468	1206	983	2503	1551
	R_2	2047	**1330**	1201	1122	2656	1787
	R_3	2932	2101	**2106**	2155	5647	4077
	R_4	1762	1329	1495	**1911**	5833	5250
	R_5	1416	1034	1180	1803	**7225**	10366
	R_6	579	348	329	410	2896	**14903**

Table 11 – Confusion Matrix RNN 360 Days.

Another illustration of the difference between LSTM and RNN can be seen in Figures 13 and 15. These figures show the training accuracy and loss function evaluations for short and long-term predictions respectively. For the short-term prediction, we can see that the methods exhibit similar curves indicating that both models learn along the iterations. However, it is noticeable that the RNN can not decrease the training error or improve the performance on the test set for the long-term prediction task.

5.2 Asymmetric Ordinal Health Degree Prediction

In this section, we propose an improvement over the method discussed in section 5.1 that explores particularities in the task of remaining useful life prediction for HDDs. More specifically, the method discussed in this section takes into account two important factors for this task: the ordinal nature between the health degrees and also a bias towards mispredictions that indicate worse health levels. To do so, we propose an encoding scheme for the health degrees that takes into account these two aspects. We also assess this method in a real-world dataset, collected by a storage cloud service provider.

The following sections are organized as follows: In section 5.2.1 we present our class encoding scheme for ordinal biased classification tasks. Then, in section 5.2.6 we briefly discuss about competing encoding techniques for the ordinal classification task that will be used as baseline. Finally, in section 5.2.7 we evaluate our proposed solution in the problem of HDD health degree prediction and compare it to four baseline solutions, three of them designed for ordinal tasks. We evaluate the method applied to two different RNN architectures, LSTMs and GRUs.

5.2.1 Proposed Method

As we are targeting the problem of health assessment of hard disk drives it is straightforward to interpret each health level as a separate class. This naturally leads to the modeling of this problem as a classification task. In addition to that, since there is a clear ordering between the different health classes that a device can be labeled, the problem becomes an ordinal classification task. Moreover, misclassifications that cause a device to be predicted in a better health state, i.e. predicted as healthier, than its true state, could obviously lead to more severe consequences (e.g. data loss, service outage). So, a mechanism to prevent such scenarios must be employed, such as penalizing misclassifications differently depending on the positioning of the prediction relative to the ground truth.

In order to solve this asymmetric ordinal classification problem, a novel method is proposed. This new method encompasses two changes to the usual architectures of neural networks employed on standard classification problems. One of them is to define an encoding on the labeling of classes that could lead to an improved ordinal classification model, taking into account the asymmetry towards lower health levels. This also implies in employing a more

adequate loss function for optimizing the parameters of the network for such encoding. The other change is to define an additional static layer to the neural network, capable of decoding the predicted encodings to their corresponding classes (i.e. health level).

Figure 16 summarizes the proposed method whose steps will be described in the following sections.

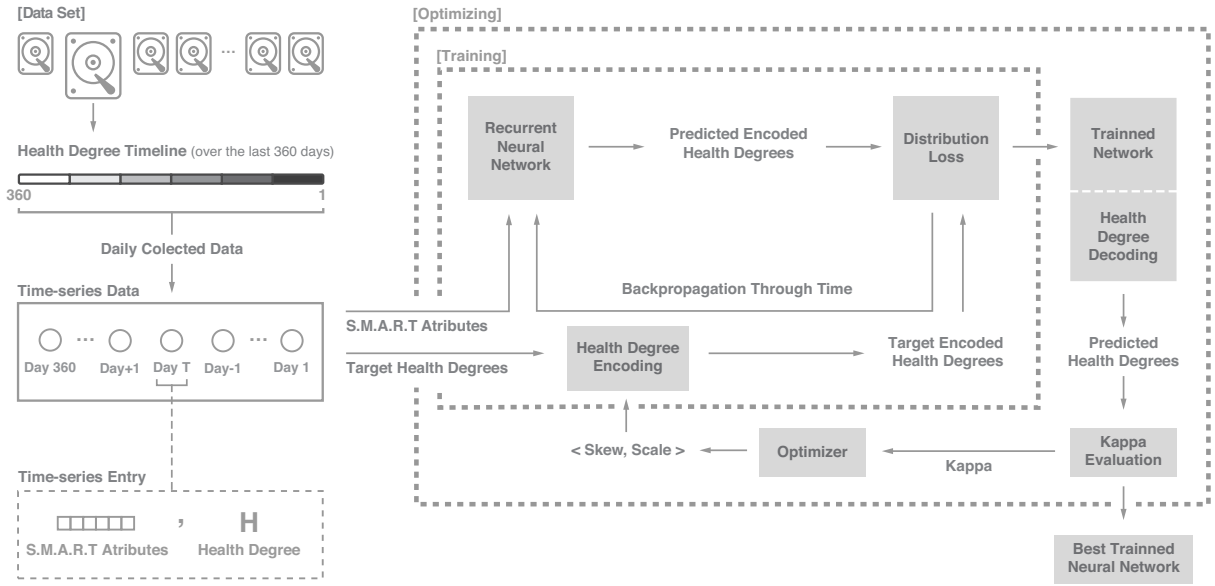

Figure 16 – Depiction of the overall process performed in order to produce the best trained neural network for a given hard disk drive S.M.A.R.T. time-series dataset.

5.2.2 Custom Encoding

The usual encoding applied in classification tasks is the 1-of-m (also known as one-hot) encoding. This consists in producing for each class a target that is a vector of the form $(0,...,0,1,0,...,0)$, with only an element set to 1 and all others as 0. With this formulation, a model is expected to produce probabilities of a data point to belong to different classes. The main issue with this scheme for ordinal classification is that no ordering relation between the classes is obtainable. For ordinal classification problems, it is usual to define the encoding to be an integer encoding (i.e. to map each class to an integer, respecting their ordering). Taking such encoding, it is possible to measure a distance between two target values, allowing for different penalization depending on it. The work (BECKHAM; PAL, 2016) can be seen as a variation of this technique. (CHENG *et al.*, 2008) propose a different encoding for ordinal classification that labels any data point that belongs to a class K, to belong to all lower classes as well. This allows for the models to quantify for each prediction produced under such encoding

by how many classes it is misplaced. Thus, a clear ordering between classes can be modeled. For solving the asymmetric aspect, not only the distance between classes must be taken into account, but also the direction of such distance. We aim to propose an encoding that handles both issues.

Like the one-hot encoding approach, ours also defines a probability distribution over the categories. A significant difference, though, is that we model the encoding to guarantee that other classes also get probability values other than zero. The fundamental idea is to define class probabilities such that smaller values mean undesired result. For both lower and higher classes, we ensure that their probability values are set in a decremental fashion, based on their distance to the ground truth. In order to enforce a higher penalization for predictions in the wrong direction, a skew is applied to this distribution, where it is expected to have overall higher probabilities concentrated below the ground truth if compared to classes above it. In other words, we enforce a penalization for mispredictions lower than the ground truth, and even greater penalizations for mispredictions higher than it, all based on their distance to the true label. Thus, if a data point belongs to class C, we define a function f, that maps each position in the target vector t, such that $t_k = f(k)$, with $f(k) > f(k-1)$, for $k \leq C$ and $f(k) > f(k+1)$ for $k > C$, with $\sum_{k=0}^{C} f(k) > \sum_{k=C+1}^{n} f(k)$.

A probability function whose characteristics can be explored in order to satisfy such requirements is the asymmetric Laplace distribution (KOZUBOWSKI; PODGORSKI, 2000). Thus, as a starting point for defining the function f, the asymmetric Laplace probability mass function was employed, with a later procedure for normalizing the resulting target vector to ensure the values sum one. This probability function is governed by three parameters. Namely, the location m, which moves the center of the distribution, the skew μ, that measures the asymmetry, and the scale λ, that defines how sharp the distribution is. A natural choice for the location is the classes ground truth values themselves. The other two parameters then, become hyper-parameters of the encoding, which must be tailored for the problem. A constraint that must be applied to these parameters, though, is that the original class values fed to the function f must be recoverable, that is, the encoding t_x produced by $f(x)$ for a data point x, when applying a decoding function $q(t_x)$, must produce the original x class. Equation 5.1 shows the probability density function p explained above and Figure 17 shows the probability density function in three cases: no skew, positive skew and negative skew.

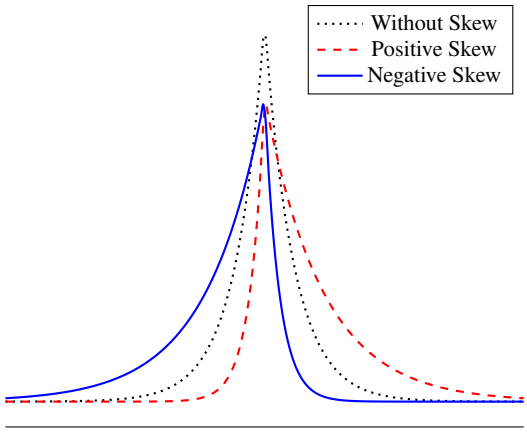

Figure 17 – Probability density function of the asymmetric Laplace distribution in three cases: no asymmetry, positive asymmetry and negative asymmetry.

$$p(x;m,\lambda,\mu) = \left(\frac{\lambda}{\mu+1/\mu}\right) \cdot \begin{cases} \exp((\lambda/\mu)(x-m)) & \text{if } x < m \\ \exp(-\lambda\mu(x-m)) & \text{if } x \geqslant m \end{cases} \tag{5.1}$$

5.2.3 Decoding

For decoding target values produced by the encoding procedure described above, we add an additional layer past the output layer. This layer performs an operation whose results are in the range $[0, k-1]$, with k being the number of classes in the problem. Since the output layer of our network produces probability distributions over all classes, we defined that this operation performed is the probability-weighted average over all class values, which is precisely the definition of the expected value of a discrete random variable. This operation can be written as

$$\hat{y_x} = \sum_{n=0}^{k-1} n\hat{t_x}[n] \tag{5.2}$$

where k is the number of classes, $\hat{t_x}$ is the output of the network for the data-point x and $\hat{y_x}$ is the final output of the network after decoding.

5.2.4 Cost Function

To find the parameters of the network an adequate loss function must be defined. Since the network produces as an intermediate step a discrete probability distribution over the

classes and the target vector is also a distribution, a function that measures the distance of such distributions can be employed. For this, the Jensen-Shannon divergence (JSD), which is a dissimilarity measure between probability distributions, based on the Kullback-Leibler divergence (KLD), was used. Clear advantages of this measure over KLD as a loss function are its boundness, smoothness, and symmetry. It is defined by

$$JSD(P \parallel Q) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M) \tag{5.3}$$

where M is

$$M = \frac{1}{2}(P+Q) \tag{5.4}$$

and KLD in turn, for the discrete case, is calculated as follows

$$D_{\mathrm{KL}}(P\|Q) = -\sum_i P(i) \log \frac{Q(i)}{P(i)}, \tag{5.5}$$

5.2.5 Finding the Encoding Parameters

As aforementioned, the encoding is defined by a distribution mass function extracted from an asymmetric Laplace probability density function. The two parameters are the skew μ, that measures the asymmetry, and the scale λ, that defines how sharp the distribution is. In addition, we need to ensure that the parameters will lead to predictions that respect class ordering and that will favor lower class values. For doing so, we need a metric that can be tuned specifically for these two requirements.

The Cohen's Kappa (COHEN, 1960) is a metric which compares the agreement of two categorical raters that can be used to measure how close the output of a trained classifier is to the ground truth output. However, the Cohen's Kappa treats disagreements equally, which is unwanted for the problem of ordinal classification. Therefore, the weighted Cohen's kappa (COHEN, 1968), that allows for custom importance to the disagreements, is applied to this problem. The weighted Cohen's kappa κ, as shown in (BECKHAM; PAL, 2016), is defined as follows:

$$\kappa_w = 1 - \frac{\sum_{i,j} W_{ij} O_{ij}}{\sum_{i,j} W_{ij} E_{ij}}$$

where O is the $k \times k$ confusion matrix of the output of the trained neural network and E is a $k \times k$ matrix related with the expected values. The matrix E is defined as the outer product between

the vector of column sums of the expected one hot encoded class and the vector of columns sums of the predictions, and then E is normalized to have the same total sum as O. Also, W is a $k \times k$ matrix of weights that is the cost of disagreement between different misclassifications. The matrix W is defined by us and it is important to notice that, for $i = j$, W_{ij} must be 0, which means we must not penalize the agreements between the two matrices O and E. If $\kappa_w = 0$ then the trained classifier does no better than a random choice classifier. If $\kappa_w = 1$ then the classifier agrees completely with the expected values.

Since we can define the weight matrix W, we can use costs that can naturally penalize more distant misclassifications by defining higher costs as they are more distant to the diagonal. One possibility is using the symmetric κ_w matrix that determines the quadratic weighted kappa, defined as the following:

$$W_{ij} = (i-j)^2, \text{ for } i, j \in \{0, 1 \ldots k-1\}$$

In a similar way, we can define weights that ensure the aforementioned asymmetrical aspect of the problem, establishing lower cost values below the diagonal. We also defined the following asymmetric κ_w matrix:

$$W_{ij} = \begin{cases} (i-j)^2 & \text{if } i \leqslant j \\ (i-j)^2/\alpha & \text{if } i > j \end{cases}, \text{ for } i, j \in \{0, 1 \ldots k-1\}$$

where α is an asymmetry factor, that the higher its value is, bigger is the impact of misclassifications in the undesired direction.

Therefore, we can model the search of these parameters as an optimization problem, as shown in Equation 5.6.

$$\max_{\lambda,\mu} \quad g(\lambda, \mu) \tag{5.6a}$$

$$\text{subject to} \quad |\hat{y}_k - k| - 0.5 < 0, \quad k = 0, \ldots, K-1. \tag{5.6b}$$

where g returns the kappa κ_w for a fixed asymmetric kappa matrix W as described above, for a fixed α, and for the matrix O obtained from the predictions. The predictions, in turn, are produced by a classifier trained with the custom encoding generated by the asymmetric Laplace probability mass function defined by the parameters λ and μ.

Also, in the constraint presented in Equation 5.6b, $\hat{y}_k$ is the decoded value of class k of the encoding governed by the parameters λ and μ, as shown in Equation 5.2. This is a recoverability constraint that allows us to recover the original class k, ensuring that the expected value of the discrete distribution is not far from the integer value k.

To solve this optimization problem, several methods can be applied, such as meta-heuristics and Bayesian optimization. As this optimization problem involves a full training of a neural network for its evaluation, which is a time-consuming task, and since we are in a domain where our knowledge about the impacts of the parameters on the Kappa are scarce, Bayesian optimization is particularly suitable.

5.2.6 Baseline Encoding Schemes for Ordinal Classification

In the context of the ordinal classification problem, Cheng (CHENG *et al.*, 2008) proposed a different way of representing the class labels where, if a data point belongs to a category k, it also belongs to all the lower categories $\{0, \ldots, k-1\}$. They achieved good results compared to a standard neural network method in datasets with the ordinal aspect.

Also, Beckham and Pal (BECKHAM; PAL, 2016) explored the possibility of the addition of a new layer after the final softmax layer to deal with the ordinal aspect of the problem. They added a layer with output size one and with fixed values (referred as vector a) and employed the squared error loss. Also, they tried to learn the values of the vector a. According to the authors, their reformulation can achieve better values in the quadratic weighted kappa metric, arguing that in fact it directly optimizes this metrics, and produces competitive results when compared to other ordinal classification techniques.

5.2.7 Experimental Results

All methods were implemented in Python 3. Additional libraries were used, namely Numpy 1.14.2 (OLIPHANT, 2006), Tensorflow 1.7.0 (ABADI *et al.*, 2016) for the creation and execution of the models, Pandas 0.22.0 (MCKINNEY, 2008–) and scikit-learn 0.18.1 (PEDREGOSA *et al.*, 2011) for data preprocessing. Also, GPyOpt 1.2.1 (The GPyOpt authors,) was employed to perform Bayesian Optimization.

5.2.7.1 Dataset

All experiments were performed on a public dataset provided by the Backblaze Company (BACKBLAZE, 2016) to evaluate the performance of the proposed and baseline methods. This dataset contains daily SMART observations of thousands of HDDs of different models from various manufacturer between April 2013 and December 2016. These observations were collected until either the disk stops working (e.g. won't power up, does not respond to commands anymore), or until it has shown signs that it will stop soon (e.g. SMART statistics of critical attributes indicate a failure). Then the disk is marked as failed in the dataset.

It was assumed that HDDs of a single model from the same manufacturer have similar degradation over time. Therefore, to perform the experiments, it was selected only disks of the model Seagate ST4000DM000, which is the one with the biggest number of samples in the dataset. From this specific model, there are 36,555 observed disks, of which only 1,729 have failed. Also, 32 were excluded because of inconsistencies. Specifically, either their observations were interrupted without a label indicating a failure or they had additional observations submitted after being flagged as failed. In the end, 1,697 instances were used. From the final set of disks, it was used a) 997 disks for training, b) 333 disks for validation, and c) 367 disks for testing.

Due to technicalities, only a limited number of the SMART attributes collected in the dataset for this manufacturer model were used. The observed raw attributes returned by the feature selection process performed in (BOTEZATU *et al.*, 2016) have been chosen. These features are described in Table 12.

5.2.7.2 Results and Discussion

To verify the performance of the proposed method applied to the health assessment problem in hard disk drives, two types of deep recurrent neural networks namely, LSTM and GRU networks, were used. The architecture of both networks was the same presented in (LIMA *et al.*, 2017), which defines two recurrent neural network layers with state vector size of 64 and a single fully connected single perceptron as the output layer. The network is represented in Figure 12.

We compared our method with other four approaches: a) standard classification strategy with one-hot encoding and cross-entropy as the loss function, b) the work of (BECK-

Attribute ID	Attribute Name
SMART 1	Read Error Rate
SMART 5	Reallocated Sectors Count
SMART 7	Seek Error Rate
SMART 184	End-to-End error
SMART 187	Reported Uncorrectable Errors
SMART 188	Command Timeout
SMART 189	High Fly Writes
SMART 190	Temperature Difference
SMART 193	Load Cycle Count
SMART 194	Temperature
SMART 197	Current Pending Sector Count
SMART 198	Uncorrectable Sector Count
SMART 240	Head Flying Hours
SMART 241	Total LBAs Written
SMART 242	Total LBAs Read

Table 12 – SMART attributes used in the classification task.

HAM; PAL, 2016), with the squared-error reformulation and a fixed vector a, c) the work of (BECKHAM; PAL, 2016), with the squared-error reformulation and a trainable vector a, and d) the work of (CHENG *et al.*, 2008), with the multi-threshold encoding approach and mean squared error as the loss function.

As aforementioned in section 5.2.1, there are two hyperparameters in the proposed custom encoding that comes from the Laplace probability function: the skew and the scale of the distribution. In order to find these parameters, since the training of the recurrent neural network classifier is a time-consuming process, a Bayesian Optimization with restrictions was performed over the already defined problem in Equation 5.6. The goal of the optimization was to maximize the weighted Cohen's Kappa with the asymmetric W matrix, whose asymmetry factor parameter was defined as $\alpha = 4$ showed in equation 5.7.

$$\kappa_{asymw} = \begin{bmatrix} 0 & 1 & 4 & 9 & 16 & 25 \\ 0.25 & 0 & 1 & 4 & 9 & 16 \\ 1.00 & 0.25 & 0 & 1 & 4 & 9 \\ 2.25 & 1.00 & 0.25 & 0 & 1 & 4 \\ 4.00 & 2.25 & 1.00 & 0.25 & 0 & 1 \\ 6.25 & 4.00 & 2.25 & 1.00 & 0.25 & 0 \end{bmatrix} \tag{5.7}$$

The range of possible values of the skew and scale was within the interval $[0, 100]$. The optimization was performed once for each type of network (LSTM and GRU) and the

results of the best hyperparameters found by the process can be seen in Table 15. Also, Figures 18 and 19 shows the encoding created by these parameters for the LSTM and GRU networks, respectively. In those figures, the bars represent the probability, or contribution, of each class in the encoding according to the distribution.

To compare the performance of the methods, three metrics were considered: i) accuracy of the classification, ii) the symmetric quadratic weighted kappa with the weight matrix, W, shown in Equation 5.8, and iii) the asymmetric weighted kappa with the weight matrix W being the same as the one used to execute the optimization, shown in Equation 5.7. Table 16 presents the average metric values of 3 repetitions for all models.

$$\kappa_{symw} = \begin{bmatrix} 0 & 1 & 4 & 9 & 16 & 25 \\ 1 & 0 & 1 & 4 & 9 & 16 \\ 4 & 1 & 0 & 1 & 4 & 9 \\ 9 & 4 & 1 & 0 & 1 & 4 \\ 16 & 9 & 4 & 1 & 0 & 1 \\ 25 & 16 & 9 & 4 & 1 & 0 \end{bmatrix} \tag{5.8}$$

As can be observed, LSTM had the best overall performance when considering all metrics even though the performance gap between GRU and LSTM was small, except for the accuracy, which showed approximately 6% difference. It is noticeable that for the regular methods, both GRU and LSTM, their accuracy exceeded all other methods. This is expected, since they were designed to optimize for accuracy, not taking into account both the class imbalance and ordering issues.

Now for the Symmetric Kappa metric, the method based on the encoding defined by (CHENG *et al.*, 2008) showed the best results, for both recurrent network models. This comes as a surprise, since (BECKHAM; PAL, 2016) based solutions were designed to directly optimize the Quadratic Weighted kappa metric, which is precisely the Symmetric Kappa metric. Our method outperformed all others under the Asymmetric Weighted Kappa metric.

A more detailed analysis regarding the performances of the methods can be seen in the confusion matrices in Tables 17, 18, 19, 20, 21, 22, 23, 24, 25 and 26. As expected our method led to the smallest concentration of predictions above the diagonal of the matrices, which indicates smaller errors suggesting that a device is healthier than it really is. This can be noticed for both networks, LSTMs and GRUs.

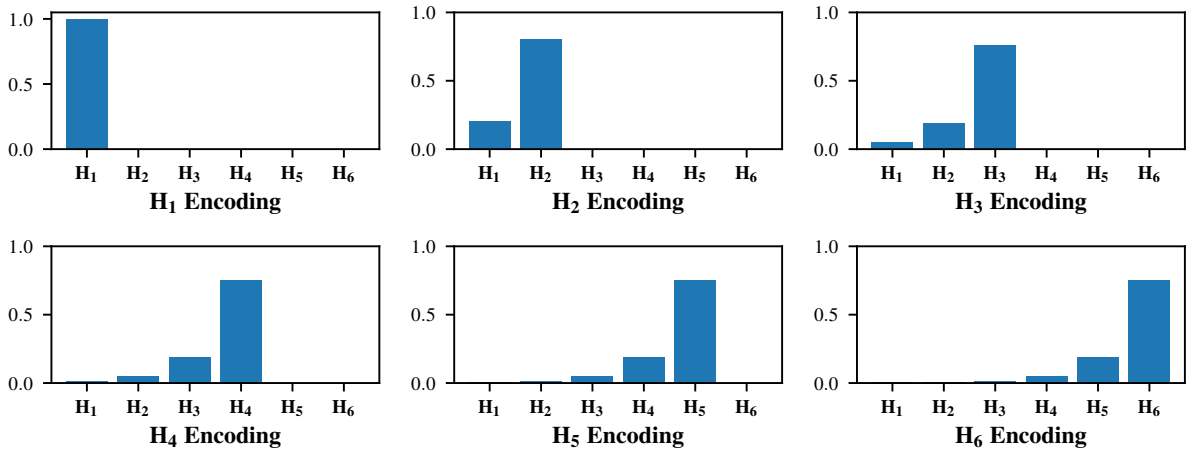

Figure 18 – Best encoding found by the method for the LSTM model. The bars represent the probabilities for each health degree (class).

Health Degree	H_1	H_2	H_3	H_4	H_5	H_6
H_1 Encoding	1.00	0.00	0.00	0.00	0.00	0.00
H_2 Encoding	0.20	0.80	0.00	0.00	0.00	0.00
H_3 Encoding	0.05	0.19	0.76	0.00	0.00	0.00
H_4 Encoding	0.01	0.05	0.19	0.75	0.00	0.00
H_5 Encoding	0.00	0.01	0.05	0.19	0.75	0.00
H_6 Encoding	0.00	0.00	0.01	0.05	0.19	0.75

Table 13 – Numeric values for each health degree of the best encoding found by the method for the LSTM model.

Health Degree	H_1	H_2	H_3	H_4	H_5	H_6
H_1 Encoding	1.00	0.00	0.00	0.00	0.00	0.00
H_2 Encoding	0.20	0.80	0.00	0.00	0.00	0.00
H_3 Encoding	0.05	0.19	0.76	0.00	0.00	0.00
H_4 Encoding	0.01	0.05	0.19	0.75	0.00	0.00
H_5 Encoding	0.00	0.01	0.05	0.19	0.75	0.00
H_6 Encoding	0.00	0.00	0.01	0.05	0.19	0.75

Table 14 – Numeric values for each health degree of the best encoding found by the method for the GRU model.

Another interesting aspect that can be observed in the confusion matrices produced by our method is that there is an improvement in the accuracy of classification for the classes H_1 and H_2 if compared to the other methods. These classes can be considered as the most critical ones since they are the nearest classes to the devices' end-of-life. So, our method also produced an improved classification accuracy for those critical classes.

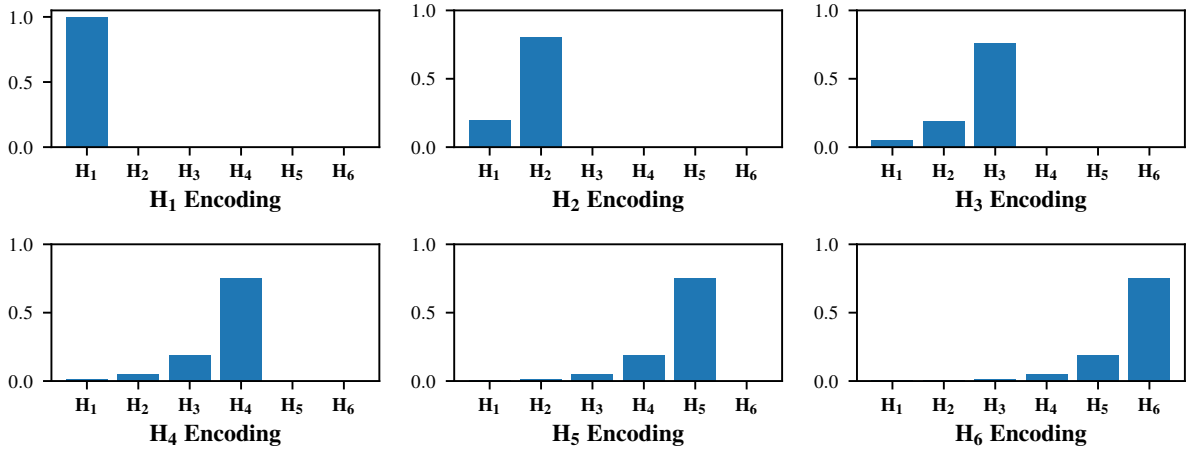

Figure 19 – Best encoding found by the method for the GRU model. The bars represent the probabilities for each health degree (class).

Network	**Skew**	**Scale**
LSTM	72.04848	100.0
GRU	61.240208	81.48958

Table 15 – Bayesian Optimization results for the two types of network

		Results		
Network	Method	Accuracy	Sym. Kappa	Asym. Kappa
LSTM	Regular	**76.0799%**	0.8310	0.8154
	Ord 1	68.9132%	0.8566	0.8775
	Ord 1 train	69.7843%	0.8577	0.8728
	Ord 2	73.3878%	**0.8596**	0.8640
	Ours	67.8942%	0.8568	**0.8896**
GRU	Regular	**70.7535%**	0.8210	0.8318
	Ord 1	59.5668%	0.8394	0.8641
	Ord 1 train	64.7180%	0.8440	0.8648
	Ord 2	69.2230%	**0.8543**	0.8697
	Ours	67.5716%	0.8401	**0.8800**

Table 16 – Performance of the methods under different classification metrics.

5.3 Conclusion

In this chapter we have approached the problem of hard drive failure prediction as a classification task, also termed as health status assessment problem.

In section 5.1 we proposed a deep learning based solution which employed LSTMs as a building block for a neural network architecture. This model achieved similar results if compared to the method of (XU *et al.*, 2016) in the short-term prediction task (i.e. one-month interval). In the long-term prediction (i.e. twelve months interval) the proposed model outperformed the baseline in all metrics. Also, the method produced less severe (or near) errors, that

		Predicted					
		H_1	**H_2**	**H_3**	**H_4**	**H_5**	**H_6**
Actual	**H_1**	**6315**	594	1224	678	1459	376
	H_2	435	**5820**	1423	717	1347	401
	H_3	217	177	**13432**	1263	3049	880
	H_4	84	0	1220	**12329**	3161	786
	H_5	49	0	466	520	**20535**	1454
	H_6	235	0	0	0	1827	**17403**

Table 17 – Confusion matrix of the regular LSTMs.

		Predicted					
		H_1	**H_2**	**H_3**	**H_4**	**H_5**	**H_6**
Actual	**H_1**	**5307**	2065	1476	1518	189	81
	H_2	177	**6189**	2148	1402	143	84
	H_3	69	2106	**13308**	2788	608	139
	H_4	38	244	3415	**13082**	698	103
	H_5	0	11	1833	4828	**16068**	284
	H_6	0	0	24	2282	2794	**14365**

Table 18 – Confusion matrix of the LSTMs with (BECKHAM; PAL, 2016) method.

		Predicted					
		H_1	**H_2**	**H_3**	**H_4**	**H_5**	**H_6**
Actual	**H_1**	**5461**	1711	1422	1804	167	81
	H_2	249	**5881**	1988	1797	113	115
	H_3	150	1421	**13144**	3724	446	133
	H_4	120	95	2598	**13885**	741	141
	H_5	33	44	854	5319	**16440**	334
	H_6	0	36	99	2206	1586	**15538**

Table 19 – Confusion matrix of the LSTMs (BECKHAM; PAL, 2016) with trainable vector method.

		Predicted					
		H_1	**H_2**	**H_3**	**H_4**	**H_5**	**H_6**
Actual	**H_1**	**6225**	1033	1402	1271	577	138
	H_2	493	**6031**	1584	1318	572	145
	H_3	209	1227	**13111**	2950	1137	384
	H_4	60	205	2180	**13570**	1122	443
	H_5	11	5	1002	3520	**17629**	857
	H_6	0	0	19	1601	1085	**16760**

Table 20 – Confusion matrix of the LSTMs with (CHENG *et al.*, 2008) method.

		Predicted					
		H_1	**H_2**	**H_3**	**H_4**	**H_5**	**H_6**
Actual	**H_1**	**6592**	1142	1624	1049	164	75
	H_2	902	**6347**	1720	989	112	73
	H_3	130	3406	**13131**	1803	418	130
	H_4	6	791	4320	**11858**	550	55
	H_5	1	36	2859	4309	**15606**	213
	H_6	0	0	260	1997	2954	**14254**

Table 21 – Confusion matrix of the LSTMs with ours method.

		Predicted					
		H_1	**H_2**	**H_3**	**H_4**	**H_5**	**H_6**
Actual	**H_1**	**5562**	1428	2688	146	524	298
	H_2	795	**4457**	3831	190	574	296
	H_3	352	59	**16350**	724	898	635
	H_4	95	0	5145	**10446**	1236	658
	H_5	70	0	4035	755	**17242**	922
	H_6	252	0	1141	0	1502	**16570**

Table 22 – Confusion matrix of the regular GRUs.

		Predicted					
		H_1	**H_2**	**H_3**	**H_4**	**H_5**	**H_6**
Actual	**H_1**	**3672**	3662	1777	1290	182	63
	H_2	192	**4112**	4430	1179	162	68
	H_3	60	1776	**12657**	3866	553	106
	H_4	30	187	4241	**9733**	3319	70
	H_5	0	0	2606	4594	**15187**	637
	H_6	0	0	69	2415	3969	**13012**

Table 23 – Confusion matrix of the GRUs with (BECKHAM; PAL, 2016) method.

		Predicted					
		H_1	**H_2**	**H_3**	**H_4**	**H_5**	**H_6**
Actual	**H_1**	**5041**	2260	1392	1761	128	64
	H_2	263	**5943**	1941	1826	109	61
	H_3	80	1791	**11993**	4654	373	127
	H_4	46	177	2608	**14025**	597	127
	H_5	0	127	729	8053	**13830**	285
	H_6	0	11	43	2577	4026	**12808**

Table 24 – Confusion matrix of the GRUs (BECKHAM; PAL, 2016) with trainable vector method.

		Predicted					
		H_1	**H_2**	**H_3**	**H_4**	**H_5**	**H_6**
Actual	**H_1**	**5111**	2151	1469	1527	272	116
	H_2	679	**5052**	2437	1617	264	94
	H_3	233	1522	**12305**	4102	688	168
	H_4	65	292	2215	**13579**	1196	233
	H_5	15	22	1339	4562	**16607**	479
	H_6	8	29	110	1962	2032	**15324**

Table 25 – Confusion matrix of the GRU (CHENG *et al.*, 2008) method.

		Predicted					
		H_1	**H_2**	**H_3**	**H_4**	**H_5**	**H_6**
Actual	**H_1**	**6474**	1801	1066	897	318	90
	H_2	833	**6857**	1201	918	266	68
	H_3	565	3372	**12851**	1298	826	106
	H_4	243	1457	3372	**11626**	827	55
	H_5	33	686	2460	4029	**15611**	205
	H_6	0	0	537	1194	4414	**13320**

Table 26 – Confusion matrix of the GRUs with ours method.

is, misclassifications were closer to the ground truth if compared to the baseline.

In section 5.2 we proposed a novel method for encoding classes for ordinal classification problems. Additionally, we assessed this method, together with other four methods for the task of failure prediction in Hard Disk Drives. All methods were tested with a dataset of 1,697 HDDs collected on a period of almost 4 years. This encoding scheme was applied in two deep neural networks, one based on LSTMs and other on GRUs. In order to tune the hyperparameters of the encoding scheme, Bayesian optimization was applied. Our experiments showed that all methods that propose different encoding for the classes outperformed the standard classification approach when assessed on metrics that consider the ordinal nature of the tasks. Also, the proposed encoding scheme had the best performance for the metric that takes into account both the asymmetry and ordinal aspects of the task. Besides that, it also improved the classification accuracy for the most critical classes, the ones near the device end-of-life.

Future work includes exploring different encoding schemes parameters for each class separately, which would require an optimization whose amount of parameters is proportional to the number of classes. Also, another topic is finding more suitable loss functions that would directly optimize the kappa, including the encoding parameters.

6 CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this dissertation, we explored the problem of health assessment of hard disk drives with deep neural networks. The problem was modeled with two different approaches, as a regression task, and as a classification task. The regression task consisted in predicting the remaining useful life of the device in days up to its failure. The classification task, in its turn, simplified the regression problem by splitting the prediction interval into sub-intervals, or health degrees.

For the regression task, DNNs based on CNNs, LSTMs and GRUs were assessed. The topology and hyperparameters of both feedforward and recurrent networks were tuned through Bayesian Optimization. The resulting optimized networks outperformed SRNs when evaluated under three classical prognostics metrics. In addition to that, three hidden state vector initialization strategies were assessed for the recurrent neural networks. These initializations had a significant impact in almost all prognostics metrics without significantly affecting the iterations required for training.

In the classification approach, we proposed an LSTM based solution that outperformed the state-of-the-art method, which was based on SRN, in the long-term prediction horizon, while producing similar results in the short-term setting. Also, the modeling of the health assessment problem as a classification task led to particularities such as the ordinal and asymmetric aspects between the resulting classes, that were tackled by the proposal of an adjustable encoding scheme. This encoding scheme resulted in improved classification results.

All experiments were conducted with real-world data of 1,697 HDDs collected on a period of almost 4 years.

6.2 Future Work

Open research opportunities in the topics covered in this dissertation include the evaluation and proposal of different loss functions that can improve the prognostics metrics discussed in section 4.1. In addition to that, other network architectures could be explored, by allowing additional parameters to be optimized through the Bayesian optimization procedure discussed in section 4.3.

Another avenue to be explored is, in the context of the classification task, the evaluation of employing different encoding schemes parameters for each class separately. Also, another topic is finding more suitable loss functions that would directly optimize the kappa, including the encoding parameters.

BIBLIOGRAPHY

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M. *et al.* Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **arXiv preprint arXiv:1603.04467**, 2016.

BACKBLAZE. **Hard Drive Data and Stats**. 2016. [Online; accessed 2017-04-26]. Available at: https://www.backblaze.com/b2/hard-drive-test-data.html.

BECKHAM, C.; PAL, C. A simple squared-error reformulation for ordinal classification. **arXiv preprint arXiv:1612.00775**, 2016.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. **IEEE transactions on neural networks**, IEEE, v. 5, n. 2, p. 157–166, 1994.

BOTEZATU, M. M.; GIURGIU, I.; BOGOJESKA, J.; WIESMANN, D. Predicting disk replacement towards reliable data centers. *In*: ACM. **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [*S.l.*], 2016. p. 39–48.

CHAUVIN, Y.; RUMELHART, D. E. **Backpropagation: theory, architectures, and applications**. [*S.l.*]: Psychology Press, 2013.

CHAVES, I. C.; PAULA, M. R. P. de; LEITE, L. G.; QUEIROZ, L. P.; GOMES, J. P. P.; MACHADO, J. C. Banhfap: A bayesian network based failure prediction approach for hard disk drives. *In*: IEEE. **Intelligent Systems (BRACIS), 2016 5th Brazilian Conference on**. [*S.l.*], 2016. p. 427–432.

CHENG, J.; WANG, Z.; POLLASTRI, G. A neural network approach to ordinal regression. *In*: IEEE. **Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on**. [*S.l.*], 2008. p. 1279–1284.

CHO, K.; MERRIËNBOER, B. V.; GULCEHRE, C.; BAHDANAU, D.; BOUGARES, F.; SCHWENK, H.; BENGIO, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. **arXiv preprint arXiv:1406.1078**, 2014.

COHEN, J. A coefficient of agreement for nominal scales. **Educational and psychological measurement**, Sage Publications Sage CA: Thousand Oaks, CA, v. 20, n. 1, p. 37–46, 1960.

COHEN, J. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. **Psychological bulletin**, American Psychological Association, v. 70, n. 4, p. 213, 1968.

CONNEAU, A.; SCHWENK, H.; BARRAULT, L.; LECUN, Y. Very deep convolutional networks for text classification. *In*: **Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers**. [*S.l.*: *s.n.*], 2017. v. 1, p. 1107–1116.

ELMAN, J. L. Finding structure in time. **Cognitive science**, Wiley Online Library, v. 14, n. 2, p. 179–211, 1990.

GRAVES, A.; SCHMIDHUBER, J. Framewise phoneme classification with bidirectional lstm and other neural network architectures. **Neural Networks**, Elsevier, v. 18, n. 5-6, p. 602–610, 2005.

HAMMER, B. On the approximation capability of recurrent neural networks. **Neurocomputing**, Elsevier, v. 31, n. 1-4, p. 107–123, 2000.

HERMANS, M.; SCHRAUWEN, B. Training and analysing deep recurrent neural networks. *In* : **Advances in Neural Information Processing Systems**. [*S.l.* :*s.n.*], 2013. p. 190–198.

HERSHEY, S.; CHAUDHURI, S.; ELLIS, D. P.; GEMMEKE, J. F.; JANSEN, A.; MOORE, R. C.; PLAKAL, M.; PLATT, D.; SAUROUS, R. A.; SEYBOLD, B. *et al.* Cnn architectures for large-scale audio classification. *In* : **IEEE. Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on**. [*S.l.*], 2017. p. 131–135.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural networks**, Elsevier, v. 2, n. 5, p. 359–366, 1989.

KOZUBOWSKI, T. J.; PODGORSKI, K. of laplace distribution. **Computational Statistics**, v. 15, p. 531–540, 2000.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *In*: **Advances in neural information processing systems**.[*S.l.*: *s.n.*] , 2012. p. 1097–1105.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, IEEE, v. 86, n. 11, p. 2278–2324, 1998.

LI, J.; JI, X.; JIA, Y.; ZHU, B.; WANG, G.; LI, Z.; LIU, X. Hard drive failure prediction using classification and regression trees. *In* : **2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**. [*S.l.* : *s.n.*] , 2014. p. 383–394. ISSN 1530-0889.

LIMA, F. D. dos S.; AMARAL, G. M. R.; LEITE, L. G. de M.; GOMES, J. P. P.; MACHADO, J. de C. Predicting failures in hard drives with lstm networks.*In* : IEEE. **2017 Brazilian Conference on Intelligent Systems (BRACIS)**. [*S.l.*], 2017. p. 222–227.

LIMA, F. D. dos S.; PEREIRA, F. L. F.; LEITE, L. G. de M.; GOMES, J. P. P.; MACHADO, J. de C. Remaining useful life estimation of hard disk drives based on deep neural networks. *In* : IEEE. **Neural Networks, 2018. IJCNN 2018.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on**. [*S.l.*], 2018.

MCKINNEY, W. **Pandas: a python data analysis library**. 2008. [Online; accessed 2017-04-26]. Available at: http://pandas.sourceforge.net.

MURRAY, J. F.; HUGHES, G. F.; KREUTZ-DELGADO, K. Machine learning methods for predicting failures in hard drives: A multiple-instance application. **Journal of Machine Learning Research**, v. 6, n. May, p. 783–816, 2005.

OLAH, C. **Understanding LSTM Networks**. 2015. [Online; accessed 2017-04-26]. Available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

OLIPHANT, T. E. **A guide to NumPy**. [*S.l.*]: Trelgol Publishing USA, 2006. v. 1.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V. *et al.* Scikit-learn: Machine learning in python. **The Journal of Machine Learning Research**, JMLR. org, v. 12, p. 2825–2830, 2011.

PINHEIRO, E.; WEBER, W.-D.; BARROSO, L. A. Failure trends in a large disk drive population. *In*: **Proceedings of the 5th USENIX Conference on File and Storage Technologies**. Berkeley, CA, USA: USENIX Association, 2007. (FAST '07), p. 2–2.

Ponemon Institute LLC. **Cost of Data Center Outages**. [*S.l.*], 2016. [Online; accessed 2018-08-23]. Available at: https://www.vertivco.com/en-us/insights/articles/pr-campaigns-reports/benchmark-series/.

QUEIROZ, L. P.; RODRIGUES, F. C. M.; GOMES, J. P. P.; BRITO, F. T.; CHAVES, I. C.; PAULA, M. R. P.; SALVADOR, M. R.; MACHADO, J. C. A fault detection method for hard disk drives based on mixture of gaussians and non-parametric statistics. **IEEE Transactions on Industrial Informatics**, IEEE, 2016.

RINCóN, C. A. C.; PâRIS, J.; VILALTA, R.; CHENG, A. M. K.; LONG, D. D. E. Disk failure prediction in heterogeneous environments. *In*: **2017 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)**. [*S.l.*: *s.n.*], 2017. p. 1–7.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958.

SAXENA, A.; CELAYA, J.; SAHA, B.; SAHA, S.; GOEBEL, K. On applying the prognostic performance metrics. *In*: **Proceedings of the Annual Conference of the Prognostics and Health Management Society, 2009**. [*S.l.*: *s.n.*], 2009. p. 1–16.

SCHROEDER, B.; GIBSON, G. A. Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you? *In*: **FAST**. [*S.l.*: *s.n.*], 2007. v. 7, n. 1, p. 1–16.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. **Information Processing & Management**, Elsevier, v. 45, n. 4, p. 427–437, 2009.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. **The Journal of Machine Learning Research**, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

The GPyOpt authors. **GPyOpt: A Bayesian Optimization framework in python**. [Online; accessed 2017-06-26]. Available at: http://github.com/SheffieldML/GPyOpt.

WANG, Y.; MA, E. W. M.; CHOW, T. W. S.; TSUI, K. A two-step parametric method for failure prediction in hard disk drives. **IEEE Transactions on Industrial Informatics**, v. 10, n. 1, p. 419–430, Feb 2014. ISSN 1551-3203.

WANG, Y.; MIAO, Q.; MA, E. W. M.; TSUI, K.; PECHT, M. G. Online anomaly detection for hard disk drives based on mahalanobis distance. **IEEE Transactions on Reliability**, v. 62, n. 1, p. 136–145, March 2013. ISSN 0018-9529.

WERBOS, P. J. Backpropagation through time: what it does and how to do it. **Proceedings of the IEEE**, IEEE, v. 78, n. 10, p. 1550–1560, 1990.

XU, C.; WANG, G.; LIU, X.; GUO, D.; LIU, T.-Y. Health status assessment and failure prediction for hard drives with recurrent neural networks. **IEEE Transactions on Computers**, IEEE, v. 65, n. 11, p. 3502–3508, 2016.

YE, Z.-S.; XIE, M.; TANG, L.-C. Reliability evaluation of hard disk drive failures based on counting processes. **Reliability Engineering System Safety**, v. 109, p. 110 – 118, 2013. ISSN 0951-8320.

ZHU, B.; WANG, G.; LIU, X.; HU, D.; LIN, S.; MA, J. Proactive drive failure prediction for large scale storage systems. *In*: IEEE. **2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)**. [*S.l.*], 2013. p. 1–5.

ZIMMERMANN, H.-G.; TIETZ, C.; GROTHMANN, R. Forecasting with recurrent neural networks: 12 tricks. *In*: **Neural Networks: Tricks of the Trade**. [*S.l.*]: Springer, 2012. p. 687–707.