Lehrstuhl das Fach Verteilte Informations und Multimedia-Systeme

Health Status for Hard Drive Failure Detection

Masterarbeit von

Miguel Vieira Pereira

1. Prüfer	2. Prüfer
Prof. Dr. Harald Kosch	Prof. Dr. Michael Granitzer

July 30, 2025

Contents

Abstract

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The amount of information produced and processed in the world has seen a steady increase over the past decades. The world internet traffic is increasing exponentially since a few decades. Edholm's Law[1] predicts that this behaviour should continue until at least 2030. It shows that telecommunications data rates are rising in a manner analogous to the one predicted by Moore's Law[2]: doubling every 18 months. Even 20 years after this prediction was done, it is still used by organizations responsible by developing the infrastructure to transmit huge amount of data [3].

As the amount of data generated by users all around the world increases, the need to store and access this data increases accordingly. This evolution is matched by an expansion on the number of digital services offered to users: musics, videos, books and other types of files and media that can be accessed from anywhere. The solution companies found to these problems was Cloud Computing. Even though Cloud Computing is a vague term, one way to define it is "a technique where IT services are provided by massive low-cost computing units connected by IP networks"[4].

However, even though Cloud Computing is sold as a way for users to access computing resources they don't have physical access to, the hardware responsible for this computing power must be placed somewhere. This is achieved by the establishment of multiple data centers all around the world to which devices from anywhere can connect in order to get access to the desired services. Microsoft, for example, has 300 data centers worldwide to provide their services to their clients[5].

A data center is a complex installation that consists of thousands of hard drives connect by kilometers of optical fiber cables. Therefore, there are massive amounts of investment done in order to create and maintain these facilities. Microsoft is investing $80 billion

in order to improve their data centers[5]. So there is a demand for services related to the maintenance and improvement of data centers.

In this context, one of the main aspects of Cloud Computing in general is the strong virtualization and reliability of the system[4], meaning that even if some of the hardware fails the service should still be provided without interruption. So, as hardware components fail, they need to be replaced to ensure that the system keeps working for a long time.

Moreover, in order to maintain a reliable system it is not enough to only replace the hardware after it fails. In order to prevent some serious issues such as data loss without having to permanently have a copy of all the data it is necessary to predict the hardware failures.

Therefore, one of the main problems faced by data centers is the need to detect which pieces of hardware are going to fail before a fatal crash occurs to allow it to be replaced. This allows both the prevention of data loss.

Out of all the components that are part of a computer, 80% of the failures occur due to the problems on the hard drives, as indicated by Table 1.1 Therefore, there is a special focus on predicting failures on disks.

Hardware vendors are well aware of this situation and the difficulties related to managing thousands of hard drives. So, in order to allow their users to better tackle these problems, they added a monitoring system to their drives. This technology is called SMART (Self-Monitoring, Analysis, and Reporting Technology).

Using this protocol, vendors can provide to users indicators of the disk status to the user. Some of these attributes are Power-On Hours, Air Flow Temperature and Reallocated Sector Count (number of sectors that had to be copied elsewhere on the disk after a failed read or write operation in order to prevent data loss)[7].

The SMART system can also include status of different attributes. However, these are limited to threshold not exceeded or threshold exceeded for each attribute, where the threshold is a reference value set by the vendor[7].

Moreover, even when there is an indication that a threshold has been exceeded, it does not mean that the drive will suffer an unrecoverable failure. Sometimes it can be a sign that the drive will work more slowly than its specifications indicate.

Device	Proportion
HDD	81.84 %
Miscellaneous	10.20 %
Memory	3.06 %
Power	1.74 %
RAID card	1.23 %
Flash card	0.67 %
Motherboard	0.57 %
SSD	0.31 %
Fan	0.19 %
HDD backboard	0.14 %
CPU	0.04 %

Table 1.1: Data center failure percentage by component - [6]

In addition to that, these status provided by the vendors consider each attribute independently. They do not take into account the fact that some problems in the hard drive can cause different indicators to increase simultaneously, so by considering the relationship between different attributes would allow problems to be more accurately detected. For example, having a huge number of Power-On cycles when compared to Power-On hours may indicate a problem in the disk that causes it to crash and restart constantly.

Finally, this ad-hoc approach uses static values and do not consider the evolution of the attributes over time. Imagine the situation of a disk that has not reallocated any disks so far, and suddenly rate at which sectors have to be reallocated increases. By taking into account how this attribute evolves over time, this problem can be more quickly detected before the threshold value is reached.

However, even though the built-in failure detection system is not very effective, it doesn't mean that the data itself cannot be used to obtain more interesting results, after all the problem of failure detection must still be solved. Current research uses mostly machine learning approaches such as SVMs and Neural Networks to tackle this problem. Further details of how these methods work will be detailed in Chapter 2.

This project has three main objectives. The first one is to create a library that implement some of the current approaches used in research. The idea is to also made it in such a way that it can be extended with new methods in order to allow them to be easily tested and compared to other ones.

After creating the library, further tests can be performed on existing methods to understand how they evolve as their parameters change. For example, the research of Zhu et al. on backpropagation neural networks for disk failure[8] uses a neural network with one single hidden layer with a fixed amount of 30 nodes on it. With our library, we can change the number of nodes of a neural network analogous to the one presented by them and study how the results change when the number of layer or nodes is altered.

The third objective is to extend existing methods to also make use of the concept of Health Status introduced by Xu et al.[9]. Their idea is to give a score from 1 to n to each sample depending on how close it is to a failure, instead of using only a pass/fail status.

So, the goal is to extend other methods to support more than two classes and study their reaction to this change. Different models can be extended this way, whether they are other neural network based methods such as [8] or even if they use other approaches such as the Random Forest one presented by Shen et al.[10].

1.2 Problem

Let $x_{i,t}$ be the vector in which the j^{th} component designs the measure of the j^{th} SMART attribute for disk i at time t. As input, we also have a value y_i that is either 0, if this disk has eventually failed or 1 if, until the end of the observation period, the disk kept working properly. Also, let the number of different smart attributes be equal to m.

Then given a series of vectors with the SMART attributes for an unseen disk, $\hat{x}_t$, we want to output a value $\hat{y}_t$. Here, $\hat{y}_t$ should be equal to 1 if the algorithm predicts that the samples $(\hat{x}_1 \dots \hat{x}_t)$ are closer to the samples in the input that did not fail than to the ones that malfunctioned, and 0 otherwise.

In a real world scenario, if the value $\hat{y}_t$ is equal to 1, then a warning should be sent to the people responsible for maintaining the data center in order to replace the concerned disk.

This corresponds to a classical classification problem. The numerical vectors as input and a binary output suggest that machine learning methods such as Classification Trees[11], Support Vector Machines[8] and Neural Networks[9] are well adapted to tackle this problem.

However, this problem has a few specific characteristics. First of all, the samples are not completely independent, since the values $(x_{i,1}, x_{i,t})$ are a time series that describe the status of disk i through time. As we will see on Chapter 3, this can be a motivation to use methods such as LSTM (Long Short-Term Memory) networks that can encode and work with time dependencies.

Moreover, even other methods such as Random Forests that can't easily deal with time dependencies can be extended to make use of the time series aspect of the data. This can be one by appending new components to $x_{i,t}$ corresponding to the value $x_{i,t}[j] - x_{i,t-\delta}[j], j \in \{1, \ldots, m\}$. In this case, δ is a constant that allows the algorithm to detect sudden changes in the value of an attribute[10].

A second point that needs to be mentioned is that the amount of disks in each class is not similar at all. Since a hard drive has a service life of around 3 to 5 years[12], and observations are performed for a few weeks, no failure will be observed for most of the disks. The ratio of failing disks over the total amount is between 0.4% and 1.9%[9]. So, any approach to this problem has to handle this imbalance in the input data.

Thirdly, the meaning of each SMART attribute is not the same for different vendors[7]. The protocol is standardized, but what it reports is not. So, an algorithm for failure prediction cannot be trained on data from different vendor disks. Moreover, in general, it is not a good idea to mix data from different disks on the same dataset, since they can have different failure causes profiles.

Most of the time, this does not pose a problem to the datasets generated by the data centers. This is due to the fact that, in practice, a data center uses hundreds of copies of the hardware of the same model and has at most a couple of different models. This allows for the replacement if failing disks do be done in a cheaper and more swiftly manner, since it simplifies stock management.

When evaluating an algorithm there are two main metrics that must be taken into account. The first one is the FDR (Failure Detection Rate). This is equal to the ratio

between the number of disks that are correctly predicted to be failing and the total amount of failing disks. Its value should be as close to 1 as possible

The second metric is the FAR (False Alarm Rate). This is equal to the ratio between the number of disks that are wrongly predicted to be failing and the total amount of disks that do not fail. Its value should be as close to 0 as possible.

The challenge is to find algorithms that find an equilibrium between this values. If it predicts every $\hat{x}_t$ to give an output $\hat{y} = 1$, the FAR will be equal to 0, but the FDR too. On the other hand, if it predicts every $\hat{x}_t$ to give an output $\hat{y} = 0$, the FDR will be equal to 1, but the FDR too. So, we notice that a trade off must be done between the FDR and the FAR.

It does not mean, however, that both metrics should be treated identically. Suppose that during a period of one month, 2% of the hard drives in a data center fail. This corresponds to a lifetime of about 4 years, which may represent a real world scenario. If the FDR decreases by 1%, then only 0.02% of the disks of the data center will fail without a warning during one month.

In contrast, if the FAR increases by the same amount, then 0.98% of the disks of the whole data center will be unnecessarily replaced during the same period. This is close to 50% of the disks that actually need to be replaced and can incur a substantial cost. So, most algorithms in the literature prefer to have a bias towards classifying a disk as working rather than failing.

Aditionnaly, it is impossible to reach perfect values for both indicators. This is due to the fact that hard drives are subject to real world conditions. For example, a short circuit on a disk may be caused by an electrical surge, but it does not depend on the internal state of the disk and thus cannot be predicted by the approach using SMART attributes.

Some additional steps can be made after executing the machine learning algorithm. A common one is the use a voting algorithm. The simplest way this can be done is as follows[10]: each sample $\hat{x}_t, t \in \{1, \ldots, T\}$, often including the change rate components, is evaluated independently. A sequence $\hat{y}_t, \{1, \ldots, T\}$ is obtained. Then, a fraction f is chosen. If more than $f \cdot T$ of $\hat{y}_t$ is equal to 0 then the output of the system is 0, otherwise it is 1.

The main advantage of this method is that it allows to easily control the trade off between the FDR and the FAR which can be desired as we have seen above. It suffices to increase f in order to decrease the FAR at a cost of also decreasing the FDR.

2 Background

Given the way the problem can be formulated according to Section 1.2, the failure detection problem can be viewed as a classification problem.

In this case the SMART attributes are the input variables. We have two classes: **good**, which corresponds to the disks that did not fail during the period that they were observed, and **bad**, which are the ones that did fail.

Therefore, machine learning approaches are the most appropriate to tackle the problem. In the literature, some of the machne learning methods used are Decision Trees, Random Forests, Recurrent Neural Network and Long Short-Term Memroy Networks.

In the next few sections we will discuss how these approaches were implemented and the results that they obtained.

2.1 Decision Tree

Decision Trees represent a flowchart in the form of if-else statements. Because of this, one of their main advantages is the fact that it is fairly easy to interpret a decision tree and to understand how it works and how it arrives at its conclusions.

Formally, a Decision Tree is a rooted tree $G = (V, E), E \subseteq V^2$. The leaves of the tree are labeled with one of the possible results of the decision problem. In our case, it will be either **good** or **bad**.

The other nodes are called decision nodes. When evaluating an unseen sample, consists in traversing the tree starting from the root and taking a left or right child of the current node depending on the conditions in it until a leaf is reached, which corresponds to an output.

For the following discussion, it may be useful to have at least a superficial understanding of the training process of a Decision Tree. The tree starts with a single node, the root, and all the training samples are placed there. The root is then added to a queue.

While the queue is not empty, the first node in it is processed. The processing step starts by choosing a splitting value c for each attribute.

Then, for each attribute i, the samples are divided in two sets depending on wheter its value is bigger or smaller than a threshold value c_i. One way to obtain the threshold value is to compute the average of the attribute i over the samples assigned to the node being processed.

Then the algorithm computes which of the partitions maximize a certain criterion function. A common criterion to use here is the information gain, which is the opposite of the Shannon Entropy [13].

Then, two new nodes and edges are created. The original node is then labeled by (i, c_i) where i is the index of the attribute that maximizes the criterion function. The training samples that were assigned to the node are they split between its two children according to the value of their attribute i.

If some condition for one of the children is met such as a certain depth or all samples are in the same node, then the is not added to the queue. Instead, it is kept as a leaf with a label that corresponds to the class that appears most frequently in the samples assigned to it.

Otherwise the node is added to the queue to be processed later.

After being trained, when the tree needs to predict the class of a certain sample x it will start traversing the tree from its root. Then, at each non-leaf node, it will read the label (i, c) and if x_i is smaller than c then the next node in the traversal is the left child, else it is the right child.

The training and the evaluation processes are illustrated by **Algorithm 1** and **Algorithm 2** respectively.

Algorithm 1: TrainDecisionTree(T: train set)

```
1  r ← Node(), q ← Queue();
2  r.samples ← T;
3  q.push(r);
4  while not q.empty() do
5      n ← q.pop(), maxi ← ∞, att ← −1;
6      c ← −1;
7      for i ← 1 to m do
8          c′ ← avg(n.samples, i);
9          s1 ← {x ∈ n.samples | x_i ≤ c′};
10         s2 ← {x ∈ n.samples | x_i > c′};
11         gain = criterion(s1, s2);
12         if gain > maxi then
13             mini ← gain;
14             att ← i;
15             c ← c′;
16             n.left.samples ← s1, n.right.samples ← s2;
17         end
18     end
19     n.attribute ← att;
20     n.threshold ← c;
21     for child of n do
22         if shouldProcess(child) then
23             q.push(child);
24         end
25         else
26             child.isLeaf ← true;
27             child.result = mostCommon(child.samples);
28         end
29     end
30 end
```

Algorithm 2: EvaluateDecisionTree(*tree*: Decision Tree, x: sample)

```
cur ← tree.root;
while not cur.isLeaf do
    if x[cur.attribute] ≤ cur.threshold then
        cur ← cur.left;
    end
    else
        cur ← cur.right;
    end
end
return cur.result;
```

The explicability of this model comes from the fact that when a sample is evaluated, the decision steps that resulted in the corresponding output can be followed, as show in **Algorithm 2**. So, it is possible to verify which parameters are taken into account and to predict what would happen if one of then was changed. Therefore the impact of each attribute can be interpreted.

Decision Trees come in two flavors: Classification and Regression Trees. The former corresponds to classifying samples in discrete, independent classes. So, Classification Trees will treat **good** and **bad** samples as different entities.

Regression Trees are concerned in predicting a continuous value. So, they allow us to use the concept of health status.

Suppose we try to give an integer score from 1 to m to each drive. The ones that do not fail correspond to a value of m while the samples of the failing drive are given values from 1 to $m - 1$ depending on how close they are to the moment of the breakdown. Then a Regression Tree will not try to simply predict an integer from 1 to m, intead it will try to find an exact value to it.

We can explain the difference between the two approaches when using multiple classes as follows: imagine we have a training sample whose desired output value is 1. Then, for the Classification Tree the error is the same when it puts it in the class 2 and in the class m.

On the other hand, a Regression Tree uses a mean squared error funcion, so even if it doesn't put the sample in the class 1 it will consider it worse the cases in which it is put

in class m compared to when it is put in class 2. It may even predict a value of 1.5 to the sample, even though there is no sample in the training set with this value for the dependent variable.

Compare this to the task of classifying a shape as a square, a triangle or a circle. Then, unless there is some additional issue specific to the context of the application, misclassifying a circle as a square is not worse than classifying it as a triangle . This fundamentally differs from the task of predicting a health status in which it is better to predict a 1.5 to a sample whose expected output is 1 than to predict a 5 for example.

Decision Trees were used by Li et al. [11] in order to obtain an algorithm capable of predicting hard drive failure. They tested both Classification and Regression trees.

In their approach, they were able to obtain an FDR of more than 95% while keeping the FAR below 1%.

Apart from the result in itself this work introduce some additional concepts that are useful for the implementation of other methods and is actually used by later researchers. First of all, they add a feature selection step that keeps only a subset of the SMART features passed as input.

More interesting though is the fact that they add columns to their table. This corresponds to the variation of the value of certain features over an interval. It allows the algorithm to take into account the fact that the training samples for a specific hard drive represent a time series.

More precisely, they choose a value for T. Let the samples for the i^{th} hard drive be the list x_i in which each entry corresponds to a sample and they are ordered in the order they were taken. Let $x_{i,j}[t]$ be the j^{th} feature of the sample taken at instant t. Then the value of the new column for the vector $x_i[t]$ is:

$$x_{i,n+j}[t] = x_{i,j}[t] - x_{i,j}[t-T], j \in \{0, \ldots, n-1\}$$

Where n is the original number of features in the vector.

This is an elegant way to include the time dependence aspect of the problem. The main advantage is that it can be applied to any method to try to improve methods that are not designed to work with time series.

In their work, they also use a voting algorithm. This way, in order to classify an unseen hard drive, they take the last N samples and evaluate each of them using the Decision Tree. If more than $\frac{N}{2}$ of them are put in the **bad** class, then the hard drive is classified as failing.

However, their research on Decision Trees does not implement some other techniques used by other research projects and that could be included. For instance, they do not test different thresholds for the voting algorithm. They always use a ratio of 0.5.

In addition to that, they always train the model with a constant ratio between good and bad disks in the training set. For each 3 **bad** ones, they include 7 **good** ones. They do not study what happens if, for example, for each **bad** HD there are 10 **good** ones, which more closely represents the real world scenario if no data is filtered.

In the end, the results they obtained are promising. They obtained FARs below 0.5% while keeping the FDR around 95% for the Classification Tress.

The Regression Tree model used a continuous value between -1 and 0 as the health status value, linearly dependent on how much time before the breakdown of the drive is. This approach was able to increase the FDR by 1% while also decreasing de FAR by around 0.2% when compared to the Classification Tree implemented by them.

2.2 Random Forest

One of the problems faced by Decision Trees, specially when they become large is overfitting [14]. This is due to the fact that the split values for each attribute used by the tree are directy taken from the values on the training set. So, when the tree is deep and there are only a few training samples in a node, the splitting values will sharply follow the ones in the training set.

Another way to explain this is that since a Decision Tree can closely follow the patterns observed in the training set because it can choose the splitting threshold independently of the ancestor nodes in the tree, it presents a small bias. But in Machine Learning there is a principle known as the Bias-Variance tradeoff that states that a model with low bias will present a high variance and vice-versa [15].

More concretely, for our problem at hand, suppose that there are only a few samples in the training set with a certain cause of failure. During training, if they are in a node **N** and are mixed with other samples, we may be able to correctly put them in a leaf that is a child of **N**.

So, the model will have a good performance on the training set which corresponds to a small bias, since the expected and predicted value will be the same. However, imagine that when evaluating an unseen sample there is an attribute, that is not important, to identify this cause of failure, is slightly different from the values observed in the training set.

Then, the path taken will not go through **N** and thus it won't go through the same evaluation process as samples that are simillar. As a result, a small change in a random attribute can cause a huge differnce on the path traversed and thus on the result. Having an algorithm that can behave very differently for slightly different inputs corresponds to a model that has a high variance.

In order to tackle this, the concept of Random Forests has been introduced [16]. The idea is to train multiple, independent Decision Trees at once, each one trained with a different subset of the training data. Notice that these subsets do not have to be disjoint, specially when the train set is relatively small.

Random Forests have performed well for a variety of tasks, ranging from image recognition to Alzheimer's disease detection and prediction [17].

The set of trees has, therefore, a smaller bias when compared to the one trained using all the data at once. This is due to the fact that we can reduce the probability of overfitting, since there will have a larger varierty of decision nodes.

If we return to the example of a specific cause of failure, the training samples corresponding to it will not even necessarily be on the same tree anymore. So, there is a larger set of nodes that have been trained with samples corresponding to this failure. Therefore, the probability of going through one of them when traversing the tree is higher even if the unseen sample does not correspond perfectly.

The drawback is that since the training samples with the same failure cause will be spread, the probability of having a node that had multiple ones when training is smaller, which can increase the error on the training set and on samples very simillar to the ones in the training set.

Once the set of trees is created and trained, it remains to decide how to regroup all of them.

The most common approach when asking for the model to predict the result for a sample is to take the value predicted by each tree and then combine them. For a classification problem, it can be returning the most common predicted class, while for a regression problem it can be done by taking the average of the outputs of the trees.

This can be done quite efficiently, since, at each node, it suffices to compare the value of a specific attribute with a threshold and go to the corresponding child. Moreover, since the traversal processes are independent, they can even be done in parallel.

A research by Shen et al. [10] used Random Forests for hard drive failure prediction. One of the most interesting aspects of their research is how they combined the results of the different tree of the forests.

Instead of simply checking whether most trees had an output of **good** or **bad** for a specific sample, they giving different trees different weights based on a clustering algorithm.

More specifically, they performed the clustering process for the good and bad samples independently. Then, when predicting the outcome for an unseen sample, the clusters c_1 and c_2 to which it would belong if it were a good or bad one are computed.

Since it is known which samples of the training set were used to train each tree, it is possible to evaluate the accuracy of each tree for the training samples in c_1 and in c_2. Based on these accuracies, it is possible to give a bigger weight to trees that better learned the samples of the clusters to which the sample being evaluate belongs.

They used a simple algorithm in which the weight of a tree is either 0 or 1 depending on whether its accuracy for its training sample belonging to either c_1 or c_2 is at least 0.5. However, it is not difficult to imagine a slightly more complex algorithm that results in a weight $w \in [0, 1]$ which is directly proportional to the accuracy.

Despite this simplicity, they were able to achieve a great performance. The FDR obtained was above 98% while the FDR was kept under 0.1% for multiple scenarios.

What allowed them to get these excellent outcomes, besides of course the algorithm, was their dataset that contained more than 2 million samples of SMART attribute snapshots.

2.3 Backpropagation Neural Network

A Neural Network (NN) is a Machine Learning model that can be used to perform both classification and regression tasks. Its name comes from the fact that its development was inspired by how a brain work: with neurons that can be interpreted as nodes and synapses that connect them. The first implementation of this approach is credited to Rosenblatt with his Perceptron architecture [18].

In general, this architecture can be represented as a sequence of layer of nodes (the neurons) and a set of directed edges from every node in layer l to every node in layer $l+1$. Moreover, each node has an output value o_j and each edge (u, v) has a weight w_{uv}.

The first and last layers are special. The output values of the nodes of the first one correspond to the input values to the network. The values in these nodes represent the sample being evaluated. The last layer, in contrast, represents the output values of the network.

From the values on the input layer we can compute the output values of every node in the network. In order to do that, we need to introduce the concept of activation function φ.

The activation function is a non-linear function that will map the input given to a node to its output. The input to a node v is the sum of the ouput values of u such that (u, v) is in the network times the weight of the edge (u, v). Formally:

$$o_v = \varphi\left(\sum_{u=1}^{n} w_{uv} o_u\right)$$

Where w_{uv} is taken to be 0 when the edge (u, v) does not belong to the graph.

Therefore, from an input to the first layer, we can compute the values of the outputs of the second layer. This process can be repeated until the output value for the last layer is calculated.

The fact that the activation function takes as input a linear combination of the ouputs of the nodes of the previous layer explains why it shouldn't be linear. If φ were linear, then the ouput of each neuron would be a linear with respect to the ones of the last layer. So, the output of the last layer would be a linear function of the inputs.

Having a non-linear activation function is desired, therefore, not because otherwise the math would not work. Instead it is due to the fact that there are other, simpler methods to learn linear patterns such as SVM or simply the minimum squares method. The power of a neural network is exactly that it is able to recognize non-linear patterns, which can only be done when using non-linear activation functions.

Here we note that the activation function does not need to be the same on every layer, but it does not change the analysis we are performing here. Some of the most commonly used activation functions are the ReLU $ReLU(x) = \max(0, x)$ and the logistic function $\phi(x) = \dfrac{1}{1 + e^{-z}}$.

A classic problem that can illustrate how useful a non-linear function can be is the binary classification of samples in two classes 0 and 1. Suppose that on the last layer a function that maps $(-\infty, \infty)$ to $[0, 1]$, such as the logistic function, is used. Then, the output of the network will be a real number between 0 and 1.

A classic interpretation of this value is the probability p of the sample belonging to class 1. Trivially, the probability of the sample belonging to class 0 is $1 - p$. So, the model is able to not only classify the sample, but also to indicate how confident it is that this is the correct result.

The output value of every neuron, including the output ones, can be computed from the input values and the weights. So, in order to fully describe a network, the only attributes that we need are the weights. The challenge is, therefore, how to determine the weights that are appropriate for the problem at hand.

The method used by the Backpropagation Neural Networks to compute the weights of the network is called backpropagation. It is inspired by the Gradient Descent approach first developed by Cauchy [19] long before the advent of computers.

The intuition behind the approach proposed by Cauchy is as follows: imagine there is a person on a region that is full of hills. When the person is at point (x, y), the height relative to the sea level is $h(x, y)$. Now suppose that the person is trying to find the lowest point in the plane.

This can be easily mapped to the problem of trying to minimize a function. Moreover, even though our discussion uses as example a function with two independent variables, it can be extended for functions with more variables without loss of generality.

The gradient descent technique uses the fact that the gradient of a function $\nabla h(x, y)$ always points in the direction in which the incline is non-negative and the steepest at (x, y). Therefore, $-\nabla h(x, y)$ is the direction in which the value of h decreases as fast as possible at (x, y). So, the idea is to take a small step in that direction since the change of height when traveling in that direction in the neighborhood of (x, y) is non-negative.

From the above description it is possible to see that this method tends to arrive at a local minimum. The most notable exception is when the function has some discontinuities or some points in which its values decreases very rapidly. In this case, if the "length" of the step (called learning rate) is too big, it may happen that we jump over the region that would yield a smaller value.

The solution can be to decrease the learning rate, but this causes the algorithm to take longer to converge. So, there is a tradeoff between the time required to train the model and its capacity to find the best values.

In the context of neural networks, the function that the model tries to minimize is the loss function over the training samples and its parameters are the weights of the connections of the network.

It can be proven [20] that at each step, each weight of the network should be update by:

$$\Delta w_{ij}^{l} = -\eta e_i^{(l)} o_j^{(l-1)}$$

Where η is the learning rate $e_i^{(l)}$ is called the error signal of the i^{th} node at layer l. The error signal can be computed recursively as follows:

$$\begin{cases} e_i^{(k)} & = \varphi'(u_i^{(k)}) \\ e_i^{(l)} & = \varphi'(u_i^{(l)}) \sum_j w_{ij}^{(l)} e_j^{(l+1)}, l \in \{1, \ldots, k-1\} \end{cases} \tag{2.1}$$

One notable aspect of Equation 2.1 is that the error signal of neurons on layer l depend only on the error signals on layer $l+1$. So, the errors for each node of a given layer can be computed in parallel.

This property is so important that the capacity of computing such values in parallel made it possible to introduce neural network training in GPUs [21] which was one of the main causes of the AI spring that has been happening on the last 20 years.

Zhu et al. [8] applied the backpropagation neural network approach presented above to the problem of hard drive failure detection. The lowest FAR they obtained was 0.48% while keeping an FDR of 94.6%. They were even able to obtain a model that had and FDR of 100% on their test set, but with a relatively higher FAR of 2.26%.

However, their work has some limitations. Specially due to the fact that they only varied one parameter. The only variable they changed on their experiment was the number of samples per failing hard drive to be used in the training and test processes. So, if the time window was 12 hours, for example, only the samples taken at most 12 hours before a hard drive failed were included in the **bad** class.

No results were presented about what happens with a bigger or smaller network or with more or less input parameters, for instance.

This highlights the importance of the current work that intends to test how different parameters influence the results for a variety of models. Moreover, the objective of creating a piece of software that can be easily configured to train a model with different parameters will allow the study of the impact of different parameters easier, even when the results are not readily available.

2.4 Recurrent Neural Network

A Backpropagation neural network is a relatively simple yet powerful model capable of solving multiple regressions and classification problems. However, due to its simplicity it can have some limitations.

One of the most notable ones is how it evaluates one sample at a time. It does not care about previous samples and doesn't have any type of memory.

However, in many problems the concept of time or, more generally, of sequences is present. Due to how they work, backpropagation neural networks are not capable of efficiently working with sequences.

This limitation was found early on and there are works from the 1980s already trying to figure out how a neural network could store a state depending on what the previous samples were [22].

However, long before computer science tried to tackle the problem of modeling sequences, neuroscience had already tried to find a method that allowed the brain to do so. The question that they were trying to answer was how does memory in the human brain. More specifically, how does neurons and synapses allow the brain to react to stimuli it received in the past.

The first clues to an explanation were found at the end of the 19^{th} century, when researchers were suggesting that the brain had "recurrent semicircles" [23] implying that the signal of a neuron at a certain time t could influence its value at a time $t' > t$.

As a consequence, later on, one of the first mathematical models of a neuron that was developed to try and explain brain activity was the MCP [24], which allowed loops between neurons in the network. In their paper presenting the MCP, McCulloch and Pitts were able to show that their architecture allowed the activity of the network to be influenced by activity indefinitely far in the past [24].

And, as was common at the advent of the development of neural networks, researchers used the model of the brain as an inspiration. This culminated on the work of Jordan [22] and Elman [25]. They proposed a model that is nowadays labeled as Simple Recurrent Neural Networks (SRNN).

In an SRNN, the first layer of the network doesn't have only input nodes, it also has state nodes. These state nodes are connected to the output of the network. Figure 2.1 shows a schema of how such a netwrok is connected.

So, when evaluating a sample, the input layer receives not exclusively the values of the independent variables of the sample, but also some values that depend on its state at previous times.

The ability to modify the output depending on the previous sample evaluated is what allows the to have some type of memory. Models that are able to take don't evaluate each sample independently but are instead able to take into account a context, are useful for applications such as Natural Language Processing (NLP).

This is due to the fact that when trying to predict the next word in a sentence for example, it is important to take into account evey previous word [26]. A backpropagation neural network, due to its nature, would only be able to take into account a fixed number of words.

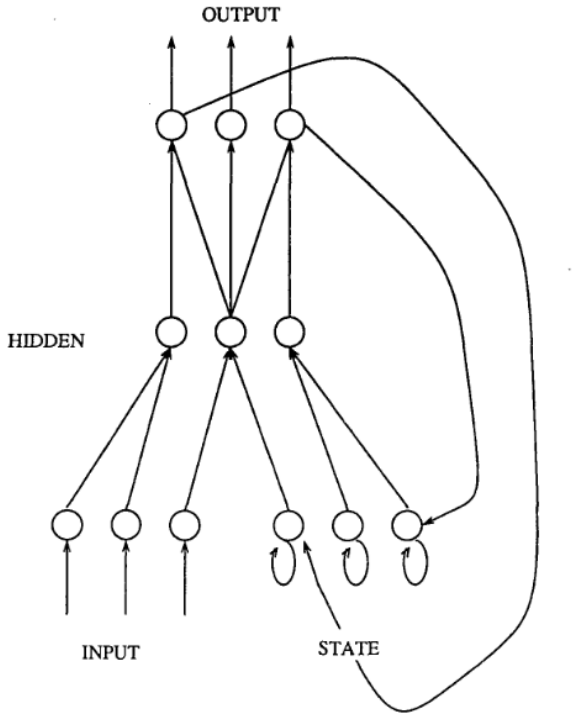

Figure 2.1: Schema of a Jordan Network - [25]

Nowadays, other methods beyond SRNNs are used to do this task. Most notably, in the last few years, Transformers[27] became very popular. Nevertheless, models that are able to take into account a sequence of samples are still very much needed.

In the context of hard drive failure prediction, our samples are not words, but they do consist on a time series and thus form a well defined sequence. This indicates that SRNNs may be a good model to tackle the problem.

Indeed, Xu et al [9] made use of Recurrent Neural Networks to the problem at hand. Their setup for the neural network is quite standard, up until the point in which they introduce the concept of health status.

So, differently from every other approach in the litterature they do not divide the samples on good or failing. They actually use a more general method in which samples are divided in N classes.

The ones from hard drives that do not fail during the observation period are given the label N. The samples from failing disks are divided in the other $N-1$ classes, labeled

from 1 to $N-1$. The algorithm they use to assign the failing samples divides them depending on how close they are to the failure by using fixed-length intervals.

We can formally define the approach that they explained in natural language in their paper as follows: suppose that the last n samples from each failing disk are kept and need to be divided in $N-1$ bins numbered from 1 to $N-1$. Let the last sample from a specific hard drive before it fails be taken at time T_i. Then, supposing that a sample is taken every 1 unit of time (usually once per hour) the class $c_i(t)$ of the sample of the disk i at time t is given by:

$$c_i(t) = \left\lfloor \frac{(N-1)(T_i - t)}{n} \right\rfloor + 1, 0 \leq T_i - t < n \tag{2.2}$$

Here, $\lfloor x \rfloor$ is the floor of x, meaning the largest integer smaller than or equal to x. For example, $\lfloor 1.8 \rfloor = 1$, $\lfloor 3 \rfloor = 3$.

By analizing this formula, we see that the $\frac{n}{N}$ samples closest to the moment of failure are assigned to class 1, the next $\frac{n}{N}$ are assigned to class 2 and so on.

We can also prove that the formula indeed does what is desired, that is, it assigns a number from 1 to $N-1$ to each sample. In order to do so, we notice that since we have n samples taken every 1 unit of time, and the last one is taken at time T_i, the first one is taken at $T_i - n + 1$.

We can compute $c_i(T_i) = 1$ and $c_i(T_i - n + 1) = \left\lfloor (N-1)\left(1 - \frac{1}{n}\right) \right\rfloor + 1$. Since $\left(1 - \frac{1}{n}\right) < 1$, $\left\lfloor (N-1)\left(1 - \frac{1}{n}\right) \right\rfloor < N-1$. Moreover, $c_i(t)$ is clearly monotonic, since it is a composition of the floor function and elementary mathematical operations.

So, we can guarantee that the formula given above for $c_i(t)$ correctly assigns each sample to a class from 1 to $N-1$ in which the closer the sample is to the moment the drive fails, the smaller is the value of its class label.

Here it is important to stress that the implementation of health status on the study by Xu et al [9] is different from the one by Li et al. that used this concept to train their decision trees. The main distinction is that the later used continuous values in order to train a regression tree. The former, on the other hand, divides the samples in discrete classes.

So, when training the recurrent neural network model, the N classes are completely independent. Concretely, it does not use the fact that when the correct output for a certain sample is 1, it is better to classify it as belonging to class 2 than to class N.

Even though the health status is a useful abstraction, in the end there is a need to output a prediction of whether the sample belongs to the **good** or **bad** hard drive. In other words, the model needs to state if the hard drive is going to fail soon or not which is the actual problem we are trying to solve.

In order to do that, when they want to predict the status of a certain drive, they take its last N samples. Then, in order from the oldest to the most recent they put each sample as input to the network. The order in which they are evaluated is important, since the model being used is an RNN and therefore has memory.

For each sample, they classify it as belonging to the class whose associated output neuron value is maximum. By doing this, it is obtained a sequence of integers $c = (c_1, \ldots, c_n), 1 \leq c_i \leq N$.

For each $j \in \{1, \ldots, N\}$, let C_j be the cardinality of j in c. It is clear that $\sum_{j=1}^{N} C_i = n$. Then, they define two algorithms, the Voting Algorithm which Tends to Health (VAT2H) and the Voting Algorithm which Tends to Failure (VAT2F):

$$\text{VAT2H}(C) = \begin{cases} \text{Healthy, if} & \sum_{j=1}^{N-2} C_j \geq C_N \\ \text{Failure, if} & \sum_{j=1}^{N-2} C_j < C_N \end{cases} \tag{2.3}$$

$$\text{VAT2F}(C) = \begin{cases} \text{Healthy, if} & \sum_{j=1}^{N-2} C_j > C_N \\ \text{Failure, if} & \sum_{j=1}^{N-2} C_j \leq C_N \end{cases} \tag{2.4}$$

Notice that the contribution of the class $N - 1$ is ignored in both algorithms. This implies that these algorithms only work for when $N \geq 3$.

So, in order to tackle the problem they start by expanding the number of classes from 2 to N using equation 2.2. Then, after the network classifies a sequence of samples of the same disk in the classes from 1 to N they use equation 2.3 or 2.4 to obtain the final binary classification output.

They perform their tests on three different datasets corresponding to different models of hard drives. The neural network for each model is trained independently since, as discussed, not only the behavior of hard drives of different models can be different, but also, for different vendors, the values of the SMART attributes are not guaranteed to be compatible.

They obtained promising results of their approach. The FDR of their models was around 97% while the FAR was almost always kept below 0.1%. As was expected, the VAT2H algorithm resulted in a lower FDR and a lower FAR than the VAT2F algorithm.

Nevertheless, their study admitted to having some limitation such as not studying the impact of other health status algorithms different from the one in equation 2.2. Moreover, they used a fixed amount of classes with $N = 6$ all over their research as well as a constant number of nodes in the hidden layer of their network.

2.5 Long Short-Term Memory Network

Time has proved that RNNs are an effective method to tackle multiple sequence-based problems such as NLP [26] and video processing [28].

However, RNNs suffer from a limtation called the vanishing gradient problem. This limits their efficiency when they need to learn long term dependencies.

This has been proven theoretically by Hochreiter [29] as well as by Bengio using different approaches [30]. The intuition behind this result can be explained using the concept error signal defined in Equation 2.1.

If we expand the bottom expression to explicitly write $e_i^{(l)}$ in terms of the error signals of layer $l + 2$ instead of $l + 1$, we obtain:

$$e_i^{(l)} = \varphi'(u_i^{(l)}) \sum_j w_{ij}^{(l)} \varphi'(u_j^{l+1}) \sum_k w_{jk}^{(l+1)} e_k^{l+2}$$

From this, we can see that $\frac{e_i^{(l)}}{e_i^{(l+2)}} \propto \varphi'(u_i^{(l)})\varphi'(u_j^{l+1})$. So, it is proportional to the product of two derivatives. We can exend this process to show that the impact induced by layer $l + m$ on layer l must be scaled by the product of m derivatives.

But, if we redraw our RNN to be represented by Figure , we see that the input of sample t can be interpreted as being connected to the imput of sample t' by $(t' - t + 1)k$ layers. So the impact of the sample t on sample t' is proportional to $\prod_{i=1} (t' - t + 1)k\varphi'(x_i)$.

But, if the values of $\varphi'(x_i)$ are all smaller than 1, not only the scaling term will approach 0, but it will do so exponentially fast with respect to the distance between the two samples.

What Bengio showed on his paper [30] is that, as a network learns, these derivatives decrease and become smaller than 1. From this result, he was able to prove that as the distance between two samples increase, the impact of the earlier one on the other on the RNN tends to zero, as long as the network has enough time to learn.

The only assumption he made was that the model was not sensitive to noisy perturbations. This is equivalent to stating that if two data sets are similar, then the model is able to learn approximately the same pattern, which is coherent to the problem at hand.

So, there is a theoretical demonstration that for long enough sequences, the RNN architecture will be subjected to the vanishing gradient problem. However, theory alone is not enough to define what is a long enough sequence. In order to state that the vanishing gradient problem has an impact on RNNs it is needed to observe such phenomenon in real-world scenarios.

And, indeed, experiments show that problems as diverse as sentiment analysis ([31]) and greenhouse gas predictions ([32]) can be better learned by networks that try to solve the vanishing gradient problem such as LSTMs.

Other research projects show that applying techniques explicitly developed to curb the vanishing gradient problem can result in better results even without needing to change the architecture at all. For instance, in [33] they adapted the activation functions that increase the value of their derivatives.

With this evidence, both theoretical and experimental, it is clear the need to develop networks that are able to better handle or completely avoid the vanishing gradient problem.

So, after proving the vanishing grdient problem in RNNs, Hochreiter developed a network that was not subjected to the same problem. His solution was the Long Short-Term

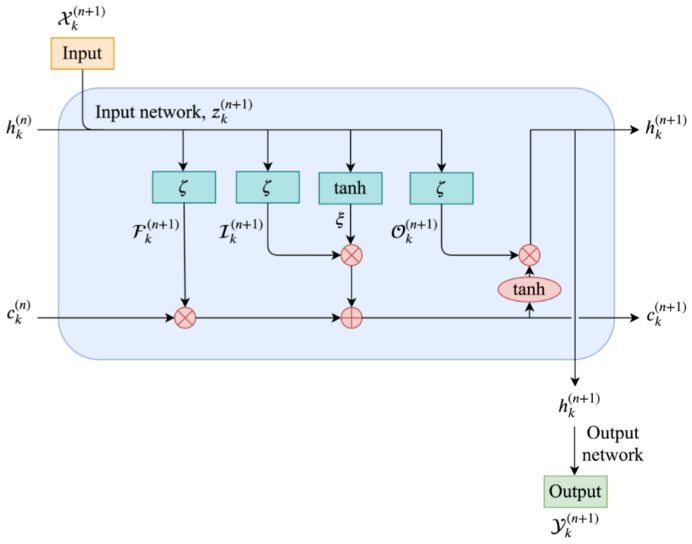

Figure 2.2: Schema of an LSTM - [34]

Memory architecture (LSTM). A schematic view of such network is presented on Figure 2.2.

Simillarly to the RNN architecture, we combine the output of the previous sample (represented by $h^{(n)}$ on the image) with the input data for the current sample ($\chi^{(n+1)}$). However, the main innovation is the presence of the cell state, the channel represented by the horizontal lane in the bottom half of the image and dented by $c^{(n)}$.

It is not updated using the same logic of the neurons used by BPNNs and RNNs so it is not subject to the vanishing gradient problem. The operations performed by an LSTM can be explained in three steps, each one called a gate.

The first one is called the forget gate and is represented in the image by the leftmost ζ node. It takes the input of the current sample ($\chi^{(n+1)}$) and the last output ($h^{(n)}$) of the network and generates a forget factor vector $P^{(n+1)}$ which is then multiplied component by component with the current value of the state $c^{(n)}$.

The name from this gate comes from the fact that it controls how much the network remembers from the past samples. In the limit case in which $P^{(n+1)} = \mathbf{0}$, the value of c is reset and no information is kept from previous interations.

The next step is the input gate, which in the image corresponds to the central ζ and tanh nodes. If the forget gate decides how much from previous iterations should be kept in

the cell state, the input gate controls which data from the current one should be added to the cell state.

The input gate is divided in two steps: the first is to generate the candidate values ξ and the second is to compute the weights $I^{(n+1)}$. Then each component of the candidate values vector is multiplied by its corresponding weight and added to the cell state.

Finally, there is the output gate that will actually compute the predicted ouput value for the sample. In the image it is represented by the rightmost ζ and tanh gates.

This is the step that uses the cell state as input. It combines the cell state value as well as the sample input $x^{(n+1)}$ and the previous sample output $h^{(n)}$ to compute a prediction $h^{(n+1)}$ for the current sample.

In the previous paragraphs, we used generic terms such as compute and generate. However, for an LSTM, the equations to calculate each term presented above is well defined and are given by:

$$\begin{cases} P^{(n+1)} &= \zeta(W_f \cdot [h^{(n)}, x^{(n+1)}] + b_f) \\ I^{(n+1)} &= \zeta(W_i \cdot [h^{(n)}, x^{(n+1)}] + b_i) \\ \xi &= \tanh(W_\xi \cdot [h^{(n)}, x^{(n+1)}] + b_\xi) \\ C^{(n+1)} &= P^{(n+1)} \odot C^{(n)} + \xi \odot I^{(n+1)} \\ O^{(n+1)} &= \zeta(W_o \cdot [h^{(n)}, x^{(n+1)}] + b_o) \\ h^{(n+1)} &= O^{(n+1)} \odot \tanh(C^{n+1}) \end{cases} \tag{2.5}$$

In the equation above, $[x, y]$ is the concatenation of vectors x and y and $x \odot y$ is the Hadamard or element-wise product. Also, W_x and b_x represent, respectively, the weights and bias of the network when computing variable x. These are the values that need to be learned during the training process.

When it comes to applying this architecture to solving the problem of prediciting hard drive failure in data centers, there are not many research projects that focused in this problem that have been published. Nevertheless, there are some related works.

The first of them was done by Zhang et al. ([35]). The main objective of their work was to develop a symbolization-based feature extraction algorithm to improve event detection in LSTMs.

To show the impact of their approach they did perform an experiment on hardware failure detection. However, they were presenting a more general method. Consequently, they used different metrics such as balanced accuracy instead of FAR and FDR that would allow us to directly compare with other methods.

Despite us not having access do the desired metrics we can deduce some aspects of their results that allow us to compare to other works. In order to do that, let positive be the event of the hard drive failing and negative the event of it not failing.

Then the true positives TP are the hard drives whose failure was correctly predicted. The true negative TN are the hard drives that did not fail and that were not flagged as going to fail. The false negatives FN are the hard drives that failed but were not flagged as so. And, the false positives FP are the hard drives that did not fail but that were predicted as going to fail.

This allows us to rewrite the FDR and the FAR as follows:

$$\begin{cases} FDR = \dfrac{TP}{TP + FN} \\ FAR = \dfrac{FP}{FP + TN} \end{cases} \tag{2.6}$$

We then introduce the definition of Balanced Accuracy(BA), which was the metric used in [35]:

$$BA \equiv \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right) \tag{2.7}$$

But by manipulating 2.7, it is possible to rewrite it in terms of the FDR and the FAR:

$$BA = \frac{FDR + (1 - FAR)}{2} \tag{2.8}$$

So, even though it is not possible to retrieve the FDR and the FAR by themselves, it is possible to find a range of possible values for them from the balanced accuracy value.

In [35], the best BA they achieved over all the experiments with the LSTMs, with or without their feature extraction algorithm, was 85.2%. But, since BA is the average of

FDR and $(1 - FAR)$, it implies that one of these values is at most 85.2%, else their average would be bigger.

So, either the FDR they achieved is under 85.2% or their FAR is above 14.8% which is much worse than the one achieved with any other methods we have discussed.

This indicates that their research cannot be used as a reference for the hard disk failure detection problem.

There is an additional paper by Das et al. ([36]) that also applies LSTM to hardware failure detection. However, it is not for hard drives in data centers, but rather to nodes in a supercomputer which leads to a profile much different than the papers discussed so far.

The first main difference is that the time scales in which it operates is much smaller. The TIA they achieve is of the order of 100 seconds, which indicates a very different behavior than the one found in data centers in which it is possible to predict a failure more than 100 hours in advance ([11], [8]).

But the most important difference is that in [36] they try to predict failures that happen in components other than the hard drive. So, they do not make use of SMART attributes, they instead analyze system logs.

Therefore, their motivation to use LSTMs is to detect patterns in text rather than to study time-series.

The FDR they achieve with their LSTM-based approach is between 85.1% and 87.5%. However it is not possible to compare it to the other results presented above since their problem is quite different from the detection of hard drive failures in data centers.

2.6 Other Techniques

There are two additional methods that have been used to tackle the hard drive failure prediction problem and that are worth mentioning. Since these approaches will not be implemented by our current work, we will limit ourselves to a briefer explanation of the theory behind them.

2.6.1 Support-Vector Machine

A Support-Vector Machine (SVM) is a non-probabilistic classifier [37]. The idea behind it is to plot the samples in an n-dimensional space, in which n corresponds to the number of features in the vector.

Then, the algorithm finds the hyperplane that minimizes its loss function. The points on one side of the hyperplane correspond to the positive class and the other to the negative class.

The main drawback of SVMs is that drawing a hyperplane means that it can only detect linear dependencies. So, if there are points in a 2D-plane and the boundaries between the good and bad class is the circumference centered at the origin with radius 1, there is no way that a line can be drawn to correctly divide both regions.

The solution to this is to use the kernel-method, in which dimensions of the original vector are combined using non-linear function and the output of the function is assigned to a new component of the sample vector. This allows the SVM to correctly learn non-linear dependencies between the variables.

In the example above, we can transform the vector $v = (x, y)$ into $v' = (x, y, x^2 + y^2)$ and then and apply the SVM to the set of v'. Then the SVM can correctly generate the plane $x_3 = 1$ and thus solve the classification problem without needing to learn explicitly non-linear dependencies.

When it comes to applying SVMs to predicting hard drive failure, the best results were achieved in [8]. They used hyperparameters on their model in order to prioritize either a small FAR or a big FDR.

When optimizing for the smallest FAR possible, Zhu et al. obtained a FAR of 0.03% and an FDR of 68.5%. When priority was giving to increasing the FDR, they obtained a FAR of 0.3% but the FDR was increased to 80.0%.

Support-Vector Machines, when compared to other approaches, present, therefore, a method that is able to achieve a smaller FAR. However, in contrast, it is not able to attain FDRs as good as other methods while keeping an acceptable FAR.

In the research by Zhu et al. they kept results that had an FAR smaller than 5%, so it implies that improving the FDR above 80% requires increasing the FAR to unacceptable levels.

2.6.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep learning machine learning model. It is usually applied to computer vision tasks. Just like BPNNs discussed in Section 2.3, it is not recurrent meaning that the output for a certain sample is not propagated to following ones.

The main aspect that differs a CNN from a BPNN is the presence of convolutional layers [38]. In a convolutional layer, a neuron in layer l is not connected to every neuron in layer $l-1$. Instead, it is connected to a few consecutive nodes of layer $l-1$.

This is useful for handling data with many input parameters, such as images, in which each pixel corresponds to an input. For a 100x100 image, fully connecting two layers would require 10,000 weights. By using a 5x5 convolution window with shared weights, it is possible to reduce this value to only 25 weights, or 400 times less. This allows the network to be much deeper.

A relevant attribute of a CNN is its translation invariance [39]. The use of shared weights implies that shifting some values by a certain constant value does not hinder the network's ability to recognize a certain pattern. This is because the convolutions will be performed with the same weights no matter in which part of the image a certain sequence of values is.

It was exactly this aspect of the CNN that motivated Sun et al. ([40]) to adapt CNNs to handle time series in order to tackle the hard drive failure prediction problem. For a time-series, translation invariance implies that a pattern due to a sequence of values or events can be detected indepndently of where in the sequence it occurs.

In order to obtain, they also had to adapt the loss function to handle the imbalanced dataset that contained many more good samples than failing ones, which is an intrisic aspect of the problem at hand.

So, they gave a bigger weight to failing samples that were misclassified, else the model would be prone to classify almost every disk as a good one. This is due to the fact that,

with a classic loss function, even if the model misclassified every failing disk, its total loss would be much smaller than if it misclassified even a small percentage of the good disks.

In their results, the metric they presented that we can use to compare with other methods was the precision, which corresponds to the FDR. They achieved a FDR of 75% which is considerably smaller than the one attained by other models.

But this is mainly due to the fact that they used an heterogeneous dataset, meaning that they did not train different networks for different hard drive models. So, it is difficult to compare this results with the ones presented so far.

3 Methods

Describe the method/software/tool/algorithm you have developed here

4 Results

Describe the experimental setup, the used datasets/parameters and the experimental results achieved

5 Discussion

Discuss the results. What is the outcome of your experimetns?

6 Conclusion

Summarize the thesis and provide a outlook on future work.

Bibliography

[1] Steven Cherry. "Edholm's law of bandwidth". In: *IEEE spectrum* 41.7 (2004), pp. 58–60 (cit. on p. 1).

[2] Gordon E Moore. "Cramming more components onto integrated circuits". In: *Proceedings of the IEEE* 86.1 (1998), pp. 82–85 (cit. on p. 1).

[3] Aarne Mammela and Antti Anttonen. "Why Will Computing Power Need Particular Attention in Future Wireless Devices?" In: *IEEE Circuits and Systems Magazine* 17.1 (2017), pp. 12–26. DOI: 10.1109/MCAS.2016.2642679 (cit. on p. 1).

[4] Ling Qian et al. "Cloud computing: An overview". In: *IEEE international conference on cloud computing*. Springer. 2009, pp. 626–631 (cit. on pp. 1, 2).

[5] *Microsoft Datacenters*. https://datacenters.microsoft.com/. Accessed: 2025-05-14. Microsoft (cit. on pp. 1, 2).

[6] Guosai Wang, Lifei Zhang, and Wei Xu. "What Can We Learn from Four Years of Data Center Hardware Failures?" In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2017, pp. 25–36 (cit. on p. 3).

[7] *Communicating With Your SSD - Understanding SMART Attributes*. Tech. rep. Accessed: 2025-05-16. 2013, pp. 24–25 (cit. on pp. 2, 5).

[8] Bingpeng Zhu et al. "Proactive drive failure prediction for large scale storage systems". In: *2013 IEEE 29th symposium on mass storage systems and technologies (MSST)*. IEEE. 2013, pp. 1–5 (cit. on pp. 4, 5, 19, 29, 30).

[9] Chang Xu et al. "Health status assessment and failure prediction for hard drives with recurrent neural networks". In: *IEEE Transactions on Computers* 65.11 (2016), pp. 3502–3508 (cit. on pp. 4, 5, 21, 22).

[10] Jing Shen et al. "Random-forest-based failure prediction for hard disk drives". In: *International Journal of Distributed Sensor Networks* 14.11 (2018) (cit. on pp. 4–6, 15).

[11] Jing Li et al. "Hard drive failure prediction using classification and regression trees". In: *2014 44th annual ieee/ifip international conference on dependable systems and networks*. IEEE. 2014, pp. 383–394 (cit. on pp. 5, 12, 29).

[12] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. "Characterizing cloud computing hardware reliability". In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC '10. Indianapolis, Indiana, USA: Association for Computing Machinery, 2010, pp. 193–204 (cit. on p. 5).

[13] Claude E Shannon. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), pp. 379–423 (cit. on p. 9).

[14] Xue Ying. "An overview of overfitting and its solutions". In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing. 2019 (cit. on p. 13).

[15] Erica Briscoe and Jacob Feldman. "Conceptual complexity and the bias/variance tradeoff". In: *Cognition* 118.1 (2011), pp. 2–16 (cit. on p. 13).

[16] Tin Kam Ho. "Random decision forests". In: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282 (cit. on p. 14).

[17] Anjaneyulu Babu Shaik and Sujatha Srinivasan. "A brief survey on random forest ensembles in classification model". In: *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2018, Volume 2*. Springer. 2019, pp. 253–260 (cit. on p. 14).

[18] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton:(Project Para)*. Cornell Aeronautical Laboratory, 1957 (cit. on p. 16).

[19] Claude Lemaréchal. "Cauchy and the gradient method". In: *Doc Math Extra* 251.254 (2012), p. 10 (cit. on p. 17).

[20] Shun-ichi Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* 5.4-5 (1993), pp. 185–196 (cit. on p. 18).

[21] Dave Steinkraus, Ian Buck, and Patrice Y Simard. "Using GPUs for machine learning algorithms". In: *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE. 2005, pp. 1115–1120 (cit. on p. 18).

[22] MI Jordan. *Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986*. Tech. rep. California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, 1986 (cit. on pp. 19, 20).

[23] Juan Manuel Espinosa-Sanchez, Alex Gomez-Marin, and Fernando de Castro. "The Importance of Cajal's and Lorente de Nó's Neuroscience to the Birth of Cybernetics". In: *The Neuroscientist* 31.1 (2025), pp. 14–30 (cit. on p. 20).

[24] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133 (cit. on p. 20).

[25] Jeffrey L Elman. "Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211 (cit. on pp. 20, 21).

[26] Kanchan M Tarwani and Swathi Edem. "Survey on recurrent neural network in natural language processing". In: *Int. J. Eng. Trends Technol* 48.6 (2017), pp. 301–304 (cit. on pp. 20, 24).

[27] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 21).

[28] Satya Prakash Yadav et al. "Survey on machine learning in speech emotion recognition and vision systems using a recurrent neural network (RNN)". In: *Archives of Computational Methods in Engineering* 29.3 (2022), pp. 1753–1770 (cit. on p. 24).

[29] Sepp Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116 (cit. on p. 24).

[30] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. "The problem of learning long-term dependencies in recurrent networks". In: *IEEE international conference on neural networks*. IEEE. 1993, pp. 1183–1188 (cit. on pp. 24, 25).

[31] Muhammad Raheel Raza, Walayat Hussain, and José Maria Merigó. "Cloud sentiment accuracy comparison using RNN, LSTM and GRU". In: *2021 Innovations in intelligent systems and applications conference (ASYU)*. IEEE. 2021, pp. 1–5 (cit. on p. 25).

[32] Simone A Ludwig. "Comparison of time series approaches applied to greenhouse gas analysis: ANFIS, RNN, and LSTM". In: *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE. 2019, pp. 1–6 (cit. on p. 25).

[33] Zheng Hu, Jiaojiao Zhang, and Yun Ge. "Handling vanishing gradient problem using artificial derivative". In: *IEEE Access* 9 (2021), pp. 22371–22377 (cit. on p. 25).

[34] Sk M Rahman et al. "Nonintrusive reduced order modeling framework for quasi-geostrophic turbulence". In: *Physical Review E* 100.5 (2019), p. 053306 (cit. on p. 26).

[35] Shengdong Zhang et al. "Deep learning on symbolic representations for large-scale heterogeneous time-series event prediction". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 5970–5974 (cit. on pp. 27, 28).

[36] Anwesha Das et al. "Desh: deep learning for system health prediction of lead times to failure in hpc". In: *Proceedings of the 27th international symposium on high-performance parallel and distributed computing*. 2018, pp. 40–51 (cit. on p. 29).

[37] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning* 20 (1995) (cit. on p. 30).

[38] Keiron O'shea and Ryan Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015) (cit. on p. 31).

[39] Osman Semih Kayhan and Jan C van Gemert. "On translation invariance in cnns: Convolutional layers can exploit absolute spatial location". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 14274–14285 (cit. on p. 31).

[40] Xiaoyi Sun et al. "System-level hardware failure prediction using deep learning". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. 2019, pp. 1–6 (cit. on p. 31).

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie, dass ich die Masterarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, July 30, 2025

Miguel Vieira Pereira