Deep Learning for Predictive Maintenance in Impoundment Hydropower Plants

Maxime Rutagarama

Ecole polytechnique fédérale de Lausanne
Laboratory of Artificial Intelligence
Supervisor: Prof. Boi Faltings

Alpiq AG
Asset Management Support
Supervisor: Nicolas Rouge

This dissertation is submitted for the degree of
Master in Computer Science

March 2019

Abstract

Predictive maintenance aims to anticipate industrial system failures in order to optimize maintenance schedules, and thus the ratio between expenses and revenues. This approach is particularly interesting for the hydroelectric production sector, due to its inherent heavy costs and challenges. By using machine learning, and in particular deep learning, to understand the behaviour of machines using data collected by the sensors, we are able to automatically detect early warning signs of potential failures. Given the lack of incident data, it is preferable to apply semi-supervised learning methods, which learn to identify the normal behaviour of the system, and then identify anomalies. By explicitly simulating the expected behaviour of the machines, it will even be possible to obtain interpretable predictions, which allows the operator to identify the source of the problem and act accordingly. Although this work focuses on hydropower plants, it is intended to establish a predictive maintenance framework applicable to all industries.

Table of contents

Chapter 1

Introduction: Impoundment Hydropower Plants Need Smart Maintenance

Since the early 2010's, the industrial sector has the will to digitize itself, and to integrate the latest advances in data analysis and machine learning, to improve efficiency and reduce costs. This is included in the scope of what is called Industry 4.0, that is, the fourth industrial revolution after mechanization in 1784, electrical power along with mass production in 1870, and automated production in 1969. Most of the industrial machines are equipped with sensors and their data are collected on servers, forming the Industrial Internet of Things (IIoT). With this new paradigm come a multitude of new possibilities to optimize the functioning of industrial facilities, and in particular maintenance programs. As a heavy industry requiring complex and critical machinery, the field of hydroelectricity is no exception.

1.1 The criticality of hydropower plants

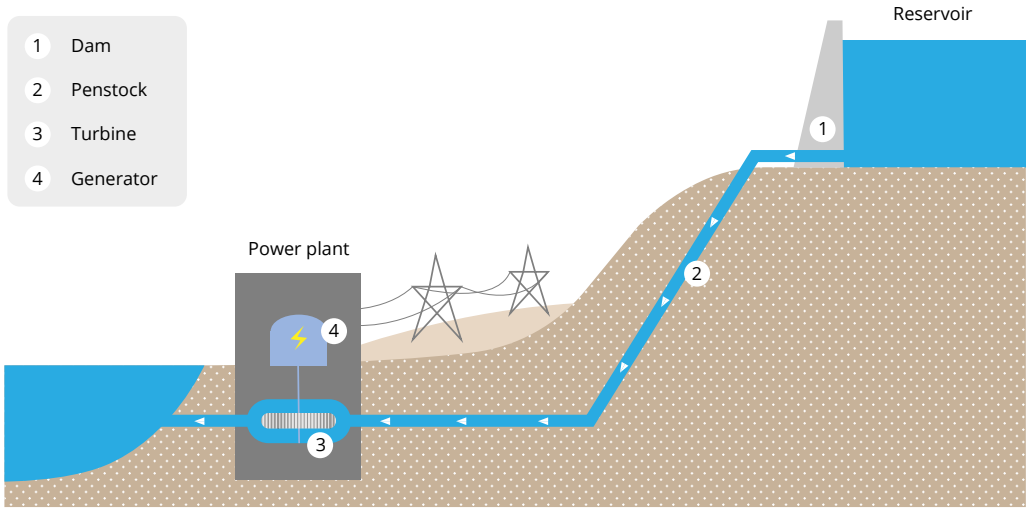
Hydropower is the world's leading renewable energy source. It uses the movement of water as an energy source in the following way: water is captured and sent to a turbine, which converts its kinetic energy into mechanical energy, then a generator converts this mechanical energy into electrical energy.

When the facility has no storage capacity and turbines the water as it comes naturally, it is called a run-of-river power plant.

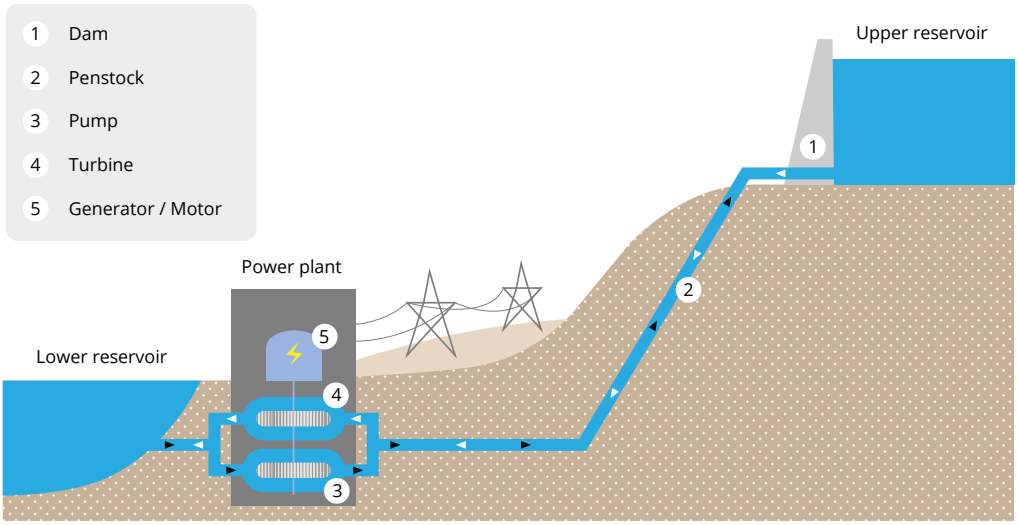
Conversely, a hydropower plant is called an impoundment facility if it uses a dam to store water in a reservoir. The purpose of this type of facility is to store energy in the form of water

that can be transformed into electricity at the right time. The stored water can be released at any time into a pipe that leads to a factory below. Its potential energy is then transformed into kinetic energy so that it hits the turbine with high speed. .

Some impoundment hydropower plants allow water to be pumped into the storage reservoir from a lower reservoir: these are pumped-storage plants. The goal of these facilities is to pump water when energy demand is low, so that it can be turbined later to produce energy when demand is high. The operation of a classical impoundment plant and of a pumped-storage plant are shown respectively in Figure 1.1a and Figure 1.1b.



(a) Classical impoundment plant



(b) Pumped-storage plant

Fig. 1.1 Operation of impoundment hydropower plants.

The issues related to hydropower plants are multiple and crucial, which is why it is necessary to ensure their proper functioning as much as possible. On the one hand, incidents that make power plants unavailable can have a significant economic impact, due to the importance of hydroelectric production in the energy panel. On the other hand, the most critical incidents can be catastrophic for the environment and people. Maintenance is therefore a fundamental task. However, since machines are very heavy and expensive, and unavailability is costly, maintenance operations must be planned in the smartest way possible. This is all the more true for impoundment facilities, since the stress on the machines is irregular over time and their fatigue is therefore more difficult to anticipate than that of run-of-river plants. That is why in this work we only considered impoundment facilities although the discussed methods are also valid for other types of power plants.

For more information on the hydropower domain as a whole, one can refer to [23] for a book in english, [25] for a book in french, or [7] for a more technical reference book in German.

1.2 From corrective maintenance to predictive maintenance

According to the norm NF EN 13306 X 60-319, maintenance is the "combination of all technical, administrative and managerial actions during the life cycle of an item intended to retain it in, or restore it to, a state in which it can perform the required function." These actions aim to ensure the reliability of machines but they have a cost: this is the cost/reliability trade-off. Hence, it is crucial to perform smart maintenance plans, trying to maximize machines reliability while minimizing maintenance costs. That is why industry opts increasingly for smarter maintenance strategies.

Corrective maintenance. Maintenance is considered corrective if it is "carried out after fault recognition" (NF EN 13306 X 60-319). This strategy minimizes the maintenance costs since actions are most of the time carried out only when a failure occurs, that is when the machine is unable to perform a required function. On the other hand, it does not ensure well machines reliability because nothing is done to prevent failures to occur.

Preventive maintenance. In order to be proactive and to act before a failure occurs, one can choose to perform preventive maintenance, that is maintenance "carried out at predetermined intervals or according to prescribed criteria" (NF EN 13306 X 60-319). For example, periodic maintenance improves reliability but implies recurrent costs that are not always necessary.

Predictive maintenance. The best option would be to perform maintenance operations only when the machine really needs it, that is when a failure is likely to occur. That is the idea of predictive maintenance, which is a strategy of maintenance "carried out following a forecast derived from the analysis and evaluation of the significant parameters of the degradation of the item" (NF EN 13306 X 60-319). By analyzing machines data, we should be able to evaluate and predict the evolution of their state, in order to act accordingly.

A first step for predictive maintenance is to set threshold to trigger alarms on sensors or combinations of sensors. But we could analyze the data much more finely and trigger smarter alarms, which capture problems that are indistinguishable with simple thresholds. Some papers already present methods for predictive maintenance in hydropower plants ([35], [34], [22]), but our goal is to offer solutions using the latest technologies in machine learning.

1.3 Machine learning for predictive maintenance

As seen previously, predictive maintenance consists in the evaluation of a system's state based on significant data. The goal is to predict this state automatically, and at this time, the most efficient tool to perform prediction from observed data is undeniably machine learning.

Machine learning is a branch of artificial intelligence that uses statistical approaches to give computers the ability to learn from data. Since this document is also intended to serve asset managers that are not necessarily familiar with machine learning and computer science, we will try not to take too much for granted. To have a good comprehension of the field, we recommend reading [1], which is a reference for machine learning.

In the case of predictive maintenance, the main source of data is time series coming from machines sensors. A time series is a series of numerical values representing the evolution of a specific quantity over time. Formally, it takes the form of a set $\{\boldsymbol{x}^{(t)}\}_{t\in\mathcal{T}}$, in which each element is a vector of Q measurements. If the current time is T, the idea is to learn from past observations $\{\boldsymbol{x}^{(t)}\}_{t\leq T}$ in order to evaluate the system's state when new measurements $\{\boldsymbol{x}^{(t)}\}_{t>T}$ are available. The various possible approaches will be described and explored in the second chapter of this report.

Machine learning is a very powerful tool that is being used in every industry domains, but it has some drawbacks. Unlike physical models, machine learning models are based solely on data and therefore tend to be black boxes. A machine learning model will in most cases only give a prediction, but will not explain it. This is not enough for many real-world industrial applications, especially when it comes to security [4].

When the algorithm outputs an alarm, the person in charge of the machine has to act accordingly, which is very difficult if the source of the problem is unknown. For this reason,

it is essential that the model provide insights on the nature of the problem. In addition, the interpretability of a model is the main key to a successful production launch. Indeed, the model will be much more accepted if its predictions are understandable, that is if users are able to see logic in them. In addition, it is much easier to debug or audit a model if it is interpretable. Hence, developing interpretability ensures better reliability and robustness and helps building trust, but it forces us to think beyond the usual performance indicators, to empathize with the user, and to choose carefully the methods and model architectures used. This will be one of the main challenges of this work.

1.4 Document structure

The second chapter will focus on the theory behind the machine learning methods for predictive maintenance. We will describe, explain and evaluate available approaches.

In the third chapter, we will apply the methods described previously to real data provided by the Swiss energy company Alpiq. We will train different models and show how they act on normal behaviors, along with anomalies and failures data. Each method's effectiveness will be evaluated according to several criteria, taking into account their performance, but also their relevance in an operational industrial environment, notably through their interpretability.

Finally, in the fourth and last chapter, we will draw conclusions from the experiences of this project.

Chapter 2

Theory: Machine Learning for Predictive Maintenance

The purpose of this chapter is to explain the machine learning concepts and methods used in this project. The chapter begins with an overview of available machine learning approaches for predictive maintenance. We discuss which approach we chose over the others for our case, that is for impoundment hydropower plants, and motivate our choice. We then present classical algorithms to implement this approach and discuss their efficiency. Finally, we introduce deep learning and its application to predictive maintenance, which are the methods we have focused on during this project.

2.1 Problem framing

From now, we will consider the following setup. Let $\{\boldsymbol{u}_t\}_{t=1}^{T}$, $\boldsymbol{u}_t \in \mathbb{R}^Q$ be a time series gathering past sensors measurements of an industrial system, collected on a regular time interval. Our goal is to predict, given continuously incoming new measurements $\{\boldsymbol{u}_t\}_{t>T}$, if a failure is likely to happen, or at least to detect potential harbingers of a failure. This problem can be framed in various ways, which we will describe below.

2.1.1 Limitations of supervised learning

An intuitive way to proceed would be to use the most common approach in machine learning, namely supervised learning. By observing exemplary functioning cases and failure cases, an algorithm could learn to distinguish between risky behaviours and normal behaviours.

A task is said to be supervised when data is provided as pairs of inputs and desired outputs, called labels. Given a set $\mathbb{X} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{N}$, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^Q$ are the inputs and $\boldsymbol{y}^{(i)} \in \mathbb{R}^D$ the labels, a supervised learning algorithm will try to approximate the relation $\boldsymbol{y}^{(i)} = f(\boldsymbol{x}^{(i)})\ \forall i \in [N]$. In the case of predictive maintenance, the labels should naturally come from the observed cases of failure.

Let us say that we observed K failures, that happened at times $\{t_1, \ldots, t_K\}$, and suppose that harbingers generally show up within the F time steps preceding a failure. We could frame the problem as asking the question "will the system fail in the next F time steps ?" like in [16]. The labels are then:

$$y^{(t)} = \begin{cases} 1, & \text{if } \exists\ i \text{ such that } 0 \leq t_i - t \leq F, \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Another way to proceed would be to predict the actual time before the next failure, also called the remaining useful life (RUL) [10]. This is a supervised regression task where the labels are set as $y^{(t)} = t_i - t$, where $i = \min\{j : t_j > t\}$.

Then, one only has to use any supervised learning algorithm such as SVM, random forests, or neural networks, to learn the relation between the inputs $\boldsymbol{x}^{(t)} := \boldsymbol{u}_t$ and the labels $\boldsymbol{y}^{(t)}$. In order to detect defective behaviours observable over time, and not punctually, it is necessary to consider sequences as input to the model, and no longer only punctual measurements. Thus we can instead use as inputs sub-sequences of size S, such that $\boldsymbol{x}^{(t)} = (\boldsymbol{u}_s)_{s=t}^{t+S-1}$[1]. This method, called sliding window, is illustrated in Figure 2.1.

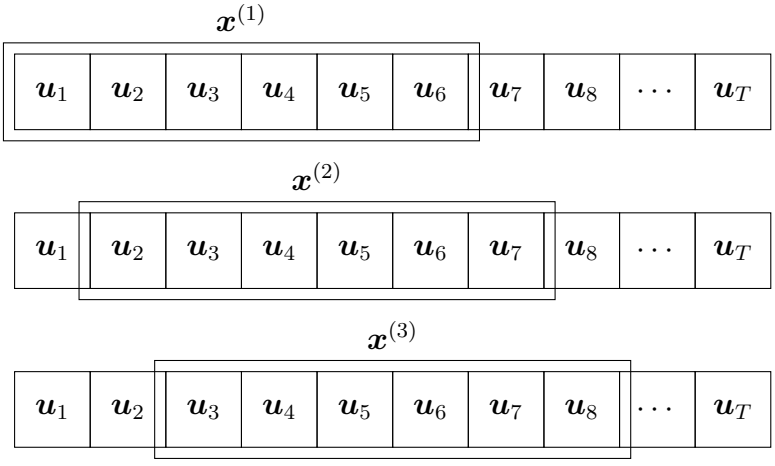


Fig. 2.1 Sliding window method

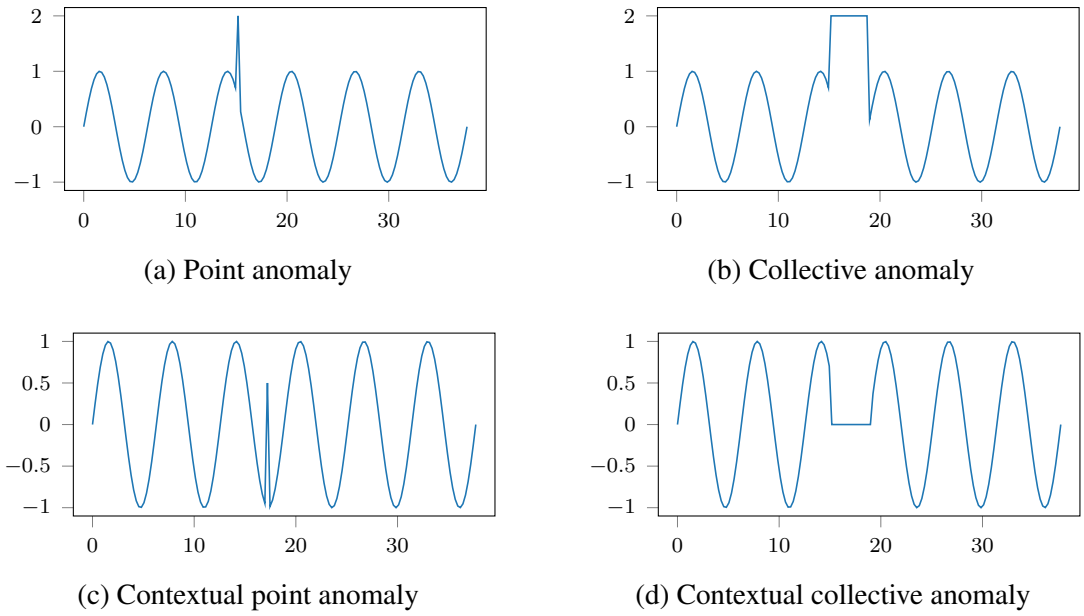
[1] If that produces too much sequences to process, one can introduce a stride value R, and take only one out of R consecutive sequences.

The drawbacks of these supervised approaches are on one hand that all data have to be labelled, and on another hand that we need enough examples of each possible behaviour to allow proper training. Unfortunately, in the case of predictive maintenance, this is often not the case. In particular, in hydropower plants, even if failures are all identified and labelled as such, they are happily rare, and they appear to be most of the time very different from one another. This is why supervised learning is not appropriate in our case. We would rather use other approaches to address the problem of unbalanced data and the diversity of failure cases.

2.1.2 Predictive maintenance as a semi-supervised anomaly detection task

Since it is difficult to learn to identify all possible cases of failure with the available data, the problem can be viewed from a different perspective. The idea is to model the normal behaviour of the system and detect behaviours that deviate too much from it [6]. This is called a semi-supervised learning task since we use data from one class only, that is the normal behaviours samples. The goal is no longer to predict failures, but to detect anomalies and trigger an alarm as these are potentially early signs of a failure. Hence we frame the problem as an anomaly detection task.

More specifically, we want to detect anomalies in a multivariate time series, which is a problem that has been widely studied. The first step is to ask ourselves what should be called an anomaly. Generally, anomalies in time series are divided according to two criteria [2]. First of all, an anomaly can be punctual or collective, depending on whether it concerns only one observation at a given time, or on the contrary a sequence of observations. Then, an anomaly can be global, if it concerns outliers with respect to the whole time series, or it can be contextual, if the values are incoherent at the scale of a certain neighbourhood only. These two criteria can be combined, and the major challenge is to detect contextual collective anomalies, which are often the most subtle but also the most relevant. Figure 2.2 shows the four possible types of time series anomalies that we just discussed. Note that these graphs show sequences of dimension one, whereas we usually look at multidimensional time series, but the principle remains the same.



(a) Point anomaly

(b) Collective anomaly

(c) Contextual point anomaly

(d) Contextual collective anomaly

Fig. 2.2 Different types of anomalies in time series.

Since our area of interest is predictive maintenance, anomalies must be detected in real time, so that we can identify the problem and react accordingly immediately. To summarize, the idea is to frame the problem as a semi-supervised online anomaly detection task. From now, we will assume our dataset $\{\boldsymbol{u}_t\}_{t=1}^{T}$, $\boldsymbol{u}_t \in \mathbb{R}^Q$ to contain only observations coming from normal behaviour of the system.

2.1.3 Classical algorithms for semi-supervised anomaly detection

Now that the problem has a well-defined frame, it remains to define the algorithms that we want to use. Anomaly detection is a task that has been widely studied for various applications, from bank fraud prevention to intrusion detection, and many classical techniques have been developed. As we will see later, these techniques are not well suited for our case, and even if this document focuses on deep learning, we considered important to introduce these methods, in order to be comprehensive and to justify the use of deep learning.

Density estimation. Given normal data samples $\boldsymbol{x}^{(i)} \in \mathbb{R}^Q,\ i \in [N]$, we can construct an estimate of the underlying probability density function (PDF), and classify new observations that lie in a low-density region as anomalies.

The most famous density estimator, which is very popular in data visualisation, and dates from the 19$^{\text{th}}$ century, is the histogram. A histogram divides the input space $\mathbb{R}^Q$ into small

hypercubes and counts the number of observations lying in each hypercube. It approximates the probability density function as:

$$\hat{f}(\boldsymbol{x}) := \frac{1}{Th^Q} \sum_{i=1}^{N} \mathbb{1}_{C_h}(\boldsymbol{x}^{(i)} - \boldsymbol{x}) \tag{2.2}$$

where $h \in \mathbb{R}^+$ is the size of the hypercubes, and $\mathbb{1}_{C_h}$ is the indicator function[2]:

$$\mathbb{1}_{C_h}(\boldsymbol{y}) = \begin{cases} 1, & \text{if } \boldsymbol{z} \in C_h = \{\boldsymbol{z} : \|\boldsymbol{z}\|_\infty \leq \frac{h}{2}\}, \\ 0, & \text{otherwise.} \end{cases}$$

But the histogram estimator builds a step probability density function, which is not smooth. To address this issue, one can replace the indicator function in formula 2.2 by a more general kernel function.

This is what Kernel Density Estimation (KDE), also called Parzen-Rosenblatt estimation, aims to do, by approximating the probability density function as:

$$\hat{f}(\boldsymbol{x}) := \frac{1}{N} \sum_{i=1}^{N} K_{\mathbf{H}}(\boldsymbol{x}^{(i)} - \boldsymbol{x}) \tag{2.3}$$

where $K : \mathbb{R}^Q \to \mathbb{R}^+$ is the kernel function, that is a symmetric density function[3], $\mathbf{H} \in \mathbb{R}^{Q \times Q}$ is a symmetric positive-definite[4] matrix corresponding to the bandwidth, controlling the smoothing of the density estimation and $K_{\mathbf{H}}$ is defined as $K_{\mathbf{H}}(\boldsymbol{x}) := |\mathbf{H}|^{-1/2} K(\mathbf{H}^{-1/2}\boldsymbol{x})$. The choice of the kernel function is not crucial, and we usually use the Gaussian kernel $K_{\mathbf{H}}(\boldsymbol{x}) = (2\pi)^{-d/2}|\mathbf{H}|^{-1/2} e^{-\frac{1}{2}\boldsymbol{x}^\top \mathbf{H}^{-1}\boldsymbol{x}}$.

For more information, [27] and [28] describe these methods with more details, as well as other non-parametric density estimation techniques.

These density estimation methods are called non-parametric, since we make no assumption on the underlying distribution. But we could also assume that the data was generated by a specific distribution, or combination of distributions, with unknown parameters. This is what mixture models [21] aim to do. The most common mixture model is the Gaussian Mixture Model (GMM), that assumes that our observations were generated by a combination of K normal distributions. The probability density function is then written as:

$$\hat{f}(\boldsymbol{x}) := \sum_{i=1}^{K} \phi_i \mathcal{N}(\boldsymbol{x} | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \tag{2.4}$$

[2]The infinity norm $\|\cdot\|_\infty$ is defined as $\|\boldsymbol{z}\|_\infty := \max(|z_1|, \ldots, |z_Q|)$, $\forall \boldsymbol{y} = (z_1, \ldots, z_Q) \in \mathbb{R}^Q$
[3]This means we require $K(\boldsymbol{x}) > 0$, $K(-\boldsymbol{x}) = K(\boldsymbol{x})\ \forall \boldsymbol{x} \in \mathbb{R}^Q$, and $\int K(\boldsymbol{x}) d\boldsymbol{x} = 1$,
[4]This means $\boldsymbol{x}^\top \boldsymbol{H} \boldsymbol{x} > 0,\ \forall \boldsymbol{x} \in \mathbb{R}^Q$

where $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ is the i^{th} multivariate normal distribution of mean $\boldsymbol{\mu}_i$ and covariance matrix $\boldsymbol{\Sigma}_i$,

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{(2\pi)^{Q/2}\left|\boldsymbol{\Sigma}\right|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1}(x-\boldsymbol{\mu}_i)\right), \tag{2.5}$$

and $\phi_i,\ i = 1, \ldots, K$ are the weights attributed to each distribution.

The parameters $\boldsymbol{\Theta} := (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K, \phi_1, \ldots, \phi_K)$ are usually learned using the Expectation Maximization (EM) algorithm. EM is not detailed here, so one can read [24] to learn more about it, and to have a more in depth explanation of GMM.

Clustering. Another way to proceed is to group observations by similarity, to create clusters. We can then identify new inputs that lie too far away from previously built clusters as anomalies. Clustering is basically the natural implementation of unsupervised learning, and an overview of the field can be found in [12]. We will here introduce only two clustering algorithms but the other existing ones are also relevant.

k-means, originally proposed in [18], is probably the most popular clustering method. Starting from k given vectors, the algorithms iterates until it finds optimal values for the cluster centers. Once the algorithm has converged, we obtain k clusters defined by their means $M = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_k)$. The Algorithm 1 describes this procedure. For each new observation $\boldsymbol{z}$ we can evaluate its anomaly score by looking at its distance to the closest cluster $d_M(\boldsymbol{z}) := \min_j \|\boldsymbol{z} - \boldsymbol{m}_j\|$.

Algorithm 1 The k-means algorithm

function k-MEANS($\{\boldsymbol{x}^{(t)},\ t = 0, \ldots, T\},\ k$)
  $(\boldsymbol{m}_1, \ldots, \boldsymbol{m}_k) \leftarrow$ GETINITIALMEANS($\{\boldsymbol{x}^{(t)},\ t = 0, \ldots, T\},\ k$)
  repeat
    for $t \leftarrow 0$ to T **do**
      $c^{(t)} \leftarrow \operatorname{argmin}_j \|\boldsymbol{x}^{(t)} - \boldsymbol{m}_j\|^2$ ▷ Compute clusters
    end for
    for $i \leftarrow 1$ to k **do**
      $C_i \leftarrow \{t : c^{(t)} = i\}$
      $\boldsymbol{m}_i \leftarrow \frac{\sum_{t=0}^{T} \mathbb{1}_{C_i}(t) \cdot \boldsymbol{x}^{(t)}}{\sum_{t=0}^{T} \mathbb{1}_{C_i}(t)\}}$ ▷ Update cluster means
    end for
  until cluster means have converged
end function

We could also use a clustering algorithm that is density-based, unlike k-means which is centroid-based, for example density-based spatial clustering of applications with noise (DBSCAN). Given two parameters m and ϵ, DBSCAN builds clusters around so-called core points, that have more than m points lying in their ϵ-neighbourhood. Once clusters $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ are built, one can use the distance to the closest cluster point:

$$d_{\mathcal{C}}(\boldsymbol{z}) = \min_{\boldsymbol{x}^{(i)} \in \bigcup_j C_j} \left\| \boldsymbol{z} - \boldsymbol{x}^{(i)} \right\| \tag{2.6}$$

as the anomaly score. DBSCAN algorithm is not detailed here, and more information can be found in the original paper [5].

One class SVM. A first version of Support Vector Machine (SVM) for novelty detection was introduced in 2000 [26]. The idea is to map via a function $\boldsymbol{\Phi} : \mathbb{R}^Q \to F$ the normal observations to a feature space F and separate them from the origin with a hyperplane in F, whose distance to the origin is maximized. We choose $\boldsymbol{\Phi}$ with the kernel trick, by defining a kernel function such that $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{\Phi}(\boldsymbol{x}) \cdot \boldsymbol{\Phi}(\boldsymbol{y})$. This gives us a decision function, able to classify new observations as anomalies if they lie on the wrong side of the hyperplane in F. Thus we search for the optimal hyperplane parameters $\boldsymbol{w}$ and ρ solving

$$\begin{aligned} \min_{\boldsymbol{w} \in F, \boldsymbol{\xi} \in \mathbb{R}^N, \rho \in \mathbb{R}} \quad & \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{1}{\nu N} \sum_{i=1}^{N} \xi_i - \rho \\ \text{subject to} \quad & \left(\boldsymbol{w} \cdot \boldsymbol{\Phi}(\boldsymbol{x}^{(i)}) \right) \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad \forall i \in [N]. \end{aligned} \tag{2.7}$$

The vector $\boldsymbol{\xi}$ contains slack variables allowing some observation to lie on the wrong size of the hyperplane, that are penalized in the objective function. The parameter $\nu \in (0, 1)$ is controlling the trade-off between the distance minimization and the regularization term. Then, the decision function is

$$f(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{w} \cdot \boldsymbol{\Phi}(\boldsymbol{x}) - \rho) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t \kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}) - \rho \right), \tag{2.8}$$

where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_T)$ is the solution of the lagrangian dual problem

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}), \\ \text{subject to} \quad & 0 \leq \alpha_i \leq \frac{1}{\nu N}, \ \forall i \in [N], \ \sum_{i=1}^{N} \alpha_i = 1. \end{aligned} \tag{2.9}$$

Another approach, proposed in 2004 [31] uses a hypersphere boundary instead of a hyperplane. The minimization problem becomes

$$\begin{aligned} \min_{R\in\mathbb{R},\boldsymbol{\xi}\in\mathbb{R}^N,\boldsymbol{c}\in F} \quad & R^2 + \frac{1}{\nu N}\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & \left\|\boldsymbol{\Phi}(\boldsymbol{x}^{(i)}) - \boldsymbol{c}\right\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0 \quad \forall i \in [N]. \end{aligned} \tag{2.10}$$

The decision function checks if the point lies in the hypershpere:

$$f(\boldsymbol{x}) = \operatorname{sign}\left(R^2 - \|\boldsymbol{\Phi}(\boldsymbol{x}) - \boldsymbol{c}\|^2\right) \tag{2.11}$$

$$= \operatorname{sign}\left(R^2 - \boldsymbol{\Phi}(\boldsymbol{x})\cdot\boldsymbol{\Phi}(\boldsymbol{x}) + 2\boldsymbol{\Phi}(\boldsymbol{x})\cdot\boldsymbol{c} - \boldsymbol{c}\cdot\boldsymbol{c}\right) \tag{2.12}$$

$$= \operatorname{sign}\left(R^2 - \kappa(\boldsymbol{x},\boldsymbol{x}) + 2\sum_{i=1}^{N}\alpha_i\kappa(\boldsymbol{x}^{(i)},\boldsymbol{x}) - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\kappa(\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)})\right) \tag{2.13}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)$ is the solution of the lagrangian dual problem

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{N}\alpha_i\kappa(\boldsymbol{x}^{(i)},\boldsymbol{x}^{(i)}) - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\kappa(\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq \frac{1}{\nu N},\ \forall i \in [N],\ \sum_{i=1}^{N}\alpha_i = 1. \end{aligned} \tag{2.14}$$

Generally, we use a Radial Basis Function (RBF) kernel, also called Gaussian kernel, defined by $\kappa_{\text{RBF}}(\boldsymbol{x},\boldsymbol{y}) := e^{-\gamma\|\boldsymbol{x}-\boldsymbol{y}\|^2}$, where γ is the spread of the kernel.

All the previously discussed algorithms (density estimators, clustering and one-class SVM) are tried-and-tested methods for semi-supervised anomaly detection, but they have some limitations, especially in our case. First of all, if we use the punctual observations $\boldsymbol{x}^{(t)} = \boldsymbol{u}_t$ as input of the models, this does not allow seeing anomalies that are time dependent, namely contextual and collective anomalies. As it is, these algorithms will only detect punctual global anomalies, which is not sufficient at all. A workaround would be to use as input, not only punctual observations, but sliding window sub-sequences of size S, that is $\boldsymbol{x}^{(t)} = \boldsymbol{u}_t \oplus \boldsymbol{u}_{t+1} \oplus \ldots \oplus \boldsymbol{u}_{t+S-1}$ (c.f. Figure 2.1)[5]. But, since we already focus on multivariate times series, with dimension Q possibly of order 2 or 3, if we use the sliding window technique, the dimension is multiplied by the size of the window, and can become pretty big. This brings up another problem, which is the curse of dimensionality. All these methods suffer from this curse, which means they perform bad on high-dimensional spaces, because the value added by additional dimensions is much smaller compared to overhead

[5] $\oplus$ describes the concatenation operation, $(x_1,\ldots,x_m)\oplus(y_1,\ldots,y_n) := (x_1,\ldots,x_m,y_1,\ldots,y_n)$.

it adds to the algorithm, which is exponential. In the next section, we will explore a set of algorithms that are able to handle time series inputs and that manage to overcome the curse of dimensionality.

2.2 Semi-supervised deep learning for predictive maintenance

Deep learning is a branch of machine learning that relies on a well known biomimetic system: deep artificial neural networks. Created in the 1950's, it has developed exponentially since the 2010's, thanks to new architectures and techniques and especially thanks to the strong increasing of the computing power. Deep learning has quickly outperformed other algorithms in many areas, and anomaly detection is no exception. In this section we will first of all introduce the field of deep learning through a technical overview. We will then present a one-class neural network classifier called autoencoder, and finally show how we can do forecasting-based semi-supervised anomaly detection with deep neural networks. To have a deeper understanding of deep learning as a whole, we recommend [15] and [8].

2.2.1 An overview of Deep Learning

As in Section 2.1.1 for supervised learning, let us consider the set $\mathbb{X} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{N}$, $\boldsymbol{x}^{(i)} \in \mathbb{R}^Q$ being the inputs and $\boldsymbol{y}^{(i)} \in \mathbb{R}^D$ being the labels. We suppose there exists a function $f : \mathbb{R}^Q \to \mathbb{R}^D$ describing the underlying phenomenon, such that the data is generated according to

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon(\boldsymbol{x}), \tag{2.15}$$

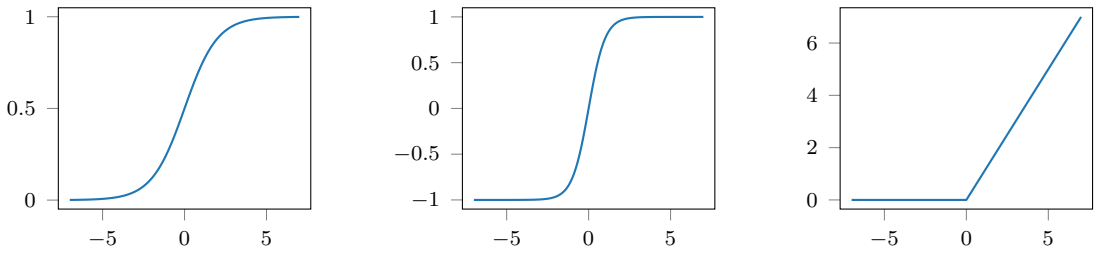
$\epsilon : \mathbb{R}^Q \to \mathbb{R}^D$ describing a potential observation noise. We are trying to approximate f by a function $\hat{f}$ using only the information given by the samples of E.

A linear model [29] could do the job by setting $\hat{f}(\boldsymbol{x}) = \boldsymbol{x}\mathbf{W} + \boldsymbol{b}$ where $\mathbf{W} \in \mathbb{R}^{Q \times D}$ is the weights matrix and $\boldsymbol{b} \in \mathbb{R}^D$ is the bias. These parameters can then be found by minimizing a loss function, which is basically a measure of the error between the real data $\boldsymbol{y}^{(i)}$ and the predictions $\hat{\boldsymbol{y}}^{(i)} := \hat{f}(\boldsymbol{x}^{(i)})$, typically the mean squared error:

$$\text{MSE} := \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)} \right\|^2. \tag{2.16}$$

But most of the real world processes are highly non-linear, so we need to consider more complex approximation functions.

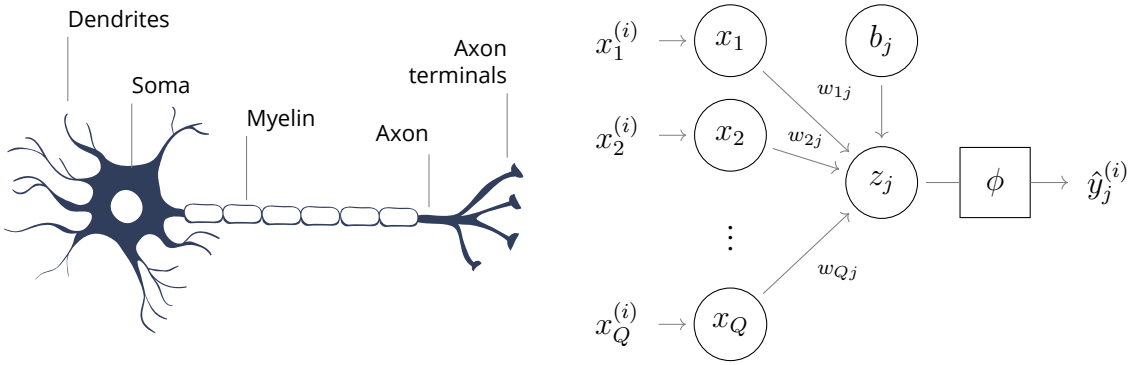
As a first step we can apply some non-linear activation function ϕ to the linear model such that $\hat{f}(\boldsymbol{x}) = \phi(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})$. Some popular activation functions are shown in Figure 2.3.



(a) Sigmoid (b) Tanh (c) ReLU

Fig. 2.3 Usual activation functions.

This functional system is the basis of artificial neural networks, since it roughly models the structure of biological neurons as we can see in Figure 2.4.



(a) A biological neuron (b) An artificial neuron

Fig. 2.4 A biological neuron compared to an artificial neuron.

An artificial neural network is the combination of several of these artificial neurons, where the output of a neuron is the input of another, such that a hidden layer of N_1 neurons is created between the input and the output layers. The approximation function hence becomes

$$\hat{f}(\boldsymbol{x}) = \phi^{[2]}\left(\phi^{[1]}\left(\boldsymbol{x}\boldsymbol{W}^{[1]} + \boldsymbol{b}^{[1]}\right)\boldsymbol{W}^{[2]} + \boldsymbol{b}^{[2]}\right), \tag{2.17}$$

where $\boldsymbol{W}^{[1]} \in \mathbb{R}^{Q\times N_1}$, $\boldsymbol{b}^{[1]} \in \mathbb{R}^{N_1}$, $\boldsymbol{W}^{[2]} \in \mathbb{R}^{N_1\times D}$, $\boldsymbol{b}^{[2]} \in \mathbb{R}^{D}$. It is proven that this class of approximation functions are able to approximate any continuous function, as long as

N_1 is big enough, which means neural networks with one hidden layer only are universal approximators.

Now deep learning is simply about adding more hidden layers to the network, which allows to model more easily complex functions. A neural network is usually said to be deep if it has $L > 1$ hidden layers. From now we will use the following notations:

- L is the number of hidden layers of the network,
- N_l is the number of neurons in the l^{th} layer, with $N_0 = Q$ and $N_{L+1} = D$,
- $\boldsymbol{a}^{[l]} := \phi^{[l]}\left(\boldsymbol{a}^{[l-1]}\boldsymbol{W} + \boldsymbol{b}^{[l]}\right),\ l = 1, \ldots, L+1$, is the output of layer l,
- $\boldsymbol{x} = \boldsymbol{a}^{[0]}$ is the input layer and $\hat{\boldsymbol{y}} = \boldsymbol{a}^{[L]}$ the output layer,
- $w_{ij}^{[l]} = \boldsymbol{W}_{(ij)}^{[l]}$ is the weight between i^{th} neuron of layer $l-1$ and j^{th} neuron of layer l,
- $b_j^{[l]}$ is the bias of the j^{th} neuron of layer l.

Let $\hat{f}^{[l]}(\boldsymbol{x}) = \phi^{[l]}(\boldsymbol{x}\boldsymbol{W}^{[l]} + \boldsymbol{b}^{[l]})$ be the function corresponding to the l^{th} layer of the network. Then the general formula for a deep neural network can be written as

$$\hat{f}(\boldsymbol{x}) = (\hat{f}^{[L+1]} \circ \hat{f}^{[L]} \circ \ldots \circ \hat{f}^{[1]})(\boldsymbol{x}). \tag{2.18}$$

Figure 2.5 shows an example of such a deep neural network.

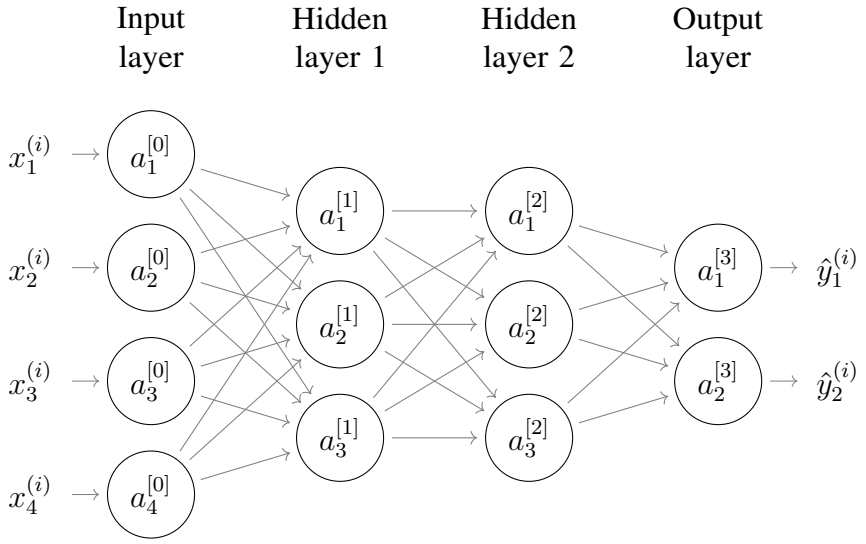


Fig. 2.5 A deep artificial neural network with $L = 2,\ N_0 = 4,\ N_1 = N_2 = 3,\ N_4 = 2$. We omitted the weights and the biases for aesthetic reason.

Let $\boldsymbol{\Theta}$ be the set of all the weights and bias values of a neural etwork:

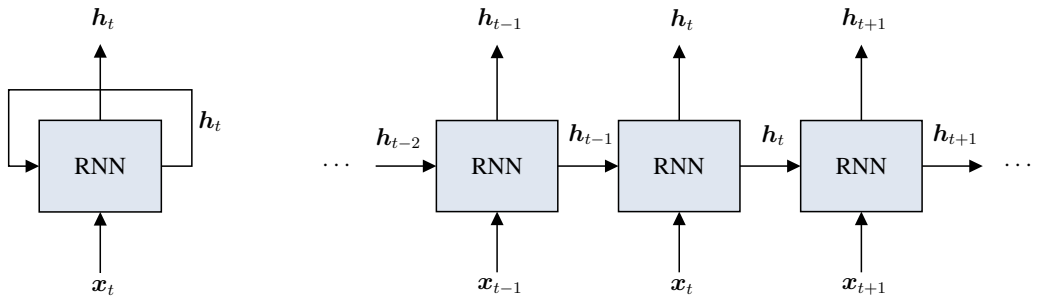
$$\boldsymbol{\Theta} = \left\{ w_{ij}^{[l]} : \; l \in [L+1], \; i \in [N_{l-1}], \; j \in [N_l] \right\} \cup \left\{ b_j^{[l]} : \; l \in [L+1], \; j \in [N_l] \right\}.$$

The goal is to tune these paramaeters in order to minimize a loss function $\mathcal{L}(\boldsymbol{\Theta})$ (for example the MSE function defined in Equation 2.16). This training process is iterative, and consists in changing the parameters step by step in the opposite direction of the gradient $\nabla \mathcal{L}(\boldsymbol{\Theta})$, that is the partial derivatives of the loss with respect to each parameter. The gradient is computed with the backpropagation algorithm. More information is given on neural networks learning process in Appendix A. Now that we have introduced the core concepts of deep learning, we will focus on some neural network architectures, that are crucial when it comes to time series data.

Artificial neural networks as those we have previously discussed are called feedforward neural network, because the information goes straight from input nodes to output nodes, through hidden nodes, without any cycle. As for classical machine learning algorithms presented in Section 2.1.3, this does not allow to take in account temporal dependencies inherent to time series data. Recurrent neural networks (RNN) were designed to address this problematic, by naturally handling temporal sequences as input. The idea is to transform the neurons of the network into RNN cells, by adding to them a hidden state and a recurrent connection with themselves. When consecutive observation of a sequence arrive in the network, each cell takes also as input the previous hidden state transmitted by the recurrent connection, and outputs the new hidden state. The computation of the new hidden state is made with weights shared across time, that are also learned during training:

$$\boldsymbol{h}_t = \phi_h(\boldsymbol{W}_h \boldsymbol{x}_t + \boldsymbol{U}_h \boldsymbol{h}_{t-1} + \boldsymbol{b}_h), \tag{2.19}$$

where $\boldsymbol{h}_t \in \mathbb{R}^h$ is the hidden state, $\boldsymbol{W}_h \in \mathbb{R}^{h \times Q}$ and $\boldsymbol{U}_h \in \mathbb{R}^{h \times h}$ are the weights respectively for the input and the previous state, $\boldsymbol{b}_h \in \mathbb{R}^h$ is the bias, and ϕ_h is the hidden activation function, usually $\tanh$. We usually add a classical fully connected layer at the end of a recurrent neural network with an output activation function chosen according to the target values range.



(a) Simplified view of a recurrent neural network

(b) Unfolded view of a recurrent neural network

Fig. 2.6 Recurrent neural network structure

The general formula of a recurrent neural network differs from the one written in Equation 2.18 since it is now time dependent:

$$\hat{f}_t(\boldsymbol{x}_t) = \left(\hat{f}_t^{[L+1]} \circ \hat{f}_t^{[L]} \circ \ldots \circ \hat{f}_t^{[1]}\right)(\boldsymbol{x}_t), \tag{2.20}$$

where $\hat{f}_t^{[l]}(\boldsymbol{x}_t) := \boldsymbol{h}_t^{[l]}$ is the output of layer l, as defined in Equations 2.19. Hence, given a sequence $(\boldsymbol{x}_t)_{t=1}^T$, a RNN will output a new sequence $(\hat{\boldsymbol{y}}_t)_{t=1}^T$ such that

$$(\hat{\boldsymbol{y}}_t)_{t=1}^T = \hat{f}\left((\boldsymbol{x}_t)_{t=1}^T\right) = \left(\hat{f}_t(\boldsymbol{x}_t)\right)_{t=1}^T. \tag{2.21}$$

Unfortunately, RNN have difficulties to take into consideration events that arised too long ago, due to the vanishing gradient problem. Long short term memory networks [11] partially solve this issue by adding a cell state to the network units, playing the role of memory:

$$\begin{aligned}
\boldsymbol{F}_t &= \phi_F(\boldsymbol{W}_F\boldsymbol{x}_t + \boldsymbol{U}_F\boldsymbol{h}_{t-1} + \boldsymbol{b}_F),\\
\boldsymbol{I}_t &= \phi_I(\boldsymbol{W}_I\boldsymbol{x}_t + \boldsymbol{U}_I\boldsymbol{h}_{t-1} + \boldsymbol{b}_I),\\
\boldsymbol{O}_t &= \phi_O(\boldsymbol{W}_O\boldsymbol{x}_t + \boldsymbol{U}_O\boldsymbol{h}_{t-1} + \boldsymbol{b}_O),\\
\boldsymbol{c}_t &= \boldsymbol{F}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{I}_t \odot \tanh(\boldsymbol{W}_c\boldsymbol{x}_t + \boldsymbol{U}_c\boldsymbol{h}_t - 1 + \boldsymbol{b}_c),\\
\boldsymbol{h}_t &= \boldsymbol{O}_t \odot \tanh(\boldsymbol{c}_t).
\end{aligned} \tag{2.22}$$

$\boldsymbol{F}_t$ is the forget gate, which controls which information from prior steps should be thrown away or kept. $\boldsymbol{I}_t$ is the input gate, controlling the cell state update given the new input. $\boldsymbol{O}_t$ is the output gate, controlling the computing of the new hidden state.

2.2.2 Autoencoders

Autoencoders are a type of neural networks that aim to learn an encoded representation of the inputs. An autoencoder consists of two parts: an encoder network that maps the input to its encoded representation, and a decoder network that reconstructs the original data given its encoded representation. So the input and the target of an autoencoder are the same. Formally, the goal is to approximate the identity function $f(\boldsymbol{x}) = \boldsymbol{x}$ with a deep neural neural network composed of two sub-networks, such its the approximation function is $\hat{f} = (\hat{f}_D \circ \hat{f}_E)$, where $\hat{f}_E$ corresponds to the encoder network and $\hat{f}_D$ to the decoder. This process is shown in Figure 2.7.

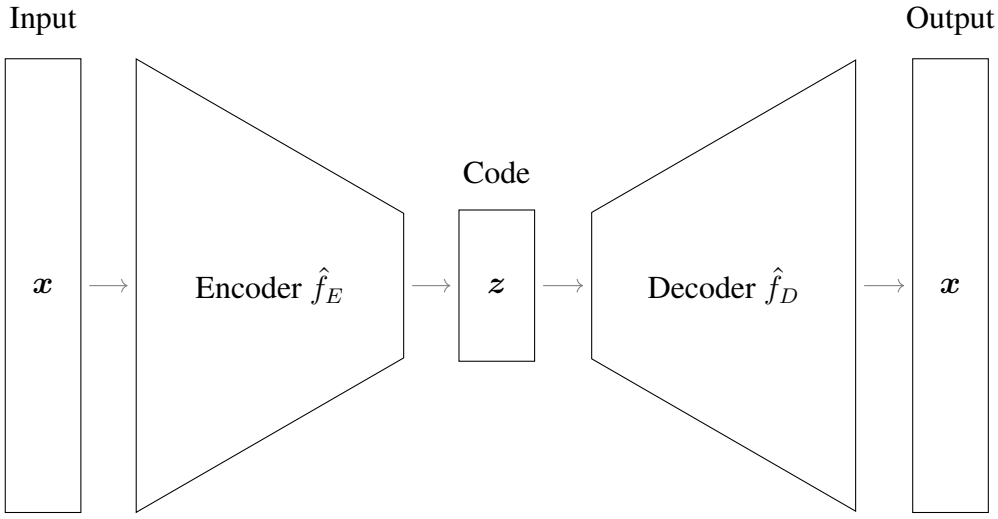


Fig. 2.7 The structure of an autoencoder

The loss is the reconstruction error:

$$\mathcal{L}(\boldsymbol{\Theta}) = \sum_{i=1}^{N} \left\| \boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)} \right\|^2 = \sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - \hat{f}(\boldsymbol{x}^{(i)}) \right\|^2 \tag{2.23}$$

Autoencoders actually learn to summarize inputs and thus learn structure in data, that is why they can be used in semi-supervised anomaly detection. Once an autoencoder has understood the distribution of normal data through its encoding-decoding process, it can guess if a new observation comes from a different distribution, by checking if it behaves as planned when encoded and decoded back. For an anomalous input, one expects the reconstruction error or Equation 2.23 to be unusually high.

If we use LSTM cells instead of classical neurons in the autoencoder, the latter can process the sliding window sequences $\boldsymbol{x}^{(t)} = (\boldsymbol{u}_s)_{s=t}^{t+S-1}$ and encode them into hidden and

cell states. The decoder can then recreate a full sequence, starting from an empty input and the encoded states, and taking the output of time t as the input of time $t+1$. This recurrent autoencoder is an extension of the Seq2Seq model [30] where the input and the target are the same. Then when a new incoming sequence contains points with an abnormally high reconstruction error, one can report them as anomalies. This approach is described precisely in [19].

The issue with the autoencoder approach is the interpretability of the results. Indeed, the input dimensions contributing to the high reconstruction error, and thus the detected anomaly, are not indicated. It is then difficult to localize the source of the problem, in order to act accordingly. This is the motivation behind forecasting-based methods.

2.2.3 Forecasting-based anomaly detection

To address the problem of lack of interpretability, one can try to explicitly model the expected behaviour of our industrial system, and compare it to the actual observations [14]. This requires some preliminary work, to understand the functioning of the system, in order to model it accurately.

First of all we define the variables $\mathcal{R} = \{Y_1, \dots, Y_R\}$ that we want to model, which are the quantities to monitor, also called dependent variables. Then, we have to find the variables $\mathcal{P} = \{X_1, \dots, X_P\}$ that are relevant to explain the values of the dependent variables; these are the independent variables. There may be other variables important to explain the dependent variables, but that are considered constant throughout the course of the investigation; we call them the control variables. Let $\{\boldsymbol{u}_t^{\mathcal{P}}\}_{t=1}^{T},\ \boldsymbol{u}_t^{\mathcal{P}} \in \mathbb{R}^P$ be the measurements corresponding to the independent variables, and $\{\boldsymbol{u}_t^{\mathcal{R}}\}_{t=1}^{T},\ \boldsymbol{u}_t^{\mathcal{R}} \in \mathbb{R}^R$ those corresponding to the dependent variables. We consider these sets to contain samples from a normal behaviour of the system only, that is, without any anomaly or failure.

Let us one more time consider the sliding window method, with sub-sequences $\boldsymbol{x}^{(t)} = (\boldsymbol{u}_t^{\mathcal{P}})_{s=t}^{t+S-1}$ and $\boldsymbol{y}^{(t)} = (\boldsymbol{u}_t^{\mathcal{R}})_{s=t}^{t+S-1}$. A LSTM network can then learn the relation between independent and dependent variables and approximate it with a function $\hat{f}$ such that

$$\hat{f}\left((\boldsymbol{u}_s^{\mathcal{P}})_{s=t}^{t+S-1}\right) = (\hat{\boldsymbol{u}}_s^{\mathcal{R}})_{s=t}^{t+S-1} \approx (\boldsymbol{u}_s^{\mathcal{R}})_{s=t}^{t+S-1}. \tag{2.24}$$

Most of the time, the independent variables $(\boldsymbol{u}_s^{\mathcal{P}})_{s=t}^{t+S-1}$ are not sufficient to forecast accurately the dependent variables $(\boldsymbol{u}_s^{\mathcal{R}})_{s=t}^{t+S-1}$. Indeed, we also need to know the state of the system at the beginning of the sequence. To do so, we can encode the b previous steps of the dependent variables $(\boldsymbol{u}_s^{\mathcal{P}})_{s=t-b}^{t-1}$, possibly with some more relevant information $(\boldsymbol{u}_s^{\mathcal{S}})_{s=t-b}^{t-1}$ called state variables, by processing them through another LSTM network, and use the

last hidden state h_{t-1} and cell state c_{t-1} as initial state of our main LSTM network. Both networks are plugged into one big structure, and are trained together.

Once our forecasting model is built, for each new sequences $(\boldsymbol{u}_s^{\mathcal{P}})_{s=t-b}^{t-1}$, $(\boldsymbol{u}_s^{\mathcal{S}})_{s=t-b}^{t-1}$, we can forecast $(\hat{\boldsymbol{u}}_s^{\mathcal{R}})_{s=t}^{t+S-1}$. Knowing the true sequence $(\boldsymbol{u}_s^{\mathcal{R}})_{s=t}^{t+S-1}$ as well, we can inspect the difference between each observed value $u_{si}^{\mathcal{R}}$ and the corresponding forecast value $\hat{u}_{si}^{\mathcal{R}}$, for $s = t, \ldots, t+S-1$ and $i = 1, \ldots, D$. If it is greater than a threshold $\lambda \in \mathbb{R}^+$, we report an anomaly.

Unlike autoencoders, this method allows some interpretability, since we know which output dimensions have an abnormal behaviour, and we can even visualise and compare each univariate predicted sequence with its corresponding actual observed sequence.

The whole process is summarized in Figure 2.8.

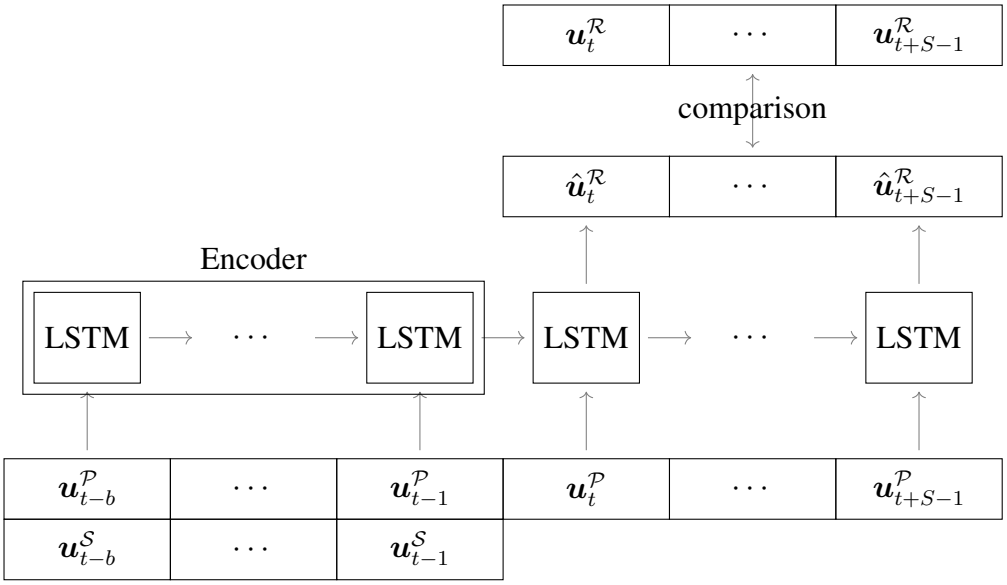


Fig. 2.8 Forecasting-based anomaly detection

2.2.4 Prediction intervals to replace thresholds

The two methods previously described, namely autoencoders (section 2.2.2) and forecasting-based anomaly detection (section 2.2.3) require to fix an arbitrary threshold on the reconstruction error and respectively the forecast error. But finding the optimal value for this threshold can be complex. In [20] and [19], the authors propose to fit a normal distribution to the errors and to set a threshold on its likelihood. We will here introduce another approach, by estimating a prediction interval, in which we are almost sure that the actual value should

lie. An anomaly is then a point that lies out of the prediction interval. We will in this section describe how to create a prediction interval for a neural network using the bootstrap method, as described in [13].

Let us recall the general deep learning setup. We have a set $\mathbb{X} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{N}$, $\boldsymbol{x}^{(i)} \in \mathbb{R}^Q$ being the inputs and $\boldsymbol{y}^{(i)} \in \mathbb{R}^D$ being the labels. We suppose there exists a function $f : \mathbb{R}^Q \to \mathbb{R}^D$ describing the underlying phenomenon, such that the data is generated according to

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon(\boldsymbol{x}), \tag{2.25}$$

$\epsilon : \mathbb{R}^Q \to \mathbb{R}^D$ describing a potential observation noise. Once we have built an approximation $\hat{f}$ of f with a deep neural network, we can decompose the error between true values and predictions as

$$\begin{aligned} \boldsymbol{y} - \hat{f}(\boldsymbol{x}) &= \boldsymbol{y} - f(\boldsymbol{x}) + f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) \\ &= \epsilon(\boldsymbol{x}) + [f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})]. \end{aligned} \tag{2.26}$$

Assuming the independence of the two terms in Equation 2.26, the total variance associated to the model outcome is

$$\sigma^2(\boldsymbol{x}) = \sigma_\epsilon^2(\boldsymbol{x}) + \sigma_{\hat{f}}^2(\boldsymbol{x}), \tag{2.27}$$

where σ_ϵ is the variance due to noise in observations, or aleatoric uncertainty, and $\sigma_{\hat{f}}$ is the variance associated to the model misspecification, or epistemic uncertainty. To estimate $\sigma_{\hat{f}}$, we will use an ensemble method called the bootstrap method. It consists in building B training sets by sampling with replacement from the original dataset, and training B different models $\{\hat{f}_b\}_{b=1}^{B}$ on these sets. The bootstrap prediction function is then

$$\hat{f}_{\text{bootstrap}}(\boldsymbol{x}) := \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(\boldsymbol{x}), \tag{2.28}$$

and the variance $\sigma_{\hat{f}}^2$ can be estimated as

$$\sigma_{\hat{f}}^2(\boldsymbol{x}) \approx \frac{1}{B-1} \sum_{b=1}^{B} \left(\hat{f}_b(\boldsymbol{x}) - \hat{f}_{\text{bootstrap}}(\boldsymbol{x}) \right)^2, \tag{2.29}$$

On another hand, we have

$$\begin{aligned}\sigma_\epsilon^2(\boldsymbol{x}) &= \sigma^2(\boldsymbol{x}) - \sigma_{\hat{f}}^2(\boldsymbol{x}) \\ &= \mathbb{E}\left[(\boldsymbol{y} - \mathbb{E}[\boldsymbol{y} \mid \boldsymbol{x}])^2 \mid \boldsymbol{x}\right] - \sigma_{\hat{f}}^2(\boldsymbol{x}) \\ &= \mathbb{E}\left[(\boldsymbol{y} - \hat{f}_{\text{bootstrap}}(\boldsymbol{x}))^2 \mid \boldsymbol{x}\right] - \sigma_{\hat{f}}^2(\boldsymbol{x})\end{aligned} \tag{2.30}$$

which can be estimated with the residual values

$$r^2(\boldsymbol{x}^{(i)}) := \max\left(\left(\boldsymbol{y}^{(i)} - \hat{f}_{\text{bootstrap}}(\boldsymbol{x}^{(i)})\right)^2 - \sigma_{\hat{f}}^2\left(\boldsymbol{x}^{(i)}\right), 0\right). \tag{2.31}$$

We can then build the set of residuals $\{(\boldsymbol{x}^{(i)}, r^2(\boldsymbol{x}^{(i)}))\}_{i=1}^N$, and train another neural network on samples that were not seen by the bootstrap models during their training (e.g. validation samples), to find a relation

$$\hat{f}_{r^2}(\boldsymbol{x}) \approx r^2(\boldsymbol{x}) \approx \sigma_\epsilon^2(\boldsymbol{x}), \tag{2.32}$$

using the maximum likelihood as loss function:

$$\mathcal{L}(\boldsymbol{\Theta}) = \frac{1}{2}\sum_{i=1}^{N}\left(\ln\left(\hat{f}_{r^2}(\boldsymbol{x}^{(i)})\right) + \frac{r^2(\boldsymbol{x}^{(i)})}{\hat{f}_{r^2}(\boldsymbol{x}^{(i)})}\right). \tag{2.33}$$

Finally, we have our estimation

$$\hat{\sigma}^2(\boldsymbol{x}) := \frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{f}_b(\boldsymbol{x}) - \hat{f}_{\text{bootstrap}}(\boldsymbol{x})\right)^2 + \hat{f}_{r^2}(\boldsymbol{x}). \tag{2.34}$$

Hence, given an input $\boldsymbol{x}$, the prediction interval with a confidence level of $(1-\alpha)$ is then given by

$$\hat{f}_{\text{bootstrap}}(\boldsymbol{x}) \pm t_{1-\alpha/2,B}\sqrt{\hat{\sigma}^2(\boldsymbol{x})}, \tag{2.35}$$

where $t_{1-\alpha/2,B}$ is the $1-\alpha/2$ quantile of the Student's t distribution with B degrees of freedom. The parameter α allows flexibility. If we reduce α, less false positive are expected, but the algorithm will be less sensitive to anomalies.

Chapter 3

Practice: Predictive Maintenance in Hydropower Plants

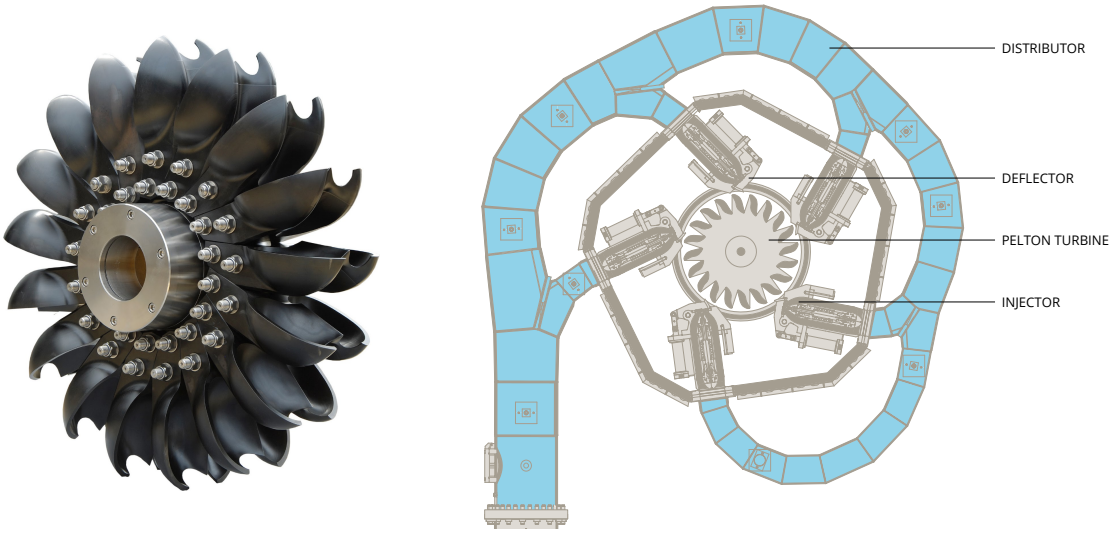
In the previous chapter we have seen that predictive maintenance can be framed as a semi-supervised anomaly detection problem, and we have introduced two deep learning methods to address this problem, namely autoencoders and forecasting based method. In this new chapter we apply these methods to real data provided by the Swiss energy company Alpiq.

3.1 Introduction

We will focus on a well known critical issue in impoundment hydropower plants, that is, the monitoring of turbine guide bearing pads temperature for vertical shafts in high head plants.

We first of all need to set the context, and introduce some basic knowledge on hydropower plants that we did not discuss in the introduction. Once again, more information on this field can be found in [23], [7] and [25].

Impoundment hydropower plants are made up of groups, which are shafts on which a generator, a turbine, and possibly a pump, are aligned. We will here consider only vertical shafts, although horizontal ones exist. For high head plants (more than 300m between the reservoir top and the downstream level), we most of the time use a Pelton turbine. In turbine-mode, injectors propel water at high speed on the Pelton turbine to make it rotate and produce electricity (see Figure 3.1).



(a) (b)

Fig. 3.1 A Pelton turbine (a) and its functioning (b).

The turbine guide bearing, made up of pads, surrounds the shaft at the turbine's level, and is separated from the shaft by a layer of oil. It aims to keep the shaft aligned vertically and to support any radial force that can take place during the operation. When the machines are working, this applies a force on the shaft, therefore the pads heat. If the shaft rubs against pads, it makes them melt and creates an major failure, that is why it is important to monitor the bearing pads temperature.

The main significant features to consider, besides the temperatures of the bearing pads, are the degree of opening of the injectors, the temperature of the oil, and the temperature of the cooling water. If the facility is a pumped-storage plant, we also consider the turbine speed or the active power of the station to distinguish turbine mode from pump mode, as the latter could influence turbine guide bearing pads temperature as well, even if the injectors are not open.

In this chapter we will consider two case studies: first a data-set from the FMHL+ power plant, and then one from Bieudron power plant.

3.2 FMHL+ power plant

The FMHL+ plant is actually an extension of the FMHL power plant, located in Veytaux on the lake Geneva shore. It is a pumped-storage plant with the lake Geneva as the lower

reservoir, and the Hongrin lake as the upper reservoir. FMHL+ added two new groups (Group 5 and Group 6) to the four existing, each of the two being vertical shafts with a power of 120MW. The structure of the two new groups is shown in Figure B.1 of Appendix B. The goal is to monitor the ten turbine guide bearing pads temperature of the Group 5.

3.2.1 Data-set and methods

The data-set is a multi-variate time series with minute granularity and covers one years of sequential observations, assumed as normal. The variables are the pads temperature, the injectors opening, the oil and cooling water temperature, and the turbine speed.

As we have seen in the chapter 2, we have to cut the time series into small sub-sequences using the sliding window technique, in order to process them an input to detect collective and contextual anomalies. We use a window size of $S = 240$ which corresponds to four hours, and a stride value of $R = 60$ to take sub-sequences each hours.

The sub-sequences $\boldsymbol{x}^{(t)} = (\boldsymbol{u}_s)_{s=t}^{t+S-1}$ form our data-set $\mathbb{X} = \{\boldsymbol{x}^{(t)}\},\ t = 1, R, \ldots, \lfloor \frac{T}{R} \rfloor$. We randomly split $\mathbb{X}$ into a train set, a validation set, and a test set, containing respectively 60%, 20% and 20% of the sequences. The train set contains 5254 sequences and the validation and test sets contain 1750 sequences each. We standardise all data using the mean and standard deviation of the training set before passing it to our neural networks, in order to avoid disproportion between features and facilitate optimization.

We define our recurrent autoencoder with LSTM cells that maps sequences to hidden and cell states of size 256 each, and decode them to reconstruct the original data. We used a batch size of 256, MSE loss, and an RMSProp optimizer as in [9], with a learning rate of 0.001. Our forecast model for its part is made of 5 hidden layers of size 128 each, with LSTM cells. We used a batch size of 32, MSE loss, and an RMSProp optimizer with a learning rate of 0.002. The dependent variables are the bearing pads temperatures, the independent variables are the injectors opening, and the state variables are the oil and cooling water temperatures. The parameters of these neural networks were optimized using the hyperband method, described in [17].

For both methods, we build $B = 10$ copies of the network and train them on bootstraped samples of the training set, in order to create prediction intervals as described in Section 2.2.4, with a confidence level parameter α to choose. We monitor the value of the loss on the validation set in order to early stop the training when the model overfits, and we keep the parameters that minimize this validation loss. In order to estimate the aleatoric uncertainty, we train another network on the residuals of the validation set, searching for the parameters minimizing the likelihood loss on the test set. This network has one hidden layer of 64 LSTM cells, and is trained with batch size 128 and learning rate 0.001.

3.2.2 Results

When working on this case study, we quickly realised a problem: we did not have access to a predictable failure or incident to proof test our models. The only performance indicator that we have is how well it fits the normal behaviour of our system. We apply the models to the test set, which contains only normal behaviour data, and look at the number of reported anomalies, that is, the false positives.

Autoencoder With the autoencoder model, we obtain a MSE loss of 0.029 on the test set. We then compute the number of time steps for which the reconstructed data is out of the α-prediction interval for any dimension. Table 3.1 shows this false positive time steps rate for some values of α and Figure 3.2 graphs the false positive rate as a function of α.

α	**False positive**	**Total**	**Ratio**
0.05	42 859	420 000	0.1020
0.01	10 055	420 000	0.0239
0.005	5966	420 000	0.0142
0.001	1895	420 000	0.0045

Table 3.1 False positive ratio for some values of α.

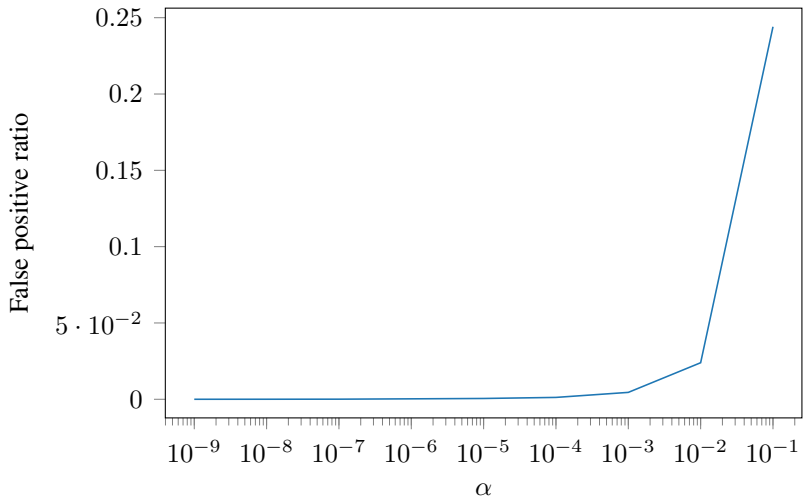


Fig. 3.2 False positive rate as a function of α

Forecasting-based method The forecasting-based model has an MSE loss of 0.0018 on the test set. Since the dimension in which the anomaly occurs is relevant, we will not only look at the time steps but rather at the total number of observed values that lie out of the prediction interval. The Table 3.2 and Figure 3.3 show the results for the forecasting-based method.

α	**False positive**	**Total**	**Ratio**
0.05	48 609	4 200 000	0.0116
0.01	27 873	4 200 000	0.0066
0.005	22 119	4 200 000	0.0053
0.001	16 355	4 200 000	0.0039

Table 3.2 False positive ratio for some values of α.

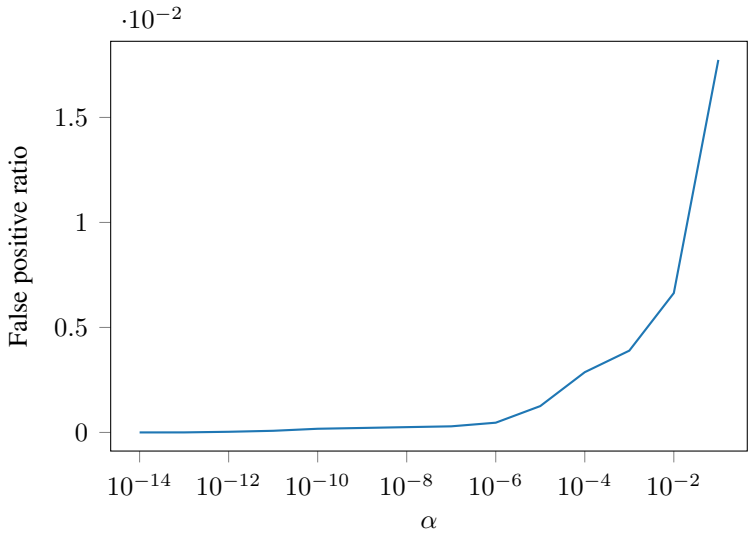


Fig. 3.3 False positive rate as a function of α

Even if both models have successfully learned the normal behaviour of the system, and although we can tune the confidence parameter α to adjust the false positive rate, these results are not sufficient to evaluate our methods. Indeed, with a big enough interval, one can achieve a false positive rate of 0%, but we risk missing all incoming anomalies. In order to really assess the performance of our models, we need to confront them to some failures or incident. That is why, staying on the turbine guide bearing pads temperature issue, we investigated another data-set, coming from another power plant.

3.3 Bieudron power plant

The Bieudron power station, in the Cleuson-Dixence complex, is a classical impoundment hydropower plant. It has the world's highest head (1883 m). It turbines the water from the Grande Dixence reservoir with three groups of 400MW each, for a total power of 1200MW. Each group is a vertical shaft, whose structure is shown in Figure B.2 of Appendix B. We aim to monitor the eight turbine guide bearing pads temperature of the Group 1.

In the Bieudron power plant, two arbitrary threshold are currently fixed on the bearing pads temperature. When a pad's temperature exceeds the first threshold, an alarm is sent, and when the second one is crossed, the group stops its activity. On August 4, 2016, the first alarm level of Group 1 bearing pad 2 was triggered. This resulted in investigation and repair works on pads 2, 5, and 6, until the final resolve on October 6, 2017. The goal is to train our models on the normal behaviour following the maintenance action of October 2017, and to test it retroactively on the faulty behaviour preceding the discovery of the problem, in July and August 2016.

3.3.1 Data-set and methods

The data-set, which has minute granularity, contains one year of the system's proper behaviour, following the repair works, and one month of potential faulty behaviour, during which the alarm on pad 2 was triggered. Since Bieudron is not a pumped-storage plant, we will consider the same features as for FMHL+ case, but without the turbine speed.

We proceed one more time by slicing the dataset into sliding window sub-sequences, such that our data-set is $\mathbb{X} = \{\boldsymbol{x}^{(t)}\}_{t=1}^{T} = \{(\boldsymbol{u}_s)_{s=t}^{t+S-1}\}_{t=1}^{T}$. For training, we only consider the set of sequences observed in the year following the repair works. We randomly split this set into a training set, a validation set, and a test set, containing respectively 60%, 20% and 20% of the sequences. The train set contains 5238 sequences and the validation and test sets contain 1746 sequences each.

We use the same architecture and parameters that we used for the FMHL+ case study, for both the autoencoder and the forecast model.

3.3.2 Results

This time, we not only estimate how well the models can fit the normal behaviour of the system, along with false positive rate. We will look at the models output on the period preceding the repair works, during which bearing temperature alarms were triggered, to see if the problem is well detected.

Autoencoder The autoencoder achieved an MSE score of 0.0164. As before, the evolution of the false positive rate with respect to the confidence parameter α is shown in Table 3.3 and Figure 3.4.

α	**False positive**	**Total**	**Ratio**
0.05	45 694	419 040	0.1090
0.01	11 796	419 040	0.0282
0.005	7634	419 040	0.0182
0.001	2484	419 040	0.0059

Table 3.3 False positive ratio for some values of α.

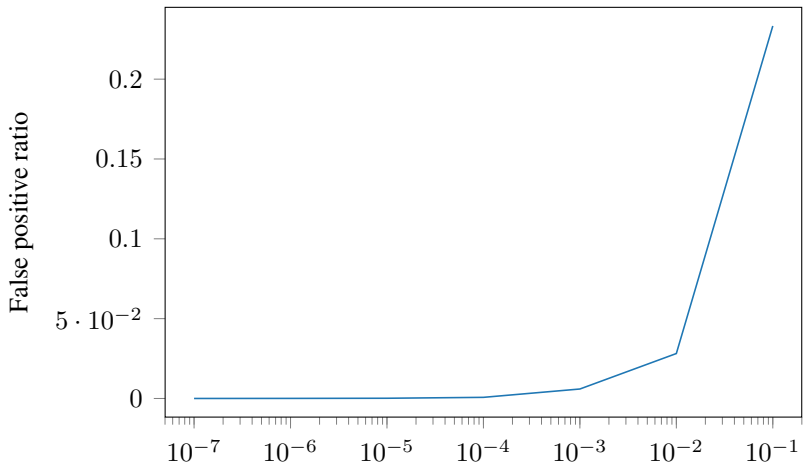
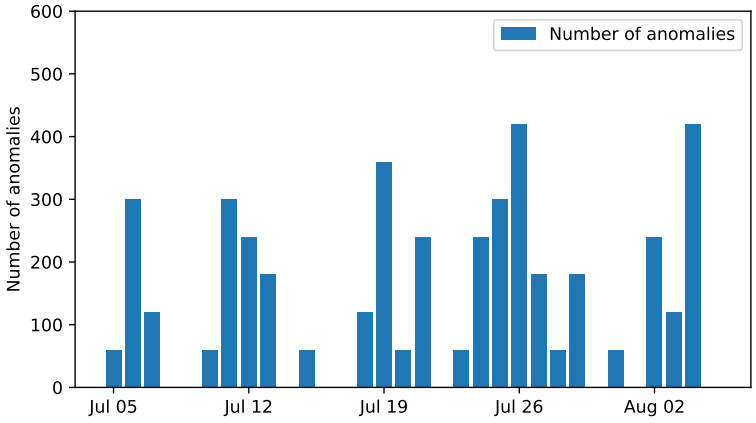
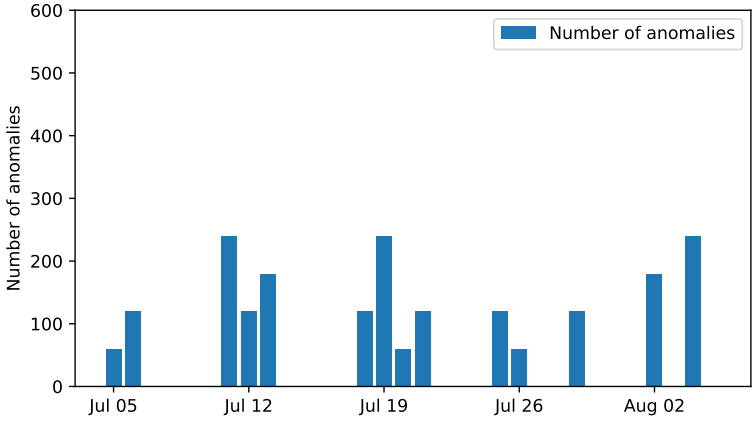


Fig. 3.4 False positive rate as a function of α

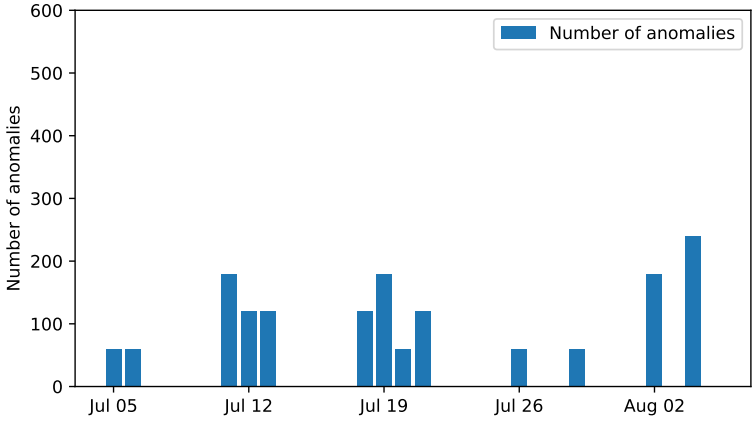
Suppose that we run the model each hour to predict the last hour of the system behaviour, and we report any value lying out of the interval as an anomaly. Figure 3.5 shows the number of anomalies reported by the autoencoder each day between July 4 and August 4, for $\alpha = 0.95, 0.99, 0.995, 0.999$.



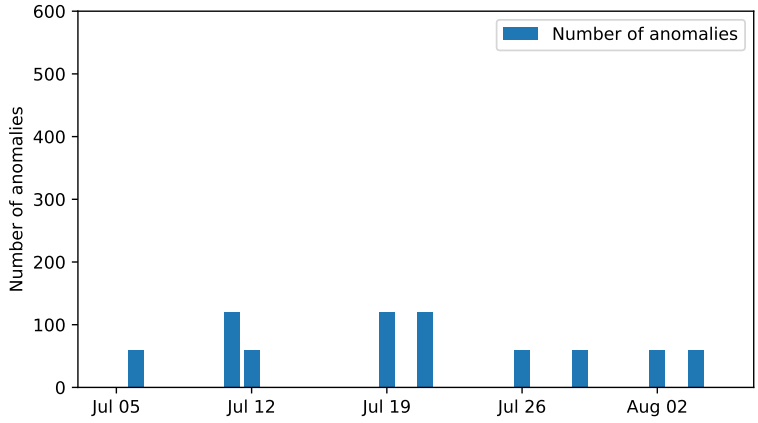
(a) $\alpha = 0.05$



(b) $\alpha = 0.01$



(c) $\alpha = 0.005$



(d) $\alpha = 0.001$

Fig. 3.5 Number of reported anomalies per day during the faulty period depending on α.

Some anomalies are detected, notably on August 4, that is, the day when the alarm triggered, and the bigger α is, the more anomalies are reported. But since we have no more information about these potential anomalies, it is hard to know if we have successfully detected relevant anomalies. This is why forecasting-based methods may be more judicious to use.

Forecasting-based method When fitting a forecasting model on Bieudron dataset, we were able to obtain an MSE loss of 0.0004. Table 3.4 and Figure 3.4 once again describe the false positive rate versus α relation.

α	**False positive**	**Total**	**Ratio**
0.05	51 638	3 352 320	0.015 40
0.01	10 561	3 352 320	0.003 15
0.005	4318	3 352 320	0.001 29
0.001	59	3 352 320	0.000 02

Table 3.4 False positive ratio for some values of α.

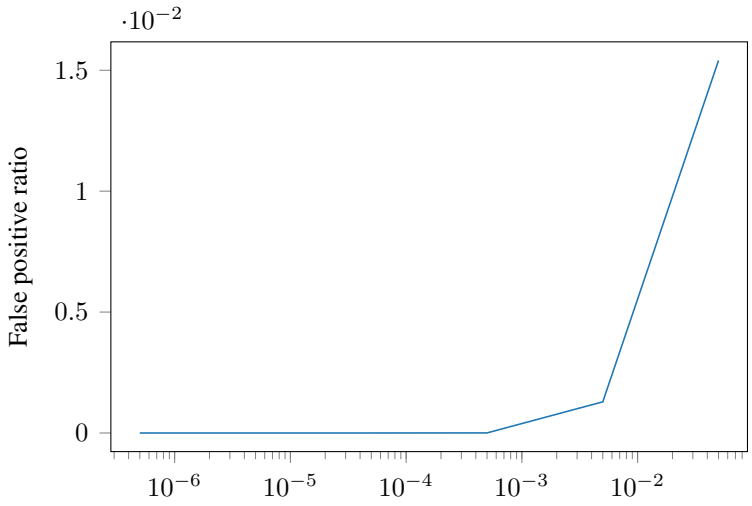
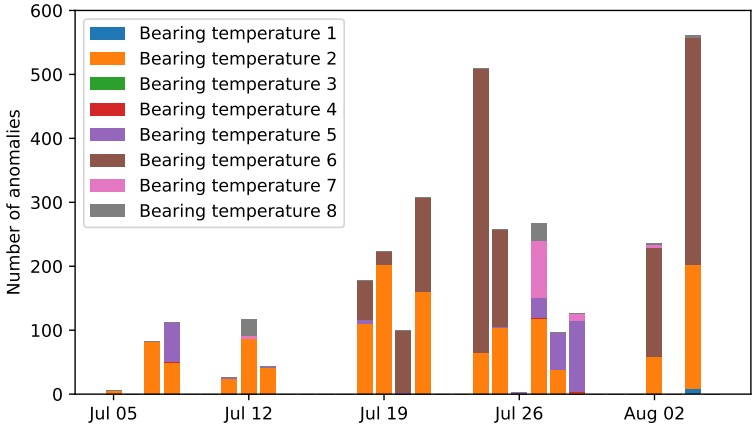
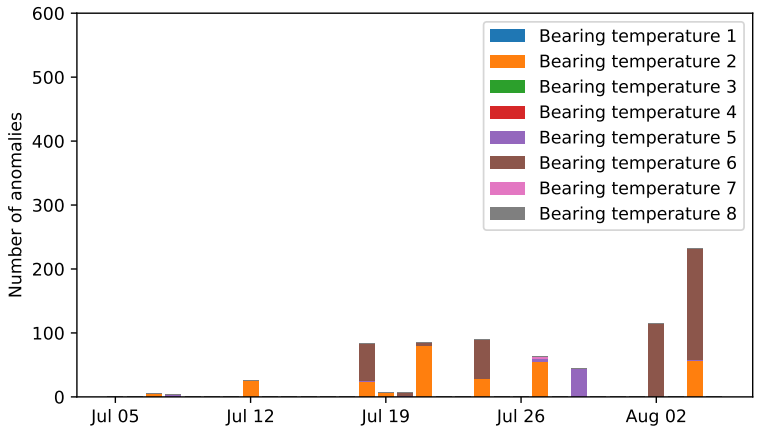


Fig. 3.6 False positive rate as a function of α

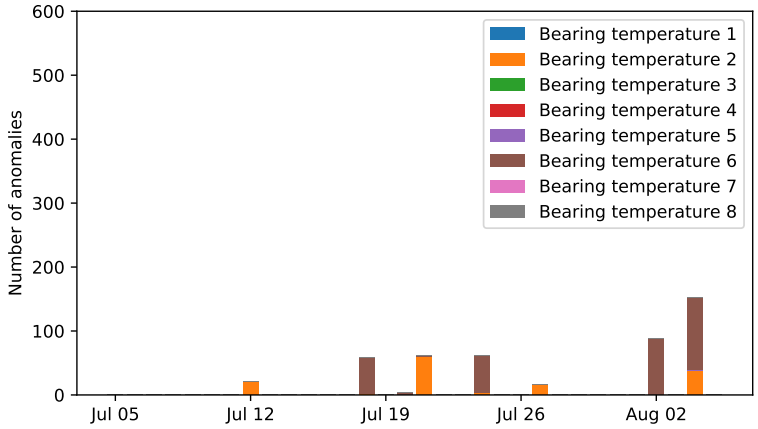
As for the autoencoder, we run the model each hour and compute the number of reported anomalies, aggregated by day, on the period containing potential early signs of the problem. This time however, with the forecasting-based method, one can locate the anomaly in the output dimensions. In other words, we know which bearing pad has an abnormal temperature, causing the anomaly report. The results are shown in Figure 3.7.



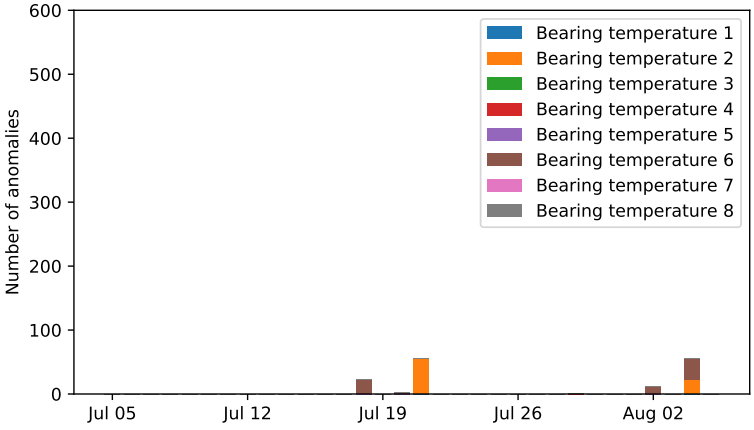
(a) $\alpha = 0.05$



(b) $\alpha = 0.01$



(c) $\alpha = 0.005$



(d) $\alpha = 0.001$

Fig. 3.7 Number of reported anomalies per day during the faulty period depending on α.

It has to be noted that the bearing pads 2 and 6 problems are properly detected, and in particular on August 4, day at which the alarm was triggered, which counts the most anomaly reports. With $\alpha = 0.001$, some early warning signs appear for pad 6 since July 18 and for pad 2 on July 21. With $\alpha = 0.005$, it appears on July 12 for pad 2 and July 18 for pad 6. When we set $\alpha = 0.01$, we have few anomalies reports on July 7 for pad 2, and more significant ones from July 12. For pad 6, we see early signs again from July 18, and we also have anomalies on pad 5 reported on July 27 and 29. With $\alpha = 0.05$, much more anomalies appear, starting on July 5 for pad 2, July 8 for pad 5, July 18 for pad 6, but also a few anomaly reports on pads 5 and 7 on July 27 notably. We see that adjusting α increases the sensitivity of the model to anomalies.

In a nutshell, one can achieve an early detection of bearing temperature issues while having a low false positive rate (see Table 3.4).

The advantage of the forecasting-based method is the interpretability, and the possibility to visualise the anomaly detection process. Figure 3.8 illustrates this, by graphing the true values and the forecast for the whole August 4, 2016 day, with confidence parameter $\alpha = 0.05$. Anomalies are represented as vertical red bars, and we can see that the temperature of bearing pads 2 and 6 are higher than what the model expects between time steps 650 and 1200 approximately. This gives an idea of a graphical user interface that could be used by operators to visualise the anomalies and interpret them.

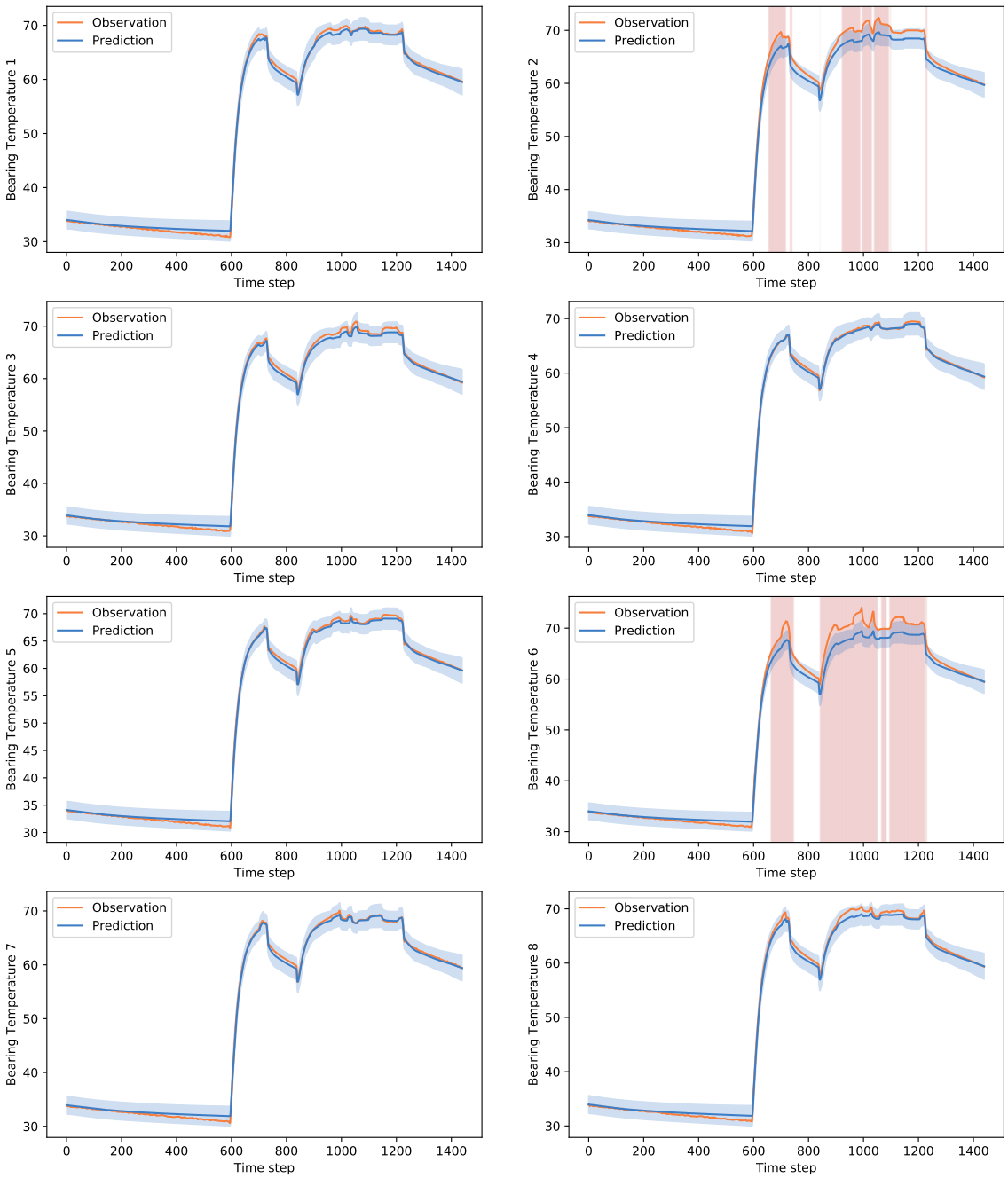


Fig. 3.8 Forecasting-based anomaly detection visualisation for August 4, 2016

Chapter 4

Conclusion and Future Work

In this chapter we will summarize the work that has been done and, based on our experiences, we will draw conclusions. The goal is to establish an appropriate strategy and framework for machine learning based predictive maintenance. We will also try to highlight possibly interesting future work.

In section 2.1.1 and 2.1.2, we have seen that supervised learning is not appropriate since failure data is lacking. That is why we reframe the predictive maintenance problem as semi-supervised anomaly detection in time series, where the goal is to learn the normal behaviour of an industrial system, and classify the diverging observations as anomalies, that need attention. In section 2.1.3, we have introduced classical algorithms that can perform this task, namely density estimation methods, clustering algorithms, and one-class SVM. But due to bad scaling, notably because of the curse of dimensionality, and because they cannot process time series data naturally, classical methods are not the best choice for predictive maintenance. To address this problem, we thus chose to focus on deep learning techniques. We have seen two methods based on deep neural networks for anomaly detection. First autoencoders, in section 2.2.2, that learn to reconstruct inputs, and on another hand forecasting-based method, in section 2.2.3, that explicitely models the system. In order to process temporal sequences, more precisely sliding window sequences from a multivariate time series data, we use LSTM cells. We proposed using a prediction interval with a given confidence level on the models output, and to report any new observation lying outside of it as an anomaly.

When we applied autoencoders and forecasting-based method with prediction intervals to real data, coming from the FMHL+ and the Bieudron power plants, we obtained several results. First of all, by tuning the parameter α of the prediction interval confidence, one can choose the sensitivity of the interval and reduce false positives as much as wanted. This of course has a drawback since it makes the interval more permissive, yet we need enough sensitivity to capture anomalies. By testing the models on a faulty behaviour example from

the Bieudron power plant, we confirmed that a forecast-based model was able to detect the test anomalies, while keeping a low false positive rate. We were not able to reproduce comparable results with autoencoders, that certainly detected anomalies, but the lack of interpretability did not allow to validate the results. Conversely, forecasting-based method with prediction interval allows a high level of interpretability for the anomaly detection process. The operator can visualise which dimension causes the anomaly, and how it differs from the expected behaviour of the system.

Forecasting-based method for predictive maintenance can be used online, with a delay equal on the sliding window size. It simulates in real-time the normal functioning of an industrial system and reports any anomalous behaviour. It offers a smarter alternative to the thresholds approach that is currently implemented, and can be combined with it. Unlike thresholds, this method captures multivariate collective and contextual anomalies. Due to its interpretability, the integration of such a solution would be easier and much more relevant than deploying a black box machine learning tool.

In this document we focused on LSTM neural networks to process time series data, but some other approaches are available, and it would be interesting to test them as well. For example, attention mechanisms [33] have been introduced for transduction problems and come in powerful for time series as well. One-dimensional convolutional networks with dilatation [32] have also shown very good results in audio sequences processing, and are less needy in terms of computer power. Very recently, [3] proposed to implement neural networks as continuous-depth models instead of discrete layers, the output being computer with differential equation solvers. This achieves impressing results on time series data and deserves to be tested on predictive maintenance problems.

References

[1] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[2] Chandola, V. (2009). *Anomaly Detection for Symbolic Sequences and Time Series Data*. University of Minnesota, 2009. Major: Computer science.

[3] Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. *CoRR*, abs/1806.07366.

[4] Doshi-Velez, F. and Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. *arXiv e-prints*, page arXiv:1702.08608.

[5] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press.

[6] Fernández-Francos, D., Martínez-Rego, D., Fontenla-Romero, O., and Alonso-Betanzos, A. (2013). Automatic bearing fault diagnosis based on one-class ν-svm. *Computers & Industrial Engineering*, 64(1):357 – 365.

[7] Giesecke, J. and Mosonyi, E. (2005). *Wasserkraftanlagen: Planung, Bau und Betrieb*. Springer.

[8] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[9] Graves, A. (2013). Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850.

[10] Heimes, F. O. (2008). Recurrent neural networks for remaining useful life estimation. In *2008 International Conference on Prognostics and Health Management*, pages 1–6.

[11] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[12] Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651 – 666. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR).

[13] Khosravi, A., Nahavandi, S., Creighton, D., and Atiya, A. F. (2011). Lower upper bound estimation method for construction of neural network-based prediction intervals. *IEEE transactions on neural networks*, 22(3):337–346.

[14] Kroll, B., Schaffranek, D., Schriegel, S., and Niggemann, O. (2014). System modeling based on machine learning for anomaly detection and predictive maintenance in industrial plants. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–7.

[15] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521:436–44.

[16] Li, H., Parikh, D., He, Q., Qian, B., Li, Z., Fang, D., and Hampapur, A. (2014). Improving rail network velocity: A machine learning approach to predictive maintenance. *Transportation Research Part C: Emerging Technologies*, 45:17 – 26. Advances in Computing and Communications and their Impact on Transportation Science and Technologies.

[17] Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2016). Efficient hyperparameter optimization and infinitely many armed bandits. *CoRR*, abs/1603.06560.

[18] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

[19] Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., and Shroff, G. (2016). Lstm-based encoder-decoder for multi-sensor anomaly detection. *CoRR*, abs/1607.00148.

[20] Malhotra, P., Vig, L., Shroff, G., and Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. In *ESANN*.

[21] McLachlan, G. J., Lee, S. X., and Rathnayake, S. I. (2019). Finite mixture models. *Annual Review of Statistics and Its Application*, 6(1):null.

[22] Molina, J., Isasi, P., Berlanga, A., and Sanchis, A. (2000). Hydroelectric power plant management relying on neural networks and expert system integration. *Engineering Applications of Artificial Intelligence*, 13(3):357 – 369.

[23] Raabe, J. (1985). *Hydro power: the design, use, and function of hydromechanical, hydraulic, and electrical equipment*. VDI-Verlag.

[24] Reynolds, D. (2015). *Gaussian Mixture Models*, pages 827–832. Springer US, Boston, MA.

[25] Sarlos, G., Haldi, P., and Verstraete, P. (2003). *Systèmes énergétiques: offre et demande d'énergie : méthodes d'analyse*. Traité de génie civil de l'Ecole polytechnique fédérale de Lausanne. Presses polytechniques et universitaires romandes.

[26] Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., and Platt, J. (1999). Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 582–588, Cambridge, MA, USA. MIT Press.

[27] Scott, D. W. (2015). *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons.

[28] Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London.

[29] Stanton, J. M. (2001). Galton, pearson, and the peas: A brief history of linear regression for statistics instructors. *Journal of Statistics Education*, 9(3):null.

[30] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

[31] Tax, D. M. and Duin, R. P. (2004). Support vector data description. *Machine learning*, 54(1):45–66.

[32] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499.

[33] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

[34] Wang, S., Wang, K., and Li, Z. (2016/11). A review on data-driven predictive maintenance approach for hydro turbines/generators. In *6th International Workshop of Advanced Manufacturing and Automation*. Atlantis Press.

[35] Åsnes, A., Willersrud, A., Kretz, F., and Imsland, L. (2018). Predictive maintenance and life cycle estimation for hydro power plants with real-time analytics. In *Predictive maintenance and life cycle estimation for hydro power plants with real-time analytics*.

Appendix A

How neural network actually learn

As we have seen in 2.2.1, neural networks learn by moving their parameters in the opposite direction of the loss gradient by a small quantity γ. This procedure is described in 2.

Algorithm 2 The gradient descent algorithm

function GRADIENTDESCENT($\mathcal{L}, \boldsymbol{\Theta}^{(0)}, \gamma$)
  repeat
    $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}^{(0)}$
    for $\theta \in \boldsymbol{\Theta}$ **do**
      $\theta \leftarrow \theta - \gamma \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \theta}$
    end for
  until convergence
  return $\boldsymbol{\Theta}$
end function

However, the main challenge is to compute the loss gradient, that is, the partial derivative of the loss with respect to each parameter $\frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \theta}$. This is done with the most crucial algorithm of deep learning, namely backpropagation, that consists in the following steps.

1. Perform forward propagation to compute each layer output $\boldsymbol{a}^{[l]}$ for $l = 1, \ldots, L+1$, where $\boldsymbol{a}^{[L+1]} = \hat{\boldsymbol{y}}^{(i)}$.
2. Compute the error $\boldsymbol{\delta}^{[L+1]} = \nabla_{\boldsymbol{a}} \mathcal{L} \odot \phi'^{[L+1]}(\boldsymbol{a}^{[L+1]})$, where $\nabla_{\boldsymbol{a}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}_j^{[L+1]}} \right)_{j=1}^{D}$.
3. Backpropagate the error $\boldsymbol{\delta}^{[l]} = (\boldsymbol{W}^{[l+1]} \boldsymbol{\delta}^{[l+1]}) \odot \phi'^{[l+1]}(\boldsymbol{a}^{[l]})$ for $l = 1, \ldots, L$.
4. Get the gradient $\frac{\partial \mathcal{L}}{\partial w_{ij}^{[l]}} = \boldsymbol{a}_i^{[l-1]} \boldsymbol{\delta}_j^{[l]}, \quad \frac{\partial \mathcal{L}}{\partial b_j^{[l]}} = \boldsymbol{\delta}_j^{[l]}$.

Appendix B

Power plant plans

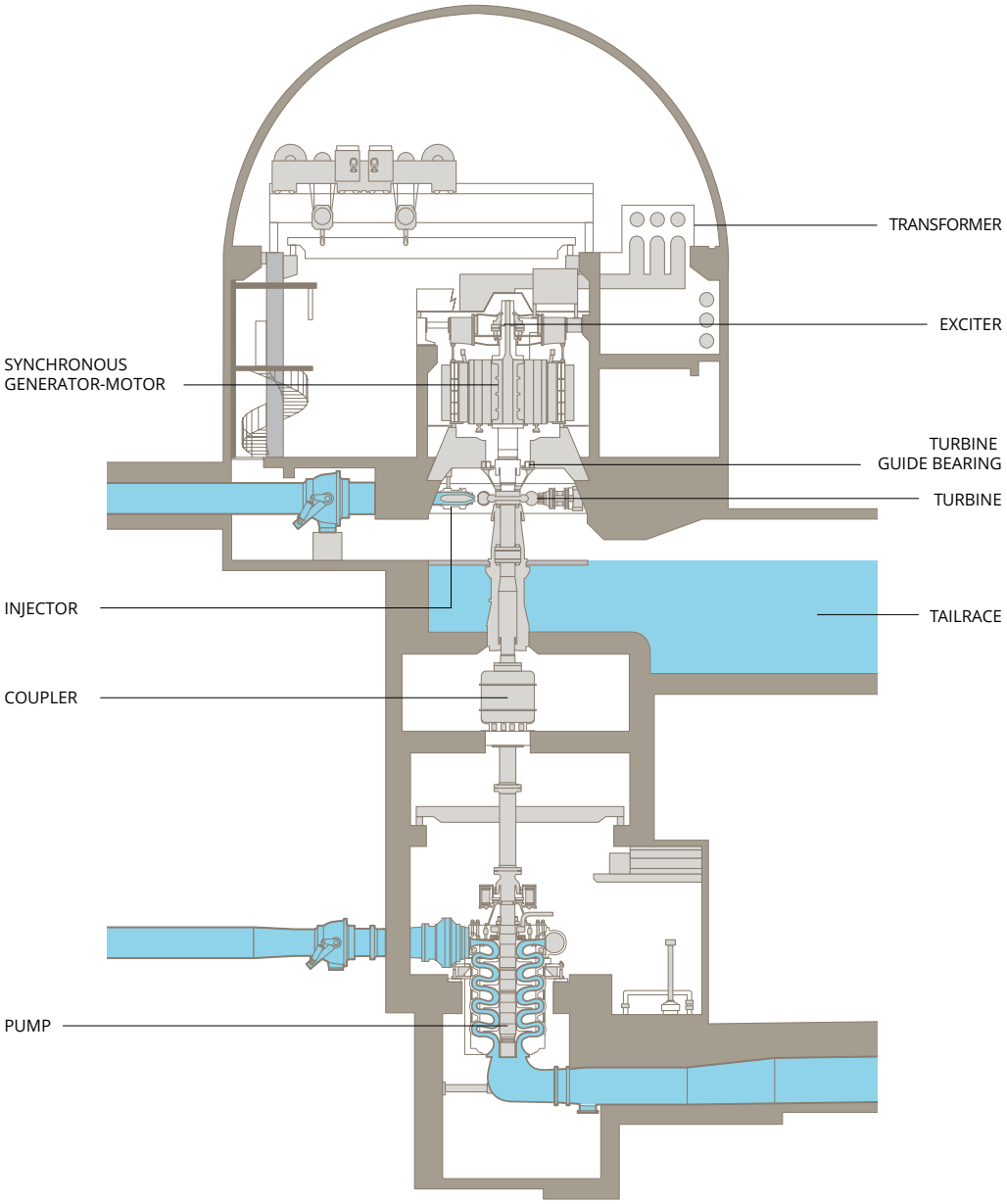


Fig. B.1 Transverse cross-section of FMHL+ groups.

1 GENERATOR SHAFT
2 EXCITER
3 ROTOR – COOLING WATER
4 ROTOR
5 STATOR
6 TURBINE GUIDE BEARING
7 PELTON TURBINE
8 RACK AND PLATFORM
9 INJECTOR
10 BUSBARS
11 COMBINED TRUST AND BEARING SUPPORT

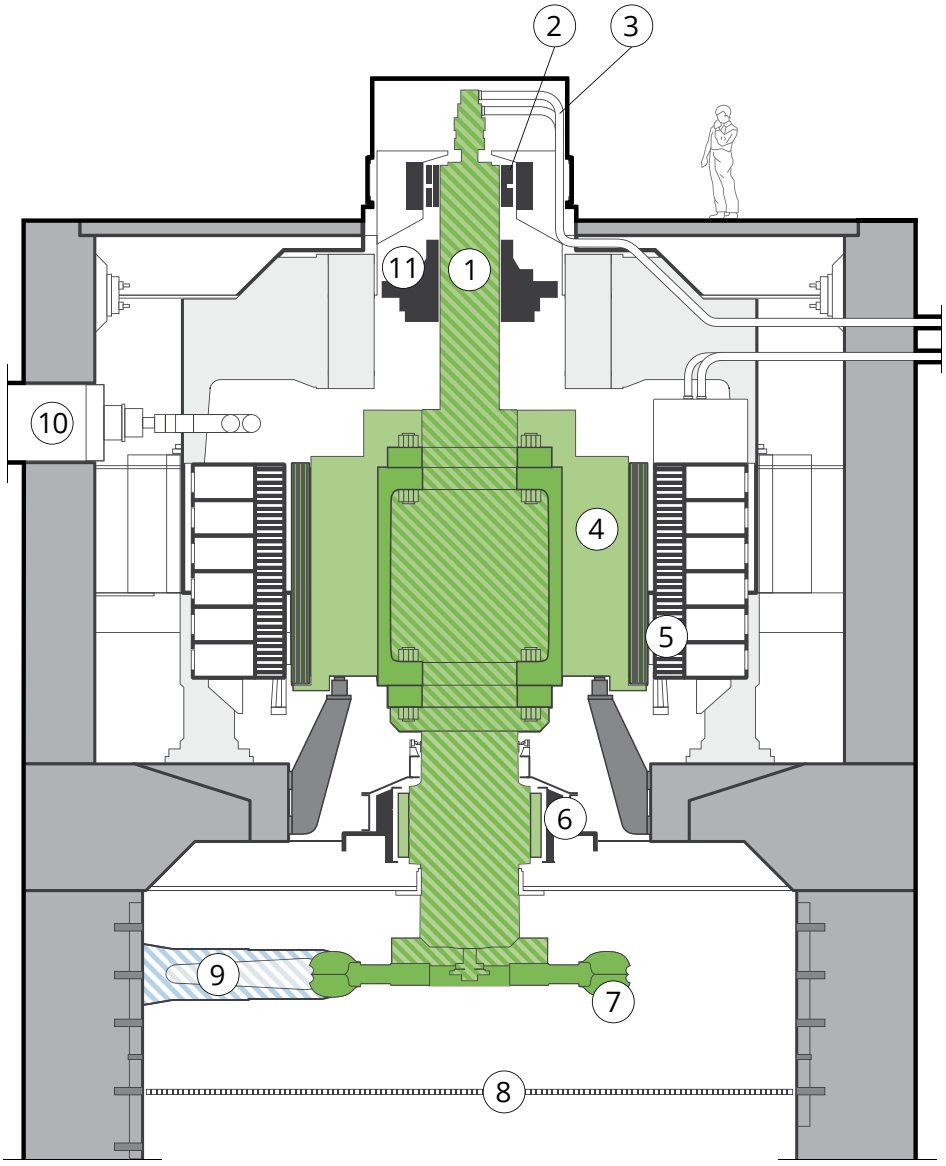


Fig. B.2 Transverse cross-section of Bieudron groups.