UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Final Dissertation

Predictive learning applied to muon chamber monitoring

Thesis supervisor:

Prof. Marco Zanetti

Candidate:

Nicolò Lai

Academic Year 2020/2021

Abstract

Advanced, optimized and well-performed data quality monitoring tasks in modern physics experiments are becoming essential, as the complexity of collected data increase with the development of more sophisticated detectors. In this thesis work, we have exploited neural networks to test the beginnings of an automated data quality monitoring procedure on drift tubes data. The algorithm uses statistical hypothesis testing to compare the monitored data sample with a reference dataset and returns information on the level of agreement between the two. The performance tests we have carried out show that our implementation is correctly working: the neural network detects anomalies in data when they are present and satisfactorily recognizes when data is of good quality.

Sommario

Lo sviluppo di rivelatori sempre più sofisticati per i moderni esperimenti di fisica, ed il conseguente aumento della complessità dei dati raccolti, richiede tecniche avanzate e ottimizzate di controllo qualità dei dati, le quali ricoprono perciò un ruolo cruciale. In questo lavoro di tesi, per testare un'innovativa procedura automatica di monitoraggio dati, sono state sfruttate reti neurali applicandole a dati provenienti da camere a deriva. L'algoritmo implementato utilizza test di ipotesi per comparare i dati da monitorare rispetto ad un campione di riferimento, restituendo informazioni riguardo il livello di accordo tra i due. Sono stati effettuati test per verificare le prestazioni dell'algoritmo: i risultati mostrano che, quando nei dati sono presenti anomalie qualitative, la rete è in grado di rilevarle e quantificarle. In caso contrario, il modello riconosce in modo soddisfacente l'accordo con il campione di riferimento.

Contents

Introduction

The realm of this thesis work is data quality monitoring (DQM for short), which deals with being able to tell whether data in a given dataset have been appropriately collected so that we can extrapolate relevant results from them, or acquired poorly, and in the latter case, datasets may also be useless in their entirety. As experiments grow more extensive, and a larger amount of data is being collected, the already crucial role of DQM turns out to be essential in modern physics experiments.

Data quality monitoring tasks have always been performed since the very first experiments: the scientist tests the experimental setup on a well-known configuration, then visually compares the collected data with his expectations and, if it looks like to him that the setup is working correctly, the scientist begins with his experiment. Of course, through time, DQM procedures have evolved consistently to keep up with the ever-increasing complexity of the experiments and the data being acquired. However, it is pretty common to encounter many human performed tasks, from visually comparing data to running advanced statistical tests. As datasets get more extensive and more complex, it could become a highly demanding and time-consuming job.

Our main goal is thus to minimize the human efforts in data quality monitoring tasks, and we intend to exploit machine learning techniques. Machine learning is the science of getting computers to perform operations without being explicitly programmed. It is a branch of artificial intelligence that aims to imitate the way humans learn, gradually improving its accuracy. It has become a most pervasive topic, finding applications in every branch of science and playing a substantial role in modern experimental physics.

In this thesis work, we have exploited a deep learning algorithm based on the maximum likelihood principle: it compares the collected data with a given reference dataset and returns the level of agreement between the two in the form of a test statistic. It has been tested on data quality monitoring tasks using datasets coming from the realm of particle physics. Precisely, we have been working on a cosmic muon telescope, a drift tubes detector, located at the Legnaro INFN National Laboratories and reproducing a small-scale replica of the CMS's muon chambers. The experimental setup and the employed datasets are introduced and described in Chapter 1. In Chapter 2 we offer a brief explanation of the algorithm, providing a brief yet crucial conceptual background and a general overview of its functioning. Lastly, the DQM performance of the algorithm is reported in Chapter 3, along with its technical implementation and prospects.

Chapter 1

Experimental Setup and Datasets

Let us introduce the experimental setup first: in this chapter, we discuss the detector and its data acquisition system, along with some data preprocessing. Then, we present the data distribution we have been working with and used to test our data quality monitoring algorithm.

1.1 Drift tube detector

The whole detector consists of four muon chambers, also called superlayers (SLs). Each muon chamber comprises 64 drift tubes (DTs), grouped into four staggered layers of 16 drift cells. A visual representation of the detector configuration is shown in Figure 1.1, in which only 20 out of 64 DTs have been reported. The transverse cross-section of the drift cells is $L \times h = 42 \times 13\,\text{mm}^2$ and they are filled with an Ar-CO_2 (85/15%) gas mixture at 1 atm. Each DT contains an anodic wire at a high voltage potential ($V_{\text{wire}} = +3600\,\text{V}$), while the sidewalls of the tubes are cathodes at $V_{\text{cathod}} = -1800\,\text{V}$. Furthermore, two additional electrodes at $V_{\text{strip}} = -1200\,\text{V}$ are placed on the top and bottom wall of the tube to ensure the uniformity of the electric field inside the cell.

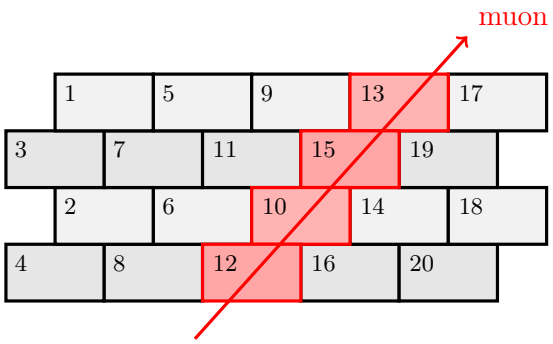


Figure 1.1: Detector staggered layers configuration

The active volume of the detector is the gas mixture, which gets ionized by ionizing particles. Electrons knocked off the gas, surrounded by the uniform electric field as described above, drift towards the anodic wire at an almost constant velocity of $v_{\text{drift}} \approx 54\,\mu\text{m/ns}$. As the electrons reach the wire, the deposit of electric charge produces an electrical signal. The actual quantity being measured is the time t of the electric signal detection, from which one could extrapolate information about the muon arrival time and the drift time t_{d}. The drift time corresponds to the time taken by the electrons to reach the anodic wire and, since the uniformity of the electric field leads to a constant drift velocity v_{drift}, one could easily compute the distance, which we call x, between the hit position of the muon and the wire.

$$x = v_{\text{d}} \times t_{\text{d}} = v_{\text{d}} \times (t - t_0) \tag{1.1}$$

As shown in Equation 1.1, in order to compute the drift time t_{d}, a time pedestal t_0 should be subtracted from the detection time t: such t_0 is a parameter holding contributions from both trigger time and electronics delay in detecting and collecting data.

1.2 Data acquisition and format

The electric charge accumulated on the anodic wire is amplified, shaped and discriminated given a predefined threshold by the front-end (FE) electronics located within the active volume of the tubes. Signals above the discriminating threshold are sent to two FPGAs that implement the time-to-digital conversion (TDC), assigning each activated channel a time value t. A pair of scintillator tiles provide an external timing reference. It is noticeable that a single FPGA can digitize the entirety of the channels from two muon chambers, and the additional TDCs available are used for digitizing external signals such as the scintillator. In addition, an external oscillator generates a clock signal, sent to both FPGAs: to emulate CERN TTC Systems a 40 MHz clock, compatible with the bunch crossing (BX) frequency at LHC, is generated.

Practically speaking, the time-stamp t returned by the TDC corresponds to the rising edge of the input signal with respect to the rising edge of the clock. On top of that, an appropriate counter assigns to each TDC measurement its corresponding BX counter: the granularity of the time measures is thus 1/30 of BX. Finally, another counter, related to the orbit of protons around the accelerator ring at LHC, is assigned to each BX. The final result is a time measurement format similar to standard time (hours : minutes : seconds), and the conversion to nanoseconds is given by

$$t = \text{ORBIT\_CNT} \times 3564 \times 25 + \text{BX\_COUNTER} \times 25 + \text{TDC\_MEAS} \times 25/30 \tag{1.2}$$

The raw data is then preprocessed to produce .txt files having as rows all the collected signals above the discriminating threshold and as columns the following features:

- **HEAD**
  Additional information discriminating different kinds of data.
- **FPGA**
  Specifies which of the two FPGAs collected the signal.
- **TDC_CHANNEL**
  Integer value in the interval [0, 127] referring to the FPGA channels.
- **ORBIT_CNT**
  Integer in the interval [0, 2^{32}].
- **BX_COUNTER**
  Integer in the interval [0, 3564].
- **TDC_MEAS**
  Integer in the interval [1, 30].

The first feature is only used to discriminate between relevant data and unnecessary data. For example, the experiment has other triggers, and their signals are stored in the same data file and distinguished by a different HEAD tag. The detector itself is a great case study for many different works of experimental physics, but of course, not all data is necessary for every kind of task we want to tackle. The second and third feature, instead, allows for tracking which drift tube has collected the signal: the FPGA tag (either 0 or 1) locates the muon chambers in pairs (each FPGA board is connected to two superlayers), while the TDC_CHANNEL, meaning the channel of the FPGA, locates the drift tube. Finally, the last three features hold the timing information as described above.

Note that a more detailed description of the experimental setup, namely the detector and its electronic chain, is accurately reported in [1]. However, as it is not crucial to our work, we do not elaborate further on the experimental setup in this thesis.

1.3 The drift time distribution

The distribution we are interested in monitoring is the distribution of drift times. It is the collection of time taken by electrons to drift towards the wire at the center of the DTs. As mentioned in Section 1.1, in order to compute the drift time t_d we need to follow a few steps. The trigger-less acquisition system reads the sensors every 25 ns and each signal above the predefined threshold is recorded. Thus we first need to discriminate events from background signals.

We define an *event* as the collection of all the electric signals produced by a muon crossing the detector. Since one orbit (the *ORBIT_CNT* counter) corresponds to approximately 90 µs, the actual probability of having two muons traversing the detector within the same orbit is low. Thus, we can assume that all the hits that have the same *ORBIT_CNT* value of an external trigger signal can be gathered into an *event*. The external trigger used is the scintillator, characterized by the special channel 128 of the second FPGA.

Once all the events within the dataset have been discriminated from the background noise, we need to convert the time-stamp values (assigned by the TDC) into nanoseconds using Equation 1.2. We will call the scintillator time stamp, expressed in nanoseconds, as t_{scint}. Then, as stated in Section 1.1, to compute the Drift Time of one hit, we should subtract the time pedestal t_0 from the hit time t:

$$t_d = t - t_0 = t - (t_{\text{scint}} + \text{scint\_offset} + \text{SL\_offset}) \tag{1.3}$$

where scint_offset is related to electronics delays while SL_offset follows from the geometry of the detector, as muons pass through the superlayers with different times depending on the particle track.

Drift time distributions are expected to be uniform, ranging between 0 ns and approximately 400 ns, and they are often referred to as *time boxes*. Hence, by plotting the $t - t_{\text{scint}}$ distribution we find the overall offset, composed by the last two terms if Equation 1.3, by shifting the distribution to have the rising edge at roughly 0 ns. An example is shown in Figure 1.2, in which the offset has been roughly calculated to be 100 ns. A more accurate and comprehensive study on the calibration of the time pedestals is reported in both [2] and [3]: although these articles refer to the CMS's muon chambers, a similar procedure can be implemented to accurately compute the calibration constants. For our purposes, however, a rough calibration is sufficient to prevent the procedure from failing.

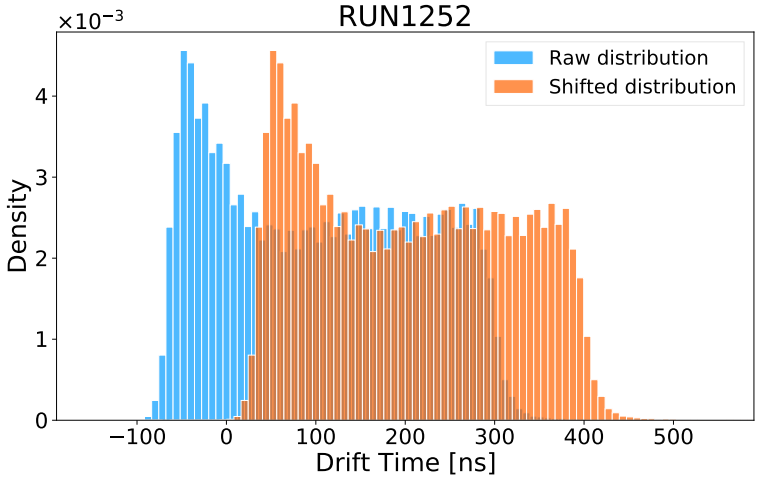


Figure 1.2: Example of a drift time distribution before and after tuning the offset parameter

Among plenty of different distributions, we have chosen to work with drift times as the shape of the time box is strictly related to the correct functioning of the detector. Numerous disparate factors can affect that distribution, from the data acquisition system to the DTs gas mixture proportions, pressure and more. Thus, if the detector is working as expected, the drift time distribution should look like the one in Figure 1.2. On the other hand, a detector failure leads to a time box with different widths or discrepancies in the edges and tails.

Chapter 2

The Deep Learning Algorithm

This chapter focuses on discussing the deep learning algorithm that has been implemented and tested throughout the work. Although we have been using this algorithm with a data quality monitoring goal in mind, it stems from the search for new physics at LHC. The fact that the algorithm can be employed in both fields clearly shows the sound flexibility of both the conceptual foundations and the practical implementation. Hence, the first part of this chapter focuses on delivering a general introduction to the algorithm's statistical background. Then, we share an overview of its actual functioning and how it can be employed in data quality monitoring.

2.1 Conceptual foundations

The whole construction of the statistical background that underpins the algorithm is accurately discussed in *Learning New Physics from a Machine (2018)* by R. T. D'Agnolo and A. Wulzer [4]. Thus, we give only a brief recap on the construction of hypothesis testing with neural networks involved.

Let us consider then a dataset of repeated measurements $\boldsymbol{\mathcal{D}} = \{x_i\}_{i=1\,\dots\,\mathcal{N}_{\boldsymbol{\mathcal{D}}}}$ of a d-dimensional random variable x. Let $n(x\,|\,\boldsymbol{\mathcal{R}})$ be its expected differential distribution as predicted by the reference model $\boldsymbol{\mathcal{R}}$ ($\equiv H_0$, i.e. the null hypothesis): it can be written as

$$n(x\,|\,\boldsymbol{\mathcal{R}}) = \mathcal{N}(\boldsymbol{\mathcal{R}})\; p(x\,|\,\boldsymbol{\mathcal{R}}) \tag{2.1}$$

where $p(x\,|\,\boldsymbol{\mathcal{R}})$ is the probability density function related to the reference hypothesis and $\mathcal{N}^{\text{exp}}_{\boldsymbol{\mathcal{R}}}$ is the total number of expected events to be found in the experimental dataset according to $\boldsymbol{\mathcal{R}}$, given by

$$\mathcal{N}(\boldsymbol{\mathcal{R}}) = \int n(x\,|\,\boldsymbol{\mathcal{R}})\,\mathrm{d}x \tag{2.2}$$

If the dataset $\boldsymbol{\mathcal{D}}$ follows the reference hypothesis, then the total number of events observed in $\boldsymbol{\mathcal{D}}$ (referred to as $\mathcal{N}_{\boldsymbol{\mathcal{D}}}$) is a random variable itself, following a Poisson distribution with $\mu = \mathcal{N}(\boldsymbol{\mathcal{R}})$.

However, to test the reference model for compatibility with the observed dataset $\boldsymbol{\mathcal{D}}$, it is necessary to introduce an alternative hypothesis. In a model-independent framework, the alternative density distribution is parameterized employing a flexible neural network: $n(x\,|\,\mathbf{w})$, where $\mathbf{w}$ are the free parameters of the network.

Moreover, it is convenient to parametrize $n(x\,|\,\mathbf{w})$ in terms of $n(x\,|\,\boldsymbol{\mathcal{R}})$, as the most interesting problems are those in which the underlying distribution of data is similar to the reference one. Considering that both functions need to be positive, a valid possibility is to write

$$n(x\,|\,\mathbf{w}) = n(x\,|\,\boldsymbol{\mathcal{R}})\; e^{\,f(x;\,\mathbf{w})} \tag{2.3}$$

where $f(x;\,\mathbf{w})$ is the output of the NN implemented to specify the alternative hypothesis. At this point, Equation 2.3 defines the space of all possible hypothesis that contains the case in which $f(x;\,\mathbf{w}) \equiv 0\,\forall x$, crucial for the validity of the Neyman-Pearson approach that follows. According to the Neyman-Pearson construction, the optimal statistical test for the reference model is based on the maximum

likelihood principle[1]. The plan is to compare the reference density $n(x \,|\, \boldsymbol{\mathcal{R}})$ with the best-fit distribution $n(x \,|\, \widehat{\mathbf{w}})$ with $\mathbf{w} = \widehat{\mathbf{w}}$ the parameter configuration that maximizes the likelihood. In other words, the reference hypothesis is compared to the alternative hypothesis that maximizes the likelihood among all possible configurations given by the flexible parametrization implemented.

Thus, the test statistic is given by Equation 2.4. It can be re-written in a cleaner form as in Equation 2.5: $\mathcal{N}(\boldsymbol{\mathcal{R}})$ and $\mathcal{N}(\mathbf{w})$ represent the total number of events expected by the reference hypothesis and by the alternative hypothesis respectively.

$$t(\boldsymbol{\mathcal{D}}) = 2 \log \left[\frac{e^{-\mathcal{N}(\widehat{\mathbf{w}})}}{e^{-\mathcal{N}(\boldsymbol{\mathcal{R}})}} \prod_{x \in \boldsymbol{\mathcal{D}}} \frac{n(x \,|\, \widehat{\mathbf{w}})}{n(x \,|\, \boldsymbol{\mathcal{R}})} \right] \tag{2.4}$$

$$= -2 \min_{\mathbf{w}} \left[\mathcal{N}(\mathbf{w}) - \mathcal{N}(\boldsymbol{\mathcal{R}}) - \sum_{x \in \boldsymbol{\mathcal{D}}} f(x;\, \mathbf{w}) \right] \tag{2.5}$$

The algorithm then compares, exploiting neural networks, a given dataset $\boldsymbol{\mathcal{D}}$ with the reference model expected density $n(x \,|\, \boldsymbol{\mathcal{R}})$. According to the parametrization of $n(x \,|\, \mathbf{w}) \propto n(x \,|\, \boldsymbol{\mathcal{R}})$ shown in Equation 2.3 it is possibile to turn the test statistic expression, given in Equation 2.5, into a loss function for neural networks without needing the explicit expression of $n(x \,|\, \boldsymbol{\mathcal{R}})$. As a matter of fact, the prediction does not usually come in analytical form, but rather in the form if a reference sample $\boldsymbol{\mathcal{R}} = \{x_i\}_{i=1 \,\dots\, \mathcal{N}_{\boldsymbol{\mathcal{R}}}}$ following the reference distribution. However, the expected $\mathcal{N}(\mathbf{w})$ can be estimated through Monte Carlo integration methods as

$$\mathcal{N}(\mathbf{w}) = \frac{\mathcal{N}(\boldsymbol{\mathcal{R}})}{\mathcal{N}_{\boldsymbol{\mathcal{R}}}} \sum_{x \in \boldsymbol{\mathcal{R}}} \frac{n(x \,|\, \mathbf{w})}{n(x \,|\, \boldsymbol{\mathcal{R}})} = \frac{\mathcal{N}(\boldsymbol{\mathcal{R}})}{\mathcal{N}_{\boldsymbol{\mathcal{R}}}} \sum_{x \in \boldsymbol{\mathcal{R}}} e^{\, f(x; \mathbf{w})} \tag{2.6}$$

where the reference sample $\boldsymbol{\mathcal{R}}$ dimension should be significantly larger than $\boldsymbol{\mathcal{D}}$ to neglect the statistical fluctuations of $\boldsymbol{\mathcal{R}}$ (when compared with the fluctuations of $\boldsymbol{\mathcal{D}}$) and to perform an accurate integration of the density function predicted by the alternative hypothesis. Thus Equation 2.5 becomes

$$t(\boldsymbol{\mathcal{D}}) = -2 \min_{\mathbf{w}} \left[\frac{\mathcal{N}(\boldsymbol{\mathcal{R}})}{\mathcal{N}_{\boldsymbol{\mathcal{R}}}} \sum_{x \in \boldsymbol{\mathcal{R}}} \left(e^{\, f(x; \mathbf{w})} - 1 \right) - \sum_{x \in \boldsymbol{\mathcal{D}}} f(x;\, \mathbf{w}) \right] \tag{2.7}$$

$$\equiv -2 \min_{\mathbf{w}} L\left[f(\,\cdot,\, \mathbf{w}) \right] \tag{2.8}$$

where $L\left[f(\,\cdot,\, \mathbf{w}) \right]$ has the form of a loss function, and its minimization with respect to NN parameters can be performed as a standard supervised training process. Hence, it is convenient to write L as a single sum over the events: introducing a target variable $y = \{0,\, 1\}$ for events in $\boldsymbol{\mathcal{R}}$ and $\boldsymbol{\mathcal{D}}$ respectively, the loss function can be re-written as

$$L[f] = \sum_{(x,\, y)} \left[(1 - y) \, \frac{\mathcal{N}(\boldsymbol{\mathcal{R}})}{\mathcal{N}_{\boldsymbol{\mathcal{R}}}} \left(e^{\, f(x; \mathbf{w})} - 1 \right) - y \, f(x;\, \mathbf{w}) \right] \tag{2.9}$$

After the training process, the trained NN has learned the maximum likelihood fit to the log-ratio between the expected density predicted by the best alternative model and the expected density predicted by the reference model

$$f(x;\, \widehat{\mathbf{w}}) = \log \left[\frac{n(x \,|\, \widehat{\mathbf{w}})}{n(x \,|\, \boldsymbol{\mathcal{R}})} \right] \approx \log \left[\frac{n(x \,|\, \boldsymbol{\mathcal{T}})}{n(x \,|\, \boldsymbol{\mathcal{R}})} \right] \tag{2.10}$$

where $n(x \,|\, \boldsymbol{\mathcal{T}})$ is the true underlying data distribution, whose log-ratio with the reference density is approximated by the NN output $f(x;\, \widehat{\mathbf{w}})$.

[1]Let us consider two hypothesis $H_0 : \theta = \theta_0$ and $H_1 : \theta = \theta_1$. Thus, the likelihood-ratio test is

$$\Lambda(x) = \frac{\mathcal{L}(\theta_0 \,|\, x)}{\mathcal{L}(\theta_1 \,|\, x)}$$

with some threshold $\eta \geq 0$ for which H_0 is rejected in favor of H_1 at a precise significance level $\alpha = p(\Lambda(x) \leq \eta \, H_0)$. The Neyman-Pearson lemma states that $\Lambda(x)$ is the best hypothesis test at such significance level α.

2.2 Algorithm overview

The neural network takes as input two datasets: the first one contains data following the reference model $\boldsymbol{\mathcal{R}}$ while the second one represents the experimental dataset $\boldsymbol{\mathcal{D}}$. The algorithm compares the two distributions during the training process and learns to approximate the expected density log-ratio. The procedure aims at computing the test statistic $t(\boldsymbol{\mathcal{D}})$, proportional to the loss function final value (Equation 2.8), as it can be exploited to quantify the agreement between $\boldsymbol{\mathcal{R}}$ and $\boldsymbol{\mathcal{D}}$.

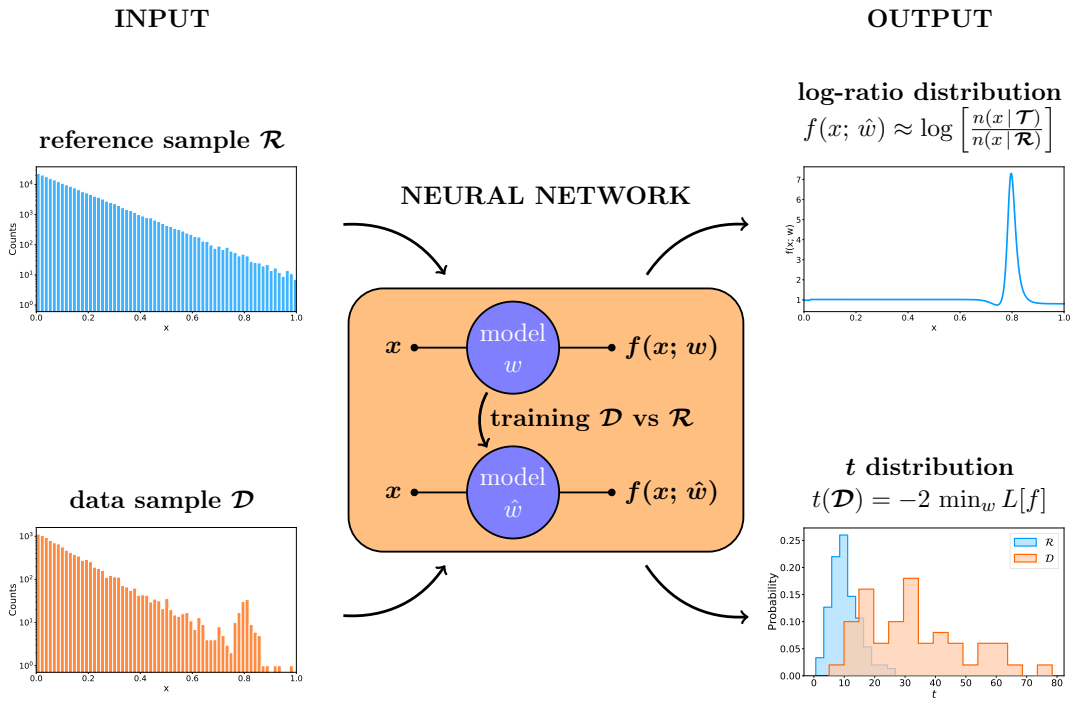


Figure 2.1: Visual representation of the algorithm

In order to quantify the probability that a discrepancy is present in $\boldsymbol{\mathcal{D}}$, it is necessary to compute the probability distribution $p(t \mid \boldsymbol{\mathcal{R}})$ (i.e., the distribution of the test statistic when the reference hypothesis is true). Hence, repeated training processes are performed on datasets $\boldsymbol{\mathcal{D}}_{\boldsymbol{\mathcal{R}}}$ sampled from the reference model itself. A convenient result by S. Wilks [5] ensures that, as the size of the data samples, tested using the likelihood ratio, approaches infinity, the distribution of the test statistic under the null hypothesis asymptotically approaches a χ^2 distribution. The number of degrees of freedom of the distribution is the difference between the parameter space dimensionality of the two hypotheses. In this case, with neural networks involved, the number of degrees of freedom is equal to the number of trainable parameters of the network. Thus, if the training process has adequately worked, the distribution of the test statistic under the null hypothesis (i.e., computed using $\boldsymbol{\mathcal{D}}_{\boldsymbol{\mathcal{R}}}$ datasets) should be compatible with the χ^2_ν distribution, where ν stands for the degrees of freedom of the network. Otherwise, more accurate tuning of the network's hyperparameters needs to be performed[2].

When the NN hyperparameters are tuned correctly, the network is ready to perform on real datasets $\boldsymbol{\mathcal{D}}$ coming from experimental sources. At this point, the procedure described above is replicated, and the test statistic returned by the network will be called t_{obs}. As previously anticipated, to quantify the probability that a discrepancy is present in $\boldsymbol{\mathcal{D}}$ it is possible to compare t_{obs} with the $p(t \mid \boldsymbol{\mathcal{R}})$ distribution. By defining the observed p-value as

$$p_{\text{obs}} = \int_{t_{\text{obs}}}^{+\infty} p(t \mid \boldsymbol{\mathcal{R}}) \, \mathrm{d}t \tag{2.11}$$

and its corresponding significance $Z(p) = \Phi^{-1}(1-p)$, where Φ^{-1} is the quantile of a normal distribution with $\mu = 0$ and $\sigma = 1$, a discrepancy in $\boldsymbol{\mathcal{D}}$ between $\boldsymbol{\mathcal{R}}$ would manifest itself as a large values of Z.

[2] *Hyperparameters* are the variables that determine the network structure and how the network is trained. A few examples are the number of hidden layers, the number of neurons in each layer, and the activation functions.

Chapter 3

Data Quality Monitoring using Deep Learning techniques

This chapter will run through the actual process of implementing the algorithm to perform data quality monitoring tasks. We begin with a brief introduction to DQM, then discuss reference and data samples, technicalities about NN parameters and, finally, the results of this work.

3.1 Data quality monitoring

Where there is data, there is the risk of bad data, and bad data is expensive. It could be expensive in terms of time, but also terms of money. To ensure that the data being collected is of reasonable quality, we need to monitor data and get notified in time to correct whatever is damaging our data quality, or, even better, have the whole monitoring process automated. The latter is what we aim to achieve, but it is a steep path.

Ideally, we would look for a fully automated, online data quality monitor. In other words, we would like to have an algorithm that independently recognizes whether the collected data is acceptable or not while the detector is collecting data. Moreover, the algorithm would raise a warning or an error whenever data is of poor quality, depending on how dangerous the quality loss may be. Eventually, we would have it take action and even shut down the detector if necessary.

We have been focusing our attention on the very first part of the last two sentences during this thesis work. Namely, we wanted to have an algorithm that can discriminate between good quality and bad quality data. Unfortunately, tests have been performed offline, meaning that data had already been collected, and no automation aside from the actual recognition has been implemented.

3.2 Monitoring the time box

As briefly anticipated at the end of Section 1.3, the data distribution chosen to be our test case is the time box, the distribution of electrons' drift times. This choice was not made by chance: time box distribution features, such as the overall shape, tails and edges, are correlated to the correct working of the detector (i.e., if the detector has some failures, those can be spotted in a time box showing statistically significant departures from the expected distribution).

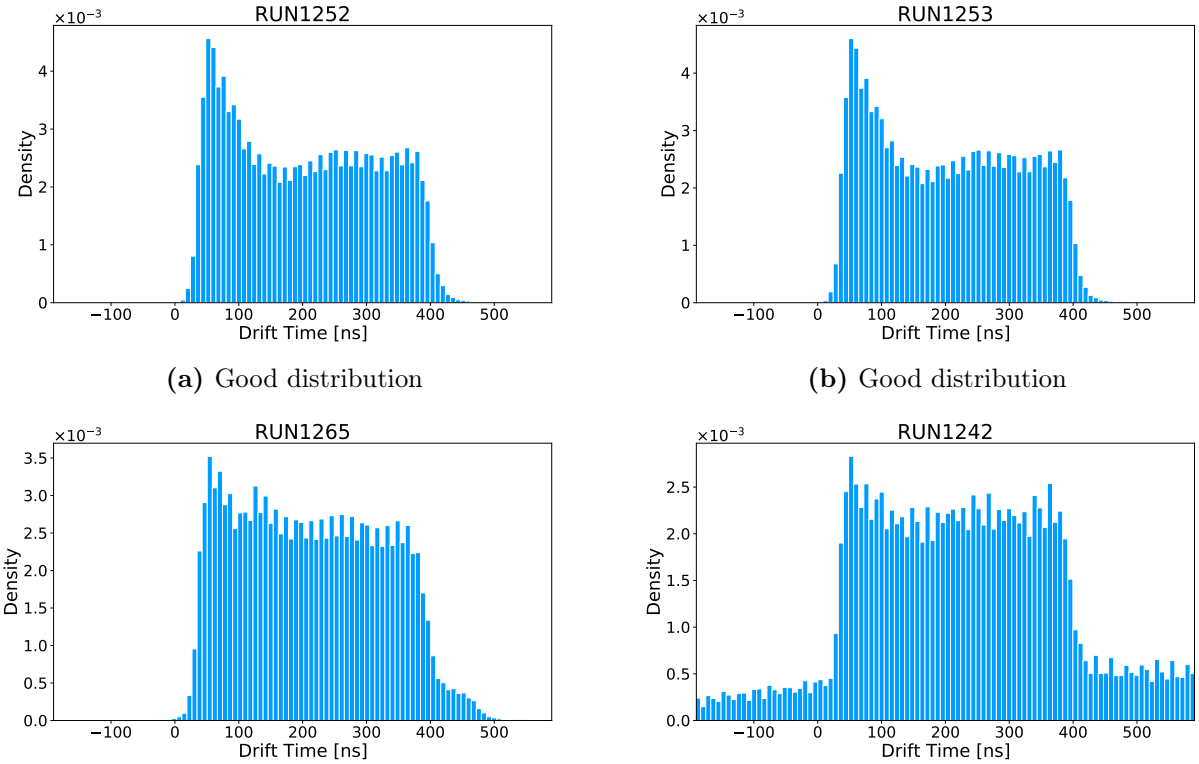
3.2.1 Experimental datasets

In the time frame in which we have worked on this thesis, the detector has collected almost 100 Gb of data acquired in multiple runs. These runs are different as they have been used to perform other tests on the experimental setup settings. However, the specifics of the runs were not known by us, so we had a flat prior to the distributions, and it was impossible to introduce any bias towards them. It seemed a good plan to us, as the whole work has been carried out in a model-independent framework.

Hence, to choose a few distributions on which to do our monitoring tests, we could not simply pick a detector's run with known working settings as our reference dataset and select others that did not

share such settings. Thus, we first selected the runs that collected noticeably larger datasets (e.g., usually ~ 20 hours runs or shorter ones collected at a higher rate). Then, we have analyzed the time box of each of those high statistics runs: our physics background and the standard working conditions suggested that the drift times distribution should be almost uniform, ranging from 0 to 400 ns (as previously explained in Section 1.3). We realized that all the time boxes analyze could fit in three categories: *good*, *discrepant* and *bad* runs.

Good runs are those that agree with our expectations, both in shape and width. Discrepant runs show some departures from the expected distribution, either in shape or width. Bad runs do not follow our predictions, both in shape and width. To clarify, in Figure 3.1 are shown four examples of time boxes. Those four distributions are the ones being tested.



(a) Good distribution

(b) Good distribution

(c) Discrepant distribution

(d) Bad distribution

Figure 3.1: Time boxes tested with our deep learning algorithm

Briefly, Figure 3.1a and Figure 3.1b are both *good* runs. Note that they share a peak placed on the left edge of the distribution, caused by the detector's geometry, which is also expected. The fact that RUN1252 and RUN1253 look the same when plotted using a 100-bins histogram tells us that statistical fluctuations are almost imperceptible to sight with that amount of data. On the other hand, Figure 3.1c falls into the *discrepant* category: though the overall shape is almost acceptable, the right-hand tail is noticeably out of range. Finally, it is more than evident that Figure 3.1d does not agree with our expectations at all: it seems that the detector collected a relevant background noise component that ruined the width of the distribution.

3.3 Implementing the deep learning algorithm

Referring to Figure 2.1 in Section 2.2, the algorithm takes as input two distributions: the reference sample $\boldsymbol{\mathcal{R}}$ and the data sample $\boldsymbol{\mathcal{D}}$. Thus, in our scenario, we first needed to identify a reference distribution: it made sense to choose as $\boldsymbol{\mathcal{R}}$ data coming from a *good* run: this way, the neural network would compare any given $\boldsymbol{\mathcal{D}}$ with our idea of what a time box should be. We have selected RUN1252 (Figure 3.1a) as our reference run.

Then, before testing the algorithm's performance on the remaining three drift time distributions,

we had to build the deep learning model: the architecture of the network, its activation functions, the optimizer. After that, NN's hyperparameters were to be tuned: we need to ensure some degree of compatibility between the distribution of the test statistic t under the null hypothesis and the χ^2_ν distribution, where ν stands for the degrees of freedom of the network. Thus, the standard procedure for optimizing hyperparameters involves feeding the algorithm with the reference $\boldsymbol{\mathcal{R}}$ and a sub-sample $\boldsymbol{\mathcal{D}}_{\boldsymbol{\mathcal{R}}}$ for many different $\boldsymbol{\mathcal{D}}_{\boldsymbol{\mathcal{R}}}$s. Then, testing for the compatibility between $p(t \,|\, \boldsymbol{\mathcal{R}})$ and χ^2_ν, and repeating the procedure with different values for the hyperparameters until satisfactory compatibility is reached.

3.3.1 Technical implementation of the model

A sufficient amount of $\boldsymbol{\mathcal{D}}_{\boldsymbol{\mathcal{R}}}$ distributions sampled from the reference dataset are necessary to tune the network hyperparameters successfully. Moreover, a changing yet significant amount of training epochs are needed to reach good compatibility with the expected χ^2_ν distribution. Thus, running the algorithm turns out to be computationally heavy: tasks parallelization or distribution is mandatory to achieve the desired results in a reasonable amount of time. Then, a cluster of machines, located in Legnaro and Padova, was used, and jobs were submitted via Condor. The t2-ui-12 machine was accessed via ssh tunnel through the INFN gate to submit the jobs to the batch system.

The algorithm has been implemented in Python using the Keras module. The optimizer used to minimize the loss function is ADAM, as implemented in Keras with the TensorFlow backend, with an initial learning rate set to 10^{-3} and parameters $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-7}$. The batch size has been kept fixed to cover the whole training sample throughout the work.

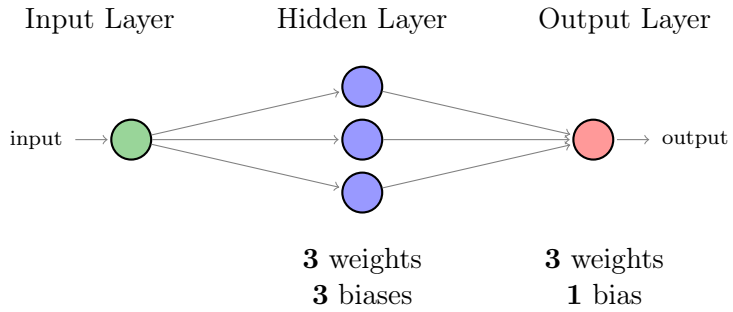


Figure 3.2: Architecture of the implemented neural network with explicit counting of free parameters

The NN architecture that has been chosen is quite a simple one: three layers (one input layer, one hidden layer and one output layer), with the hidden layer having three neurons. The number of degrees of freedom of the network, which is assumed to be equal to the number of free parameters, follows

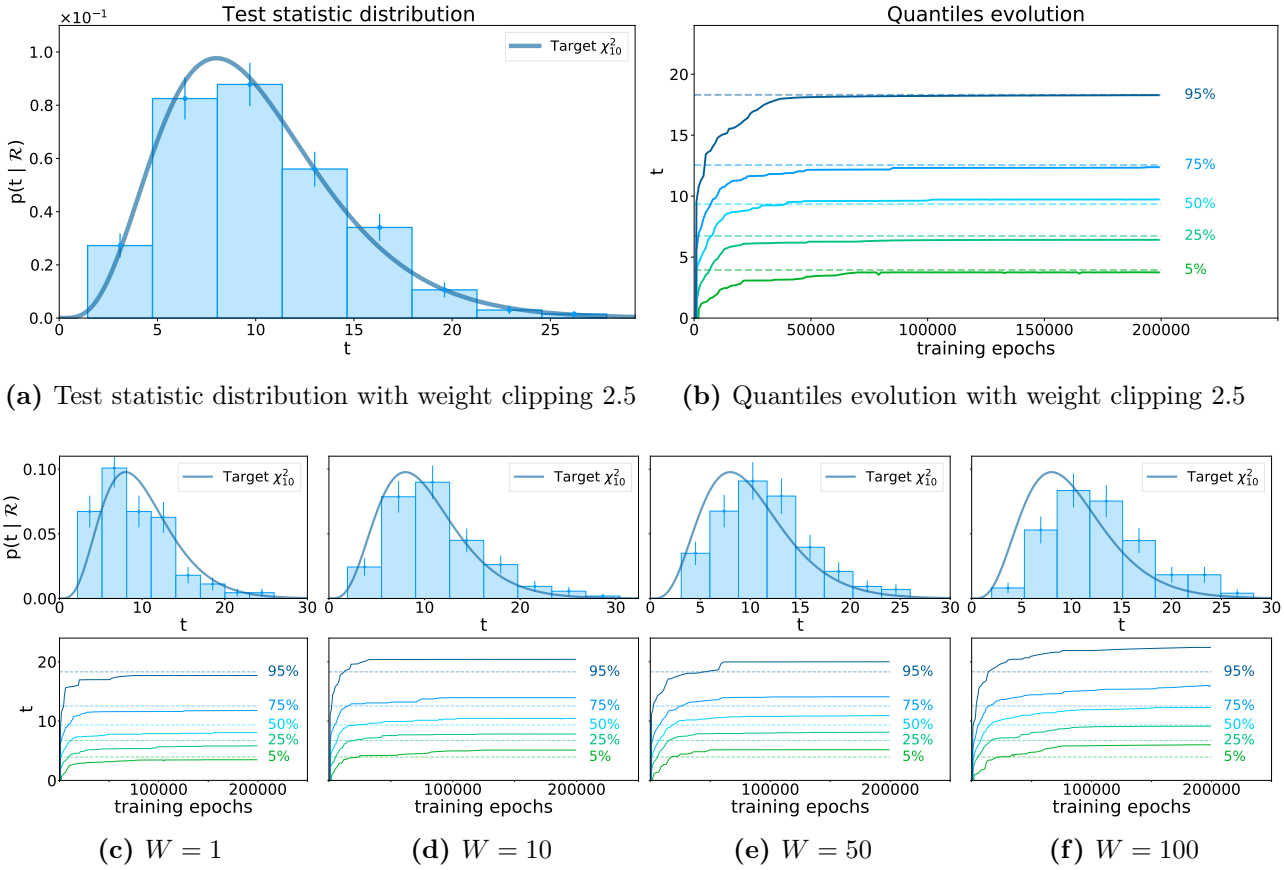
$$N_{\text{par}}(\vec{a}) = \sum_{n=1}^{L-1} a_n \, (a_{n-1} + 1) \tag{3.1}$$

where L is the number of layers and a_n represents the number of neurons for the n-th layer. Thus, for the NN employed in this work $(1 \times 3 \times 1)$, the number of free parameters is 10. A visual representation of the NN architecture and an explicit counting of the free parameters is shown in Figure 3.2.

Regarding activation functions, several have been tested. The internal activation (i.e., the hidden layer activation) that performed the best is the hyperbolic tangent. On the other hand, although a valid alternative, the sigmoid activation makes the network less elastic and converges slower to the final estimate of t. Ultimately, the algorithm necessarily requires some regularization, as the loss function (2.9) is unbounded from below, meaning that it approaches $-\infty$ if the output of the network diverges for some values of $x \in \boldsymbol{\mathcal{D}}$. The solution that has been implemented is the application of the so-called *weight clipping* W. In other words, the dangerous possibility for the loss function to reach $-\infty$ is avoided by enforcing an upper bound on the absolute value of each weight. It has to be stressed that the correct choice of this parameter W is crucial to get reasonable results. If the constraint is set too tight, then the distribution of the test statistic in the reference hypothesis will turn out to be shifted to the left of the expected χ^2_ν. On the other hand, if the constraint is set too loose, the t distribution will be shifted to the right of χ^2_ν.

3.3.2 Tuning the neural network

As previously anticipated, to accurately tune the hyperparameters, we need to feed the algorithm with a large reference sample and different, smaller reference sub-samples. Considering our reference run to be RUN1252, we've sampled from the entire dataset a $\mathcal{N}_{\boldsymbol{\mathcal{R}}} = 200000$ reference sample and 400 $\mathcal{N}_{\boldsymbol{\mathcal{D}}} = 3000$ data samples. For different values of the weight clipping parameter, ranging from 1 to 100, we have trained the neural network and selected the W value that maximizes the compatibility between the $t(\boldsymbol{\mathcal{D}})$ distribution and the χ^2_{10} distribution. The empirical $p(t\,|\,\boldsymbol{\mathcal{R}})$ distributions obtained in this way after 200000 training epochs, and some of its percentiles as a function of the number of epochs, are reported in Figure 3.3.



(a) Test statistic distribution with weight clipping 2.5 **(b)** Quantiles evolution with weight clipping 2.5

(c) $W = 1$ **(d)** $W = 10$ **(e)** $W = 50$ **(f)** $W = 100$

Figure 3.3: Test statistic distributions after 200000 training epochs and evolution of their quantiles during the training process with an optimal weight clipping parameter (above) and other weight clipping values (below)

We see that for large values of the weight clipping W (Figure 3.3f), the distribution looks shifted to the right of the target χ^2 with 10 degrees of freedom. Moreover, the evolution of the quantiles tells us that the training is not stable: the quantiles (especially the 95% one) do not achieve a plateau. On the other hand, small values of weight clipping (Figure 3.3c) make the distribution stable with training but slightly shifted towards the left of the χ^2_{10} expectation. Instead, a good compatibility is obtained with a weight clipping equal to 2.5 (Figure 3.3a and Figure 3.3b): the quantiles reach a plateau near the quantiles of the target χ^2_{10} and the distribution reproduces the χ^2_{10} formula quite accurately.

The first strategy that we have implemented to search for the optimal weight clipping parameter is thus the following:

- Starting from a large value of W (in our case $W = 100$) decrease it until the 95% quantiles reaches a plateau as a function of the training epochs (in our case $W_{\text{max}} = 50$ in Figure 3.3e).
- In the range of weight clippings below W_{max} (where the plateau is always reached) choose the largest W that also gives the highest compatibility between $p(t\,|\,\boldsymbol{\mathcal{R}})$ and χ^2_{10}.

It is essential to set the weight clipping parameter as large as possible to maximize the expressive

power of the network. However, a refined strategy has been applied to reduce the computational burden of the above search for W:

- Starting from a large and a small value of W (in our case $W = 100$ and $W = 1$), train the network on a limited number of $\boldsymbol{\mathcal{D}}_{\boldsymbol{\mathcal{R}}}$ datasets (40 for example).
- Compute the median of the $p(t \,|\, \boldsymbol{\mathcal{R}})$ distribution: it will be above the median of the target χ^2_{10} distribution for the large value of W and below the median of the target χ^2_{10} distribution for the small value of W.
- In the window of weight clippings that produce a $p(t \,|\, \boldsymbol{\mathcal{R}})$ distribution with median similar to the χ^2_{10} one, replicate the first two bullet points using a larger number of $\boldsymbol{\mathcal{D}}_{\boldsymbol{\mathcal{R}}}$ datasets (150 for example).
- Employ a compatibility test to pick up the optimal weight clipping value inside the window. In our case, we have chosen to perform a Kolmogorov-Smirnov (KS) test and maximized (in the window of optimal weight clippings) the p-value for the compatibility between the $p(t \,|\, \boldsymbol{\mathcal{R}})$ distribution and the target χ^2_{10}.
- Consider using an even larger amount of $\boldsymbol{\mathcal{D}}_{\boldsymbol{\mathcal{R}}}$ datasets (400 in our case) to ultimately test the validity of the S. Wilks result and obtain a KS p-value above a given threshold.

Note that this refined strategy, even though not explicitly, finds the largest W that leads to a 95% quantile plateau and maximizes the compatibility between the distributions.

To ultimately maximize the expressive power of the network, more complex architectures should be explored. However, as reported in [6], complexity cannot be increased indefinitely: too complex architectures cannot be made compatible with the χ^2_ν for any choice of the weight clipping parameter. Moreover, employing an overly complicated neural network could be self-defeating for our data quality monitoring tasks, as it would increase the algorithm's run time even more. In addition to that, since we are dealing with mono-dimensional datasets, it is not strictly necessary to employ the most powerful tools at our disposal.

3.4 Algorithm performance on DQM tasks

With the tuned hyperparameters of the network, we are ready to test the algorithm on real data quality monitoring scenarios. Referring to Figure 3.1, we've already stated that RUN1252 (Figure 3.1a) is our reference run. Thus, we sampled a $\mathcal{N}_{\boldsymbol{\mathcal{R}}} = 200000$ reference dataset from the whole run. Then, instead of sub-sampling from the same run as we have done for the hyperparameters tuning session, we sampled a single dataset of $\mathcal{N}_{\boldsymbol{\mathcal{D}}} = 3000$ from a different run. It is key to stick with the same number of events in $\boldsymbol{\mathcal{R}}$ and $\boldsymbol{\mathcal{D}}$ chosen to tune the network, as the weight clipping parameter heavily depends on such conditions. Namely, if we want to change $\mathcal{N}_{\boldsymbol{\mathcal{R}}}$ and $\mathcal{N}_{\boldsymbol{\mathcal{D}}}$ at some point in the process, we would need to tune the network again using $\boldsymbol{\mathcal{R}}$ and $\boldsymbol{\mathcal{D}}_{\boldsymbol{\mathcal{R}}}$ datasets having the new $\mathcal{N}_{\boldsymbol{\mathcal{R}}}$ and $\mathcal{N}_{\boldsymbol{\mathcal{D}}}$ events.

Referring again to Figure 3.1, we sampled a $\mathcal{N}_{\boldsymbol{\mathcal{D}}} = 3000$ dataset from each of the three remaining distributions (namely RUN1253, RUN1265 and RUN1242). As mentioned in Section 3.2.1, those three runs fall into three different categories. Precisely, RUN1253 (Figure 3.1b) is a *good* run, almost identical to our reference run: we would expect that our algorithm do not detect any discrepancy between $\boldsymbol{\mathcal{D}}_{1253}$ and $\boldsymbol{\mathcal{R}}$. In other words, the observed test statistic t_{obs} resulting from the training process should fall somewhere near the median of the χ^2_{10} distribution. On the other hand, RUN1265 (Figure 3.1c) is a *discrepant* run and our expectations are that the algorithm would notice a sensible deviation in $\boldsymbol{\mathcal{D}}_{1265}$ with respect to $\boldsymbol{\mathcal{R}}$. Thus, the output t_{obs} should fall somewhere in the right-hand tail of the χ^2_{10} distribution. Finally, RUN1242 (Figure 3.1d) is a *bad* run and, if the network is correctly learning, the t_{obs} in this case should be larger than the previous one, as the discrepancy between $\boldsymbol{\mathcal{D}}_{1242}$ and $\boldsymbol{\mathcal{R}}$ is more pronounced. The observed test statistics t_{obs} for each of the three tested runs after the training process are shown in Figure 3.4.

Figure 3.4: Evolution of the t_{obs} during the training process for each of the three tested runs

Starting from below, we find that t^{1253}_{obs} falls right under the target χ^2_{10} distribution, close to its median $\tilde{t}$. This outcome indicates that the algorithm cannot detect any deviation from $\boldsymbol{\mathcal{R}}$ in $\boldsymbol{\mathcal{D}}_{1253}$ and treats the latter as a reference sub-sample following the null hypothesis. On the other hand, t^{1265}_{obs} and t^{1242}_{obs} are consistently above the target χ^2_{10} median $\tilde{t}$: the neural network thus found discrepant data in both $\boldsymbol{\mathcal{D}}_{1265}$ and $\boldsymbol{\mathcal{D}}_{1242}$. Furthermore, having t^{1242}_{obs} sensibly larger than t^{1265}_{obs} shows that the algorithm found $\boldsymbol{\mathcal{D}}_{1242}$ more discrepant than $\boldsymbol{\mathcal{D}}_{1265}$. Finally, note that a few thousands of training epochs are sufficient to reach a plateau and have a stable t_{obs} value. The exact agreement between our predictions and the actual algorithm performance is a great success. It means that the procedure is working as expected, and the algorithm behavior is under control.

3.4.1 Discrepancy assessment

We have already stated in Section 2.2 that we intended to quantify the probability that a discrepancy is present in $\boldsymbol{\mathcal{D}}$ by comparing t_{obs} with the $p(t\,|\,\boldsymbol{\mathcal{R}})$ distribution. In Equation 2.11 we have introduced the observed p-value and its corresponding significance $Z(p)$. However, t^{1265}_{obs} and t^{1242}_{obs} locate too far on the right-hand tail of the distribution and Equation 2.11 cannot return a non-zero p-value. Hence, we have tried to exploit the approximation

$$p_{\text{obs}} = \int_{t_{\text{obs}}}^{+\infty} p(t\,|\,\boldsymbol{\mathcal{R}})\,\mathrm{d}t \simeq \int_{t_{\text{obs}}}^{+\infty} \chi^2_{10}(t)\,\mathrm{d}t \tag{3.2}$$

with no improvement in the results. Until new ideas come to our minds, we are forced to stick to a rather qualitative discrepancy assessment based on the distance between the observed t_{obs} and the target χ^2_{10} median $\tilde{t}$. Namely, if t_{obs} turns out to be qualitatively near the median of the distribution, then we can consider the tested dataset $\boldsymbol{\mathcal{D}}$ as *non-discrepant*, while an observed test statistic located far from $\tilde{t}$ is an represents a *discrepant* dataset $\boldsymbol{\mathcal{D}}$.

3.5 Neural network's data reconstruction

Although not strictly necessary for our purposes, it is fascinating and helpful to check what the neural network has learned throughout the training process. It allows us to understand whether the implemented model is sufficiently accurate to grasp all the deviations from $\boldsymbol{\mathcal{R}}$ that are present in $\boldsymbol{\mathcal{D}}$, or it just gives an approximate answer to it. From Equation 2.10 we know that the trained NN has learned the maximum likelihood fit to the log-ratio between the expected density predicted by the best alternative model and the reference density. Thus, we can first inspect the predicted density ratio by plotting the exponential of the output $\exp(f(x;\,\widehat{\mathbf{w}}))$. We expect this ratio to be almost constant and

close to one when there are no discrepancies in $\boldsymbol{\mathcal{D}}$. On the other hand, when the ratio consistently departs from one, the network has learned to approximate a possible deviation from the reference model. Then, we can let the trained network predict the data distribution: by feeding the NN our reference distribution, the output of that prediction can be exploited to weigh it and reconstruct the data sample that has been used during the training process. The NN reconstruction should be similar to the dataset $\boldsymbol{\mathcal{D}}$ if the network has correctly learned to approximate the log-ratio. On the other hand, if the network could not detect any discrepancy between the two datasets, the reconstruction should look like $\boldsymbol{\mathcal{R}}$.

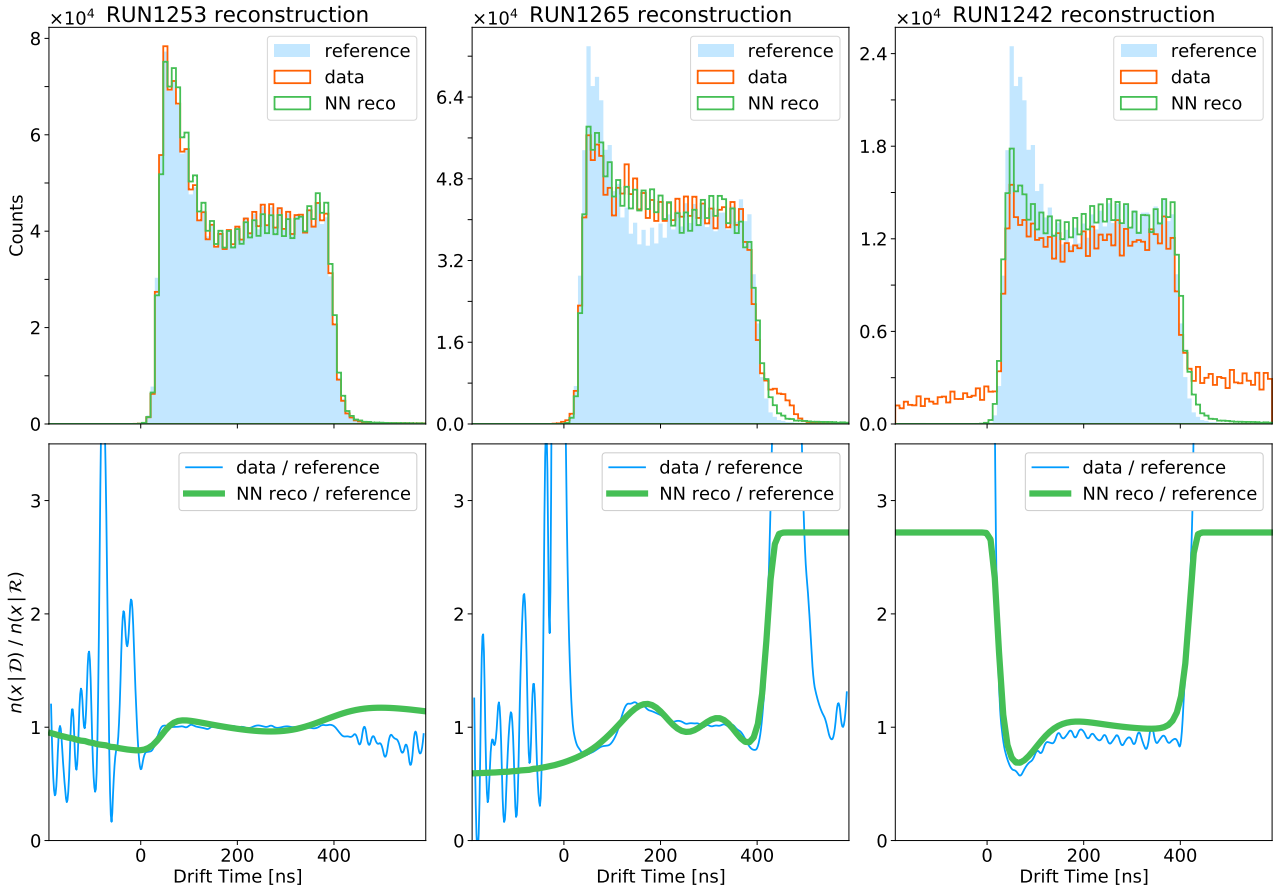


Figure 3.5: Neural network's data reconstruction for each of the three tested runs

As we can see in Figure 3.5, the neural network can learn the ratio between $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{R}}$ and grasp the overall trend of it, but cannot reconstruct the strong discrepancies between the two accurately. Precisely, in the RUN1253 reconstruction, the algorithm does not detect relevant anomalies in $\boldsymbol{\mathcal{D}}_{1253}$ (as we have already understood by studying the observed test statistic t_{obs}) and thus the $n(x \mid \widehat{\mathbf{w}}) \,/\, n(x \mid \boldsymbol{\mathcal{R}})$ stays close to one. It is consistent with the empirical ratio inside the time box's range. However, outside that range where the reference sample is almost not populated, the reconstruction is quite lacking, as it cannot detect the fast oscillations of the ratio. The same behavior is reproduced in the RUN1265 reconstruction: inside the $0 - 400\,\text{ns}$ range, the reconstructed ratio fits the empirical one accurately. Here, the network also detects the right-hand tail anomaly but does not have enough expressive power to reach the ratio's peak. Similarly, the RUN1242 reconstruction does not fit the noise excess accurately outside the time box. This analysis of the network's reconstruction shows us that the current implementation of the algorithm is not sufficient to understand the data we are dealing with entirely. Of course, the algorithm stems from the search for new physics, where the hoped-for signals have a low significance above the reference background. Thus, while sensitive to small departures from $\boldsymbol{\mathcal{R}}$, the algorithm fails to accurately understand substantial discrepancies if the NN's architecture is not complex enough. However, our implementation can fit the overall trend of the ratio of the empirical distribution and demonstrates that we are working in the right direction.

Conclusions and Future Outlook

We dedicate this last chapter to quickly run through again our data quality monitoring concept and goal, and assess what this thesis work had achieved and what is still missing out.

This thesis work has seen the implementation of a deep learning algorithm, usually employed to search for new physics at LHC, to perform data quality monitoring. The results are indeed auspicious. We have shown that the neural network successfully discriminates between good data and poor data. However, there are some issues we need to address. Fortunately, though, it seems we could have solutions for most of them. Further work will be carried out starting from this thesis, as it only proved that our initial ideas could be made concrete.

4.1 Automated monitoring

The first target of this work was to minimize the human efforts in data quality monitoring tasks. At this stage, a significant amount of human-performed work is necessary. However, this thesis laid the foundations for significant improvements.

Instead of having the scientist analyze the observed test statistic t_{obs} at the end of the monitoring process and evaluate the quality of data, we would have the algorithm do it automatically. Namely, the idea could be to set two threshold values: the first one dividing the *acceptable* t_{obs} from the *discrepant* t_{obs} while the second one dividing the latter and the *critical* t_{obs}. In other words, the two thresholds would define three regions in the space of the test statistic. Depending on which region the t_{obs} belongs to, the algorithm would act in different ways: from just raising a warning to taking concrete actions towards the experimental setup.

However, to make this automated monitoring reliable, we must set meaningful and quantitative threshold values. The initial idea was to put those thresholds to the observed p-value. As reported in Section 3.4.1, though, we cannot make this idea concrete as for now. The alternative could be to put thresholds on the distance of t_{obs} from the median $\tilde{t}$, though it does not seem satisfactory. Of even more importance, we would like to have threshold values that are as more universal as possible. Namely, we aim at having thresholds that do not depend (as far as possible) on the network's architecture and hyperparameters. It is a solid reason to consider median thresholds unsatisfactory, as they directly depend on the network's degrees of freedom. Moreover, through the numerous tests that we ran throughout this work, we noticed that the observed t_{obs} value is somehow unstable. In other words, if we feed to the network N data sub-samples (instead of just one), we would find a distribution of t_{obs} (the procedure is analogous to the tuning of the network). Its median could be used as an estimate of the overall observed test statistic and compared to the $p(t \,|\, \boldsymbol{\mathcal{R}})$ distribution. However, if the algorithm detects a discrepancy in the data samples, the t_{obs} distribution will not follow a χ^2_ν distribution. Instead, we found that the t_{obs} distribute themselves following an approximate uniform distribution with considerable variance. Thus, even using data from the same run, the results are very variable and make it difficult to choose appropriate threshold values.

A possible solution could be to have a more flexible neural network, and we thought of two possible implementations. The first one involves the employment of multiple reference samples. Namely, suppose the algorithm knows better what we label as *good* run (by seeing more of them). In that case, it could have an improved evaluation of the test statistic and thus a more stable and reliable output. Possibly, this improvement could also lead to a reduction of the observed t_{obs} values, making it possible to exploit the p-value discrepancy assessment. However, a more straightforward solution could be to

increase the network's complexity. This way, the algorithm would have an enhanced expressive power and could increase output reliability. Moreover, as shown in Section 3.5, a more complex architecture could increase the network's capability of fitting the $n(x \mid \widehat{\mathbf{w}}) \,/\, n(x \mid \boldsymbol{\mathcal{R}})$ ratio and thus returning a more aware test statistic t_{obs}.

4.2 Online monitoring

Even though increasing the network's complexity seems to solve most of the urging problems we have exposed, a substantial drawback cannot be overlooked.

At the beginning of this chapter, we have stated that our goal is to build an *online* data quality monitoring algorithm. By *online* we mean "while the detector is collecting data". The data quality monitoring procedure we have in mind is thus the following:

1. Build a large reference sample using an experimental setup known to give the expected results.
2. Optimize the network's hyperparameters by repeated training on reference sub-samples.
3. Start a new detector run, with possible unexplored configurations .
4. Begin a single training process as the detector successfully collect enough events to satisfy the $\mathcal{N}_{\boldsymbol{\mathcal{R}}}/\mathcal{N}_{\boldsymbol{\mathcal{D}}}$ ratio used during the network's tuning.
5. While training, the detector keeps acquiring data, populating a second data sample.
6. The algorithm finishes the training process *before* the second dataset has been completely built.
7. As the second dataset has been fully collected, the second training process can begin.
8. Repeat the procedure starting from 3.

The crucial point is expressed in step number 6: the algorithm training process must be shorter than the time to populate the data sample. Unfortunately, in the current implementation, the algorithm is already too slow to be put online. Precisely, to process a single dataset with $\mathcal{N}_{\boldsymbol{\mathcal{D}}} = 3000$ events against the reference of $\mathcal{N}_{\boldsymbol{\mathcal{R}}} = 200000$ it takes an average of 47 minutes, considering the minimum amount of training epochs that stabilizes the estimation of the observed t_{obs}.

Furthermore, using a more complex network architecture would slow down the training process even more, leading to a complete incompatibility between having a fully automated monitor and an online DQM procedure. 7

However, a way to achieve both goals may exist. In [7] we find an interesting alternative to neural networks: kernel methods. Kernel methods are a family of algorithms for pattern analysis and thus are employed to study general types of relations in datasets. Usually, these methods do not scale well with the dimensionality of datasets. Though, the Nyström approximation solves the problem, as reported in the cited article. The Falkon algorithm can thus be implemented to resemble the deep learning model we presented in this thesis and the related articles. The procedures introduced do not need to be changed significantly, though there are some differences. For example, using Falkon, the $p(t \mid \boldsymbol{\mathcal{R}})$ will still follow a χ^2_ν distribution. In this case, however, the degrees of freedom are not fixed by the number of trainable parameters like NNs.

Recent studies compare Falkon with our implementation of the algorithm: the overall performance on synthetic datasets are almost identical, and the χ^2_ν distribution followed by $p(t \mid \boldsymbol{\mathcal{R}})$ with Falkon always happens to have a number of degrees of freedom close to the fixed NNs one used for a certain kind of datasets. The significant advantage of Falkon's implementations is the run time performance: the same problem a NN can process in a few hours can be processed by Falkon in seconds.

Falkon capabilities seem to define the way to go. Future studies will then cover its implementation in data quality monitoring procedures following the steps defined by this pilot work. With Falkon, we ultimately aim at fulfilling our goals of having a fully automated online data quality monitoring procedure working on our detector. However, we still do not have enough information to tell whether Falkon will be enough, thus further tests need to be carried out following this direction.

Bibliography

[1] *Muon trigger with fast Neural Networks on FPGA, a demonstrator*
Matteo Migliorini et al. May 10, 2021
https://arxiv.org/abs/2105.04428

[2] *Offline Calibration Procedure of the Drift Tube Detectors*
Nicola Amapane et al. December 3, 2007
https://cds.cern.ch/record/1073687/files/NOTE2007_034.pdf

[3] *Calibration of the CMS Drift Tube Chambers and Measurement of the Drift Velocity with Cosmic Rays*
The CMS Collaboration. January 14, 2010
https://arxiv.org/abs/0911.4895

[4] *Learning New Physics from a Machine*
Raffaele Tito D'Agnolo, Andrea Wulzer. June 8, 2018
https://arxiv.org/abs/1806.02350

[5] *The large-sample distribution of the likelihood ratio for testing composite hypotheses*
S. S. Wilks. Ann. Math. Statist. 1938

[6] *Learning Multivariate New Physics*
Raffaele Tito D'Agnolo et al. January 27, 2021
https://arxiv.org/abs/1912.12155

[7] *Kernel methods through the roof: handling billions of points efficiently*
Giacomo Meanti et al. June 18, 2020
https://arxiv.org/abs/2006.10350

[8] *Neural Networks and Deep Learning*
http://neuralnetworksanddeeplearning.com