Università degli Studi di Milano

FACOLTÀ DI SCIENZE E TECNOLOGIE
Corso di Laurea Magistrale in Fisica

Master's Thesis

Investigating Anomaly Effects in High Energy Physics with Generative Models

Candidate:
Marco Rossi
Matricola 897863

Thesis Advisor:
S. Carrazza

Research Supervisors:
M. Pierini
A. Wulzer

Anno Accademico 2018-2019

Contents

Abstract

The thesis arises in the context of high energy physics (HEP), we aim to introduce tools based on machine learning techniques in order to address problems that are bothering the scientific community: in first place no anomalous signal of beyond standard model physics has been observed since Higgs boson's discovery in 2012, this led to the birth of new ideas regarding searching for new physics, however none of them has unfortunately scored a goal yet; second, the refining of experimental measures and theoretical computations requires a higher and higher need of large dataset from Monte-Carlo (MC) generators, that are incredibly time and computational resource consuming.

We address this issues with the help of a particular class of machine learning generative models, called generative adversarial networks (GANs). The two entities in adversarial training, namely the discriminator $\mathcal{D}$ and the generator $\mathcal{G}$, are respectively employed as tools for new physics model independent search and MC-like events' generator. The model independency allow us to look for signals in real-data experiments with no prior bias on the nature of new physics responsible for the discrepancies from the Standard Model. This can be achieved at the price of a little performances' loss compared to traditional model dependent classifiers. The GAN generator instead shortcuts the entire MC procedure in order to save memory storage and time in complicated simulations. We perform an assessment of $\mathcal{G}$ working out a feasibility study concerning the classification of anomalous signals within our toy dataset. We test the possibility to use the generative model to increase the classification rate.

Chapter 1

Introduction

Machine Learning (ML) had a great success in recent years in tackling a wide spectrum of problems. In fact, ML is specifically designed in such general terms, that in principle there are no limits to its employements. Besides this special flexibility, it had promising test performances, very often exceeding existing classical methods and workflows with little effort. This ideas strongly support the development of the present thesis work.

In particular, two issues in High Energy Physics (HEP) are suited to be addressed with a branch of ML algorithms called generative models. After the Higgs' discovery at LHC in 2012, the pattern of Standard Model's (SM) particles has been completed. Physicists know that SM has several structural problems and are tirelessly looking for beyond Standard Model (BSM) physics. Standard techniques in BSM research are founded on a statistical test approach, which needs the statement of a specifical alternative BSM hypothesis to be compared with the null SM one. Statistical tests are, by definition, model dependent new phyiscs searching instruments. The first goal of the present work concerns implementation of a model independent new physics tool. The idea is to train a neural net capable of signal data discrepancies from SM predictions, regardless of any particular BSM alternative scenario. We train a neural net according to the framework of a particular class of generative ML algorithms, called generative adversarial networks (GANs).

The particular design of the used ML architecture allow us to attack, at the same time, another severe problem that affects the HEP community, that is the continous and increasingly compelling need of computational resources, due to MC simulations. Improvements in theoretical and experimental physics require more and more precise simulations in order to predict the outcome of an experiment. We aim to prove the consistency of GANs as supports for MC algorithms. This new tools, then, can limit costs in terms of time, computational power and memory storage. In this sense, we assess improvements in orders of magnitude, providing a great aid to face in future years the impending big data problem of HEP research centres.

In the following section we discuss more in detail the physical issues that motivated the present work. In particular in 1.1 we describe the traditional framework of model dependent new physics search, that physicists usually employ. Appendix B reviews the terminology of statistical hypothesis testing. In section 1.2, instead, we present a complete description of the ML algorithms employed throughout the thesis.

1.1 Anomalies in High Energy Physics

Model dependent new physics search is the common approach in HEP. A statistical test often is performed in order to accept or reject a null hypothesis (SM) in favour of an alternative one

(BSM). An experiment-tailored test statistics can be employed to extract from data a p-value, a quantity that represents the probability, under the null hypothesis, to observe measurements more in tension with the SM, than the actual data. If the p-value is less than a pre-determined number called significance α, usually fixed at 5% or 1%, the statistical test tells to reject the null hypothesis. Assuming that one is testing the right alternative model, this approach is very effective in discovering signals. On the other hand, since the test is completely focusing on that particular alternative scenario, it would neglect the crowd of other possibilities missing the eventual anomaly present in the measurements.

So far at LHC, and any other experiments, none of this method's applications have worked. It is possible that a future BSM model is not among those usually tested. The problem could be also analyzed from the point of view of the big-data problem in HEP. As stated in [2], at LHC, for example, 40 million proton-beam collisions are produced every second, but only 1000 collision events/sec can be stored by the ATLAS and CMS experiments, due to limited bandwidth, processing and storage resources. It is possible to imagine BSM scenarios that would escape detection, simply because the corresponding new phyiscs events would be rejected by a typical set of online selection algorithms.

We employ generative models to work out a model independent new physics search. The ultimate goal is to deploy a tool that instantly processes a specific dataset and tell if it contains or not significant departures from SM predictions. GANs architectures are comprised of two neural networks that act as a generator and a discriminator. The trained generator is a source of new artificial data resembling the training ones, while the discriminator outputs a probability that answers the question: "is the input data real or artificial?". This discriminator's ability can be exploited to perform the mentioned anomaly detection task in a model independent way.

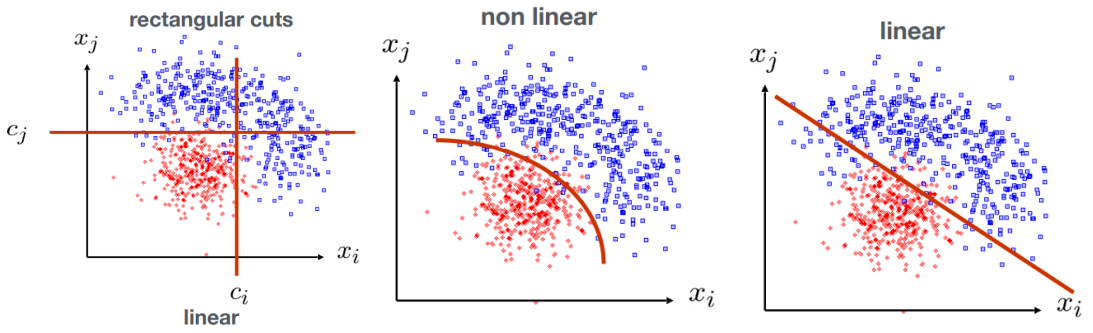
In the following section we give an insight of event classification algorithms in high energy particle physics. The topic is presented listing the various methods that are usually employed to discover anomalous signals from measurements. Their final aim is to define a critical region boundary in order to correctly label points in data space with some probability to be anomalous (BSM), or following the null reference model (SM).

1.1.1 Classification Algorithms in HEP

We present now methods for events classification that are part of a topic in statistical data analysis called multivariate analysis. We consider events that can be either signal (BSM) or background (SM). Usually they come from measurements in detectors' accelerators, represented by a d dimensional vector $\mathbf{x}$ of features. In order to classify an event, a mapping from feature space to a one dimensional variable $y(\mathbf{x})$ is defined. Then, we refer to imposing the inequality $y > c$, with $c \in \mathbb{R}$, as cutting data space. Indeed, what we are doing here is selecting an hyper-surface in feature space and defining a critical region of anomalous points as the set $\{\mathbf{x} : y(\mathbf{x}) > c\}$. The hyper-surface is an hyper-plane in case of linear cuts, but in literature other variants exist, as depicted in fig. 1.1. Once the critical region is determined, classifier's performances can be assessed with methods explained in section 1.2.3.

Traditional methods of events classification are founded on the likelihood ratio test statistics. The importance of this particular test statistics is remarked by the Neynman-Pearson (NP) lemma:

> Let $H_0 : \theta = \theta_0$ and $H_1 : \theta = \theta_1$ be two hypotheses of a test with test statistics given by the following quantity, called likelihood ratio:
>
> $$\Lambda(x) = \frac{\mathcal{L}(x|\theta_0)}{\mathcal{L}(x|\theta_1)}$$



(a) Rectangular cuts (b) Nonlinear cuts (c) Linear cuts

Figure 1.1: Different kind of cuts in feature space. Panels show a two dimensional problem in order to help visualization of cuts (red lines).

where $\mathcal{L}(\theta|x)$ is the likelihood function, and x is a one dimensional, for simplicity, experimental outcome. The NP lemma states that $\Lambda(x)$ is the most powerful test at a significance level α.

This statement implies the way of finding the best threshold η to define the optimal critical region in data space. Where η is a number that satisfies: $\alpha = \mathrm{P}(\Lambda(x) > \eta|H_0)$. Power in a statistical tests implies that the test is easily able to tell apart the two hypotheses' distribution functions. Therefore, more power, as this expression suggests, is linked to a more effective test.

The problem with this test statistics is of course that we do not usually know the exact conditional pdfs of data under a particular hypothesis. In special cases, theory might provide an approximate analytical form for pdfs, but in general the true underlying generating function is not known and its estimate must be provided to work out the test. Standard techniques to face this problem lead to a so called likelihood classifier. Examples of such algorithms are histograms, K-nearest neighbour (K-NN) or Kernel Density Estimation methods. This approach can be circumvented by other classification models, which determine decision boundaries directly from their output; neural networks (NN), rectangular cut optimization, linear discriminant analysis (Fisher and H-matrix discriminants), decision trees and support vector machines (SVM) are examples that take advantage of this alternative approach. In the following paraghraphs we introduce a brief description of some of these algorithms, regarding either pdf estimators and alternative classifier models.

Pdf's estimators for likelihood classifiers

Histograms method This is a naive option to solve the task of pdf's estimation. It requires simply to compute the histogram of the measurements. Data space is then divided in $\mathbf{M}$ $d-$dimensional cubic cells. The fraction of events that fall within each bin yields a direct estimate of the density at the value of the feature vector $\mathbf{x}$. The estimate, however, is binning dipendent, since small binning width can wash out all the signal, while large cell's volume leads to a too spiky density function. The deepest problem with this method is the so called "curse of dimensionality", that refers to the fact that the number of points required to work out this method increases exponentially with the dimensionality of feature space. This because also the number of cells to be filled with points grows as $\sim \mathbf{M}^d$.

K-NN method This method allow to approximate the pdf at a point $\mathbf{x}$ in data space. It consists in placing an hyper-cube of side h centered in $\mathbf{x}$ and counts the number of points that lie inside that portion of space. After finding k points divided in background k_b and signal k_s, the probability estimate is then given by:

$$\hat{p}(\mathbf{x}) = \frac{k_s(\mathbf{x})}{k_s(\mathbf{x}) + k_b(\mathbf{x})}$$

K-NN method can be improved properly changing the small volume shape: a sphere with radius R, accounting for the correlation between features, may be a better choice than a simple hyper-cube. Therefore we introduce the Mahalanobis distance:

$$R = \Big(\big(x^{(i)} - x^{(j)}\big)^{\intercal} \mathbf{V}^{-1} \big(x^{(i)} - x^{(j)}\big) \Big)^{1/2}$$

where $\mathbf{V}$ is the covariance matrix a $x^{(i)}$ is a vector of the $i-$th feature of all data. The K-NN classifier has best performance when the boundary that separates signal and background events has irregular features that can not be easily approximated by parametric learning methods. See fig. 1.2a.

Kernel Density Estimation method This algorithm outputs an estimate of the pdf of a sample $(\mathbf{x_1}, \ldots, \mathbf{x_n})$, based on the following equation:

$$\hat{p}(\mathbf{x}) = \frac{1}{nh} \sum_{i=1}^{n} \mathrm{K}\Big(\frac{\mathbf{x} - \mathbf{x_i}}{h} \Big)$$

where h is a smoothing parameter called bandwidth and K is the kernel, a positive function that is usually taken to be as a multivariate gaussian:

$$\mathrm{K}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}} \exp\Big(-\frac{|\mathbf{x}|^2}{2} \Big)$$

The kernel K apply a smearing function over data and every contribution from the sample set is summed to give the estimate. Bandwidth controls the overlapping of kernels in data space: in the gaussian kernel example greater values of h implies more spreaded function added to the final sum, instead when h goes to zero then each function approximate a delta function peaked on the corresponent measure $\mathbf{x_i}$. See fig. 1.2b.

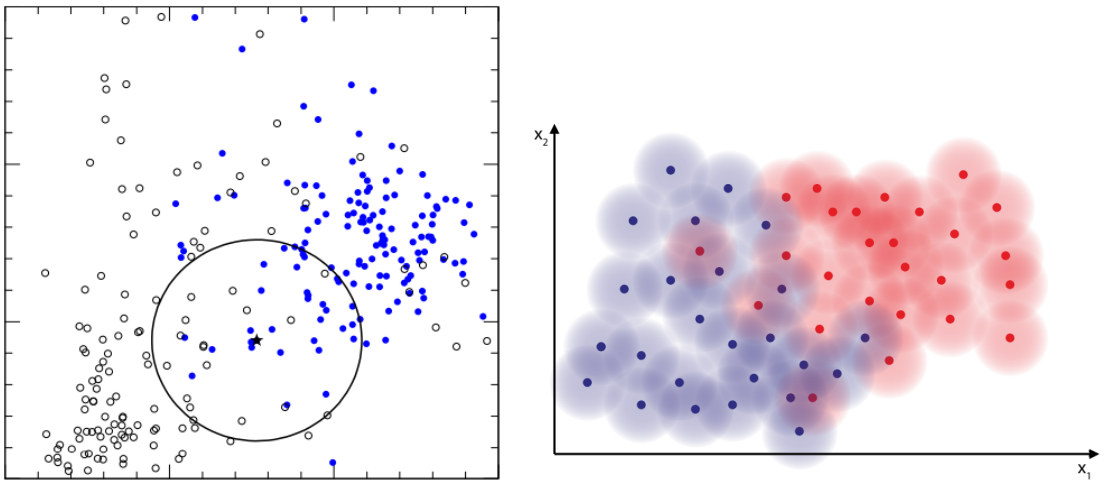
Alternative classifiers

Linear discriminant analysis Fisher's linear discriminant is a simple algorithm, nevertheless it can still be optimal if the amount of training data n is limited. It consists in making an ansatz for the test statistics:

$$y(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i$$

where $\{w_i\}$ is a set of weights. This parameters are chosen in order to maximize signal and background separation given, in Fisher's method, by:

$$J(\mathbf{w}) = \frac{(\tau_s - \tau_b)^2}{\Sigma_s^2 + \Sigma_b^2}$$



(a) K-NN method, a hyper-sphere of radius R includes a fraction of measurements.

(b) K-DE method, a smearing function is computed at each point.

Figure 1.2: Visual examples of some pdf's estimation algorithms.

where τ and Σ are respectively mean and covariance matrix of $y(\mathbf{x})$ for signal or background distributions. In formulas [1]

$$\tau_{s,b} = \int y(\mathbf{x})\, p(\mathbf{x}|H_{1,0})\, d\mathbf{x} \equiv \mathbf{w}^\intercal\, \mu_{s,b}$$

$$\Sigma_{s,b} = \int (y(\mathbf{x}) - \tau_{s,b})^2\, p(\mathbf{x}|H_{1,0})\, d\mathbf{x} \equiv \mathbf{w}^\intercal\, \mathbf{V}_{s,b}\mathbf{w}$$

Then the numerator and denominator of $J(\mathbf{w})$ become:

$$(\tau_s - \tau_b)^2 = \sum_{i,j=1}^{n} w_i w_j (\mu_i^s - \mu_i^b)(\mu_j^s - \mu_j^b)$$

$$\Sigma_s^2 + \Sigma_b^2 = \sum_{i,j=1}^{n} w_i w_j (\mathbf{V}^s + \mathbf{V}^b)_{ij}$$

Intuitively, this algorithm tells the model to maximes the ratio of the separation between classes (means) and the separation within classes (covariances).

Decision Trees This model is a directed graph. It develops from a starting node called root node through the branches of the tree (hidden nodes) to final leaves nodes. At every node, the algorithm implements a cut in feature space. Compositions of cuts allow the classification of data, which is made at the level of a leaf node. During training the tree keeps growing until all the nodes are pure, which means that the quantity called purity p reaches either the maximum or the minimum for every node. In this algorithm, a weight w_i is assigned to each sample in the training data. Purity is defined as:

$$p = \frac{\sum_s w_i}{\sum_s w_i + \sum_b w_i}$$

[1] $\mu_i^{s,b} = \int x_i\, p(\mathbf{x}|H_{1,0})\, d\mathbf{x}$ is a vector of means and $\mathbf{V}_{ij}^{s,b} = \int (x_i - \mu_i^{s,b})(x_j - \mu_j^{s,b})\, p(\mathbf{x}|H_{1,0})\, d\mathbf{x}$ is a covariance matrix on data.

And the improvements in signal backgroud separation after splitting a set A into two subsets B and C are measured with the quantity:

$$\Delta = W_A G_A - W_B G_B - W_C G_C$$

where G is Gini index $G = p(1-p)$ and $W_X = \sum_X w_i$. The situation in which leaf nodes are all pure must be avoided, since the model is clearly overfitting data. Then a pruning operation is carried out removing some internal nodes, according to the fact that a simpler model is less prone to overfitting than a more complex one.

Since decision trees are really sensitive to statistical fluctuations in training sample, a boosted version of the algorithm exists. Boosting consists is a general methods consisting in combining together different classifiers. In the case of decision trees, we switch from a single tree graph to a multitude of trees forming a forest (random forest algorithm). Trees are derived from the same training set, re-weighting events. Individual trees are then combined with a weighted average of individual models to form a powerful classifiers with smaller error.

Support Vector Machines (SVM) This algorithm constructs an hyper-plane decision boundary in feature space in order to maximize the distance between points from different classes in a binary problem. A set of measurements has to be classified according to their label, which can be either 1 or -1. The training set is a collection of n couples $\{(\mathbf{x}_i, y_i) : i \in [1, n]\}$, where $\mathbf{x}_i$ is a vector of features and y_i is a label. The problem can be stated initially for linearly separable datasets, where exists a set of hyperplanes described by the equation $\mathbf{w} \cdot \mathbf{x} - b = 0$ and parametrized by a normal vector $\mathbf{w}$ and an offset parameter b, these hyper-surfaces split data space into two regions, each containing points just from a particular class. Fig. 1.3a reflects this situation, where the red line represents the decision hyper-plane. The coloured stripe is called margin and geometrical considerations imply that it has width $\frac{2}{\|\mathbf{w}\|}$. Then, maximizing the margin is equivalent of minimizing the norm of the vector $\mathbf{w}$ orthogonal to the hyperplane. After minimization, a couple $(\mathbf{w}, b)$ defines a classifier that implements the mapping from data space to label space: $\mathbf{x} \longmapsto \text{sgn}(\mathbf{w} \cdot \mathbf{x} - b)$.

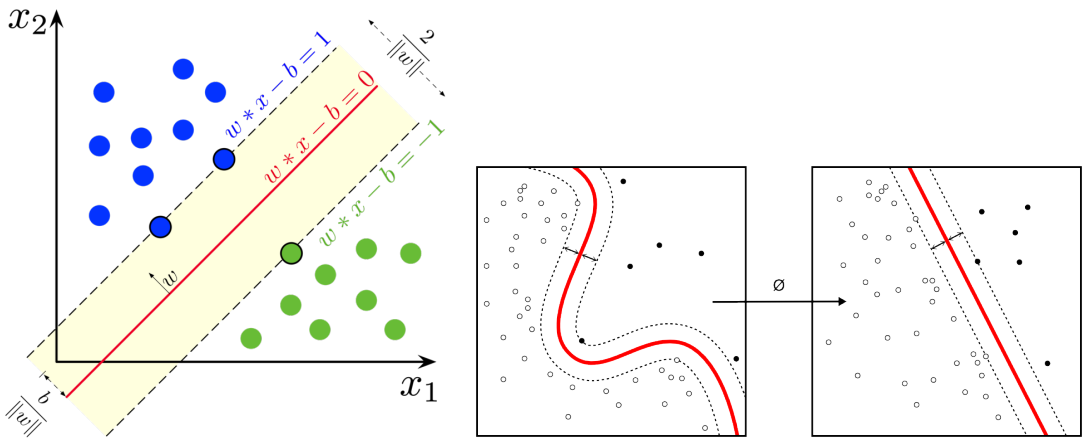
In a non-separable problem instead, the function to be minimized has to be defined more carefully. It can be proven that the inequality $1 - y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \leq 0$ for all possibile values of i is satisfied only if the correspondent point lies in the correct side of data space with respect to the boundary line. Then the so called hinge function $\max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i - b))$ takes 0 values, if the related point is correctly classified, it takes a value proportional to the distance between the point and the hyperplane otherwise. Finally, the function to be minimized has two terms:

$$\frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i - b)) + \lambda \|\mathbf{w}\|^2$$

where λ is a parameter that determines the trade-off between reducing the sums of distances between mis-classified points from the decision hyper-plane and the reciprocal of the width of the margin. This equation can be generalized to include non-linear decision boundaries, replacing the dot product $\mathbf{w} \cdot \mathbf{x}_i$ with a kernel function $k(x_i, x_j)$. This kernel trick can be viewed as a mapping Φ from feature space, in which the kernel boundary has a non-linear shape, to a transformed space in which the boundary is in fact linear. This behaviour is depicted in fig. 1.3b.

1.2 Machine Learning

In this section we present the basics principles of machine learning (ML). We firstly define what a learning algorithm is and which kind of tasks ML tries to solve, following the exhaustive exposure



(a) SVM feature space

(b) SVM with kernel trick

Figure 1.3: SVM

of reference [3, ch. 5]. Then we introduce the fundamental building blocks of neural networks, namely artificial neurons, and how they are organized in feed-forward (FF) layers to form a first simple example of neural network (NN). We also review a particular variant of FF layers, widely employed throughout this work: the convolutional layer. In the remaining part of this section we focus on how these layers can be stacked to shape models that can be trained to perform desired tasks. We discuss two kinds of models: networks performing classification tasks, also called classifiers, with the theoretical formalism behind them and a particular class of generative models called generative adversarial networks (GANs), with a review of their architectures and charateristics.

1.2.1 Learning Algorithms

A ML algorithm is a model that is able to learn from data. In 1997 Mitchell, in [12, p. 2], provided a definition of what such a learning algorithm is.

> **Definition.** A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

In the following subsections we briefly give intuitive descriptions and examples of the three quite abstract entities T, P and E.

The Task, T

In the definition given above, the task is the goal of the process of learning, so learning itself is not the task, it is the way of achieving the ability to do the task. Tasks in ML are usually described in terms of the way the model is able to process examples. An example is a collection of n quantitative features, measured from some object, that we want the model to inspect. This way we can represent an example as a vector $\mathbf{x} \in \mathbb{R}^n$. For instance we treat a squared balck and white image as a vector in $\mathbb{R}^{n \times n}$, where each component takes an integer value in the interval $[0, 255]$, representing the corresponding pixel's brightness.

In the following, we report a summary of the most common machine learning tasks.

- **Classification**: in this type of task, the program is asked to specify which of k categories some input belongs to. Hence, the model will attempt to learn a function $f : \mathbb{R}^n \longrightarrow \{1 \dots k\}$, which maps an example $\mathbf{x}$ to its category $y = f(\mathbf{x})$. We will deeply employ classification networks in section 2.
- **Regression**: in this type of task, the model is asked to predict a numerical value given some input, i.e. to learn a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ as in classification, with the only exception of a continous codomain. A classical example of a regression task is to predict the amount of money an insured person will be expected to claim.
- **Machine Translation**: in this type of task, the network takes as input a sequence of symbols in some language and aims to convert them from that native language to another. ML algorithms are employed also in natural languages translations, like English-Italian translation.
- **Density estimation**: in this type of estimation problem, the model is trained to output a function $p_{model} : \mathbb{R}^n \longrightarrow \mathbb{R}$, where $p_{model}(\mathbf{x})$ can be interpreted as a probability density function, on the space where the examples were drawn from.

The Performance, P

In order to assess the algorithm's ability to accomplish tasks, we have to define a measure that quantifies its performances. P has to be designed specifically for each task we would like the model to be good at. For classification and transcription we can keep track of the accuracy (*acc.*), namely the ratio between the number of examples where the model makes a correct prediction over the total. Alternatively we can measure the error rate (called also 0-1 loss), which is equal to $1 - acc.$. Of course, it does not make sense to compute these metrics when solving other kinds of tasks, like density estimation. In these cases we have to build a task-specific loss function that reflects the features we would like the model to learn and of course also penalizes system's undesired behaviours.

A central concept in ML is generalization: we do not want the algorithm to fit the distribution of the input data, instead we would like it to learn features underlying data, in order to have good predictions on examples it has never seen before. Therefore is common use to have two distinct collections of examples (datasets): the training set and the test set. The former is used to train the model, while the latter is employed to assess its performances.

The Experience, E

ML algorithms, depending on what kind of experience they are allowed to have during the learning process, can be separated in two big categories: supervised and unsupervised.

- Unsupervised learning algorithms experience a dataset containing many features. It is model's duty to capture these features, learning the true probability distribution function $p(\mathbf{x})$ underlying the examples.
- Supervised learning algorithms also experience a dataset with several features, but each example now comes with a target or label. For instance the MNIST database is a collection of $60,000$ black and white 28×28 pixelled images of handwritten digits, plus a vector of lables that identifies each image's correct category. In this training mode a model tries to predict the correct label y from an example $\mathbf{x}$, or, in other words, it tries to reproduce the conditional probability density function $p(y|\mathbf{x})$.

The term supervised arise from the fact that the model is teached by the labels what to do, while in unsupervised learning the database completely lacks this information. It is worth noting that this two categories are not formally defined and always well separated as there are models that can be used to accomplish both tasks. We also would like to mention that other variants of the learning paradigm exist, such as semi-supervised learning, reinforcement learning or multi-instance learning.

Generalization

As we stated above, the central challenge in ML is to build models which perform well on new unseen data. This idea tells us the difference between an optimization problem and a ML algorithm: the former is a process in which we seek the model's best configuration in a parameter space in order to reduce the training error, while in the latter we want the generalization error (also called test error) to be small. The generalization error is the espected value of the error on a new unseen input. We evaluate it over the collection of examples called test set as stated in the section regarding the performance P.

In the most general situation, a priori, we can say nothing on the test error, knowing just the training error, since train and test sets are two well separated collections. In order to be able to affect the generalization error with training, we make two claims on how the datasets are collected: first, all the examples in the datasets are indipendent from each other (indipendence) and second, they are drawn from the same probability distribution p_{data}, called data generating distribution (identically distributed samples). These two hypotheses ensure that the training error will set an upper limit for the generalization error. We can now influence the final perfomance of a ML algorithm, monitoring and reducing two fundamental quantities: the training error itself and its gap from the test error.

When we do not manage to make small one of these quantities, we face two classical undesired behaviours of an algorithm, called respectively underfitting and overfitting. Underfitting means that we did not optimized enough the model's parameters, so we must run further training steps in order to lower the fundamental quantities. Overfitting means that the algorithm is not actually learning the true data generating distribution, but it does no more than fit the training samples points with a complicated function. Borderline cases are displayed in Fig. 1.4.

To better explain the concept, we can intuitively think about a student. Of course underfitting corresponds to the situation in which the student learnt the topic just superficially. The student, when is able to generalize, has mastered the subject and can ingeniously apply what he has learnt, is the best; whereas when he just parrot back the lesson, he is simply overfitting the issue.

Within the ML field, we protect ourselves from these two bad behaviours, by stopping the optimization algorithm at the right time. The question is of course, when the right time is. We address this problem with a technique called early stopping. It requires a method which assesses the relationship between training error and generalization error. The method we employed in this work is cross validation: from the training set we hold a 15% of examples that will not be employed to train the model. This samples are used during training to have an online insight of the generalization error over the test set. We call this collection of retained data validation set. In Fig. 1.5 we show the typical trend of error functions during training: according to early stopping, we should stop the optimization algorithm when the gap between training and validation error starts increasing in order to trade off between underfitting and overfitting regimes.

In the present section we have described what a ML algorithm is, following Mitchell's definition. We have called it interchangeably with the term ML model, to stress that it can learn from experience by processing vectorial examples. Learning consists, indeed, in modelling the

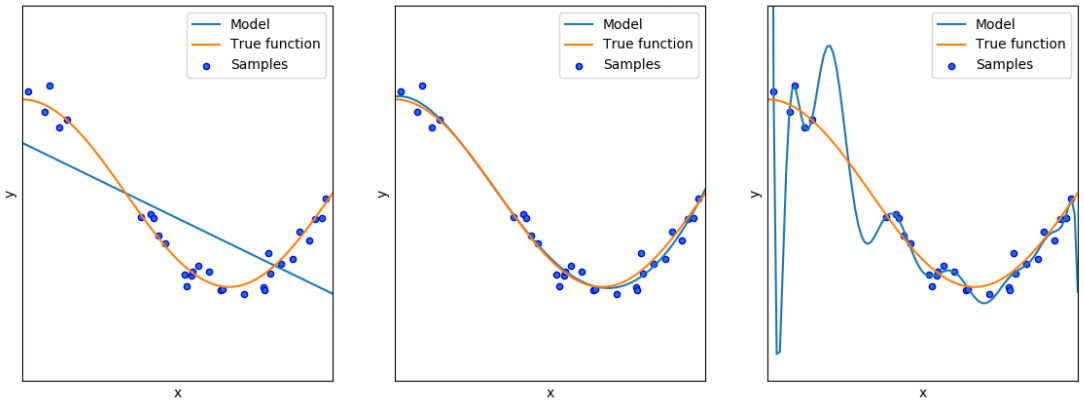


Figure 1.4: Three limit cases. *Left panel*: the model has a linear output, it has not captured the curvature present in the data; training errors are high: underfitting. *Central panel*: the model fits well all the data, with a little training error at each point: good generalization. *Right panel*: the model overfits the samples with a complicated function; training errors go to zero: overfitting.

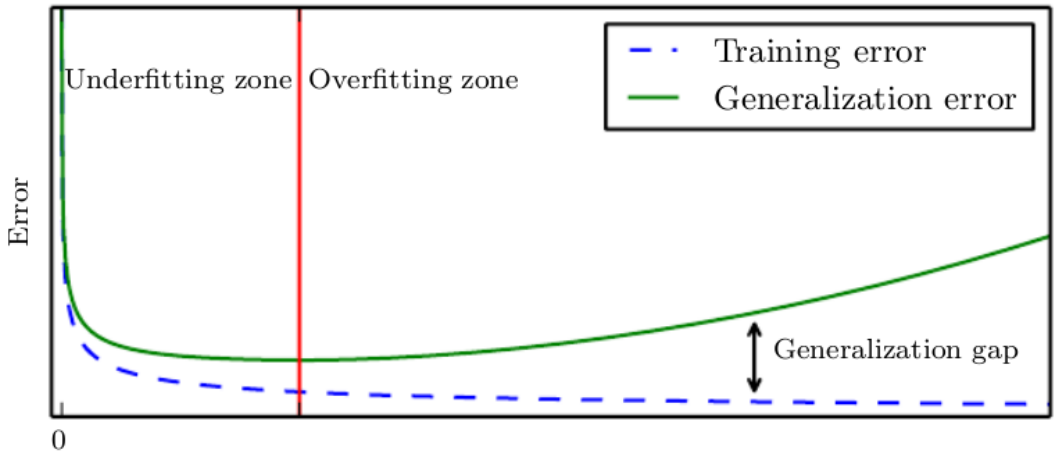


Figure 1.5: Typical qualitative trends of training and generalization (over the validation set) errors during training. The red line signals the best moment to stop the optimization algorithm, namely, as soon as the generalization gap starts increasing.

Table 1.1: Summary of the activation functions.

Name	Equation	Range
Linear	$\varphi(x) = x$	$(-\infty, \infty)$
ReLU, Rectified Linear Unit	$\varphi(x) = \max(0; x)$	$[0, \infty)$
LeakyReLU ($\alpha > 0$)	$\varphi(x) = \begin{cases} \alpha x & \text{is } x < 0 \\ x & \text{if } x \geqslant 0, \end{cases}$	$(-\infty, \infty)$
ELU ($\alpha > 0$), Exponential Linear Unit	$\varphi(x) = \begin{cases} \alpha(e^x - 1) & \text{is } x \leqslant 0 \\ x & \text{if } x > 0, \end{cases}$	$(-\alpha, \infty)$
Logistic (a.k.a. sigmoid)	$\varphi(x) = \sigma(x) = \dfrac{1}{1 + e^{-x}}$	$(0, 1)$
Tanh	$\varphi(x) = \tanh(x)$	$(-1, 1)$
Softmax†	$\varphi_k(x_k) = \dfrac{e^{x_k}}{\sum_{k=0}^{m-1} e^{x_k}}$	$(0; 1)$

† The softmax activation gives a different output for each neuron in a layer, it is used in multiclass classification to give results normalized across the whole layer, when classes are exhaustive and mutually exclusive.

algorithm's output on a distribution function underlying data, called data generating distribution, that in principle can implicitly capture samples' intrinsic high level features. Since the distribution p_{data} may be quite complicated, we have to be sure that our model will be capable of correctly reproduce it. This motivations, supported by the universal approximation theorem, as explained in [7], lead us to introduce neural networks (NN) as function approximants that can definitely achieve the aims of a ML algorithm.

1.2.2 Neural Networks

In this section we explain what a NN is: starting from basics we introduce artificial neurons, called this way because their behaviour recall that of a biological neuron. Neurons can then be arranged together to form layers, which are the building blocks of networks, that can accomplish an incredible variety of tasks if properly designed and tuned. We overview the different types of layers which are employed in this thesis' work and explain how is it possible to fix a very large number of parameters in order to allow the process of learning. With this in mind we hint at the main ideas behind the most common optimization and hyperparameter search algorithms.

Neurons and Layers

An artificial neuron is defined as a function $f_{\mathbf{w},b} : \mathbb{R}^n \longrightarrow \mathbb{R}$ that maps a collection of m input signals $\mathbf{x} = \{x_0, \ldots, x_{m-1}\}$ to an output:

$$y = f_{\mathbf{w},b}(\mathbf{x}) = \varphi_{\mathbf{w},b}(w_j x_j + b) \tag{1.3}$$

where $\mathbf{w}$ is a vector of m weights and b is a real coefficient called bias; in general we refer to these quantities as model's parameters θ. It is interesting the role of the so called activation function φ, because it introduces the opportunity to have non-linearities in an otherwise affine function. Tab. 1.1 lists common activations used in ML models.

We can group a set of n neurons together to form a dense layer. Hence, the vector of w_j weights and the bias b become a $n \times m$ matrix w_{kj} and an n-dimensional vector b_k respectively. The layer has now n outputs:

$$y_k = f_k(\mathbf{x}) = \varphi(w_{kj}x_j + b_k) \tag{1.4}$$

where we employed Einstein's convention of implicitly summing over repeated indices. Fig 1.6 displays pictorially the k-th neuron inside a layer.

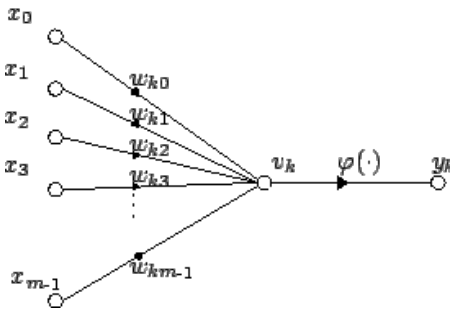


Figure 1.6: Artificial neuron

We now look to a different class of layers: convolutional layers. We employ convolutional layers to process examples that have a grid-like shape. In fact, these layers are just a slight modification of the dense type: the difference lies in the fact that they employ convolution in place of matrix multiplication between inputs and weights.

Convolutional Layers

Convolution is a mathematical operation on two functions of a real valued argument.

Definition. Let $f, g : \mathbb{R} \longrightarrow \mathbb{R}$ be two real valued functions. Their convolution is the function $(f * g) : \mathbb{R} \longrightarrow \mathbb{R}$, defined by the mapping: $t \longmapsto \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau$.

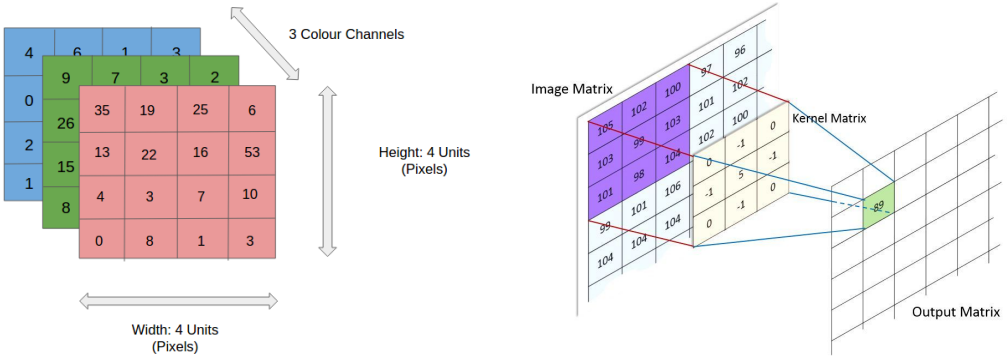
In general, we can look at convolution, as an operator that applies a filtering function g, called kernel in ML applications, on an input function f: the output is sometimes named feature map. Furthermore, if g is also a probability distribution function (pdf), the output $f * g$ is an average of f, weighted with the pdf g.

In order to apply convolution within a ML framework, we introduce a discretized form of this operation: real functions' arguments become integer indeces. Convolutional layers are usually employed, with great success, to process images, which can be seen as a grid of pixels, as in fig. 1.7a, and described with rank 3 tensors: the first two indeces of the tensors tell the row and the column in the grid, while the third one is for pixel's channels, that mixes together primary colours. One-channeled images are black and white, otherwise descriptions with three or four channels are suited for coloured ones.

The convolutional kernel K contains all the architecture's information and it is represented by a tensor with the following indeces' structure: its first two labels refer to the size of the filtering window (row and columns), the third one runs from 1 to the number of channels in the input image c_{in}, while the last correspondes to the number of channels in the output image c_{out}. Thus we can visualize K, as a set of c_{out} three dimensional tensors with the same volume: tensors' width and height correspond to the number of rows n_r and columns n_c of the filtering windows, that are stacked one on top of the other along the third dimension, which has size c_{in}.

Convolution is then the operation in which we apply these multidimensional filters to different subsets, with shape equal to the filtering window, of consecutive pixels in the input image. Mathematically we write:

$$O_{i,j,k} = \sum_{l=0}^{n_r-1} \sum_{m=0}^{n_c-1} \sum_{n=1}^{c_{in}} [I_{i\times r+l, j\times s+m, n}\, K_{l,m,n,k}] \tag{1.5}$$



(a) Image as grid (b) Convolution

Figure 1.7: Convolution

where we introduced the possibility to have strides r and s. Fig. 1.7b visually shows the convolution operation with single-channeled input, kernel and outputs.

The stride parameters tell the model not to inspect each consecutive subset of pixels: the convolution in this case skips respectively r and s image's cells in each direction before taking again the convolution operation. The stride option affects the information overlap between near pixels in the output image: having minimum strides, equal to one in each direction, ensures that the maximum amount of information is retained within the output image. Nonetheless, it is obviously computationally expensive and sometimes, in fact, does not improve performances of the model. Therefore a specific choice of these parameters must be selected accordingly to the current dataset.

Looking at equation 1.5, it is clear that there is an issue when the kernel deals with image's cells next to the boundaries: there, the sum's indeces would go out of range for the input image. One way out consists in carrying on the convolution operation, only until K lies entirely inside the image. This option is called "valid" convolution by ML libraries. Of course the output image will be shrinked in comparison to the input one. The opposite behaviour is the "same" convolution, in which the layer implicitly zero-pads the image to have input and output images of the same shape. The "same" option has the drawback that pixels near the borders of the input image influece a smaller amount of cells in the output, than the ones in the middle. This observation motivates the introduction of a third possibility, called "full", in which each pixel in the input image is allowed to be inspected by the kernel for the same number of times. In this situation, the problem is reversed, since the output pixels at the edges are influenced by a smaller number of input cells, than the ones in the middle. Optimality for a model in zero-padding, of course, is not an absolute fact, but strongly depends on the current dataset: it lies, in general, somewhere between the "valid" and the "same" convolution.

There are three key ideas behind the introduction of convolutional layers in neural networks: sparse interactions, parameter sharing and equivariant representations. In the following we give a brief introduction to these concepts, in order to motivate the intense usage of convolutional layers througout the present work.

Fig 1.6 shows the connection of a neuron in a dense layer to the input; in particular, we highlight that each neuron inside a layer is linked with each component of the input vector. Because of this aspect, dense layers are also called fully connected layers. This way, the number of weights used by a single layer equals the product of incoming input vector's components, times

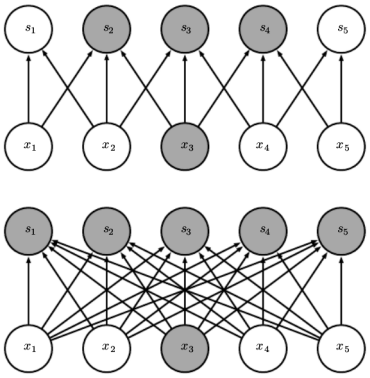


Figure 1.8: Sparse interactions vs full connectivity. *Top*: convolutional layer with kernel size equal to three, stride one and same padding. Grey shaded circles highlight which output's components are directly influenced by the central input's component: only next to units are affected. *Bottom*: fully connected layer. Each neuron in the output is linked to every neuron in the input. Even with this simple model we have a large number of edges in the graph.

the outcoming vector's ones plus one. It is clear from these considerations, that if samples in the dataset are images containing a lot of pixels (sometimes in the orders of thousands or even millions), it will be very computationally expensive to store the whole amount of weights and also to work out the matrix multiplication defining the dense layer operation.

The introduction of convolutional layers allow us to have a smaller number of weights, depending only on the size of the kernel K, which has often a far smaller size than the input image. We refer to this property of a convolutional layer saying that it presents sparse interactions or sparse connectivity. Moreover, in image analysis we are often interested in looking for patterns arising within a small portion of the image: with fully connected layers, pixels at one end of the image are linked also with cells from the opposite end by weights that could simply be unnecessary. Sparse connectivity is, then, the natural way to address these issues. Fig. 1.8 graphically reviews the ideas behind sparse interactions.

We previously mentioned that storing the enormous number of weights of dense layers may become expensive in terms of memory usage. Then, convolutional layers provide a simple answer to this problem, called parameter sharing. When we train a fully connected layer, the model has to learn the correct weight for each link in the graph. In convolutional layers, instead, the model has to learn a small set of kernel weights and then re-apply them (this consists in sharing parameters) to inspect each portion of the image, resulting in a dramatic reduction of the total amount of memory needed to store the model.

Due to this form of parameter sharing, the model inherits the well-desired property called equivariance to translations. In particular, we say that a function f is equivariant to a function g if $f(g(x)) = g(f(x))$. In the present case, convolution is equivariant to translations means that the application order of the two processes on the input image does not matter: if we slightly move the input image and then compute the convolution, the result will be the same as if we made the convolution and then shifted the output. We can also say that a convolutional layer looks for certain features in the input, no matter where they are: for example, if we applied convolution to an image of a cross section of any physical process, looking for spikes in the plot, the output map would find those features no matter where they are shifted from a certain point, or even no matter how many they are.

Convolutional layers represent very useful and efficient tools to analize images, but they

usually come with another operation that modifies further their output's values. This operation is called pooling. Different types of pooling layers exist, but they all exploit the same idea: they replace the value of each output's unit of a convolutional layer by a statistical metric that summarizes the nearby outputs. Of course, different metrics are possible: the most used are max pooling, average pooling, weighted average pooling and L^2 norm pooling. Pooling is useful because it makes the output invariant under small translations of the input. Indeed, if we apply max pooling over a small area, results would be the same if we shifted by a small amount the input image before, because the maximum would be obtained inside the same small area.

Optimization

In the previous sections we discussed dense and convolutional layers, which are the building blocks of the neural networks employed in the present thesis' work. We can build a feed forward neural network (FFNN) by stacking different layers. The simplest model is made of three such units: the first one is the input layer; the one in the middle is called hidden, because we do not usually look at its outputs, as we are concerned the most by the outputs of the final layer, which is consistently called output layer. This kind of network is named feed forward because information flows straightforwardly from the input to the output: there are no loops in the net. If we included the possibility to have feedback connections, in which the output of a layer can be fed back into itself, we would have built another kind of network: a recurrent neural network (RNN).

In FFNN, we can arbitrarily increase the size (or width) of the hidden layer, considering more neurons, as well as the network's depth, adding more hidden layers. With these operations we influence the number of parameters θ in the model and its complexity. A NN can be viewed as a family of functions $\{f(\mathbf{x})\}_\theta$ as θ varies in a very high-dimensional parameter space. Theoretically, as stated by the universal approximation theorem, a NN can approximate any continuous function on a compact subset of $\mathbb{R}^n$ with a particular choice of θ. Flexibility, then, makes NNs fundamental tools in ML algorithms, where we try to guess the data generating distribution. The challenge consists, of course, in finding the best point in parameter space with an efficient training algorithm. This step is called optimization.

In this section we explain the main ideas behind optimization algorithms found in ML literature: we take a look at gradient descent, while we address in the appendix A the more refined RMSProp and the adaptive moment estimation algorithm (Adam). We recall from a previous section, 1.2.1, that model's improvements in training are assessed by computing the value of a performance function, often called cost or loss function $\mathcal{L}$, associated to the model. This statement let us recast the learning problem in an optimization fashion: we would like to find the point in the multi-dimensional parameter space that corresponds to the minumum of the cost function.

For FFNNs, the optimization is accomplished in different steps:

1. feed the model with a batch of examples from the dataset and obtain the correspondent outputs (feed-forwarding);

2. from the outputs employ an automatic differentiation algorithm, always implemented by ML libraries, to find the gradient of the loss function w.r.t. the model's parameters, or related quantities (backpropagation step);

3. update the weights accordingly to a particular updating rule (depending on the optimization algorithm chosen).

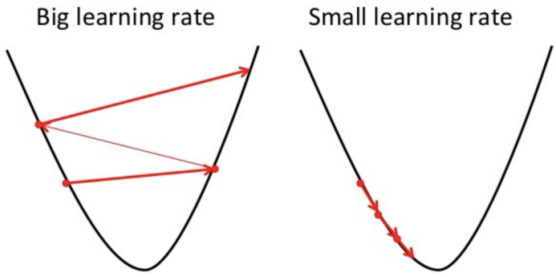


Figure 1.9: Different learning rate behaviours: one dimensional problem with quadratic loss function. This example is meaningful since every function can be approximated by a quadratic polynomial if we sit sufficiently close to a minimum.

We say that we conclude an epoch of training when all the training-set data have been processed by this algorithm. The whole training consists in iterating again and again this procedure for a fixed number of epochs until the cost function is appropriately minimized. We refer to this procedure as descending the gradient.

We note that the first step can be achieved in different ways: the naive approach is to compute the loss function processing together all the examples in the dataset and then computing the gradient with backpropagation. This way we update the parameters' values just once per epoch: the gradient, then, will contain the maximum amount of information. However this can lead to a dramatic slowdown of learning, especially when we deal with large datasets. In order to make the training faster, we can have an insight of the gradient evaluating it over just a small subset of samples, called mini-batch, and then updating the weights. Now we can update parameters multiple times in the same epoch.

Mini-batches' size must be chosen carefully in order to contain the sufficient amount of information in the gradients. The limit case is called online learning, in which mini-batches' size is equal to one. Although we can make a lot of updates in a single epoch with this trick, the algorithm will probably be unstable because a single example may not be statistically significant and consecutive gradients are likely to cancel each other, rather then lead us down the hill towards the global minimum of the cost function. Furthermore, we have to separate all the dataset in mini-batches and be sure the model will experience all the information in the dataset, in order to finish one epoch. Because of the randomness with which we select all the dataset's partitions, this algorithm is called stocastic gradient descent (SGD).

Steps 2 and 3 of the algorithm are guided by the definition of the gradient of a function itself: in particular, the gradient of the cost function w.r.t. the parameters of the NN is, by definition, a vector, that at each point of the parameter space, has the direction of the greatest increase of the cost function. Therefore, in principle, at each step of our optimization algorithm, we could follow the opposite direction of the gradient, updating the parameters proportionally to it to find the best configuration. The update rule for this algorithm is simply:

$$\theta \longleftarrow \theta - \eta \nabla_\theta \mathcal{L} \tag{1.6}$$

Where η is called learning rate and it is probably the most important non-trainable parameter of a neural network.

The learning rate controls the process of descending the gradient and must be fine tuned for every architecture we build and every dataset we have to inspect. Two undesired behaviours can arise when learning rate is not properly set. Fig. 1.9 shows what happens, in a simple one-

dimensional case with a quadratic loss function, if we try to optimize a model with a learning rate which is too large: we can not reach the minimum because the parameters receive big increments and the particle jumps from one side to the other of the well and definitely escapes from it, causing the training to diverge. The figure displays that even a too small learning parameter is not appropriate, because the training will be too slow to reach the minimum in a reasonable number of iterations. These effects are incredibly enhanced when we deal with the optimization of a non-convex multi-dimensional loss function, causing training failure. Other issues, like instability of the optimization or vanishing gradients on a plateau, motivate the research of more effective algorithms.

There are parameters related to a network, such as the learning rate, that are not trained by the process of optimization. We call them hyper-parameters. Examples include the model architecture itself (type, depth and width of the hidden layers), parameters that control the behaviour of particular layers, such as coefficients in activation functions (e.g. α coefficient in leakyReLU, tab. 1.1), mini-batches' size, number of epochs to train the model. The choice of hyper-parameters is strictly bound to the dataset we have to deal with. The ultimate goal will be to find the configuration which perfoms better on the validation set: test set is used to assess performances of the final model and must not be used, in principle, to take any decision regarding either trainable or non-trainable parameters. There are several types of hyper-parameter search that we can employ, among them we find: manual search, grid search, random search and bayesian search.

In manual search, as suggested by its name, we have to tune manually the model's hyper-parameters. This task requires a lot of experience in model designing and could be particulary difficult to work out. In grid search, instead, we write a table of hyper-parameters and their correspondent possible values, then we train each configuration and select the best performing architecture. This method is more rigorous than random search because we plan in advance the possible configurations to be tried, but it takes a lot of time when the number of hyperparameters increase. The random search consists in drawing random values uniformly from pre-fixed ranges in order to set hyper-parameters; then for each configuration we train the correspondent model and record the best performing one.

The bayesian search is based on Mockus' paper in early 70's, [13]: his strategy treats the objective function to be minimzed (loss or accuracy depending on the case) as a random function and places a prior over it. The prior captures beliefs about the behaviour of the function before its evaluations. After initializing a particular configuration of the model and collecting the objective functions after training, the prior is updated to form the posterior distribution over objective function. The posterior distribution, in turn, is used to build an aquisition function from which the next hyper-parameter configuration is drawn. This ideas are implemented by different algorithms. We used throughout the present work the hyperopt library, that employs the Tree Parzen Estimator (TPE) algorithm to make the decision on the next hyper-parameter configuration.

In the present section we gave an insight of the algorithms that constitutes the core concepts of training a model in ML. With gradient descent and hyper-parameter search we built a framework where a neural network can be shaped to reproduce the desired data generating distribution. These steps are the most important and delicate in the process of learning.

1.2.3 Classification Models

In this subsection we describe tools to assess the results of a classification task: in particular we introduce the confusion matrix formalism and the ROC curve analysis. These tools are firstly described for binary classifications, but where it is possible we show also the generalization to

multiclass tasks. Throughout the discussion it can be useful to have in mind terminology and formalism of statistical hypothesis testing, reviewed in appendix B.1, since we will employ the same symbols used there to highlights the common points of the two topics.

As written in the beginning of section 1.2.1, a classifier is a model that tries to learn which of k categories some input belongs to. In binary classification, a label 0 or 1 is attached to each sample of the dataset. Then, we design a model such that its output will be a scalar in the interval $[0, 1]$. This outcome is in fact a random variable T, since it is a function of a sample in a dataset, which is drawn from the data generating distribution p_{data}. We fix a decision threshold t that divide the unit interval into two parts: if the model's prediction is less than t, a predicted label of 0 is assigned to the sample, 1 otherwise. t is fixed a priori to $1/2$ to ensure the correct balance between decisions, but it will be sometimes interesting changing this value in order to assess classifier's performances in different regimes. Within the multiclass framework, with k categories, we shape the model such that it has a vectorial output with k components, each ranging in the unit interval. The predicted label of a sample will be, in this case, the component's index with the highest output value.

The comparison between the true label and the predicted label of a collection of samples tells the goodness of the trained model. We divide predictions in four subsets:

- true negative (TN), number of 0s correctly predicted;
- false positive (FP), number of 0s predicted as 1s, type I error;
- false negative (FN), number of 1s predicted as 0s, type II error;
- true positive (TP), number of 1s correctly predicted.

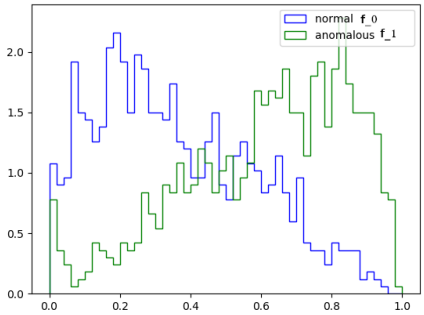
Fig. 1.10a shows how we can visualize results from a binary classifier as a 2×2 matrix. We can print the confusion matrix either with each matrix element corresponding to the absolute number of predictions in that subset, or rather normalizing across columns and setting each element to a percentage between 0 and 1. A good classifier has high diagonal entries, while off-diagonal ones are close to zero. In the normalized format, the mean computed across the diagonal values equal the total accuracy of the model. In the multiclass classification it is possible to print the classification matrix as well, but this time it will be a $k \times k$ matrix and of course all the nomenclature about true/false positive/negative subsets will then be nonsense.

From a trained model we can also compute the scrores' distributions of true 0 and 1 labeled data. Practically, we have to make a large number of predictions and work out the histogram of the results. This method allow us to visualize the f_0 and f_1 pdfs of true 0s and true 1s labelled data. We can assess the goodness of a discriminator if it manages to keep separate the two distributions: an optimal model will draw the f_0 and f_1 as two delta distributions centered respectively in 0 and 1, while the worst possible classifier will represent the two pdf as flat distributions. Fig 1.10b shows an example of such a plot. A decision threshold would divide each pdf's plot into two regions, as in the statistical test discussed in the appendix. We now define some related quantities:

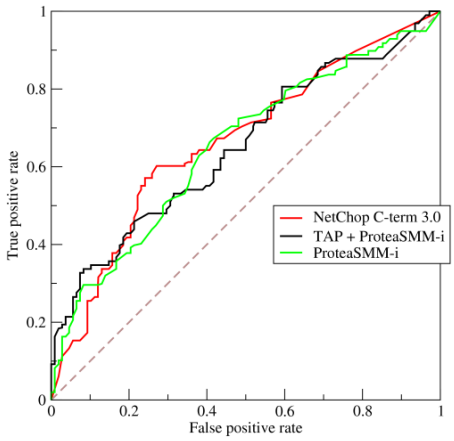
- true negative rate, $\mathrm{TNR} = \mathrm{P}\Big(\mathrm{T} < t | f_0\Big) = \frac{\mathrm{TN}}{\mathrm{TN+FP}}$;
- false positive rate, $\mathrm{FPR} = \mathrm{P}\Big(\mathrm{T} > t | f_0\Big) = \frac{\mathrm{FP}}{\mathrm{TN+FP}}$;
- false negative rate, $\mathrm{FNR} = \mathrm{P}\Big(\mathrm{T} < t | f_1\Big) = \frac{\mathrm{FN}}{\mathrm{FN+TP}}$;
- true positive rate, $\mathrm{TPR} = \mathrm{P}\Big(\mathrm{T} > t | f_1\Big) = \frac{\mathrm{TP}}{\mathrm{FN+TP}}$.

TRUE LABEL			
	0	TN	FP
	1	FN	TP
		0	1
		PREDICTED LABEL	

(a) Confusion matrix of a binary classifier.



(b) Example of the two scores' pdfs, normal refers to true 0s, while anomalous means true 1s.



(c) Examples of ROC curves.

Figure 1.10: Classifier assessment tools.

The last tool we want to introduce shows the behaviour of the classifier when the decision threshold t is changed. The four ratios directly depend on the value of t via an integration interval's extremum. If we consider the couples $\big(\mathrm{FPR}(t), \mathrm{TPR}(t)\big)$ and move continuously the value of the threshold, they will describe a line in a plane, called Receiver Operating Characeristic (ROC) curve. By construction, any ROC curve always goes through two fixed points: $(0, 0)$ and $(1, 1)$. In fig. 1.10c we display three examples of such curves. There is also plotted the dashed diagon line, which is called "line of no discrimination" and signals the situation of random guessing the labels. In the ROC space there are two special lines: the curve referring to the best possible discriminator, which starts from the origin and goes up until the point $(0, 1)$ and from then turns right and remains constan up to $(1, 1)$, and the other curve corresponding to the worst possible classifier, that stays flat from $(0, 0)$ to $(1, 0)$ and then joins the latter point with $(1, 1)$. In between of these two lines there are infinite options, depending on the particular model.

The area under the curve (AUC) is a parameter that tells how good is the classifier. It can be proved that this quantity tells the probability that picked two random samples, the first with true label 0 with score t_0 and the second with true label 1 with score t_1, the quantity t_1 will be greater than the value t_0:

$$\mathrm{AUC} = \mathrm{P}(t_1 > t_0 \,|\, t_0 \sim f_0,\, t_1 \sim f_1)$$

The area under the curve ranges in the interval $[0, 1]$ and if it is greater than 0.5, the model is doing better than random guessing the labels.

1.2.4 Generative Models - GANs

In ML learning literature there are a lot of examples of generative models, such as auto encoders (AEs) and their variational form (VAEs), boltzmann machines and generative adversarial networks (GANs). We focus on GANs, which were firstly proposed by Goodfellow et al. in [4] and since 2014 have been refined and improved in several variants. GANs are deep neural net architectures comprised of two nets: the generator $\mathcal{G}$ which tries to learn the data generating distribution and the discriminator $\mathcal{D}$, that outputs the probability that a input sample came from the training data, rather than from $\mathcal{G}$. In the original paper authors explain their framework ideally as two teams competing one against the other: the first one (the generator) is made by counterfeiters, that try to produce fake currency and use it without being detected; opponents (the discriminator) are analogous to the police, that try to uncover the counteirfeiters' actions.

Mathematically, we start from samples $\mathbf{x}$ from a dataset and a prior distribution $p_{\mathbf{z}}$ used to draw the input $\mathbf{z}$ of the generator's net. We choose $p_{\mathbf{z}}$ to be a distribution with high entropy, such as a normal gaussian. Then $\mathcal{G}$ is a FFNN with parameters θ_g that defines a mapping to data space $\mathcal{G}(\mathbf{z}; \theta_g)$, so we can think the generator as a tool that sample random artificial data with pdf p_g starting from an input noise $\mathbf{z}$. We implement a second neural network $\mathcal{D}(\mathbf{x}; \theta_d)$ that outputs a single scalar value, which represents the probability that the input came from the training dataset, rather than from $\mathcal{G}$. The novelty of the proposal consist in the way the nets are trained, formally they play a so called minimax game:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathcal{L}(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{x \sim p_{data}(x)}[\log \mathcal{D}(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - \mathcal{D}(\mathcal{G}(z)))] \tag{1.7}$$

The expected values are taken, during training, over a whole mini-batch of samples. The equation for the loss function has therefore two contributes that are related to the accuracy of the discriminator. The first one is the expected value for the logarithm of the probability of $\mathcal{D}$ to correctly classify a sample $\mathbf{x}$ from the training set, in binary classification theory we would call this prediction as true negative. The second term is the expected value of the logarithm of the probability of $\mathcal{D}$ to correctly classify an artificial sample taken from $\mathcal{G}$, again this prediction

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. k is the number of steps to apply to the discriminator.

for number of epochs **do**

  for k steps **do**

  - Draw a mini-batch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $\mathbf{z} \sim p_{\mathbf{z}}$
  - Select a mini-batch of m samples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ from the training set
  - Update the discriminator's parameters θ_d by ascending its stochastic gradient:

$$\theta_d \longleftarrow \theta_d + \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \Big[\log \mathcal{D}\Big(x^{(i)}\Big) + \log \Big(1 - \mathcal{D}\big(\mathcal{G}(z^{(i)})\big)\Big)\Big]$$

  end for

  - Draw a mini-batch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $\mathbf{z} \sim p_{\mathbf{z}}$
  - Update the generator's parameters θ_g by descending its stochastic gradient:

$$\theta_g \longleftarrow \theta_g - \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \Big(1 - \mathcal{D}\big(\mathcal{G}(z^{(i)})\big)\Big)$$

Table 1.2: Pseudo-code for training a vanilla GAN. Note that SGD can be replaced by every preferred algorithm.

corresponds to a true positive in a binary classification scheme. In the minimax game, then, we want to adjust $\mathcal{D}$'s parameters in order to maximize the loss function, while θ_g has to be optimized in the opposite direction, namely to grow the chances of $\mathcal{D}$ making a mistake. The training algorithm is summarized in the pseudo-code in tab. 1.2.

It can be proven that if the discriminator is trained till optimality, theoretically, minimizing the GAN's loss function equals to minimize the Jensen-Shannon divergence (JSD) between the generating data distribution p_{data} and p_g, which is defined as:

$$\mathrm{JSD}(p_{data} \| p_g) = \mathrm{KL}\Big(p_{data} \Big\| \frac{p_{data} + p_g}{2}\Big) + \mathrm{KL}\Big(p_g \Big\| \frac{p_{data} + p_g}{2}\Big) \tag{1.8a}$$

$$\mathrm{KL}(p \| q) = \int_{\Omega} p(x) \log \Big(\frac{p(x)}{q(x)}\Big) d\mu(x) \tag{1.8b}$$

where KL is the Kullback-Leibler divergence. JSD is a mesure of distance between probabilities. In practice it is computationally proibitive to re-train $\mathcal{D}$ till optimality at every epoch, then we employed the k steps for loop in the algorithm, that unfortunately does not ensure that the discriminator is trained enough. This is the main source of instability in GANs' training.

In practice we have to build two nets: the discriminator itself and a combined model of the generator and the discriminator with freezed weights, that allow us to update $\mathcal{G}$'s parameters in a supervised way. Although the idea of realizing an unsupervised model through adversarial training of two supervised nets is quite interesting, the authors of [4] themselves faced problems of instability especially during initial phases of training and they proposed to change the second term in the loss function with: $\log \mathcal{D}(\mathcal{G}(\mathbf{z}))$. This change leads to the same fixed point in the dynamics of $\mathcal{G}$ and $\mathcal{D}$, but provides much stronger gradients in the intial phases of training. However this trick does not solve the instability problem in training. This is why a lot of

alternative improved forms of the GAN architecture have been proposed in recent years. In the present work we discuss ls-GANs, wGANs and wGANs-gp.

The least squares GAN (ls-GAN) is an alternative form of generative adversarial networks, introduced in [11]. In the article, authors redefine the objective function of eq. 1.7 and replace it with:

$$\min_{\mathcal{D}} \mathcal{L}_{\text{lsGAN}}(\mathcal{D}) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}\Big[\big(\mathcal{D}(\mathbf{x}) - b\big)^2\Big] + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\Big[\big(\mathcal{D}(\mathcal{G}(\mathbf{z})) - a\big)^2\Big] \tag{1.9a}$$

$$\min_{\mathcal{G}} \mathcal{L}_{\text{lsGAN}}(\mathcal{G}) = \frac{1}{2}\,\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\Big[\,\big(\mathcal{D}(\mathcal{G}(\mathbf{z})) - c\big)^2\,\Big] \tag{1.9b}$$

where a and b represent the label scheme for the discriminator, respectively for artificial and real data. The number c instead is the value that $\mathcal{G}$ wants $\mathcal{D}$ to believe for fake data. In our code, we always fix $b = c = 0$ and $a = 1$. It can be proven that minimizing the objective function of ls-GANs yields minimizing the Pearson χ^2 divergence between $p_{data} + 2p_g$ and $p_{data} + p_g$, defined as:

$$\chi^2_{Pearson}(p\|q) = \int_{\Omega} f\Big(\frac{p(x)}{q(x)}\Big) q(x) d\mu(x) \tag{1.10}$$

where $f(t) = (t-1)^2$.

This alternative architecture for GANs does not solve all the problems of the vanilla GAN: in our work we found lsGAN very likely to fail converging to a good minimum of the loss function and we then discarded it and turned to a stabler version of the model, the wGAN. As we previously said, GANs are really difficult to train because the two models have to improve in parallel: neither has to overwhelm the other, both have to become better step by step, otherwise the training simply stops. It is clear that in an adversarial training $\mathcal{D}$ has to inform $\mathcal{G}$ about the direction to follow in order to output better fake data, but this only works if $\mathcal{G}$ is in fact on the right track. That is why if $\mathcal{D}$ is too powerful it can easily uncover the fake data, whereas if $\mathcal{G}$'s loss suddenly goes down, it means that the generator has found the way to fool $\mathcal{D}$ with some garbage data that does not in fact even resemble the true samples.

Wasserstein GAN is an adversaral network framework, defined in [1], in which the cost function is modified in order to minimize the following earth mover's distance, or wasserstein-1, between probabilities:

$$W(p_{data}\|p_g) = \inf_{\gamma\in\Pi(p_{data},p_g)} \mathbb{E}_{(x,y)\sim\gamma}\Big[\|x-y\|\Big] \tag{1.11}$$

where $\Pi(p_{data}, p_g)$ is the set of all joint distributions that have as marginals p_{data} and p_g. We can think of this quantity as the smallest amount of work needed to transport the mass difference $\|x-y\|$ along each possible path γ in the distribution space. The fact that W goes to zero implies that also the difference between pdfs is going to zero.

Since the infimum in equation 1.11 is highly intractable from a computational point of view, we end up minimizing the following objective function:

$$\mathcal{L} = \mathbb{E}_{\mathbf{x}\sim p_{data}}\Big[\mathcal{D}(\mathbf{x})\Big] - \mathbb{E}_{\tilde{\mathbf{x}}\sim p_g}\Big[\mathcal{D}(\tilde{\mathbf{x}})\Big] \tag{1.12}$$

It can be proven that this equation sets un upper limit for the wasserstein-1 distance, if the function defined by $\mathcal{D}$ is a 1-Lipschitz function. In order to enforce the Lipschitz constraint, we introduce a weight clipping procedure on discriminator's parameters. Weight clipping consists in replacing the value w of every weight in a net with the quantity $(\text{sign}(w)\cdot c)$ if $|a| > c$, where c is a real positive value. This way we are sure that weights lie in the compact interval $[-c, c]$. The weight clipping procedure must be applied every time we update the discriminator weights

Algorithm 2 wGAN with gradient penalty. Default values are: $\lambda = 10$, $n_d = 5$. n_d is the number of $\mathcal{D}$ iteration per $\mathcal{G}$ iteration in a single epoch. m is the mini-batch size.

for number of epochs **do**
  for n_d steps **do**
    for $i = 1, \dots, m$ steps **do**
      - Sample real data $\mathbf{x} \sim p_{data}$, noise $\mathbf{z} \sim p_{\mathbf{z}}$, a random number from uniform distribution $\epsilon \in [0, 1]$
      - $\tilde{\mathbf{x}} \longleftarrow \mathcal{G}(\mathbf{z}; \theta_g)$
      - Interpolate real and fake samples: $\hat{\mathbf{x}} \longleftarrow \epsilon\mathbf{x} + (1-\epsilon)\mathcal{G}(\mathbf{z}; \theta_g)$
      - $\mathcal{L}^{(i)} \longleftarrow \mathcal{D}(\tilde{\mathbf{x}}) - \mathcal{D}(\mathbf{x}) + \lambda(\|\nabla_{\hat{\mathbf{x}}}\mathcal{D}(\hat{\mathbf{x}})\|_2 - 1)^2$
    end for
    - $\theta_d \longleftarrow \text{Adam}(\nabla_{\theta_d} \frac{1}{m} \sum_{j=1}^{m} \mathcal{L}^{(j)})$
  end for
  - Sample a mini-batch of noise from prior distribution $\mathbf{z} \sim p_{\mathbf{z}}$
  - $\theta_g \longleftarrow \text{Adam}(-\nabla_{\theta_g} \mathbb{E}[\mathcal{D}(\mathcal{G}(\mathbf{z}))])$
end for

Table 1.3: Pseudo-code for training a wGAN-gp. We discuss Adam algorithm in appendix A.2.

and unluckily it is very computationally expensive, since we have to directly access and edit all the parameters in the net.

In order to avoid the time consuming weight clipping process we exploit an improved version of the wGAN with gradient penalty (wGAN-gp), which was introduced in [5]. The algorithm described in tab. 1.3 implement the pseudo-code for a minimax game with objective function:

$$\mathcal{L} = \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g}[\mathcal{D}(\tilde{\mathbf{x}})] - \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r}[\mathcal{D}(\mathbf{x})] + \lambda \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{x}}}[(\|\mathcal{D}(\hat{\mathbf{x}})\|_2 - 1)^2] \tag{1.13}$$

where λ is a parameter which ensures that the three terms in the loss function are equally important. The third contribute to $\mathcal{L}$ is a quantity that penalizes the model if the expected value for the gradients in an iteration differ from unity. There exists a theorem which states that a differentiable function is 1-Lipschitz if and only if it has gradients at most 1 everywhere, therefore the introduction of gradient penalty in the objective function defines a way to enforce the Lipschitz constraint alternative to the weight clipping procedure.

Chapter 2

Applications

In this chapter we present our work involving the wGAN-gp architecture and its applications to two particular issues: data augmentation and anomaly detection. We present our work as follows: first we discuss the dataset used for training and testing our setup, then we focus on training the GAN architecture in section 2.2, section 2.3 gives a description of the data augmentation problem, while in section 2.4 we deal with unsupervised anomaly detection. In order to help the flow of exposure, we moved some big figures at the end of the chapter.

2.1 Dataset

In this section we introduce the dataset used as input of the algorithm. Our studies focus on learning the probability distribution function of a one dimensional random variable x, which takes values in the interval $[0, 1]$. We employ a generator of pseudo-random numbers to simulate outcomes of x: we use functions implemented in NumPy library, in particular in the module called random [6]. A sample in the dataset is an array of values corresponding to an histogram of 256 bins computed from $10k$ points of outcomes of x. We refer to an example in the dataset also as an image, since it has a grid shape and can be viewed as a one dimensional image. The dataset is a mixture of images from different classes, depending on the particular functional form of the pdf that x follows. We discriminate between six different probability distribution functions (pdfs):

- exponential pdf $\rho(x) \sim \exp(-8x)$;
- linear pdf: $\rho(x) = 1 - x$
- exponential pdf plus gaussian signal in the tail: $\rho(x) \sim \exp(-8x) + A\mathcal{N}(0.8, 0.02)$, where $\mathcal{N}(\mu, \sigma)$ is the gaussian distribution centred at mean μ with variance σ^2;
- exponential pdf plus gaussian signal in the bulk: $\rho(x) \sim \exp(-8x) + A\mathcal{N}(0.2, 0.02)$;
- linear pdf plus triangular signal in the tail: $\rho(x) \sim 1 - x + A\mathcal{T}(0.8, 0.02)$, where $\mathcal{T}(a, b)$ is the triangular distribution peaked in a, with lower and upper limit equal to $(a - b)$ and $(a + b)$ respectively;
- linear pdf plus traingular signals in the tail and in the bulk: $\rho(x) \sim 1 - x + A[\mathcal{T}(0.8, 0.02) + \mathcal{T}(0.2, 0.02)]$.

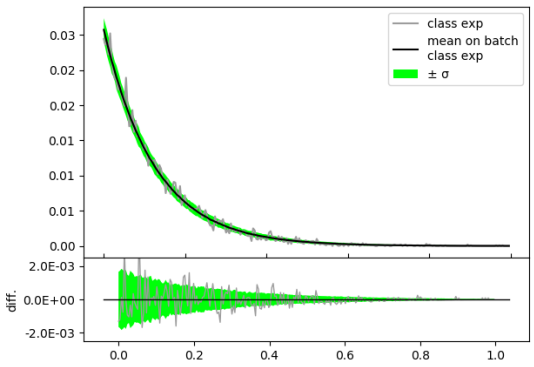
class	pdf
0	exponential or linear
1	exp + gaussian $\mathcal{N}$ peak in the tail
2	exp + gaussian $\mathcal{N}$ peak in the bulk
3	linear + triangular $\mathcal{T}$ peak in the tail
4	linear + $\mathcal{T}$ peak in the bulk and in the tail

Table 2.1: Summary of classes in the dataset

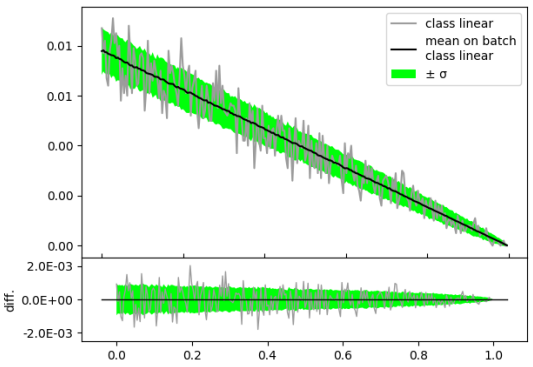
A is a coefficient that measures the relative height of the anomalous peak: in our experiments is fixed to $5 \cdot 10^{-3}$.

We arrange the pdfs in in five classes: we group together exponential and linear pdfs and refer to them as normal or background, while the others are usually called anomalous. Tab. 2.1 summarizes dataset composition, while fig 2.0 (end of the chapter) shows different panels with normal and anomalous images randomly picked out from the dataset. The number of images per class in the dataset depends on the model we have to train: for GANs' training, indeed, only images from a specific class are employed, whereas for classifiers' training the number of images must be balanced between classes, otherwise the models can easily reach better values for the loss simply learning the most represented class and neglecting others.

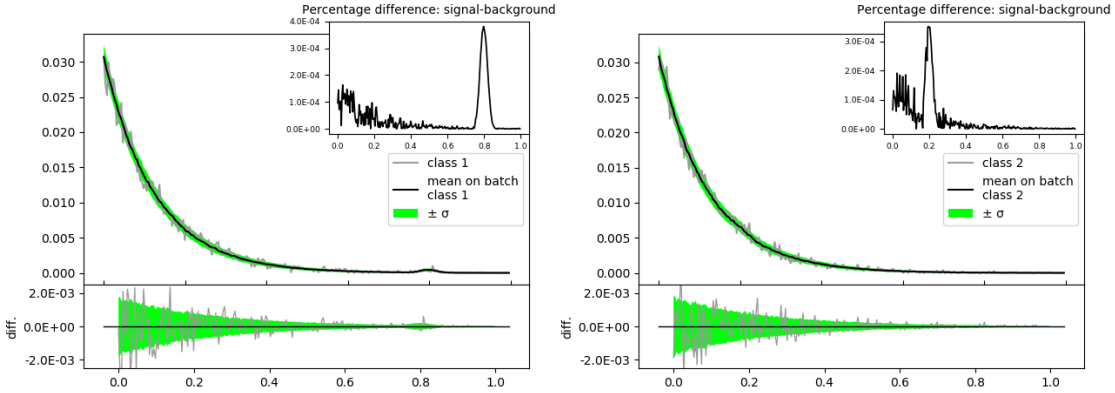
We have just described the composition of training sets. Test sets are a little different, since we would like to be sure that processing them with trained models gives a good estimate of the generalization error. We refer to test sets with the term "featured", in order to underline that we sometimes apply little transformations on images, such as left/right shifting or smearing parameters that control the shape of either the pdfs or the anomalous signals (A parameter, mean standard deviation of the singal). This way we ensure that the model has never seen such images during training.



(a) Exponential pdf

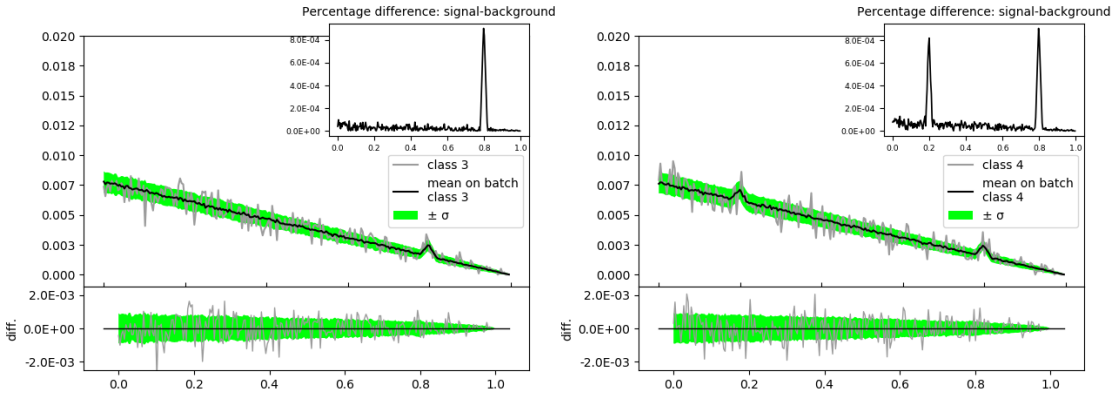


(b) Linear pdf



(c) Class 1

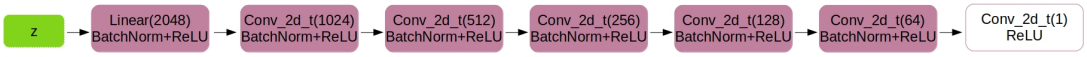
(d) Class 2



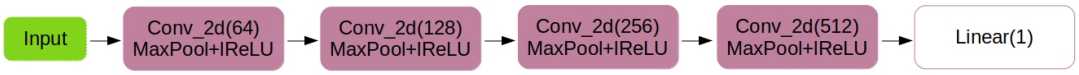
(e) Class 3

(f) Class 4

Figure 2.0: Example of classes in the dataset. Plots *(a)-(b)* refer to normal images. Plots *(c)-(f)* show anomalous images, the curve in each upper box describes the ratio $\frac{y_i^s - y_i^b}{y_i^b + 1E-10}$, where $y_i^{b/s}$ is the mean value (across the background/signal images in the dataset) of the i-th bin in the histograms.



(a) Generator $\mathcal{G}$.



(b) Discriminator $\mathcal{D}$.

Figure 2.1: wGAN-gp model's architecture.

2.2 GAN training

Our work focuses on implementing different types of GANs as discussed in section 1.2.4. According to our experience, the best performing architecture is the wGAN-gp. The problem in training a GAN is mainly due to the instability of the minimax game: in adversarial training the algorithm converges if a dynamical equilibrium is reached between the discriminator and the generator. In game theory this situation is called Nash equilibrium. The issue concerning Nash equilibrium is that it is really difficult to find it searching in the multi-dimensional parameter space. In principle, training history, namely records of all loss function's values at each epochs, is not informative regarding the success of training: a small loss' value does not imply good quality of images generated by $\mathcal{G}$. Nevertheless, there are some recurrent patterns that clue a successful training.

Although we tried different architectures as explained before, our experiments are based only on wGAN-gps, since their training is more stable than that of other kinds of GAN. They require minimal parameter tuning compared to the original model proposed in [5] and are less likely to collapse generating only random noise images. Both generator and discriminator are deep models: fig. 2.1 shows how layers are stacked in order to form these nets. 2-dimensional convolutional layers are used to inspect images: in order to fit convolutional kernel's size, samples are pre-processed adding a second dimension of lenght one in the array. According to the discussion in section 1.2, convolutional layers, unlike dense ones, introduce some benefits (sparse interactions, parameter sharing, equivariant representations) that we would like to exploit.

It is important to note the role of activation functions: inside the generator in particular we used ReLU because the output images are non-negative arrays of values. Further improvements, beyond the scope of the present thesis, can be made in the design of the model in the following sense: since ReLU activation function allows the generator to output exact zero values for bins in the tail, one possible issue could be that after training $\mathcal{G}$ cuts to zero tails of distributions. We will in fact highlight this again in a following paragraph. We suggest to cure this behaviour substituting the final ReLU with an ELU, which provides a more smooth trend where the pdf described in an image approaches zero values. ELU activation indeed does not cut to zero negative values, it tries to interpolate the transition from positive to negative values with an exponential form.

The code is written in Python language and based on version 1.10 of TensorFlow library [8]. None of the high level Keras Application Programming Interfaces (APIs) are used to implement neural networks, because we would like to have the best possible control over the model. Therefore we code the entire algorithm starting from low level tensorial manipulations. We define functions containing instructions on how each layer performs (we work with dense, 2D convolutional and 2D convolutional transposed layers) and then we implement a class called GAN that groups

Parameter	Value
α	0.2
momentum	0.48641
ϵ	$1E-5$
optimizer	Adam
initial lr	$1E-4$
batch size	128
epochs	30
$\mathcal{D}$ capacity	861 953
$\mathcal{G}$ capacity	6 317 185

Table 2.2: GAN hyperspace.

properties describing all the GAN's parameters and methods that allow the compiler to initialize (in this stage we build the model stacking the corresponding layers and defining its loss function), train (here we set model's optimizer) and test the model. We note that the same approach is employed with classifiers with the implementation of a class called DISC.

In every described experiment we will always train a GAN over a set of $25k$ images belonging to the desired class, for 30 epochs and fixing the batch size for the optimization at 128 examples from the dataset. We decide to set the number of epochs to such a relatively small value, considering that the algorithms take approximately 30 minutes to finish on an NVIDIA GeForce GTX 1070 GPU and in the following, during simulations, we will implement 60 different models. Finally and more importantly, we see that the quality of images does not substantially improve from 30 epochs up to 100 epochs. Tab. 2.2 lists GAN's hyperparameter values. Momentum and ϵ are parameters of the BatchNormalization layer, defined in [9].[1] α is the parameter used in LeakyReLU activation.

Within the list of hyper-parameters figures the initial learning rate of Adam optimizer, corresponding to the value of $1E-4$. We call it "initial" because we apply a learning rate schedule: the plan consists in halving the learning rate once the discriminator's loss is sufficiently small, in particular when it is in the range $[-0.25, 0.25]$ after ten epochs of training. Since Adam is an adaptive learning method, it is not suggested to implement learning schedules, but we find this *ad hoc* strategy very effective, allowing us to generate more precise artificial images, as pointed out in fig. 2.2, shown at the end of this section.

During training, we checkpoint the model at each epoch. The big issue concerning GANs is how to find the best model within the set of saved nets: simply looking at loss' values do not ensure good performances. We would like to implement a selection algorithm that signal which is the generator that output the best looking artificial samples. We initially used to choose best $\mathcal{G}$, inspecting each generator by eye, but this not automated method takes too much time when there are a lot of models. Furthermore, we would like to erase any form of subjective decision from our experiments, according to the experimental method. In order to overcome this problem, at the end of each epoch of training, we sample from $\mathcal{G}$ a batch of images and perform over them a chi-squared test. This test evaluates the distance of the distribution of the mean over the generated sample from the reference distribution of the mean over a batch of real images from

[1] BatchNormalization layer helps to reduce covariate shift. The term covariate shift refers to a change in the distributions of layers' inputs. In deep FFNN, final layers experience as input the output of the upstream layers, then a change in net's parameters can be amplified while feedforwarding, forcing later layers to continously adapt to new distributions. This behaviour can dramatically slow down learning, then BatchNormalization layers act as architecture embedded regularizers.

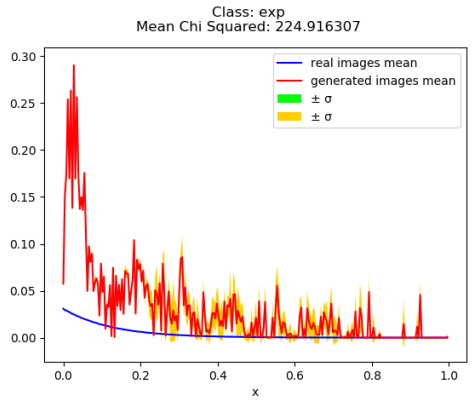
the training set. Formally, a chi squared test is defined by:

$$\chi^2 = \sum_{i=1}^{n_{\text{bins}}} \frac{(E_i - O_i)^2}{E_i}$$

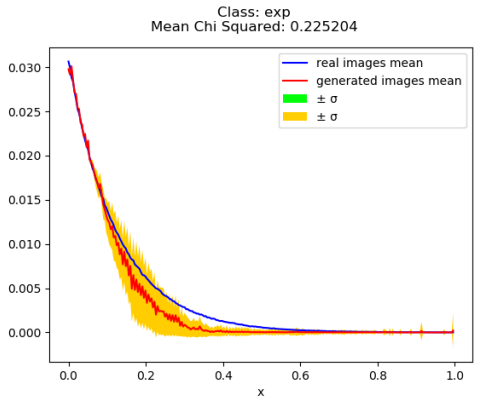
where E_i and O_i are respectively expected (real images) and observed (artificial images) histogram's values. We keep only the five models with the lowest chi squared result and select the best one from them inspecting by eye this smaller subset.

This proposed algorithm saves some time, but unfortunately do not solve at all the subjectivity issue described above. We leave the addressing of this problem for further studies, noting that one way out could be keeping track of several statistical metrics across epochs (not just chi squared) and selecting the best model according to the one that performs better in most of these quantities. We mention some tests and metrics that can be useful in this direction, such as Kolmogorov-Smirnov test, Kullback-Leibler divergence, Jensen-Shannon divergence, Wasserstein 1 distance and in general all the f-divergences. A little of post-processing in this evaluation could be needed: it would be interesting to know how the chosen metrics change when little trasformations, like shifting or smearing, are applied to artificial images. The best model should have the well-desired quality of being roubust against this small perturbations.

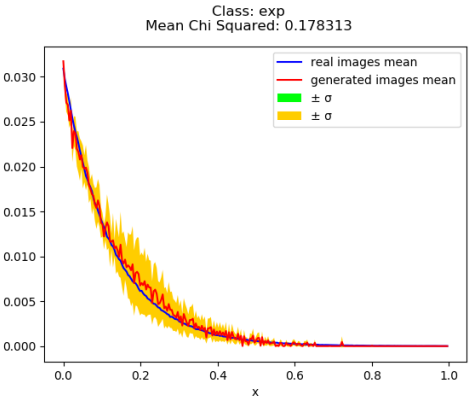
In the following we perform a data augmentation task with GANs, enhancing the dataset of a new classifier, in order to test improvements in classifications. We train several wGAN-gp models for each of the classes in the dataset described in section 2.1. Fig 2.3 displays some examples of fake and real images from trained models.



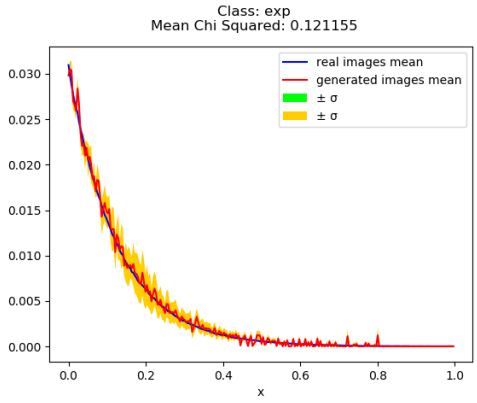
(a) Epoch 3



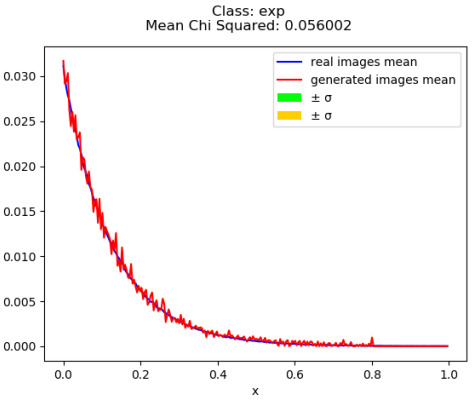
(b) Epoch 8



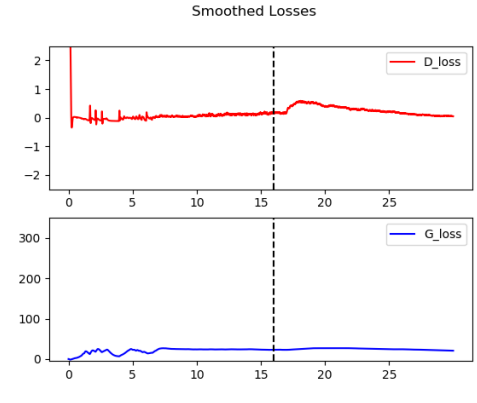
(c) Epoch 16



(d) Epoch 17

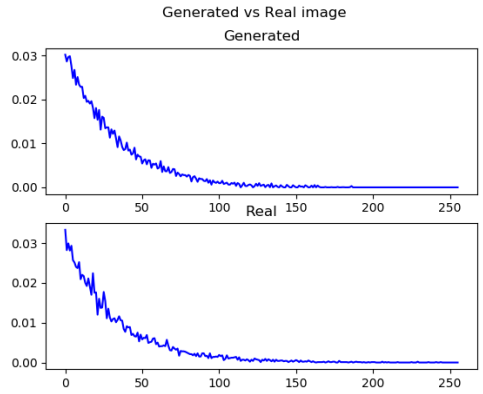


(e) Epoch 29

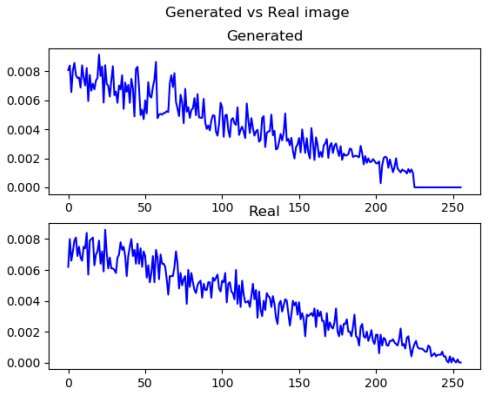


(f) Smoothed Losses

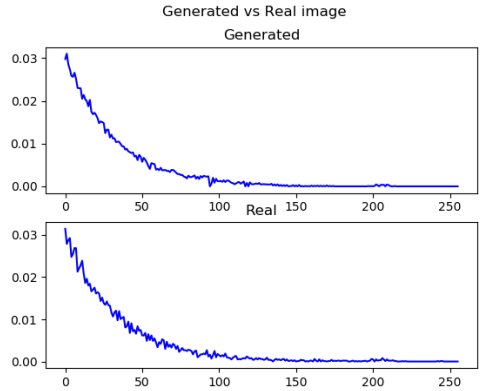
Figure 2.2: wGAN-gp training session. Plots *(a)-(e)* display the evolution during epochs of the images generated by $\mathcal{G}$. Note the decrease of generated images' standard deviation on the mean (yellow bands) from epoch 16 to epoch 17, due to the special learning rate schedule. Box *(f)* shows loss functions' trends: discriminator's loss contains informative details. During the initial 5 epochs, the curve is spiky, reporting that training has to get stabler yet and correspondent images are dominated by random noise (compare it with figure *(a)*). The vertical dashed line corresponds to learning rate's halving. Right after the line a little bump appears in the curve, due to the employed Adam optimization algorithm. We showed that despite this enhancement, this method improves the image quality. Generator's loss instead has the typical pattern of the blue curve. It is not informative, in the sense that lower values of the cost function do not always imply better images.



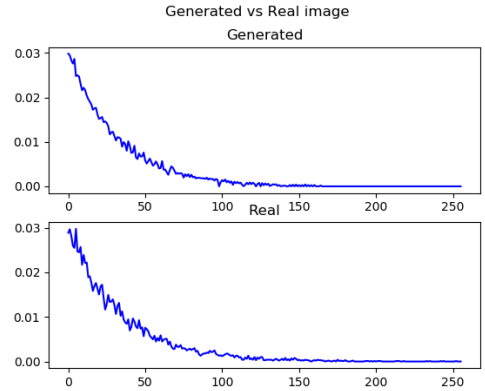
(a) Class 0, exp.



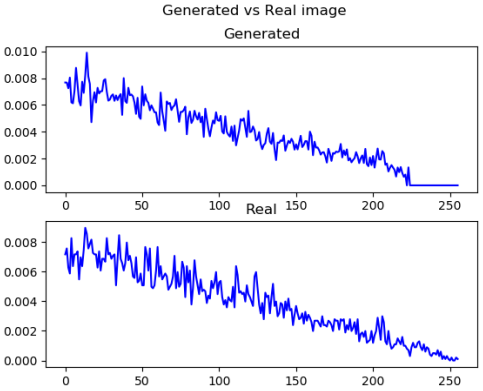
(b) Class 0, linear.



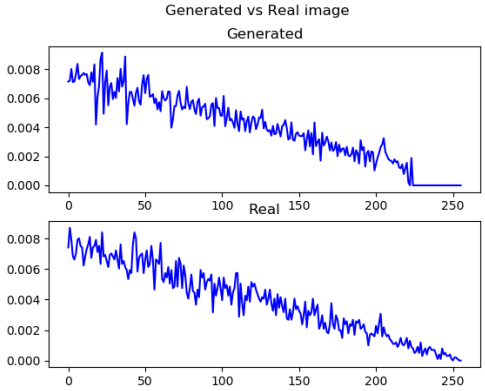
(c) Class 1.



(d) Class 2.



(e) Class 3.



(f) Class 4.

Figure 2.3: Examples of generated and real images. x axis' labels correspond to the numer of bin in the image, rather than the value of the random variable. Conversion is made with the formula: $x = n_{\text{bin}}/256$. Fake ones are almost undistinguishable from reals, the only issue is the one regarding the relationship between ReLU and ELU activation functions that causes the cut in tail clearly visible in plots *(e)-(f)*.

2.3 Data Augmentation

Data augmentation in ML literature is a technique exploited to enhance small datasets. In medicine and other fields of application supervised techniques are strongly penalized by the lack of enough labelled data. In medicine, for example, data that come from invasive medical tests and exams can not be simply carried out for the purpose of collecting large datasets. New images are usually created from original ones, applying on them little transformations, such as rotations, introduction of gaussian noise, cropping, flipping and scaling. In HEP, instead, a potentially infinite number of data is available thanks to MC simulations.

In this thesis the aim of data augmentation is not just to enhance a small dataset, but also assess the quality of GAN generated images. The experiment consists in training a GAN for each of the classes of images in the training set of a multiclass discriminator. Hence, we exploit $\mathcal{G}$ to output an arbitrary number of artificial images to be inserted either in the classifier's training set or in its test set, rather than in both of them. We expect two possible outcomes of the experiment: in the first case the artificial data have a bad quality and spoil the classification, alternatively they are good enough to mantain performances of the classifier as it were before augmentation or even improve them.

We perform two different experiments: a reduced one and a complete one. In the former we implement a binary classifier, that tries to distinguish between just two particular classes of the dataset described in section 2.1, namely images belonging to the class 0 with exponential pdf and class 1, the exponential pdf with a signal in the tail. In the latter we use the full dataset including images drawn from all the possible pdfs and we implement a multiclass discriminator. Data augmentation is made with images coming from two and six GANs respectively. Fig. 2.4 shows the architectures of the classifiers, while hyper-spaces are summarized in tab. 2.2a and tab. 2.2b. Note that the number of $\mathcal{C}$'s trainable parameters, the capacity, in the reduced problem is less than the complete form of $\mathcal{C}$ by a factor of 10. This is a natural choice, since model's capacity parameter has to be proportionate to the complexity of the dataset to be learnt.

Each training set is made of $10k$ images, equally divided between the classes of the corresponding classifier. From training set we hold 15% of the images to be used as a validation set. Test set, instead, counts $5k$ featured images equally divided between classes as well. In order to fix the hyper-parameters of the two classifiers, for each we run 200 trials of hyperopt optimization algorithm, maximizing model's accuracy on validation data. The classifier loss function in both cases is the sparse softmax crossentropy. Models are trained for a high number of epochs, namely 500. But since early stopping is employed to prevent overtraining, every instance of $\mathcal{C}$ trains for a different amount of time. We set early stopping with a patience of 10 epochs and stop training when model's accuracy has not experienced any improvement. We check for this improvements

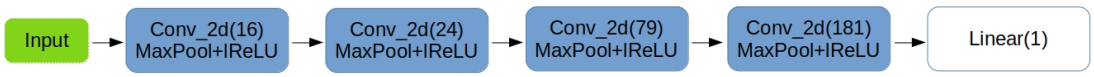
(a) Reduced problem.

Parameter	Value
α LeakyReLU	0.2
batch size	74
optimizer	RMSProp
learning rate	$2.0452 \cdot 10^{-4}$
$\mathcal{C}$ capacity	83 639

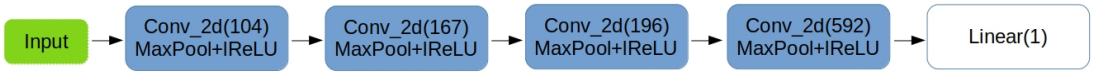
(b) Complete problem.

Parameter	Value
α LeakyReLU	0.2
batch size	71
optimizer	Adam
learning rate	$4.1561 \cdot 10^{-4}$
$\mathcal{C}$ capacity	832 904

Table 2.3: $\mathcal{C}$'s hyperspace.

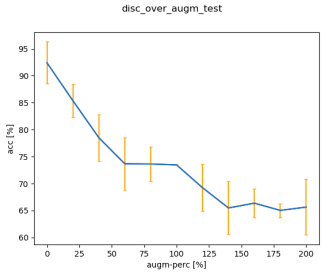


(a) $\mathcal{C}$ reduced problem.

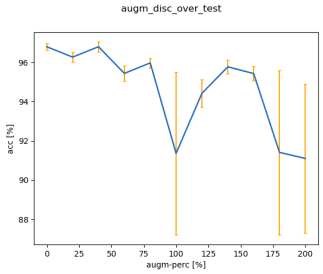


(b) $\mathcal{C}$ complete problem.

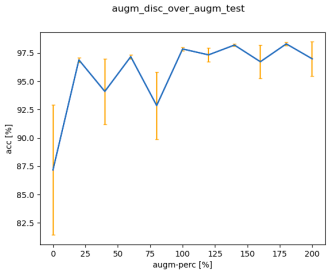
Figure 2.4: Classifiers $\mathcal{C}$.



(a) Augmenting test set.



(b) Augmenting traininig set.



(c) Augmenting both test and training set.

Figure 2.5: Reduced problem: classifier's accuracy as a function of the augmentation percentage. Taking the mean of the accuracy in the three instances of $\mathcal{C}$ at $p = 0$ yields the reference value of $92 \pm 6\%$.

only after the 50-th epoch and from that time, every 10 epochs.[2]

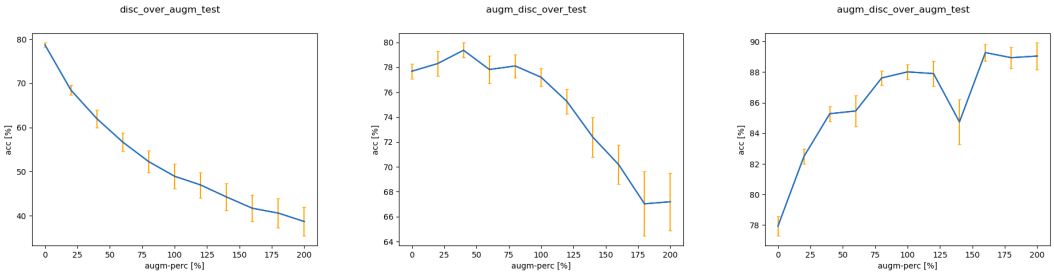
In summary, each experiment consists in training a GAN for each class in the problem. Then we train three instances of the classifier $\mathcal{C}$, with a fixed augmentation percentage p: one augmenting its test set, one augmenting its training set and one augmenting both.[3] The aim is to check performances of $\mathcal{C}$ when p ranges in the interval $[0, 2]$ with steps of 20%. Since the algorithm is not deterministic, we are interested of presenting results with a consistent statical support. Therefore we repeat the experiment 10 times, in order to compute mean values and standard errors on the mean.

Fig. 2.5 and fig. 2.7 - 2.9 display results for the reduced experiment in terms of accuracy trends and classification matrices respectively. Plots show that the reference value for accuracy, when augmentation percentage p is zero, equals to $92 \pm 6\%$. This estimate is provided by the mean of the values extracted from the three confusion matrices with $p = 0$. With this in mind we present results divided in the three different approaches to data augmentation of a classifier's dataset.

- Augmenting test set: classification accuracy clearly decreases as p increases. This behaviour is perfectly natural, since we are testing models on images that they have never processed. Confusion matrices tell in particular that models completely mis-classify exponential images

[2]The optimization in early stages of training is unstable: loss and accuracy fluctuates before settling. Then early stopping may be triggered before the model has even learnt nothing.

[3]Augmentation percentage is the fraction of GAN generated images we add to a particular dataset over the total number of examples in the same dataset before augmentation.



(a) Augmenting test set.

(b) Augmenting traininig set.

(c) Augmenting both test and training set.

Figure 2.6: Complete problem: classifier's accuracy as a function of the augmentation percentage.

as p increases, they in fact simply randomly guess the label, leading to an accuracy next to 50%. Even accuracy over class 1 images slightly drops in the $p = 200\%$ confusion matrix.

- Augmenting training set: accuracy distribution is basically flat, has most values are around 96%. Some points have lower accuracy with a large error, but are equally consistent with the flat trend. Therefore GAN generated images in the training set do not spoil the classification.

- Augmenting both: again the figure shows a flat trend for the accuracy, but globally this quantity is higher in this kind of augmentation than in the others. This signals that artificial images are correctly learned by $\mathcal{C}$s.

Fig. 2.6 and fig. 2.10 - 2.12 depict results of the complete experiment in the same way as before, with trends and confusion matrices. This time our algorithm records a reference accuracy of 78 ± 3 when $p = 0$. We immediately note that a multiclass discrimination is far more difficult than a binary one. In particular, models struggle in learning the difference between class 3 and class 4, resulting in off-diagonal high entries for the 2×2 submatrix in the lower right corner of confusion matrices. Another important source of mis-classification is given by mistakes between class 2 and class 0, due to the similarity of the exponential pdf and the exponential plus bump in the bulk one. Data augmentation with the completed experiment provides more marked trends than in the previous problem.

- Augmenting test set: left panel of fig. 2.6 show a smooth decrease in accuracy.

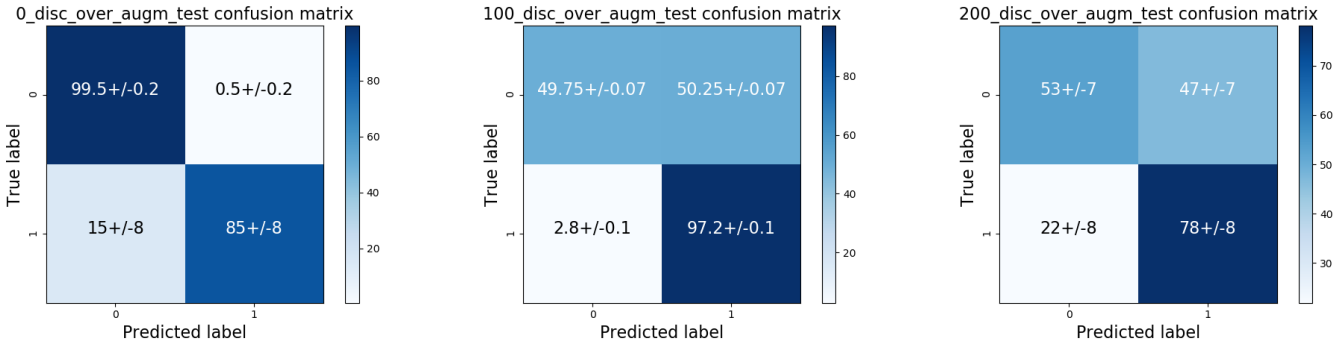
- Augmenting training set: after an initial increase in performances, accuracy drops. This probably because augmenting the training with more and more artificial images cause the model to be focused in learning their distribution which could be slightly different from the real images' one. This behaviour suggests that our algorithm leaves constant performances of the classifier up to about 100% of augmentation, in the range $[0, 50\%]$ results are even better, but after that point trend reverses and definitely worsens the accuracy. Confusion matrices underline that the problem resides in the classification between classes 3-4 and 0-2.

- Augmenting both: right panel of fig. 2.6 prove again that, in fact, GAN generator outputs good quality images, that are correctly classified by $\mathcal{C}$ with increasing accuracy as the augmentation percentage increases. Except from the point at $p = 140\%$, that corresponds to a drop in performances probably due to a failure in some of the trainings that lowers

a lot the mean value, the trend is clearly positive. We reach an accuracy of 89 ± 2 at $p = 200\%$, that corresponds to a relative increment of 14% with respect to the reference value.

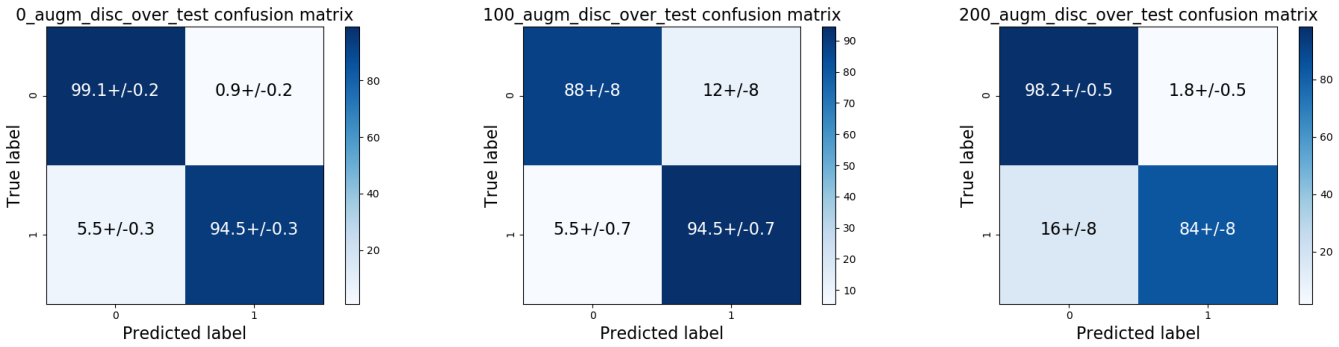
Alongside clean trends in plots, there is always a source of randomness in training that leads to unconsistent points with global trends or unexpected large errors. Randomness in our training procedure has to main contributions. The first one is the fact that GAN's generator samples images from gaussian random noise (with mean 0 and variance 1). It could be interesting to study what happens when fake images are generated by $\mathcal{G}$ from gaussian noise centered at a slightly different mean value. The second factor that contributes to randomness in the algorithm is the process of training itself: weights in the net are randomly initialized, samples in the dataset are feeded in batches to the model in casual order, shuffling of the training set is applied at each epoch end. The optimization algorithm do not always start at the same point in net's parameter space and do not even follow the same updates' path during every training.

We can conclude from our experiment that data augmentation with the trained GAN generator helps classification, even in the complete multiclass mode, if the augmentation percentage is kept below the 50% threshold. Fake images from $\mathcal{G}$ are not drawn from the same data generating distribution as the real ones. Discrepancy between the two functional forms is the main reason of performances' worsening above threshold. A deeper study in GAN architectures and model designing might improve the obtained results.



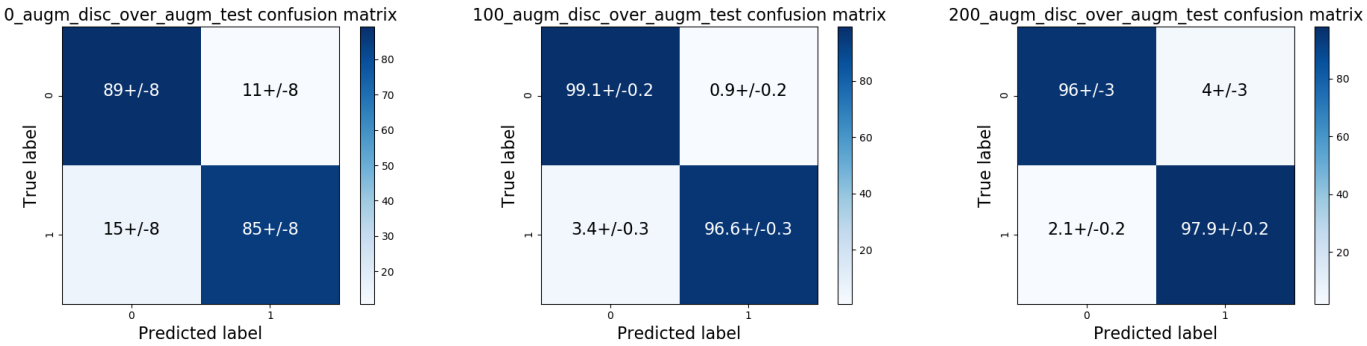
(a) $p = 0$ (b) $p = 100\%$ (c) $p = 200\%$

Figure 2.7: Reduced problem: augmenting test set



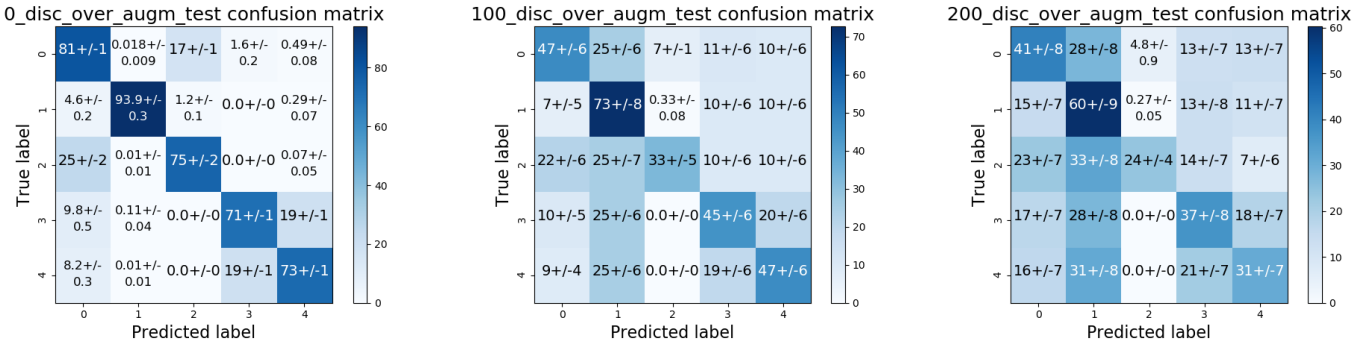
(a) $p = 0$ (b) $p = 100\%$ (c) $p = 200\%$

Figure 2.8: Reduced problem: augmenting training set



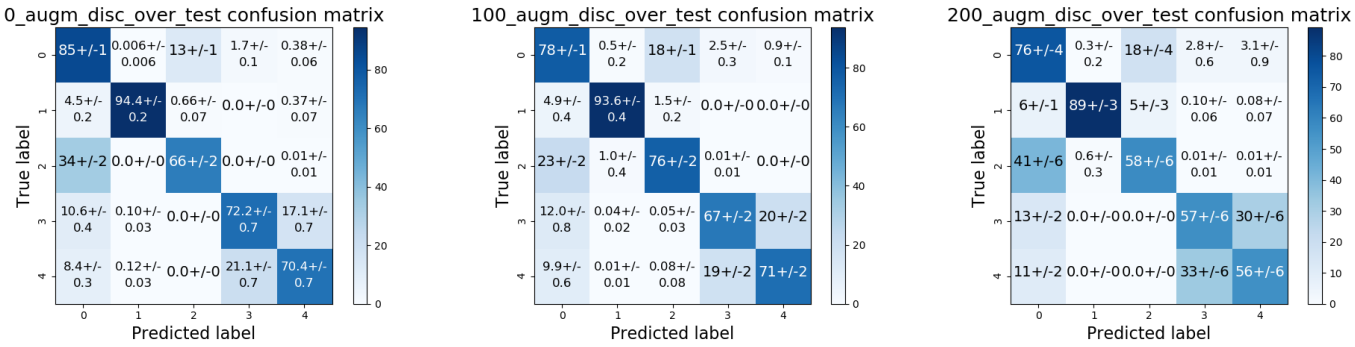
(a) $p = 0$ (b) $p = 100\%$ (c) $p = 200\%$

Figure 2.9: Reduced problem: augmenting both test and training set



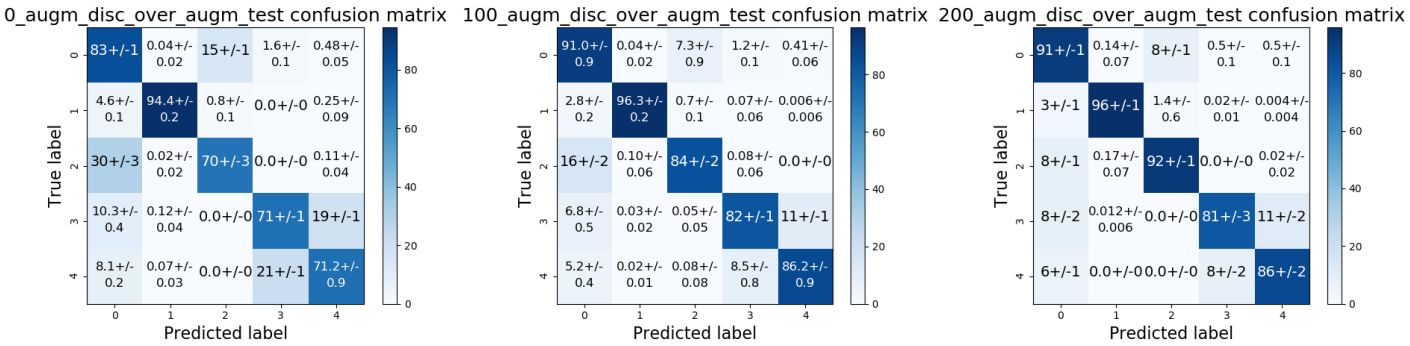
(a) $p = 0$ (b) $p = 100\%$ (c) $p = 200\%$

Figure 2.10: Complete problem: augmenting test set



(a) $p = 0$ (b) $p = 100\%$ (c) $p = 200\%$

Figure 2.11: Complete problem: augmenting training set



(a) $p = 0$ (b) $p = 100\%$ (c) $p = 200\%$

Figure 2.12: Complete problem: augmenting both test and training set

2.4 Anomaly Detection

In this section we evaluate performances of GAN's discriminator as an unsupervised tool that warns the user of a tension in inspected images, due to anomalous signals, from the reference pdf learnt during training. The idea is to employ $\mathcal{D}$s trained over background images, following either exponential or linear pdfs, to process a featured test set made of $5k$ images containing all possible types of histograms. We train 20 wGAN-gps and predict the loss score of each example in the test set. We would like this score to answer the question "is the sample fake?", therefore consider from wGAN-gp's loss function just the second term, red circled in fig 2.13.

$$\mathcal{L} = \mathbb{E}_{\tilde{\mathbf{x}}\sim\mathbb{P}_g}\left[\mathcal{D}(\tilde{\mathbf{x}})\right] - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_r}\left[\mathcal{D}(\mathbf{x})\right] + \lambda\mathbb{E}_{\hat{\mathbf{x}}\sim\mathbb{P}_{\hat{x}}}\left[(\|\mathcal{D}(\hat{\mathbf{x}})\|_2 - 1)^2\right]$$

Figure 2.13: wGAN-gp loss. Red circle highlights discriminator's estimate of the image to be anomalous: greater values signal greater probabilities of the sample to be anomalous

Of course, GANs trained over exponential images are feeded only with exponential-like images (normal exponentials, class 1 and 2), while the others experience linear-like examples. We first build for each trained $\mathcal{D}$ its estimates of the pdfs of background and anomalous images. In order to reach this aim, two histograms of scores' records are computed from background images and anomalous ones. The separation between the two curves gives the discrimination power of the corresponding $\mathcal{D}$. Since each instance of the discriminator comes from a separate training, scores range in very different intervals, then a standardization of the pdfs is applied: we process the entire gan's training set and evaluate from it the mean and standard error of the scores and with these values standardize test set's results.

Fig. 2.16 - 2.17 show outcomes of the described procedure. These plots prove that $\mathcal{D}$ in fact succed in discriminating between background and anomalous images. In order to quantitatively evaluate the goodness of these discriminators, we will separately train four different supervised classifiers and compute their pdfs over the same test set as before. Binary classifiers are trained with slices of the $10k$ images dataset (retaining the relevant classes only) used also in the data augmentation application in section 2.3. We expect from comparisons of supervised and unsupervised models a simple qualitative outcome regarding performances: supervised classifiers have a greater quantity of available information during traning, because they have access to samples' labels, therefore it is perfectly natural that an unsupervised algorithm performs worse. This unavoidable drawback in unsupervised discriminators is offset by the fact that these tools are sensitive to a wide spectrum of anomalies. We would like to compare the two approaches quantitatively: a ROC curve for each GAN's dicriminator is drawn together with ROC curves for supervised classifiers in fig 2.15.

Our final goal is to integrate all the unsupervised discriminators trained on exponential-like and linear-like images in two different tools for anomaly detection. Trasparent ROC curves in those plots make clear that each $\mathcal{D}$ performs better in a specific region of the ROC space, indeed none of the curves lies above the others in the whole range of the plot. The aim is to find a receipt that allows to process each sample in the test set with all the 10 discriminators, but records only the best score. The best score, in this case, is the one that creates a better separation between the two pdfs in the classification. The perfect situation is when this algorithm automatically selects the smallest possible output within the ten given from a background image and the greatest from an anomalous one. The problem is that, since unsupervised learning must be performed, we do not know a priori data's labels while predicting them. Moreover, in principle, we can not take any decision looking at anomalous pdfs' shapes, because before the final assessment, training is

done only with background images.[4]

We investigate two possible strategies. The first one is to retain for each image in the test set the maximum score within the 10 provided. This option is motivated by the fact that discriminators should perform well on normal images producing a pdf sharply peaked around zero, because they have already experienced background data during training. In comparison discriminators' response to anomalous samples is less predictable. This idea is confirmed from the 20 plots presented, because the light blue shaded area is almost the same across all of them, unlike the green and red step curves. Then taking maximum values should shift right the anomalous pdf of the final discriminator enhancing separation between pdfs, the hope is that the normal one stands in its place almost unchanged after this operation. The ROC curve produced by this selection algorithm is shown in the figures below with point dashed lines. The second algorithm we propose consists in selecting the best discriminator from the group of ten, that has the lowest 90-th quantile of the background pdf. The underlying idea is that the discriminator with the background pdf closest to a delta function centred in zero can hardly overlap with the anomalous pdfs. In the ROC space's plots this curve is labelled as the "best quantile" line.

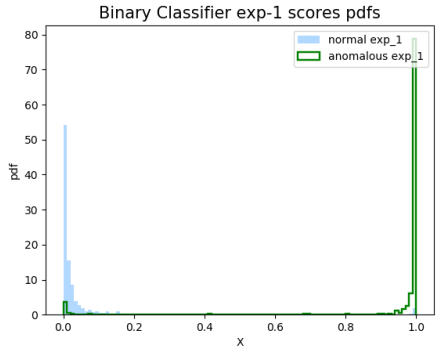
We compare performances of the unsupervised discriminators with four supervised binary classifiers trained on the following mixtures of images:

- exponential background images and class 1 (exponential plus peak in the tail);
- exponential background images and class 2 (exponential plus peak in the bulk);
- linear background images and class 3 (linear plus peak in the tail);
- linear background images and class 4 (linear plus peaks in both tail and bulk).

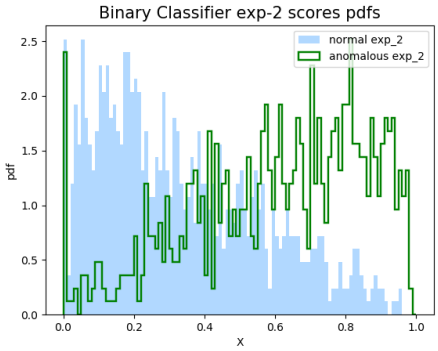
Classifiers' architecture and training methods are the same as those in the reduced problem in section 2.3. Fig. 2.14 shows the scores' pdfs built as histograms computed from slices of the $5k$ images test set. Plots prove that, except from $\mathcal{C}$ trained on exponential and class 2, binary classifiers achieve almost perfect separation between pdfs: background images are predicted as zeros, while anomalous ones are predicted as ones. Just a few images are mis-classified. Fig. 2.14b highlights that even a supervised classifier do not manage to tell perfectly apart exponential images and class 2 images. The light blue pdf has an heavy right tail, while the green function has a high peak placed at zero: there is an high probability that $\mathcal{C}$ predicts a score of zero for a class 2 image. This model is the most difficult to assess, probably a different architecture should be considered for this specific classifier in order to improve its performances.

Fig. 2.16 and fig. 2.15a provide a complete picture about anomaly detection of exponential-like images: plots *(d)-(e)-(f)* achieve the largest separations between pdfs, but, as we can see, the discrimination is mostly due to the correct prediction of class 1, while the red curve has a little contribution in the region to the right of the light blue area. Therefore class 2, as happend for the supervised classifier, is hardly told apart from background images, except in discriminators *(b)-(d)-(i)* that have a small enhancement just outside the light blue area. ROC curves prove that the GAN discriminator with the highest AUC succeds in overpowering the exponential-class 2 supervised discriminator, which, in fact, has poor discriminating powers. Maxed scores are better than the best GAN in the FPR range $[0.04, 0.2]$, after that they are almost the same, indeed their difference in AUC is only of 7‰. The best quantile curve instead is not effective in

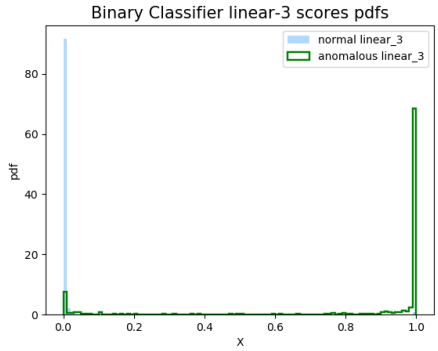
[4] In real life experiments, we can produce an arbitrary amount of background events and consequently of normal images following SM distributions from a MC generator. However, we can not specify the alternative BSM scenario, otherwise we would fall in the conventional supervised approach. Selection receipts can take advantage on the exact form of scores' pdf of background examples, but not on the shape of the one evaluated on the anomalies.



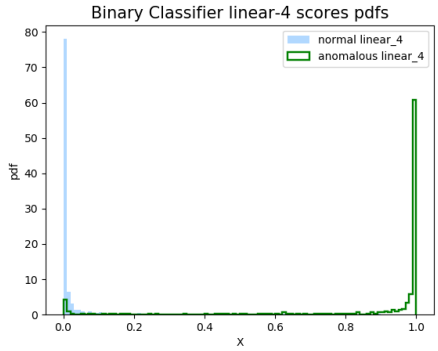
(a) $\mathcal{C}$ with linear and class 4 images



(b) $\mathcal{C}$ with linear and class 4 images



(c) $\mathcal{C}$ with linear and class 4 images



(d) $\mathcal{C}$ with linear and class 4 images

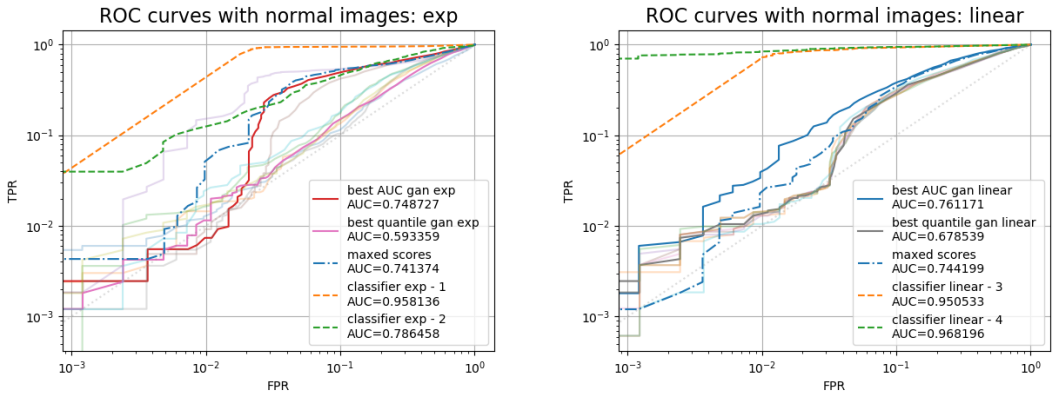
Figure 2.14: Supervised classifiers' pdfs of background and anomalous images. Almost perfect discriminations are achieved, except for the classifier trained over exponential and class 2 (exponential plus gaussian peak in the distriution's bulk) images. Panel *(b)* shows how even a supervise model struggles in this classification.

this case. Finally we note that the performance loss in employing an unsupervised tool in place of the supervised classifier (with respect to the exponential-class 1 classifier) is about of 21%.[5]

Within the plots in fig. 2.17 the best is without any doubt the a one. Also discriminator j provides good separation between the two pdfs, because the background distribution is really focused around zero with just few points in the tail. In the linear-like case discrimination is helped by class 4, that gives larger contributions outside the light blue area. This fact is probably due two the shape of the anomalous pdf of class 4 images, that has two gaussian peaks instead of one and makes calssification easier. Fig. 2.15b show that "max scores" strategy has an AUC parameter less than 2% the best GAN, this is probably because the best GAN performs far better than others, than "max" operation corresponds to consider just its outputs and neglect others. Performance loss in this case corresponds to a difference of AUC of 22% between maxed scores and the supervised classifier linear-class 4.

We note that the employed method might not be the most effective one. It is based on the

[5]Recall from the definition of Area Under Curve that this loss implies that we have 20% less probability that an anomalous image obtains an higher score from the detector than a background one.



(a) Exponential-like images

(b) Linear-like images

Figure 2.15: ROC curves.

idea that "in GAN training, best model for $\mathcal{G}$ implies also best model for $\mathcal{D}$" since they are trained together adversarially. This, in fact, is not necessarily true. A further improvement of the technique could be to select the best model for the generator and train the corresponding discriminator till optimality with $\mathcal{G}$ fixed. This trick would probably enhance performances in anomaly detection and give models that perform without big differences between each other. Moreover, if the pdfs on background and anomalous images were similar, they could be compared in order to build plots supported by statical analysis with mean values and error bars, as done in the previous section.

Plots of ROC spaces prove that for small values of FPR, the "max scores" strategy always underestimate the optimal GAN discriminator. Small values of FPR implies an high value of the threshold used to discriminate between the two classes in a binary problem: we are looking at right tails of distributions. There, the problem is that we could not have enough points to build correctly the histogram that estimates the pdf, a lot of bins remain empty and this methods does not give good results. Further improvements could be made enlarging test set in order to populate more the tails with rare events. In any case, our study assesses that the "max scores" strategy performs better than the one based on the least 90-th quantile. Maybe it could be combined with another ad hoc strategy that gives good results in low probability regions, but the reassuring fact is that in the bulk of the pdf the chosen receipt performs well.

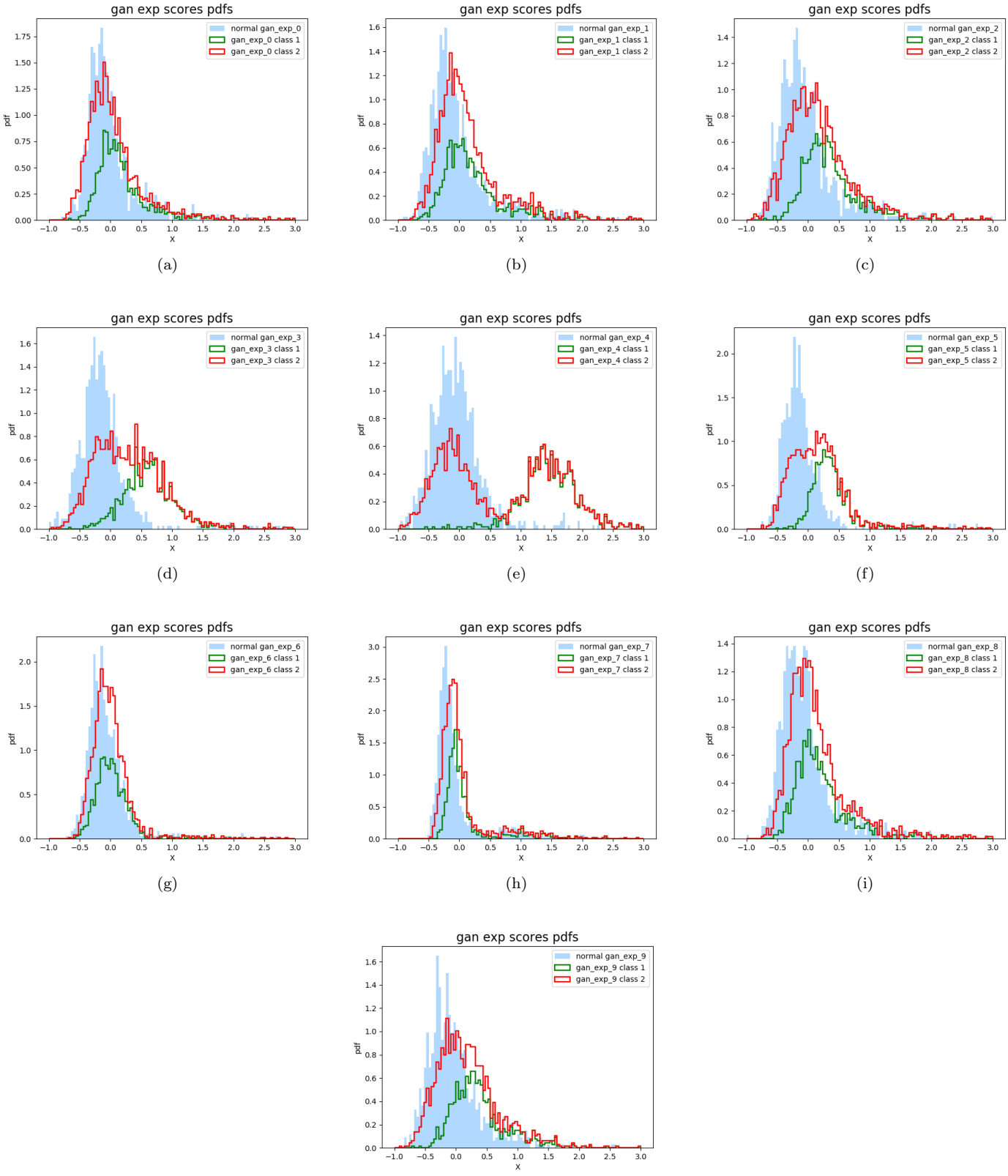


Figure 2.16: Pdfs of GAN discriminators' scores over exponential-like images. Red and green histogram are stacked to visualize the contribution of each class to the anomalous pdf. Class 1 is better separated than class 2.

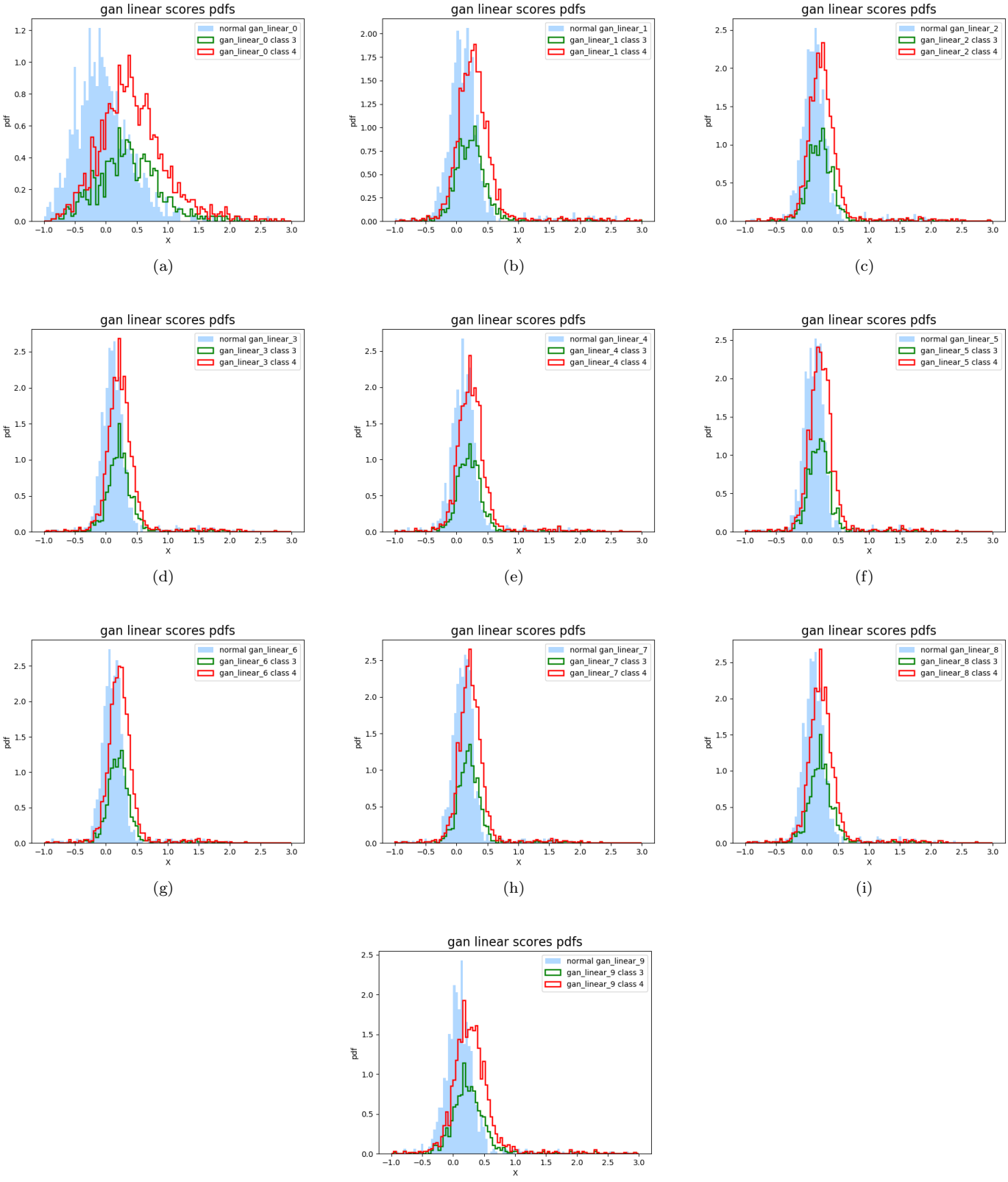


Figure 2.17: Pdfs of GAN discriminators' scores over linear-like images. Since class 4 has two peaks in its pdf, $\mathcal{D}$ has a better discriminative power for this particular anomaly.

Finally, we focus on evaluate further the quality of supervisely trained binary classifiers with data augmentation. Each of these models perform almost perfectly in its correspondent competence area, except the one trained with exponetial-class 2 images. We would like to cross test if they also have the ability to classify images from other classes. Fig. 2.18 presents the pdfs of each classifier evaluated on the test set's images.

- $\mathcal{C}$ trained with exponential-class 1 images identifies all linear-like images in the test set as aomalous, while the class 2 is mainly classified as background, but a non negligible amount of these images are predicted with score 1. We can not conclude that the model has good generalization properties, because seems that it is not sensitive to all gaussian or triangular anomalies regardless of where they are. Moreover it is strange that this discriminator classifies linear images as anomalous, since they do not present any peak in the pdf.

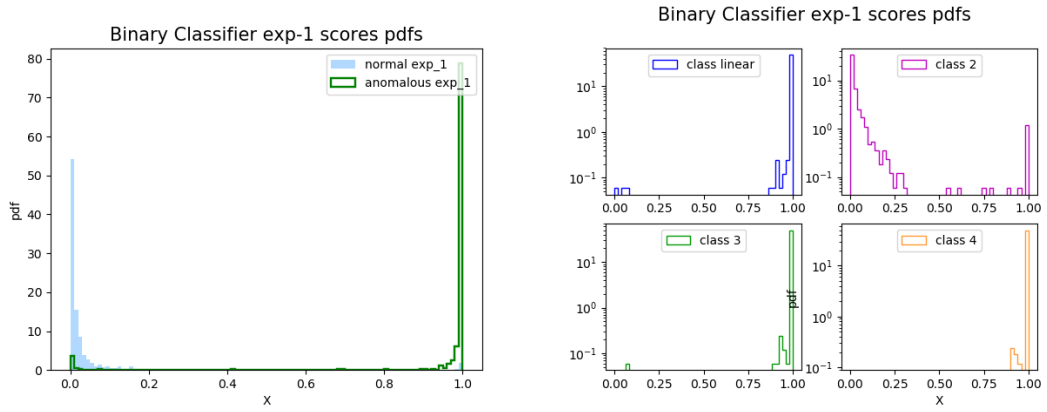
- Fig. 2.18b shows that the correspondent discriminator has difficulties in learning even its competence images, indeed class 1 images give an almost flat distribution: there is high probability to classify them as normal images. Linear-like classes instead are all almost the same, sharply peaked next to one.

- Linear-class 3 classifier's cross pdfs tell that the discriminator behave the same with exponential and class 2 images, this is understandable since they are really similar. Class 1 instead has a pdf concentrated under the threshold value of 0.5. Then all exponential-like images are told to be normal from this classifier. The most problematic issue in this cross checking is that the classification of class 4 resembles the behaviour of random guessing, probably caused by the fact that class 4 has the same peak of class 3 plus another peak in the left of the distribution that confuses the model.

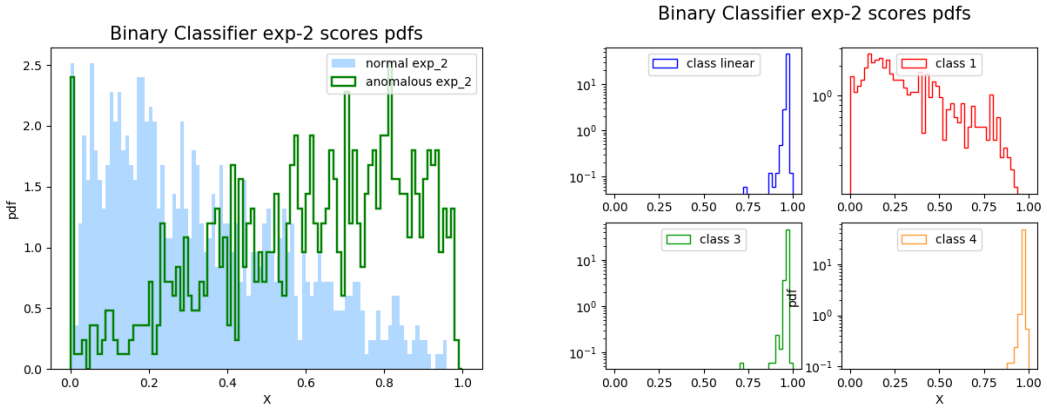
- The last classifier, namely the one trained over linear-class 4 images behaves the same as the previous one. The only difference consists in the presence of much more points from classes 1 and 2 that have scores above the 0.5 threshold.

We conclude that these models do not have the ability of generalizing to other regions of the test set. They can not be employed as anomaly detector of course, because they are strongly bounded on the functional forms of the two alternative pdfs they were trained on. This is an example of the situation pictured in the introduction, regarding the fact that supervised models can not be sensitive to signals of nature different from the one they are looking for. Nevertheless it is interesing to observe in the present case that exponential-class 1 and exponential-class 2 view linear-like examples as anomalies, whereas linear-class 3 and linear-class 4 predict exponential-like histograms as background. The motivation could be found in the fact that exponentials have thin tails while straight lines have thicker ones. In our datasets, anomalies introduce peaks in distributions, but in light of the cross check performed we might say that discriminators try to generalize to other images according to the thickness of the tails rather than on the spikes found.

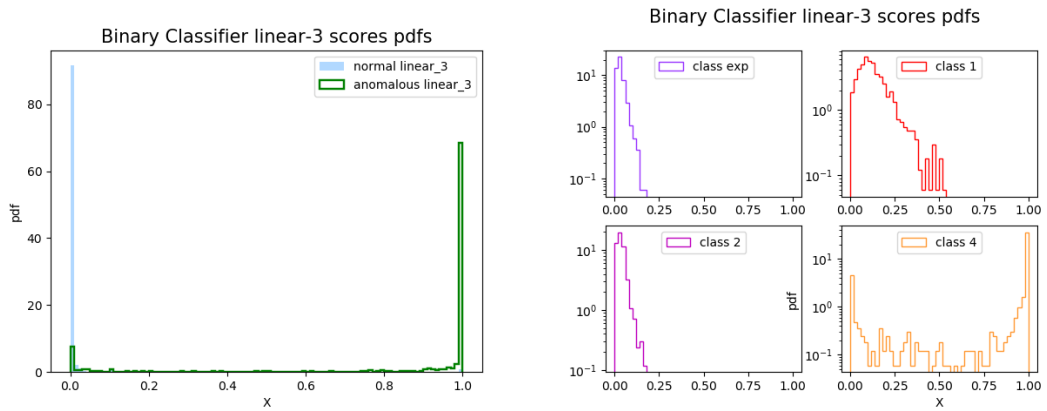
ROC space allow us to view how performances of a binary classifier vary depending on the threshold used to discriminate between classes. Fig. 2.19 show how the ROC curves change while augmenting the training set from 0 to 200%. The exponential-class 1 classifiers achieve optimal results until the FPR value of 0.02. Below 1% we have a resolution problem since we computed the background and anomalous pdfs with the histogram method with 100 bins. The AUC parameters get worse with augmentation, but the plot shows no correlation between p and AUC. Fig. 2.19b is the most interesting one because the case $p = 0$ has the worst AUC value. We measure an improvement in this parameter at most of 0.19 between $p = 0$ and $p = 20\%$ curves. This fact might imply that when the classification is not good without augmentation, the introduction of



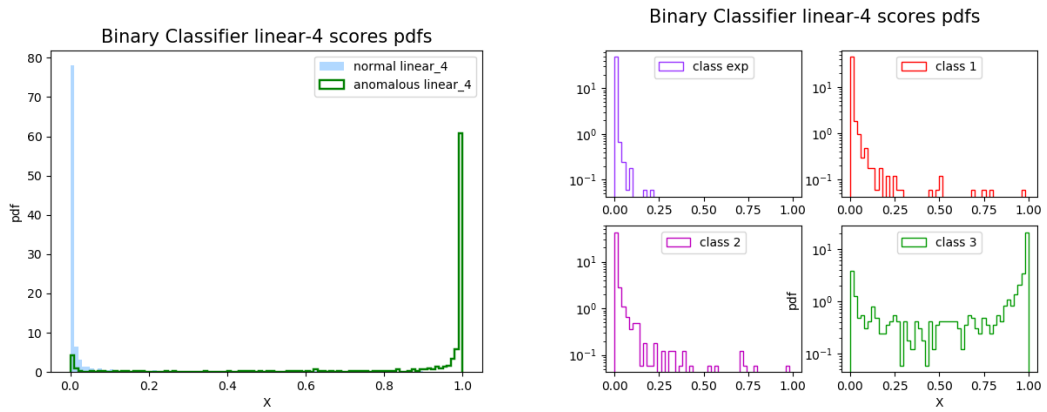
(a) $\mathcal{C}$ trained over exponential images and class 1



(b) $\mathcal{C}$ trained over exponential images and class 2

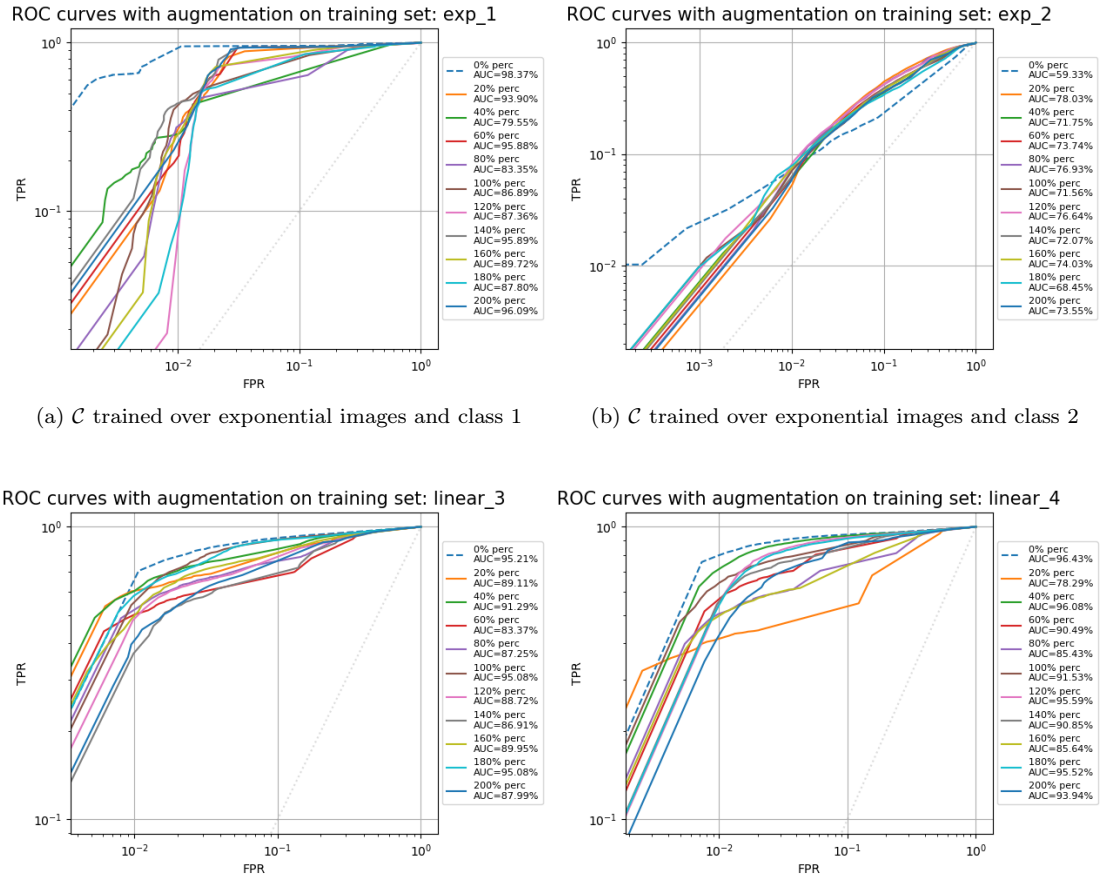


(c) $\mathcal{C}$ trained over linear images and class 3



(d) $\mathcal{C}$ trained over linear images and class 4

Figure 2.18: Cross tests of binary classifiers in and out their regimes of competence.



(a) $\mathcal{C}$ trained over exponential images and class 1

(b) $\mathcal{C}$ trained over exponential images and class 2

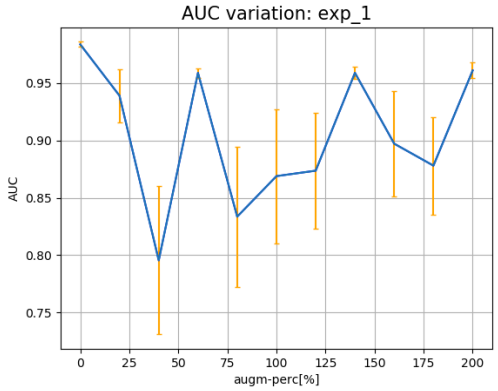
(c) $\mathcal{C}$ trained over linear images and class 3

(d) $\mathcal{C}$ trained over linear images and class 4

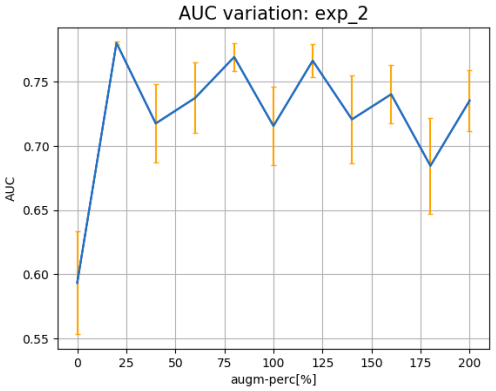
Figure 2.19: Area under curve - augmentation percentage

arificial images can help increasing performances, adding more separation between background and anomalous pdfs. The linear-class 3 and linear class-4 cases are perfectly aligned with this idea, because since their discriminations are good when $p = 0$, data augmentation do not help improving the AUC parameter.

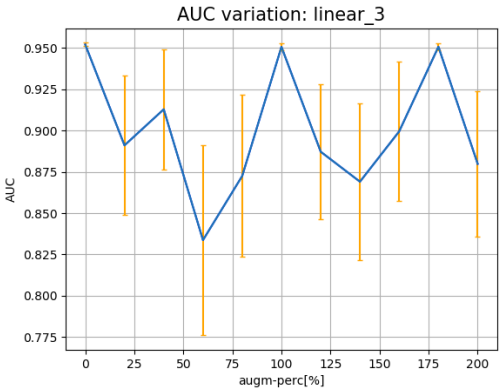
This hint is supported by plots in fig. 2.20 which prove that augmenting training sets in a situation of good classification, say with initial AUC above 95%, does not translate in improvements correlated with p. Otherwise in a bad initial situation, like in case of the exponential-class 2 classifier, our experiment shows that the introduction of artificial images in training set leads to an improvement in AUC parameter at most of about 19%.



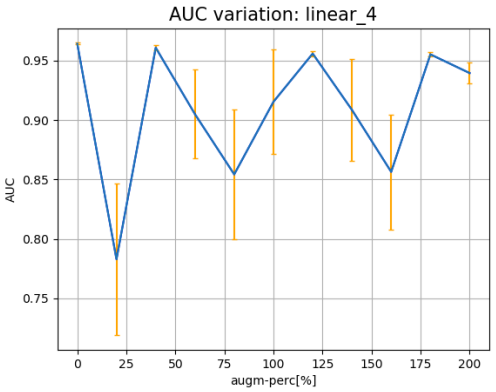
(a) $\mathcal{C}$ trained over exponential images and class 1



(b) $\mathcal{C}$ trained over exponential images and class 2



(c) $\mathcal{C}$ trained over linear images and class 3



(d) $\mathcal{C}$ trained over linear images and class 4

Figure 2.20: Area under curve - augmentation percentage

Chapter 3

Conclusions

In the present thesis we performed a feasibility study exploiting machine learning unsupervised training techniques in order to investigate anomalies in high energy physics. This preliminary study paved the way to future extensions in the context of real-life applications. We worked out several experiments on toy datasets, presenting two applications of the GAN system.

We studied how data augmentation can improve performances of a classifier either binary, either multiclass. Diagrams representing accuracy as a function of data augmentation percentage showed that our GAN can be employed, up to 50% of augmentation, as a consistent way to enhance supervised algorithms' performances. This procedure can be refined to increase this threshold value and, as explained in the introduction, to use GAN's generator as a support for MC simulations. This way, large amount of computational resources, that sample large datasets with MC generators, can be avoided. A well trained GAN generator can output a large amount of artificial data in far less time and with less storage memory needed. The issue would be to estimate the systematic error to which we expose ourselves with this method: GAN's training takes the generator to be an estimator of the data generating distribution, but always with a certain discrepancy between the real and the learnt pdfs; then we can sample a potentially infinite number of fake data that lead to zero random errors in computations, meanwhile introducing an increasingly more dominant systematical error. An efficient GAN training algorithm can then support a MC generator to improve performances of existing supervised methods, in addition to saving time and size-on-disk costs to study new experimental and theoretical models.

With the anomaly dection application we built an instrument that allow us to classify anomalies in a model independent way. This could help the scientfic community discovering new beyond standard model physics. Performance loss with respect to the supervised case is in the order of 20 percentage points in the area under curve parameter. Despite this fact, training several supervised models is not an effective way of searching for new physics in terms of training time and the possibility to miss signals present in data due to a wrong selection of the alternative hypothesis. We believe that a general question like "does data contain any kind of anomalies?" must be addressed with the most general possible strategy. Therefore the model independent approach is best suited to the problem. Once tension between the reference model and data is found, it will be possible to investigate further the sample with powerful specifical supervised methods, tailored on the dataset, in order to identify the correct model with which we represent the new physics.

Experiments in this thesis were made over a simple dataset of images representing one dimensional binned histograms. Nothing forbids to substitute this image approach with a dataset made of events of a random variable, instead of images of its probability distribution function.

Moreover GAN architecture's design can be improved and adapted to more complicated datasets, maybe based on MC generated data following realistic processes of SM and BSM physics. We judge experiments' outcomes as globally positive, meaning that the work done confirms that this research direction is a fertile ground for future developments. Some improvements were given throughout the exposition, while other issues remain still open questions, such as the choice of the best model in GAN training, either for the discriminator and the generator, or the best receipt that returns the best anomaly detection score within the pool of available ones. In machine learning literature new methods and algorithms are continously proposed and are always evolving, therefore future further improvements with respect to presented results are guaranteed.

Appendix A

Optimization Algorithms

Equation 1.6 is the core ingredient in the process of learning. Without Gradient Descent (GD) we would not be able to properly tune hundreds of thousand of parameters or even more. This algorithm is a simple idea that performs well in the case of a convex loss function. Unfortunately, when we deal with large nets, the landscape of the loss function is much more complicated and the naive GD or its Stochastic form do not ensure algorithm's convergence to a global minimum, unless perfectly tuned. This section presents two common and more improved optimization algorithms, that implement new ideas to cure some unstable behaviours of SGD.

A.1 RMSProp

SGD performs bad when it has to come across a path in the loss function landscape where gradients in one direction are greater than gradients in all other directions (like ravines). In fig. A.1 we draw this situation in a two dimensional problem. Ravines often occur around local minima and cause the algorithm to oscillate in the vertical direction, making only very little progresses in the horizontal one. A technique called momentum was invented in order to reduce oscillations and provide a stabler convergence to the minimum. As a result, stability of this method allows to set learning rates to larger values, speeding up the algorithm. The idea behind momentum is driven from classical point dynamics: if a ball is thrown down a hill, it accelerates increasing its momentum and going downhill faster and faster. When momentum is high, it is more difficult for the ball to make sharp turns in the wrong direction.

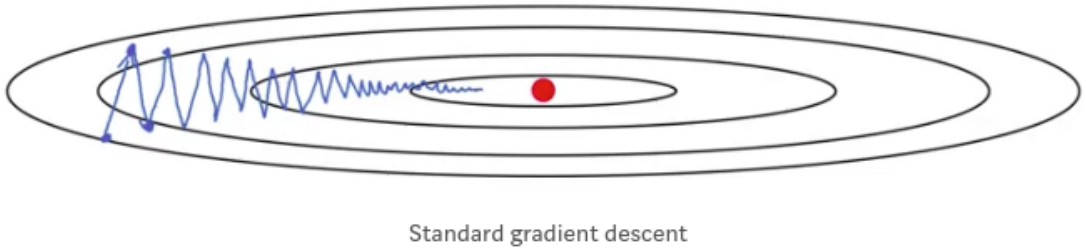


Figure A.1: Oscillations of the SGD algorithm around a local minimia. The red point is the minumum, gradients give great contributions in the vertical direction, resulting in slow convergence.

RMSProp optimizer implements these ideas. During the t-th step, it adds a fraction γ of the update vector of the past step to the current update vector:

$$\mathbf{v}(t) = \gamma \mathbf{v}(t-1) + \eta \nabla_\theta \mathcal{L} \tag{A.1a}$$

$$\theta \longleftarrow \theta - \mathbf{v}(t) \tag{A.1b}$$

where η is the learning rate as usual and γ is called momentum. Momentum term is usually set to a 0.9 or a similar value. This way the update vector $\mathbf{v}$ account for the sum of all the past collected gradients weighted with the exponential dumping parameter *gamma*:

$$\mathbf{v}(t) = \gamma^t v(0) + \eta \sum_{k=0}^{t} \gamma^{t-k} g_t \tag{A.2}$$

where g_t is a shorthand for the gradient w.r.t net's weights at step t, $\nabla_\theta \mathcal{L}(t)$. The problem with this method is that as more steps are taken in the same direction, $\mathbf{v}$ keeps increasing and when the minimum is reached its value is too high for the ball to be slowed down at the opitmum point.

Nesterov accelerated gradient (NAG) method try to face this problem, computing gradients of the loss function not in the current position θ, but approximately in the position where the ball will be after the update:

$$\mathbf{v}(t) = \gamma \mathbf{v}(t-1) + \eta \nabla_{\theta - \gamma \mathbf{v}(t-1)} \mathcal{L} \tag{A.3a}$$

$$\theta \longleftarrow \theta - \mathbf{v}(t) \tag{A.3b}$$

A.2 Adam

Different algorithms were proposed in ML literature after the introduction of RMSProp. Developement of optimization methods is an active research field, because new proposals always appear trying to gather all the benefits from previous methods while introducing others. In this sense adaptive moment (Adam) estimation optimizer is one of the newest algorithms in ML literature, proposed in [10]. It is a method that compute adaptive learning rates for each parameters, it stores an exponentially decaying average of past squared gradients v_t along with a moving average of past gradients m_t (similar to momentum). The algorithm is based on the following equations:

$$g_t \longleftarrow \nabla_\theta \mathcal{L} \tag{A.4a}$$

$$m_t \longleftarrow \beta_1 m_{t-1} + (1 - \beta_1) \cdot g_t \tag{A.4b}$$

$$v_t \longleftarrow \beta_2 v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{A.4c}$$

$$\hat{m}_t \longleftarrow m_t / (1 - \beta_1^t) \tag{A.4d}$$

$$\hat{v}_t \longleftarrow v_t / (1 - \beta_2^t) \tag{A.4e}$$

$$\theta_t \longleftarrow \theta_{t-1} - \eta \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) \tag{A.4f}$$

where parameters β_1 and β_2 control the exponential decay rate of gradients and are usually set to values 0.9 and 0.999 respectively. ϵ is a small regularizing factor suggested to be set to $1E-8$ to prevent division by zero. η is the learning rate. m_t and v_t are estimates of the first and the second raw moments, their unbiased forms $\hat{m}_t$ and $\hat{v}_t$ are used to update net's parameters. Initial values for m_0, and v_0 parameters are fixed to 0. In the ball example going downhill, Adam method can

be compared to a heavy ball with friction that then prefers flat minima in the error surface. Since this is an adaptive method, learning rate schedules are not needed, because updates' magnitude is authomatically adjusted according to the form of the loss function. Nonetheless we experienced in our experiments that while training a complex system like a GAN, reducing the learning rate can definitely help nets' optimization.

Appendix B

Statistical hypothesis testing

A statistical hypothesis test is a method of statistical inference. It is based on two hypothesis, respectively called null H_0 and alternative H_1. After observing the properties of a relevant population, we aim to accept or reject the null hypothesis at the expense or in favour of the alternative one. In order to carry out the decision process, we must select in advance the test statistic T and the significance level α we will employ. The former is a function of the observations, while the latter is a probability threshold below which the null hypothesis will be rejected (common values are 5% or 1%). Since we usually deal with empirical observations that have an intrisic probabilistic character (as we do in high energy physics with quantum mechanical observables), a function of the measurements, such as T, will be a random variable itself. Hence T follows a particular probability distribution function. Very often the two hypothesis are mutually exclusive statements about the nature of that distribution function.

In this section, to make it easier, we restrict the discussion to a one dimensional problem. We suppose that we made a set of measurements $\{x_i\}_{i=1}^n$ of a specific property of the sample in order to compute over them the observed value t_{obs} for the test statistic T. t_{obs} then follows two important conditional probability distributions f_0 and f_1 with the following partition functions:

$$F_0(t) = P(T < t|\mathrm{H}_0) = \int_{-\infty}^{t} f_0(x)\,dx \tag{B.1a}$$

$$F_1(t) = P(T < t|\mathrm{H}_1) = \int_{-\infty}^{t} f_1(x)\,dx \tag{B.1b}$$

This integrals tell the probability that T takes a value lower than t, once either the null or the alternative hypothesis is assumed to be true.

Fig. B.1 shows that fixing a significance level α sets a threshold vertical line that divides the plot in four regions (t is the $(1-\alpha)$-th quantile of the f_0 distribution: $t = F_0^{-1}(1-\alpha)$).

- $f_0 < t$: it is the probability of confirming the null hypothesis, $1-\alpha = P(\text{accept } H_0|H_0\text{is true})$
- $f_0 > t$: critical region, it tells the probability of making a type I error, $\alpha = P(\text{reject } H_0|H_0 \text{ is true})$
- $f_1 < t$: probability of making a type II error, $\beta = P(\text{accept } H_0|H_1\text{is true})$
- $f_1 > t$: defines the power of a test, namely the probability of correctly rejecting the null hypothesis, $1-\beta = P(\text{reject } H_0|H_1 \text{ is true})$

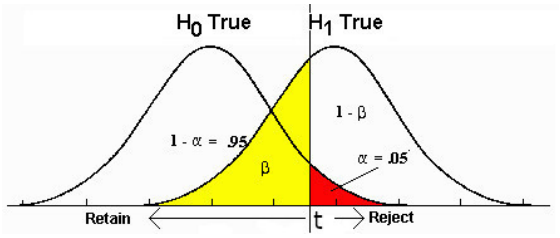


Figure B.1: Statistical Test with the threshold value t. The red region is called critical. The value $1 - \beta$ is the power of the test.

The final decision is made on the evaluation of a p-value, which is defined as the probability, under the null hypothesis, of sampling a test statistic at least as extreme as t_{obs}. Mathematically, p-value $= P(\mathrm{T} > t_{obs}|f_0)$. Since the significance α is defined as the probability of rejecting the null hypothesis when this is true, it is natural to decide of rejecting the null hypothesis when the condition p-value $< \alpha$ holds.

Bibliography

[1] Arjowsky, M., Chintala, S., Bottou, L. (2017) *Wasserstein GAN*, arXiv:1701.07875.

[2] Cerri, O., Nguyen, T. Q., Pierini, M., Spiropulu, M., Vlimant, J.R., (2019), *Variational Autoencoders for New Physics Mining at the Large Hadron Collider*, J. High Energ. Phys. (2019) 2019: 36, arXiv:1811.10276.

[3] Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. MIT Press.

[4] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y. (2014). *Generative Adversarial Nets*, Advance in neural information processin system: 2672-2680.

[5] Gulrajani, I., Ahmed, F., Arjovski, M., Dumoulin, V., Courville, A. (2017), *Improved Training of Wasserstein GANs*, arXiv:1704.00028.

[6] https://docs.scipy.org/doc/numpy/reference/routines.random.html

[7] https://en.wikipedia.org/wiki/Universal_approximation_theorem.

[8] https://www.tensorflow.org/versions/r1.10/api_docs/python/tf

[9] Ioffe, S., Szegedy, C., (2015), *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, arXiv:1502.03167.

[10] Kingma, D. P., Lei Ba, J., (2017), *Adam: a Method for Stochastic Optimization*, arXiv:1412.6980.

[11] Mao, X., Li, Q., Xie, H., Lau, R.Y.K., Wang, Z., Smolley, S.P. (2017) *Least Squares Generative Adversarial Networks*, arXiv:1611.04076.

[12] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

[13] Mockus, J., (1974). *The application of Bayesian methods for seeking the extremum*, Optimization Techniques: 400-404.