University of Turin

Department of Physics
Master's degree in Physics of Complex Systems

Master's Thesis

GAN Oversampling: an application of Generative Adversarial Networks to Imbalanced Datasets

Internal Supervisor:
Dr. André Panisson

Candidate:
Francesco Fanchin

External Supervisor:
Prof. Marco Wiering

Academic year 2017/2018

Abstract

Machine Learning has gained increasing importance in the last decade: class prediction in particular is critical for many real-world applications. Fraud analysis, cancer prediction and other more are remarkable examples of imbalanced problems: one class of the dataset under consideration contains too few istances. The usual Machine Learning algorithms, like neural networks, require a minimum amount of data to effectively learn the different classes: in an imbalanced situation, failure in minority class' prediction is observed.

In this thesis an algorithm for balancing minority class' size was developed (oversampling): the aim was to improve performances of a classifier. Generative Adversarial Network, a neural network for image generation, was used to artificially generate minority class' data. GAN was properly modified depending on the particular dataset's features. The overall framework was tested on MNIST Dataset for handwritten digits and various real-world datasets, from seismic bumps to cancer detection. In the end, a comparison with SMOTE package for imbalanced problems is provided.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In a constantly evolving and dinamic world, prediction plays an important role in many contexts: business, industry, financial markets, healthcare, scientific research, insurance, public security, sport and other more. Before the development of Machine Learning, this task was mainly accomplished by empirical and theoretical approaches: both of them lack of a full exploitation of past information. A data-driven approach became possible with the increasing storing capacity of computers. Data can be easily stored: after a suitable preprocessing, the algorithm choosen for a certain problem can be trained on the whole dataset. After training, the algorithm can perform a prediction, i.e. select a class for every incoming instance.

For example, given a dataset of people, everyone with their own features (sex, age, height,...) and a class (with cancer or without cancer), we would like to build an algorithm for cancer prediction. We can notice that a fully theoretical approach could rely on cancer anatomy, models on tumour spreading and so on... Clinical studies or personal experience of the doctor could define an empirical approach: here experience is taken into account, but the various information are not sistematically combined in a unique structure. This could lead to biases which overestimate the relevance of certain data against others. Back to cancer detection, let us apply the data-driven approach: we select the appropriate algorithm (Naive Bayes for example) and we apply it to the dataset. If the algorithm was correctly trained, we can now predict the presence of cancer in new patients.

The problem is the following: in an unbiased dataset of people, only few of them have a cancer. The algorithm has not enough examples to learn and correctly classify people with cancer: on the other side, there are lots of people without cancer and the algorithm will correctly predict this class of people in general. But it is obviously more important to correctly classify people with cancer than the others. In other words, when the task is the prediction of a minority class, the usual algorithms fails in its prediction.

The idea to overcome this problem is to artificially generate minority class data: dataset will be balanced in this way. To do so, a Generative Adversarial Network was built to provide oversampling. GAN is a Machine Learning algorithm involving two competitive neural networks: main applications include photorealistic image generation, text-to-image synthesis, resolution increasment and much more. This thesis involved the development of the GAN oversampling framework for imbalanced datasets. GAN was modified in order to artificially generate minority class data: a generic final classifier (MLP, Decision Trees, Random Forest...) will take as input the balanced dataset. Different types of GAN and final classifiers are experimentally studied with ad-hoc datasets.

Chapter 2 sums up a theoretical background of Machine Learning, from basic concepts to score metrics. Vanilla-GAN and the most remarkable variants in the state-of-the-art are described in chapter 3. In chapter 4 the algorithm of GAN oversampling is fully explained. The overall framework was validated on images (matrices) and records (vectors): precisely, on MNIST Dataset of handwritten digits (with two different types of GAN) and some real-world datasets, like seismic bumps and cancer detection. SMOTE is a scikit-learn package for imbalanced problems: a comparison with GAN oversampling is provided. Overall results and discussion about future developments are reported in the last chapters.

Chapter 2

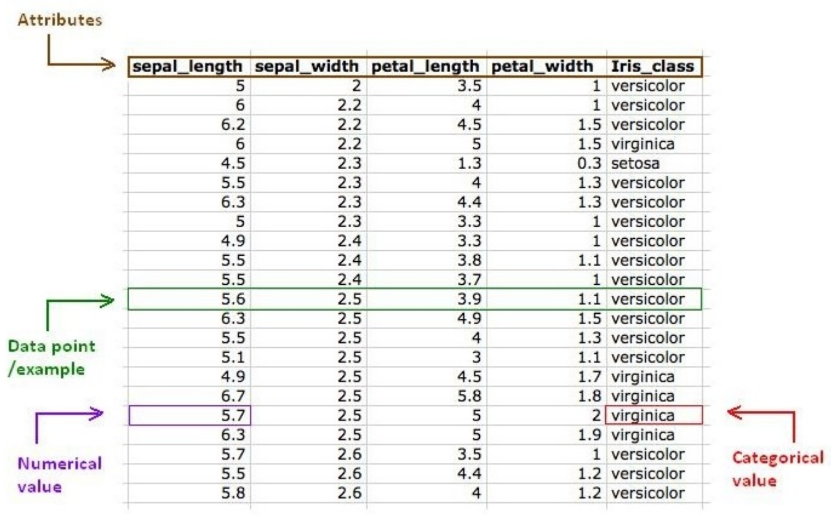
Theoretical background

Before going into details let us first define some general concepts: *Artificial Intelligence* (AI) is the attempt to mimic and reproduce all the skills of human intelligence, for example human speech, thinking, , learning, perception… *Machine Learning* is a sub-field of AI which deals with the ability of a computer system to learn from examples and apply this learning for prediction and classification mainly.[1]. In this chapter we will describe main properties and algorithms of Machine Learning.

2.1 Supervised learning

Supervised learning is a sub-field of Machine learning concerning classified examples.[2] Let us suppose to be a banker who accept or refuse credit proposal. From a history of proposals, we would like to authomatize this process, "feeding" a computer program with them. Given a dataset of people, everyone with his own features (sex, age, job, marital status…) and a class (accepted=1, refused=0), we search a function which takes a "person" as input (a datapoint) and gives us the belonging class as output. We only know some points of this function (the previous dataset).

This type of learning is a *classification problem*, because there is a discrete set of possible outputs. If the output is continous (e.g. the amount of money for the loan) we have a *regression problem* instead. From a geometrical perspective, a classification problem coincides with the problem of separation of datapoints in the space: every "cluster" will be represented by only one class. The regression problem, instead, consist in finding the best function which approximates the distribution of datapoints in the space, also known as fitting. Some SL algorithms like *neural networks*,[3] *perceptron*,[4] SVM[5] use a suitable optimization function to minimize. Some others use different approaches, like *Naive Bayes*[6] (Bayes Theorem) or *Decision Tree*[7] (ID3,C4.5,CART. . .)


Attributes
Data point /example
Numerical value
Categorical value

sepal_length	sepal_width	petal_length	petal_width	Iris_class
5	2	3.5	1	versicolor
6	2.2	4	1	versicolor
6.2	2.2	4.5	1.5	versicolor
6	2.2	5	1.5	virginica
4.5	2.3	1.3	0.3	setosa
5.5	2.3	4	1.3	versicolor
6.3	2.3	4.4	1.3	versicolor
5	2.3	3.3	1	versicolor
4.9	2.4	3.3	1	versicolor
5.5	2.4	3.8	1.1	versicolor
5.5	2.4	3.7	1	versicolor
5.6	2.5	3.9	1.1	versicolor
6.3	2.5	4.9	1.5	versicolor
5.5	2.5	4	1.3	versicolor
5.1	2.5	3	1.1	versicolor
4.9	2.5	4.5	1.7	virginica
6.7	2.5	5.8	1.8	virginica
5.7	2.5	5	2	virginica
6.3	2.5	5	1.9	virginica
5.7	2.6	3.5	1	versicolor
5.5	2.6	4.4	1.2	versicolor
5.8	2.6	4	1.2	versicolor

Figure 2.1: Iris classification, example of a labelled dataset.

2.2 Unsupervised Learning

In case of unlabelled dataset to be learnt, we are facing an *unsupervised learning* problem.[2] Analougously to the classification, we would like to separate our dataset without any class' suggestion: this task is called clustering. *k-means*, one of the most common UL algorithms, minimizes geometric distances from the central point of every cluster (centroid).[8] Other UL algorithms like GAN,[9] *Autoencoder*,[10] RBMs[11] are able to learn the probability distribution of datapoints: this can be used to generate new unseen data.

Complex phenomena are made up of various features and components interacting between them; it could be useful to know which are the most important features: some algorithms like SVD[12] or PCA[13] can provide a dimensionality reduction. SVD can be used for searching the so-called "latent variables", i.e. variables which explain the dataset distribution but are not included in the originary features.

2.3 Generative vs. Discriminative Model

The previous subdivision, UL versus SL, strictly depends on classes' availability. There is however another way to separate ML algorithms in two families: Generative and Discriminative models. A *generative model* learns to generate data according to a certain probability distribution: GANs, Autoencoders, RBMs are the most remarkable examples of GM. A *discriminative model* instead learns the conditional probability of a class given data, for classification purpose: MLP, CNN, knn,Decision Trees belong to this family.[14]

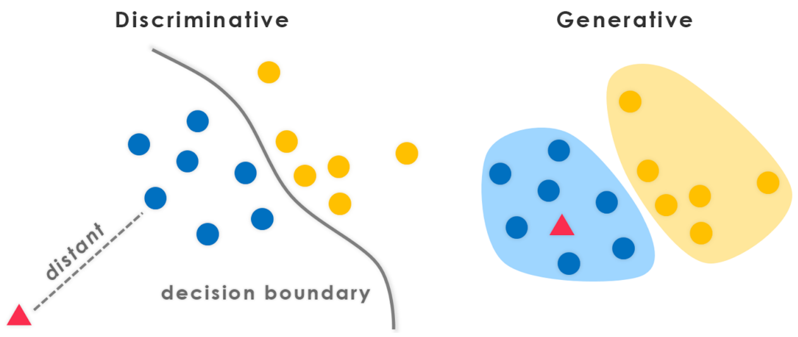


Figure 2.2: Difference between discriminative and generative models.

2.4 Training, validation and test sets

From what we have previously learnt, SL problems require a training set and an appropriate algorithm. This is not enough, in practical terms. We could ask wether some algorithms outperform the previous one and how the algorithm behaves with unseen data. To answer these questions, we need a *validation set* and a *test set*, so our dataset has to be splitted threefold.[2, 15] The former provides a *model selection*: once the algorithm is trained, we take every instance as input and we compare the output with its real class. The fraction of correct classifications gives us a score. We repeat the same procedure for all the candidate algorithms and we pick the one with the best score. The latter gives us an estimate of our accuracy when using the algorithm for classification. As in validation, we compute the fraction of correct classifications (accuracy) to indicate the algorithm's performance on that dataset. This last step is necessary because the algorithm has to be tested on unseen data. Validation set was not used in the training procedure but, due to model selection, a correlation do exists.

In order to prevent a possible bias due to a particular choice of the validation set, we can split our dataset in k folds (after leaving out the test set). We pick one fold for validation, and the remaining k-1 for training. After training, we obtain a score like in validation. We repeat the same for all the folds, and the final score will be the average of all scores. The overall procedure is called *cross-validation.*[16]

2.5 Useful metrics

We have mentioned before the term accuracy. Let us better clarify this concept and suppose to perform a binary classification (0 and 1 class values) for simplicity. The same metrics can be generalized to multiple classes.
There are four possible outcomes:
-false positives (FP)
-false negatives (FN)
-true positives (TP)
-true negatives (TN)
Considering the credit card approval example, FP could be people who would "deserve" credit and they don't instead, and so on...
Accuracy is defined as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

and measures how many times the algorithm guesses the correct classification. It's the most common and intuitive metric and other more refined metrics (like RMSE) basically derive from this. Thinking of our algorithm as a "detector" for bad borrowers, we can introduce other metrics:

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \end{aligned} \tag{2.2}$$

While the former answers to the question "how many selected events are relevant?", the latter answers to "how many relevant events are selected?"[17, 18] Other metrics are *false positive rate* (FPR) and *true positive rate* (TPR):

$$\begin{aligned} FPR &= \frac{FP}{FP + TN} \\ TPR &= \frac{TP}{TP + FN} \end{aligned} \tag{2.3}$$

We can notice that algorithms with the same accuracy may exhibit different behaviours: for example, a low recall could lead to a waste of money, accepting more borrowers than expected. This is something to count in during model selection: not all binary classifications are equal and it is important to deeply think about the consequences of all types of misclassifications.

All of these metrics need a "sharp" classification (0 or 1): this is equivalent to apply a 0.5 cut-off to a continous value. AUROC (*Area under Receiver Operating Characteristic*) instead is more general and considers all possible cut-offs.[19] First of all, a continous output between 0 and 1 is needed; ROC curve is a TPR-FPR plot with different cut-offs (from 0 to 1). A random classifier produces a straight line which intersects (0,0) and (1,1). A good classifier will generate a curve higher than the random one. AUROC is the integral of ROC curve: because AUROC of the random classifier is 0.5, a good classifier necessarily has to overcome this value.

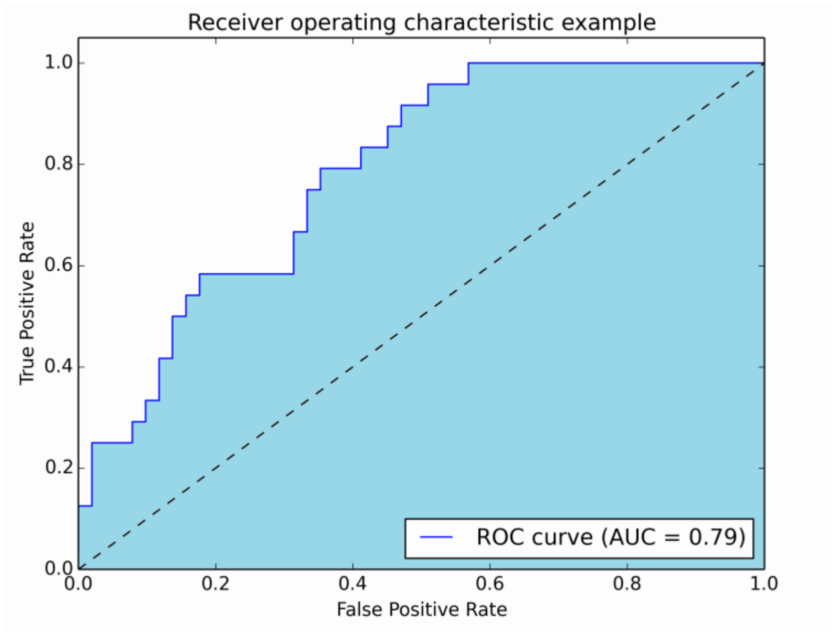


Figure 2.3: Example of ROC Curve, with AUROC=0.79; it evidently overcomes random classifier performances (0.5).

2.6 Neural Networks

Biological neural networks are a set of interconnected neurons, capable of complex tasks (the human ones especially) like perception, thinking, learning, memorization... A single neuron transmits information by electrical signals: the transmission occurs if it is excited by an external stimuli (pressure, variations in temperature, chemical signals from another neuron...) and if the stimuli itself overcomes a certain threshold. *Artificial neural networks* (ANN) try to reproduce a particular function of human brain: pattern recognition, belonging to the realm of supervised learning.[3] We are going to describe feedforward neural networks, where the output only depends from the input and not from previous outputs like in recurrent neural networks.

2.7 Perceptron

One of the first mathematical models of ANN was Rosenblatt's *perceptron* for binary classification: given a labelled (-1 or 1) dataset, perceptron linearly separates the two classes in the features' space.[4] Let $X = \{x_i\}, i = 1, ..., N$ be a vector representing a single istance of our dataset. The perceptron is a non-linear function which takes X as input and gives -1 or 1 as output:

$$f(x) = \begin{cases} 1 & if\ w \cdot X + b > 0 \\ -1 & otherwise \end{cases} \tag{2.4}$$

$w = \{w_i\}$ is a vector of parameters with the same dimension of X, and b is the bias: w determines boundaries' slope while b produces an offset in the final output.

If we redefine vector X as $X = \{x_i\}, i = 0, ..., N, x_0 = 1$ and w as $w = \{w_i\}, i = 0, ..., N, w_0 = b$, the previous function can be rewritten in a more convenient way:

$$f(x) = \begin{cases} 1 & if\ w \cdot X > 0 \\ -1 & otherwise \end{cases} \tag{2.5}$$

Given a dataset with many istances and two labels (+1 and -1), *perceptron learning algorithm* (PLA) aims to optimize w_i values, leading to the best possible separation of the two classes. If the problem is linearly separable, convergence to optimal values w_i^* is mathematically guaranteed; if not, the algorithm generally swings between some combinations of w_i. (figure linearly separable and not). PLA is made of two steps: training and weights update. Let y_{real} be the belonging class of our istance X and y_{out} the perceptron's output (so f(X) = y_{out}). The algorithm is described below:

The classification boundary is the hyperplane:

$$y = \sum_{i=0}^{N} (w_i \cdot x_i) \tag{2.6}$$

Algorithm 1 Perceptron Learning Algorithm

```
1: Initialize $w_i$ at random:
2: Compute $f(X) = sgn(w_i \cdot x_i)$ for all istances
3: while (misclassified examples in the dataset) do
4:     pick a misclassified example $X$: $y_{real} \neq y_{out}$ must be true
5:     adjust $w$ vector with the following rule: $w_{new} = w + y_{real} \cdot X$ (it can be proved considering one dimension at a time for the two classes)
6:     update $f(X)$ for all istances
7: end while
```

(For all points X in the hyperplane, $f(X) = 0$). What if we had a more complicated situations, where the two classes cannot be splitted by a simple hyperplane? The idea is to combine several perceptrons through many "layers".

Let us start from a single istance X. Instead of having one single output like before, we have now many outputs (neurons), everyone satisfying:

$$x_j = sgn(\sum_{i=0}^{N} w_{ij} \cdot x_i) \tag{2.7}$$

x_j neurons represent the first layer of our ANN (the hidden layer).Let us suppose to only have one hidden layer: we need a single output to build a classification function like before, so we can apply formula (...)

$$y_{out} = g(X) = sgn(\sum_{i=0}^{N} w_j \cdot x_j) \tag{2.8}$$

For example, with one hidden layer made by two neurons, it is possible to create AND and OR operators which connect two different perceptrons. The procedure can be similarly applied to several layers, shaping more complex boundaries. When network architecture is made of more than one layer, we generally use the term *Deep Learning.*[20] We have just built an MLP[21] (*Multi-layer perceptron*): following PLA, we know how to compute y_{out} but not how to optimize w_i^* parameters. The optimization process is not trivial and it will be explained in the next chapters.

2.8 Activation Functions

Let x be our input vector and w our vector of weights to optimize. We define s as $s = w^T x$. $sgn(s)$ is not the only possible activation function;

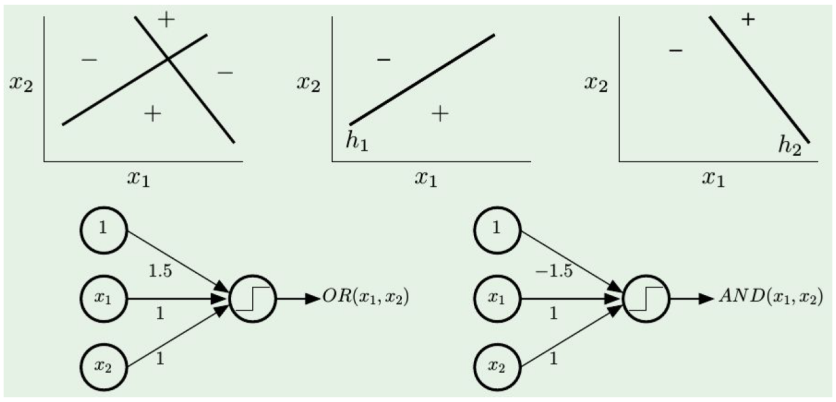


Figure 2.4: Two combined perceptrons can reproduce AND and OR operators with suitable weights.

the *logistic function* $\sigma(s)$, also called *sigmoid*, is a continous function which outputs between 0 and 1:

$$\sigma(s) = \frac{1}{1+e^{-s}} \tag{2.9}$$

Hyperbolic tangent instead outputs between 1 and -1:

$$tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} \tag{2.10}$$

ReLU (rectified linear unit) is an activation function defined as the positive part of its argument:

$$f(s) = s^+ = max(0, s) \tag{2.11}$$

2.9 Stochastic Gradient Descent and Backpropagation

Let us suppose to build a MLP with K layers and $w_{ij}^{(k)}$ parameters: index i counts the input neurons, j the output neurons, $1 \leq k \leq K$ the layers. Layer k has a number of neurons $d^{(k)}$; input neurons have an additional neuron (the bias) so, if we consider input layer $k-1$ and output layer k, $0 \leq i \leq d^{(k-1)}$ and $1 \leq j \leq d^{(k)}$. Generic input and output neurons are written as, respectively, $x_i^{(k)}$ and $x_j^{(k)}$. In order to simplify computations, the activation function is not $sgn(x)$ anymore but $tanh(x)$, which is differentiable.

Our neural network, to be useful, must correctly classify the largest number of istances. The idea behind the optimization process is the following:

we build a suitable function of parameters $w_{ij}^{(k)}$ representing the total misclassifications and try to minimize it; this will lead us to the optimal values $w_{ij}^{(k)*}$.
The *error function*, or *loss function*, is:

$$E = \sum_{n=1}^{N} (y_n^{real} - y_n^{out})^2 = \sum_{n=1}^{N} e(y_n^{real}, y_n^{out}) \tag{2.12}$$

with N the total number of istances. This function lives in the parameters' space $w_{ij}^{(k)}$.
The square guarantees the existence of a single global minimum. Notice that

$$y^{out} = x_j^{(K)} = tanh(\sum_{i=0}^{d^{(k-1)}} w_{ij}^{(K)} x_i^{(K-1)}) \tag{2.13}$$

We can define the internal argument as

$$s_j^{(k)} = \sum_{i=0}^{d^{(k-1)}} w_{ij}^{(k)} x_i^{(k-1)} \tag{2.14}$$

Because of the nonlinearity of y^{out}, setting the first derivative to zero and trying to solve the equation is not feasible: we need a numerical algorithm. *Gradient Descent* (GD) is one of the simplest numerical methods in convex optimization. We exploit a mathematical property of the gradient: its direction aims to the opposite of the steepest descent in that point. Hence, if we start from a certain point and we move to another point in the opposite direction to the gradient, we are going downhill: reiterating this procedure, we can reach global minimum.

A more powerful variant, called *Stochastic Gradient Descent*,[22] applies GD to $e(y_n^{real}, y_n^{out})$ term in Formula 2.12 . From our labelled dataset, one example (x_n, y_n) is picked at at time and $\nabla e(y_n^{real}, y_n^{out})$ is computed with respect to w weights. While in GD we follow $\nabla E(w)$ direction for weights update, in SGD we consider the average direction of all gradients $\nabla e(w)$:

$$\Delta w = -\frac{1}{N} \sum_{i=1}^{N} \nabla e(y_n^{real}, y_n^{out}) \tag{2.15}$$

However, in a neural network with many layers, the gradient's expression of $e(w)$ is very complicated: it is convenient to separate gradient's factors using Leibniz's rule. After some computations, we obtain an updating rule for weights:

$$w_{ij}^{k} \leftarrow w_{ij}^{(k)} - \eta x_i^{k-1} \delta_j^{(k)} \tag{2.16}$$

with η a number called learning rate. $\delta_j^{(k)}$ satifies the following rules:

$$\begin{aligned} \delta_1^{(K)} &= 1 - tanh(s_1^{(K)})^2 \\ \delta_i^{(k-1)} &= (1 - (x_i^{(k-1)})^2) \sum_{j=1}^{d(k)} w_{ij}^{(k)} \delta_j^{(k)} \end{aligned} \tag{2.17}$$

Hence, starting from the last, K^{th} layer, all "deltas" can be computed one after another (this is the reason for the name back-propagation) and weights' update becomes possible.

2.10 CNN

Convolutional neural networks (CNN) are a particular type of feedforward ANN where neurons are activated through a convolution between weights and previous layer. Mainly used in computer vision, they are able to identify patterns of different complexity (color, shape, rotation, even painting style...). Combination of CNNs are also used in image generation, e.g. in GANs, the main topic of this thesis. Let us start from a practical example: we need a computer system to detect cats. A sensor captures an image, (e.g. 256x256x3) and the program has to classify it correctly, like in the credit proposal's example. There is a huge difference however: now our algorithm,in the best case of a single layer, has to work in a (256x256x3)-dimensional space! In order to reduce it, the program should "extract" features which are relevant for the classification task, like a face, a body and four legs. We also notice that the same pattern (e.g. a face) might appear in different positions inside our image, activating completely different weights: the usual MLP would not take into account the correlation between close weights.
Given two matrices, a nxn "image" x_{ij} and a mxm "filter" f_{ab}, $m < n$, we can define the *discrete convolution* as:

$$s_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} f_{ab} x_{(i+a)(j+b)} \tag{2.18}$$

s_{ij} is the convolved matrix: every "pixel" of s_{ij} measures the similarity between the filter and the portion of image under convolution. The overall effect is a dimensionality reduction: output image is smaller than the input one.
There are some "degrees of freedom" (hyperparameters) which completely define convolution. *Filter size* is one of them; stride defines the filter's displacement over the image, e.g. one pixel horizontally and one pixel vertically.

Padding refers to borders' addition in the image (a column of zeros for example) in order to allow a certain stride. Back to our CNN, we build a

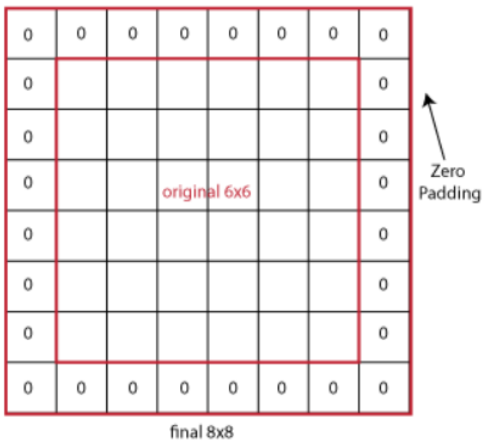


Figure 2.5: A 6x6 image becomes 8x8 with zero-padding.

convolutional layer through an activation function $\theta(s)$ (sigmoid, tanh,...). Starting from neurons' matrix $x_{ij}^{(k-1)}$ at layer $k-1$ we obtain the following output:

$$x_{ij}^{(k)} = \theta(s_{ij}^{(k-1)}) = \theta(\sum_{a=0}^{m-1}\sum_{b=0}^{m-1} f_{ab}x_{(i+a)(j+b)}^{(k-1)}) \tag{2.19}$$

The formula can be generalized to many dimensions, for example a 256x256x3 image has an additional dimension for RGB color.

Pooling is an operation which transforms size of an image by subsampling portions of the image itself. CNN hyperparameters (padding, stride,filter size) can be applied to pooling too. The two most relevant types of pooling are average and max pooling: in the first, the output is the average of all input's pixels; in the second, max value is picked instead. Typical Deep Learning architectures are made by a series of pooling and CNN layers.
Optimization strategy is the same in MLP: function E in Formula 2.12 has to be minimized with respect to filter weights f_{ab}. Gradient descent with backpropagation rule is used for weights' update: however the double sum in CNN layers complicates derivatives' computation. For CNN layers with

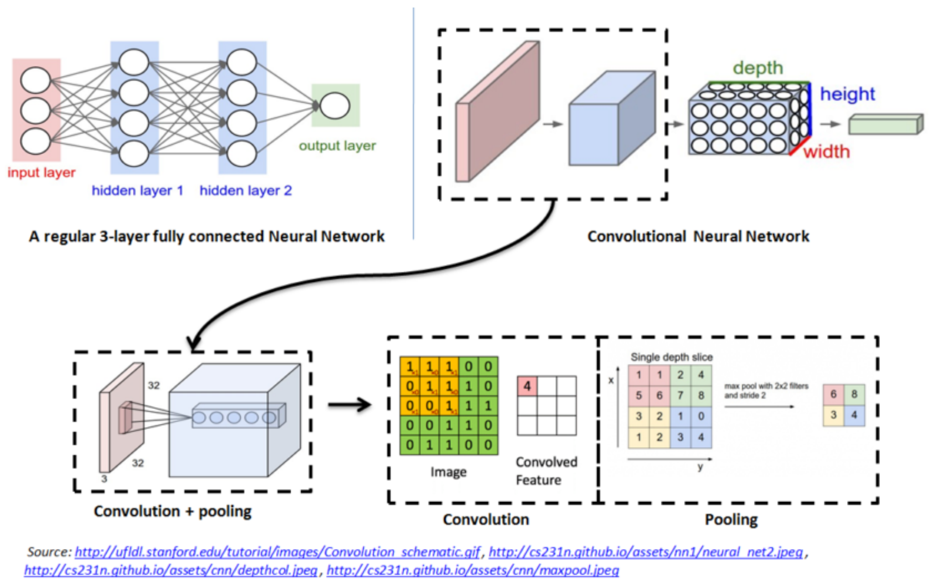

input layer
hidden layer 1
hidden layer 2
output layer
A regular 3-layer fully connected Neural Network
depth
height
width
Convolutional Neural Network
Convolution + pooling
Image
Convolved Feature
Convolution
Single depth slice
max pool with 2x2 filters and stride 2
Pooling
Source: http://ufldl.stanford.edu/tutorial/images/Convolution_schematic.gif, http://cs231n.github.io/assets/nn1/neural_net2.jpeg, http://cs231n.github.io/assets/cnn/depthcol.jpeg, http://cs231n.github.io/assets/cnn/maxpool.jpeg

Figure 2.6: Convolution and Pooling operations are visually represented.

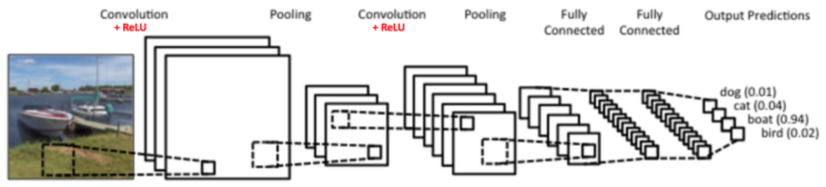


Figure 2.7: Architecture of a standard CNN for image classification.

zero-padding, filters weights are updated with the following formulas:

$$
\begin{aligned}
w_{ij}^{k} &\leftarrow w_{ij}^{(k)} - \eta \sum_{i=0}^{n-m} \sum_{j=0}^{n-m} \delta_{ij}^{(k)} x_{(i+a)(j+b)}^{(k-1)} \\
\delta_{ij}^{(k-1)} &= \theta'(s_{ij}^{(k-1)}) \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \delta_{(i+a)(j+b)}^{(k-1)} rot180(f)_{ab}
\end{aligned}
\tag{2.20}
$$

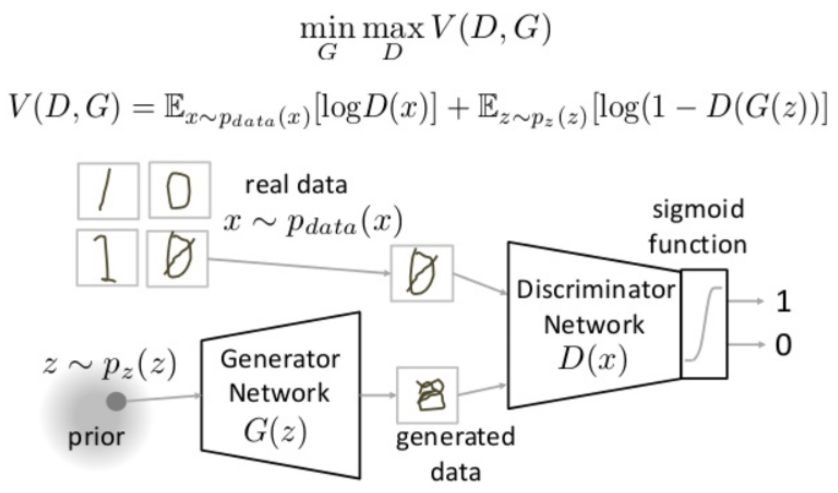
where $rot180(f)_{ab}$ is a flipped filter.

Chapter 3

Generative Adverarial Networks

3.1 Detailed algorithm

Generative Adversarial Networks (GAN), introduced by Ian Goodfellow in 2014, are an unsupervised learning algorithm for image generation.[9] The fundamental idea is a "competition" (a zero-sum game) between a generative and a discriminative model: but, implicitly, the algorithm was mainly thought for ANNs, where the optimization process is more straightforward and fits better to the backpropagation. We will simply refer to as *generator* and *discriminator* from now on. GAN's aim is to generate completely new images that, at a first glance, may appear real to a human person. In order to do so, a set of real images to mimic is needed. The generator is an ANN which takes as input a series of numbers in a certain dimension (e.g. random uniform numbers from -1 to 1) and gives as output an image. The discriminator instead takes an image as input and gives a number between 0 and 1 as output. We can think of generator and discriminator as, respectively, a counterfeiter and a policeman. Firstly, generator produces an image from a series of random numbers. Discriminator is trained with real images. Afterwards, discriminator will be fed with fake generator images, giving another output. Both nets have to be optimized: discriminator wants to correctly recognize real images, refusing the fake ones; generator wants to "cheat" the discriminator, i.e let it accepts fake images. After discriminator's optimization, its output will be passed into generator's optimization function, allowing a balanced competition between them. Mathematically speaking, generator is a differentiable function $G(z, \theta_g)$ (i.e. a MLP) which, starting from a prior $p_z(z)$ over input noise, maps on the data space and



$\min_G \max_D V(D,G)$
$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$
real data
$x \sim p_{data}(x)$
sigmoid function
Discriminator Network $D(x)$
1
0
$z \sim p_z(z)$
prior
Generator Network $G(z)$
generated data

Figure 3.1: GAN architecture and loss function.

learns a distribution p_g over data x. Discriminator $D(x, \theta)$ outputs a single scalar.
The two "actors" play the following mini-max game:

$$\underset{G}{min}\ \underset{D}{max}\ V(D,G) = \mathbb{E}_{x\in p_{data}(x)}[logD(x)] + \mathbb{E}_{z\in p_z(z)}[log(1 - D(G(z))] \quad (3.1)$$

precisely discriminator maximizes the real output and minimizes the fake one; generator instead aims to maximize the discriminator's output derived from generated images, so it minimizes $log(1 - D(G(z))$. log is necessary in order to avoid computational issues with very small numbers. The global optimum is reached when $p_g = p_{data}$. For fixed G, the optimal discriminator is:

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \quad (3.2)$$

Hence, $D^*_G(x) = 1/2$ in the global optimum.
It is proved that, $C(G) = \underset{D}{max}\ V(D,G)$ is equal to:

$$C(G) = -log(4) + KL(p_{data}||\frac{p_{data} + p_g}{2}) + KL(p_g||\frac{p_{data} + p_g}{2}) \quad (3.3)$$

where KL is the Kullback-Leibler divergence bwtween two distributions:

$$KL(p||q) = \sum_i p(i)log(\frac{p(i)}{q(i)}) \quad (3.4)$$

Formula 3.5 can be rewritten as:

$$C(G) = -log(4) + 2 \cdot JSD(p_{data}||p_g) \quad (3.5)$$

with JSD the Jensen-Shannon divergence.[23]
So, at the minimum of G, $p_g = p_{data}$ and $C^*(G) = -log(4)$. The whole algorithm relies on *minibatch gradient descent*,[1] a variant of the standard backpropagation.

Algorithm 2 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter.

```
1:  for number of training iterations do
2:      for k steps do
3:          sample minibatch of $m$ noise samples $(z_1, ..., z_m)$ from noise prior $p_z(z)$
4:          sample minibatch of m examples $(x_1, ..., x_n)$ from data generating distribution $p_{data}(x)$.
5:
6:          update the discriminator by ascending its stochastic gradient:
                $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1} m[logD(x_i) + log(1 - D(G(z))]$
7:      end for
8:      sample minibatch of $m$ noise samples $(z_1, ..., z_m)$ from noise prior $p_z(z)$
9:      update the generator by descending its stochastic gradient:
            $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1} m[log(1 - D(G(z_i))]$
10: end for
```

In the next sections we will take a look at the limits of vanilla-GAN; then, we will discuss about the state of-the art and main applications on adversarial techniques.

3.2 Problems of GAN

Goodfellow's publication was positively received from AI community: however, many problems arose. First of all, the whole algorithm is a minimax

[1]Mini-batch gradient descent performs weights' update for every mini-batch of m training examples.[24]

game, i.e. a function is minimized by an actor and maximized by another one, in a mathematical tug of war. Single optimizations are old well-known problems: many theoretical and numerical methods were developed, e.g. gradient descent, previously encountered in backpropagation algorithm. Minimax optimization instead represents nowadays an open problem: this is the main reason of difficulties in GAN's training.

GAN results are in general not very stable: a change in random seed could lead to completely different results, some fails probably. In addition, GAN's training is more computationally expensive than other neural networks, because two networks have to be optimized with alternate steps. *Mode collapse* is another problem affecting generator: when input data is made of multiple "modes", i.e. can be divided in distinct clusters, generator's output tends to fix in one of them; if the single mode cheats the discriminator, generator is not encouraged to reproduce all of them.[25] Discrete datasets (text especially)

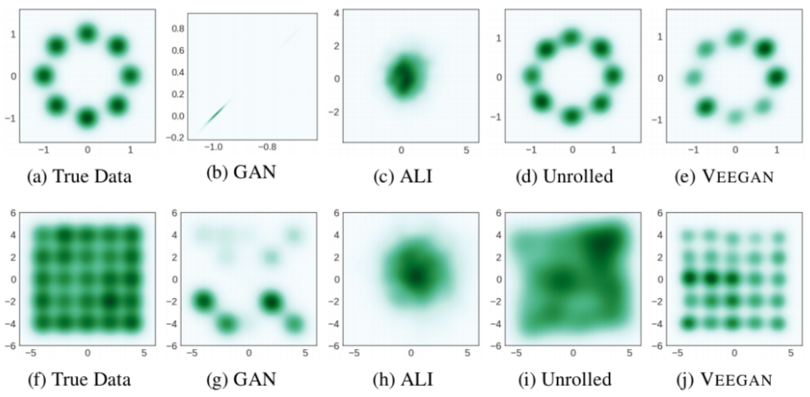


Figure 3.2: Mode collapse in various GAN architectures, with two types of true data. While Vanilla-GAN and ALI seem to suffer mode collapse, Unrolled and VEEGAN can reproduce the whole dataset.

are harder to reproduce with a certain amount of variety, because there is no direct formula for gradient descent and possible generator's outputs are strongly limited.[26]

3.3 State-of-the art and main applications

After Goodfellow's publication, several researchers tried to improve GAN's architecture. CGAN, developed by M. Mirza and S. Osindero, conditions generator's output and discriminator's input with additional information.[27]

The optimization formula slightly differs from Goodfellow's:

$$\underset{G}{min}\ \underset{D}{max} V(D,G) = \mathbb{E}_{x\in p_{data}(x)}[logD(x|y)] + \mathbb{E}_{z\in p_z(z)}[log(1-D(G(z|y))] \tag{3.6}$$

$D(x|y)$ and $G(z|y)$ are now conditioned by vector y: in the MNIST case, for example, y is a one-hot vector encoding digits from 0 to 9. DCGAN instead maintains the original optimization function, while CNN and pooling layers are inserted in discriminator and generator.[28] *Wasserstein GAN* (WGAN) uses a different loss function, based on Wasserstein, or "Earth mover" distance:

$$W(p,q) = \underset{\gamma\in\Pi(p,q)}{inf}\ \mathbb{E}_{(x,y)\sim\gamma}[||x-y||] \tag{3.7}$$

where $\Pi(p,q)$ indicates the set of all joint distributions $\gamma(x,y)$ whose marginals are respectively p and q.[29] This type of GAN is less affected by mode collapse than baseline model. In Figure 3.2 various GAN architectures try to reproduce "clustered" datasets: ALI (*Adversarially Learned Inference*) is an inference model which jointly learns a generation network and an inference network using an adversarial process.[30] In this case, generator is divided in

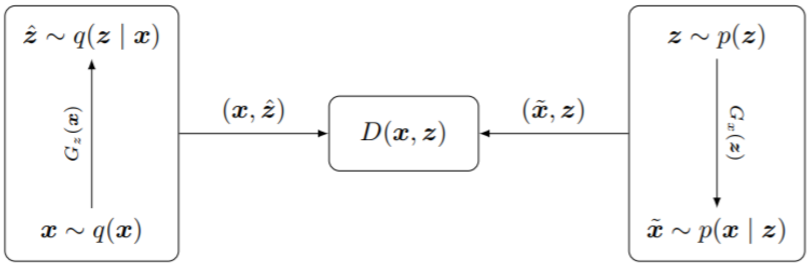

$\hat{\boldsymbol{z}} \sim q(\boldsymbol{z} \mid \boldsymbol{x})$
$G_z(\boldsymbol{x})$
$\boldsymbol{x} \sim q(\boldsymbol{x})$
$(\boldsymbol{x}, \hat{\boldsymbol{z}})$
$D(\boldsymbol{x}, \boldsymbol{z})$
$(\tilde{\boldsymbol{x}}, \boldsymbol{z})$
$\boldsymbol{z} \sim p(\boldsymbol{z})$
$G_x(\boldsymbol{z})$
$\tilde{\boldsymbol{x}} \sim p(\boldsymbol{x} \mid \boldsymbol{z})$

Figure 3.3: ALI architecure: encoder (left), discriminator (center), decoder (right).

two components: an *encoder* which maps data samples x to z-space, and the *decoder*, which maps samples from the prior $p(z)$ to the input space. Discriminator is trained with pairs (x,z) from the encoder and decoder (Figure 3.3). ALI suffers mode collapse, as we can see in Figure 3.2.

Unrolled and VEEGAN instead are able to reproduce the whole dataset. *Unrolled GAN* performs an "unrolled" optimization of the discriminator, i.e. discriminator goes on SGD for a certain number of steps: the loss function takes as input generator's weights and discriminator's weights after a number K of SGD updates(Figure 3.4).[31]

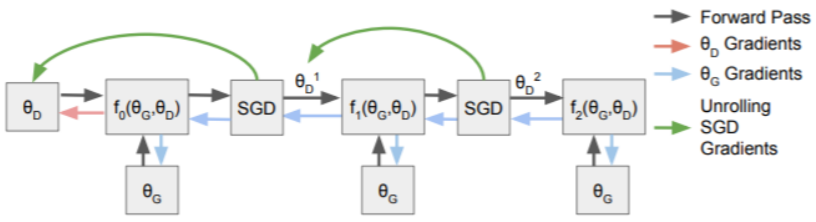


Figure 3.4: Optimization steps for Unrolled GAN with K=2. Discriminator is updated with two SGD steps forward than discriminator.

VEEGAN (Variational Encoder Enhancement to Generative Adversarial Network) introduces another network F_θ called *reconstruction network* which is learned to map data samples to a Gaussian and to approximately invert the generator network G_γ.[32] Here the generator's input prior $p_0(z)$ is a Gaussian. The idea is the following: if F_θ is trained to invert G_γ (Figure 3.5, on the left), $F_\theta \circ G_\gamma$(z) has a Gaussian distribution. If G_γ suffers mode collapse, function F_θ applied to data sample $p(x)$ is likely to produce a non-Gaussian distribution for the presence of "forgotten" modes: in this way, mode collapse is easily detected. Conversely, when F_θ is trained to map true data $p(x)$ to a Gaussian distribution (Figure 3.5, on the right), if G_γ mode collapses, i.e. generates only a part of true data distribution, F_θ applied to G_γ will probably not lead to a Gaussian for the missing modes: hence, F_θ will not map back to the original z. The optimization function counts in these two complementary aspects.

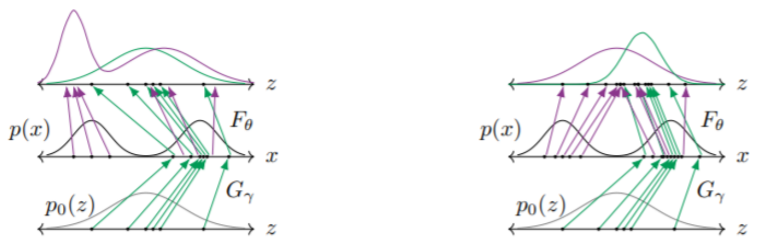


Figure 3.5: Behaviour of reconstructor function F_θ when trained to map data samples to a Gaussian and to invert the generator network G_γ.

GAN's applications are related to images. The simpler is images generation, like handwritten digits (MNIST Dataset, in the next chapter). Another interesting application is resolution increasment, with SRGAN: given pairs of the same image in high and low resolution (LRI and HRI), the aim is to create a generator which takes as input a LRI and gives as output a super-resolution image (SRI); discriminator has to separate SRI from LRI.[33]

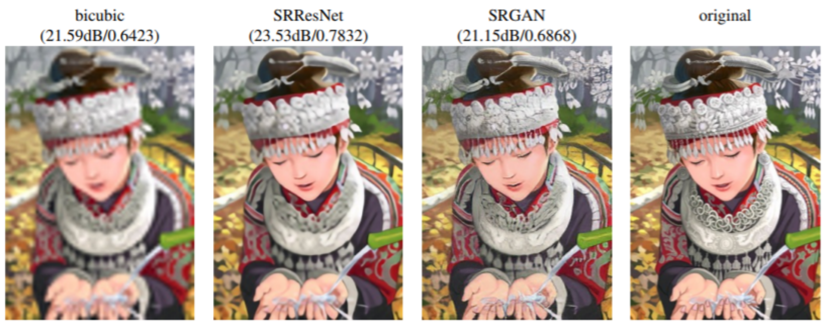


Figure 3.6: SRGAN for resolution increasment.

Text to image synthesis can be performed by a CGAN with CNN layers: text conditions generator's output by a procedure of vector embedding (a projection into a lower dimensional space).Discriminator is trained with pairs of images and the correspondent descriptive text.[34] An advancement in

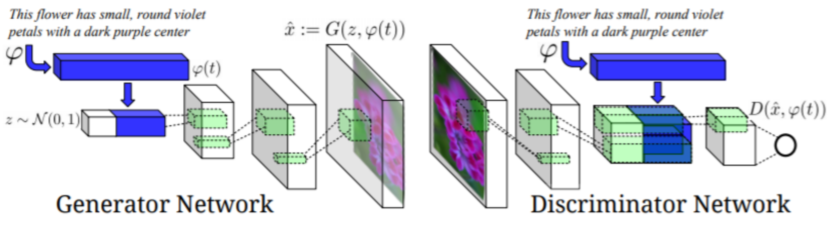

This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
Generator Network
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Discriminator Network

Figure 3.7: CGAN architecture for text to image synthesis.

images generation involves the separation between style and structure in an image: S2GAN combines two DCGANs, *Style-GAN* for textures and *Structure-GAN* for image geometry. This architecture is useful for applying different styles to images.[35]

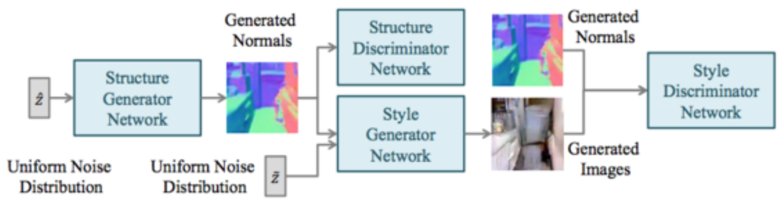

Generated Normals
Structure Discriminator Network
Generated Normals
$\hat{z}$
Structure Generator Network
Style Discriminator Network
Style Generator Network
Uniform Noise Distribution
Uniform Noise Distribution
$\tilde{z}$
Generated Images

Figure 3.8: S2GAN architecture for style and structure generation.

Chapter 4

GAN Oversampling

4.1 The problem of imbalanced datasets

From theoretical background chapter we have discovered binary classification task in supervided learning. In imbalanced datasets, almost all the instances are labelled as one class: traditional classifiers, seeking a good accuracy, fail in minority class prediction.[36] These types of problems are critical for many real-world applications like fraud detection, text classification, cancer detection, customers churn and much more. Metrics defined in chapter 2 are useful to describe the phenomenon: in such situations, accuracy reaches high values generally; other metrics like recall or AUROC exhibit low values instead.[36, 37]

The problem is the following: the classifier learns well the majority class, so high values of accuracy are observed. On the contrary, minority class' scarcity hampers a proper training: recall gets lower because represents the fraction of misclassifications in minority class. AUROC will generally show poor performances with values between 0.5 and 0.6 (0.5 is the performance of a random classifier).

4.2 The idea of oversampling

How to overcome the abovementioned problem? The idea is to artificially generate minority class' data (MCD): in this way, the whole dataset becomes balanced and the final classifier could better detect both classes, MCD especially. This is where Generative Adversarial Network comes in: as a generative model, GAN can learn to generate brand new data which resembles the original one.[1] After training with MCD, GAN performs an oversampling of the same class. However, GAN requires a certain number of istances to be trained. A sampling with replacement of the minority class is carried out: this "bootstrap" sample will feed our GAN. The algorithm of *GAN Oversampling* is described below. This framework is irrespective of the particular

Algorithm 3 Algorithm of GAN oversampling for imbalanced datasets.

```
1: extract minority class data (MCD)
2: perform MCD sampling with replacement
3: cross-validation for model selection
4: feed the GAN with the bootstrap sample
5: GAN training/optimization
6: create new MCD with GAN's Generator
7: New data and the original ones, now balanced together, will be used for
   classification, for example by an MLP
```

dataset and GAN architecture: from images to records, from vanilla-GAN to the most recent variants. Other specific operations, like feature's elimination, maybe strictly necessary or not and are not included in the list: they will be discussed in a practical analysis afterwards. In the next subsection, main datasets for algorithm's validation are described.

4.3 Datasets

Our framework will be validated on both images and records. *MNIST Dataset of Handwitten Digits* is a set of 70,000 images, divided in a training set of 60,000 examples and a test set of 10,000 examples. All digits 0-9 are equally represented in the dataset. Every image is a 28x28 matrix of rational values between 0 and 1 - the grey level. GAN and DCGAN will be tested on a subset made by two selected digits, e.g. 0 and 1. Given a certain final classifer (e.g. Random Forests), a comparison between the standard approach (classification only) and GAN/DCGAN oversampling will be performed.

UCI Machine Learning Repository is an online resource containing several datasets for machine learning purposes. Maintained by University of California, Irvine, some of their datasets were used in this thesis as a benchmark for GAN oversampling. All of them have two classes: zero and one. Just for one dataset, *"Seismic Bump"*, the whole process of data analysis will be fully explained. The table below sums up the UCI datasets:

[1]Other generative models could accomplish this task in lines of principle; this thesis is focused on GAN.

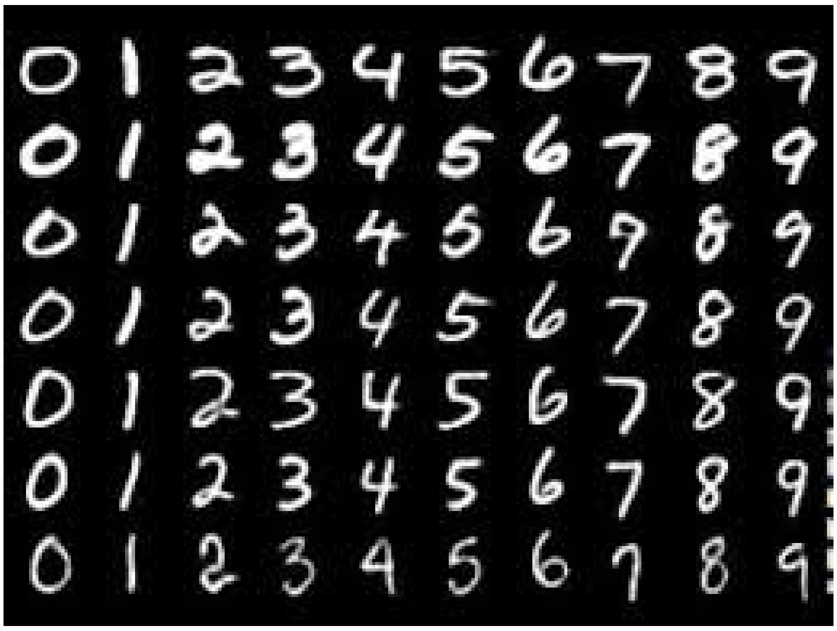


Figure 4.1: An overview of MNIST Dataset.

4.4 GAN oversampling for MNIST Dataset

Two types of GAN will be tested on MNIST Dataset: vanilla-GAN and DCGAN. Both architectures are shown below: vanilla-GAN has only one hidden layer for generator and discriminator. *ReLU* is used as activation function for hidden layers, while *sigmoid* for final output. The use of sigmoid has different meanings in generator and discriminator: in the first, the output is an image; in order to reproduce MNIST ones, output is between 0 and 1. Discriminator instead outputs a single number, between 0 and 1, because it represents the chance of taking a real image.

Both hidden layers need to be cross-validated with respect to number of neurons. However, this procedure is implicitly thought for discriminative models: the score function generally is accuracy, recall or AUROC in order to measure model's performance. For generative models, we search a function that evaluates similarity between real and generated data distribution. *Kullback-Leibler*, previously encountered in Formula 3.4 is a function which takes as input two distributions: it can be thought as a " distance" between distributions (although it is not symmetric). Hence, for every model (i.e. for every combination of neurons in discriminator and generator's hidden layer), we generate brand new data using generator and we compute, together with real data, K-L Divergence. The model with the lowest K-L will be picked.

	sector	# feat.	# istan.	imbal. ratio	# discr. feat.
Banknote	B	4	1371	0.44	0
Blood Transfusion	L	4	747	0.24	0
Default Credit Card	B	23	30000	0.22	3
German Credit Card	B	24	999	0.30	21
Heart Statlog	L	13	269	0.44	8
Seismic Bump	P	18	2584	0.07	12
Haberman's Survival	L	3	305	0.27	0

Table 4.1: UCI datasets used in this thesis. There are three different sectors (B=Business, L=Life Sciences,P=Physical Sciences.). Number of features, number of istances, imbalancement ratio and number of discrete features are shown in the other columns.

DCGAN's architecture is more complicated instead. Generator gradually de-

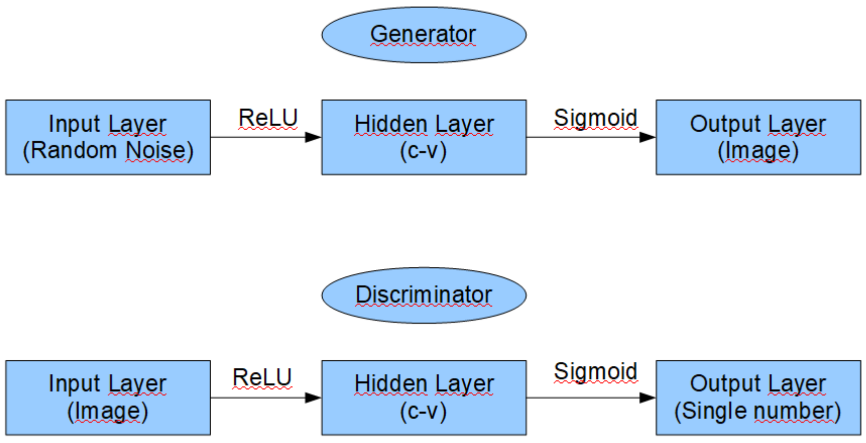


Figure 4.2: Vanilla-GAN architecture.

creases number of neurons until Deconvolutional Layer:[2] *LeakyReLUs* are used for three internal layers and *tanh* for final output. Discriminator starts with a convolutional layer and, after two dense layers, reaches a single number. LeakyReLUs are internally used too, but sigmoid give us final output as in vanilla-GAN. For both discriminator and generator, one hidden layer is cross-validated like before. Oversampling with DCGAN and vanilla-GAN will be tested against the simple classification, without oversampling. The

[2] Deconvolutional neural networks, unlike CNN, perform a transposed convolution in order to output a bigger image. See [38] for more details.

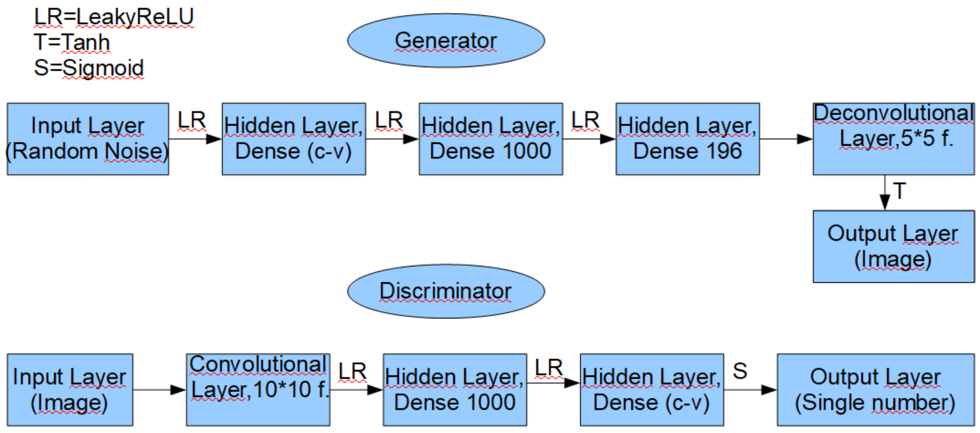


Figure 4.3: DCGAN architecture.

common classifier is a MLP. The dataset is a subset of MNIST Dataset: all zero-digits plus a variable amount of one-digits. The last will be our minority class. In Results chapter, the first plot will compare, for different imbalancement ratios, vanilla-GAN oversampling over a single classification, without oversampling. The second will do the same with DCGAN.

4.5 Seismic Bump: a practical GAN oversampling's application

In this subsection we analyze Seismic Bump Dataset and apply the framework of GAN oversampling, compared to the simple classification. Looking at the previous table, this dataset is made by 2584 total istances, whose 170 are labelled as one (imbalancement ratio = 0.07). There are eighteen features: twelve of them are discrete. A features' exploration reveals lack of variability in some features: mathematically speaking, their distributions exhibit a low variance. Some algorithms and procedures, like PCA and scaling, are unstable in case of low variance's distributions; furthermore, a feature with low variance does not add any relevant information, increasing computational cost for training. For these reasons, it is convenient to erase four features: now we have fourteen features, eight of them discrete. *Feature scaling* avoids dataset's fluctuations along any axis. A component x of a generic instance is transformed via the following formula:

$$z = \frac{x - \mu}{\sigma} \tag{4.1}$$

with μ and σ mean and standard deviation of the correspondant feature's distribution. The new distribution has mean $\mu = 0$ and standard deviation $\sigma = 1$. *Normalization* instead maps x inside range $[0, 1]$:

$$z = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.2}$$

with x_{max} and x_{min} the maximum and minimum value of feature's distribution. Between these two types of preprocessing, the second was applied because GAN's generator outputs between 0 and 1 too. Train/test split was performed with a previous shuffle. A *random seed* was set in order to completely reproduce all random numbers generations. As in the previous subsection, vanilla-GAN was optimized via cross-validation with K-L Divergence: in this way we obtain the optimal number of neurons in hidden layers. A sampling with replacement of the training set is carried out: GAN generally requires lots of istances for convergence. The new "bootstrapped" set contains 10000 istances; training can start. Generator takes as input a series of random numbers: discriminator is fed with real and generated data instead. The path for convergence goes on with alternate steps of generator and discriminator's optimizations, as described in Algorithm 3.1. The whole process ends with the last bootstrap sample. Furthermore, discretization is necessary for a proper matching between real and generated data: there are eigth features to discretize.

Now generated and real data are combined together. This new, balanced, training set is going to feed the final classifier. AUROC is used for performance evaluation. The GAN oversampling framework will be compared to the standard approach -with final classification only- and SMOTE package for imbalanced problems. In the next subsection we will describe SMOTE algorithm.

4.6 SMOTE package for imbalanced problems

SMOTE (Synthetic Minority Over-sampling Technique) is an algorithm for imbalanced datasets developed by N.V. Chawla et al. in 2002;[39] the correspondant scikit-learn package became available afterwards.[40] For each sample of the minority class, k nearest neighbours of the same class are selected:m of them, $m < k$, are chosen at random. m synthetic examples are introduced along the line segments joining the considered sample to any of the m selected neighbours. The actual position is in a random point inside the line segment. Figure 4.4 below visually represent the algorithm.

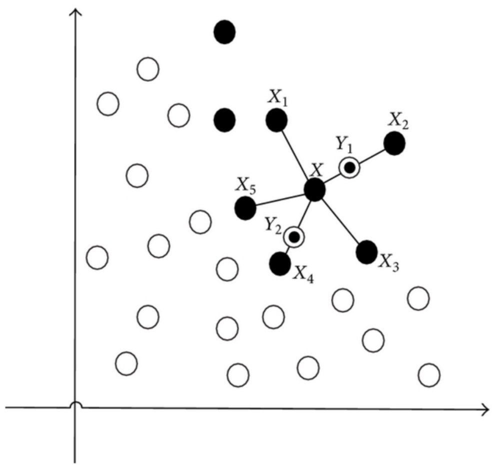


Figure 4.4: Schematic diagram of SMOTE algorithm.

Chapter 5

Results

5.1 MNIST

This chapter shows oversampling's results for MNIST and different real-world datasets. Let us start from MNIST: Vanilla-GAN and DCGAN oversamplings are compared to single classification for various imbalancement ratios (IRs). AUROC will evaluate all performances. In Figure 5.1 we can notice that Vanilla-GAN oversampling slightly outperforms single classification, especially for high imbalancement. For IR=0.5, the two approaches give the same AUROC. DCGAN oversampling instead (Figure 5.2) does not significantly differ from single classification in the high imbalancement region; it outperforms for IR $=0.1 \sim 0.2$ and then for IR=0.5 behaves like Vanilla-GAN.

5.2 Real-world datasets

GAN was applied to real-world datasets afterwards. In the table below, average AUROC is reported for all real-world datasets, using different final classifiers. For every dataset, cross-validation was performed as described before. All results are expressed as "five trials' average AUROC $\pm$ standard deviation". First, second and third AUROC values inside the cells are obtained, respectively, without oversampling, with GAN and with SMOTE.

We can notice that, in general, GAN oversampling does not lead to improvements in AUROC values, it can penalize performance sometimes; SMOTE instead generally increases AUROC as expected. There are three main reasons for GAN's failure: *instability*, *mode collapse* and *high overlap* between minority and majority class. We can visually understand the situation by looking at PCA in two dimensions. Given a dataset in a certain

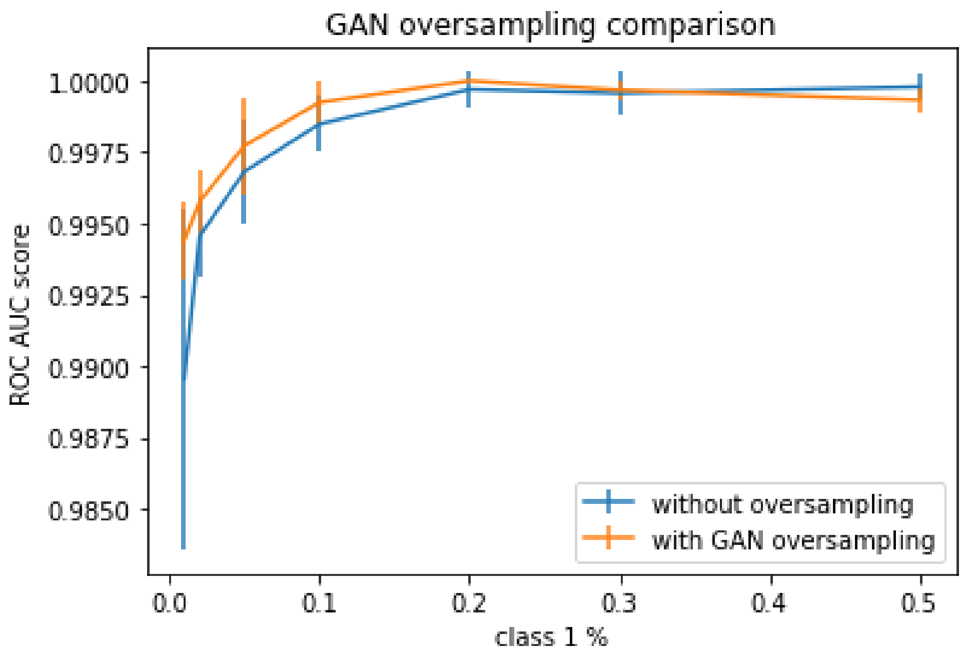


Figure 5.1: AUROC-IR plot for MNIST Dataset with vanilla-GAN. Blue line is obtained without oversampling, orange one with GAN oversampling.

space, PCA is a linear tranformation which projects the features' space in the space of principal components. All principal components are orthogonal among them. The first principal component follows the direction of the greatest variance in our dataset; the second component has to be orthogonal to the first and must have the second greatest variance, and so on for the remaining components. PCA is a useful tool for visualizing complex datasets with many features in just two dimension. Let us consider Seismic Bump for example. The following PCAs show, in blue, class-1 data (our minority class), class-0 data in orange and generated data in green. The first PCA (Figure 5.4) shows a good data reproduction: all four modes are generated, and their position is pretty close to class-1 modes. In the second PCA (Figure 5.5) a common situation is represented: GAN reproduces all modes like before, but they appear shifted and/or rotated. A change in initial conditions (i.e. in the random seed) leads to huge differences in the final output: in other words, vanilla-GAN is unstable. The third PCA (Figure 5.6) exhibits the so-called mode collapse: left,bottom mode and part of the top mode seem to be "merged" in a single, big mode in the left-top position. As a consequence, missing modes are under-represented in the dataset for final classification, so the number of misclassifications increases. Another reason of poor per-

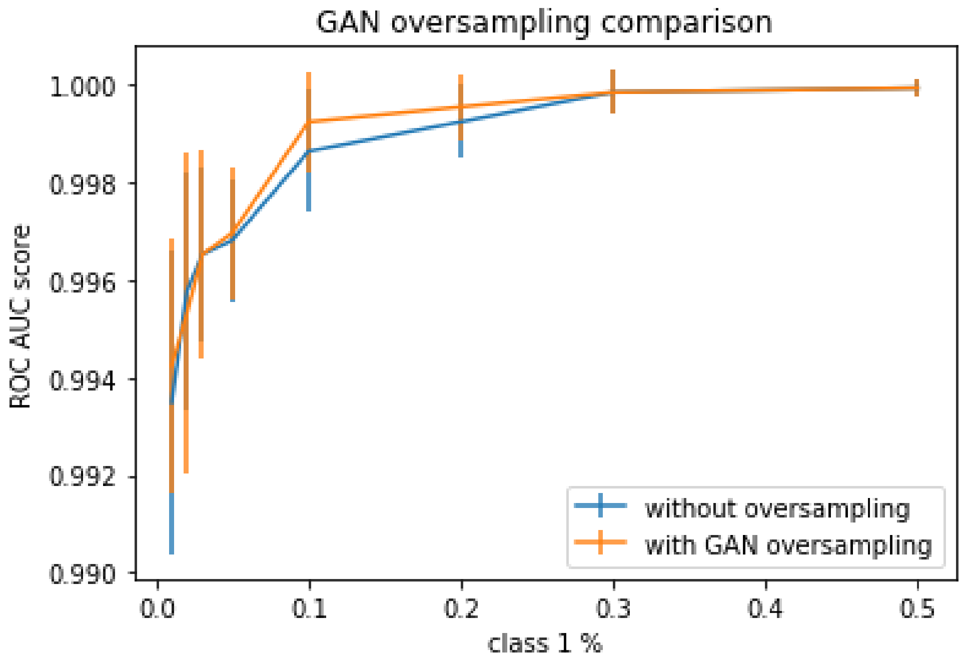


Figure 5.2: AUROC-IR plot for MNIST Dataset with DCGAN. Blue line is obtained without oversampling, orange one with DCGAN oversampling.

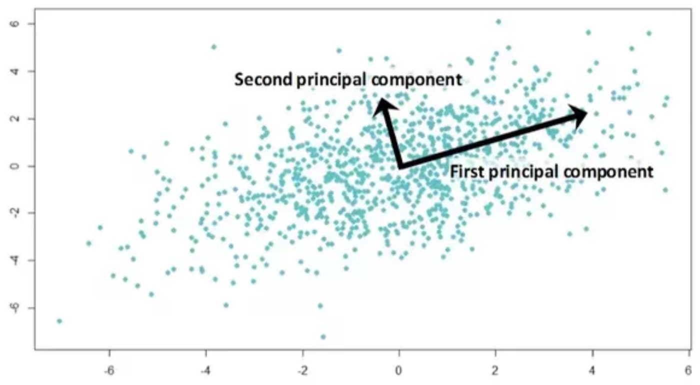


Figure 5.3: Example of PCA in two dimension: principal components are highlighted.

formances relies in the high overlap between majority and minority class: overlap lowers the number of correct classifications and “forces” generator to a greater precision in order to oversample minority class.

In MNIST Dataset, the features’ space is several orders of magnitude

	KNN2	NB	LR	CT
Banknote	0.998 ± 0.000	0.853 ± 0.000	0.957 ± 0.000	0.977 ± 0.000
	1.000 ± 0.000	0.781 ± 0.004	0.889 ± 0.044	0.974 ± 0.011
	0.998 ± 0.000	0.859 ± 0.005	0.957 ± 0.001	0.981 ± 0.002
Blood Transfusion	0.548 ± 0.000	0.541 ± 0.000	0.510 ± 0.000	0.613 ± 0.004
	0.511 ± 0.016	0.500 ± 0.017	0.528 ± 0.029	0.514 ± 0.006
	0.586 ± 0.009	0.604 ± 0.020	0.678 ± 0.009	0.581 ± 0.021
Default Credit Card	0.573 ± 0.000	0.702 ± 0.000	0.597 ± 0.000	0.609 ± 0.001
	0.515 ± 0.000	0.512 ± 0.008	0.501 ± 0.000	0.567 ± 0.002
	0.603 ± 0.001	0.577 ± 0.001	0.678 ± 0.001	0.612 ± 0.003
German Credit Card	0.573 ± 0.000	0.702 ± 0.000	0.597 ± 0.000	0.609 ± 0.001
	0.515 ± 0.000	0.512 ± 0.008	0.501 ± 0.000	0.567 ± 0.002
	0.603 ± 0.001	0.577 ± 0.001	0.678 ± 0.001	0.612 ± 0.003
Heart Statlog	0.750 ± 0.000	0.810 ± 0.000	0.829 ± 0.000	0.682 ± 0.019
	0.671 ± 0.011	0.823 ± 0.028	0.809 ± 0.017	0.750 ± 0.017
	0.760 ± 0.009	0.816 ± 0.012	0.828 ± 0.004	0.681 ± 0.026
Seismic Bump	0.505 ± 0.000	0.629 ± 0.000	0.506 ± 0.000	0.547 ± 0.010
	0.500 ± 0.011	0.514 ± 0.003	0.502 ± 0.004	0.500 ± 0.000
	0.554 ± 0.006	0.660 ± 0.013	0.676 ± 0.005	0.551 ± 0.028
Haberman's Survival	0.517 ± 0.000	0.610 ± 0.000	0.483 ± 0.000	0.623 ± 0.010
	0.509 ± 0.012	0.552 ± 0.016	0.549 ± 0.019	0.519 ± 0.018
	0.499 ± 0.021	0.621 ± 0.011	0.603 ± 0.013	0.634 ± 0.015

Table 5.1: AUROC values for all datasets (rows) and classifiers (columns) K-Nearest Neighbour 2, Naive Bayes, Logistic Regression and Classification Tree. First value in the cell is obtained without oversampling; second and third, respectively, with GAN and SMOTE.

greater than here; overlap is less likely and additional dimensions can be "exploited" for classification: this could partially explain better performances. Imbalanced problems are generally non-separable: this requires ad-hoc machine learning algorithms for the dataset under consideration. For example, looking at the columns of Table 5.2 and 5.2, Random Forest and Naive Bayes seem to perform better than other algorithms (like KMeans): these two algorithms, combined with SMOTE oversampling, could represent a valid and practical solution.

	RF	KM	MLP
Banknote	0.996 ± 0.001	0.463 ± 0.000	0.981 ± 0.000
	0.995 ± 0.006	0.537 ± 0.186	0.781 ± 0.047
	0.996 ± 0.001	0.471 ± 0.093	0.983 ± 0.000
Blood Transfusion	0.608 ± 0.004	0.537 ± 0.000	0.515 ± 0.000
	0.503 ± 0.003	0.467 ± 0.138	0.507 ± 0.016
	0.588 ± 0.012	0.569 ± 0.086	0.679 ± 0.003
Default Credit Card	0.657 ± 0.001	0.557 ± 0.000	0.638 ± 0.000
	0.510 ± 0.000	0.511 ± 0.023	0.501 ± 0.001
	0.683 ± 0.011	0.527 ± 0.056	0.696 ± 0.002
German Credit Card	0.628 ± 0.006	0.520 ± 0.000	0.500 ± 0.000
	0.516 ± 0.005	0.514 ± 0.048	0.545 ± 0.011
	0.654 ± 0.005	0.534 ± 0.060	0.720 ± 0.007
Heart Statlog	0.809 ± 0.011	0.336 ± 0.000	0.821 ± 0.000
	0.796 ± 0.008	0.681 ± 0.074	0.819 ± 0.017
	0.807 ± 0.011	0.489 ± 0.060	0.829 ± 0.010
Seismic Bump	0.506 ± 0.005	0.547 ± 0.000	0.500 ± 0.000
	0.500 ± 0.000	0.502 ± 0.076	0.499 ± 0.000
	0.560 ± 0.015	0.437 ± 0.046	0.660 ± 0.004
Haberman's Survival	0.587 ± 0.021	0.446 ± 0.000	0.538 ± 0.000
	0.503 ± 0.009	0.530 ± 0.051	0.571 ± 0.008
	0.627 ± 0.006	0.525 ± 0.061	0.603 ± 0.021

Table 5.2: AUROC values for all datasets (rows) and classifiers (columns) Random Forest and Multilayer Perceptron. First value in the cell is obtained without oversampling; second and third, respectively, with GAN and SMOTE.

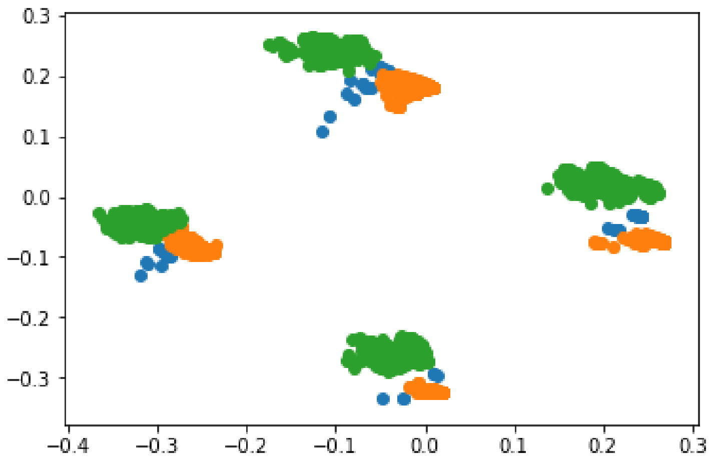


Figure 5.4: This PCA show an example of good data reproduction: generated data (green) resemble to class-1 data (blue). Class-0 data are painted orange.

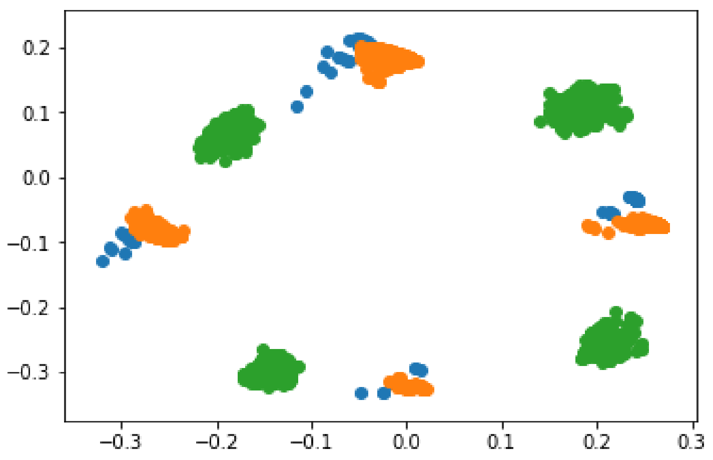


Figure 5.5: Another example of PCA: all modes are reproduced (generated data are coloured green), but they appear rotated compared to class-1 data (blue). Class-0 data are painted orange.

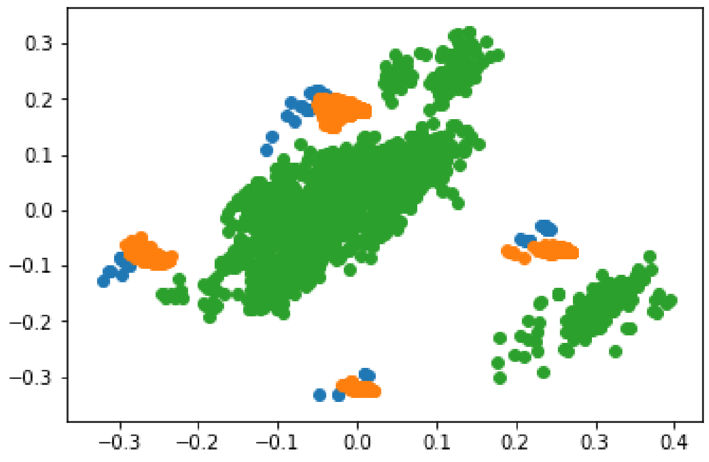


Figure 5.6: In this PCA, the bottom mode is missing: we are facing an example of mode collapse. Generated, class-1 and class-0 data are respectively painted green, blue and orange.

Chapter 6

Conclusion and further developments

In this thesis the framework of GAN oversampling for imbalanced datasets was developed. Mode collapse and vanilla-GAN instability hamper good performances with respect to single classification: other ways to improve classification performances (like hyperparameter optimization) are more convenient. In imbalanced problems, SMOTE algorithm generally performs better than GAN oversampling and single classification: it should be used (or tried at least) for fraud analysis and anomaly detection. For digits classification in MNIST, vanilla-GAN and DCGAN show slightly better or comparable performances with respect to single classification. Here the parameters' space is several orders of magnitude higher than the other datasets: the two classes can be easier separated, although the computational cost increases. In low dimensions instead, there is more chance to be overlapped, which leads to misclassifications. In addition, all real-world datasets are highly overlapped: some of them, like Seismic Bump, does not reach a high AUROC for all classifiers.

Further developments include the usage of more recent GAN (Wasserstein, Unrolled, VEEGAN) which are more stable and not affected by mode collapse. Stability could be improved by a suitable regularization, i.e. the insertion of weights' constraint in the loss function. An optimazion in GAN's training (optimization steps for generator and discriminator) and other types of noise prior $p_z(z)$ could also improve performances. Cross-validation could not only search optimal values for number of neurons in the internal layers: the general framework of GAN oversampling has to be improved too. Bootstrap and oversampling size could be cross-validated.

To sum up, in this thesis the possibility of improving a classifier's performance by GAN oversampling was investigated. Vanilla-GAN seems not appropriate to fully accomplish this task: although interesting results are obtained in higher dimensions with images, when dealing with real-world datasets it generally penalizes the overall performance. More recent GANs and a "well-round" optimization, from the specific architecture to the general oversampling framework, are required.

Bibliography

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN: 0136042597, 9780136042594.

[2] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York, 2001.

[3] Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu. "Forecasting with artificial neural networks:: The state of the art". In: *International Journal of Forecasting* 14.1 (1998), pp. 35 –62. ISSN: 0169-2070. DOI: https://doi.org/10.1016/S0169-2070(97)00044-7. URL: http://www.sciencedirect.com/science/article/pii/S0169207097000447.

[4] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review* (1958), pp. 65–386.

[5] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: ACM, 1992, pp. 144–152. ISBN: 0-89791-497-X. DOI: 10.1145/130385.130401. URL: http://doi.acm.org/10.1145/130385.130401.

[6] Harry Zhang. "The Optimality of Naïve Bayes". In: *In FLAIRS2004 conference*. 2004.

[7] J. R. Quinlan. "Induction of Decision Trees". In: *MACH. LEARN* 1 (1986), pp. 81–106.

[8] Hans-Hermann Bock. "Origins and extensions of the k-means algorithm in cluster analysis". In: 4 (Jan. 2008).

[9] Ian J. Goodfellow et al. "Generative adversarial nets". In: *In NIPS*. 2014.

[10] Pierre Baldi. “Autoencoders, Unsupervised Learning, and Deep Architectures”. In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. Ed. by Isabelle Guyon et al. Vol. 27. Proceedings of Machine Learning Research. Bellevue, Washington, USA: PMLR, 2012, pp. 37–49. URL: http://proceedings.mlr.press/v27/baldi12a.html.

[11] Geoffrey Hinton. *A practical guide to training restricted Boltzmann machines*. 2010.

[12] Carl Eckart and Gale Young. “The approximation of one matrix by another of lower rank”. In: *Psychometrika* 1.3 (1936), pp. 211–218. ISSN: 1860-0980. DOI: 10.1007/BF02288367. URL: https://doi.org/10.1007/BF02288367.

[13] Karl Pearson F.R.S. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. URL: https://doi.org/10.1080/14786440109462720.

[14] Andrew Y. Ng and Michael I. Jordan. “On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes”. In: *Advances in Neural Information Processing Systems 14*. Ed. by T. G. Dietterich, S. Becker, and Z. Ghahramani. MIT Press, 2002, pp. 841–848. URL: http://papers.nips.cc/paper/2020-on-discriminative-vs-generative-classifiers-a-comparison-of-logistic-regression-and-naive-bayes.pdf.

[15] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012. ISBN: 1600490069, 9781600490064.

[16] Sylvain Arlot and Alain Celisse. “A survey of cross-validation procedures for model selection”. In: *Statist. Surv.* 4 (2010), pp. 40–79. DOI: 10.1214/09-SS054. URL: https://doi.org/10.1214/09-SS054.

[17] Jesse Davis and Mark Goadrich. “The Relationship Between Precision-Recall and ROC Curves”. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML ’06. Pittsburgh, Pennsylvania, USA: ACM, 2006, pp. 233–240. ISBN: 1-59593-383-2. DOI: 10.1145/1143844.1143874. URL: http://doi.acm.org/10.1145/1143844.1143874.

[18] Wikipedia Contributors. *Precision and recall*. 2018. URL: https://en.wikipedia.org/wiki/Precision_and_recall.

[19] Tom Fawcett. “An introduction to ROC analysis”. In: *Pattern Recognition Letters* 27.8 (2006). ROC Analysis in Pattern Recognition, pp. 861 –874. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2005.10.010. URL: http://www.sciencedirect.com/science/article/pii/S016786550500303X.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[21] Matt W Gardner and SR Dorling. “Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences”. In: *Atmospheric environment* 32.14-15 (1998), pp. 2627–2636.

[22] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT'2010.* Springer, 2010, pp. 177–186.

[23] Jop Briët and Peter Harremoës. “Properties of classical and quantum Jensen-Shannon divergence”. In: *Physical review A* 79.5 (2009), p. 052311.

[24] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[25] Taeksoo Kim et al. “Learning to Discover Cross-Domain Relations with Generative Adversarial Networks”. In: *Proceedings of the 34th International Conference on Machine Learning.* Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 1857–1865. URL: http://proceedings.mlr.press/v70/kim17a.html.

[26] Tong Che et al. “Maximum-likelihood augmented discrete generative adversarial networks”. In: *arXiv preprint arXiv:1702.07983* (2017).

[27] Mehdi Mirza and Simon Osindero. “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784* (2014).

[28] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[29] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein gan”. In: *arXiv preprint arXiv:1701.07875* (2017).

[30] Vincent Dumoulin et al. “Adversarially learned inference”. In: *arXiv preprint arXiv:1606.00704* (2016).

[31] Luke Metz et al. “Unrolled generative adversarial networks”. In: *arXiv preprint arXiv:1611.02163* (2016).

[32] Akash Srivastava et al. “Veegan: Reducing mode collapse in gans using implicit variational learning”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 3308–3318.

[33] Christian Ledig et al. “Photo-realistic single image super-resolution using a generative adversarial network”. In: *arXiv preprint* (2016).

[34] Scott Reed et al. “Generative adversarial text to image synthesis”. In: *arXiv preprint arXiv:1605.05396* (2016).

[35] Xiaolong Wang and Abhinav Gupta. “Generative image modeling using style and structure adversarial networks”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 318–335.

[36] Nitesh V Chawla. “Data mining for imbalanced datasets: An overview”. In: *Data mining and knowledge discovery handbook*. Springer, 2009, pp. 875–886.

[37] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. “Handling imbalanced datasets: A review”. In: *GESTS International Transactions on Computer Science and Engineering* 30.1 (2006), pp. 25–36.

[38] Li Xu and Jiaya Jia. “Deep Convolutional Neural Network for Image”. In: 2014.

[39] Kevin W. Bowyer et al. “SMOTE: Synthetic Minority Over-sampling Technique”. In: *J. Artif. Intell. Res.* 16 (2002), pp. 321–357.

[40] D. Oliveira G. Lemaitre F. Nogueira and C. Aridas. *imbalanced.oversampling.SMOTE*. 2016. URL: http://contrib.scikit-learn.org/imbalanced-learn/stable/generated/imblearn.over_sampling.SMOTE.html.