A notion of robustness

Maxime Darrin
Supervised by: Pablo Piantanida

September 20, 2020

Contents

List of Figures

1 Introduction

Machine learning and more specifically deep learning have led to tremendous results. Those technologies can now learn how to play hide and seek [2], to help in the diagnostic of cancer [28] or of red blood cells disease. Their results are leading us to expand the use of these technologies to more critical domains or usecases, however there is still a long way to go to ensure the robustness of those. Indeed, even though machine learning models seem to work well, we lack formal proofs and guarantees about them. Actually, most of the models can be fooled by adversarial attacks [36, 10] or fail on some inputs with no apparent, visible reason for a human sight. The problem is that we cannot pretend to use these technologies for critical usecases without a legal framework which needs to take into account some notions of robustness, however, we lack such a notion.

For example, it is extremely difficult to give even a notion of confidence into a result produced by a neural network [23]. The naive idea would be to look at the probability vector given by the network and assess the confidence according to its entropy for example, but as a matter of fact: neural networks show confidence in errors [15]. A second point is that we do not really understand how the trained models work, what they have learned and what matters for them to take decisions.

In a context where the fight against biaises in society and more specifically in machine learning algorithms have arisen, explainability is at the core of all those questions. Explainability has the key role into detecting biaises [13] and into trying to understand how models work [30].

There are mainly two approaches to explainability *a posteriori* explainability and intrinsic explainability. The first one consists of the study of a trained model *a posteriori* as a physical object that we try to explain how it works. This subfield contains all the methods that claim to give insight on which features are used to take a certain decision, what are the main features, what features are ignored from any already trained model. Some works, for example, try to learn a model which highlights the points of interest of an image for a given model [3].

The second subfield works on the class of models used. The idea is to build models whose structure allows explainability by design. We restrict the field of hypotheses to hypotheses whose parameters can be read by a human being. This could be called instrinsic explainability. The most common methods are attention models which embed a simulation of attention which constrains the model to use only some parts of the input and therefore the model is constrained to learn what matter. This can easily be read afterward by an expert. It allows to detect what parts of an image are used to take a decision [34] or which words in a sentence matters to answer a question [9]. In order to give a notion of robustness, bayesian neural networks are trained to learn weights distribution – and not only weights – and then their outputs are several realizations of the output using different samples of weights, the standard deviation of the output gives a notion of robustness [11].

In this work we propose a method to study the robustness of a classifier *a posteriori* based on learning the prior distribution of a trained classifier. In other words, we propose to study the generative distribution associated to a model. This approach allows us to get insights on what the classifier has learnt and what patterns or shapes are relevant to it but it also provides a way to expand the notion of margins between classes which can be seen as a notion of robustness. Indeed, if we can learn sharp representation of the concepts, *ie.* good approximations of the generative distributions associated to a classifier, we can design divergence and distance on those distributions to assess the robustness, *ie.* distances between classes for instance.

1.1 Problem statement

In a classification framework, we suppose we have d classes and the feature space is $[0,1]^n$. Let say we are given an already trained classifier $C: [0,1]^n \longrightarrow \mathbb{S}_d$, where $\mathbb{S}_d$ is the simplex in dimension d. $C(x), x \in [0,1]^n$ models $\mathbb{P}(Y|X=x)$.

In what follows, we denote by $\mathbb{P}(Y=y|X=x,C)$ the probability of x being of class y given by the classifier C. In this setting we write $\mathbb{P}(X|Y=y,C)$, the conditionnal generative distribution of x according to the classifier C.

Let $y \in [|1,d|]$ be a label, the distribution $\mathbb{P}(X|Y=y,C)$ can be seen as the answer to the question "what is an object of class y for the classifier C". We refer to these distributions as concepts learned by the model or as the conditional generative distributions associated to the model.

Our goal is to model these distributions conditionned by the model we are studying and each class.

This framework is motivated by the definition of the bayes error. The bayes error mesures the inherent overlap of the data distributions of each class and therefore the minimal error we can hope to achieve. The point is that, if we had access to the true generative distributions we could see whether a sample is on the margin or well in the center of one of the distribution. It mesures the intrinsic learning complexity of the problem. Using this information we could say whether we can hope to classify the data point or if the uncertainty is too large and that nothing can be done about it. By learning the generative distributions according to the trained classifier, we hope to be able to decide whether a sample is on the margin of the class of the classifier or not. It is way to define a notion of robustness for a model: if the classes infered by the model overlap a lot, or if very close samples are classified differently, we can say that the model is not robust.

1.2 Robustness

If we can model these distributions we can study the margin of our model and particularly the distances between these distributions or more simply their entropy for example. The entropy of these distributions gives us information about the sharpness of a concept, while we can evaluate the distance between concepts using metric on distributions. The choice of these metrics should be guided by prior knowledge on the classification problem we are dealing with. In this work, we will present a proof of concept using *MNIST* [20] and different classic distances and divergence on distributions such as Wasserstein distance [35], Hellinger distance [14] or Bhattacharyya coefficient [25].

First we will recall some elements of information (Section 2) theory needed to grasp what will follow, then we will dive into two papers which paved the way for this work and present our motivations (Section 4). In Section 5 we will present our global setting to model the concepts – or generative distributions – associated to a model and in Section 7 we will present our experimental results.

2 Information theory (Groundings and recalls)

Before going further into our main topic, we need to recall and defines some of the core principles of information theory, then we describe metrics on distributions we will use later on. We will use them to drive our research. Indeed, information theory aims at modeling amount of information in random process. If at the begining it was meant to evaluate communication channel, it has a great role to take into studying the amount of informations that learnt models stores and about what they have actually learnt. For that matter we can already point out that most training losses and common regularizations using are information theory related quantities. We aim to derive meaningful explanation methods explicitly from this point of view. Most of this section came from my course notes of the lecture given by Omar Fawzi at ENS de Lyon in 2018.

2.1 Shannon entropy

Definition 1. We call $(\Omega, \mathcal{E}, \mathbb{P})$ a *finite system* with Ω begin the universe, $\mathcal{E}$ the set of events and $\mathbb{P}$ the probability measure. We'll write *random variables* in capital letters such as $X : \Omega \to \mathcal{X}$ with $\mathcal{X}$ some set, $P_X : \mathcal{X} \to \mathbb{R}, P_X(x) = \mathbb{P}\{X = x\}$ the *density* of the r.v. X, and $P_{X|E}(x) = \mathbb{P}\{X = x|E\}$ for $E \in \mathcal{E}$. In addition, when X a r.v. has distribution P, we'll write $X \sim P$.

2.2 Entropy of an event

The *entropy* is not a term specific to Information Theory. It comes from a greek word meaning "transformation". In general, *entropy* characterize the level of disorganization or unpredictability of the information content of a system.

2.2.1 Definition

We want an entropy-like real valued function defined on the set of event, representing how much information we have on the system's state; say $h : \mathcal{E} \to \mathbb{R}_+ \cup \{\infty\}$. We want it to have the following properties.

(i) $h(E)$ should only depend on $\mathbb{P}\{E\}$, that means $h(E) = f(\mathbb{P}\{E\})$ with f a function from the closed unit interval to the closed positive half-line $\mathbb{R}_+ \cup \{\infty\}$.

(ii) We want it to be normalised, for instance $f(\frac{1}{2}) = 1$.

(iii) If E and E' are independent, then knowing one of the two gives no information about the probability of occurence of the other, so we should have $h(E \cap E') = h(E) + h(E')$. So for E and E' independent with probability $\frac{1}{2}$, we have $2 = h(E \cap E') = f(\frac{1}{4})$. More generally, $f(2^{-m}) = m$ and $f(2^{-n/m}) = \frac{n}{m}$. If in addition, we assume that f is continuous then we will have $f(p) = -\log_2 p$.

Definition 2 (Entropy of an event)**.** Let $E \in \mathcal{E}$ be an event. We call *Shannon entropy* of E the quantity

$$h(E) = -\log_2 \mathbb{P}\{E\}$$

It defines a function $h : \mathcal{E} \to [0, \infty]$. If $X \in \mathcal{X}$ is a random variable, we denote by the *surprisal* of X the function

$$\begin{array}{llll} h_X : & \mathcal{X} & \to & \mathbb{R}_+ \cup \{\infty\} \\ & x & \mapsto & h(\{\mathcal{X} = x\}) = -\log_2 \mathbb{P}_X(x) \end{array}$$

Definition 3 (Shannon Entropy)**.** The *Shannon entropy* of a random variable $X \in \mathcal{X}$ is given by

$$H(X) = \mathbb{E}\{h_X(X)\} = -\sum_{x \in \mathcal{X}} \mathbb{P}_X(x) \log_2 \mathbb{P}_X(x)$$

Remark 1.

- $H(x)$ only depends on the values $P_X(x)$, that's why we also write $H(\mathbb{P}_X)$.
- Recall the convention $0 \log_2 0 = 0$.
- Some words about notations; $\mathbb{P}_X(X)$ is the random variable

  $$\mathbb{P}_X : E \in \mathcal{E} \mapsto \mathbb{P}\{X \in E\}$$

- $\mathbb{E}\{h_X(X)\}$ is not the only interesting entropy measure. There exists a very big family of them. For culture, the reader is encouraged to remember

  $$H_{\min}(X) = \min_x h_X(x)$$

2.2.2 Upper bound of the Shannon entropy

Proposition 1. *For any random variable $X \in \mathcal{X}$ we have :*

$$0 \leq H(X) \leq \log |\mathcal{X}|$$

with $H(X) = 0$ if and only if X is a constant random variable and $H(X) = \log |\mathcal{X}|$ if and only if X is uniform on $\mathcal{X}$.

Proof. The first inequality $H(X) \geq 0$ is clear. The second one is just the concavity of log .

$$\begin{aligned} H(X) &= \mathbb{E}\left\{\log \frac{1}{\mathbb{P}_X(X)}\right\} \\ &\leq \log\left(\mathbb{E}\left\{\frac{1}{\mathbb{P}_X(X)}\right\}\right) \text{ (concavity of log)} \\ &= \log\left(\sum_{x\in\mathcal{X}} \mathbb{P}_X(x)/\mathbb{P}_X(x)\right) \\ &= \log|\mathcal{X}| \end{aligned}$$

The equality condition follows from the fact that log is strictly concave. □

Example 1 (Binary entropy function)**.**

Let $X \in \{0, 1\}$ be a random variable s.t. $P_X(1) = p$. Then,

$$H(x) = -p \log p - (1 - p) \log(1 - p) := h_2(p)$$

h_2 is called the *binary entropy function.*

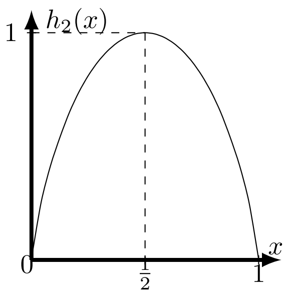


Figure 1 – h_2 graph

2.3 Joint entropy and conditional entropy

Definition 4. Let $X \in \mathcal{X}, Y \in \mathcal{Y}$ be two random variables in the same probability space. The *joint entropy* is

$$H(X, Y) = -\sum_{\substack{x\in\mathcal{X}\\ y\in\mathcal{Y}}} \mathbb{P}_{X,Y}(x, y) \log \mathbb{P}_{X,Y}(x, y)$$

We write $H(X, Y) = H(XY)$

Definition 5. The *conditional entropy* $H(X|Y)$ is defined by

$$H(X|Y) = \sum_{y\in\mathcal{Y}} \mathbb{P}_Y(y) H(X|Y = y)$$

where $H(X|Y = y) = H\left(\mathbb{P}_{X|Y=y}\right)$.

Remark that if $X = Y$, we have $H(X|Y) = 0$. On the other hand, if X and Y are independent, then $H(X|Y) = H(X)$. We are glad that such properties hold. Indeed, if $X = Y$, knowing Y implies that we completely know X as well, that is to say that the uncertainity about Y is null. Furthermore, if X and Y are independent, knowing Y gives us no information about X.

Proposition 2. *We have* $H(X|Y) = H(XY) - H(Y)$.

Proof.

$$\begin{aligned}
H(XY) &= -\sum_{x,y} \mathbb{P}_{X,Y}(x, y) \log\left(\mathbb{P}_Y(y)\mathbb{P}_{X|Y=y}(x)\right) \\
&= -\sum_{x,y} \mathbb{P}_{X,Y}(x, y) \log \mathbb{P}_y(y) - \sum_{x,y} \mathbb{P}_{X,Y}(x, y) \log \mathbb{P}_{X|Y=y}(x) \\
&= -\sum_{y} \mathbb{P}_Y(y) \log \mathbb{P}_y(y) - \sum_{y} \mathbb{P}_Y(y) \sum_{x} \mathbb{P}_{X|Y=y}(x) \log \mathbb{P}_{X|Y=y}(x) \\
&= H(Y) + H(X|Y)
\end{aligned}$$

□

Definition 6. The *mutual information* is defined by

$$I(X : Y) = H(X) - H(X|Y)$$

The name is chosen for its transparency. This operator is of course symmetric i.e $I(X : Y) = I(Y : X)$.

Observe that

$$I(X:Y) = H(X) + H(Y) - H(XY) = \sum_{x,y} P_{X,Y}(x,y) \log \frac{P_{X,Y}(x,y)}{P_X(x)P_Y(y)}$$

Example 2. If $X = Y$ then $I(X:Y) = H(X)$ whereas when X and Y are independent, then $I(X:Y) = 0$. We can represent it graphically :

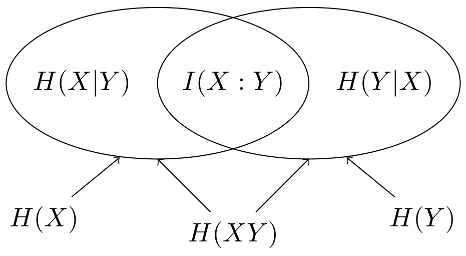


Figure 2 – a graphic representation of how entropies behave

2.4 Relative entropy

Definition 7. The *relative entropy* (also called Kullback-Leibler divergence) between two distributions P and Q on $\mathcal{X}$ is defined as

$$D(P\|Q) = \begin{cases} \infty & \text{if } P(x) > 0 \text{ and } Q(x) = 0 \\ & \text{for some } x \in \mathcal{X} \\ \sum\limits_{x\in\mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} & \text{otherwise} \end{cases}$$

Remark 2. We give here some useful properties of the relative entropy.

- $D(P\|Q)$ can be arbitrarely large.
- $D(P\|P) = 0$, and D is not symmetric, so $D(P\|Q) \neq D(Q\|P)$ in general.
- $D(P\|Q)$ has operational interpretations ; it indicates how efficiently one can differenciate two samples of distribution P and Q (hypothesis testing, also see tutorial).
- The following equalities stand.

  $I(X:Y) = D(P_{XY}\|P_X \times P_Y)$

  $H(X) = \log|\mathcal{X}| - D(P_X\|\mathcal{U}_\mathcal{X})$ where $\mathcal{U}_\mathcal{X}$ is the uniform distrib. on $\mathcal{X}$.

  $H(X|Y) = \log|\mathcal{X}| - D(P_{XY}\|\mathcal{U}_\mathcal{X} \times P_Y)$

Proposition 3. *For any distributions P, Q we have $D(P\|Q) \geq 0$ with equality if and only if $P = Q$.*

Proof. Let $S = \{x \in \mathcal{X} : P(X) > 0\}$. Using the concavity of log directly on the definition of D, we obtain

$$D(P\|Q) \geq -\log \sum_{x\in S} P(x)\frac{Q(x)}{P(x)}$$

Thus, $D(P\|Q) \geq -\log\sum_{x\in S} Q(x) \geq 0$, as $\sum_{x\in S} Q(x) = 1$

For equality condition, one should saturate concavity i.e $\frac{Q(x)}{P(x)} = c$ for all $x \in S$, and $\sum_{x\in S} Q(x) = 1$. This implies $P(x) = Q(x)$ for all $x \in \mathcal{X}$. $\square$

Remark 3. We have :

$$D(P\|Q) \geq c\|P-Q\|_1^2 \qquad \text{(Pinsker's inequality)}$$

Corollary 1. *For any random variables X, Y, we have $I(X:Y) \geq 0$,*
$H(X) \geq H(X|Y)$ and $H(X) + H(Y) \geq H(XY)$.

Remark 4. We can link the cross-entropy and the KL-divergence and thus we show that they are equal up to a constant. Therefore minimizing one or the other is equivalent.

$$\mathrm{H}(p,q) = \mathrm{E}_p[-\log q] = \mathrm{H}(p) + D_{\mathrm{KL}}(p\|q)$$

In what follows, we do not need the joint entropy and therefore we use $H(\cdot,\cdot)$ to design the cross-entropy.

2.5 Information bottleneck

The information bottleneck principle has been introduce in 2000 by N.Tishby, F.C Pereira and W.Bialek [33] as a way to provide a quantitative notion of "relevant" or "meaningful". The core idea of this principle is to keep information that are relevant for a given variable and to drop the information that are not. This is to a give a notion of "relevant for that purpose".

Let say we have a signal modeled by a random variable X and some event denoted by the random variable Y, which is also called relevance variable. The amount of relevant information in X about Y is then

$$\min_{\tilde{X}} I(X, \tilde{X}) - \beta I(\tilde{X}, Y)$$

.

Where $\tilde{X}$ is a compression of X and β the Lagrange multiplier modeling the trade-off between compression and keeping relevant information. The whole point here, is to find a $\tilde{X}$ that is as informative as possible about Y (meaningful) and less informative as possible about the source X (compression).

2.6 Distributions metrics

2.6.1 Wasserstein distance and optimal transport

Definition 8. (Optimal transport problem: Monge formulation) The modern formulation of the transportation problem has been stated by Monge and improved Kantorovich later on. The optimal transport problem consists of computing the minimal cost of moving mass from a starting distribution to another in some metric space. Let say we have two metric spaces (M, d_M), (N, d_N) and p, q the two probability densities.

Let $c : M \times N \to [0, \infty]$ be a Borel-measurable function, it is the cost function to move one unit of mass from $x \in M$ to $y \in N$. Given probability measures μ on M and v on N, Monge's formulation of the optimal transportation problem is to find a transport map $T : M \to N$ that realizes the infimum

$$\inf\left\{\int_X c(x, T(x))\mathrm{d}\mu(x) \mid T_*(\mu) = \nu\right\}$$

where $T_*(\mu)$ denotes the push forward of μ by T *ie.* the probability distribution of $T(X)$, where $X \sim \mu$ is ν. Such a map T, that reach the infimum, is the optimal transport map.

Definition 9. (Optimal transport problem: Kantorovich formulation)

Kantorovich proposes another formulation of the optimal transportation problem which is leads to the Wasserstein distance. The optimization problem becomes to find a probability distribution over $N \times M$ that meets the infimum:

$$\inf\left\{\int_{M\times N} c(x,y)\mathrm{d}\gamma(x,y) \mid \gamma \in \Gamma(\mu,\nu)\right\}$$

where $\Gamma(\mu, v)$ denotes the collection of all probability measures on $M \times N$ with marginals μ on X and v on Y.

Definition 10. Let (M, d) be a metric space. For $p \geqslant 1$, we denote $P_p(M)$ the collection of all probability measures μ on M with finite p^{th} moment.

The p^{th} wasserstein distance between two probability measures μ and ν in $P_p(M)$ is defined as

$$W_p(\mu, \nu) := \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{M \times M} d(x, y)^p \mathrm{d}\gamma(x, y) \right)^{1/p}$$

where $\Gamma(\mu, \nu)$ is the set of all measures on $M \times M$ with marginals μ and ν.

Wasserstein distances and optimal transport interpretation are of great interest in image processing since they allow to take into account the spatiality of the distribution. Therefore it can be used to measure a notion of edition distance between images whose shape would be close but just a little bit rotated or moved. We will use this interpretation to measure distance between concepts in our *MNIST* experiment. However, algorithms to compute wasserstein distance are quite expensive to execute and not tractable for problems in larger dimensions [24, 26].

2.6.2 Sliced Wasserstein distance

Even though wasserstein distance is hard to compute exactly, we can approximate it using the sliced wasserstein distance algorithms [4] which relies on the fact that computing 1-d wasserstein distance can be done exactly very efficiently.

Algorithm 2 Computes an approximation of the L_2 wasserstein distance between 2 images.

```
Require: Image 1 $I_1$, Image 2 $I_2$, $K$ the number of projection to consider.
  $\text{wd}_2 = 0$
  for $k = 1$ to $K$ do
    Sample a random projection $p$ on the unit ball.
    Computes $p_1 = p \cdot I_1$ and $p_2 = p \cdot I_2$
    Computes the 1-$d$ Wasserstein distance between $p_1$ and $p_2$: $\text{1d-wd}_2(p_1, p_2)$
    $\text{wd}_2 = \text{1d-wd}_2(p_1, p_2)$
  end for
  return $\frac{\text{wd}_2}{K}$
```

In our *MNIST* case we will use it to compute a distance between the concepts of digits, in the form of probability maps over the pixels (See Section 5.5).

2.6.3 Bhattacharyya coefficient

The Bhattacharyya coefficient is a metric on distributions which measures the amount of overlap between two distributions and it is closely related to the Hellinger distance. We will use it to measure the separability of classes in our classification framework [25].

Definition 11 (Bhattacharyya coefficient)**.** Let p, q be two distributions over X, the Bhattacharyya coefficient is defined by:

$$BC(p, q) = \int_{x \in X} \sqrt{p(x)q(x)} dx$$

In the discrete case we simply have:

$$BC(p, q) = \sum_{x \in X} \sqrt{p(x)q(x)} dx$$

A value of 1 means that the two distributions perfectly overlap where a value of 0 means that the two distributions do not overlap at all.

This measure can be easily evaluated for discrete distributions and its computation is tractable.

2.6.4 Hellinger distance [14]

Definitions

Definition 12. (Continuous case)

Let p, q be probability density functions, *ie.* p, q are absolutly continuous with respect to the Lebesgue measure.

$$H^2(p, q) := \frac{1}{2} \int (\sqrt{p(x)} - \sqrt{q(x)})^2$$

Definition 13. (Discrete case)

Let $p = (p_1, \cdots, p_n)$ and $q = (q_1, \cdots, q_n)$ be probability vectors, we define the Hellinger distance as:

$$H^2(p, q) := \frac{1}{\sqrt{2}} \sum_{k=1}^{n} (\sqrt{p_k} - \sqrt{q_k})^2$$

Properties

Proposition 4.

$$H(p, q) = \sqrt{1 - BC(p, q)}$$

with $BC(p, q)$ the Bhattacharyya coefficient between p and q.

3 Machine learning Background

3.1 Neural Networks

We quickly recall some basics about neural network and more specifically we recall that they are universal approximator and thus they are a good choice to model any continuous function when no close form exists. We will use them exactly for that purpose in what follows: we will use them to model the generative probability distribution associated to a classifier. We first define a *formal neuron* the most basic building block of a neural network.A formal neuron takes a vector as inputs, then computes the sum of the inputs weighted by a weights vector and eventually passes the result to an activation function to introduce non linearity.

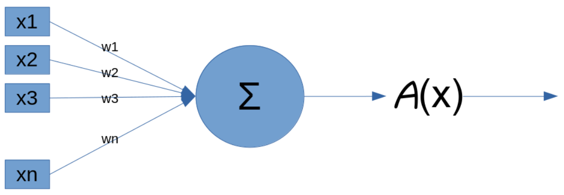


Figure 3 – A formal neuron

We denote $\mathcal{A}$ the activation function. If $x \in \mathbb{R}^n$, then by convention $\mathcal{A}(x) \in \mathbb{R}^n$ and it is the element-wise application of $\mathcal{A}$ on the vector. To each neuron is assigned a weights vector w with one weight per entry of the neuron, we can then write a formal neuron as a function of the form:

$$\begin{aligned} &\mathbb{R}^n \to \mathbb{R} \\ &x \mapsto \mathcal{A}(\textstyle\sum_{i=0}^{n} w_i x_i) \end{aligned}$$

A neural network is then a directed acyclic graph where nodes without predecessors are inputs, nodes without sucessors are outputs and inner nodes are formal neurons.

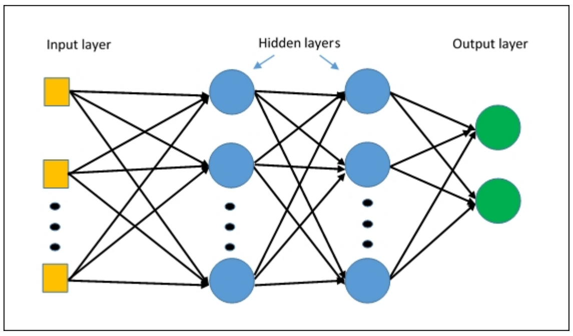


Figure 4 – Example of multi-layer perceptron

Definition 14. A multi layer perceptron is a neural network organized as a collection of layers where 2 neurons of the same layers are disconnected and the neuron of two different layers are fully connected.

Let W_i be the weights matrix from layer i to layer $i+1$ and x_i be the output of the layer i we can write the outputs of output $i+1$: $\mathcal{A}(W_i x_i)$.

If $\mathcal{W} = (W_1..W_N)$ represents all the weights of the model, then the neural multilayer perceptron can be expressed as:

$$f_{\mathcal{W}} : \begin{array}{l} \mathbb{R}^n \to \mathbb{R}^m \\ x \mapsto f_{\mathcal{W}}(x) \end{array}$$

Since neural network and multi-layer perceptron are universal approximators, which means that they can approximate any continous function on compacts subset of R^n, we can use them to approximate the functions underlying the observed data. We have an input domain $\mathcal{X}$ and an output domain $\mathcal{Y}$. In the case of supervised learning we have a set D of solved cases: $D = (x_i, y_i)_{i \in \mathbb{N}}$. These are points on the curve we want to approximate, we can use them to train our neural network to learn this function.

To train a neural network, ie to tune the weights to get a good approximation, we usually use the gradient descent algorithm in order to minimize a well chosen error on a training dataset. So, first we need to define a loss function which will gives us some kind of distance (it is not necessary a distance in mathematical way) between two elements of $\mathcal{Y}$. The only property that is needed is that it has to be non-negative.

$$l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$$

Let's denote the neural network we want to train by $f_{\mathcal{W}}$. In order to train our network on a batch β of data we define the regret $\mathcal{R}^{\beta}_{f_{\mathcal{W}}}$ achieves by $f_{\mathcal{W}}$ on the points in β.

$$\mathcal{R}^{\beta}_{f_{\mathcal{W}}} = \frac{1}{s_i} \sum_{j=1}^{|\beta|} l(f_{\mathcal{W}}(x_j), y_j)$$

The core principle is to approximate the gradient of the regret on a batch and then take a step in that very direction. We mainly use the backpropagation algorithm to train neural networks[32]. (see Annex A.1)

4 Motivation and inspiration

The information bottleneck method has been published 20 years ago [33] and describes a quantity to minimize to get the optimal compression while keeping as much as possible the relevant data. With the recent developpement of artificial intelligence and machine learning models, some research have proposed to apply information theory and more precisely this method. We will here describe two recent works on explainability based on information theory and from which I based the new approach I proposed during the internship.

4.1 Learning to Explain: An Information-Theoretic Perspective on Model Interpretation [6]

This paper proposes a framework to think intepretability from an information theoretical point of view by considering the mutual information between the ouput of a model and the explanation. It can be applied for classification or regression model, but for the sake of simplicity we will explore the classification case. We suppose we have access only to the probability distribution output by a model m, $\mathbb{P}_m(\cdot|X=x)$ for a response variable Y. Mutual information has been widely use in features selection, in that perspective, it aims to produces a feature selection between the input and the output of the model (and not the true random variable). We suppose that X lives in $\mathbb{R}^d$ and $Y \in [|1, N|]$.

In order to construct an instance wise explanation we consider the pair (X, Y) which is characterized by the marginal distribution $\mathbb{P}_X(\cdot)$ and a family of distribution over the targets of the form $Y|x \sim \mathbb{P}_m(\cdot|x)$. This family is defined by the model we try to explain. The goal is to build a mapping from the feature space $\mathbb{R}^d$ to the set of all the subset of size k of the features *ie.* $\left\{S \subset 2^d ||S| = k\right\}$, where k is the number of most informative features and whose choice is left to the user. The explanation is such a mapping $\mathcal{E}$, it takes an input vector x (a data point) and produces the set of features that matters $S = \mathcal{E}(x)$. We denote x_S the vector formed from the input vector keeping only the features in S.

This instancewise feature selection problem can put in the following form: finding $\mathcal{E}$ which maximizes

$$\max_{\mathcal{E}} I(X_S, Y)$$

, *ie.* we want the explainer which maximizes the mutual information between the selected features and the output produced by the model.

The optimal global solution to this optimization problem problem is in fact the explainer which minimizes the expected length of encoded message for the model $\mathbb{P}_m(Y|x)$ using $\mathbb{P}_m(Y|x_S)$.

However this method is not tractable in that shape.

First we need to rewrite the Mutual information between X_S and Y. It can be expressed in terms of the conditional distribution of Y given X_S :

$$\begin{aligned}
I(X_S, Y) &= \mathbb{E}\left[\log \frac{\mathbb{P}_m(X_S, Y)}{\mathbb{P}(X_S)\,\mathbb{P}_m(Y)}\right] = \mathbb{E}\left[\log \frac{\mathbb{P}_m(Y \mid X_S)}{\mathbb{P}_m(Y)}\right] \\
&= \mathbb{E}\left[\log \mathbb{P}_m(Y \mid X_S)\right] + \text{ Const.} \\
&= \mathbb{E}_X \mathbb{E}_{S|X} \mathbb{E}_{Y|X_S}\left[\log \mathbb{P}_m(Y \mid X_S)\right] + C
\end{aligned}$$

However, it is impossible to compute expectations in general under the conditional distribution $\mathbb{P}_m(\cdot \mid x_s)$. Indeed, when we select features we change significantly the input vector by removing the non-selected features, thus, the base model cannot be trust to work on these. To avoid this problem, the article proposes to introduce a variational family to approximate the base model on features selected vectors. Actually, the goal of these networks is to approximate the behavior of the classifier but with masked inputs: we dot not reduce the dimension, just set to 0 the non-selected features.

$$\mathcal{Q} := \{\mathbb{Q} \,|\mathbb{Q} = \{x_S \to \mathbb{Q}_S(Y \mid x_S), S \in \wp_k\}\}$$

Each member $\mathbb{Q}$ of the family $\mathcal{Q}$ is a collection of conditional distributions $\mathbb{Q}_S(Y \mid x_S)$, one for each choice of k -sized feature subset S.. It derived a lower bound on the mutual information by applying the Jensen inequality for any $\mathbb{Q}$:

$$\begin{aligned}
\mathbb{E}_{Y|X_S}\left[\log \mathbb{P}_m(Y \mid X_S)\right] &\geqslant \int \mathbb{P}_m(Y \mid X_S) \log \mathbb{Q}_S(Y \mid X_S) \\
&= \mathbb{E}_{Y|X_S}\left[\log \mathbb{Q}_S(Y \mid X_S)\right]
\end{aligned}$$

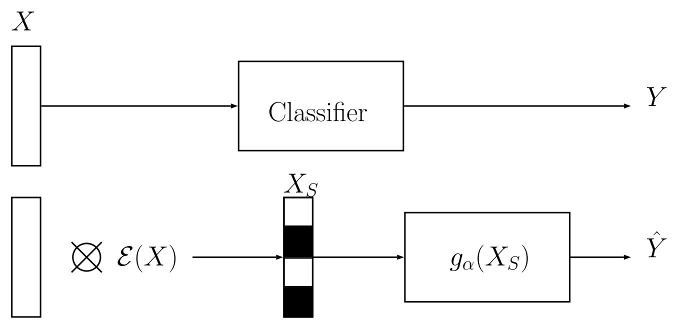


Figure 5 – Scheme of features selection. We use the explainer to select features by setting the non-selected features to 0 and train g_α to mimic the true classifier using as input only the selected features.

There is equality if and only if $\mathbb{P}_m\left(Y \mid X_S\right)$ and $\mathbb{Q}_S\left(Y \mid X_S\right)$ have the same distribution. This lower bound on the mutual information $I\left(X_S; Y\right)$ allows us to relax the maximization problem as maximizing the variational lower bound, over both the explanation $\mathcal{E}$ and the conditional distribution $\mathbb{Q}$:

$$\max_{\mathcal{E},\mathbb{Q}} \mathbb{E}\left[\log \mathbb{Q}_S\left(Y \mid X_S\right)\right] \quad \text{such that } S \sim \mathcal{E}(X)$$

The idea is to model $\mathbb{Q}(\cdot|x_S)$ by a neural network. To proceed, it defines a neural network $g_\alpha : \mathbb{R}^n \longrightarrow \mathbb{S}_{N-1}$, where $\mathbb{S}_{N-1}$ is the simplex of dimension $N-1$. Then $Q_S(Y|x_S) = g_\alpha(\tilde{x_S})$, where $\tilde{x_S}$ is the same as x for the features in S and the others are features are replaced by zero.

There is still the problem of sampling over the subset of size k. Instead of sampling over the subsets directly, the authors propose to sample k features according to a learned discrete probability distribution over the features using the Gumbel Softmax Trick [16]. We will not describe it here since we will use it and dive into it later (Section 5.5).

4.2 Explaining a Black-box model by using a deep variational information bottleneck approach [3]

This paper propose to learn an explainer whose goal is to highlight the relevant parts of the input to the model. This goal is achieved by training the explainer to select groups of pixels (in the case of images) or words (in the case of natural language processing). The notion of significance here is defined by an information bottleneck objective whose goal is to produce selection that are as informative as possible about the predicted class by the classifier and as less informative as possible about the input. In a nutshell, it compresses the information from the input to keep only what matters to the model.

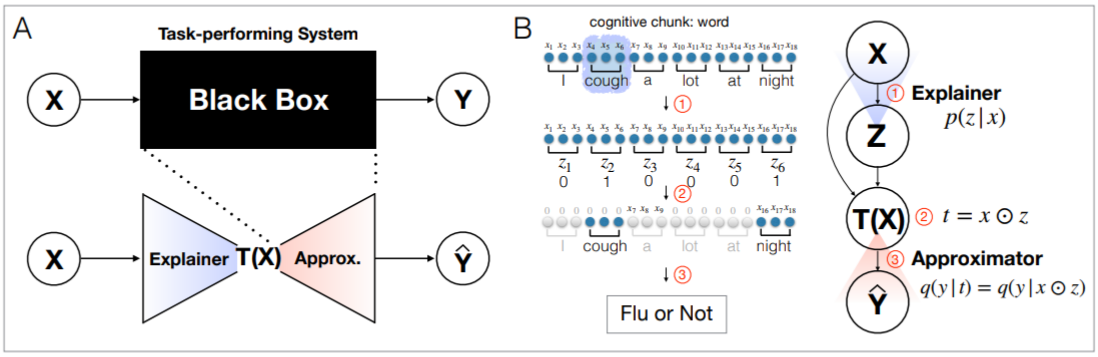


Figure 6 – Explainer scheme

More formally, this paper proposes to build an explainer which produces a binary mask z following a learnt distribution $p(z|x)$ where x is the input to the model. Let t denote the masked input, *ie.* $t = x \cdot z$, where $\cdot$

is the elementwise multiplication. The explainer is trained to minimize the following information bottleneck objective:

$$p(\mathbf{z} \mid \mathbf{x}) = \underset{p(\mathbf{z}|\mathbf{x}),p(\mathbf{y}|\mathbf{t})}{\arg\max} \; \mathbf{I}(\mathbf{t}, \mathbf{y}) - \beta \mathbf{I}(\mathbf{x}, \mathbf{t})$$

where $\mathbf{I}(\cdot, \cdot)$ is mutual information of two random variables and y follows the distribution $p(\cdot|t)$, the probability distribution over the classes output by the classifier for the input t.

I partially reproduced the results to fully grasp the method and use it as basis for what follows [7].

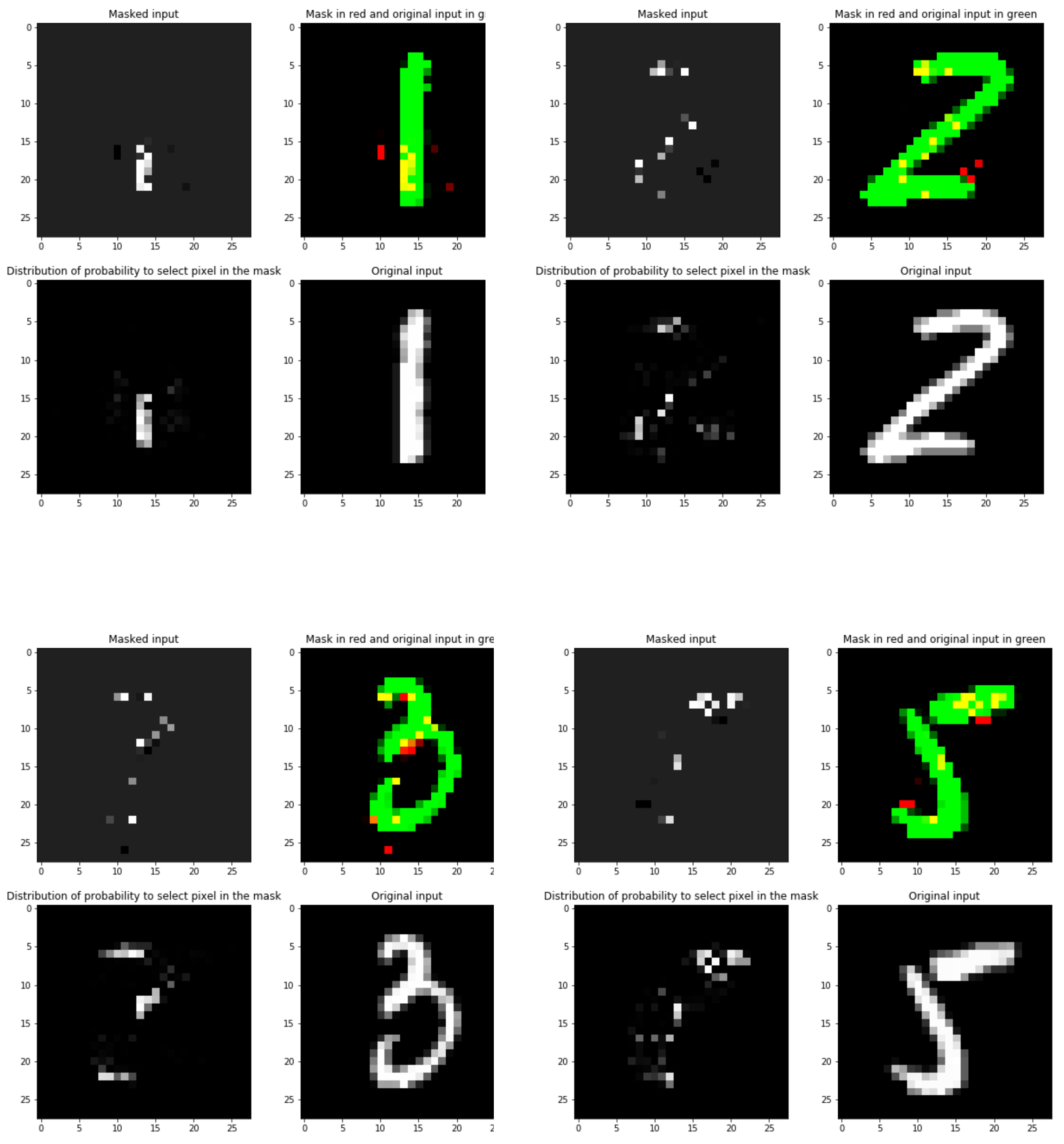


Figure 7 – Examples of explanation using [3]

This method provides an instance wise level of explainability, which means it gives insights about the specific point of interest for a specific but it does not give information the whole classifier. The explainer learns to detect what would matter inside an input to the classifier and build a probability distribution one those point of interest. If we could explain the explainer we would get information about the classifier as a whole but turn around here.

The method we propose is based on the same idea of this one but it does not learn to select what matters in a specific instance, it learns to build inputs specifically designed to be classified as we want to the model. This way we get a modelwise explainability framework.

5 Modeling concepts

5.1 Global setting

As presented in Section 1.1 our goal is, given a classifier C to model its associated generative distribution. We will work in the case where C is a differentiable classifier and we will learn a differentiable generative model using gradient descent. First we will describe precisely the settings of this optimization problem (Section 5.2), then we will need to develop models to model the generative probability distributions. For that matter, we shall recall how does work the reparametrization trick (Section 5.4) and then use it to develop a categorical model based on Gumbel distributions (Section 5.5) and a classic multivariate gaussian model (Section 5.6). In a last part we will dive into normalizing flows [18] (Section 5.7) for the sake of comprehensiveness even though they are not tractable for the purpose we pursue.

5.2 Optimization problem

Definition 15. (Optimization problem)

Let C be our classifier, X the features space, $Y = [|1, n|]$ the classes and $\mathbb{S}_n$ the simplex of dimension n.

In order to model for each class y, $\mathbb{P}(X|Y = y, C)$ we propose to train a parametric distribution $(\pi_{\theta_y})_{y \in Y}$ by minimizing

$$\mathbb{E}_{x \sim \pi_{\theta_y}} [l(C(x), y)]$$

Where $l : \mathbb{S}_n \times \mathbb{S}_n \longrightarrow \mathbb{R}$ is a loss function and $C(x)$ is the probability distribution over the classes for input x according to the classifier.

ie. we want that for all class y, the samples $x \sim \pi_{\theta_y}$ are classified as y by the classifier.

In addition to the problem of the class of models to use, we need to ensure that the generated samples are diverse enough. An ordeal we could face is that our distribution will output always the same sample with little variation. To avoid this pittfall we will propose an information based regularization (Section 5.3).

In this case the loss function l is the cross-entropy, of p and q two probability distributions over the same set of events, mesures the number of bits needed to identify an element of the underlying set if these elements are encoded using a coding scheme optimized for q instead of a coding scheme optimized for the true distribution p. We can remark that the cross-entropy with p as reference distribution is equal to the KL-divergence up to a constant, thus it is equivalent to optimize one or the other.

$$\mathbb{E}_{x \sim \pi_{\theta_y}} [H(C(x), y)]$$

Where $H(\cdot, \cdot)$ is the cross-entropy – and not the joint entropy – between two distributions and $C(x)$ is the probability distribution over the classes for input x according to the classifier.

5.2.1 Sharp representation

By training the generator to produce samples that maximizes the cross-entropy (or KL-divergence) presented above, it produces only examples that are strongly classified *ie.*, that are well classified as y with high probability according to the classifier. It gives an idea of the strongest and sharpest example of sample of class y.

But, it does not explore the edges of the classifier, the areas where the sample would be well classified as y but with a probability barely above $\frac{1}{2}$.

5.2.2 Fuzzy representation

In order to explore a larger space we can edit our objective function to take into account only the samples that are in the end not well classified, *ie.* the ones for which the probability of being y according to the classifier is below $\frac{1}{2}$. Which leads to the following objective

$$\mathbb{E}_{x\sim\pi_{\theta_y}}\left[\mathbb{1}_{\arg\max C(x)\neq y)}H(C(x),y)\right]$$

. Here we use y indifferently to denote the true distribution for class y, *ie.* a dirac distriubtion on y (a onehot vector) and to denote the class itself.

5.3 Entropy regularization and information theory interpretation

If the cross-entropy loss allows the generator to learn to build well classified samples, we need to enforce some regularization to force the distribution to cover more of the space. Actually, we want that for each class, the conditionnal generative distribution is as uncertain as possible. We want each generative distribution to be complex enough to produce a wide range of samples that will be well classified. A way to enforce such an uncertainty is to maximize the conditional entropy of the generative distribution knowing the requested class. In terms of information theory it exactly means building distributions which is uncertain enough. It leads to the following regularized objective.

$$\mathbb{E}_{x\sim\pi_{\theta_y}}\left[H(C(x),y)\right]-\beta H_{x\sim\pi_{\theta_y}}(x|Y=y)$$

Where β models the trade-off between exploration and precision of the samples.It has been shown for GANs that this kind of regularization allows the generator to model larger part of the source distribution without focusing only on some samples.

5.4 Reparametrization trick

As we have seen in Section 5.2, our goal is to minimize $\mathbb{E}_{x\sim\pi_{\theta_y}}\left[l(C(x),y)\right]$ over θ_y for each class y. Therefore we need to compute:

$$\begin{aligned}\nabla_{\theta_y}\mathbb{E}_{x\sim\pi_{\theta_y}}\left[l(C(x),y)\right] &= \nabla_\theta\Big[\int_z \pi_{\theta_y}(z)l(C(x),y)dx\Big]\\ &= \int_z \nabla_\theta\Big[\pi_{\theta_y}(z)l(C(x),y)\Big]dx\\ &= \int_z l(C(x),y)\nabla_\theta\pi_{\theta_y}(z)dx\end{aligned}$$

However, $\int_z l(C(x),y)\nabla_\theta\pi_{\theta_y}(z)dx$ is not tractable in that shape. The reparametrization trick proposes to define s as a function of a random seed of the parameters. Let $\epsilon\sim p(\epsilon)$ be a base (or source) distribution and g_θ a function that takes a random seed and transforms it.

Let $x=g_{\theta_y}(\epsilon)$ be our new random variable, we can rewrite the computation as follows:

$$\nabla_{\theta_y}\mathbb{E}_{x\sim\pi_{\theta_y}(x)}\left[l(C(x),y)\right]=\nabla_{\theta_y}\mathbb{E}_{\epsilon\sim p(\epsilon)}\left[l(C(g_{\theta_y}(\epsilon)),y)\right]\qquad=\mathbb{E}_{\epsilon\sim p(\epsilon)}\left[\nabla_{\theta_y}l(C(g_{\theta_y}(\epsilon)),y)\right]$$

In this shape we can differentiate the function that produces our samples (and thus backpropagate through our probability distribution) by sampling only on the source distribution which is generally a Gaussian distribution or in some case a Gumbel distribution as we will see (Section 5.5).

5.5 Categorical method

Since *MNIST* is made of gray scale images, with a strong contrast we will use a model to decide which pixels need to switch on to get an image of a given class according to the model. The idea is to explore the space of possible images, for that matter we use the Gumbel softmax trick to sample images without to actually having to explore the whole space. First we will dive into the Gumbel distributions and describe precisely how this instance of the reparametrization trick works, and finally we will present the model we propose with some examples of the samples it produces.

5.5.1 Gumbel distribution

The Gumbel distribution is a distribution used to model extreme phenomenon, especially maxima. Here, we will use it to sample discrete random variable while being able to differentiate the process. It has interesting properties since it can be easily sampled and is tractable.

Definitions and properties

Definition 16 (Gumbel distribution)**.** The Gumbel distribution as $\mathbb{R}$ for support and can be parametrized by its location μ and scale β which defines its cumulative distribution function:

$$\forall x \in \mathbb{R}\ F(x) = \exp\left(-\exp\left(-\frac{(x-\mu)}{\beta}\right)\right)$$

If $\mu = 0$ and $\beta = 1$, it gives the standard Gumbel distribution.

Proposition 5. *Let G a standard gumbel random variable and $\alpha \in \mathbb{R}$, $X = G + \log(\alpha)$ a Gumbel law with location $\log(\alpha)$.*

We have the cumulative distribution function of X: $\forall x \in \mathbb{R}\ F_X(x) = \exp(-\exp(-x + \log\alpha))$.

In what follows we will denote $F_{log(\alpha)}$, $\alpha \in \mathbb{R}$ the cumulative distribution function of the Gumbel density with scale 1 and location $\log(\alpha)$. We also denote $f_{\log(\alpha)}$ the associated probability density function.

Proposition 6 (Product of the cumulative distribution function)**.** *1. Let $\alpha_1, \alpha_2 \in \mathbb{R}$, $F_{\log(\alpha_1)}F_{\log(\alpha_2)} = F_{\log(\alpha_1)+\log(\alpha_2)}$*

2. Let $\alpha_1, \cdots, \alpha_n \in \mathbb{R}$, $\Pi_{i=1}^n F_{\log(\alpha_i)} = F_{\sum_{i=1}^n \log(\alpha_i)}$

Proof. For 1: $\forall x \in \mathbb{R}$:

$$\begin{aligned} F_{\log\alpha_1}(x)F_{\log\alpha_2}(x) &= \exp(-\exp(-x)\alpha_1 - \exp(-x)\alpha_2) \\ &= \exp(-\exp(-x)(\alpha_1+\alpha_2)) \\ &= F_{\log(\alpha_1+\alpha_2)}(x). \end{aligned}$$

We show 2 by induction using1 as induction hypothesis. □

Simulation

Theorem 1. *Let F_X be the cumulative distribution function of the real random variable X. Let G defined on $]0,1[$, be the pseudo-inverse de F_X:*

$$\forall \omega \in]0,1[\ \ G(\omega)) = \inf\{x \in \mathbb{R} \mid F(x) \geqslant \omega\}$$

Then, $G(U)$, with $U \sim \mathcal{U}(0,1)$, is a random variable whose cumulative distribution function is F, hence $G(U)$ follows the same distribution as X.

This is a fundamental result in random variable simulation. We can easily sample using a tractable base random variable such as the uniform distribution. In the case of the Gumbel distribution the cumulative distribution function is invertible and therefore its pseudo-inverse is simply its inverse which can be derived explicitly: $\forall x \in \mathbb{R}, F^{-1}(x) = -\log(-log(x)))$.

Discrete distribution sampling

Theorem 2 (Discrete distribution sampling[16])**.** *Let X a discrete random variable with support $[|0, n|]$, $n \in \mathbb{N}$ we denote $\alpha_1, \cdots, \alpha \propto \mathbb{P}[X = 0], \cdots, \mathbb{P}[X = n]$ (ie. the the probabilities of each event are known up to a normalazing constant $S = \sum_i \alpha_i$). Let $(G_k)_{k \leqslant n}$ be a sequence of independent identically distributed standard gumbel variables.*

X and $\arg\max_k(\log \alpha_k + G_k)$ follow the same distribution.

Proof. (Theorem 2)

We denote by K the index of the maximum (the $\arg\max$). The random variables $A_k = \log(\alpha_i) + G_i$ are Gumbel variables with location $\log(\alpha_i)$ and scale 1. They are characterized by their cumulative distribution function $F_{log(\alpha_i)}$.

Let $k \in N$, $g_1, \cdots, g_n \in \mathbb{R}$,

First we write down the joint distribution to make it in a suitable shape.

$$\begin{aligned}
\mathbb{P}[K = k, A_1 = g_1, \cdots, A_n = g_n] &= \prod_{i=1}^{n} f_{\log \alpha_i}(g_i) \mathbb{1}_{[g_k \geqslant g_i]} \\
&= \frac{F_{\log \sum_{i \neq k} \alpha_i}(g_k)}{F_{\log \sum_{i \neq k} \alpha_i}(g_k)} \prod_{i=1}^{n} f_{\log \alpha_i}(g_i) \mathbb{1}_{[g_k \geqslant g_i]} \\
&= \frac{f_{\log \alpha_k}(g_k) F_{\log \sum_{i \neq k} \alpha_i}(g_k)}{F_{\log \sum_{i \neq k} \alpha_i}(g_k)} \prod_{i \neq k} f_{\log \alpha_i}(g_i) \mathbb{1}_{[g_k \geqslant g_i]} \\
&= \frac{f_{\log \alpha_k}(g_k) F_{\log \sum_{i \neq k} \alpha_i}(g_k)}{\prod_{i \neq k} F_{\log \alpha_i}(g_k)} \prod_{i \neq k} f_{\log \alpha_i}(g_i) \mathbb{1}_{[g_k \geqslant g_i]} \\
&= f_{\log \alpha_k}(g_k) F_{\log \sum_{i \neq k} \alpha_i}(g_k) \prod_{i \neq k} \frac{f_{\log \alpha_i}(g_i) \mathbb{1}_{[g_k \geqslant g_i]}}{F_{\log \alpha_i}}(g_k)
\end{aligned}$$

$$\begin{aligned}
\mathbb{P}[K = k, A_1 = g_1, \cdots, A_n = g_n] &= f_{\log \alpha_k}(g_k) F_{\log \sum_{i \neq k} \alpha_i}(g_k) \prod_{i \neq k} \frac{f_{\log \alpha_i}(g_i) \mathbb{1}_{[g_k \geqslant g_i]}}{F}_{\log \alpha_i}(g_k) \\
&= \exp(-g_k + \log \alpha_k) F_{\log \alpha_k}(g_k) F_{\log \sum_{i \neq k} \alpha_i}(g_k) \prod_{i \neq k} \frac{f_{\log \alpha_i}(g_i) \mathbb{1}_{[g_k \geqslant g_i]}}{F_{\log \alpha_i}(g_k)} \\
&= \exp(-g_k + \log \alpha_k) F_{\log S}(g_k) \prod_{i \neq k} \frac{f_{\log \alpha_i}(g_i) \mathbb{1}_{[g_k \geqslant g_i]}}{F_{\log \alpha_i}(g_k)}
\end{aligned}$$

This leads us to the following form whose we are able to extract what we need. In particular, we remark that the $\arg\max$ and the max are independent.

$$p(K = k, A_1 = g_1, \ldots, A_n = g_n) = \frac{\alpha_k}{S} f_{\log S}(g_k) \prod_{i \neq k} \frac{f_{\log \alpha_i}(g_i) \mathbb{1}_{[g_k \geqslant g_i]}}{F_{\log \alpha_i}(g_k)}$$

We eventually got the result: $\mathbb{P}(K = k) = \frac{\alpha_k}{S}$. $\square$

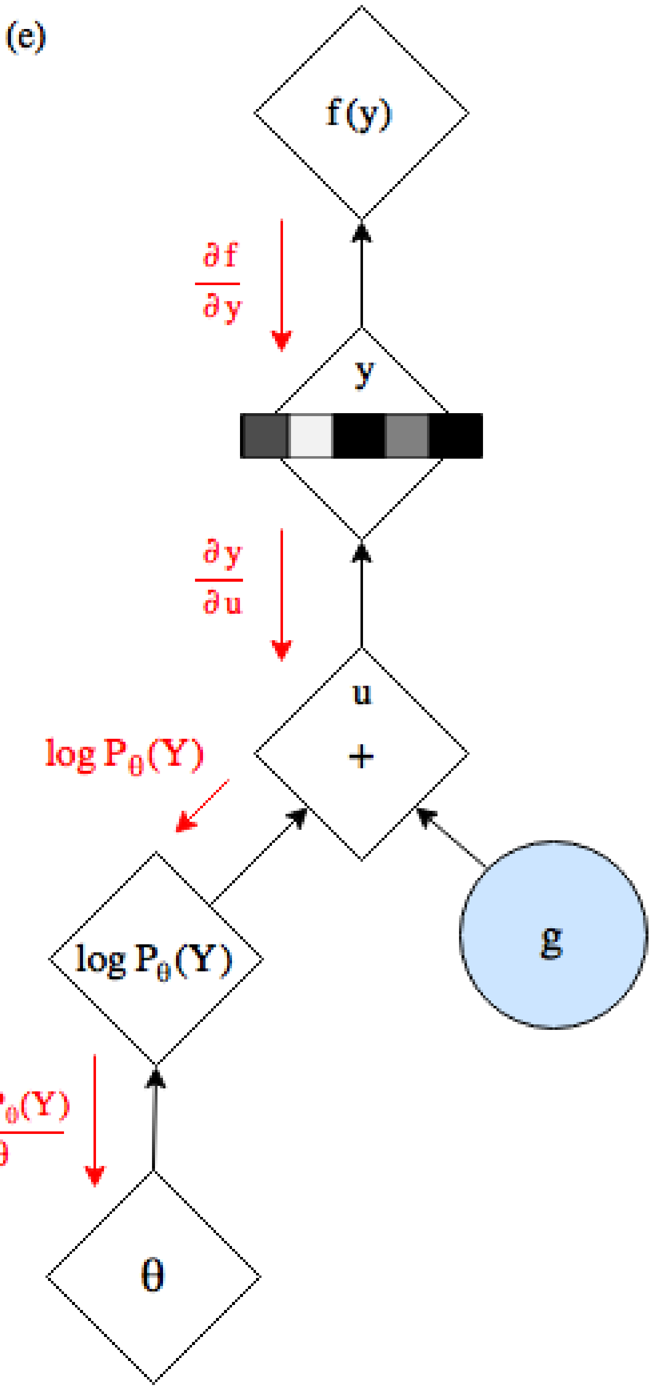

(e)
f(y)
$\frac{\partial f}{\partial y}$
y
$\frac{\partial y}{\partial u}$
u
+
$\log P_\theta(Y)$
$\log P_\theta(Y)$
g
$\frac{\partial \log P_\theta(Y)}{\partial \theta}$
θ

Figure 8 – Computation graph of the gumbel softmax trick

Continuous relaxation of the maximum The arg max function is not differentiable and thus all our work is still not enough to allow backpropagation through this random node. We will use a continuous relaxation of the problem. Instead of considering the arg max function as a function that gives us an index we will work in the simplex and consider the softmax function. The relaxed-arg max is a function that returns the index under the form of a one-hot vector, whose hot spot is at the arg max. The convex hull of these vectors is the simplex:

$$\mathbb{S}^{n-1} = \left\{ x \in \mathbb{R}_+^n, \sum_{k=1}^{n} x_k = 1 \right\}$$

Therefore, we will allow our variable to take their values into this set as a relaxation of the problem and we will use a softmax mapping (parametrized by a temperature) to extract the arg max as a soft onehot vector:

$$\text{softmax}_\tau(x)_k = \frac{\exp\left(x_k/\tau\right)}{\sum_{k=1}^{n} \exp\left(x_k/\tau\right)}$$

with this definition we can define (instead of the discrete valued random variable X) the sequence of simplex-valued random variables:

$$X^\tau = \left(X_k^\tau\right)_k = \text{softmax}_\tau(\log\alpha + G) = \left(\frac{\exp\left(\left(\log\alpha_k + G_k\right)/\tau\right)}{\sum_{i=1}^{K} \exp\left(\left(\log\alpha_i + G_i\right)/\tau\right)} \right)_k$$

This mapping is differentiable and allows us eventually to sample from a discrete distribution while maintaining our ability to backpropagate through it and to train our model by classical gradient descent methods.

5.5.2 Categorical model for *MNIST*

Since *MNIST* is composed of gray scale images we want to decide which pixel should be alight for an image. We want to learn a conditional generator which will produce images from each class. To achieve this goal we will build a neural network that will produce a probability map. For each pixel of the image it will give the probability that this particular is alight knowing we want to produce an image of class k. When we have this probability map we sample K pixel to switch on to produce the final image, using the previously seen Gumbel softmax trick to sample from this discrete distribution.

More formally, we define a conditionnal parametric distribution over the pixels of an image of size 28×28 (the size of an *MNIST* image) $\pi_{\theta_y}(P_{ij})$, where P_ij is the event "Pixel at coordinate (i, j) is alight knowing that we want to produce an image of class y. Using the Gumbel softmax trick we produce K images from this probability map. These images are one-hot images, which means they have only one pixel with a high luminosity. In order to produce the final image we sum these vectors and clip the values to keep them inside $[0, 1]$.

In our experiments we use the probability maps over the pixels for each class as a proxy for the distribution of the generated samples. This allows us to easily evaluate the measures described in Section 2.6. Indeed, under this setting all the mesures can be computed efficiently, pixel wise but the wasserstein distance which we approximate using the slice wasserstein algorithm (Section 2.6.2). In this case we call "concepts" the probability maps over the pixels. For the sake of simplicity we denote in this case $(\,map_y)_y$ the probability distribution over the pixels learnt by a given generator for a requested class y. $\text{map}_y(i, j)$ is the probability of pixel (i, j) to be alight.

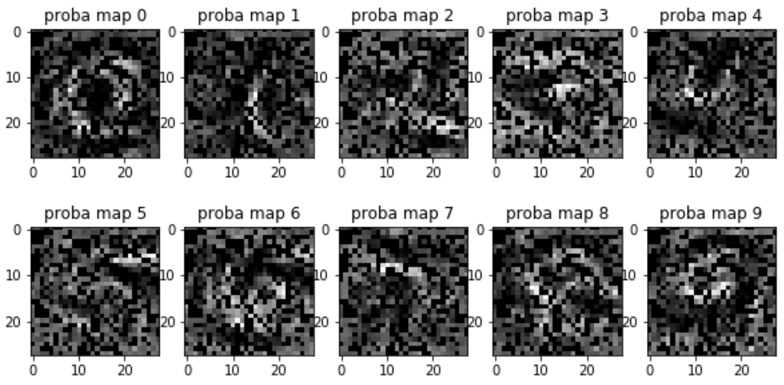


Figure 9 – Example of probability maps over the pixels for generators for each class of *MNIST* train on a usual classifier

Entropy The entropy of the concept of y is simply the entropy of the associated probability map

$$H(\mathrm{map}_y) = -\sum_i \sum_j \mathrm{map}_y(i,j)\log_2(\mathrm{map}_y(i,j))$$

Bhattacharyya coefficient The Bhattacharyya coefficient, which we refers only as BC for obvious reasons, between two concepts has a simple form in this case. For two classes y, y' and their learnt concepts $\mathrm{map}_y, \mathrm{map}'_y$

$$BC(\mathrm{map}_y, \mathrm{map}'_y) = \sum_{i,j} \sqrt{\mathrm{map}_y(i,j)\,\mathrm{map}'_y(i,j)}$$

. We take advantage of this to compute the Hellinger distance

$$H(\mathrm{map}_y, \mathrm{map}'_y) = \sqrt{1 - BC(\mathrm{map}_y, \mathrm{map}'_y)}$$

Wasserstein distance For two classes y, y' and their learn concepts $\mathrm{map}_y, \mathrm{map}'_y$, we compute the earth-moving distance between the two gray scale images formed by the probability maps.

Probability density function In order to evaluate the probability of a sample being generated from one of the distribution we compute the product of the probabilities of the alight pixels to be alighten. More precisely, if I is the sample and map_y the probability maps corresponding over the pixels for the class y we compute $\prod_{ij} I_{ij}\,\mathrm{map}_y(i,j)$. It is the probability that all the alight pixels from I would have been set alight according the the probability distribution.

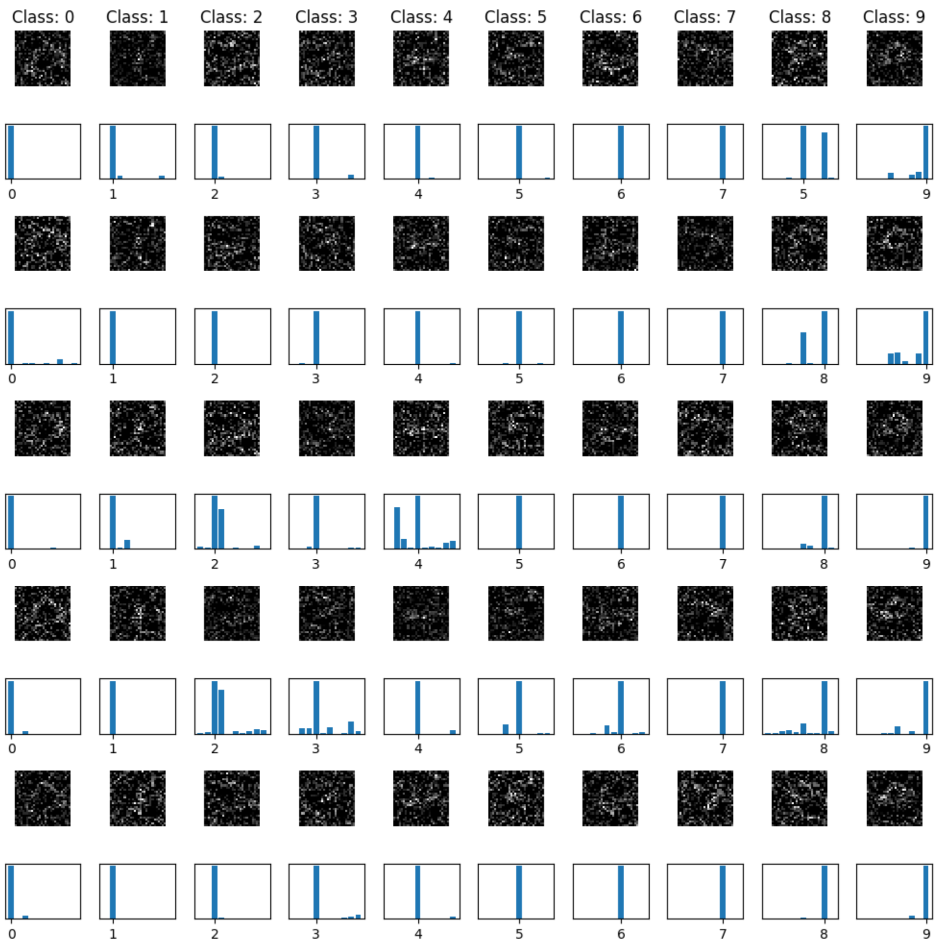


Figure 10 – Samples for each class generated by the discrete gumbel generative distributions with 500 points and their classification according to the targeted classifier.

5.6 Multivariate normal distribution

The parametrization of multivariate gaussian by learnt mean and covariance matrix is a well known method mainly used for variational autoencoders [1, 17, 31] in combination with the reparametrization trick, indeed it has a quite straightforward formulation in that case.

Reparametrization . The reparametrization (and thus simple sampling) of a multivariate normal distribution is based on the properties of normal random vectors. A random vector X is said to be a normal random vector of mean μ and covariance matrix Σ, if and only if there exist Z a random vector such that $\forall_i, Z_i \sim \mathcal{N}(0,1)$ and a matrix A such that $X = AZ + \mu$ with $\Sigma = A^t A$. Hence, in order to learn a multivariate gaussian distribution, one only needs to learn a matrix A and a mean μ and transform a normal random seed using the formula above to sample from it.

We can therefore train normal conditional distributions to produce samples that are well classified by the classifier and use the properties of normal distributions to compute the metrics presented in Section 2.6.

Entropy The entropy of a multivariate gaussian of covariance matrix Σ is given by $\ln(\sqrt{(2\pi e)^N |\Sigma|})$. Remark that it is definite only if Σ is invertible, *ie.* the distribution is non degenerate.

Hellinger distance The squared Hellinger distance between two multivariate normal distributions $P \sim \mathcal{N}(\mu_1, \Sigma_1)$ and $Q \sim \mathcal{N}(\mu_2, \Sigma_2)$ is

$$H^2(P,Q) = 1 - \frac{\det\left(\sum_1\right)^{1/4} \det\left(\sum_2\right)^{1/4}}{\det\left(\frac{\sum_1 + \sum_2}{2}\right)^{1/2}} \exp\left\{-\frac{1}{8}\left(\mu_1 - \mu_2\right)^T \left(\frac{\sum_1 + \sum_2}{2}\right)^{-1} \left(\mu_1 - \mu_2\right)\right\}$$

Wasserstein distance . The 2-Wasserstein distance between two normal distributions p, q of means $\mu_1, \mu 2$ and covariance matrices Σ_1, Σ_2 is given by $W_2\left(\mu_1, \mu_2\right)^2 = \|\mu_1 - \mu_2\|_2^2 + \operatorname{trace}\left(\sigma_1 + \Sigma_2 - 2\left(\Sigma_1 \Sigma_2\right)^{1/2}\right)$, assuming that these distributions are non degenerate

Density function In the case of a non-denerate normal random variable $P \sim \mathcal{N}(\mu_1, \Sigma_1)$, we can evaluate its density probability function

$$f_{\mathbf{X}}(x_1, \ldots, x_k) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}}$$

.

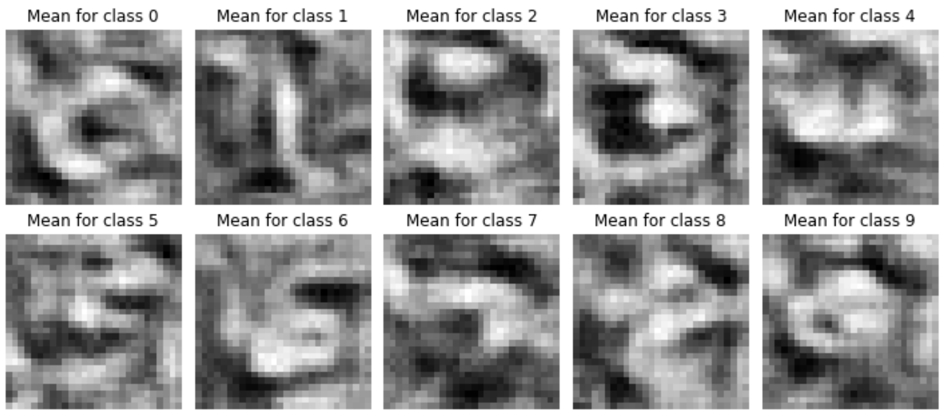


Figure 11 – Means of the normal distribution learnt for each class.

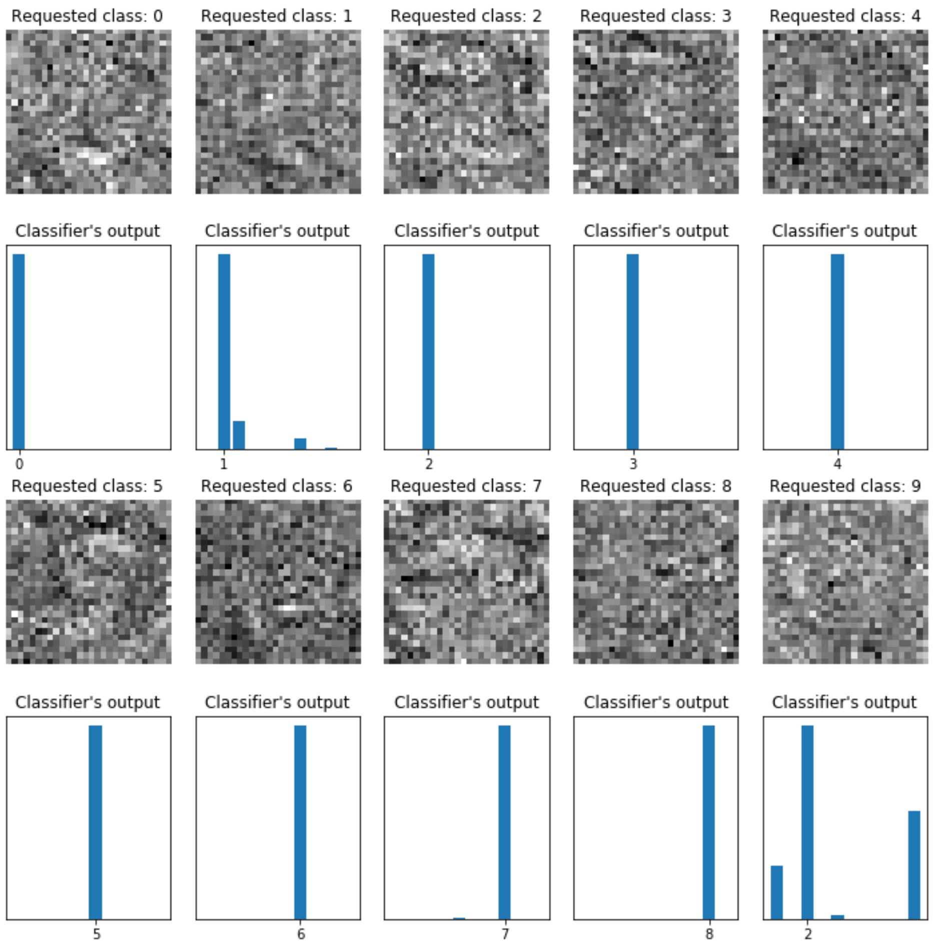


Figure 12 – Samples drawn from the generative distributions and their classification vector output by the classifier

In order to parametrizes a non degenerate multivariate normal distribution we actually need to parametrize a symmetric positive definite matrix, *ie.* an invertible covariance matrix. To do so, we take advantage of the alternative choleski factorization.

Proposition 7. *Let L be a lower triangular matrice with 1 on the diagonal, and D a diagonal matrix, with non-zero coefficient.*

$L^t DL$ is a symetric positive definite matrix.

In our case, we learn the parameters for L and D and use $L^t DL$ as covariance matrix.

5.7 Normalizing flows

Since we are trying here to model probability distribution, we have to discuss the normalizing flows. These are based on the fact that we can derive the density probability of a transformed random variable if the transformation is a diffeomorphism. It has been popularized by Rezende and Mohamed [29] in 2015, but the framework had been already defined in other contexts.

Definition 17 (Recall diffeomorphism)**.** Given two sets E and F, A $f : E \to F$ is called a diffeomorphism if it is a bijection and its inverse $f^{-1} : F \to E$ is differentiable as well. diffeomorphism.

If a mapping is a diffeomorphism, we can talk about the differential of its inverse whose Jacobian matrix is well defined and inversible everywhere.

Definition 18 (Recall Jacobian Matrix)**.** The Jacobian matrix of $\mathbf{f}$ is the matrix of the partial derivative operators $\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

The fundamental idea of normalizing flows is to transformed a known tractable distribution into an arbitrary one, whose we will be able to compute the distribution by reversing the transformation and evaluate the probability of the source distribution. It is actually a generalization of the reparametrization trick.

More formally, let say we have $\mathbf{Z} \in \mathbb{R}^D$ a random variable whose probability density function $p_{\mathbf{Z}} : \mathbb{R}^D \to \mathbb{R}$ is known and tractable. Let f be an invertible function and $\mathbf{Y} = \mathbf{g}(\mathbf{Z})$.

The change of variable theorem assures us that the probability distribution of $\mathbf{Y}$ can be computed using:

$$\begin{aligned} p_{\mathbf{Y}}(\mathbf{y}) &= p_{\mathbf{Z}}(\mathbf{f^{-1}}(\mathbf{y}))|\det \mathrm{D}(\mathrm{f}^{-1}(\mathbf{y})| \\ &= p_{\mathbf{Z}}(\mathbf{f^{-1}}(\mathbf{y}))|\det \mathrm{Df}(\mathbf{f^{-1}}(\mathbf{y}))|^{-1}, \end{aligned}$$

where $\mathrm{Dg(y)} = \frac{\partial \mathrm{g}}{\partial \mathrm{y}}$ denotes the Jacobian of g at point y. This new density function $p_{\mathbf{Y}}(\mathbf{y})$ is known as the pushforward of the density $p_{\mathbf{Z}}$ by the function f and denoted by $\mathrm{f}_* p_{\mathrm{Z}}$.

In that setting f is the generative function that pushes the source density p_Z (also known as the noise), by contrast, its inverse is the normalizing function: the one which retransform the density towards the known density.

The whole point of the field is to design parametric functions f, that can be proved to be invertible and that are expressive enough to model any probability density function. Indeed, it has been proven that under reasonable assumptions we can model any distribution from a source distribution using this method [18]. Moreover, those functions needs to have a jacobian matrix that is tractable to allow the use of the change of variable formula. The main method to achieve these goals, *ie.* building models that are bijections, whose jacobian are tractables and complex enough, is to build up the complexity of a model by composing simple blocks. Indeed, the composition of bijection is bijective, but we also can exploit the fact that the jacobian of a composition is the product of the jacobian matrices.

Let say that $f = f_1 \circ f_2 \circ \cdots, \circ f_k$, then

$$\det \mathrm{Df}(\mathbf{y}) = \prod_{i=1}^{N} \det \mathrm{Df}_i\,(\mathbf{x}_i)$$

where $\mathbf{x_i} = f_i^{-1} \circ \cdots f_1^{-1}(\mathbf{y})$.

Therefore, we can focus on desining simple blocks which meet the tractability requierements and whose composition will form a complexe model. We will present some of these blocks and propose a new one.

5.7.1 Elementwise flows

The most naive method to build a bijective function on vectors is to apply a bijective function along each coordinate. In machine learning bijective activation function enter this category. However, they are not powerful enough by themselves to model complex distribution.

5.7.2 Linear transformations

Modern machine learning models are composition of linear transformations and activation functions but these linear transformation are not guaranteed to be invertible in general. In order to build normalizing flows we shall parametrize invertible matrices, we also remark that the differential of a linear application is itself, therefore we need the determinant of these matrices to be tractable.

Diagonal matrices The first trivial option are strictly non zero diagonal matrices.

Triangular Another option are triagular matrices with non-zero diagonal coefficients. The determinant of a triangular matrix is the product of its diagonal elements. Thus, diagonal matrices are probably the most straightforward and expressive method to begin to work with normalizing flows. However, their expressiveness is still not enough to model any distribution.

Orthogonal matrices The absolute value of the determinant of an orthogonal matrix is 1, so it is easily tractable and one can parametrize an orthogonal matrix using the *Householder transform.* It states that for any non-zero vector $v \in \mathbb{R}^n$, $H = Id - 2\frac{v^T v}{||v||^2}$ is orthogonal.

Cauchy flows Another option that seems to me quite straightforward but that I could not find anywhere, so I propose it, is to work with Cauchy matrices since their determinant is known explicitly. A Cauchy matrix whose elements a_{ij} are in the form

$$a_{ij} = \frac{1}{x_i - y_j}; \quad x_i - y_j \neq 0, \quad 1 \leqslant i \leqslant m, \quad 1 \leqslant j \leqslant n$$

where x_i and y_j are elements of a field $\mathcal{F}$, and (x_i) and (y_j) are injective sequences. The Hilbert matrix is a special case of the Cauchy matrix, where

$$x_i - y_j = i + j - 1$$

For matrices of that particular form, the determinant is tractable. Indeed, the determinant of a Cauchy matrix A is given by

$$\det \mathbf{A} = \frac{\prod_{i=2}^{n} \prod_{j=1}^{i-1} (x_i - x_j)(y_j - y_i)}{\prod_{i=1}^{n} \prod_{j=1}^{n} (x_i - y_j)}$$

The definition of Cauchy matrices guarantees that this determinant is always non-zero, hence Cauchy matrices are invertible. The parametrization is tricky in practice, since we need to ensure that the sequences (x_i) and y_i are injective and that for any i, j, $x_i - y_j \neq 0$. A simple way to achieve injectivity is to build sequence of positive reals and to considera the cumulative sums. Let say we have $a_1 \cdots a_n$ and $b_1, \cdots b_n$ sequence of positive reals, we take:

$$x_i = \sum_{k=1}^{i} a_k \qquad\qquad y_i = \sum_{k=1}^{i} b_k$$

To ensure the second property, we can carefully choose the intervals from which we take the a_i, b_i. Further investigation is needed on this topic to determine whether is usable in practice and to assess the expressiveness of this class of models since it was not the main topic of the present work.

6 Adversarial attacks and adversarial training

The lack of robustness of machine learning models and more specifically of neural networks is often highlighted by the unreasonable effectiveness of adversarial attacks. These are methods that aim at producing inputs that are very close to natural image but slightly edited in a way that makes the network produce an error.

Those attacks could have dramatic consequences if used against critical systems such as self driven cars. Indeed, some of those attacks can be realized in real life and not only virtually. For example, it is possible to build stickers to apply on road signs to fool self driving car into speeding up instead of slowing down [21].

The fact that those attacks work shows that neural networks tend to have very steep slopes or cliffs. It means that in a small neighborhood of a point that would be correctly classified, there are points that have a completly different classification according to the model. This idea has led to the developpement of numerous works on how to enforce smoothness of neural network [27, 8], for example by regularizing the training loss by the lipschitz constant of the model. If the lipschitz constant is low it ensures that the network is smooth. Actually, the lipschitz constant of a model is theoretically a good way to measure its robustness by providing guarantees of the form: (assuming E, d_E and F, d_F are metric spaces and $f : E \longrightarrow F$ and where k is the Lipschitz constant):

$$\forall (x,y) \in E^2, d_F(f(x), f(y)) \leqslant k d_E(x,y)$$

This would tell us exactly how much a well classified image should be modified to change of classification and therefore it gives us information about the confidence we should have in this prediction. However, it is usually very hard to compute this constant and even harder to use it at training time [12]. One way to train the network to stay smooth around training point at least and to make it more robust against adversarial attacks is to train it against images produced by adversarial attacks. This led to adversarial training [22].

Our goal is to design metrics to assess the robustness of a model. We propose to study the effects of different training methods that are supposed to improve robustness on those metrics. Indeed, we expect our metrics to capture this improvement.

6.1 Adversarial attacks

The goal of adversarial attacks is to produce perturbations that are small enough not to be noted but that leads the classifier to missclassify the example: it is of class Y for a human expert without any doubts but the invisible small perturbation fools the network to think otherwise. A way to keep the perturbation small is to find the example which fools the most the classifier inside a ball centered on the source image. To find such an adversarial example we use the projected gradient descent algorithm.

6.1.1 Projected gradient descent

Projected gradient descent is a method to solve a convex constrained optimization problem of the form:

$$\min_x \quad f(x) + i_C(x)$$

where i_C is the indicator function associated with the constraint, *ie.*:

$$i_C(x) = \begin{cases} 0 & \text{if } x \text{ verifies C} \\ \infty & \text{otherwise} \end{cases}$$

We will explain here the core principles that lead to the projected gradient descent algorithm and show that it converges in the case of convex problems.

The first order condition tells us that $x^\dagger \in C$ is a minimizer if and only if $0 \in \partial (f + i_C) \left(x^\dagger\right)$. We also know that the subdifferential of a sum contains the sum of subdifferentials (Hence if $0 \in \nabla f \left(x^\dagger\right) + \partial i_C \left(x^\dagger\right)$ then $x^\dagger$ is a minimizer.

by definition of a subgradient, if y is a subgradient of i_C at a point $x \in C$:

$$i_C(z) \geqslant i_C(x) + \langle z - x, y \rangle, \quad \forall z$$

For any $z \notin C$, this inequality is true because the left hand side is infinite. Letting $z \in C$ and as long as $x \in C$ so that $i_C(x) = i_C(z) = 0$, we end up with

$$0 \geqslant \langle z - x, y \rangle, \quad \forall z \in C$$

This inequality does not only define the subdifferential ∂i_C but it defines the normal cone to the convex set C at x. We denote this cone $N_C(x)$.

To sum up,

$$\partial i_C(x) = \{y \in \mathbb{R}^n \mid \langle z - x, y \rangle \leqslant 0, \forall z \in C\} = N_C(x)$$

Finally we get as final implication,

$$0 \in \nabla f\left(x^\dagger\right) + N_C\left(x^\dagger\right) \Longrightarrow x^\dagger \in \arg\min_{x \in C} f(x)$$

We will use this equation to design a fixpoint iteration to actually find a proper $x^\dagger$ which will solve our optimization problem.

The whole of the projected gradient descent is to take the projection of the solution onto the feasible set in order to keep our solution feasible. We can link the the normal cone to the normal projection.

The orthogonal projection onto C (denoted π_C) is defined as the following optimization problem for any $z \in \mathbb{R}^n$:

$$\pi_C(z) = \arg\min_{x \in C} \frac{1}{2}\|x - z\|_2^2$$

Using the implication we derived we can rephrase, if:

$$0 \in \left(x^\dagger - z\right) + N_C\left(x^\dagger\right)$$

then $x^\dagger$ is a minimiser. And consequently, $x^\dagger = \pi_C(z)$.. But this is the same as requiring $z \in x^\dagger + N_C\left(x^\dagger\right)$ or, equivalently, $z \in \left(\mathbb{I} + N_C\right)\left(x^\dagger\right)$ ..

Eventually, we can replace $x^\dagger$ by $\pi_C(z)$ and re-arranging terms gives

$$\pi_C(z) = \left(\mathbb{I} + N_C\right)^{-1}(z)$$

We here get the classical relationship, if C is a convex subset of $\mathbb{R}^n$ then we can define the orthogonal projection onto it by:

$$\pi_C \equiv \left(\mathbb{I} + N_C\right)^{-1}$$

From this, we can now design the fixpoint expression which underlies the projected gradient algorithm.

But first let remark that the normal cone is invariant to a non-negative scalar multiplication, *ie.*:

$$\text{if } u \in N_C(x) \text{ then } \alpha u \in N_C(x), \forall \alpha \geqslant 0$$

We can rewrite this property: for any $\alpha \geqslant 0$ and $u \in N_C(x)$

$$(x + \alpha u) - x \in N_C(x)$$

If we define $z := (x + \alpha u)$ we can now write that $z \in \left(\mathbb{I} + N_C\right)(x)$. But, we showed earlier that $\pi_C \equiv \left(\mathbb{I} + N_C\right)^{-1}$, hence this is equivalent to $x = \pi_C(z)$.

Equivalently we get,

$$x = \pi_C(x + \alpha u), \quad (\alpha \geqslant 0, u \in N_C(x))$$

The first order condition gives us that if $x^\dagger$ is such that $-\nabla f\left(x^\dagger\right) \in N_C\left(x^\dagger\right)$ then $x^\dagger$ is a minimiser. This eventually gives us the fixedpoint iteration iteration we were looking for:

$$x^\dagger = \pi_C\left(x^\dagger - \alpha \nabla f\left(x^\dagger\right)\right)$$

From this fixedpoint form we derive the algorithm which start from point x_0 and generates the sequence $\{x_0, x_1, \ldots\}$ using

$$x_{k+1} = \pi_C \left(x_k - \alpha_k \nabla f \left(x_k \right) \right)$$

where $\alpha_k > 0$ is the step size.

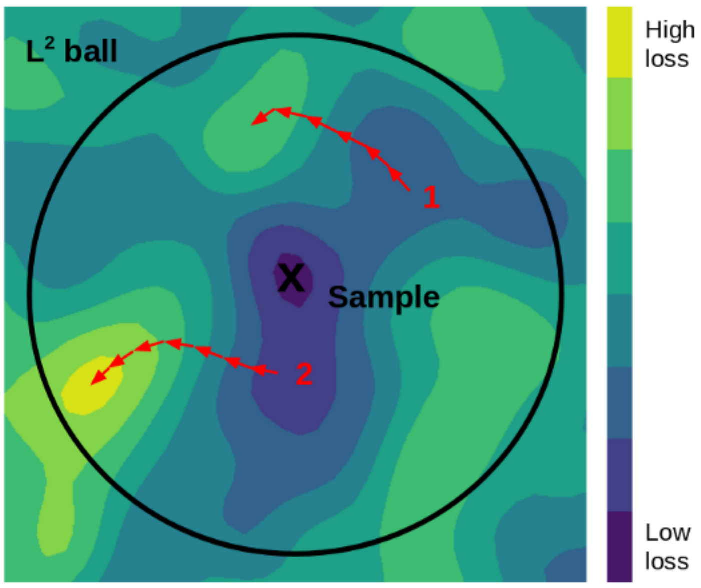


Figure 13 – Projected gradient descent

6.1.2 Targeted attacks and untargeted attacks

Targeted attacks consist of finding a perturbation that leads to misclassify the adversarial example in a controled way: we construct the adversarial example to make the classifier believe that it is of a given class y. It is usually made by minimizing the crossentropy between the wanted class and the prediction given by the network. Untargeted attacks on the other hand do not have a specific target it just tries to induce a mistake knowing the true class. We will use here the latter.

6.2 Adversarial training

We have seen that neural networks are subject to adversarial attacks and that they can easily be fooled. They are fooled by some images which are built by taking advantage of the major slopes in the model. A way to avoid this pittfall – at least to alleviate its effects – is to train the network to classify correctly adversarial examples. The idea is during training to feed the network not only with true images but to generate synthetic data using adversarial attacks. This has the effect of increase the smoothness of the network in the neighborhood of the training point and therefore it reduces the risk of import slopes that lead to adversarial weekness [22].

6.3 Lipschitz regularization

As we said previously, most of adversarial attacks takes advantage of the steep slopes of the training model. If we could reduce them as much as possible we would probably alleviate the efficiency of adversarial attacks. A way to achieve this is to regularize the objective function used to train the model this the Lipschitz constant. Indeed, the lipschitz constant is the max of the derivative of the the function. This kind of regularization has been studied and led to some improvements. These methods have been widely studied and seem to perform relatively well [5]. However, because of time constraints we could not test them here and it should be taken care of in following works.

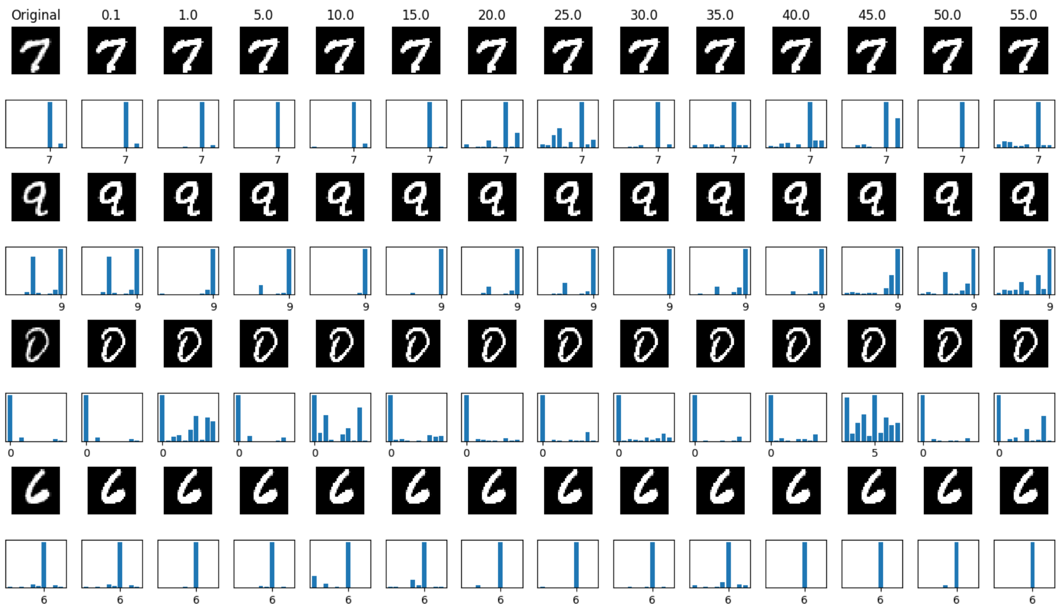


Figure 14 – Untargeted adversarial attacks on a *MNIST* classifier with different sizes of ball using the L_1-norm

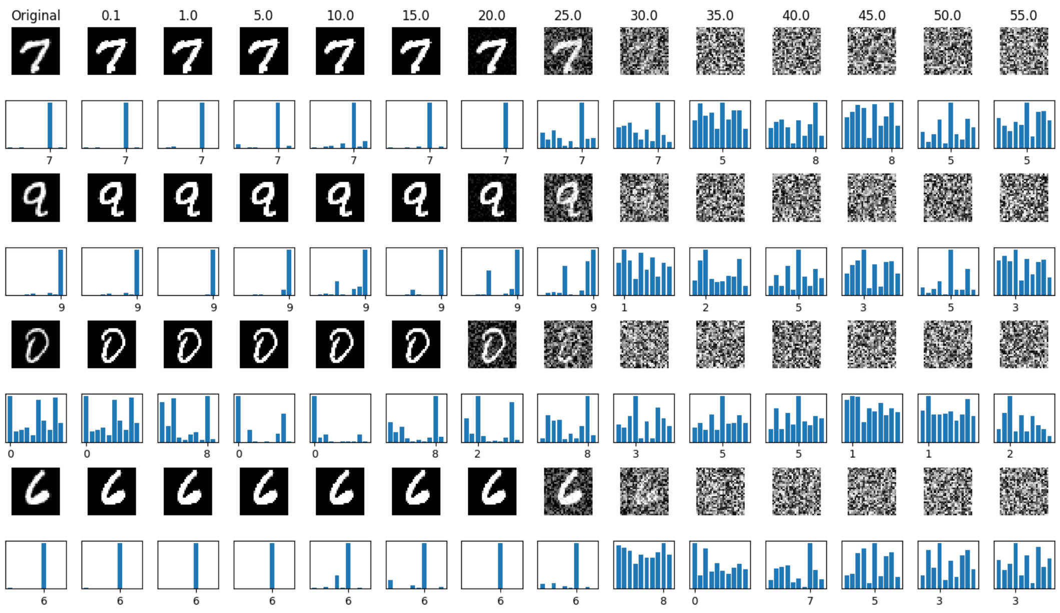


Figure 15 – Untargeted adversarial attacks on a *MNIST* classifier with different sizes of ball using the L_2-norm

7 Experiments and results

Using the models described in Section 5, I learned the generative distributions associated to classifiers trained with different types of adversarial training and no adversarial training at all. The goal was to produce a proof of concept first, that we can actually learn some form of reverse/generative distribution associated to a classifier and then use them to make classifications and measure a good notion of robustness.

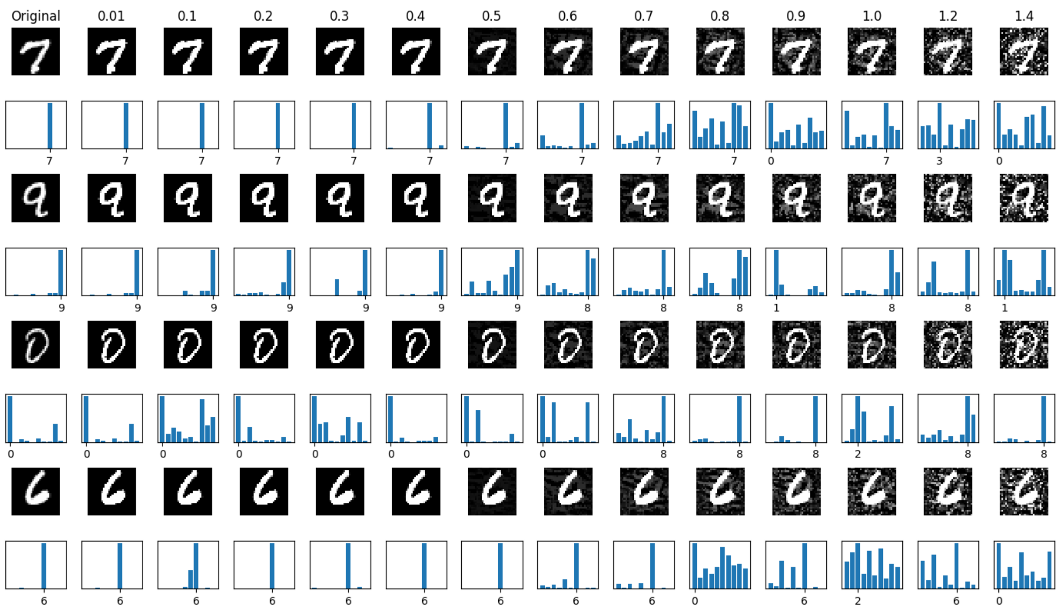


Figure 16 – Untargeted adversarial attacks on a *MNIST* classifier with different sizes of ball using the L_{inf}-norm

7.1 Global experimental settings

In order to show a correlation between the ball used during adversarial training and the notions of robustness I propose (Section 1.2), I trained 100 classifiers and their associated generative distributions for each ball size and attack type. The following plots show the mean and the standard deviation over these runs for each combination of parameters. I mainly present here the results I got using the L_{inf} norms for the adversarial training because as seen previously (Section 6.1.2) the L_2 and L_1 norms produce noisy adversarial examples whereas the L_{inf} attacks tend to produce more sharp and macroscopic attacks. Moreover, attacks L_1 using L_1 norm produce little or no effects at all. I was not able to produce the results for the multivariate normal distributions models for computational cost reason. Indeed, for each covariance matrices we have 784×784 parameters, for each of the ten classes, for 100 runs and around 45 combinations of ball size and attack type, in float32 norms, it would take around 110Go of data and several weeks of computation time, even machines with accelerators. Therefore, only some qualitative examples are presented.

Algorithm 3 General algorithm used to produces the statistics

```
for all attack $\in$ Attacks do
  for all $p \in$ Parameters do
    for $k = 1$ to 100 do
      Train classifier $C$ on *MNIST*
      Train generative distributions $G_C$ of $C$
      Produces the measures and saves them.
    end for
  end for
end for
```

Attack type	Ball sizes
L_{inf} PGD Attack	$0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1., 1.2, 1.4, 1.6$
L_1 PGD Attack	$0.1, 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55$
L_2 PGD Attack	$0.1, 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55$

Figure 17 – Adversarial training parameters

The model we use is the example model used in any *MNIST* classification problem examples, it is a convolutional model piped into a dense network to make the prediction. This model reaches around 95% accuracy on the *MNIST* test set. The use of a convolutional network is here crucial because it is *a priori* less sensitive to adversarial attacks and produces more robust models whereas dense neural network tend to have blind spots for the weights associated to always zero pixels. Therefore, the metric I propose should capture that dense network are less robust than convnet. I produced the same experiment but with a dense network as classifier and showed that the distances between the generative distributions is indeed smaller for the dense networks.

```python
class mnistConv(nn.Module):
    def __init__(self, device='cpu'):
        super(mnistConv, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5).to(device)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5).to(device)
        self.conv2_drop = nn.Dropout2d().to(device)
        self.fc1 = nn.Linear(320, 50).to(device)
        self.fc2 = nn.Linear(50, 10).to(device)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

Figure 18 – Classification model for *MNIST*

It is important to point out that the training of the generative distributions associated to the classifier is quite time consuming and that running all the experiments to get significant results takes a couple days of training, especially for the multivariate gaussian based distributions.

7.1.1 Gumbel discrete distributions

We train a neural network that takes as input a one-hot vector representing the requested class and this network produces the probility map. We train it for 100 iterations using batches of length 256 and we build each images by sampling 500 pixels from the probability maps (For examples see Figure 10).

7.1.2 Multivariate normal distributions

7.2 Classification using generative distributions

In order to show the relevance of the learned generative distributions, I evaluate their classifications abilities (See Figure 19) and I take the opportunity to check whether adversarial learning improves it. Even though the results are not that sharp, It seems adversarial training does indeed improve the accuracy of the classification done using the generative distributions as we can see on Figure 20. Evaluating this classification method on the *MNIST* test dataset we found out that it reaches 70% accuracy which is significantly lower than the base classifier, however, it is not that bad considering it learns only from what the classifier had learnt and therefore never sees any true data before the test.

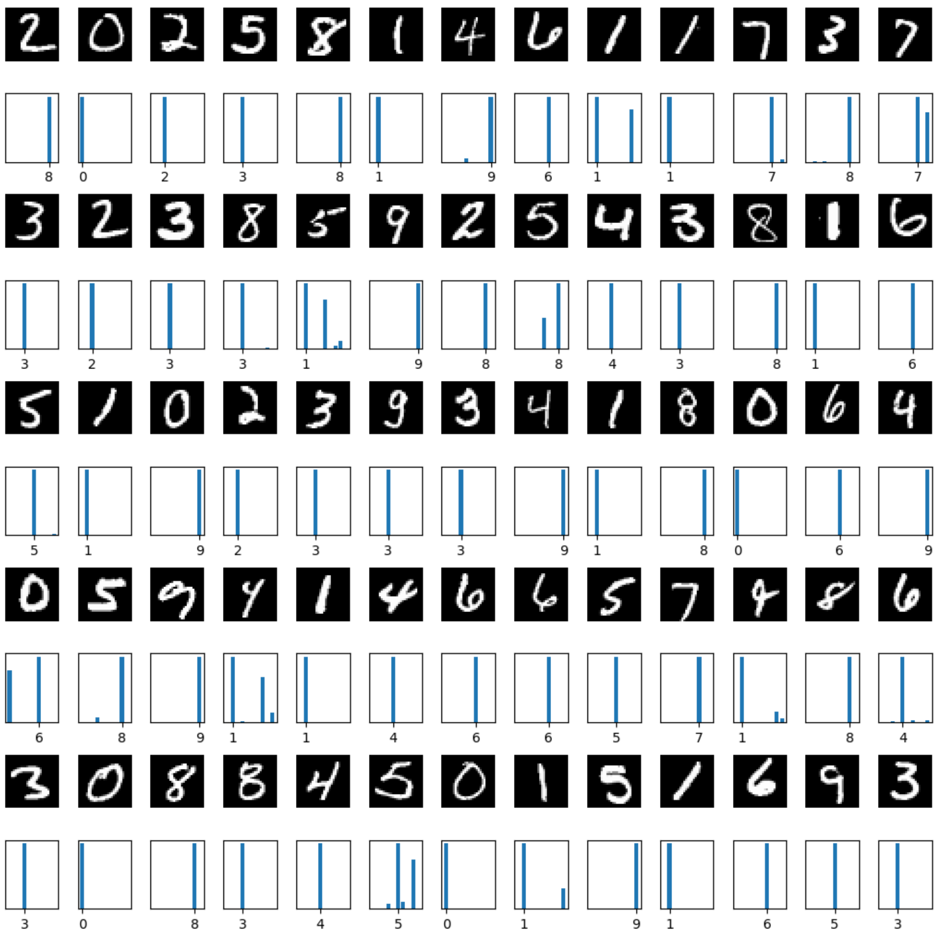


Figure 19 – Classification using the generative distribution associated to classifiers trained with L_{inf} PGD-Attacks with different ball sizes.

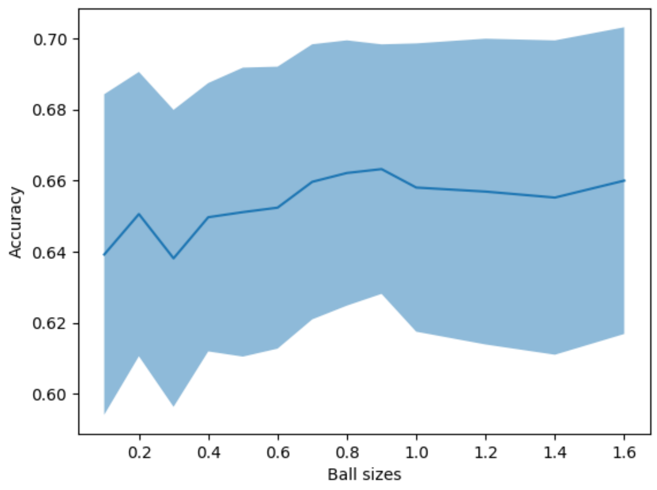
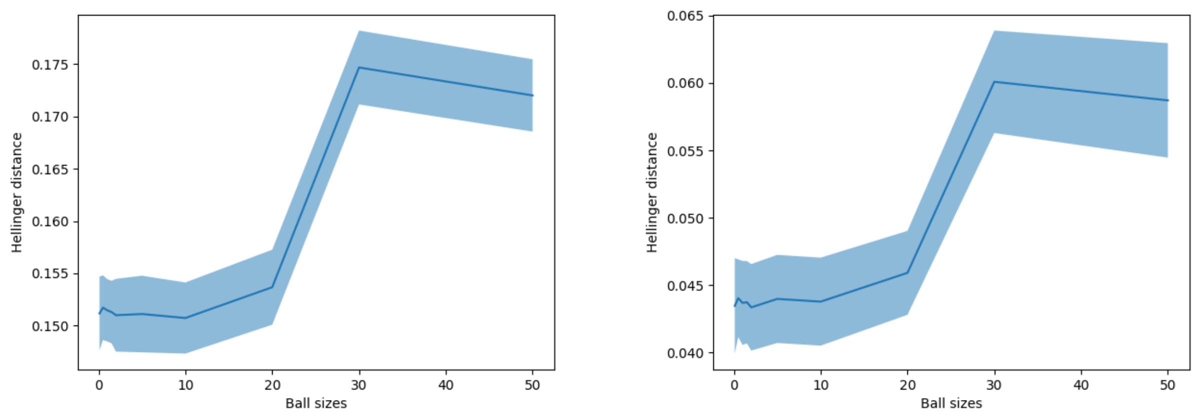


Figure 20 – Impact of the adversarial training on the accuracy of the classification based on generative distribution.

7.3 Robustness evaluation

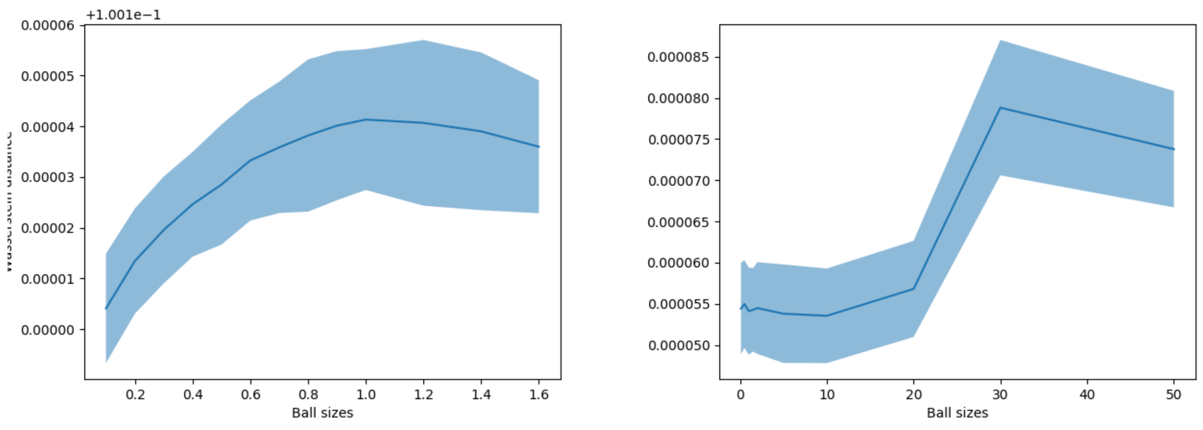
We have been able to show a correlation between the size of the ball use to adverarially train the classifier and the distance between concepts for the L_1 and L_{inf} norms and training the generative distribution to produce sharp samples (See Figures 21, 21, 22, 24 25) but we also got similar results with generators trained to fit as most as possible the margins of the classifier.

7.3.1 L_2 PGD Attack



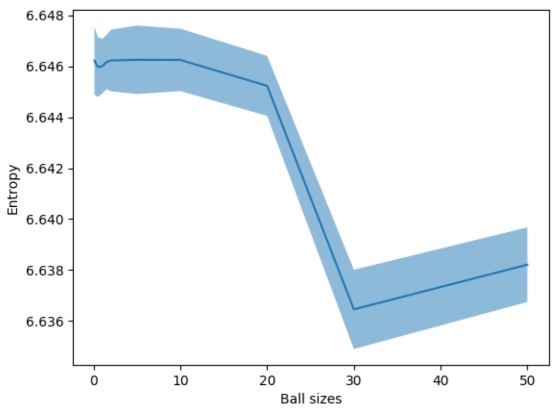
(a) Avergage hellinger distance between concepts (b) Min helinger distance between concepts

Figure 21 – Correlation between the Hellinger distance and the size of the ball used for $L2$-PGDAttack adversarial training

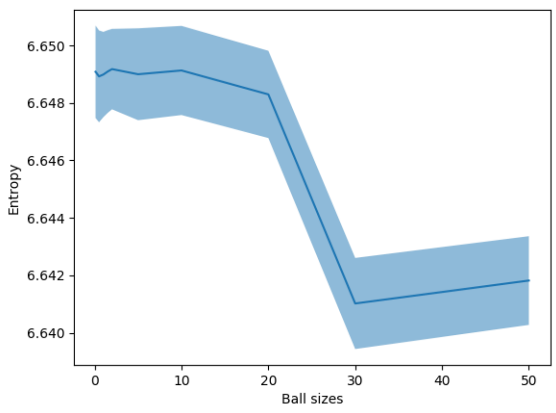


(a) Average wd_2 distance between concepts (b) Min wd_2 distance between concepts

Figure 22 – Correlation between the wasserstein distance (with norm L_2) of the generative distributions and the size of the ball used for $L2$-PGDAttack adversarial training



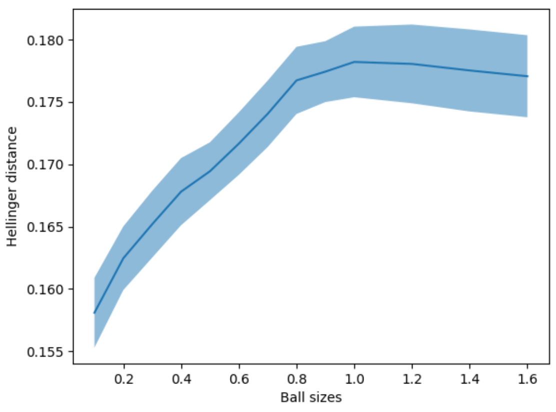
(a) Avergage entropy of the generative distributions



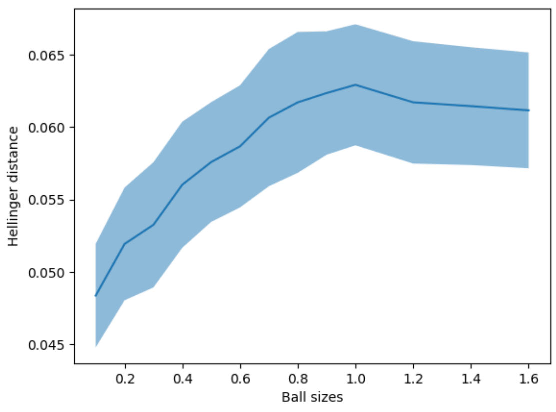
(b) Maximum entropy of the generative distributions

Figure 23 – Correlation between the entropy of the generative distributions and the size of the ball used for $L2$-PGDAttack adversarial training

7.3.2 L_{inf} PGD Attack

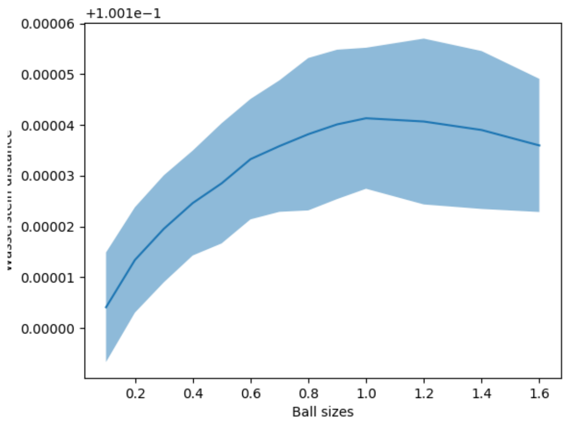


(a) Avergage hellinger distance between concepts

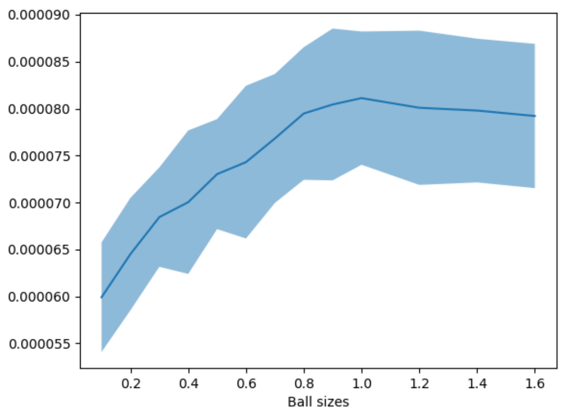


(b) Min helinger distance between concepts

Figure 24 – Correlation between the Hellinger distance and the size of the ball used for L_{inf}-PGDAttack adversarial training

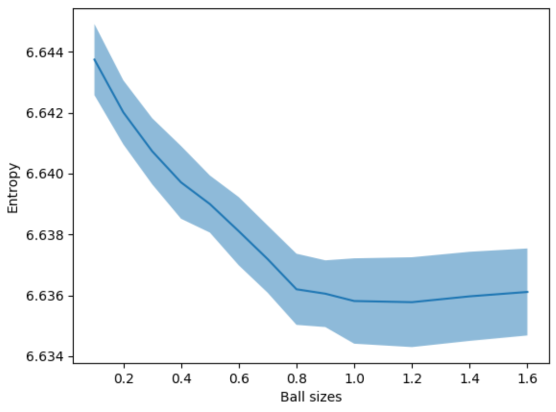


(a) Average wd$_2$ distance between concepts

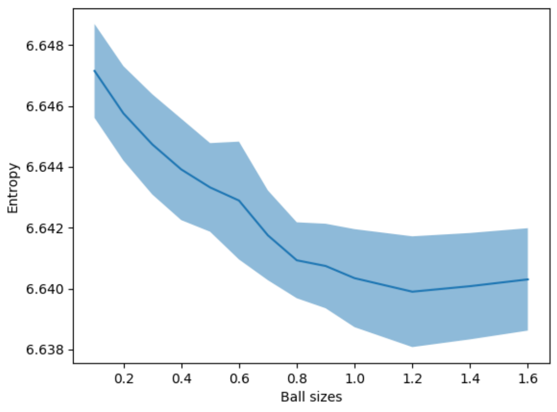


(b) Min wd$_2$ distance between concepts

Figure 25 – Correlation between the wasserstein distance (with norm L_2) of the generative distributions and the size of the ball used for L_{inf}-PGDAttack adversarial training



(a) Avergage entropy of the generative distributions

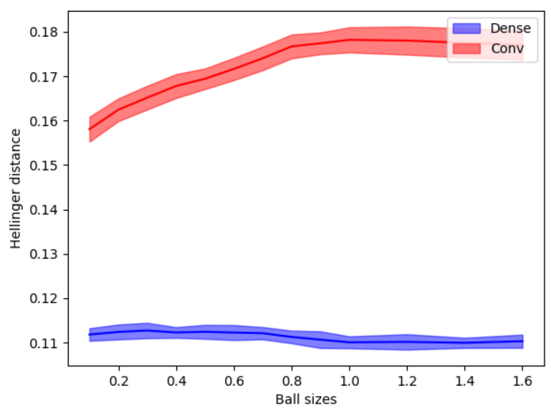


(b) Maximum entropy of the generative distributions

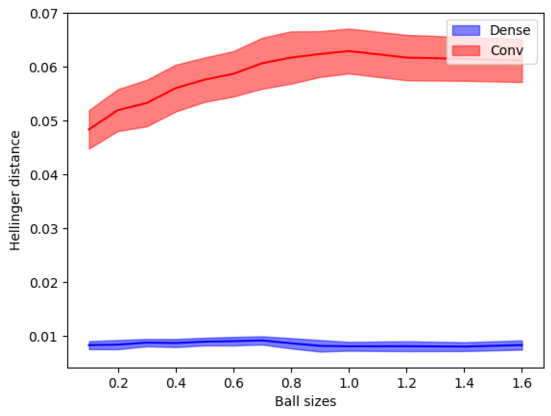
Figure 26 – Correlation between the entropy of the generative distributions and the size of the ball used for L_{inf}-PGDAttack adversarial training

7.3.3 Comparison between dense classifiers and convolutionnal classifiers

As expected dense classifiers seem to be less robust, at least with the metric we propose. However, it is remarkable that adversarial training has no effect on the sharp generative distributions (Figures 27 28), *ie.* generative distributions trained only to build well classified samples and not highly confident well classified samples (See Section 5.2.2). On the other hand, if we study fuzzy representations (Figures 29 30), we see that adverarial training is beneficial for small balls and tends to worsen the situation when balls size increases. We suspect that the decline we observe is due to too large balls that cause the images used for training to loose sense, leading to only noise. In this perspective it is then reasonable that learnt classes get closer. Actually, only the margin of the classes are affected since as we see in Figures 27 28 the sharp representations do not seem affected. It seems that adversarial training with larger balls tends to blur the distributions but it does not move their means. We also see that adversarial training has no effect on the margins of the convolutionnal classifiers.

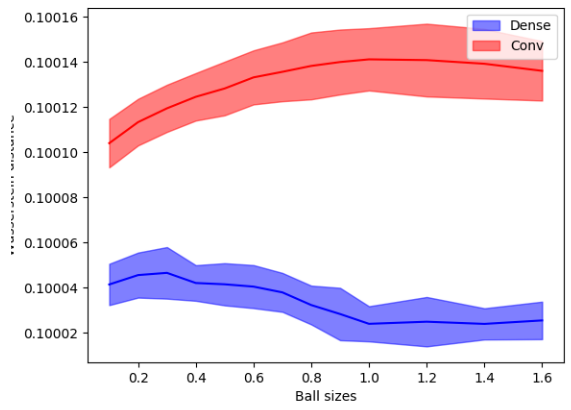


(a) Avergage hellinger distance between concepts

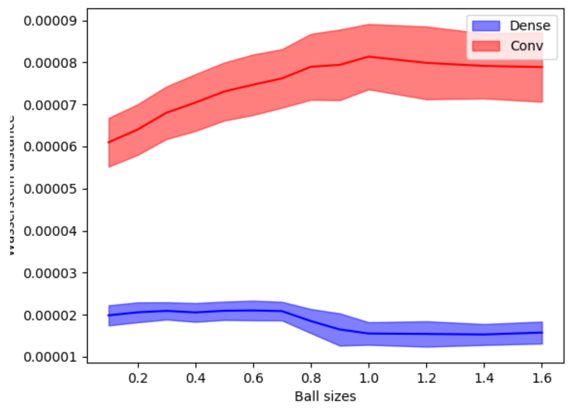


(b) Min helinger distance between concepts

Figure 27 – Correlation between the Hellinger distance and the size of the ball used for L_{inf}-PGDAttack adversarial training for convolutionnal classifiers and dense classifiers

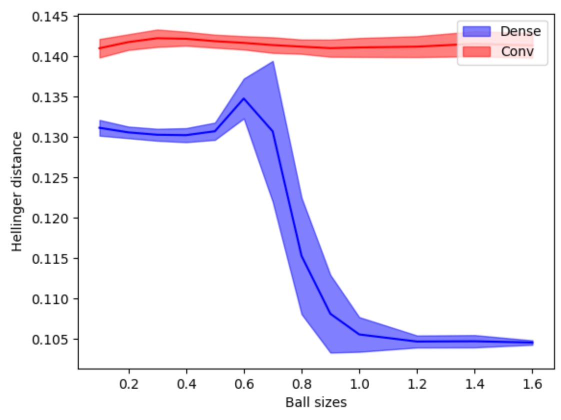


(a) Average wd_2 distance between concepts

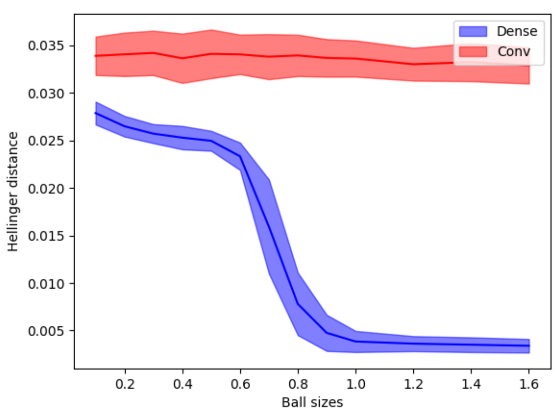


(b) Min wd_2 distance between concepts

Figure 28 – Correlation between the wasserstein distance (with norm L_2) of the generative distributions and the size of the ball used for L_{inf}-PGDAttack adversarial training

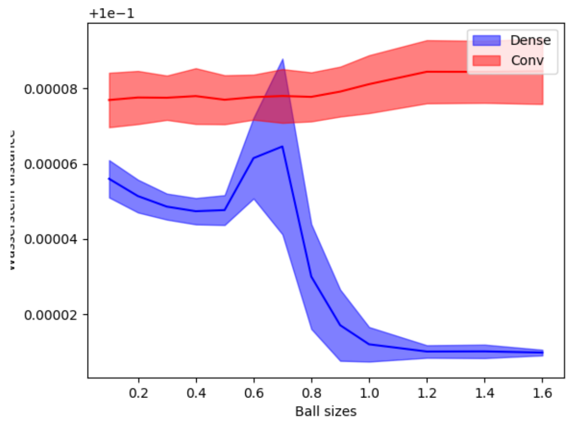


(a) Avergage hellinger distance between concepts

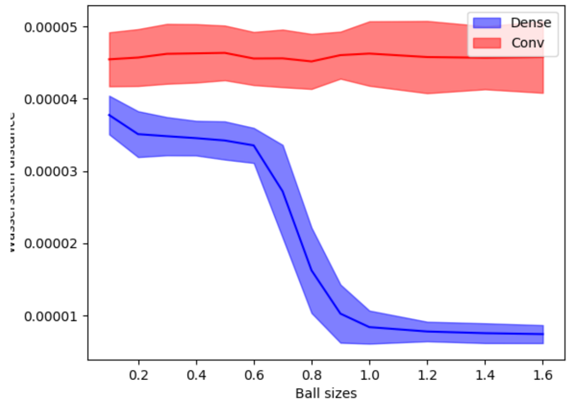


(b) Min helinger distance between concepts

Figure 29 – Correlation between the Hellinger distance and the size of the ball used for L_{inf}-PGDAttack adversarial training for convolutionnal classifiers and dense classifiers for fuzzy representations



(a) Average wd_2 distance between concepts



(b) Min wd_2 distance between concepts

Figure 30 – Correlation between the wasserstein distance (with norm L_2) of the fuzzy generative distributions and the size of the ball used for L_{inf}-PGDAttack adversarial training

8 Conclusion and future work

We demonstrated that it is possible to learn to some extent the generative distribution associated to the class of a classifier and that these distributions embedded efficiently informations about the margin and the concepts learned by the classifier. In the *MNIST* case we can easily use the generative distribution to visualize what the classifier has learnt, however, it is also useful to measure the robustness of the classifier even if there is no easy visualization available. Indeed, by measuring distances between the concepts we can evaluate between two models the most robust one.

Despite this promising results, we need to work on more challenging kind dataset in order to be able to use this method in practice. Our next goal will be to produce similar results on the well known *cifar* dataset [19]. Moreover, we think that there still is a lot of work to be done in the choice of the metrics used to measure the distances between concepts and in the training process of the generative distributions to better fit the generators to the margin of the classifier, thus getting a better view of the actual margin of the classifier.

A Neural networks

A.1 Batch Gradient descent algorithm

The whole point of gradient based training is to estimate the direction of the steepest descent and then to take a step in that direction. Batch training means that we compute that direction using a subset of the data and not only one point, this ensure a better approximation of the true direction and thus avoid over fitting.

Algorithm 4 Gradient descent algorithm

```
for $s = 1$ to $Steps$ do
    Sample a batch $\beta$
    Computes gradient $g_i = \nabla_W \mathcal{R}^{\beta}_{f_W}$
    $W = W - \eta * g_i$
end for
```

A.2 Classification problem

A classification problem is a problem in which we want to label inputs with categories. In this particular case we want to approximate the probabily distribution of the data in order to provide a probabily vector of membership.

For that purpose we want to minimize the *categorical cross entropy* produced by our model. This error denotes the difference between two distribution of probabily.

Definition 19 (Categorical Cross-Entropy)**.** Let $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ and $f_{\mathcal{W}} : \mathbb{R}^n \to \mathbb{R}^m$

$$l(f_{\mathcal{W}}(x), y) = -\sum_{i=1}^{m} f_{\mathcal{W}}(x) \log(y)$$

Usually we use a softmax as activation function for the output layer in order to get a probability vector.

Definition 20 (softmax)**.** let $x \in R^n$. $\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$

Proposition 8. $\forall x \in \mathbb{R}^n, \sum_{i=0}^{n} \sigma(x)_i = 1$

B References

[1] Haleh Akrami, Anand A. Joshi, Jian Li, Sergul Aydore, and Richard M. Leahy. Robust variational autoencoder, 2019.

[2] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula, 2019.

[3] Seojin Bang, Pengtao Xie, Heewook Lee, Wei Wu, and Eric Xing. Explaining a black-box using deep variational information bottleneck approach, 2019.

[4] Nicolas Bonneel, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. Sliced and Radon Wasserstein Barycenters of Measures. *Journal of Mathematical Imaging and Vision*, 1(51):22–45, 2015.

[5] Taylan Cemgil, Sumedh Ghaisas, Krishnamurthy (Dj) Dvijotham, and Pushmeet Kohli. Adversarially robust representations with smooth encoders. In *International Conference on Learning Representations*, 2020.

[6] Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. Learning to explain: An information-theoretic perspective on model interpretation, 2018.

[7] Maxime Darrin. Source code. `https://github.com/icannos/explaining_blackbox_infobottleneck`, 2020.

[8] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks, 2019.

[9] Andrea Galassi, Marco Lippi, and Paolo Torroni. Attention in natural language processing, 2019.

[10] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning, 2019.

[11] Ethan Goan and Clinton Fookes. Bayesian neural networks: An introduction and survey. *Lecture Notes in Mathematics*, page 4587, 2020.

[12] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J. Cree. Regularisation of neural networks by enforcing lipschitz continuity, 2018.

[13] Patrick Hall, Navdeep Gill, and Nicholas Schmidt. Proposed guidelines for the responsible use of explainable machine learning, 2019.

[14] Michiel Hazewinkel. *Encyclopaedia of mathematics.* Springer-Verlag, Berlin New York, 2002.

[15] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem, 2018.

[16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2016.

[17] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trendső in Machine Learning*, 12(4):307392, 2019.

[18] Ivan Kobyzev, Simon Prince, and Marcus Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 11, 2020.

[19] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[20] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[21] Juncheng Li, Frank R. Schmidt, and J. Zico Kolter. Adversarial camera stickers: A physical camera-based attack on deep learning systems, 2019.

[22] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[23] Jooyoung Moon, Jihyo Kim, Younghak Shin, and Sangheum Hwang. Confidence-aware learning for deep neural networks, 2020.

[24] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[25] Ewa Nowakowska, Jacek Koronacki, and Stan Lipovetsky. Tractable measure of component overlap for gaussian mixture models, 2014.

[26] Adam M. Oberman and Yuanlong Ruan. An efficient linear programming method for optimal transportation, 2015.

[27] Liang Pang, Yanyan Lan, Jun Xu, Jiafeng Guo, and Xueqi Cheng. Locally smoothed neural networks, 2017.

[28] Abhijeet Patil, Dipesh Tamboli, Swati Meena, Deepak Anand, and Amit Sethi. Breast cancer histopathology image classification and localization using multiple instance learning, 2020.

[29] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows, 2015.

[30] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models, 2017.

[31] Andriy Serdega and Dae-Shik Kim. Vmi-vae: Variational mutual information maximization framework for vae with discrete and continuous priors, 2020.

[32] Greg Stuart, Nelson Spruston, Bert Sakmann, and Michael Häusser. Action potential initiation and backpropagation in neurons of the mammalian CNS. *Trends in Neurosciences*, 20(3):125–131, March 1997.

[33] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method, 2000.

[34] Naofumi Tomita, Behnaz Abdollahi, Jason Wei, Bing Ren, Arief Suriawinata, and Saeed Hassanpour. Attention-based deep neural networks for detection of cancerous and precancerous esophagus tissue on histopathological slides. *JAMA Network Open*, 2(11):e1914645, Nov 2019.

[35] Cedric Villani. *Optimal transport : old and new*. Springer, Berlin, 2009.

[36] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil K. Jain. Adversarial attacks and defenses in images, graphs and text: A review, 2019.