Project report

Project S8 - June 2022

AI robustness against adversarial attacks

Neural Network compression of ACAS-Xu

Pierre OLLIVIER
Tom LABIAUSSE
Thomas GHOBRIL
Shruthi SUNDARANAND
Vincent MICHELANGELI

CentraleSupélec

Contents

1 Introduction

1.1 A project with IRT SystemX

Our project of *AI robustness against adversarial attacks* is in partnership with IRT SystemX which is a French Institute for Technological Research that was founded under the "Investing for the Future" (PIA) program in 2012. IRT SystemX leads research projects in digital engineering for the future mixing industrial and academic partners.

The aim of our project is to characterize an AI model by its robustness and evaluate it against various adversarial attacks. From there, we can compare the performances of several attacks and quantify the validity of some properties of the network. The AI model that we have to consider is a collection of neural networks which are part of a broader system referred as the ACAS-Xu system. ACAS-Xu is a state of the art airborne collision avoidance system designed for unmanned aircrafts such as drones. For example, giant companies, such as Amazon, plan to deploy massive fleets of drone for deliveries in the next few years. However, that may only become reality if the company achieves to set up a reliable anti-collision system for the aircrafts. Hence, by studying the safety of the neural networks involved in ACAS-Xu, SystemX is tackling an element of this complex challenge.

1.2 What is ACAS and ACAS-Xu ?

ACAS stands for *Airborne Collision Avoidance System* and is a complex system aiming to prevent aircrafts collisions in the sky. ACAS-Xu is a new version of ACAS focused on unmanned aircrafts, that belongs to the ACAS-X family. It is an optimization based approach relying on probabilistic models[1].

Consider that there is an encounter between two aircrafts, the ownship and an intruder. The aircrafts are in a trajectory where they will enter the collision volume of each other if there is no change in trajectory. The system's goal is to prevent a Near Mid-Air Collision (NMAC) by monitering a collision avoidance threshold larger than the collision volume[1].

There are two subproblems[1] that ACAS-Xu hopes to solves :
- Threat detection : is this aircraft in the collision threshold ?
- Threat resolution : how to avoid it ?

The main hypothesis[1] made when considering a collision problem are the following :
- the aircraft involved in the encounter have constant velocity vectors.
- the conflict takes place in the horizontal plane.

ACAS-Xu uses dynamic programming to determine horizontal or vertical resolution advisories to avoid collisions with minimal disruptive alerts. This results in a large numeric lookup table consisting of scores associated with different maneuvers (in a finite number) from millions of different discrete states. The procedure of ACAS-Xu is then to, while considering a set of inputs describing the encounter between two aircrafts (velocities, angles, previous decisions...)and using the advice given by the system, maneuver corresponding to the lower score in the look-up table for the situation at hand.

1.3 Why does ACAS-Xu need neural networks ?

The look-up table introduced before is extremely large and needs to be sampled down for real applications. States are removed in areas where the variation between values in the table are smooth to minimize the degradation of the quality and down sample the data, but this still results in over 2GB of floating point storage[2].

A clever approach to this conflict between data storage and safe decisions making is the usage of neural networks. Neural networks can serve as a robust global function approximator and can be used to represent large data sets. It's then possible to store the information one wants to access with only few MB[2]. Moreover, the decision process is very fast (it only requires an estimation from the network, which are basically matrix products) compared to the time needed to search look-up tables.

The neural network built for ACAS-Xu has to take into account 7 inputs which are the following :

Symbol	Description	Units
ρ	Distance from ownership to intruder	m
θ	Angle to intruder relative to ownship heading direction	rad
ψ	Heading angle of intruder relative to ownship heading direction	rad
v_{own}	Speed of ownship	m/s
v_{int}	Speed of intruder	m/s
τ	Time until loss of vertical separation	s
a_{prev}	Previous advisory	advisory

Table 1: Inputs of ACAS-Xu system

One can represent some of these inputs with the following scheme :

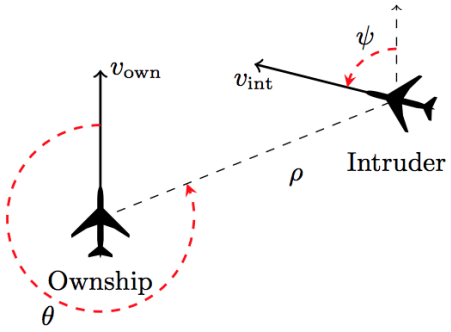


Figure 1: Aircraft encounter (adapted from *Katz et al. 2017*)

The last input a_{prev} corresponds to the last outcome/advice of the same network. Indeed, given a set of inputs, the job of the neural network is to order the maneuvers that the aircraft could take from best to worst by assigning a score to each one of them. There are five possible outcomes listed below :

Symbol	Description
COC	Clear Of Conflict
WR	Weak Right
WL	Weak Left
SR	Strong Right
SL	Strong Left

Table 2: Outpus of ACAS-Xu system

In practice, COC means that the aircraft doesn't need to change its trajectory for the moment. Of course, the embedded system of the aircraft is constantly reevaluating the situation and one hopes that if an immediate danger occurs, the system will not advice COC anymore.

In order to improve the performances of the approximation, ACAS-Xu isn't actually made of a single neural net but 45 of them. In fact, the values of the inputs τ and a_{prev} were sampled to form two sets of possible values S_τ with $Card(S_\tau) = 9$ and $S_{a_{prev}} = \{COC, WR, WL, SR, SL\}$. Therefore there are 45 possible values for the pair (τ, a_{prev}).

In order to increase the performances of the ACAS-Xu neural networks approximations, it has been decided to create a neural network for each of the 45 possible values for the last two inputs. Therefore, τ and a_{prev} are no longer considered as inputs anymore but rather as a way to index the different models (neural nets). Each of the 45 networks is then trained to approximate the entries of the look-up table corresponding to a specific value of (τ,a_{prev}). It's important to note that each neural net has the same structure described in **figure 2** below.

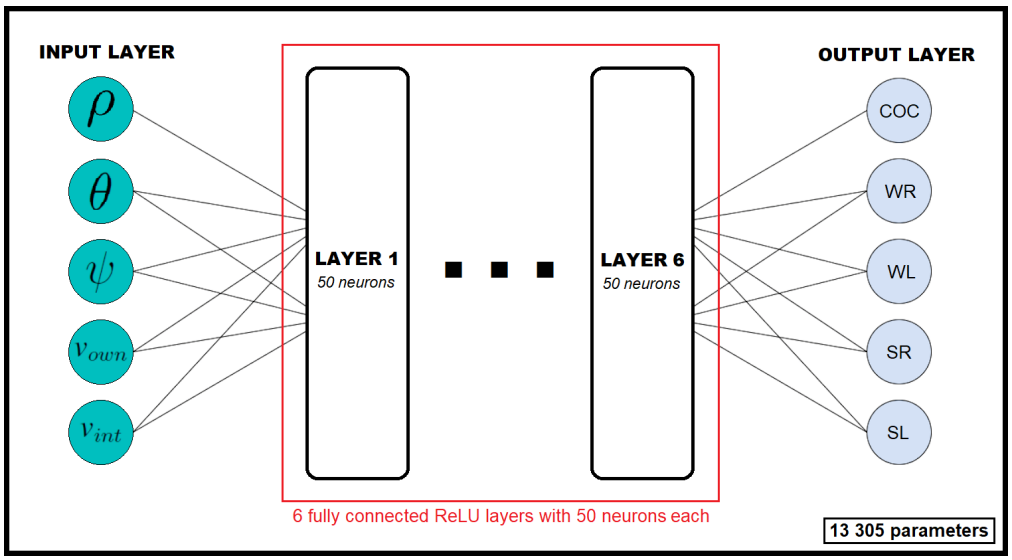


Figure 2: Scheme of one of the 45 neural networks of ACAS-Xu

1.4 Safety properties for the neural networks[4]

As neural networks approximate a look-up table on a finite set of pairs $(input, output)$, one wants to be sure that it will not predict absurd and dangerous maneuveurs on some unseen specific situations. Therefore, in order to make ACAS-Xu even more trustworthy and scalable to real applications, one wants some safety properties to be satisfied. A list of 10 of these properties has been given by SystemX and some of them are presented below.

Property ϕ_1 :

- Description : If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.
- Tested on: all 45 networks
- Input constraints: $\rho \geq 55947.691,\ v_{own} \geq 11.45,\ v_{int} \leq 60$
- Desired output property: the score for COC is at most 1500

Property ϕ_2 :

- Description : If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will never be maximal.
- Tested on: $N_{x,y}$ for all $x \geq 2$ and for all y
- Input constraints: $\rho \geq 55947.691,\ v_{own} \geq 11.45,\ v_{int} \leq 60$
- Desired output property: the score for COC is not the maximal score

Property ϕ_3 :

- Description : If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal.
- Tested on: on all networks except $N_{1,7},\ N_{1,8},\ and\ N_{1,9}$
- Input constraints: $1500 \leq \rho \leq 1800,\ -0.06 \leq \theta \leq 0.06,\ \phi \geq 3.10,\ v_{own} \geq 980,\ v_{int} \leq 960$
- Desired output property: the score for COC is not the minimal score

Property ϕ_4 :

- Description : If the intruder is directly ahead and is moving away from the ownship but at a lower speed than that of the ownship, the score for COC will not be minimal.
- Tested on: on all networks except $N_{1,7},\ N_{1,8},\ and\ N_{1,9}$
- Input constraints: $1500 \leq \rho \leq 1800,\ -0.06 \leq \theta \leq 0.06,\ \phi = 0,\ v_{own} \geq 1000,\ 700 \leq v_{int} \leq 960$
- Desired output property: the score for COC is not the minimal score

1.5 ACAS-Xu and the dangers of adversarial attacks

Adversarial machine learning is a technique that attempts to fool models with deceptive data. By giving the model misleading information or using deceiving data, we can implement attacks to see how the model reacts. A common example of adversarial attacks are spam emails. Spammers often embed attacks into these emails for a number of reasons. This can be combatted by filtering out spam emails, so the user does not open them and make them vulnerable to the attack. A more precise description of adversarial attacks is given in the state of the art in **Section 2**.

It's not hard to imagine the consequences that attacks on ACAS-Xu neurals nets could have for the aircrafts as well as for the human victims of drone crashes. By researching and implementing adversarial attacks, we can study and interpret how the system will react. We can also understand its weakness and find solutions to protect and educate the system against these attacks[8].

2 State of the art

2.1 Overview on adversarial attacks

There exists many types of adversarial attacks. A first way to classify them is to know if the attacker has access to the parameters of the model or not. In the case of ACAS-Xu, the parameters of the neural networks are already available online. Therefore, we can focus on the white-box[6] attacks, those where the models' parameters are known by everybody. That being said, one can see adversarial attacks under different angles.

In the perspective of the influence on classifiers[6], security threats towards machine learning can be classified into two categories :

(a) Causative attack. It means that adversaries have the capability of changing the distribution of training data, which induces parameter changes of learning models when retraining, resulting in a significant decrease of the performance of classifiers in subsequent classification tasks.
(b) Exploratory attack. Such attack does not seek to modify already trained classifiers. Instead, it aims to cause misclassification with respect to adversarial samples or to uncover sensitive information from training data and learning models.

In the perspective of the security violation[6], threats towards machine learning can be categorized into three groups :

(aa) Integrity attack. It tries to achieve an increase of the false negatives of existing classifiers when classifying harmful samples.
(bb) Availability attack. Such attack, on the contrary, will cause an increase of the false positives of classifiers with respect to benign samples.
(cc) Privacy violation attack. It means that adversaries can obtain sensitive and confidential information from training data and learning models.

In the perspective of the attack specificity[6], security threats towards machine learning have two types as follows :

(aaa) Targeted attack. It is highly directed to reduce the performance of classifiers on one particular sample or one specific group of samples.
(bbb) Indiscriminate attack. Such attack causes the classifier to fail in an indiscriminate fashion on a broad range of samples.

Knowing these features, it's now possible to identify more clearly the types of threats that we want to study with ACAS-Xu : white-box exploratory attacks affecting the integrity of the system. Nevertheless, we have to take into consideration both targeted and indiscriminate attacks for the moment. In fact, the kind of attacks we just identified can be referred as **evasion attacks**.

Studying evasion attacks often provide a way to identify adversarial inputs for a given network, that is a zone of the input space where the network doesn't behave as we could expect. Once we identified these points, it can be very efficient to retrain the network on it so that it is able to correct its past errors and to be even more closer to the perfect behaviour that we can dream of. That technique called **adversarial training**[8] will not be a part of our project but one can easily understand that our work aims to prepare this kind of network enhancement.

Thanks to the work done from September 2021 to January 2022 by a previous S7-team (they worked during semester 7 in CS) composed of Grégoire Desjonqueres, Aymeric Palaric, Victor Fernando Lopes De Souza and Amadou Sékou Fofana, we already have a state of the art concerning evasion attacks. We will not detail them again but if the reader is interested, he can consult the report[3] that has been written by the team. In short, they focused on some evasion attacks listed below and compared them on the MNIST database (database of black and white handwritten digits on 28x28 images) . It's important to keep in mind that they didn't have knowledge of ACAS-Xu nor the issues we are facing in our project. Their work was for us an introduction to the field of adversarial AI and some evasion techniques. The table below presents the names of the attacks tested by the S7-team.

Evasion attacks
Wasserstein attack
Virtual Adversarial Method
Shadow attack
NewtonFool attack
Fast Gradient Sign Method
Carlini & Wagner attack
DeepFool attack
Iterative Frame Saliency attack
Auto-Projecting Gradient Descent

Table 3: Adversarial evasion attacks tested on MNIST by the S7-team

2.2 Metrics and evaluation of the network's robustness[13]

2.2.1 Theoretical definitions

We denote $\mathcal{X}$ the input set, $\mathcal{D}$ the probability distribution of the input data and $\mathcal{N}$ the probability distribution of the noise, and f the decision function of the neural network.

We define the pointwise adversarial robustness of the network in point $x \in \mathcal{X}$ (the maximal perturbation that will not modify the decision at x) as :

$$\rho(f,x) := \sup\left\{\tau \geq 0 \mid \forall x' \in \mathcal{X} : \|x - x'\|_\infty \leq \tau \implies f(x') = f(x)\right\}$$

We say that a network is ε-robust if and only if $\rho(f,x) > \varepsilon$.

We define the ε-adversarial frequency of the network (probability that the network fails to be ε-robust) as :

$$\phi(f,\varepsilon) := \mathbb{P}_{x\sim\mathcal{D}}\Big[\rho(f,x) \leq \varepsilon\Big]$$

We define the ε-adversarial severity of the network (the mean maximal allowed perturbation when the network fails to be ε-robust) as :

$$\mu(f,\varepsilon) := \mathbb{E}_{x\sim\mathcal{D}}\Big[\rho(f,x) \mid \rho(f,x) \leq \varepsilon\Big]$$

2.2.2 Statistical estimators

In order to estimate those theoretical values, since we can't calculate the integral over the whole input set, we use statistical estimations : given a sample $X \subset \mathcal{X}$ drawn i.i.d. from probability distribution $\mathcal{D}$, we can estimate ϕ and μ with the standard estimators, assuming we can compute ρ.

We define the ε-adversarial frequency estimator of the network for the sample X as :

$$\hat{\phi}(f,X,\varepsilon) := \frac{\Big|\Big\{x \in X \mid \rho(f,x) \leq \varepsilon\Big\}\Big|}{|X|}$$

We define the ε-adversarial severity estimator of the network for the sample X as :

$$\hat{\mu}(f,X,\varepsilon) := \frac{\sum_{x\in X} \rho(f,x)\mathbb{I}\Big[\rho(f,x) \leq \varepsilon\Big]}{\Big|\Big\{x \in X \mid \rho(f,x) \leq \varepsilon\Big\}\Big|}$$

2.2.3 Estimation of the robustness

We define the l-targeted unreliability of the network at point x (the minimal perturbation an attacker needs in order to modify the output to label l) as :

$$\epsilon(f,x,l) := \inf\left\{\varepsilon \geq 0 \mid \exists x' \in \mathcal{X} : f(x') = l \quad \text{and} \quad \|x - x'\|_\infty \leq \varepsilon\right\}$$

Then the pointwise adversarial robustness can be written as

$$\rho(f, x) = \min_{l \neq f(x)} \epsilon(f, x, l)$$

To compute $\epsilon(f, x, l)$ we will express the existance problem as conjunctions and disjunctions of constraints, and we will study the feasability set : a conjunction of constraints is the intersection of the feasability sets, a disjunction is the union. A linear constraint has a convex feasability set, and we know how to check if the intersection of convex sets is not empty thanks to the optimization course. That's why to be able to know if a feasability set is non-empty, we need to express it as a conjunctions of linear constraints.

- The condition $f(x') = l$ on the evaluation function of the neural network (only linear and ReLU functions) can be expressed as conjunctions and disjunctions of linear constraints, see [13] for more details. We will approximate the problem by using "convex restriction", which will be a good approximation. We will replace the disjunctions due to ReLU functions with conjunctions : the idea is that the perturbation is small enough so that we can replace the piecewise linear ReLU function applyed to x' by a linear function, depending on its behavior on x. If $x < 0$, the ReLU acts as a $x \mapsto 0$, and if $x \geq 0$, the ReLU acts as a $x \mapsto 0$, so we replace the ReLU by the corresponding linear function. Again see [13] for more details. We denote $\hat{C}_{f,x,l}$ the conjunction of linear constraints of the approximation problem.

- We add the proximity constraint $P_{x,\varepsilon} : \|x - x'\|_\infty \leq \varepsilon$, which is a conjunction of linear constraints.

Now we can define

$$\hat{\epsilon}(f, x, l) := \inf \left\{ \varepsilon \geq 0 \mid \hat{C}_{f,x,l} \wedge P_{x,\varepsilon} \quad \text{is feasable} \right\}$$

and finally

$$\hat{\rho}(f, x) = \min_{l \neq f(x)} \hat{\epsilon}(f, x, l)$$

We now can compute numerically approximations of all the metrics we defined earlier by replacing ρ with $\hat{\rho}$, which we will try to implement in the second part of the project.

2.3 Other existing approaches of the problem

2.3.1 The DEEPSAFE[5] method

To check the robustness of a network, we can check if the output changes when we perturb individual data points. However, this provides only a limited guarantee on the robustness, since we only check points individually. DEEPSAFE propose an approach based on "safe regions" of the input space where the network is robust against attacks. It is based on a clustering algorithm partitioning the input space into same label regions (label-guided clustering, extension of kMeans)[5]. These regions are then checked for robustness using targeted robustness (we verify that we can not obtain an invalid input by attacking the region, see **figure 3** from *Julian 2016*[2]). This notion of targeted robustness is very useful for our problem, since in some situations we want the network to give us a plausible output but not necessarily a fixed one, so we verify that a region does not map to a specific invalid output). This method also allows to be more precise than "the network is safe/unsafe" : we find how safe is each region. Another advantage is that we can check regions in parallel, and since verification can be a long process (NP-complete problem) this provides a more scalable verification.

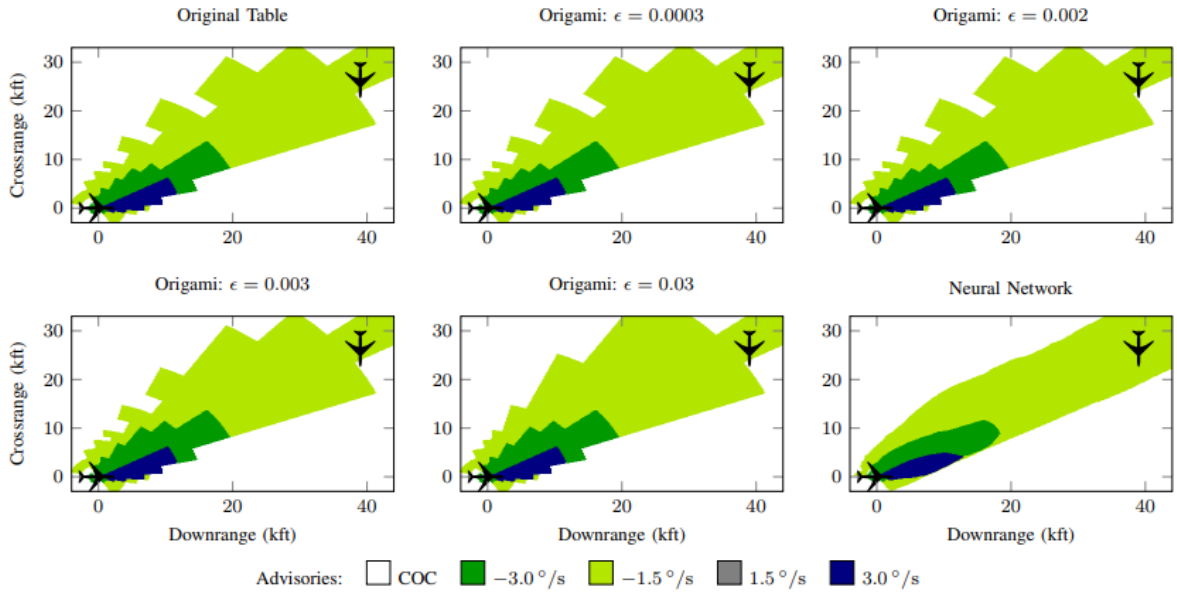


Figure 3: Advisories for a 90° encounter with $a_{prev} = -1.5°/s, \tau = 0s$

2.3.2 The RELUPLEX[9] method

As seen in the optimization course, we know how to solve a linear problem with linear constraints thanks to the simplex method. In our case, we want to verify that the solution given by the neural network verifies the properties $\Phi_1, \ldots, \Phi_{10}$, which are linear ones, however, the objective function is not linear because of non-linear RELU functions, so a new method has been developed to adapt this algorithm to linear and RELU functions. Then this algorithm is used to try to find adversarial inputs which predictions contradict the properties : for example, they found that the property Φ_8 can be violated for the first network. This method can also be applied to adversarial robustness : given a point x, and a maximal perturbation amplitude δ, can we find an adversarial point (the infinite norm can be simulated with RELU functions as in **figure 4**).

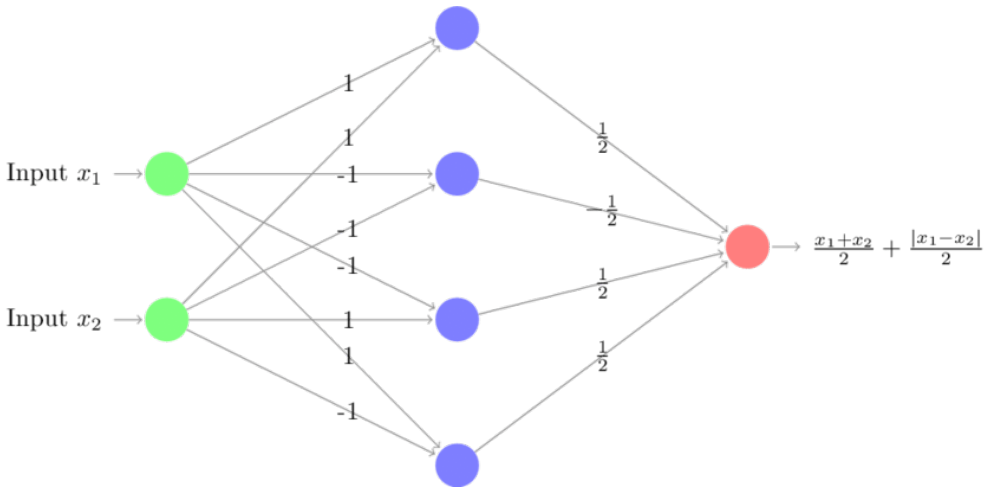


Figure 4: Simulate a max with a ReLU network

3 Structure of the project

3.1 Our goal, baseline strategy and its evolutions

As we said, adversarial attacks represent in theory a serious threat for ACAS-Xu neural networks. Therefore, we want to identify and quantify the risk in order to be able to take reinforcement measures such as adversarial training. Our approach can be seen as an **attack-based method**. Indeed, our work aims to be in keeping with the first part of this project done by the S7-team. Moreover, establishing global safety properties on neural networks is an active and complex research field. Therefore, given our recent experience in adversarial AI, it is legitimate for us to focus more on an attack-based approach than a formal one. In any case, studying the impact of existing attacks on ACAS-Xu will without a doubt benefit its comprehension and its robustness if one tries to correct its flaws based our results.

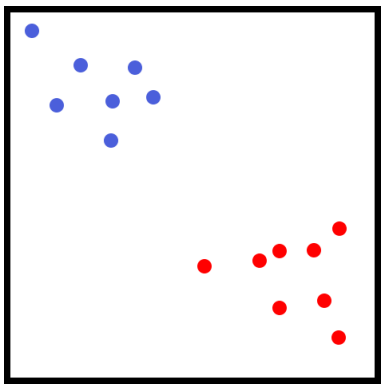
Our goals can be splitted in two different stages :

(1) Classical model robustness analysis
- Perform standard attacks that evaluate robustness, sensibility, and sensitivity of the neural network models.
- Choose simple attacks and increase the complexity of the approach and algorithms as we work through the problem.
- Present the different metrics and algorithms and synthesize the results to give a conclusion on the robustness of the provided models.

(2) Business oriented robustness analysis (properties checking)
- Check how the flight properties are respected by sampling a large amount of random points in the input domain
- Analyze the response of the attacked models with respect to the ten safety properties.
- Try to find attacks acting directly on the properties.
- Measure the robustness of the neural networks with respect to the flight properties with statistics and data visualisation.

3.1.1 STAGE 1 : Classical model robustness analysis

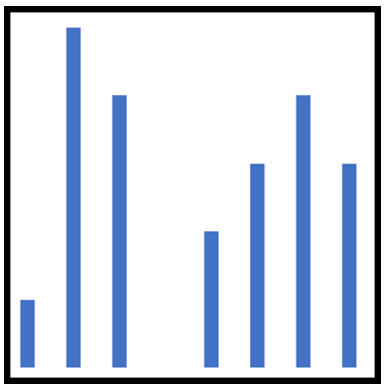
We first decide to set a collection of points in the input domain of the neural networks. Then, for each network, we compute statistics about the performance of each attack in order to see which ones are the most effective when it comes to create adversarial examples. It's also possible to look at the efficiency of a given attack on all the neural networks of ACAS-Xu. At last, we also aim to find cluster of adversarial data points shared by the 45 networks for instance. Moreover, it could be interesting to use data visualization techniques on random or attacked points. In brief, **figure 5** sum up our strategy for stage 1 in a visual form.



(a) Data visualization

Attack	Performances			
FGSM	0,45	0,52	0,16	0,57
Wasserstein	0,10	0,30	0,61	0,57
Shadow	0,28	0,04	0,64	0,35
C & W	0,83	0,05	0,12	0,64
...	0,18	0,91	0,23	0,46

(b) Efficiency of the attacks



(c) Statistics on a given network

Figure 5: Classical model robustness analysis strategy

Regarding the set of points to consider to derive the previous statistics, we have two possibilities.

First, we can generate random data points in the input space and consider that the neural networks will give a correct answer in average. In other words, if we note f the function represented by the neural net and

x_i a random data point, we will consider that x_i' is an adversarial example if it is "close enough" to x and if $f(x_i) \neq f(x_i')$.

However, if we could access to the initial data points that were used to train the networks (the look-up tables), we could derive our statistics directly from real and verified data.

3.1.2 STAGE 2 : Business oriented robustness analysis (properties checking)

With ACAS-Xu, we also have to deal with additional safety properties. However, there isn't any clear methodology in the scientific literature when it comes to deal with property checking on neural networks.

Therefore, we thought that we could reuse the attacks performed during stage 1 to see if the adversarial examples generated still satisfy the safety properties.

Moreover, we plan to design attacks to specifically attacks the properties. Of course, we will start from existing attacks such as the ones of **table 3** and modify them to fit our expectations.

For example, we already thought to use a modified version of FGSM[10] in order to find a data point which would invalidate **Property 1**. In short, starting from an input point x with desired label y, FGSM can be understood with the following formula where J is a loss function and ϵ the perturbation :

$$x_{adv} = x + \epsilon sign(\nabla_x J(\theta, x, y))$$

We thought to compute the loss relative to the first output COC of the networks. Then, FGSM will tend to find a point x_{adv} with a high score for COC. As the output restriction of **Property 1** is an upper bound of 1500 on COC, we can hope to cross that frontier with FGSM and therefore generate an adversarial point.

With a well-chosen loss, we could hope to apply FGSM to try to invalidate other properties. Indeed, their output restriction is not as simple as the one in **Property 1**.

3.1.3 Strategy adjustments

After further examination we chose not to pursue the business oriented analysis for a few reasons : first as we couldn't touch the neural networks themselves, it would have been really complicated. Also, the time constraint didn't allow us to go in that direction considering the work that STAGE 2 of our baseline strategy represents.

That is why we focused on classical models to judge the robustness of the networks. We implemented and adapted a number of attacks to our problem, tested them and analysed the results. Even though some attacks presented promising results in terms of adversarial points, they didn't deliver on the complexity and run time of the algorithms (as it is explained in section 4.3 of the report). Therefore, we decided that FGSM was the best compromise between run time and efficiency of the attacks and we delve deep into its analysis.

The first part of the work was to select the networks that we wanted to test out as 45 networks seemed a bit over the top considering they deliver very similar results. We used clustering of the networks based on the typical distribution of their output and we chose a few that would be representative of the global behaviour of the networks. Then the idea was to attack those networks, using FGSM (and in the beginning the other attacks) and make relevant statistics around the attacks (typically confusion matrices). Finally we used some metric-based evaluation to try to quantify the best we could the vulnerability of the networks independently of the attacks.

3.2 Planning and tasks distribution

Our organization during the first stage of the project was the following:

Tasks	Feb 3	Feb 10	Feb 14	Feb 17	Mar 10	Mar 24	Apr 5	Apr 7	Apr 12	Apr 14	Apr 19	Apr 21	Apr 26
Read papers about ACAS Xu neural nets/security/property checking				Shruthi, Tom, Thomas	Shruthi, Tom, Thomas	Shruthi, Tom, Thomas			Thomas				
Read papers/concepts/algorithms about adversarial attacks	Everyone	Everyone	Everyone	Everyone	Everyone	Everyone	Shruthi, Pierre	Shruthi, Pierre	Shruthi, Pierre	Shruthi, Pierre			Shruthi, Pierre
Implement random search method to get an overview				Tom, Thomas	Tom, Thomas	Tom, Thomas	Tom	Tom	Tom	Tom	Tom	Tom	Tom
Apply adversarial attacks to generated data points					Pierre, Vincent	Pierre, Vincent	Vincent	Vincent	Thomas, Vincent	Thomas, Vincent, Pierre	Thomas, Vincent, Pierre	Thomas, Vincent, Pierre	
Convert ACAS Xu files into tensorflow files (debug or construct tensorflow file directly with the weights from .nnet file)			Tom	Pierre, Vincent	Pierre, Vincent	Over!							
Communicate and write the reports							Shruthi, Pierre	Shruthi, Pierre	Shruthi, Pierre	Everyone	Everyone		

Figure 6: Planning of the project from February to April

We work on team once or twice a week, depending on the scheduled time we have. We meet with the client around once every two weeks. Planned tasks always evolves as our comprehension of the system's does and we imagine new strategies and analysis. The following table presents how we organized ourselves during the second part of the project :

Tasks	Apr 26	Apr 28	May 12	May 17	May 19	May 30	May 31	Jun 1	Jun 2
Find and visualize clusters of networks	Shruthi, Pierre	Shruthi, Pierre	Pierre, Vincent	Vincent	Vincent	Vincent	Vincent	Vincent	
Apply FGSM on data points	Tom	Tom	Tom	Tom	Tom	Tom	Tom	Tom	
Apply other attacks on data points	Thomas, Vincent	Thomas, Vincent	Shruthi	Pierre	Pierre	Pierre	Pierre	Pierre	
Compute metrics				Thomas	Thomas	Thomas	Thomas	Thomas	
Analyze the ethical aspects of the project					Shruthi	Shruthi	Shruthi	Shruthi	
Write the final report					Everyone	Everyone	Everyone	Everyone	Everyone

Figure 7: Planning of the project from April to June

3.3 Risk analysis

After analyzing the problem we came up with the following risk matrix where 5 is considered as an extremely serious risk compared to a not very disturbing risk at 1 :

Risk	**Gravity (1 to 5)**	**Solution**
Not understand what the client wants	4	Communicate with the client
Lacking time to finish the project	3	Organize ourselves via a task planning
Lacking time to finish the reports	2	Beginning to write the report 2 weeks before the oral defense
Being overwhelmed by the quantity of attacks and networks to test	4	Respect the task planning
Wasting time with implementing a solution that already exists	2	Take time at the beginning to explore the state of the art
Being surprised and destabilized by something unplanned	4	Anticipate

Table 4: Risk analysis of the project

3.4 Deliverables

First of all, we provide the client a methodology to evaluate different evasion attacks using the adversarial-robustness-toolbox on the 45 neural networks of ACAS-Xu. Our attack-based analysis method is made of 3 parts precisely described in parts 4.1, 4.2 and 4.3. In addition, we also provide 3 python notebooks allowing anyone to run their own experiments thanks to many comments besides the code. These notebooks are the following :

- **networks_analysis.ipynb** - [*Section 4.1*] : Analyse the similarities between networks and presents a way to construct clusters and reduce the study from 45 to 9 networks.
- **confusion_matrix_analysis.ipynb** - [*Sections 4.2 and 4.3*] : Provides graphic tools to visualize the behaviours of different networks when facing attacks with various intensities.
- **adv_pts_displacement_analysis.ipynb** - [*Sections 4.2 and 4.3*] : Analyses the displacement of attacked points and gives graphical representations of the change in their coordinates.

Moreover, we also tackled a our robustness evaluation mission using a robustness metric as presented in part 4.4. The python notebook associated to this work is the following :

- **robustness_metric_analysis.ipynb** - [*Section 4.4*] : Analyses the robustness of ACAS-Xu networks from a statistical point of view independently of the attacks, using linear optimization.

3.5 Structure of the team

Our team is composed of five members : Thomas, Tom, Vincent, Pierre and Shruthi. Shruthi is an American student who joined CentraleSupélec for the semester 8. The four other members are French students in the regular engineering curriculum at CentraleSupélec.

The team can also count on Rémy Hosseinkhan which is a PhD student in computer science in Paris-Saclay University. We also want to thank him for the precious support that he gave us all along this four months project.

We use several working tools: Python (with the Tensorflow package) for programming (either using Visual Studio Code or Google Colab), Slack and WhatsApp to communicate within the team, Teams to organize meetings and communicate with the teachers and the clients, Overleaf and Word to produce the written documents.

We store our code and all our work on Github. The address of our repository is the following :

`https://github.com/thomasghobril/adversarial-attack`

4 Study of ACAS-Xu neural networks

This section presents the results and the analysis concerning our study of the ACAS networks. Due to time constraints, we mainly focused on STAGE 1 of our strategy described in **3.1.1** i.e a classical model robustness analysis in which we didn't take into account the safety properties introduced in **1.4**.

4.1 Basic representations and classes of neural networks

As our task is to study as many well-known attacks as possible on the 45 neural networks of ACAS-Xu, one can easily understands that it requires a consequent amount of machine time for computation and human time for results analysis. Unfortunately, even if the computation power might be at our reach, we cannot imagine to correctly process the information we would get from different attacks with various intensities applied to each neural network.

Hence, we first decided to focus on a narrow range of neural networks from ACAS-Xu. However, by doing that, we took the risk to miss critical networks or to analyse very similar behaviours. In order to counterbalance that, we tried to classify the 45 networks in a manner that would allow us to pick a network from a class with the assumption that it has more or less the same behaviour as its neighbours in the same class.

In order to put this in application, we had to find a way to represent the networks and to compute a measure of similarity between them. The simplest thing we could think of was to get an image of each network by evaluating them (get the predicted labels) on a discrete amount of points randomly generated. Moreover, a large amount of points is essential to get a accurate approximation of the networks' behaviours. This lead us to a have a very high dimensional for the networks representation. As clustering techniques don't perform well in high dimensional spaces and especially when the number of points is weak in comparison, we decided to count the number of points predicted as COC for each network. In this way, we get only one scalar per network. Doing that for each label, we are now able to represent each network in a 5D space corresponding to the number of output labels. We implemented this method on a randomly generated set of one million points in the inputs domain. The following tab gives an insight of the representations we obtained for the networks :

Networks	**COC**	**WR**	**WL**	**SR**	**SL**
1-1	854791	29945	35506	42838	36920
1-7	971651	14367	13602	158	222
1-9	999337	304	359	0	0
2-8	875671	124312	0	17	0
3-2	872825	3	58063	11088	58021
...	...	...	...	...	...

Table 5: 5D representation of the networks through evaluation on 1M points

One important thing to observe is the presence of many low or null values in the table. It means that some networks will probably never predict a given set of labels. For example, network 1-9 is likely to never advise a strong right or left turn. That gives a first and important insight on the system's behaviour. In **figure 8** below, one can easily see that the *no prediction* phenomenon also occurs on a large scale among all networks.

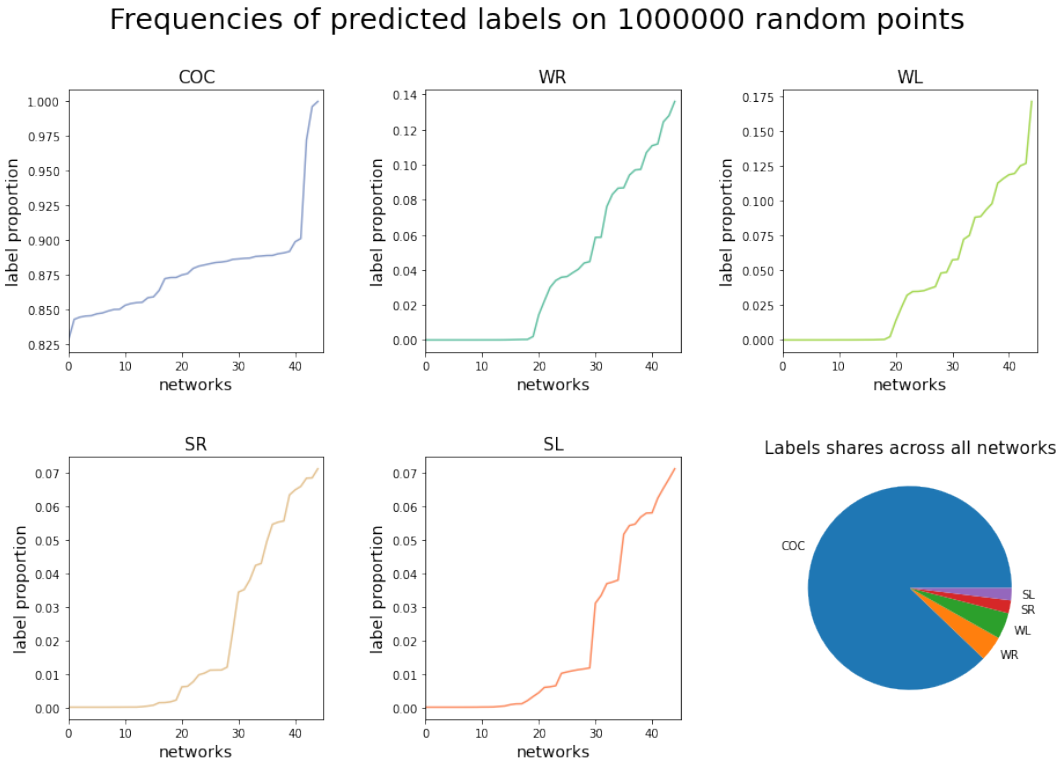


Figure 8: Repartition of labels frequencies among the networks

One important technical detail here is that there is no common x-axis between the 5 plots of **figure 8**. The networks are just sorted according to their label proportion for each label. Thanks to these plots, it's easy to see that almost all networks predict between 80% and 90% of the points as COC. On the contrary, for each other label y, there are almost 20 networks which never predict y.

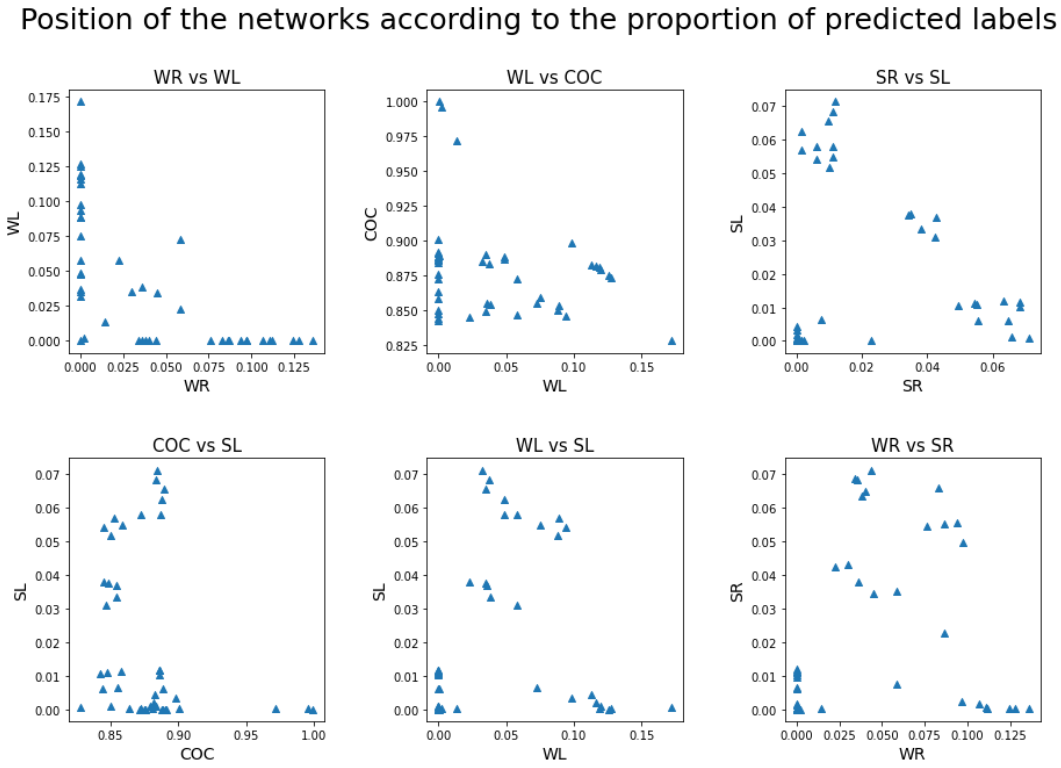


Figure 9: Position of the networks according to the predicted labels

The graphs in **figure 9** aim to facilitate the comparison between networks which is not possible with **figure 8** alone. Indeed, it becomes interesting to see that there are similarities between networks. For example, one can easily discern four clusters on the plot SR *vs* SL. Therefore, we had every reason to believe that our representation gave us enough information to differentiate the networks but is still easy to implement and to interpret.

As we had a 5D representation of each network, we were able to use classic clustering techniques such as K-Means, hierarchical clustering, spectral clustering... However, we were eager to keep the meaning of our representation through the clustering. In other words, we decided to perform the clustering "by hand" by taking advantage of the previous comment about low or null values. Our specific clustering method is defined by the following equivalence relation between two 5D vectors X_1, X_2 representing two neural networks :

$$X_1 \sim X_2 \Leftrightarrow \forall k \in [0,4],\ \Big(X_1[k] < \epsilon * N \Leftrightarrow X_2[k] < \epsilon * N\Big)$$

ϵ is a scalar parameter between 0 and 1 choosen very close to 0. In simple terms, ϵ represents the threshold level controlling when a component of a 5D vector representing a network can be considered as 0 or not. We chose $\epsilon = 0.1\%$ meaning that, through the equivalence relation, all vector components between 0 and 1000 are considered as 0. Thanks to this equivalence relation, we were able to define several classes of neural networks named clusters and listed below :

Cluster	**Networks**	**Mask**
1	23 - 24 - 25 - 26 - 43	10100
2	41 - 42 - 44	10101
3	14 - 15 - 16 - 17	11000
4	31 - 32 - 33 - 35	11010
5	9 - 10 - 11 - 12 - 13 - 27 - 28 - 29 - 30	11011
6	18 - 19 - 20 - 21 - 22 - 36 - 37 - 38 - 39 - 40	10111
7	6 - 7	11100
8	0 - 1 - 2 - 3 - 4 - 5	11111
9	8	10000

Table 6: Clustering of neural networks with N=1000000 points and $\epsilon = 0.001$

The *mask* indication in the table corresponds to the equivalence classes. For example, *10100* means that the cluster is made of networks whose 5D representations have only two components greater than the threshold set at $\epsilon \times N$.

Using this notation, it's easy to see that cluster 7 is the class of networks where all components seem to be significant. In other words, networks 0,1,2,3,4 and 5 seem to be able to predict any of the 5 labels with a non-negligible probability. Then, it might probably be easy to create adversarial points for these networks as they already present a diversified behaviour.

On the contrary, looking at cluster 9, one can see that network 8 seems to act as a constant function (if we only focus on the predicted label, not the 5 scalar outputs of the neural network). Therefore, adversarial points may be very dangerous for such a network because it would totally modify its behaviour.

As we started to see interesting patterns in the clustering, we wanted to validate *a posteriori* our clustering technique on the 5D representations of the vectors. For this to happen, we used a *Principal Component Analysis* or *PCA* to reprent the 5D vectors in the plane formed with the first two principal axis. Then, we gave to each network in this 2D representation a color depending on the cluster of the network. We obtained the following plot :

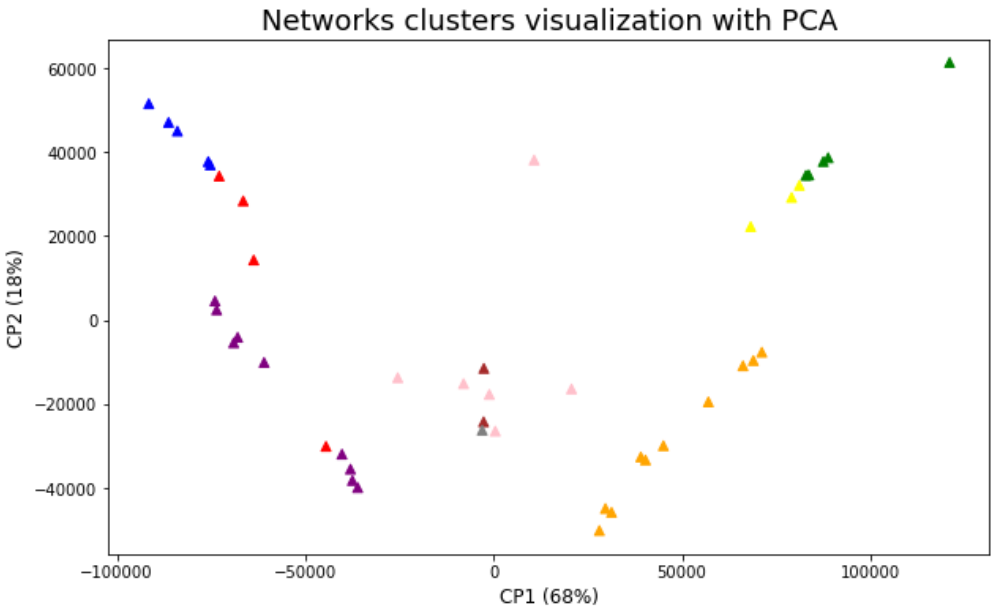


Figure 10: First two principal components of the 5D representations of the neural networks

It's obvious that the clusters we built with our method correspond pretty well to the groups that one can identify using the principal plane. Therefore, we decided to keep the clusters identified in **table 6**. For the reamaining experiments and analysis, we arbitrary picked a network from each cluster thus reducing the number of effective networks to deal with from 45 to 9. **table 7** below sums up the chosen ones.

Networks (index)	**Networks (original name)**	**Cluster**	**Mask**
23	3-6	1	10100
41	5-6	2	10101
14	2-6	3	11000
31	4-5	4	11010
9	2-1	5	11011
18	3-1	6	10111
6	1-7	7	11100
0	1-1	8	11111
8	1-9	9	10000

Table 7: Chosen neural networks after clustering

It's important to keep in mind that our approach was motivated by time and material constraints. Most of all, we don't claim that all networks in the same cluster are very close from a mathematical point of view but rather that two networks from different clusters are in all likelihood quite different.

In the following paragraphs, we present the results of our attack-based experiments on the networks from ACAS-Xu. Our approach was to randomly generate a large quantity of points, perform a well-known attack on them and analyse the shift in the behaviour of the neural networks when asked to classify the attacked points and the base points. The attacks were chosen among those in **table 3**. However, we had to face a technical issue being that some attacks are very time-consuming compared to some others. Indeed, many of them are based on iterative algorithms with a number of iterations varying from an execution to the other. Therefore, we had to dramatically reduce the range of our experiments on such attacks and more specifically the number of points to test. However, the methods and the associated pipelines of code we propose are adapted for both kind of attacks and can be combined with higher performances machines or parallel computing techniques to process a larger amount of data in a reasonable time slot.

4.2 Proof of concept of our analysis method with a non-iterative attack : FGSM

This section aims to explain our method in details as well as to present the resulting analysis concerning the impact of the attack FGSM on the 9 neural networks of **table 3**. As a matter of fact, FGSM is based on gradient computations performed via backpropagation through a neural network and is considered as a fast one among all its counterparts such as Carlini&Wagner, NewtonFool, DeepFool... This makes FGSM a perfect non-iterative candidate to run our method on a large range of points and intensity of the attack.

4.2.1 An approach based on confusion matrices

The corner stone of our approach is the very natural concept of confusion matrix explained thereafter. Let's consider a network X, an attack f and a set of N base points. We first evaluate the network on every base point. Then we perform attack f on each base point leading to create a set of N adversarial points on which we evaluate X again. Finally, for every pair of labels (i, j), we count the number of points that were at first classified with label i but ended with label j after attack f and record that quantity in $c_{i,j}$. Hence, as the neural networks from ACAS-Xu classify points into 5 possible labels, we end up with $5 \times 5 = 25$ coefficients $c_{i,j}$ forming what is usually called a confusion matrix. **Figure 11** below gives an example of such a matrix :

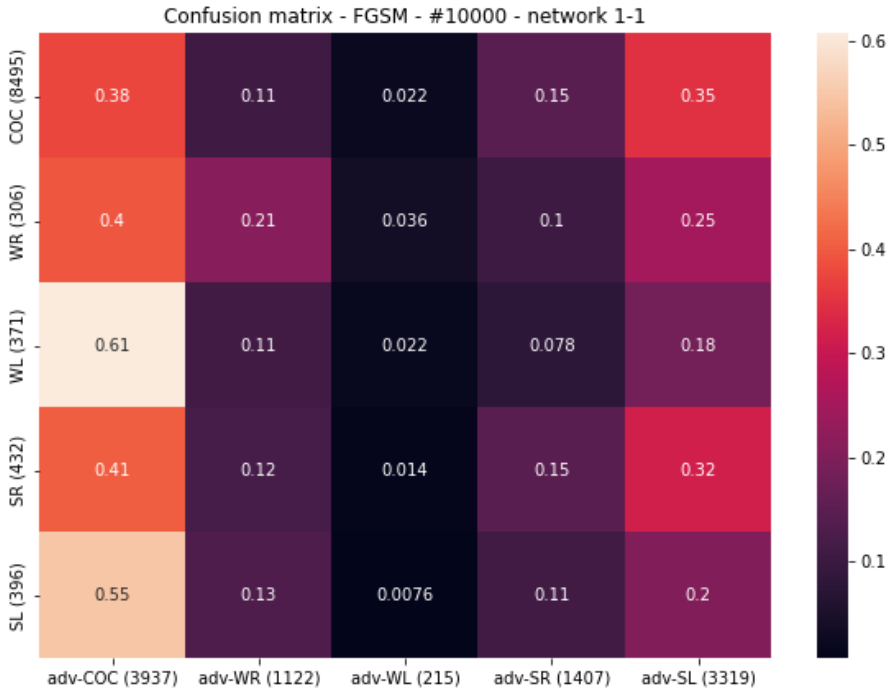


Figure 11: Example of confusion matrix with FGSM performed on 10000 points with net 1-1

Here are some useful indications to read the matrix :

- Numbers given between parenthesis next to the labels of the lines and columns correspond to the number of points matching the description of the line/column. For example **SR(432)** means that 432

were originally classified as SR and **adv-SR(1407)** means that 1407 points ended up with label SR after applying FGSM on the base set.

- If we note $c_{i,*}$ the number of points that were originally classified as label i, then coefficient at position (i, j) in the matrix of **figure 11** matches the quantity $\frac{c_{i,j}}{c_{i,*}}$. Hence, we can say for example that, considering the experiment that lead to **figure 11**, 35% of the points that were first classified as COC got label SL after applying FGSM.
- The color distribution of the matrix gives an indication on the strength of the attack. For example, let's consider an extreme case where the attack is the identity transformation. It's not really what we would call an attack but since the adversarial points would exactly match the base points, the confusion matrix would be a diagonal one. We can then state the following rule of the thumb : the more different is the confusion matrix from a diagonal matrix, the more powerful is the attack.

Analysing the matrix of **figure 11**, we could say that the specific use of FGSM on network 1-1 seems to reduce the number of points predicted as WL compared to a near ten-fold increase in the number of points labelled as SL. In fact, there are many indications given by that kind of matrix.

However, one could notice that we didn't specify the value of intensity that was chosen to perform FGSM. As many adversarial attacks, there exists a parameter often called intensity and written ϵ controlling the maximal deviation allowed from a base point when generating an adversarial example. Needless to say that if the order of magnitude of the intensity is similar to the size of the input domain then the adversarial point may not be very specific to the base point. That's the reason why the intensity of the attack is a key element in our approach.

In order to avoid false analysis from ill-chosen values of ϵ, we decided to explore many possible values for the intensity. First of all, following the definition of ϵ for FGSM, and taking into account the fact that the 5 inputs are reduced, there was no need to test values higher than 1 for ϵ. We then decided to test a range of values between 0 and 1 on each network meaning that we generate a confusion matrix at each iteration on each network. The following pseudo code gives a more precise insight of the process :

Algorithm 1 Compute confusion matrices for a given attack on a range of intensities and networks

Require: an attack f ; a range of intensity I ; a range of networks to evaluate L_X ; a set of N base points A
Ensure: a set of $|I| \times |L_X|$ confusion matrices to analyse

$A_{labels} \leftarrow get\_labels(X, A)$
for $i \in I$ **do**
  for $X \in L_X$ **do**

    $A^{i,X} \leftarrow generate\_adversarial\_points(f, A, i, X)$

    $A^{i,X}_{labels} \leftarrow get\_labels(X, A^{i,X})$

    $M^{i,X} \leftarrow create\_conf\_mat(A_{labels}, A^{i,X}_{labels})$

  end for
end for

We chose to run **Algorithm 1** with FGSM, all networks from **table 3**, 1000 base points and a uniformly distributed range from 0 to 1 with step 0.01 for ϵ. Concerning the number of points, we first thought to use at least 10^5 points. In fact, the idea was to discretize each input dimension with at least 10 points leading to a total of 10^5 because we have inputs of dimension 5. However it turned out that with that setting, **Algorithm 1** would take almost 3 hours to run. Even if it was still acceptable, it will not be possible to consider the same amount of points with iterative attacks as we will discuss later. Hence, in order to reduce the gap between the number of points that we consider for non-iterative and iterative attacks, we decided to lower the number of points to 1000 for FGSM. The total running time of **Algorithm 1** was then reduced from 3 hours to 10 minutes.

4.2.2 Analysis of the deviations from label to label with the chosen networks

From a technical point of view, we then obtained a 4D table, first dimension corresponding to the networks, second to the intensities and third/fourth dimensions for the confusion matrices. We decided to start the analysis with the evolution of the top-left coefficient of the confusion matrices as a function of the intensity and that for each neural network. That led us to the following plot :

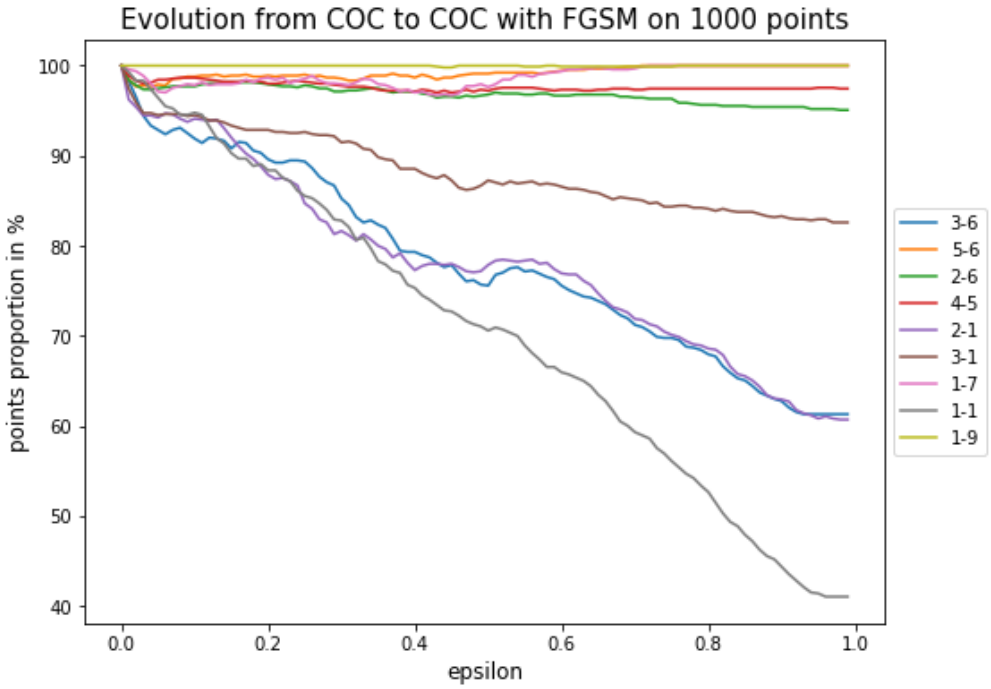


Figure 12: Evolution of coefficient ($COC \to COC$) as a function of ϵ from 0 to 1

There are many useful comments to make on **figure 12** :

- We can first observe the end point of each curve. It's then possible to separate networks in two groups. First one is composed of networks whose line doesn't deviate much from the 100% level, namely : 5-6, 2-6, 4-5, 1-7 and 1-9. The second group brings together networks 3-6, 2-1, 3-1 and 1-1 whose end points is located between 40% and 90%. Clearly, networks from group 2 seem to be much more sensitive to FGSM when it involves label COC than networks from group 1.

- We can also look at the extreme curves. As we already said, attacking a base point with an intensity of 1 when inputs are reduced means that the associated adversarial point can be very far from the base one. Therefore, as FGSM tries to find a point with a different label from the base one, we could expect to find all curves reaching 0% when $\epsilon \to 1$ on **figure 12**. However, the experiment totally contradicts that intuition. In fact, the lower point is around 40% and is reached by network 1-1. Looking back at **table 7**, one can see that network 1-1 is in a cluster with mask 11111. On the opposite, network 1-9 with mask 10000 barely moves from level 100% in **figure 12**. These observations confirm our feeling that some networks almost always predict COC when some others are more balanced.

- If we try to fit the curves with a specific analytic function, it would not be a big mistake to use linear functions. Indeed, almost all curves tend to follow a straight path on **figure 12**. However, we can note that curves of networks 3-6 and 2-1 tend to be very similar with a rebound for $\epsilon \approx 0.5$. That's an interesting fact given that these networks come from cluster 1 and 5 which seem to be very distinct at first glance if we look at their mask on **table 6**.

Thanks **figure 12**, we are able to extract information about the networks but not really on the impact of FGSM on it because we mainly look at high values of ϵ. Therefore, we decided to zoom on smaller values using **Algorithm 1** with range 0 to 0.1 and step 0.001 as it is shown in **figure 13**.

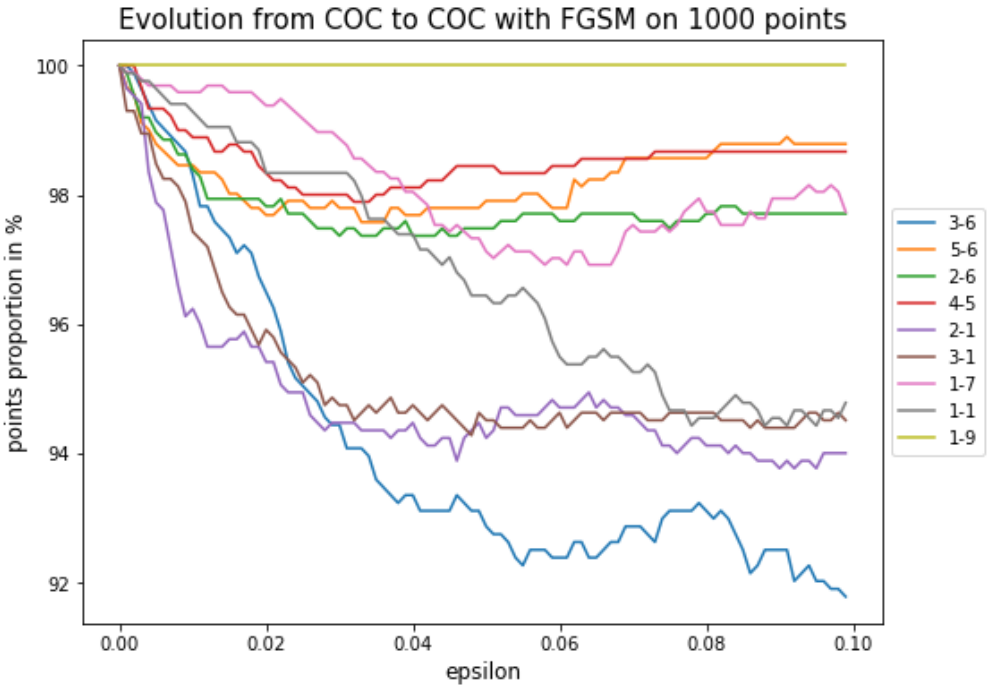


Figure 13: Evolution of coefficient ($COC \to COC$) as a function of ϵ from 0 to 0.1

One can observe that some of the findings made with **figure 12** still hold. Indeed, label COC seem to be more sensitive against FGSM in networks 3-6, 3-1 and 2-1 as the curves present a more abrupt slope for $\epsilon \in [0, 0.01]$. Moreover, network 1-1 doesn't seem to be very affected by FGSM for low values of ϵ which is in contradiction with the previous observation for higher intensity values. Another interesting fact is the strongly non-linear relation observed in **figure 13**. In fact, we can even see a small increase for networks 5-6 and 4-5 around 0.05. Unlike the previous plot, it is reasonable to consider the phenomenon at hand as a non-linear one because we're much more interested in low values of intensity than large ones for a realistic adversarial points study.

Of course, it's possible to decline the two previous graphs with many pairs of labels. However, given that there are 25 possible combinations of them, it's difficult to analyse all of them correctly. The code we wrote is nevertheless perfectly suited for graphs generation and can be used very simply. The following figure gives an insight of some other interesting plots :

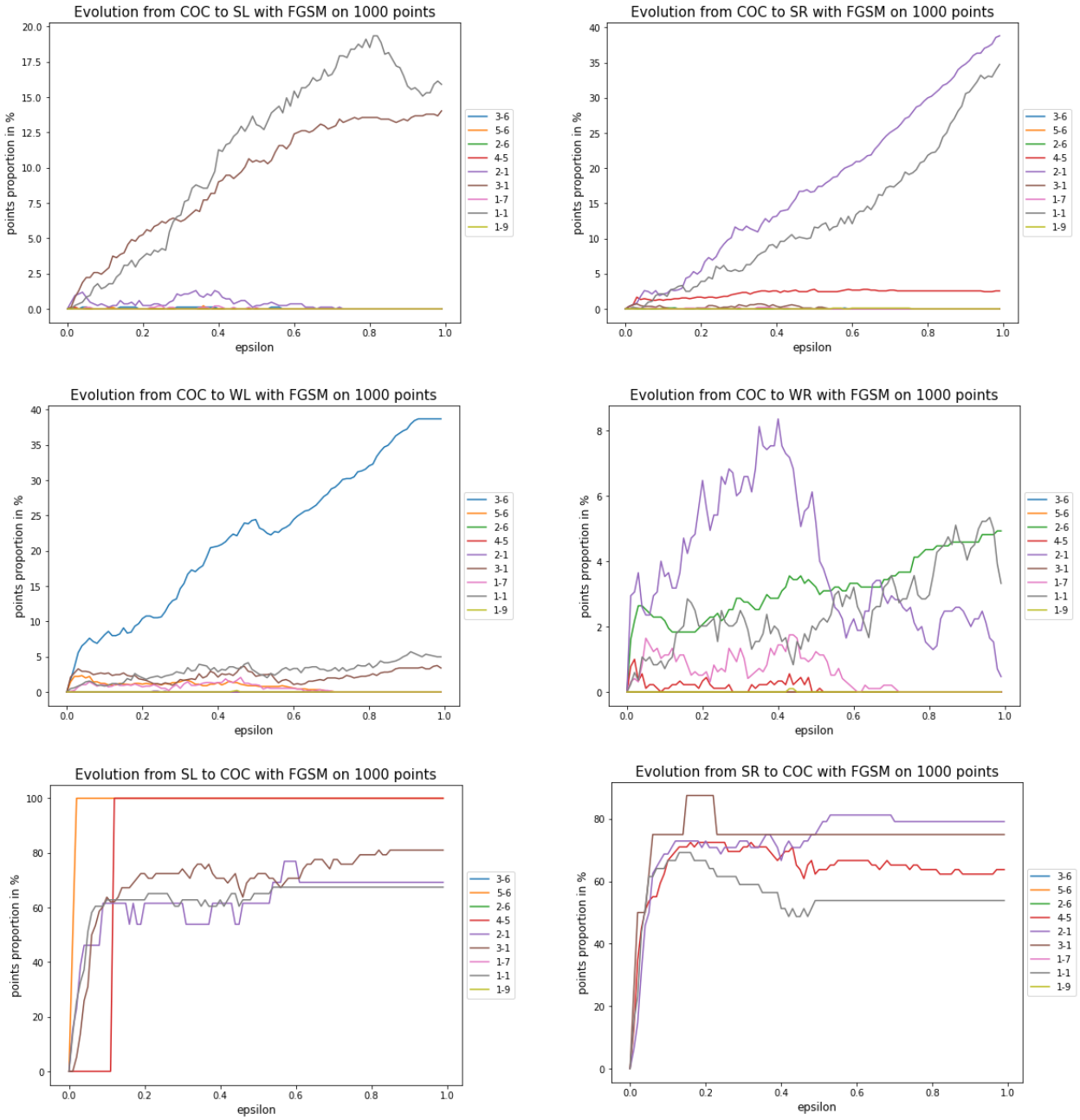


Figure 14: Examples of different coefficients in the confusion matrices as functions of ϵ from 0 to 1

There are again many behaviours to analyse. We will discuss about the main ones :

- If we look at the plots ($COC \to SL$) and ($COC \to SR$), we clearly see that network 1-1 tend to change label from COC to extreme moves namely *Strong Left* (SL) and *Strong Right* (SR). Moreover, network 3-1 does the same for COC to SL but no at all to SR. Conversely, network 2-1 tends to change COC to SR but not to SL.

- If we look at the plots ($COC \to WL$), we do not get the same conclusions as previously. Indeed, even if the plot has a similar appearance, network 1-1 doesn't have the same position at all. One can see that network 3-6 is the one showing the most important deviation from COC.

- Looking at plot ($COC \to WR$), one can think to a much more chaotic behaviour of the networks. However, looking at the scale on the y-axis, we can conclude that there isn't any network crossing the 10% deviation level for ($COC \to WR$). Even more striking is that network 3-6 has a strong deviation ($COC \to WL$) and a null one for ($COC \to WR$) (which justifies the fact that we can't see the curve on the plot).

- The two last plots ($SL \rightarrow COC$) and ($SR \rightarrow COC$) are maybe the more important one for the ACAS-Xu system. Indeed they give indications about the deviation from extreme moves to the absence of action. It's then easy to understand that we would prefer to control these deviation because they may probably be responsible for aircraft collisions. Unfortunately, one can see that every curve on the two last plots increases very quickly and cross the 50% level before $\epsilon = 0.1$. This has very concrete consequences that can be illustrated with the following statement : **if you attack a SL or SR point (no matter the network used for prediction) with FGSM at intensity 0.1, you may change the label to COC 50% of the time**.
- If we focus on plot ($SR \rightarrow COC$), we can note some differences between the end curve level of networks 1-1, 4-5, 3-1 and 2-1 from 50% to 80%. Once more, it gives indications on the sensitivity of the networks against FGSM. However, not all the 9 networks are represented on the plots meaning that some networks never predicted SL or SR a single time on the 1000 base points. That is indeed a direct consequence of the random choice of points in **Algorithm 1**. We made the choice to keep the natural behaviour of the networks by choosing a neutral and mutual set of points. In fact, we could try to construct specific sets of points for each network with a balanced distribution of predictions among the 5 labels. For our study, we decided to keep the natural behaviour of the networks rather than to force them on some parts of the inputs domain.

4.2.3 Deviation study of some specific neural networks

In **section 4.4.3**, we were able to identify some specific networks which retained our attention many times like networks 1-1, 2-1, 3-1 and 3-6. Now, we present a way to condense the information concerning them. For each neural network, we create a graph for each of the 5 labels. Each graph gives an indication of the deviation of a specific label when the attack's intensity increases. The following figure gives an example of such a representation for network 1-1 :

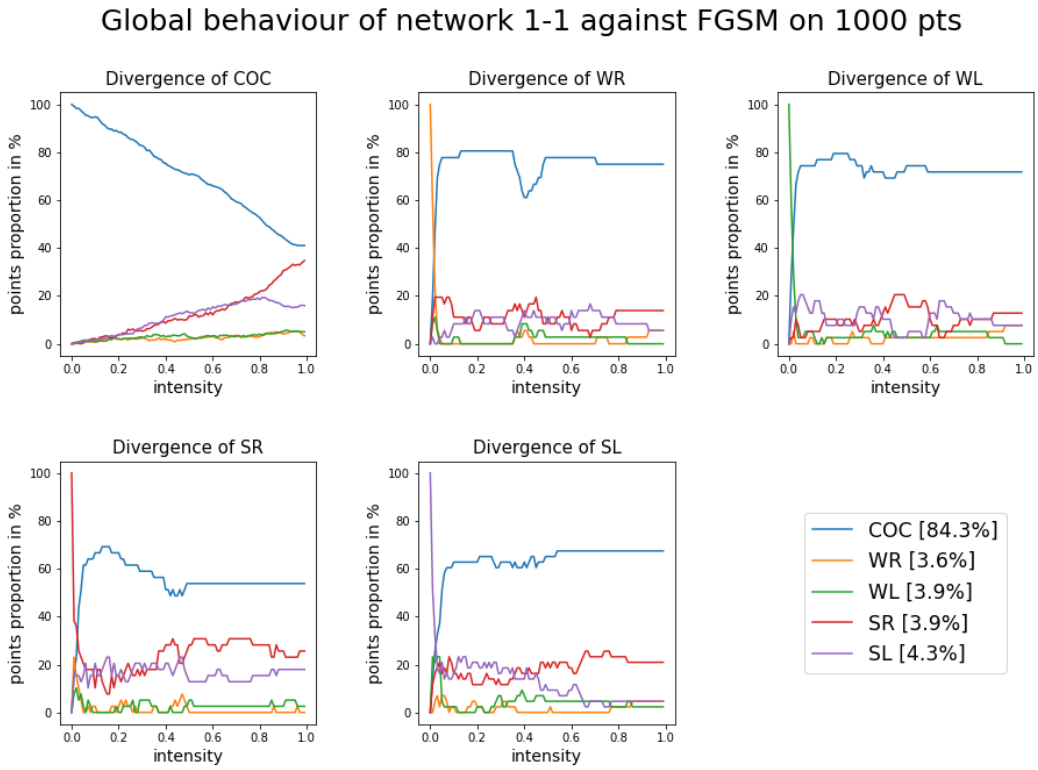


Figure 15: Deviation curves for FGSM on network 1-1 with 1000 points

On the bottom right of **figure 18**, we give the initial layout of the predicted labels among the base points. As always with ACAS-Xu networks, a great majority of points is predicted as COC. Therefore, curves on the top left graph giving the divergence of COC appeared much more smooth than all the other curves in **figure 12**. That said, we can make the following remarks about the graphs :

- Looking at all the COC (blue) curves on the 5 graphs, one can see that they reach a kind of stationary level for all labels except for COC itself. If we focus on these stationary level, we can see that it's located between 60% and 80% for weak moves as WR and WL. However, SL reaches roughly 60% and SR is even below that level. In fact, there is almost a gap of 30% between the end levels of WR and SR which are the only two moves to the right. Moreover, WR and SR present strange irregularities for specific values of intensiy. In fact, the COC curve in the SR graph reaches a peak at almost 70% for $\epsilon \approx 0.2$ and then decreases until $\epsilon \approx 0.4$. On the contrary, the COC curve in WR presents a reverse peak around $\epsilon \approx 0.4$ temporary loosing 20%. At last, looking at the COC curve in the COC graph, one can be surprised by the linear shape. In fact, the proportion of points that are still predicted as COC after an attack of FGSM drops from 100% for 0 intensity to 40% for maximal intensity at a rate of approximately 6% every 0.1 units of intensity. From that behaviour, we can now claim that **as long as the intensity of FGSM is below 0.3, one has 80% chance that a point labelled as COC will not be mislabelled after the attack**.

- If we focus on other curves, we see much less diverse behaviours. In fact, if we consider the y curve in the y graph for each label y except COC, we can see that it drops very quickly under 20% (before $\epsilon = 0.1$). However, the SR curve in the SR graph seems to be a little more resistant than its counterparts with a slow rebound for intensities greater than 0.4. One can note that the low level of that SR curve matches the peak of the COC curve in the same graph. We can then conclude that **an attacker who wants to change a point from SR to COC with FGSM is more likely to be successful if the intensity of the attack is around 0.2**. Indeed, the SR curve never reaches any score below 10% for other values of intensity. On the contrary,the SL curve in the SL graph, slowly diminishes and crosses 10% for $\epsilon \approx 0.5$. Surprisingly, one can see that strong labels (SR and SL) are more resistant than weak ones (WR and WL) as the curves for WR and WL are below 5% for almost every value of intensity.

- We can also focus on transition curves meaning for example the SR and SL curves in the COC graph. Indeed, one can observe that **for almost any value of intensity, FGSM is more likely to change a COC point into a strong move label**. For high values of ϵ (¿0.8), it becomes even 2 times more likely that a COC point gets label SR than label SL. In fact, strong moves often present more complex behaviours as weak moves. In the SL graph for example, it's clear that a SL point is more likely to be sent on SR than on WR or WL. That can be very annoying because SR and SL moves are the most extreme ones leading to very different consequences in the real world. **Knowing that approximately 1 point over 5 is changed from SL to SR when applying FGSM with any intensity is worth taking into account**.

As we saw, from the representation given in **figure 18**, one can deduce many useful empirical properties on the neural networks. The figures above can also be analyses in a similar way :

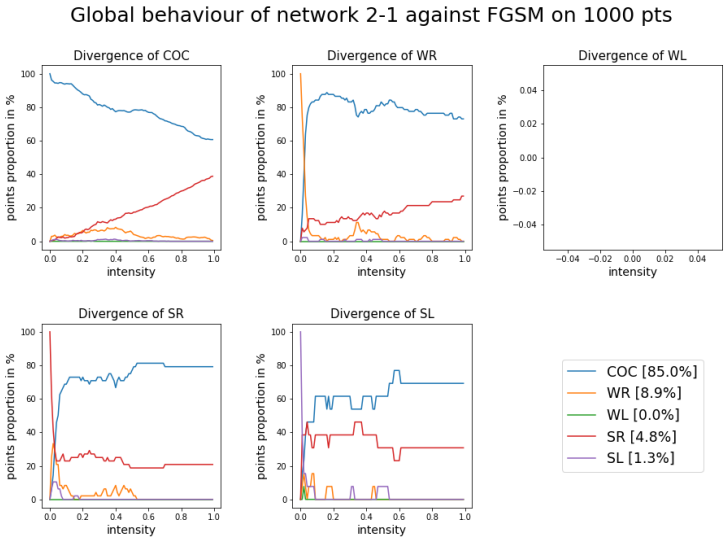


Figure 16: Deviation curves for FGSM on network 2-1 with 1000 points

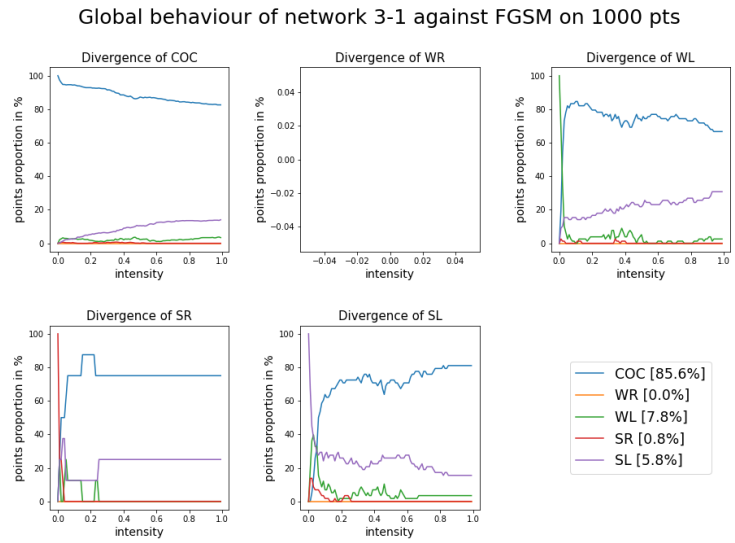


Figure 17: Deviation curves for FGSM on network 3-1 with 1000 points

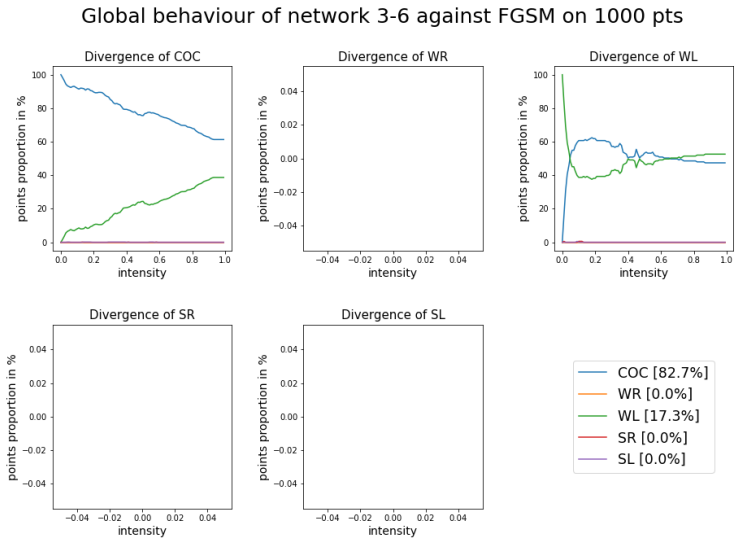


Figure 18: Deviation curves for FGSM on network 3-6 with 1000 points

One can note that some graphs are empty. It naturally comes from the fact that some networks as 2-1, 3-1 and 3-6 ignore some labels. Concretely, they don't predict some labels when we consider a amount of 1000 random points as we did here.

4.2.4 Clusters of adversarial points

We tried to see if, for a huge number of points (and adversarial points), the adversarial points were occupying the whole space or if there were clusters. Once again, we only used FGSM because it is much faster but, if we were provided better machines, we could do the same for other attacks.

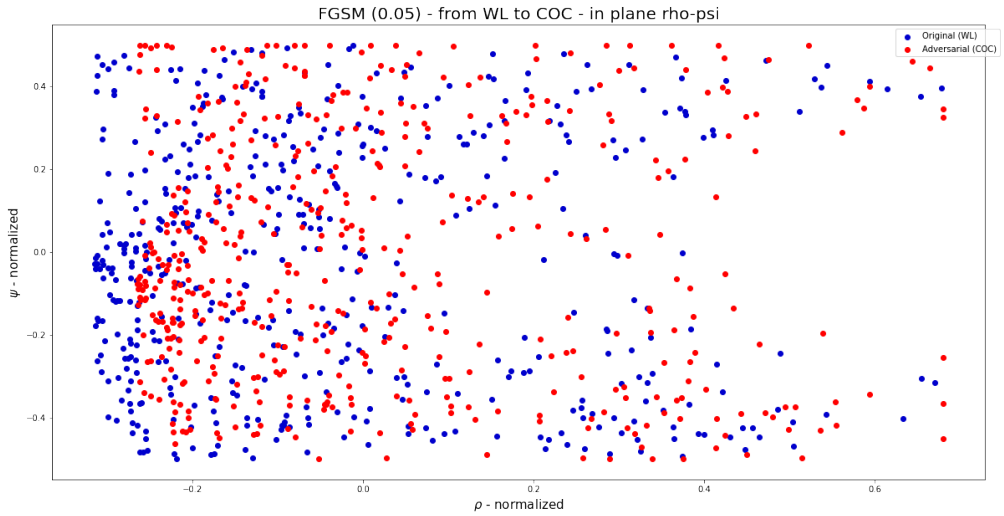


Figure 19: Original WL points(in blue) that became COC after the use of the FGSM attack.

What is interesting here is that most of the points are located in the left part of the graph. This means that the best place to find adversarial points would be for a normalized ρ value below 0. the next figure shows that this result is not due to the random points generator: there are as many light blue points in the left as in the right of the figure.

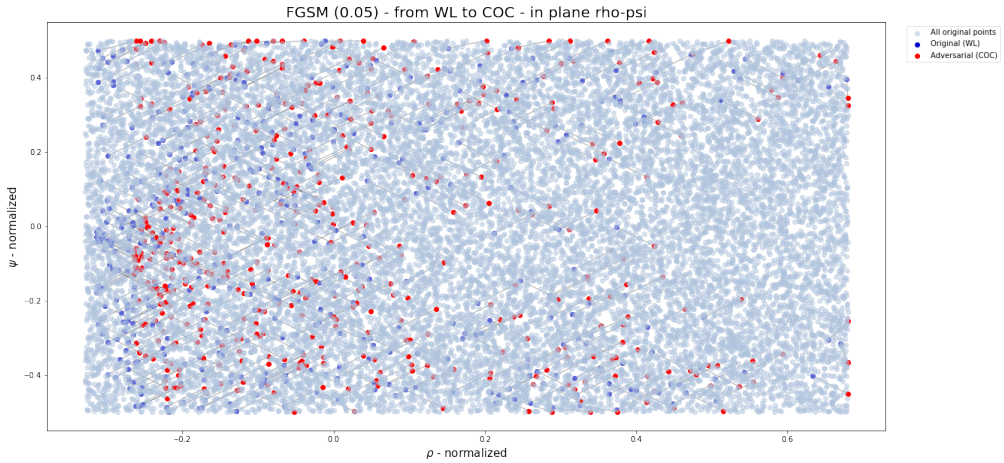


Figure 20: Same image but including the points on which the attack did not succeed. The black lines link a successfully-attack point and its adversarial version together.

4.2.5 From an original point to its adversarial version: evolution of the components

We tried to see how each of the 5 components of a point were affected by an attack. With the same attack parameters as above we obtained the following result:

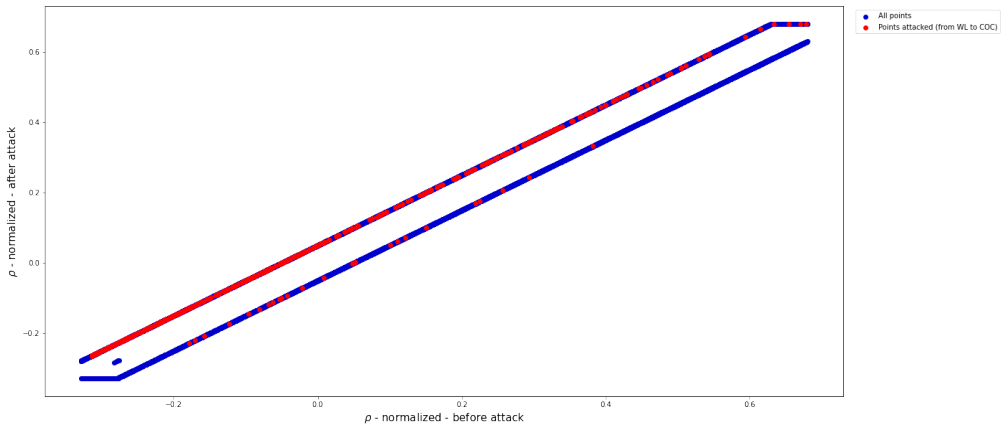


Figure 21: Evolution of the ρ component before and after the attack.

The straightness of the lines show once again that FGSM is a purely linear attack. Here we see that the attack had two possible influences on the rho component: either to increase or to decrease it (the two possibilities concerning about the same number of points). Among the ones that were successfully attacked, almost all had their rho component increased.
We tried to draw the same figure for a non-linear attack, but we can attack much less points due to time limitations and the results are not so convincing. This is the best we got when applying NewtonFool:

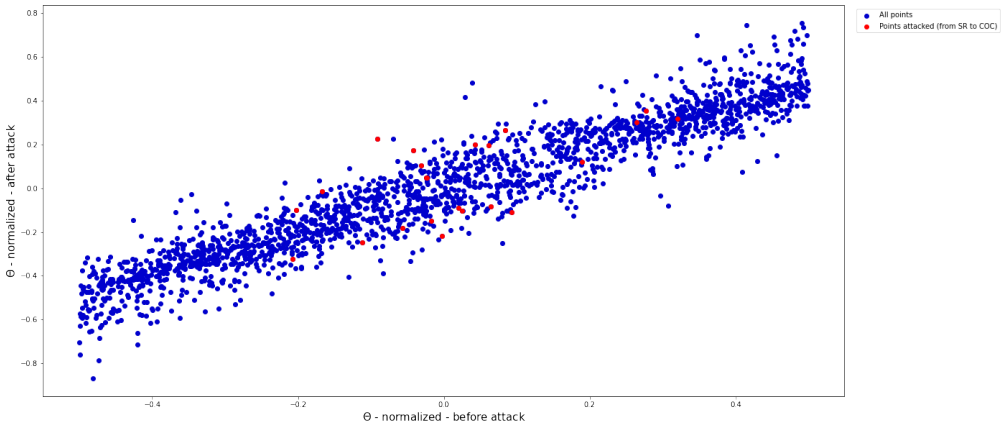


Figure 22: Evolution of the θ component before and after the attack.

We can barely say that the θ component did not change by that much and that the NewtonFool attack on extreme values of θ is rarely efficient. But more points are needed to conclude.

4.2.6 Analysis of the strengths of the attacks

We first studied how the strength of the attacks could change the result of the attack. So we decided to pick 1000 points and use different strengths of the FGSM attack.

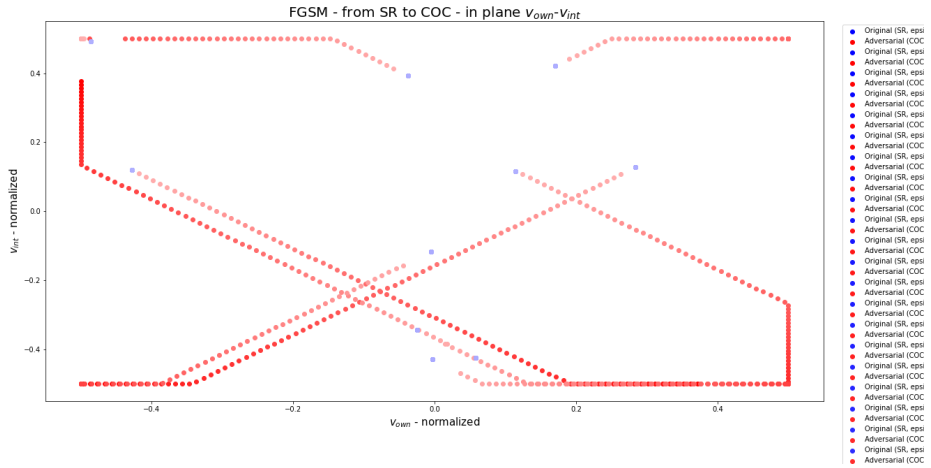


Figure 23: Set of initial SR points (depicted in blue) that become COC after the attack (red points). The more red the point, the more intense the attack.

We can see that:
- because the FGSM attack is linear, the red points draw lines. This is not the case for other attacks like NewtonFool.
- when it is too strong, FGSM pushes the points to the border of the domain.
- there is sometimes a gap between a blue point and the red dots. This means that the attack was no strong enough to change the SR point into a COC one.

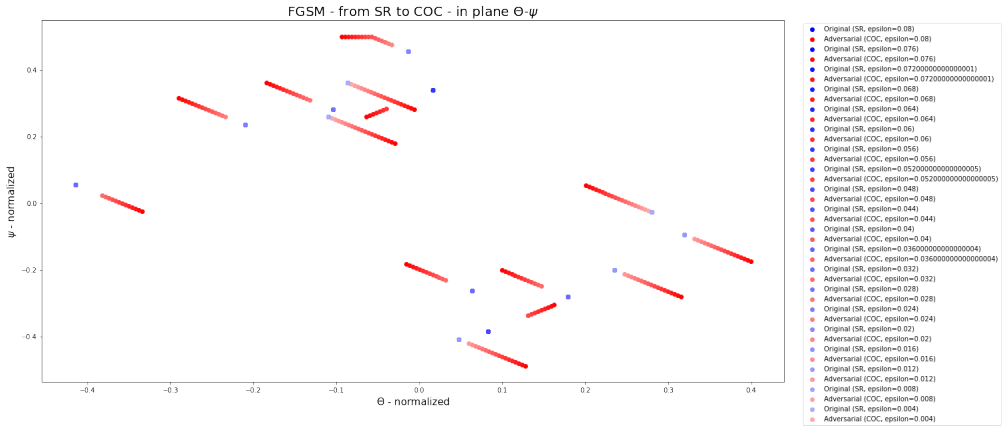


Figure 24: Same conditions but with lower attack strengths.

Here with 10 times lower strength values, we clearly see that some points are only affected when the attack strength is big enough. But, when the attack is stronger, the points are still affected. So, the higher the attack's strength is, the more points will be transfered to COC.

We also see that, for some points, even a very small perturbation can be enough to make the final output change. In the context or aircraft this is realistic: there is no need for the attack to change angles by 30° to change the final output. The attack can go unnoticed.

4.3 An attempt to apply our analysis method with iterative attacks

In the following, we will try to apply the same analysis method that we described with FGSM on iterative attacks namely : CarliniWagner, NewtonFool and DeepFool. We will analyse the computation constraints that we're facing with these kind of algorithms and give precise instructions on how to run our method with a given computing setup.

There are different factors that lead us to make the distinction between iterative and non-iterative attacks. First of all, the attacks studied by the S7-team and listed in **figure 3** were not all tailored for our task. For example, Wassertein attack was originally designed for attacking deep neural networks used for computer vision like CNN. Therefore, we had to select a subset of attacks to test on ACAS-Xu. We also realised that some attacks were built on direct algorithms whereas some others were using loops and iterative procedures. As we had already produced results for FGSM while implementing our analysis method, we chose to focus on iterative attacks. Indeed, for these kind of attacks, we would have to fit our analysis method to the computation time needed. In order to do that, we chose 3 famous iterative attaks named : **Carlini&Wagner**, **DeepFool** and **NewtonFool**.

The following figure presents the python code we used to load the attacks from the adversarial-robustness-toolbox library :

```python
def load_attack(method, acas_model, cw_conf=0.0, fgsm_eps=0.1, nf_eta=0.01, df_eps=0.1, miter=10):
    """ Choose your attack with 'method'. """

    if method == "fgsm":
        # epsilon [float] : absolute perturbation added to each component of a point to form an adversarial example.
        return(FastGradientMethod(estimator = acas_model,
                                  eps = fgsm_eps))

    elif method == "cw":
        # confidence [float] : higher produce examples that are farther away, but more strongly classified as adversarial.
        return(CarliniLInfMethod(classifier = acas_model,
                                 confidence = cw_conf,
                                 max_iter = miter,
                                 verbose = True))
    elif method == "nf":
        return(NewtonFool(classifier = acas_model,
                          eta = nf_eta,
                          max_iter = miter,
                          verbose = True))

    elif method == "df":
        return(DeepFool(classifier=acas_model,
                        epsilon = df_eps,
                        max_iter = miter,
                        verbose = True))

    else:
        raise Exception("'{0}' is not a valid attack.".format(method))
```

Figure 25: Python code used to load the attacks from the adversarial-robustness-toolbox (ART)

As one can see, we made it possible to load either FGSM, Carlini&Wagner, NewtonFool or DeepFool with specific parameters. FGSM has only one parameter : the intensity ϵ. That intensity parameter can also be found with the 3 iterative attacks : $confidence$ for Carlini&Wagner, eta for NewtonFool and also ϵ for DeepFool. However, these 3 iterative methods also have a supplementary feature called max_iter and that is the one we want to focus on. There are also some other technical parameters tailored to each attack but we will not take them into account here for simplicity reasons. They will just stay fixed at their usual value given in the original papers and the adversarial-robustness-toolbox (ART).

An iterative attack means that, in order to produce a adversarial point from a base point, the algorithm has to enter a loop which will run as long as the procedure used for the specific attack has not found the expected point. The drawback of this technique is that there is no guarantee for the algorithm to end in a reasonable time. Therefore, the loop condition is changed to allow a maximal number of loops in the process and that is precisely the key role of parameter max_iter.

For each iterative attack, there exists a default value for max_iter which is also given in the ART. For the iterative methods at hand, we get $max\_iter \in [10, 100]$. Needless to say that choosing a max_iter that is too low would undermine the efficiency of the attack leading to downplay the dangerosity of an attack. On the

other hand, a high value would lead the algorithm to run for a long time preventing us from applying it to a large enough number of points. Therefore, we have to find a compromise between efficiency and computation time. To do this, we decided to model the execution time of an iterative attack as a function of the parameter *max_iter* when applying the attack on the networks.

For the sake of simplicity, we made several hypothesis :

- The execution time of an iterative attack doesn't depend on the intensity parameter. In fact, the intensity is often used as a multiplicative factor, hence its value doesn't really impact the computation time. For the experiments, we then chose to fix the intensities to their default values given in ART.
- The execution time of an iterative attack doesn't depend on the network to attack. In fact, as all the networks don't have the same behaviour, it can be criticized. However, we will model the execution time with network 1-1 as it one of the most interesting one.

In order to get the execution time of an iterative attack, we had to consider several points in order to smooth little variations by taking the mean time. We quickly realised that iterative attacks were so time consuming that it was not possible to consider more than 100 points even for $max\_iter = 10$. Therefore, we randomly chose 100 points to run the tests. The following figure presents the results of the experiments conducted with 10 equally spaced values from 10 to 100 for *max_iter* :

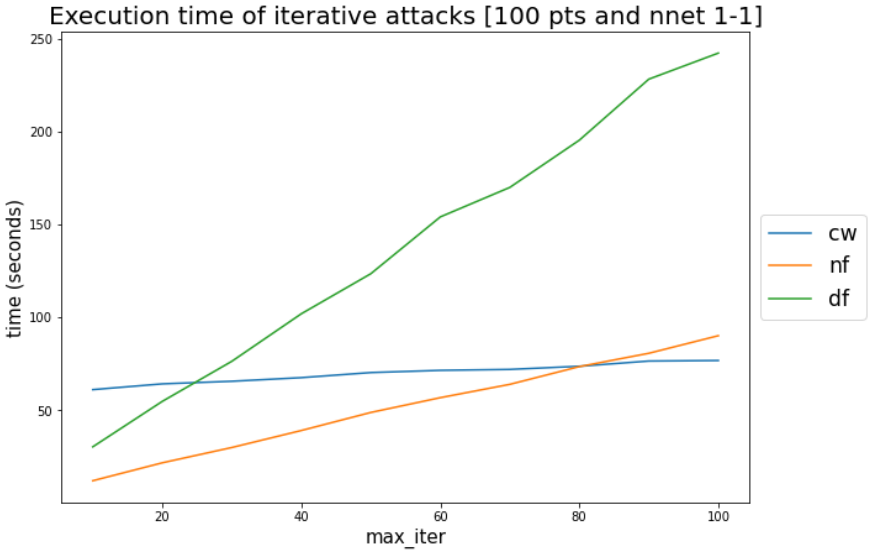


Figure 26: Experiments on the execution time of iterative attacks as a function of parameter *max_iter*

As expected, the execution time increases when allowing more iterations in the loop of the attack's algorithm. However, the 3 attacks don't have the same behaviour. It's very tempting to see a linear relation on the previous figure. Therefore, we decided to fit a linear function to quantify the law between execution time and *max_iter*. For each attack i, we were able to find coefficients a_i and b_i in order to model the execution time as it follows :

$$execution\_time_i = a_i * max\_iter_i + b_i$$

The following tab gives the coefficients (a_i, b_i) that we deduced from **figure 26** :

Attack	a_i	b_i
Carlini&Wagner	1.7	60.6
NewtonFool	8.6	4.5
DeepFool	24.0	6.0

Table 8: Coefficients of the linear model of the execution time as a function of *max_iter*

Remember that all the values are given for 100 points as it is hard to really give the time needed for only one point. Thanks to the model, we can now give an estimation of the time needed to apply our method to these iterative attacks. Unfortunately, we then understood that if we were to consider the same amount

of points (1000 pts), the precision for the range of intensities ($\approx$ 100 values), we would need from 20 hours with Carlini&Wagner to roughly 70 hours for DeepFool and that with *max_iter* fixed at 10 (which is its lowest reasonable value) and for only one neural network. Therefore, we decided to reduce the range of intensity from 100 to 10 values. As it was not enough, we had to reduce the number of points to test. From there, we reversed the time relation in a way that knowing the amount of time we have, we can compute the maximal number of points to test. The following tab gives an insight of the possible configurations and the corresponding number of points allowed for a single network :

Attack	**Intensity range**	**max_iter**	**Max time**	**Number of points**
Carlini&Wagner	10	10	0.5h	232
Carlini&Wagner	10	10	2h	928
Carlini&Wagner	10	10	7h	3248
NewtonFool	10	10	0.5h	199
NewtonFool	10	10	2h	797
NewtonFool	10	10	7h	2789
DeepFool	10	10	0.5h	73
DeepFool	10	10	2h	293
DeepFool	10	10	7h	1026

Table 9: Computing settings and execution time

Therefore, we can't hope to run our method in a reasonable time being less than 1h per network. Indeed, as our approach is a statistical one, we cannot adjust our number of points to the execution time. That would lead us to choose between 50 and 200 points which is not sufficient if you want to grasp the behaviour of a neural network with 5D inputs and outputs. However, it doesn't put a term to our method. Indeed, as our strategy is highly parallel, it's perfectly tailored for clusters computing or cloud computing. Unfortunately, it was not possible for us to set up this dimension of the project considering the time at our disposal.

4.4 A classic metric to quantify robustness independently of the attacks

4.4.1 Introduction of the method

As we saw, our attack-based analysis method requires a certain amount of time and computing power that we cannot afford. Therefore, we also chose to study the robustness ACAS-Xu networks independently of the attacks but with what is called a robustness metric, which will be a values assessing the global robustness of a network.

We also want a way to evaluate the robustness of a network independently of the attack, that's why we chose this method, which is based on linear optimization to find the minimal perturbation needed in order to get a certain label starting from an input point.

As described in part 2.2, we will used statistical methods to evaluate those metrics : $\phi(f, \varepsilon)$ the ε-adversarial frequency of the network and $\mu(f, \varepsilon)$ the ε-adversarial severity of the network.

4.4.2 Technical implementation

To be able to compute the functions described in part 2.2, we need a linear optimizer. We chose the python linear optimizer Gurobi. For more details on the technical implementation, the Jupiter notebook is given in the deliverables.

We get rid of the last layer (softmax) because it is non linear and we don't need it in order to know which label is chosen by the network (the stronger score), it is only used to have a probability.

First, we checked that we find the same result as what the network answers by simulating each layer manually thanks to the weights. Once this verification done, we proceeded to code the function computing $\epsilon(f, x, l)$. We then compute $\rho(f, x)$.

Repeating these computations to many input points x, we are able to estimate $\phi(f, \varepsilon)$ and $\mu(f, \varepsilon)$.

We also compute $\phi_l(f, \varepsilon)$ and $\mu_l(f, \varepsilon)$ corresponding to the ε-l-targeted-adversarial frequency and the ε-l-targeted-adversarial severity : in the formula of $\phi(f, \varepsilon)$ and $\mu(f, \varepsilon)$ we replace $\rho(f, x)$ by $\epsilon(f, x, l)$. We did this to see what happens if we work on targeted attacks instead of untargeted ones.

We draw graphs plotting the value of ϕ and μ for the untargeted attack and for each targeted attack, in function of the maximal allowed perturbation ε.

This gives us a global idea of the robustness of each network. The computations were very long because of the complexity of the constrained optimization algorithm, but we managed to have good results by doing the statistics on 120 points, on 4 different networks. The points chosen were a subset of the points used to study the FGSM attack.

4.4.3 Possible ways to go further

Of course, the algorithm can be run again on more powerful machines with more points and more networks to have a more precise idea of the robustness of the system.

Furthermore, as described in the state of the art, we can use the same method to find adversarial points to the properties $\phi_1, \cdots, \phi_10$, and for that we need to rewrite them as linear constraints. This could in theory be done without much difficulties but we didn't have the time to tackle the issue. However, the hard part of the job is done, only some constraints must be changed, this gives us the RELUPLEX method.

4.4.4 Results

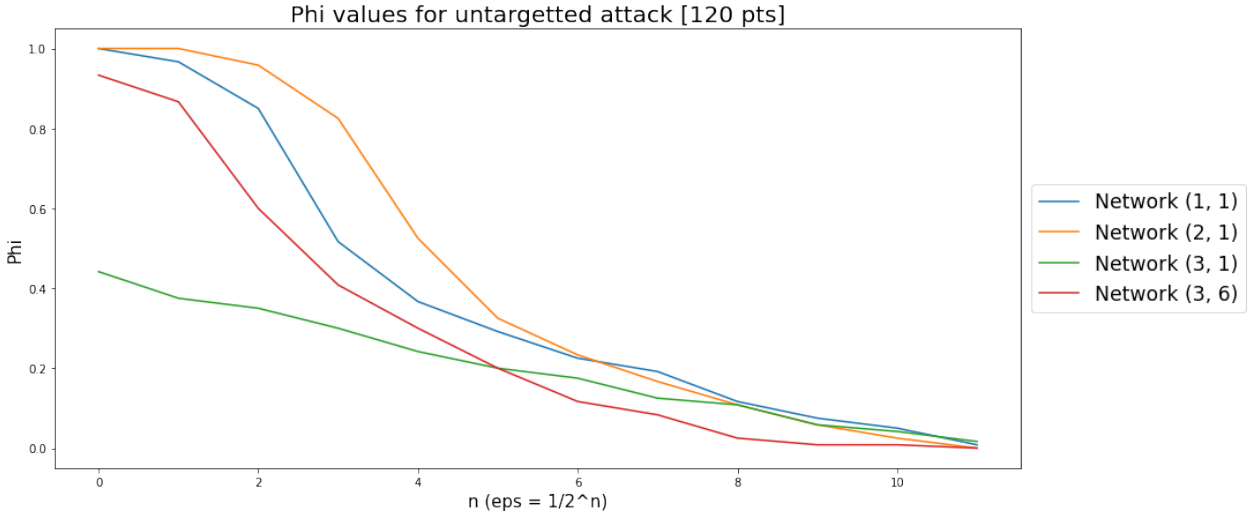
For example, we see that for untargeted attacks, the network (3,1) is a lot more robust that the 3 other ones for strong attacks but it is comparable for more subtle attacks. Similarily, the network (2,1) is less severely perturbed by the strong attacks than the other networks.

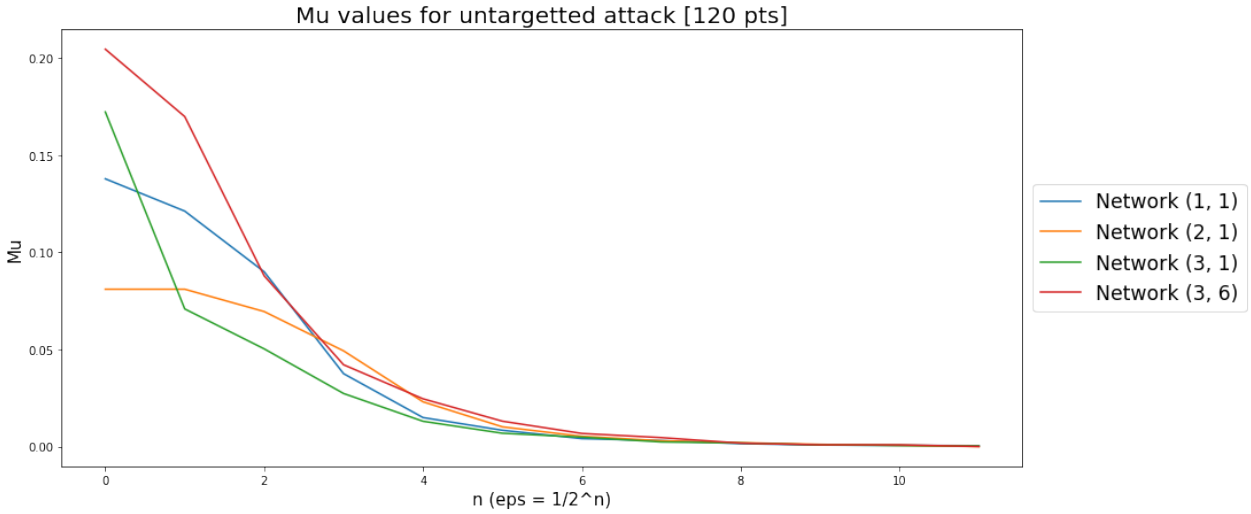
There are some strange behaviours at first sight, for example the WL-targeted attack on the network (2,1) seems to be working each time, no matter how weak the attack is : ϕ stays at 1. This is because the

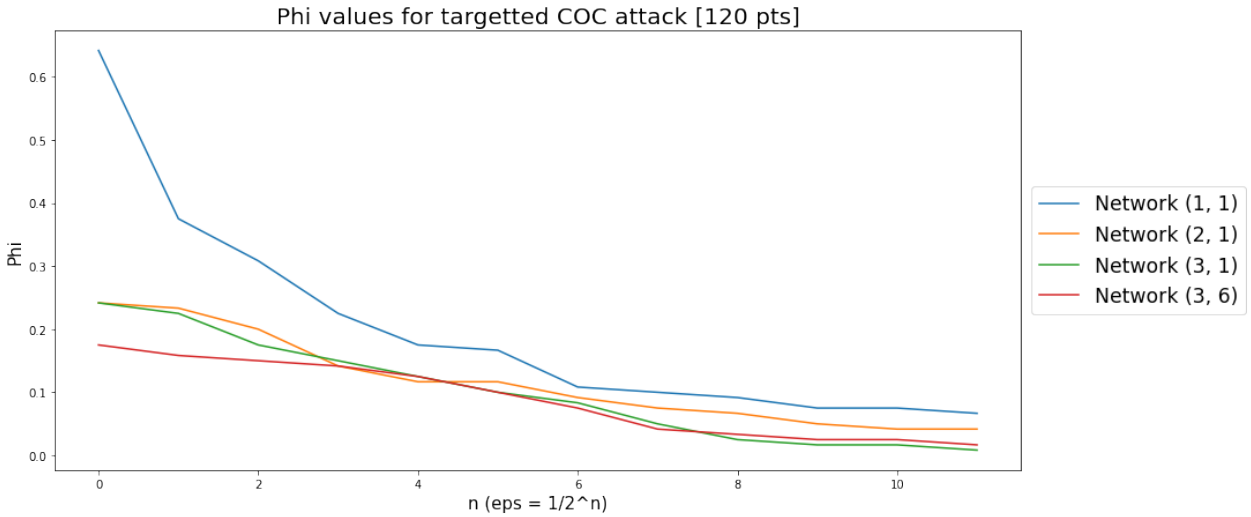
network (2,1) maps many points to WL, so it is very easy to target this label : it is already the good label, that's why we see that the μ associated is zero.

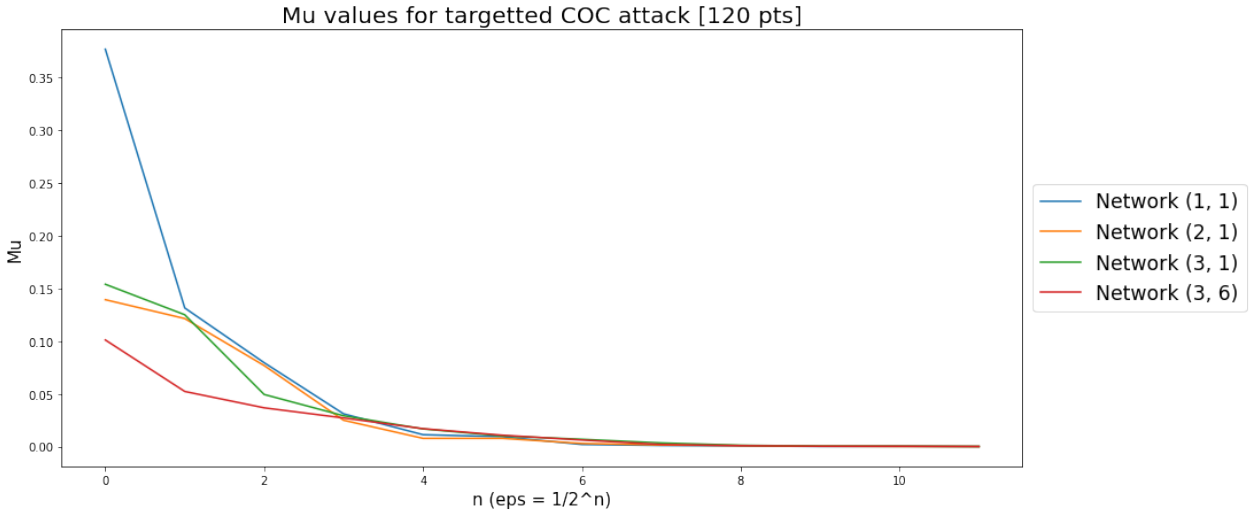
This is a good way to quantify globally how robust a network is depending on the allowed perturbation. It allows to compare networks together and to measure the robustness independently from the attack : the optimizer finds the exact smallest perturbation needed, it is not gradient-based.

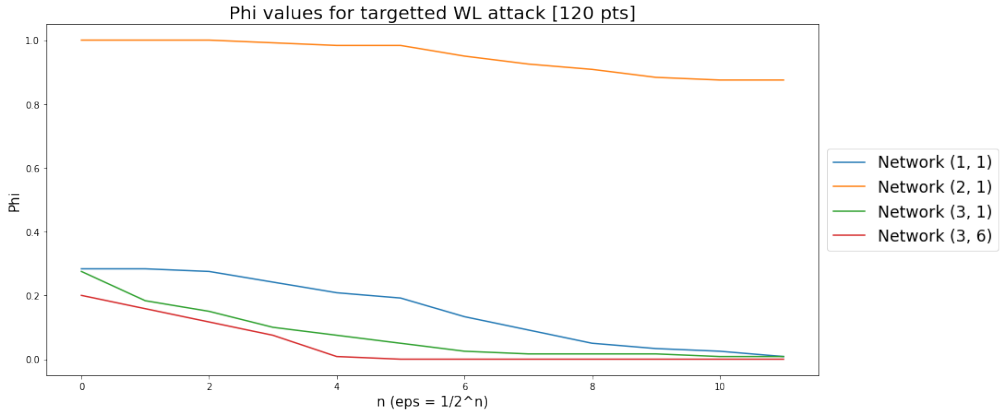
The 12 graphs obtained for the untargeted and the targeted attacks can be found on the following pages, as well as the pseudo-code for the algorithm to compute $epsilon(f, x, l)$ using the linear optimizer :

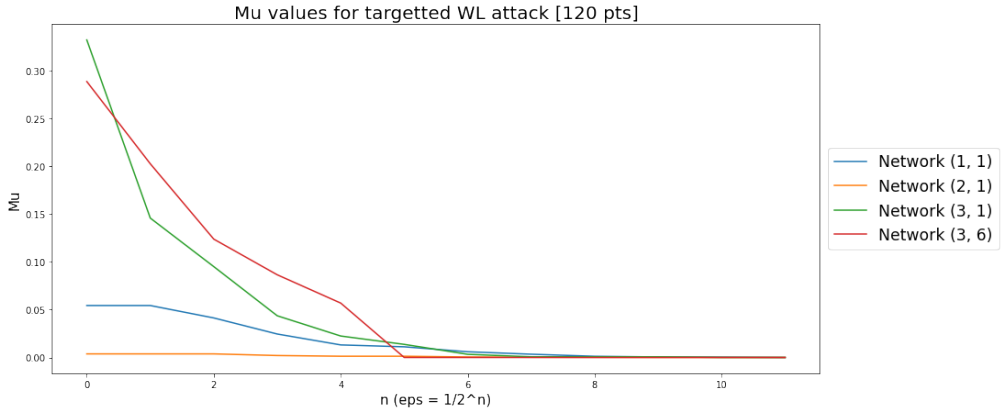


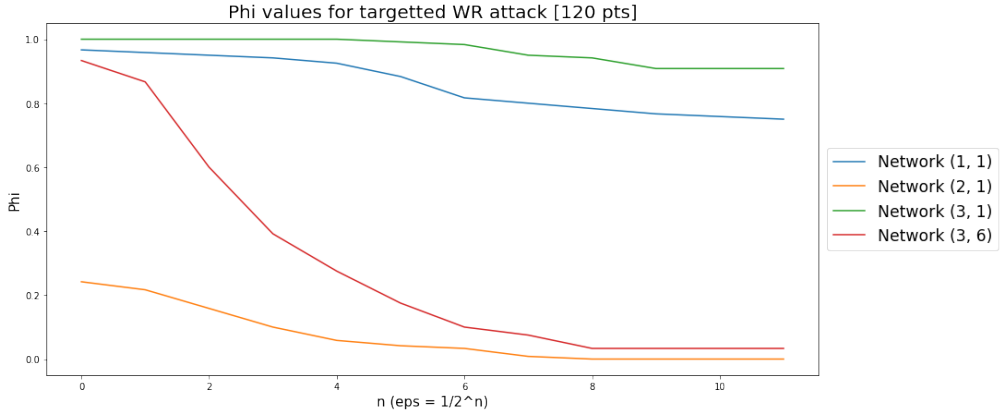


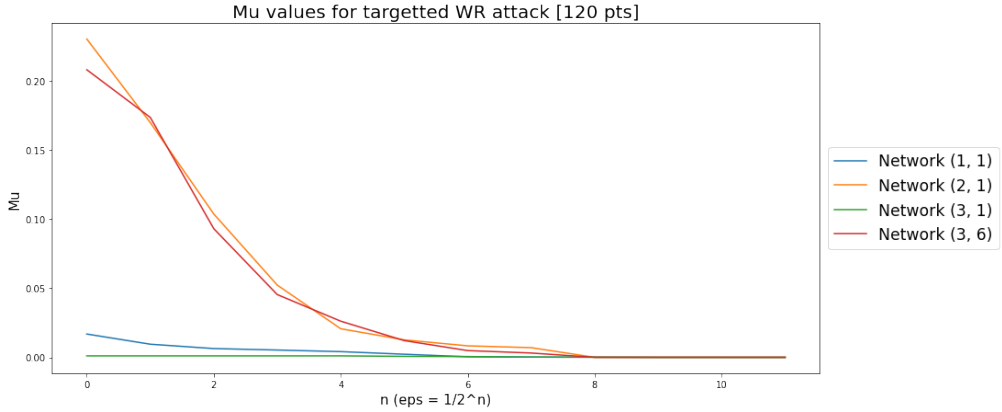

Phi values for targetted COC attack [120 pts]
Phi
n (eps = 1/2^n)
Network (1, 1)
Network (2, 1)
Network (3, 1)
Network (3, 6)

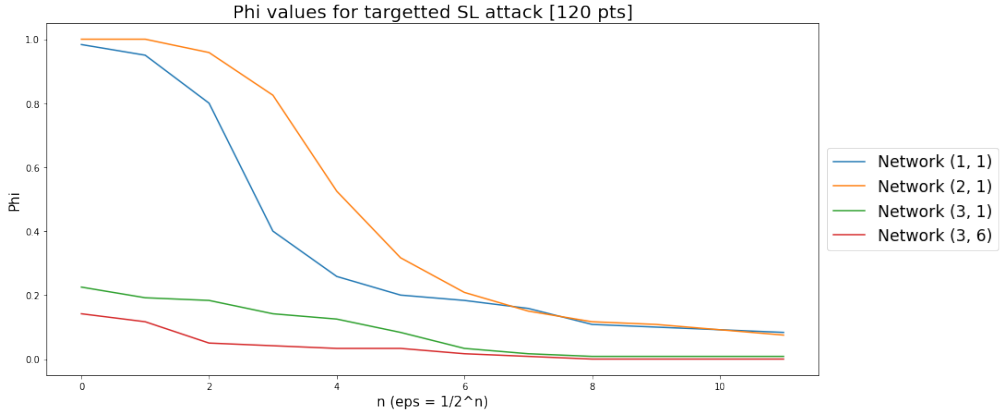

Mu values for targetted COC attack [120 pts]
Mu
n (eps = 1/2^n)
Network (1, 1)
Network (2, 1)
Network (3, 1)
Network (3, 6)

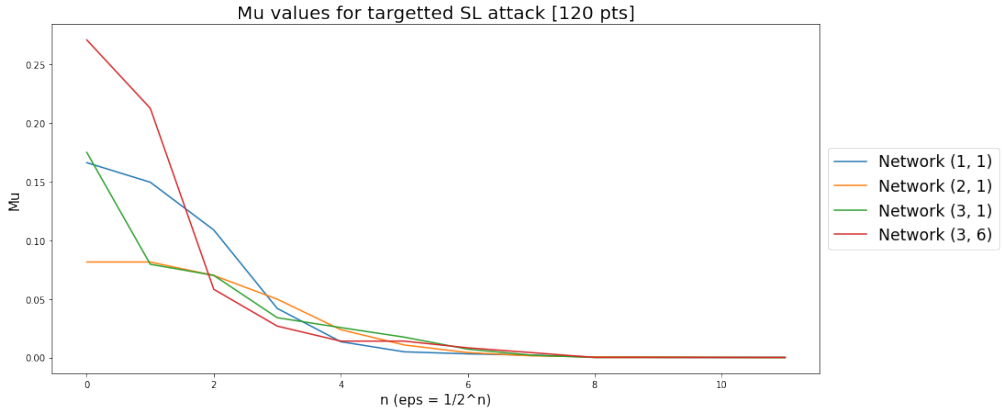

Phi values for targetted WL attack [120 pts]
Phi
n (eps = 1/2^n)
Network (1, 1)
Network (2, 1)
Network (3, 1)
Network (3, 6)

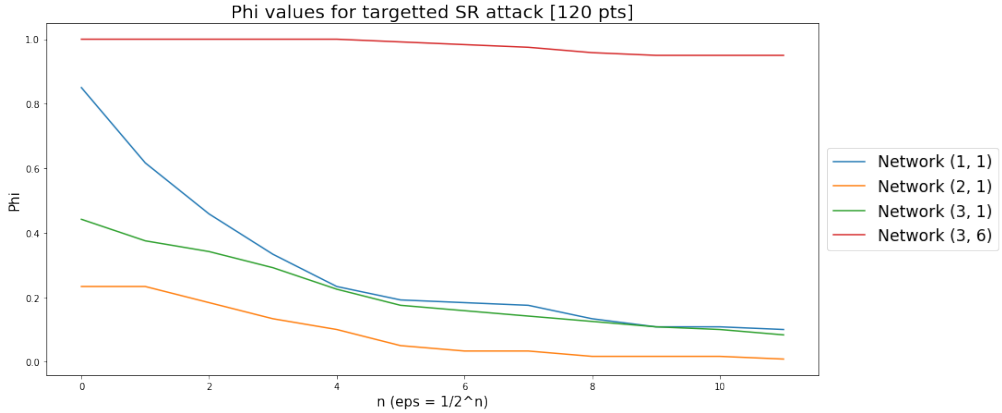

Mu values for targetted WL attack [120 pts]
Mu
n (eps = 1/2^n)
Network (1, 1)
Network (2, 1)
Network (3, 1)
Network (3, 6)

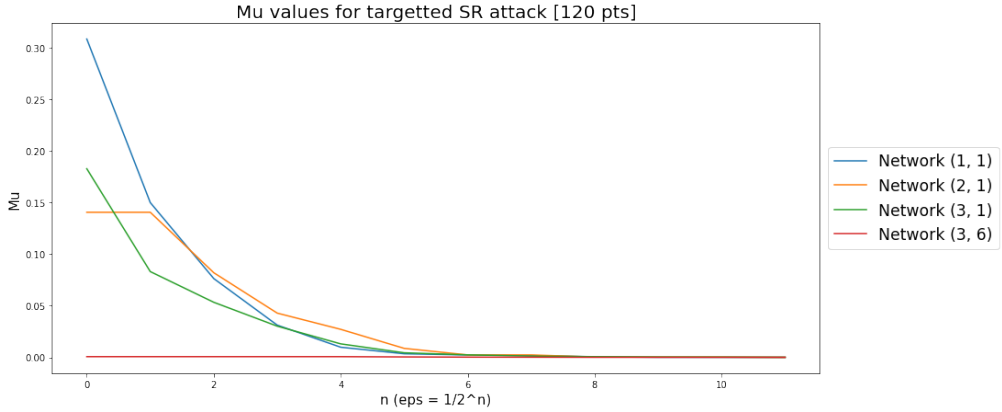

Phi values for targetted WR attack [120 pts]
Phi
n (eps = 1/2^n)
Network (1, 1)
Network (2, 1)
Network (3, 1)
Network (3, 6)


Mu values for targetted WR attack [120 pts]
Mu
n (eps = 1/2^n)
Network (1, 1)
Network (2, 1)
Network (3, 1)
Network (3, 6)


Phi values for targetted SL attack [120 pts]
Phi
n (eps = 1/2^n)
Network (1, 1)
Network (2, 1)
Network (3, 1)
Network (3, 6)


Mu values for targetted SL attack [120 pts]
Mu
n (eps = 1/2^n)
Network (1, 1)
Network (2, 1)
Network (3, 1)
Network (3, 6)


Phi values for targetted SR attack [120 pts]
Phi
n (eps = 1/2^n)
Network (1, 1)
Network (2, 1)
Network (3, 1)
Network (3, 6)


Mu values for targetted SR attack [120 pts]
Mu
n (eps = 1/2^n)
Network (1, 1)
Network (2, 1)
Network (3, 1)
Network (3, 6)

Algorithm 2 Compute $\epsilon(f, x, l)$

Require: a model f ; an input x ; a target label l
Ensure: t the minimal perturbation needed to attack the network with the targeted label

$n \leftarrow$ number of inputs in f
$m \leftarrow$ number of layers in f
$l_i \leftarrow$ number of weights of each layer i

Decision variables : t perturbation (float), $(x'_i)_i$ input after attack (float), $(y'_{j,k})_{j,k}$ output of each layer after attack (float)

$w \leftarrow get\_weights\_of\_layers(f)$
$y \leftarrow get\_outputs\_of\_layers(f, x)$

for $i = 0$ to $n - 1$ **do**
    Add constraint $x_i - x'_i \leq t$
    Add constraint $x'_i - x_i \leq t$
end for

for $k = 0$ to $l_0 - 1$ **do**
    Add constraint $y_{0,k} = b_0 + \sum_{l=0}^{l_0} w_{0,l} * x_l$
end for
for $j = 1$ to $m - 1$ **do**
    if j is activation layer **then**
        for $l = 0$ to $l_{j-1} - 1$ **do**
            if $y_{j-1,l} < 0$ **then**
                Add constraint $y_{j,l} = 0$
            else
                Add constraint $y_{j,l} = y_{j-1,l}$
            end if
        end for
    else
        for $k = 0$ to $l_j - 1$ **do**
            Add constraint $y_{j,k} = b_j + \sum_{l=0}^{l_j} w_{j,k} * y_{j-1,l}$
        end for
    end if
end for

for $i = 0$ to l_{m-1} **do**
    if $i \neq l$ **then**
        Add constraint $y_{m-1,i} \leq y_{m-1,l}$
    end if
end for

Set objective : minimize t

Optimize

5 Reflection

5.1 Skills acquisition

Long-term projects allow us as students to develop and study something deeply. Together, we have created and developed substantial information of how adversarial attacks affect the ACAS-Xu system. The data that we gathered can create value for the client because it can be used to adjust and tweak the system to not fall for these attacks. The research that we have done can be used to promote the validity of the system. While it might not be everything that the client needs, it is a starting point in testing the system and making it the most efficient that it can be.

This project has also allowed us to work with others to grow and develop soft skills. Soft skills are just as important as the physical, concrete work that is produced in a group. During the last semester, we have learned how to work as a group to establish a working relationship with each other and the client. We started out with very ambitious goals for this project: testing multiple attacks on 45 different neural networks. Over time, we realized that this might be a bit too enthusiastic to accomplish in one semester. Instead, we adapted and honed our research. While this was not our original goal, we communicated with the client and figured out the research that would be valuable to them. We effectively were able to come to a solution that worked best for us and the client through open and clear communication. While it might be exactly what we originally set out to do, we were able to convince and work with the client to find a solution that we are capable of finding that is also useful for SystemX.

5.2 Ethics aspects and social responsibility of our work

In addition to the concrete work that we produced with our project, there is also the other aspect of ethics and principle that come into play. While working together, we have been able to create a model that can help others in the future. It has a positive impact for those who work with ACAS-Xu. Since we studied and analyzed how the system would react to different adversarial attacks, we have a sense of understanding on how it will act in future situations. This, not only, lets us understand its weaknesses and educate the system on the attacks, but it also fosters a safer environment for those working with and around the unmanned vehicles. Studying adversarial attacks on the system increases the stability of the system and, therefore, making the model more stable and safe, lowering the chances of unexpected situations. The study of basic or well known attacks can lead to a weakness to a "blind spot attack", where the input resides in low density regions or "blind spots". As the studies on adversarial attacks continue, these attacks must also be taken into consideration to ensure that there are no surprise attacks on the system and it is truly less susceptible to all attacks.

Since we are working to create a safer environment for this working with and around the unmanned vehicles, there is a moral responsibility to ensure that we are producing real, honest work. We are expected to create a model and involve ourselves in a project that is reliable and constructive to the client. It is critical of us to have accurate data from our research because working with unmanned devices can be dangerous, especially when working with a system whose purpose is to avoid collisions. Having authentic data allows us to make predictions and learn how the system will work, so having inauthentic data can put people and devices in danger of getting hurt or damaged.

6 Bibliography - List of figures - List of tables

[1] - **An introduction to ACAS-Xu and the Challenges Ahead** - *Guido Manfredi, Yannick Jestin* - ENGIE Ineo - Sagem UAS Chair - 2016

[2] - **Policy Compression for Aircraft Collision Avoidance Systems** - *Julian, Lopezy, Brushy, Owenz, Kochenderfer* - Stanford University - 2016

[3] - **Robustesse de l'IA face aux attaques adverses** - *Grégoire Desjonqueres, Aymeric Palaric, Victor Fernando Lopes De Souza and Amadou Sékou Fofana* - Projet S7, Pôle IA, CentraleSupélec - 2022

[4] - **ACAS-Xu Properties** - Provided by *IRT SystemX*

[5] - **DeepSafe : A Data-driven Approach for Checking Adversarial Robustnes in Neural Networks** - *Gopinath, Katz, Pasareanu, Barrett* - Carnegie Mellon University, Stanford University - 2020

[6] - **A Survey of Privacy Attacks in Machine Learning** - *Maria Rigaki, Sebastian Garcia* - University of Prague - 2021

[7] - **A drone program taking flight** - *Jeff Wilke : former CEO of Amazon Worldwide Consumer* - 2019 : https://www.aboutamazon.com/news/transportation/a-drone-program-taking-flight

[8] - **Recent Advances in Adversarial Training for Adversarial Robustness** - *Bai, Luo, Zhao, Wen, Wang* - Nanyang University, Wuhan University - 2021

[9] - **Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks** - *Guy Katz, Clark Barrett, David Dill, Kyle Julian and Mykel Kochenderfer* - Stanford University, USA - 2007

[10] - **Explaining and Harnessing Adversarial Examples** - *Goodfellow, Shlens, Szegedy* - Google Inc., Mountain View, CA - 2015

[11] - **Theoretical evidence for adversarial robustness through randomization** - *Rafael Pinot, Laurent Meunier, Alexandre Araujo, Hisashi Kashima, Florian Yger, Cédric Gouy-Pailler, Jamal Atif* - 2019

[12] - **Metrics and methods for robustness evaluation of neural networks with generative models** - *Igor Buzhinsky* - ITMO University, St. Petersburg, Russia - 2020

[13] - **Measuring Neural Net Robustness with Constraints** - *Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, Antonio Criminisi* - 2017

List of Figures

List of Tables