Résistance aux attaques et corrections adversaires

Résumé

Ce travail est motivé par l'observation de deux phénomènes intéressants dans le fonctionnement d'un algorithme particulier d'attaque adversaire. Premièrement, on observe une corrélation entre la difficulté à effectuer cette attaque (quantifiée par le concept de résistance) et la justesse de la prédiction du réseau. On en déduit alors une nouvelle expression de l'assurance d'un réseau classificateur. Deuxièmement, l'attaque adversaire proposée, lorsqu'elle est effectuée à partir d'une image incorrectement classifiée, produit presque toujours une image dont la catégorie est celle de l'image initiale.

L'exploitation combinée de ces deux phénomènes conduit naturellement à une méthode pour augmenter la précision d'un réseau classificateur quelconque. Le manque de finesse de la résistance nous pousse à introduire un réseau discriminateur, qui cherche à prédire si le classificateur se trompe ou non, pour ensuite corriger cette erreur par correction adversaire.

1. Les attaques adversaires

1.1 Les exemples adversaires

Les réseaux de neurones sont notoirement vulnérables aux *exemples adversaires* [1] : il s'agit d'entrées imperceptiblement perturbées pour induire en erreur un réseau classificateur.

Plus concrètement, en considérant $Pred$ la fonction qui à une image associe la catégorie prédite par réseau ; et en considérant une image img de $[0,1]^n$ (c'est à dire à n pixels N&B ou $\frac{n}{3}$ pixels RGB), on cherche une perturbation $r \in [0,1]^n$, de norme minimale, telle que :

— $img + r \in [0,1]^n$
— $Pred(img + r) \neq Pred(img)$

Dans toute la suite, on utilisera la norme euclidienne. D'autres normes sont évidemment possibles, mais sans amélioration sensible des résultats.

1.2 Les attaques adversaires

On s'intéresse à un algorithme qui détermine un exemple adversaire à partir d'une image donnée. On dit qu'un tel algorithme réalise une *attaque adversaire*.

Une méthode d'attaque possible est la suivante. Introduisons $Conf_c$ la fonction qui à une image associe la probabilité (selon le réseau) que l'image appartienne à la catégorie c ; et soit une image img, prédite de catégorie c par le réseau. On cherche alors à minimiser par descente de gradient (en utilisant l'algorithme `Adam`) sur r initialisé à 0^n la fonction $Loss_1$ suivante :

$$Loss_1(r) = \begin{cases} \|r\| & \text{si } Conf_c(img + r) \leqslant 0.2 \\ Conf_c(img + r) + \|r\| & \text{sinon.} \end{cases}$$

Cette première fonction est expérimentalement peu satisfaisante car l'attaque échoue presque toujours : la perturbation r reste "bloquée" en 0, et n'évolue pas. On pourrait corriger ce problème en initialisant la perturbation à une faible valeur aléatoire, mais cela enlèverait toute possibilité d'étudier la direction privilégiée par la descente de gradient. Pour pallier ce problème, on oblige alors la perturbation à grossir en ajoutant un troisième cas de figure quand $Conf_c(img + r) > 0.9$, c'est à dire quand la perturbation n'est pas du tout satisfaisante :

$$Loss_2(r) = \begin{cases} \|r\| & \text{si } Conf_c(img + r) \leqslant 0.2 \\ Conf_c(img + r) + \|r\| & \text{si } Conf_c(img + r) \leqslant 0.9 \\ Conf_c(img + r) - \|r\| & \text{sinon.} \end{cases}$$

Cette deuxième fonction est plus satisfaisante, même si elle échoue parfois. Son taux de succès est cependant acceptable (et elle n'échoue jamais là où cela sera important), et c'est donc celle-ci qui sera utilisée par la suite.

La Figure 1 montre le résultat d'une attaque adversaire : à gauche l'image originale, au milieu la perturbation et à droite l'image adversaire.

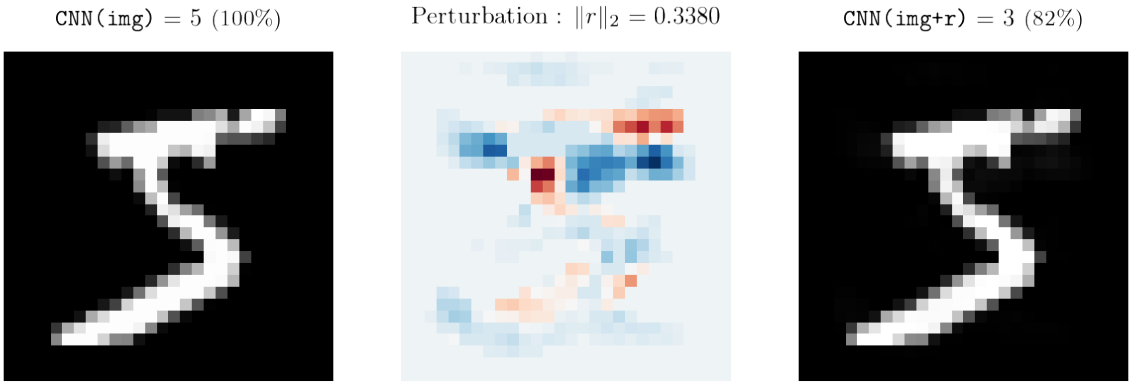

FIGURE 1 – Résultat d'une attaque adversaire sur une image de `MNIST`

On appellera $Pert_N$ la fonction qui à une image associe la perturbation obtenue après N étapes de descente de gradient (algorithme `Adam`, avec un taux d'apprentissage $\eta = 10^{-3}$).

1.3 Réseaux classificateurs et bases de données utilisées

On réalisera toute cette étude sur deux réseaux de type `AlexNet` (CNN avec Dropout) [2], appliqués respectivement aux problèmes de la classification des images de `MNIST` [3] et de `FashionMNIST` [4]. Les architectures utilisées pour ces réseaux sont détaillées dans l'Annexe A.

Les bases de données `MNIST` et `FashionMNIST` sont divisées de la manière suivante :

- 50000 images d'entraînement (`train`)
- 10000 images de validation (`val`)
- 10000 images de test (`test`)

Les réseaux sont entraînés à partir des images de `train`, et on utilise les images de `val` pour évaluer la généralisation de l'apprentissage, c'est à dire leur performance sur de nouvelles images. On ajuste alors les paramètres d'entraînement afin de maximiser la performance du réseau sur les images de `val`.

On étudiera les résultats sur les images de `test` dans ce qui suit, afin de travailler sur des images qui n'ont eu aucun rôle dans l'apprentissage du réseau, pour ne pas fausser les résultats.

On obtient les performances suivantes sur ces deux réseaux :

— 62 erreurs sur les 10000 images de `test` de `MNIST`,
— 876 erreurs sur les 10000 images de `test` de `FashionMNIST`.

Ces performances peuvent être améliorées avec des architectures plus efficaces, mais ces réseaux seront suffisant pout l'étude qui va suivre. Observons simplement que la classification de `FashionMNIST` est significativement plus difficile.

2. Résistance à une attaque

2.1 Images "faciles" et "difficiles" à attaquer

On réalise des attaques adversaires sur les images de `test`, en effectuant 500 étapes de descente du gradient de $Loss_2$, avec un taux d'apprentissage $\eta = 10^{-3}$, et on s'intéresse aux valeurs prises par $\|r\|$ et $Conf_c$ au cours de l'attaque.

La Figure 2 a été obtenue en attaquant deux images différentes de `test` de `MNIST`.

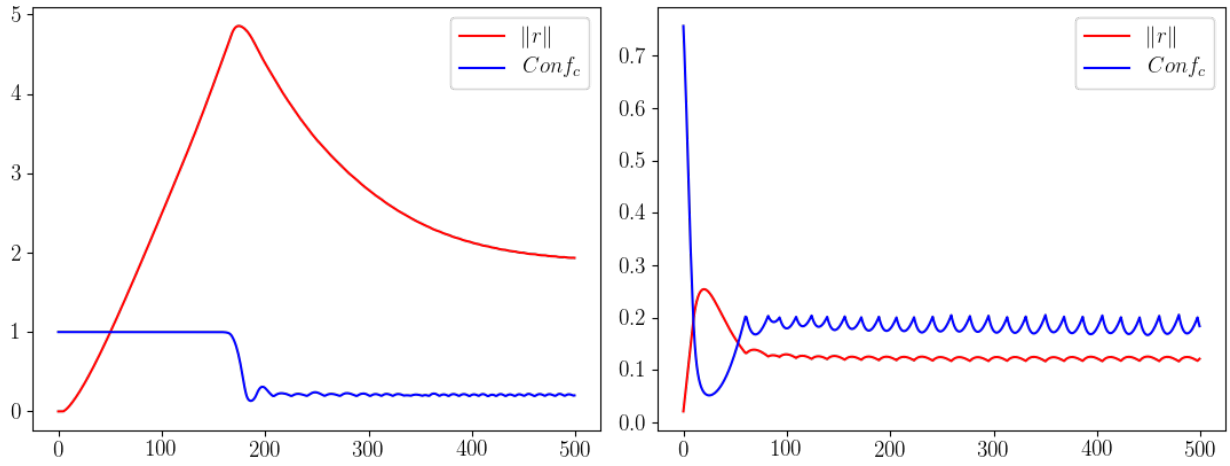

FIGURE 2 – Évolution de $Conf_c$ et de $\|r\|$ au cours d'attaques adversaires

Qualitativement, la norme de la perturbation augmente jusqu'à ce que $Conf_c$ passe en dessous de 0.9, à partir de quoi la norme diminue tout en gardant une valeur de $Conf_c$ stabilisée autour de 0.2.

L'image de gauche peut être qualifiée de "difficile à attaquer" : il a été nécessaire d'augmenter très fortement la norme de la perturbation pour réussir à casser la prédiction du réseau, ce qui ne se produit qu'après un grand nombre d'étapes, et la norme finale de la perturbation est élevée.

L'image de droite peut au contraire être qualifiée de "facile à attaquer" : bien moins d'étapes ont été nécessaires pour casser la prédiction du réseau, la norme finale est très basse, et le pic de très faible amplitude.

On voit nettement ici l'influence de la valeur du seuil à 0.2 dans la fonction $Loss$. Dès que $Conf_c$ est en dessous de 0.2, l'algorithme a pour seul objectif de réduire la norme de la perturbation, et fatalement $Conf_c$ repasse au dessus de 0.2. Il s'agit alors de réduire à la fois $\|r\|$ et $Conf_c$, jusqu'à ce que $Conf_c$ repasse en dessous de 0.2, etc.

D'autres exemples d'attaques d'images "faciles" ou "difficiles" à attaquer sont présentés dans l'Annexe B.

Résumons les principales différences qualitatives entre ces deux types d'images :

	Images “faciles”	Images “difficiles”
Pic	absent ou faible	haut
Étapes nécessaires	moins de 50	plus de 200
Norme de r finale	faible	élevée

Pour quantifier plus précisément cette difficulté à attaquer une image, introduisons le concept de *résistance*.

2.2 Quantification de la résistance à une attaque

Pour chaque image, on essaie de quantifier la résistance, du réseau à une attaque adversaire. Plusieurs définitions sont possibles, par exemple la norme de la perturbation minimale mettant en échec le réseau :

$$Res_\infty(img) = min\big\{\|r\| \ ; \ Pred(img + r) \neq Pred(img)\big\}$$

Cette expression de la résistance n'est que d'un faible intérêt en pratique, car incalculable. On utilisera donc plutôt les trois définitions suivantes :

Res_N la norme finale obtenue après un certain nombre d'étapes dans l'attaque adversaire :

$$Res_N(img) = \|Pert_N(img)\|$$

Res_{max} la hauteur du pic de la norme de la perturbation :

$$Res_{max}(img) = \max\big\{\|Pert_N(img)\| \ ; \ N \in \mathbb{N}\big\}$$

Res_{min} le nombre d'étapes qu'il a fallu pour abaisser $Conf_c$ à 0.2 :

$$Res_{min}(img) = \min\big\{N \in \mathbb{N} \ ; \ Conf_c(Pert_N(img)) < 0.2\big\}$$

Dans les cas où l'attaque échoue, on prendra systématiquement $Res = +\infty$.

2.3 Une corrélation avec la justesse de la prédiction

Les images attaquées dans l'Annexe B n'ont pas été choisies au hasard : les premières sont toutes classifiées correctement par le réseau, et les suivantes correspondent à des erreurs de classification. Étudions la généralisation de ces résultats, en observant la répartition des valeurs de la résistance sur des images correctement classifiées (notées `V`), et incorrectement classifiées (notées `F`) de `test`.

Sur `MNIST`, avec 500 images dans `V` et les 62 erreurs dans `F` :

`MNIST`	Res_N	Res_{max}	Res_{min}
90% de `V`	> 0.97	> 2.8	> 109
90% de `F`	< 0.57	< 1.3	< 58

Et sur `FashionMNIST`, avec 500 images dans `V` et dans `F` :

`FashionMNIST`	Res_N	Res_{max}	Res_{min}
80% de `V`	> 0.28	> 0.544	> 26
80% de `F`	< 0.29	< 0.543	< 25

Selon que les images sont correctement classifiées ou non, la répartition des résistances est très inégale : on trouve des valeurs des résistances qui discriminent de part et d'autre respectivement 90% (voire 95%) des images `V` et `F` dans le cas de `MNIST`, et tout juste 80% pour `FashionMNIST`.

On remarque également que l'attaque n'échoue pour aucune des images incorrectement classifiées de `MNIST`, et sur seulement 2 des 500 incorrectement classifiées de `FashionMNIST` étudiées. Inversement, l'attaque échoue sur respectivement 200 et 95 des 500 images correctement classifiées de `MINST` et `FashionMINST`. Par la suite, le succès de l'attaque ne sera important que sur les images incorrectement classifiées, ce qui valide donc le choix de la fonction $Loss_2$ en partie 1.

Une corrélation se dessine donc nettement entre la résistance et la justesse de la prédiction du réseau : une résistance élevée est souvent associée à une prédiction juste, et une résistance faible à une erreur de classification.

2.4. Une méthode de détection des exemples adversaires

On observe des résultats similaires dans le cas des attaques adversaires : les exemples adversaires sont plus "facile" à attaquer que les "vraies" images.

2.4.1 Génération d'exemples adversaires

La partie 1 présente une méthode efficace de génération d'exemple adversaire. On souhaite cependant se prémunir contre le plus grand nombre d'attaques possibles, et c'est pourquoi on confiera la génération d'exemples adversaires à la bibliothèque `CleverHans` [5].

⟩ *Expliquer ici rapidement les attaques utilisées.*

2.4.2 Identification des exemples adversaires

⟩ *Étudier de la répartition des résistances sur les images `V` (vraies images) et `A` (exemples adversaires).*

4.4.2 Des exemples adversaires qui trompent cette méthode ?

⟩ *À compléter.*

3. Une nouvelle expression de l'assurance d'un réseau

L'assurance du réseau sur sa prédiction, dénotée par la fonction $Conf_c$ où c est la catégorie prédite n'est souvent pas une grandeur pertinente : elle est la plupart du temps très proche de 0 ou de 1 (ce qui est dû à l'algorithme de descente de gradient lors de l'apprentissage), et les exemples adversaires montrent que cette valeur n'est pas toujours fiable : le réseau peut facilement se tromper tout en étant certain à 99% de sa prédiction.

Une bonne fonction d'assurance devrait permettre de savoir si le réseau est véritablement sûr de sa prédiction, ou bien s'il se trompe peut-être, ou encore si il est face à un exemple adversaire.

De plus, on a vu qu'une résistance haute correspond le plus souvent à une prédiction juste, et qu'une résistance faible correspond ou bien à une erreur de classification, ou bien à une attaque du réseau. En utilisant cela, on peut proposer une autre expression de cette assurance, dont la valeur aura un sens concret :

$$Conf2_c(img) = \frac{\frac{Res}{K}}{1 + \frac{Res}{K}}$$

où K est une valeur de Res qui sépare le mieux possible les images correctement et incorrectement classifiées, c'est à dire telle qu'à la fois le plus possible d'images correctement classifiées soient de résistance plus grande que K, et le plus possible d'images incorrectement classifiées de résistance plus petite que K (par exemple pour `FashionMNIST`, on pourra choisir $K = 0.28$, $K = 0.54$ et $K = 25$ pour respectivement Res_N, Res_{max} et Res_{min}, en lisant le tableau de valeurs de la partie 2.3).

De la sorte, une valeur de $Conf2$ au dessus de 0.5 correspondra le plus souvent à une prédiction correcte, et une valeur en dessous de 0.5 indiquera une probable erreur de prédiction.

4. Les corrections adversaires

Le deuxième phénomène observé est le suivant : l'attaque adversaire décrite partie 1, effectuée sur une image incorrectement classifiée par le réseau, produit presque toujours une image dont la catégorie sera celle de l'image initiale. On qualifiera ce phénomène de correction adversaire.

Ainsi, avec les réseaux précédents : sur les 62 erreurs commises sur la base test de `MNIST`, 53 sont rattrapées par les corrections adversaires ; et sur les 876 commises sur `FashionMNIST`, 613 sont rattrapées, soit respectivement 85% et 70%.

On a donc un premier résultat : à partir d'un réseau d'erreur *Top 1* donnée, on peut en déduire un système d'erreur *Top 2* sensiblement moindre : dans les exemples précédents, on passe d'erreurs *Top 1* de 0.53% et 8.7% à des erreurs *Top 2* de respectivement 0.09% et 2.6%.

Cette stratégie peut être intéressante dans une tâche de type `ImageNet`, où l'on s'intéresse à l'erreur *Top 5* commise par le classificateur. Utilisons l'algorithme suivant : sur les 5 meilleures prédictions du réseau, on ne conserve que les 3 meilleures, et on détermine deux autres catégories en réalisant des corrections adversaires a partir des deux premières.

Faute de moyens techniques, cet algorithme n'a pu être expérimenté (taille gigantesque de la base de données), mais son efficacité à améliorer les résultats du classificateur est conjecturée

5. Une méthode pour réduire l'erreur du réseau

Dans toute cette partie, on travaillera dans l'hypothèse de l'absence d'exemples adversaires dans les bases de données étudiées. Une résistance faible est alors presque toujours associée à une erreur de classification. On cherche alors à identifier le plus finement possible les erreurs de classification du réseau, pour les corriger ensuite avec la méthode de la correction adversaire.

Dans un milieu "hostile" (c'est à dire où les images en entrée du réseau ont pu être altérées de manière malveillante), un tel raccourci ne sera plus valable : il est possible de modifier une image de sorte à abaisser sa résistance, faisant ainsi croire au réseau qu'il se trompe.

5.1 Une première méthode...

La méthode "naïve" est la suivante : On détermine la résistance de chaque image du réseau. Si elle est supérieure à un certain seuil alors on considérera que la prédiction du réseau est correcte, sinon on choisit comme prédiction le résultat de la contre-attaque adversaire.

Sur un lot de 275 images de `test` de `FashionMNIST` (250 justes, 25 erreurs, proportion représentative de la base totale), avec respectivement $Res_{N=500}$, Res_{min} et Res_{max}, on obtient le nombre d'erreurs commises en fonction du seuil choisi, Figure 3.

Avec des seuils à 0, on retrouve naturellement 25 erreurs, puisque l'on n'a rien modifié aux prédictions du réseau.

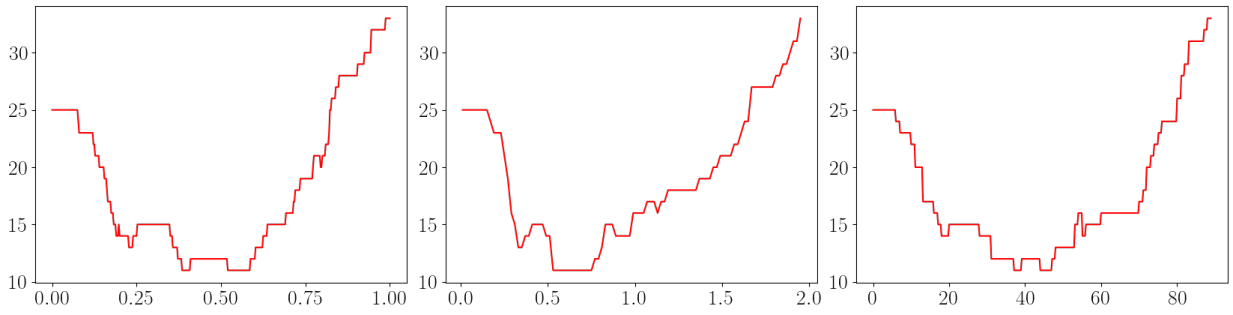

FIGURE 3 – Nombre d'erreurs en fonction du seuil choisi

En revanche, avec des seuil respectivement à 0.4, 0.685 et 45, le réseau ne commet plus que 11 erreurs.

5.2 . . . peu efficace à grande échelle.

En appliquant les méthodes précédentes à l'ensemble de 10000 images de `test` de `FashionMNIST`, on ne réussit qu'à faire passer le nombre d'erreurs de X à X dans le meilleur des cas. Le choix arbitraire d'un seuil fixé n'est donc pas une méthode efficace ici.

Ceci s'explique simplement : le nombre d'erreurs corrigées est trop faible devant le nombre de faux-positifs (images bien classées, mais considérées comme des erreurs par le seuil), annulant tout le gain obtenu.

5.3 Réseau discriminateur

Le choix arbitraire d'un seuil et la représentation de la résistance par une seule valeur ne sont donc pas des méthodes efficaces pour réduire l'erreur du réseau. Essayons alors d'affiner la distinction entre les images correctement ou incorrectement prédites. Pour cela, on cherche à entraîner un réseau de neurones, appelé *discriminateur*, à faire la distinction entre les images qui seront bien classifiées et celles qui seront mal classifiées.

5.3.1 Quelles données en entrée du réseau ?

Les données étudiées seront les valeurs de $\|r\|$ et de $Conf_c$ au cours de l'attaque des images. On ne conservera que 50 valeurs sur les 500 de chaque attaque, pour limiter la taille des bases de données obtenues.

En sortie du réseau, on aura un flottant compris entre 0 et 1, qui indiquera une erreur de prédiction au dessus de 0.5, et une prédiction, juste en dessous de 0.5.

On construit ces bases de données à partir des images de `train` et `val` uniquement, de sorte à observer la généralisation des résultats sur les images de `test`, sur lesquelles le but final est d'améliorer la prédiction du réseau classificateur.

On appelera `norms` et `confs` ces bases de données.

5.3.2 Structure et entraînement du réseau

On utilise un réseau de neurones très simple : 50 neurones sur la couche d'entrée, 30 neurones sur la couche intermédiaire, et un seul neurone en couche de sortie. On utilisera la fonction de transfert $ReLU$ pour les deux premières couches, et une fonction $Softmax$ pour le neurone de sortie.

On entraîne ce réseau en 40 étapes (*epochs*), avec des paquets de taille 32 (*mini-batches*). On utilise $Cross - entropy$ comme fonction d'erreur, et l'algorithme `Adam` pour réaliser la descente de gradient, avec un taux d'apprentissage $\eta = 5 \cdot 10^{-4}$.

5.3.3 Généralisation des résultats obtenus

Observons enfin les résultats obtenus par le réseau discriminateur sur les 10000 images de `test`, en fonction du problème étudié (`MNIST` ou `FashionMNIST`), et de la base de discriminateur utilisée (`norms` ou `confs`) :

Erreurs	Faux positifs	Faux négatifs	Total
`MNIST` (`norms`)	412	4	416 (4.2%)
`MNIST` (`confs`)	360	4	364 (3.6%)
`FashionMNIST` (`norms`)	2424	109	2533 (25%)
`FashionMNIST` (`confs`)	2071	130	2201 (22%)

On appelle *faux positifs* les images correctement classifiées, mais prédites comme des erreurs par le discriminateur ; et *faux négatifs* les images incorrectement classifiées mais prédites comme des images correctement classifiées par le doscriminateur.

Ces résultats sont décevants : on retrouve une separation d'une finesse du même ordre de celle partie 2.3, à la nuance près que cette séparation a cette fois-ci été faite sans connaître au préalable les images de `test`.

Pour que la correction ait un intérêt, il faut que la précision du réseau discriminateur soit plus fine que celle du réseau classificateur, ce qui n'est manifestement pas le cas ici : respectivement 364 contre 62 erreurs et 2201 contre 876 erreurs sur les 10000 images de `test` pour `MNIST` et `FashionMNIST`. On en déduit que les bases de données `norms` et `confs` ne contiennent pas assez d'informations pour réaliser finement cette discrimination entre images correctement classifiées, et incorrectement classifiées.

Conclusion

⟩ *À compléter.*

Bibliographie

[1] C. Szegedy, I. Goodfellow & al. CoRR, **Intriguing Properties of Neural Networks.** (Déc. 2013)

[2] A. Krizhevsky, I. Sutskever & G. Hinton. NIPS'12 Proceedings, **ImageNet Classification with Deep Convolutional Neural Networks .** Volume 1 (2012), Pages 1097-1105

[3] Y. Le Cun & C. Cortes. **The MNIST handwritten digit database.**

[4] H. Xiao, K. Rasul & R. Vollgraf. **Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.** *arXiv:1708.07747*

[5] N. Papernot, N. Carlini, I. Goodfellow & al. **CleverHans v2.0.0: an adversarial machine learning library.** *arXiv:1610.00768*

A. Structure et entraînement des réseaux classificateurs

Les deux réseaux ont la même architecture, inspirée de AlexNet :

- Entrée : 32×32
- Convolution : 32 couches, noyau 5×5
- ReLU
- MaxPool : noyau 2×2
- Convolution : 64 couches, noyau 3×3
- ReLU
- MaxPool : noyau 2×2
- Dropout
- Couche complète : $64 \times 5 \times 5 \rightarrow 120$
- ReLU
- Dropout
- Couche complète : $120 \rightarrow 10$
- Softmax

B. Quelques attaques supplémentaires

En Figure 4, les valeurs prises par $\|r\|$ et $Conf_c$ au cours de l'attaque de 6 images "difficiles" à attaquer.

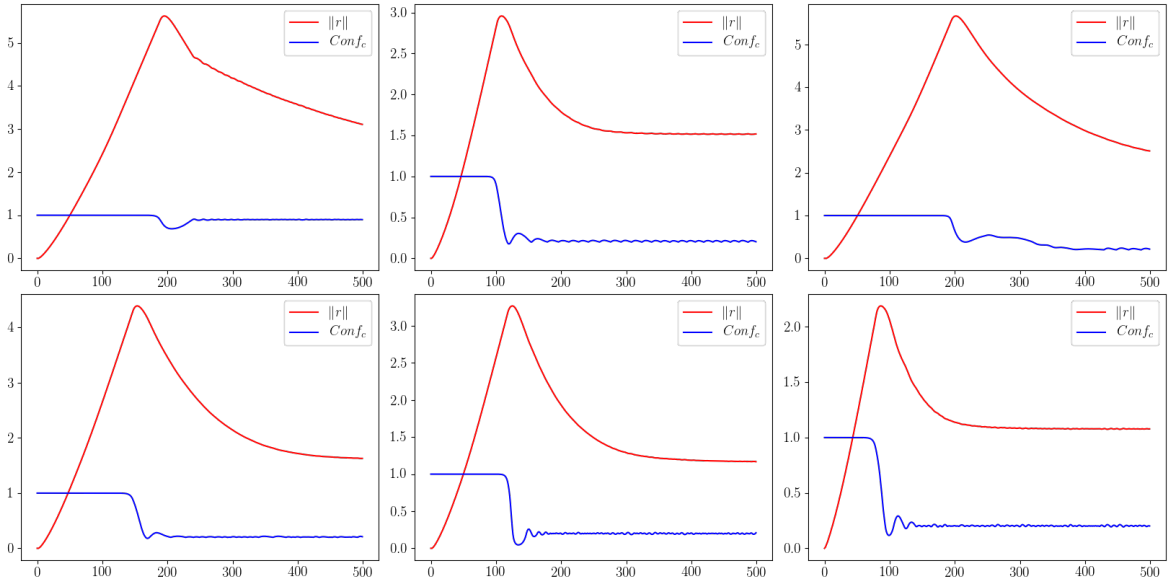

FIGURE 4 – Attaques adversaires "difficiles"

En Figure 5, même chose avec 6 images "faciles" à attaquer.

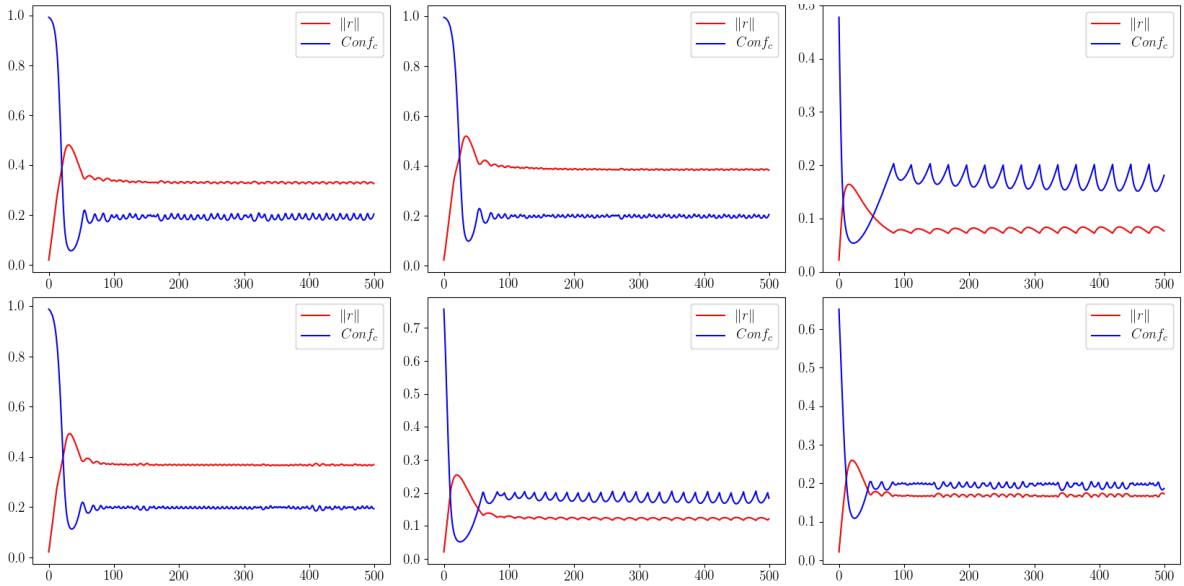

FIGURE 5 – Attaques adversaires "faciles"