Robustesse des Réseaux de neurones par apprentissage antagoniste

Matthieu Poyer

10 septembre 2020

Table des matières

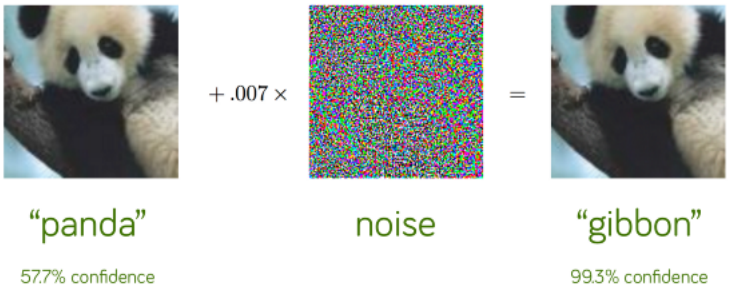
Remerciements

Je tiens à remercier personnellement et chaleureusement le CEA qui m'a accueilli pendant 4 mois, Jean-Marc Martinez mon maître de stage qui m'a consacré beaucoup de temps et qui m'a accompagné dans toutes mes idées aussi saugrenues soient-elles. Sakaria Diarrassouba un autre stagiaire travaillant sur un sujet proche du mien avec qui j'ai beaucoup échangé.

Chapitre 0

Introduction au problème

Depuis quelques années les réseaux de neurones sont utilisés pour résoudre une grande variété de problèmes. Ils sont notamment derrière AlphaGo, le célèbre programme informatique qui bat les joueurs professionnels de Go, les algorithmes de reconnaissance d'images ou même certains algorithmes de lutte contre la fraude fiscale. Pourtant malgré de magnifiques victoires (OpenAI qui bat les meilleurs joueur de Dota 2, Tesla qui peut passer en auto-pilote), sont-ils infaillibles ? La réponses est, vous vous en doutez : non ! L'image ci-dessous, que l'on retrouve partout, le montre. En effet, il est possible de modifier imperceptiblement une image, pour que l'algorithme de reconnaissance se trompe complètement (ie : ne reconnaisse pas la même chose qu'un humain).



Il est donc possible et l'on verra qu'il est même facile de tromper les algorithmes de reconnaissance d'images. Ainsi peut-être qu'en modifiant la valeur de certains pixels du capteur de la caméra embarquée d'une voiture autonome on pourrait empêcher une voiture autonome de reconnaître un feu rouge. Ceci montre l'importance de rendre ce genre d'algorithmes moins sensibles à de petites perturbations. Ainsi la problématique de ce stage est :

Comment peut-on rendre les réseaux de neurones robustes, et donc peu sensibles à de petites perturbations ? Pour y parvenir il faut se renseigner sur le fonctionnement des réseaux de neurones, apprendre à créer des adversaires, regarder les méthodes de défense qui existent, et en penser de nouvelles.

Chapitre 1

Les Réseaux de Neurones

Nous allons dans cette partie présenter rapidement les réseaux de neurones.

1.1 Qu'est ce qu'un réseau de neurones

Un neurone formel, utilisé dans les réseaux de neurones informatiques, est composé :

- d'entrées $(x_1, \ldots, x_m)$,
- d'une fonction d'activation ϕ,
- de paramètres $(w_1, \ldots, w_m)$ de poids et b de biais,
- et d'une sortie réelle.

Ce que renvoie le neurone est alors $\phi(b + \sum_k w_k x_k)$. La fonction d'activation est donnée par le statisticien au moment de la création du réseau et les entrées sont données directement ou indirectement par la base de données sur laquelle on s'entraîne. Les réseaux de neurones sont alors des couches de neurones que l'on connecte entre elles.

Grâce à la figure 1.1, on remarque que tous les neurones d'une même couche ont les mêmes entrées et en général on utilise la même fonction d'activation ϕ. Soit une couche à n neurones du réseau, en posant $x = (x_1, \ldots, x_m)^\top$ le vecteur des entrées de cette couche $W = (w_{i,j})_{i,j}$ la matrice qui associe à chaque ligne les poids d'un neurone composant cette couche, et $b = (b_1, \ldots, b_n)$, on peut alors modéliser cette couche par :

$$\begin{cases} M_{m,1}(\mathbb{R}) \times M_{n,m}(\mathbb{R}) \times M_{n,1}(\mathbb{R}) & \longrightarrow M_{n,1}(\mathbb{R}) \\ x, W, b & \longmapsto (\phi(W_i x + b_i))_i, \end{cases}$$

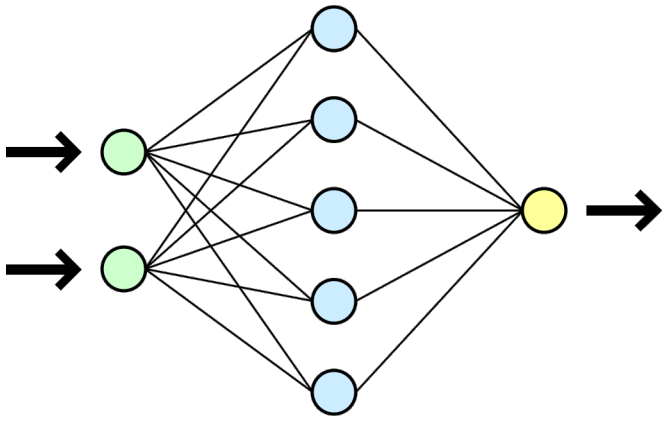


FIGURE 1.1 – *Cette figure illustre un réseau possible avec deux entrées, une sortie et une couche cachée (ce sont les couches qui ne correspondent ni à l'entrée ni à la sortie). Les cercles correspondent ainsi aux neurones.*

avec W_i la ième ligne de la matrice W. Cette fonction sera notée par abus d'écriture : $\phi(x, W, b)$ ou $\phi(Wx + b)$. Un réseau de neurones n'est alors rien de plus qu'une simple composition de fonctions. Ainsi un réseau de neurone f_θ à K couches et de paramètres $\theta = (W, b)$ peut s'écrire comme ceci :

$$f_\theta(x) = \phi_K(\phi_{K-1}(\dots \phi_2(\phi_1(x, W_1, b_1), W_2, b_2) \dots, W_K, b_K),$$

où ϕ_k est la fonction d'activation de la couche k, W_k et b_k sont respectivement la matrice de poids et le biais associés.

1.2 Une méthode d'optimisation

Dans la partie précédente nous nous sommes intéressés à la création des réseaux de neurones, nous allons ici comprendre comment ils sont utilisés pour résoudre des problèmes d'optimisation.

Un réseau de neurones n'est rien d'autre qu'une fonction avec beaucoup de paramètres à déterminer. Dans le mode supervisé l'enjeu consiste alors à ce que le réseau de neurones corresponde à la fonction f_θ qui minimise la perte $\mathcal{L}(f_\theta(x), y)$. Pour cela on va chercher un paramètre θ tel que pour x, y des données de la base d'apprentissage (y étant l'étiquette associée à x) on ait :

$$\theta \in \arg\min_{x,y} \mathcal{L}(f_\theta(x), y).$$

On rappelle que $\theta = (W, b)$, il s'agit donc de trouver les meilleurs W et b qui minimisent la fonction perte $\mathcal{L}$. Une simple descente en gradient suffit :

$$\begin{aligned} W &= W - \mu \nabla_W \mathcal{L}(f_\theta(x), y) \\ b &= b - \mu \nabla_b \mathcal{L}(f_\theta(x), y). \end{aligned}$$

Nous avons vu précédemment que la fonction f_θ est une composition de fonctions d'activation. Il va alors falloir jouer avec la formule de la dérivation d'une composition. Ainsi, si on pose :

$$\begin{aligned} z^l &= W_l a^{l-1} + b^l \\ a^l &= \phi_l(W_l a^{l-1} + b^l) = \phi_l(z^l), \end{aligned}$$

alors on trouve :

$$\begin{aligned} W_K &= W_K - \mu \phi'_K(z^K) \nabla_x \mathcal{L} \ (a^{K-1})^\top \\ W_{K-1} &= W_{K-1} - \mu \phi'_{K-1}(z^{K-1}) W_K^\top \phi'_K(z^K) \nabla_x \mathcal{L} \ (a^{K-2})^\top \\ &\dots \\ W_1 &= W_1 - \mu \phi'_1(z^1) W_2^\top \phi'_2(z^2) \dots \phi'_{K-1}(z^{K-1}) W_K^\top \phi'_K(z^K) \nabla_x \mathcal{L} \ (a^0)^\top, \end{aligned}$$

où $\phi'_i(x)$ est la matrice diagonale avec $(\phi'_i(x_j))_j$ comme coefficients diagonaux. Pour aller plus vite il y a une petite astuce qu'on appelle la *rétropropagation* du gradient (ou *backpropagation* en anglais), qui consiste à modifier les poids dans l'ordre précédent. En effet si on note :

$$\delta^i = \phi'_i(z^i) W_{i+1}^\top \phi'_{i+1}(z^{i+1}) \dots \phi'_{K-1}(z^{K-1}) W_K^\top \phi'_K(z^K) \nabla_x \mathcal{L}$$

on remarque alors que :

$$\delta^{i-1} = \phi'_{i-1}(z^{i-1}) W_i^\top \delta^i$$

et

$$W_i = W_i - \mu \delta^i (a^{i-1})^\top.$$

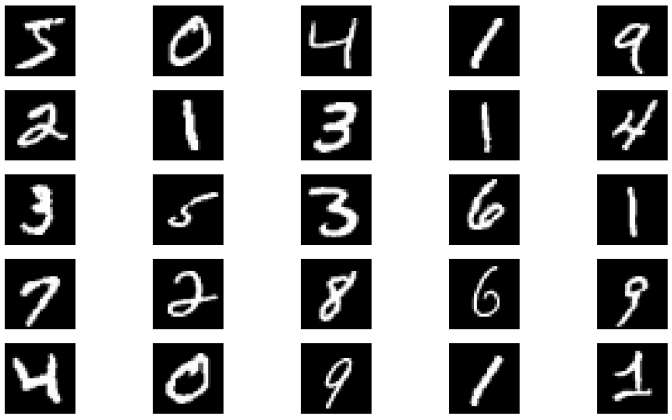
Cette astuce nous permet de ne pas refaire plusieurs fois les mêmes calculs (ceux des δ_i), contrairement à ce qui se serait passé si on avait traité le problème dans l'ordre classique (ie : W_1, puis W_2, etc.) ; d'où le nom de rétropropagation. Il ne reste plus qu'à trouver b. En utilisant le même raisonnement on trouve :

$$b_i = b_i - \mu \delta^i.$$

Cette partie montre qu'une descente en gradient sur nos paramètres permet à notre réseau de résoudre un problème d'optimisation.

1.3 Un réseau de classification

Nous avons vu comment cela fonctionne en théorie, regardons ce que cela donne en pratique. Nous nous sommes intéressés au problème très classique de la classification des chiffres de la base de données MNIST. Le problème est très facile : on a une base de données de 60 000 images d'entraînement et 10 000 images de tests, chacune représente un chiffre (cf figure ci-dessous). Le but est donc de créer un classifieur qui prend en entrée une image de $28 \times 28 = 784$ pixels et de renvoyer son *étiquette*, sa *classe*, aussi parfois appelée *label* (ie : la valeur du chiffre affichée).



Les paramètres du réseau que nous avons utilisés pour créer notre classifieur sont en fait ceux du premier article que nous avons lu et qui sera présenté au chapitre 2 ([7]). Le réseau a une structure *full connected*, c'est quand tous les neurones d'une couche sont connectés à tous les neurones de la couche précédente ; il a 2 couches cachées de 100 neurones chacune dont les fonctions d'activation sont des sigmoïdes. (La fonction sigmoïde est la fonction $x \mapsto \frac{1}{1+e^{-x}}$). La fonction d'activation utilisée pour la couche de sortie est la fonction softmax et la fonction perte que l'on cherche à minimiser est la perte d'entropie croisée : $\mathcal{L} : (x, y) \mapsto -\sum_i y_i \log(x_i)$.

Intéressons nous un instant à la couche softmax. La fonction softmax est définie par $f_j : x \mapsto \frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}}$. Cette fonction peut être interprétée comme donnant la probabilité d'être dans telle ou telle classe. Ainsi, si notre réseau renvoie un vecteur de la forme : [0, 0.9, 0, 0, 0.1, 0, 0, 0, 0, 0], on peut l'interpréter comme il y a 90% de chances que l'image soit un 1 et 10%

de chances que ce soit un 4. Cette idée repose sur quelques arguments : les composantes sont positives, la somme des composantes vaut 1 et on sélectionne la classe qui a la plus grande valeur, soit la "plus probable". Il est clair que cet estimateur de la confiance d'être tel ou tel chiffre n'est qu'une interprétation, en effet, quand on compare les résultats réels à la confiance, on remarque que les résultats diffèrent (confiance estimée : 99.2%, précision réelle : 97.7%). Mais par abus d'usage on confond la confiance estimée avec la confiance réelle.

Remarque. Il existe d'autres architectures de réseaux de neurones, notamment le CNN mais nous ne nous y attarderons pas dans ce rapport.

Les résultats obtenus sont remarquables : en une vingtaine d'étapes, on atteint des taux de reconnaissance de la base de test de 97.7%. C'est à travers ce réseau que nous avons envisagé, étudié et testé les méthodes de robustesse.

Chapitre 2

Les exemples adverses et la robustesse (état de l'art)

Rappels : Nous nous intéressons à la classification de la base MNIST et nous cherchons à trouver un classifieur "robuste" $f_\theta : [0,1]^{784} \to [0,1]^{10}$. Pour trouver le meilleur classifieur, il faut alors trouver le meilleur paramètre θ, celui qui minimise la perte d'entropie croisée $\mathcal{L}$. Pour un échantillon de N données on estime θ grâce à :

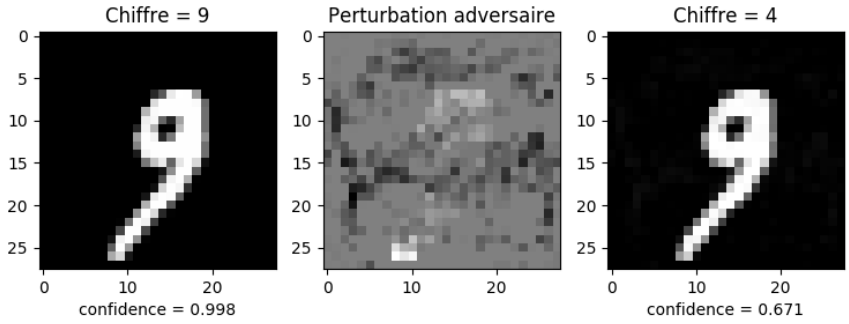
$$\hat{\theta} \in \arg\min \left\{ \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i)) \right\}.$$

On rappelle que la perte d'*entropie croisée* (ou *cross-entropy* en anglais) est définie par :

$$\mathcal{L}(x, y) = -\sum_i y_i \log(x_i).$$

2.1 Création d'un adversaire

Un *adversaire* ou *image adverse* est une image artificielle x'_0 dérivée d'une image réelle x_0, qui trompe le réseau. C'est-à-dire qu'un humain ne verrait que peu ou pas de différences entre x_0 et x'_0 et donc leur associerait la même classe tandis que le classifieur renverrait deux classes différentes (et ce avec des confiances parfois très élevées).



Pour créer un adversaire il n'y a rien de plus simple, il suffit d'avoir un classifieur $f_{\hat{\theta}}$ et une image à perturber x_0 (de label y_0). Il suffit ensuite de chercher le x qui maximise la fonction :

$$x \mapsto -y_{0,c_0} \log(f_{\hat{\theta},c_0}(x)),$$

ou de manière équivalente celui qui minimise :

$$\mathcal{L}_{\text{adv}} : x \mapsto -y_{0,c_0} \log(1 - f_{\hat{\theta},c_0}(x)).$$

Une simple descente en gradient suffit en général. L'article qui nous a servi de base [7], quant à lui minimisait : $x \mapsto \mathcal{L}_{\text{adv}}(x) + c\|x - x_0\|_2^2$, où c est un paramètre permettant de régler l'importance que l'on veut donner au terme de régularisation. (Plus c est petit, plus on augmente les chances de trouver un adversaire, plus c est grand plus on impose une proximité avec l'image de départ, quitte à pouvoir créer moins d'adversaire ou à avoir des adversaires moins efficaces.)

Remarque. Nous ne sommes pas obligés de choisir la norme $\|.\|_2$ comme le font Szegedy et al. [7], d'ailleurs beaucoup d'articles utilisent la norme infinie comme distance entre l'adversaire et l'image originale [4], [3], [1].

La création d'adversaires est très facile et beaucoup d'articles montrent que certains adversaires sont réutilisables dans des réseaux parfois très différent [7], mais comme ce n'est pas le sujet du stage nous ne les avons pas étudiés. En revanche cela semble montrer que tromper un réseau de neurones est vraiment simple.

2.2 La Robustesse

Définition 2.1 (ϵ-adversaire pour la norme p et l'image x_0)**.**
On appelle ϵ-adversaire pour la norme p et l'image x_0 (de label y_0) tout

élément x' vérifiant :

$$\begin{cases} \|x' - x_0\|_p < \epsilon \\ \arg\max(f(x')) \neq \arg\max(y_0) = c_0. \end{cases}$$

Remarque.

- En général on ne parlera que d'ϵ-adversaires, soit on sera dans le cas général, soit on saura de quelle norme on parle.
- On utilise l'arg max, car finalement le chiffre reconnu par le classifieur f est bien le chiffre qui a la plus grande probabilité. (Si une image x a 90% de chances d'être un 8 *a priori* c'est un 8).

Définition 2.2 (ϵ-robustesse autour d'un point)**.** On dit qu'un classifieur f est *ϵ-robuste autour de* x_0 (de label y_0) pour la norme $\|.\|$ si :

$$\forall \|r\| < \epsilon, \quad \arg\max(f(x_0 + r)) = \arg\max(y_0) = c_0.$$

Définition 2.3 (ϵ, α-robustesse autour d'un point)**.** On dit qu'un classifieur f est *ϵ, α-robuste autour de* x_0 (de label y_0) pour la norme $\|.\|$ si :

$$\max_{\|r\|<\epsilon} |f_{c_0}(x_0) - f_{c_0}(x_0 + r)| \leq \alpha.$$

Remarque. On remarque que dans notre cas la robustesse est une notion locale, il faut donc l'étendre. Je vois deux méthodes pour le faire :

- *version forte :* une fonction est robuste si elle l'est autour de tous les points x_0 (de sa base de données initiales),
- *version faible :* ou alors en regardant l'espérance.

L'idéal serait évidemment d'avoir des résultats sur la version forte, mais beaucoup d'articles ne s'intéressent qu'à la version faible [4], [1].

2.3 Quelques petits résultats

2.3.1 Une première approche : l'idée

Il semble que l'ϵ, α-robustesse est plus facilement étudiable. Le raisonnement naturel consiste donc à appliquer une inégalité des accroissements finis sur $f_{\hat{\theta}}$. Vérifions les hypothèses, $\mathbb{R}^{784}$ est un Banach pour la norme p avec $1 \leq p \leq +\infty$, soit $U = \mathcal{B}_{\mathbb{R}^{784}}(x_0, \epsilon)$, $f_{\hat{\theta}}$ peut être étendue à U (mais n'a de sens que sur $[0,1]^{784}$) et on la restreint à la coordonnée non nulle c_0 du label y_0 ($c_0 = \arg\max(y_0)$). Cette nouvelle fonction est notée $f_{\hat{\theta},c_0}$, supposons cette fonction différentiable, on aurait alors :

$$\forall r \in U, \quad |f_{\hat{\theta},c_0}(x_0+r) - f_{\hat{\theta},c_0}(x_0)| \leq \left(\sup_{t\in[x_0,x_0+r]} \|\nabla f_{\hat{\theta},c_0}(t)\|_q \right) \|r\|_p.$$

En particulier, on a :

$$\sup_{r\in U\cap\mathbb{R}^{784}} |f_{\hat{\theta},c_0}(x_0+r) - f_{\hat{\theta},c_0}(x_0)| \leq \left(\sup_{t\in U\cap\mathbb{R}^{784}} \|\nabla f_{\hat{\theta},c_0}(t)\|_q \right) \epsilon.$$

Remarque. L'hypothèse de différentiabilité de la fonction $f_{\hat{\theta},c_0}$ n'est pas très contraignante, en effet toutes les fonctions d'activation classiques sont différentiables, la fonction ReLU exceptée (plus de détails sont données dans la partie suivante). Même dans le cas de la fonction ReLU, une inégalité des accroissements finies existe :

$$\forall x \in \mathbb{R},\ \forall r > 0, \quad |\mathrm{ReLU}(x+r) - \mathrm{ReLU}(x)| \leq r = \left| \max_{\mathbb{R}^*} \mathrm{ReLU}'(x) \right| r.$$

Remarque. On ne se limite qu'à étudier une coordonnée pour ne s'intéresser qu'à celle qui nous intéresse vraiment. Je m'explique : si on veut qu'une image soit perçue comme un 8, il suffit de regarder la coordonnée associée au 8 du classifieur et il ne semble alors pas nécessaire de regarder les autres.

Rappel d'un petit résultat : On rappelle que le dual de L^p s'identifie à L^q pour $1 \leq p \leq +\infty$ et $\frac{1}{p} + \frac{1}{q} = 1$, c'est ce qui justifie le q dans les formules précédentes. La preuve est en annexe.

Il s'agit maintenant de majorer $\|\nabla f_{\hat{\theta},c_0}(t)\|_q$ pour un t quelconque, ce qui donnerait un résultat d'ϵ, α-robustesse forte.

2.3.2 Une première approche : complément

Nous allons dans cette partie déterminer un majorant global à $\|\nabla f_{\hat{\theta},c_0}(t)\|_q$. Étant donné qu'il nous faut des informations sur les fonctions d'activation on va traiter les fonctions d'activation les plus connues.

On peut modéliser un réseau de neurone f_θ à K couches et de paramètres $\theta = (W, b)$ comme ceci :

$$f_\theta(x) = \phi_K(\phi_{K-1}(\dots \phi_2(\phi_1(x, W_1, b_1), W_2, b_2) \dots, W_K, b_K),$$

où ϕ_k est la fonction d'activation de la couche k. Étant donné que l'on ne s'intéresse qu'à la coordonnée c_0, on peut aussi écrire (quitte à changer W_K et b_K) :

$$f_{\theta,c_0}(x) = \phi_K(\phi_{K-1}(\dots \phi_2(\phi_1(x, W_1, b_1), W_2, b_2) \dots, W_K, b_K).$$

On rappelle la formule de la jacobienne d'une composition de deux fonctions :

$$J(f \circ g) = (J(f) \circ g) \times J(g),$$

on en déduit que pour tout $k \in \{1, \dots, K\}$ (sous réserve d'existence de ϕ_k') :

$$J(\phi_k(W_k x + b_k)) = W_k^\top \phi_k'(W_k x + b_k).$$

Si on reprend tout ce qu'on a et qu'on le met bout à bout on peut peut-être trouver un majorant de $\|\nabla f_{\hat{\theta},c_0}(t)\|_q$. Commençons par majorer ϕ_k' (sous réserve d'existence) pour tout $k \in \{1, \dots, K\}$ par un L_k. On obtient alors que :

$$\begin{aligned}\|\nabla f_{\hat{\theta},c_0}(t)\|_q &\leq \left\| \prod_{i=1}^{K} L_i W_i^\top \right\|_q \\ &= \prod_{i=1}^{K} L_i \left\| \prod_{i=1}^{K} W_i^\top \right\|_q .\end{aligned}$$

Le cas sigmoïde Rappelons que la sigmoïde est définie par : $\phi : x \mapsto \frac{1}{1+e^{-x}}$. La fonction ϕ est dérivable sur $\mathbb{R}$ de dérivée :

$$\forall x \in \mathbb{R}, \quad \phi'(x) = \frac{e^{-x}}{(1+e^{-x})^2}.$$

Or on a que pour tout $x \in \mathbb{R}$, on a $\frac{x}{(1+x)^2} \leq \frac{1}{4}$ (obtenu pour x = 1). Donc par identification on a que $\forall x \in \mathbb{R},\ \phi'(x) \leq \frac{1}{4} = \phi'(0) = L$. La constante L associée à une fonction d'activation sigmoïdale est donc $\frac{1}{4}$.

Le cas softplus Par définition la fonction softplus est $\phi : x \mapsto \log(1+e^x)$. La fonction ϕ est dérivable pour tout $x \in \mathbb{R}$, et sa dérivée vaut :

$$\begin{aligned}\forall x \in \mathbb{R}, \quad \phi'(x) &= \frac{e^x}{1+e^x} \\ &= \frac{1}{1+e^{-x}}.\end{aligned}$$

On reconnaît la fonction sigmoïde qui a une image incluse dans $[0, 1]$, donc la constante L associée à la fonction ϕ est 1.

Le cas softsign La définition de la fonction softsign est donnée par : $\phi : x \mapsto \frac{x}{1+|x|}$. La fonction ϕ est dérivable sur $\mathbb{R}^*$.

$$\forall x \in \mathbb{R}^*, \quad \phi'(x) = \frac{1}{(1+|x|)^2}.$$

La fonction ϕ' est prolongeable par continuité en 0 est elle vaut 1. Or $\forall x \in \mathbb{R}^*,\ \phi'(x) \leq 1$, donc la constante L associée à la fonction ϕ est 1.

Le cas tangente hyperbolique Rappelons que la définition de la fonction tangente hyperbolique est : $\phi : x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Sa dérivée vaut : $1 - tanh^2$, donc la constante de L associée à $tanh$ est 1.

Le cas ReLU On dit que la fonction ϕ est une fonction ReLU si elle vérifie $\phi(x) = \max(0, x)$. La fonction ϕ est dérivable sur $\mathbb{R}^*$, et sa dérivée est majorée par 1. Le seul problème c'est qu'on n'a pas d'information en 0.

Nous pouvons nous dire qu'après tout la probabilité qu'atteindre pile 0 a une probabilité nulle d'arriver et par conséquent on peut garder $L = 1$ dans le cas ReLU. C'est d'autant plus vraie que par convention la dérivée de la fonction ReLU vaut 1 en 0. On peut alors voir la fonction ReLU non plus grâce à sa forme classique, mais plutôt comme la primitive de $\mathbb{1}_{R_+}$ qui s'annule en 0.

Le cas softmax La fonction softmax est définie par $f_j : x \mapsto \frac{e^{x_j}}{\sum_{k=1}^n e^{x_k}}$, donc le gradient de f_j est bien défini pour $x \in \mathbb{R}^n$ et vaut :

$$\nabla f_j(x) = (\nabla f_{j,i}(x))_i,$$

avec

$$\nabla f_{j,i}(x) = \begin{cases} \dfrac{-e^{x_j} e^{x_i}}{(\sum_{k=1}^n e^{x_k})^2} & \text{si } i \neq j \\ \dfrac{e^{x_i}}{\sum_{k=1}^n e^{x_k}} \left(1 - \dfrac{e^{x_i}}{\sum_{k=1}^n e^{x_k}}\right) & \text{sinon} \end{cases}.$$

Contrairement aux autres cas, on n'a pas une fonction ϕ à variable réelle (ϕ' n'existe pas), mais c'est un faux problème, il suffit de trouver la norme infinie de $\nabla f_j(x)$. Pour $i = j$ on peut se ramener à la fonction $x \mapsto \frac{xc}{(x+c)^2}$ avec c une constante strictement positive et $x > 0$. Grâce à une étude de fonction on trouve que $\|\nabla f_{j,j}\|_\infty \leq \frac{1}{4}$. Soit $i \neq j$, on pose $z = e^{x_j} e^{x_i}$, et

$c > 0$ une constante.

$$\begin{aligned}\left|\frac{-e^{x_j}e^{x_i}}{(\sum_{k=1}^{n} e^{x_k})^2}\right| &= \frac{z}{c + 2z + \sum_{k\neq j} e^{x_k}e^{x_j} + \sum_{k\neq i} e^{x_k}e^{x_i}} \\ &\leq \frac{z}{c+2z} \\ &\leq \frac{1}{2}.\end{aligned}$$

Donc le cas softmax c'est comme si on avait une constante $L = \frac{1}{2}$.

Remarque. On remarque que l'on a déjà des premiers résultats de robustesse (une ϵ, α-robustesse forte) qui dépendent des paramètres du réseau de neurones. Hélas il n'est pas facile de contrôler W, en revanche si on a un réseau déjà entraîné on peut récupérer les poids W et ainsi calculer cette borne.

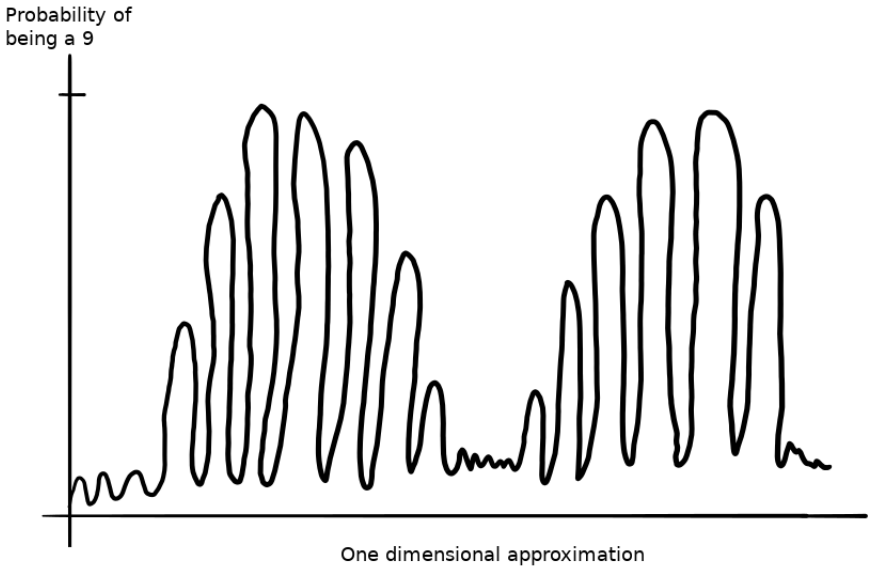
Chapitre 3

Nos stratégies de défense

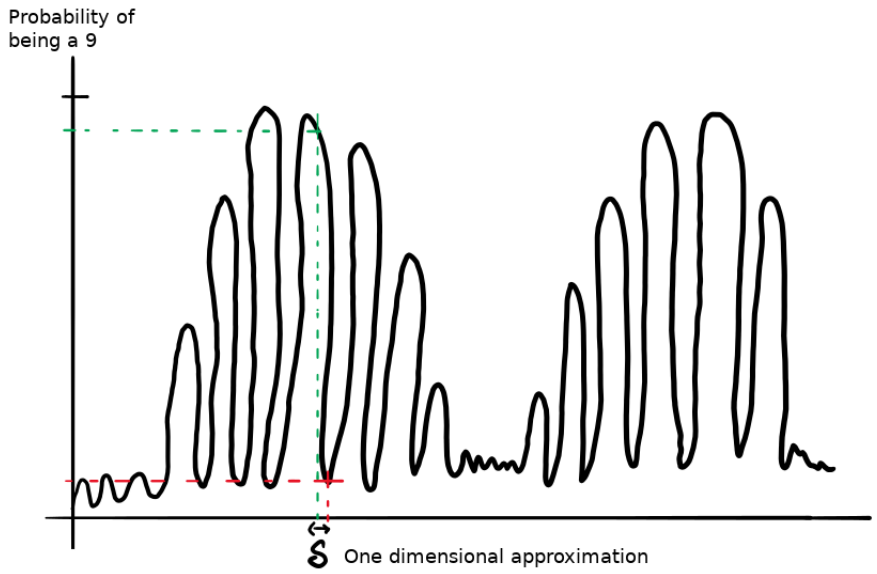
Je souhaiterais présenter ici l'intuition qui m'a accompagné tout au long de ce stage et ainsi mieux percevoir l'impact de chacune des méthodes. L'intuition se fait à partir de deux remarques que nous avons mises en évidence dans le chapitre précédent :

1. notre classifieur est plutôt bon (peu d'erreur et taux de confiance élevé) (chapitre 1),
2. il est très facile de trouver un adversaire proche de l'image originale (chapitre 2).

Nous nous attendons donc à trouver une fonction qui a à peu près cette forme.

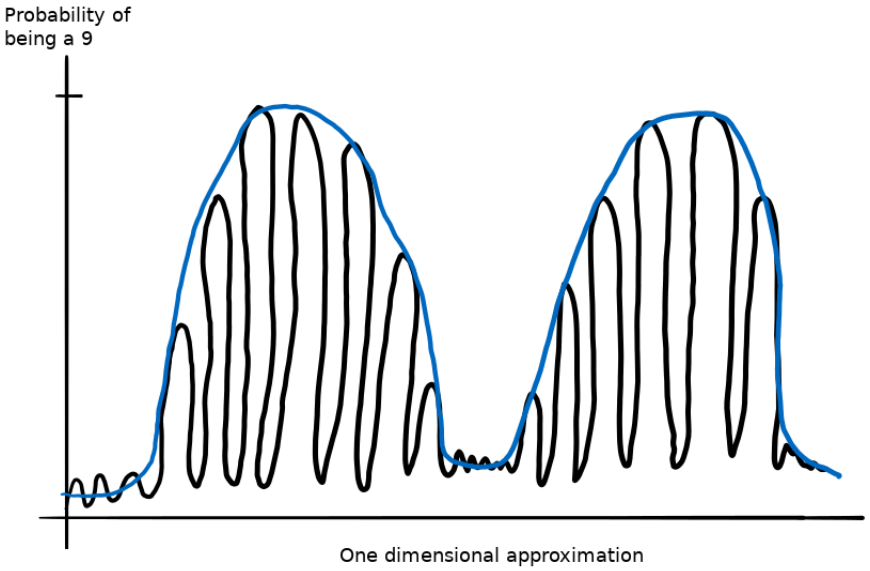


Avant d'aller plus loin, arrêtons nous sur cette image. On a en ordonnée la probabilité d'appartenir à une classe (ici 9, choisie arbitrairement), et en abscisse des images. Notre classifieur est bon, ce qui justifie les grands pics ; il est facile de créer des adversaires, ce qui explique la finesse de ces pics (comme il n'y a pas de raisons pour que le classifieur n'ait qu'un seul mode, il y en a deux qui sont dessinés). On a à peu près compris la forme de la fonction tracée, mais pas vraiment ce qu'était l'abscisse, en effet dans le cas de la base MNIST la dimension de l'antécédent est 784, or ici on n'a qu'une dimension. En réalité je pense que c'est un dessin qui marche dans à peu près toutes les dimensions et idéalement il faudrait réussir à se représenter des pics en dimension 784, ce qui pour ma petite tête est bien difficile, donc je les imagine en une dimension et j'obtiens ce dessin.



Sur l'image précédente on a représenté en vert une image réelle (elle est bien classifiée), et en rouge un adversaire de cette image réelle. Les images semblent proches (δ petit) et on constate un grand écart dans la classification.

A priori on souhaiterait donc obtenir la courbe bleue de l'image ci-après. Le but de cette partie est donc de présenter quelques méthodes qui ont pour but de modifier notre classifieur pour qu'il ressmble le plus possible à cette courbe bleue.



3.1 Les stratégies de régularisation (régulariseur l_2, l_1, dropout, W)

Cette partie part avant tout d'une idée assez simple : si les réseaux de neurones peuvent facilement se faire tromper c'est sûrement à cause d'un sur-apprentissage. La stratégie naturelle dans ces cas là est d'ajouter des termes de régularisation. Il y a alors plusieurs stratégies :

1 - ajouter des régulariseurs au sein de chacune des couches du réseau de neurones,

2 - utiliser du dropout

Si on ajoute un régulariseur à chaque couche du réseau de neurones on s'attend à avoir un phénomène de régularisation et donc à voir une augmentation de la robustesse. Cette intuition est assez naturelle car, c'est une stratégie classique de statisticien. Cette stratégie permet d'améliorer un petit peu la robustesse de notre classifieur, mais elle a un inconvénient majeur : ce qu'elle gagne en robustesse elle le perd en précision, c'est ce que montre la Figure 3.1.

Le dropout est une stratégie qui, à chaque étape, sélectionne une proportion de neurones que l'on va éviter. Ainsi lors d'une étape d'apprentissage, tous les poids ne sont pas mis à jour (car certains neurones ont été omis). Cela crée un phénomène de régularisation, car étant donné que le réseau ne sait jamais quels seront les neurones évités, il va donc éviter de donner

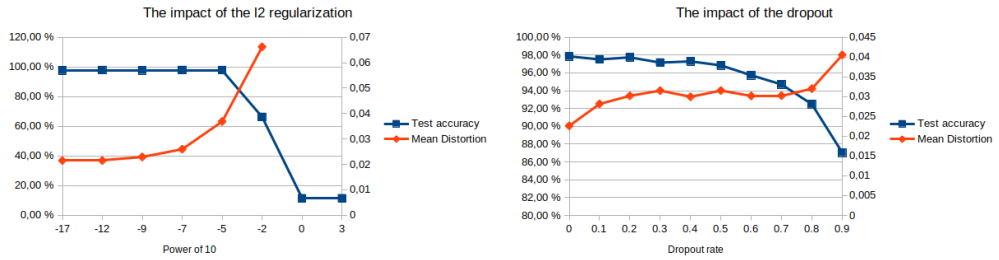


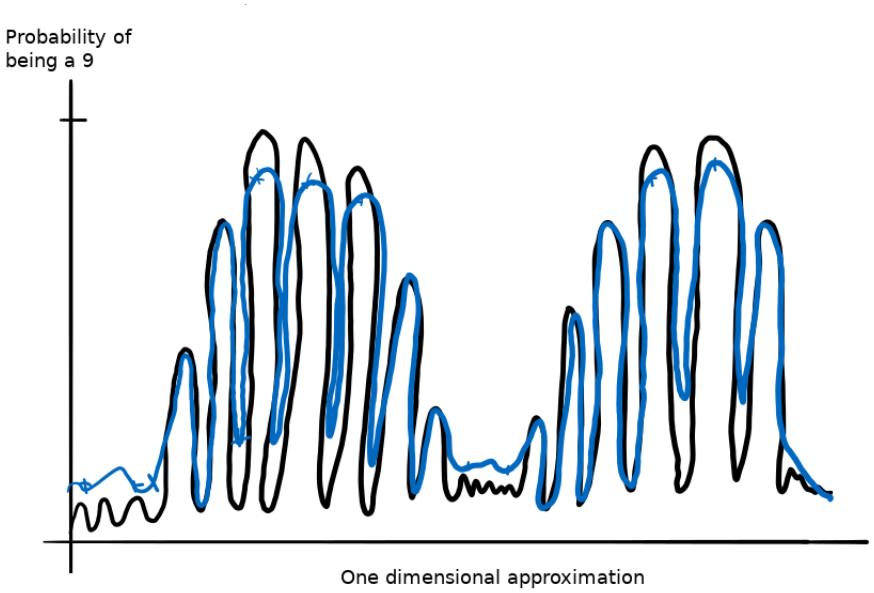
FIGURE 3.1 – *Le premier graphique montre l'impact de la régularisation l_2 que l'on a ajoutée à chacune des couches du réseau de neurones. Le second, quant à lui, s'intéresse à l'impact du dropout. On retrouve ce que l'on s'attendait à voir, c'est-à-dire plus la régularisation/dropout est important, plus notre classifieur est robuste. Le problème c'est que ce qui est gagné en robustesse est perdu en précision.*

trop d'importance à certains neurones (qui risqueraient d'être gelés). L'information va donc être mieux répartit sur les poids ce qui va avoir un effet régularisant.

Au regard du chapitre précédent, on pourrait aussi se demander s'il ne suffit pas de mettre des conditions sur les paramètres W du réseau de neurones.

3.2 La défensive distillation

Nous avons vu que la sortie d'un réseau de neurones peut être interprétée comme une probabilité de confiance (chapitre 1). Mais on peut se demander si on ne peut pas nuancer ces probabilités. En effet, il arrive que nous ne soyons pas sûrs de bien reconnaître les chiffres, car ils sont mal écrits ou utilisent d'autres conventions d'écritures (les anglais font leur 7 bizarrement, on les confond avec les 1). Ainsi Papernot et al. [5] proposent dans leur article de revoir cette fonction softmax qui nous donne nos "probabilités" *a posteriori* d'appartenir à une certaine classe afin de faire apparaître cette incertitude. Ainsi un 7 qui ressemble à un 1 pourrait avoir une confiance de 70% d'être un 7 et 30% d'être un 1. Si on revient à notre courbe d'intuition, cela revient à descendre moins bas et monter moins haut.



La méthode utilisée pour créer cette nuance consiste à ajouter un paramètre appelé température à la fonction softmax. Elle était initialement définie par $f_j : x \mapsto \frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}}$, elle est transformée en $f_{j,T} : x \mapsto \frac{e^{x_j/T}}{\sum_{k=1}^{n} e^{x_k/T}}$ (la fonction softmax originale est celle correspondant a $T = 1$). Regardons dans un premier temps les effets de la température.

La température n'a pas d'impact sur les résultats du classifieur. Le classifieur sélectionne la classe la plus probable. Ainsi dans le cas softmax classique, cela correspond à la classe j qui a le x_j le plus grand (car la fonction exponentielle est croissante). Dans le cas softmax modifié c'est pareil. Donc le résultat renvoyé par le classifieur quand on utilise une version modifiée du softmax est le même.

La température a bien pour effet de nuancer les résultats. Pour cela étudions la fonction $g_j : T \mapsto f_{j,T}(x)$ pour un x fixé. La fonction g_j est dérivable sur $\mathbb{R}_+^*$, et sa fonction dérivée vaut pour un x fixé :

$$\begin{aligned} \forall T > 0, \quad g_j'(T) &= \frac{-\frac{x_j}{T^2} e^{x_j/T} C(x) + e^{x_j/T} \sum_{k=1}^{n} \frac{x_k}{T^2} e^{x_k/T}}{C(x)^2} \\ &= \frac{e^{x_j/T}}{T^2 C(X)^2} \left(\sum_{k=1}^{n} (x_k - x_j) e^{x_k/T} \right), \end{aligned}$$

avec $C(x) = \sum_{k=1}^{n} e^{x_k/T}$. On remarque que pour le x_j maximal, alors pour tout k, $x_k - x_j \leq 0$, donc la fonction est décroissante en T. On en déduit alors que si $T > 1$, alors la confiance va diminuer. Comme on a affaire à une probabilité et que la somme des sorties vaut 1, d'autres classes vont voir leur confiance augmenter. Le phénomène de nuance est ainsi obtenu.

La température "aplatit" la jacobienne. Diminuer la jacobienne permet d'affiner notre travail de 2.3.2. Voyons ce qui change au développement de cette partie. La fonction $f_{j,T} : x \mapsto \frac{e^{x_j/T}}{\sum_{k=1}^{n} e^{x_k/T}}$ admet un gradient pour $x \in \mathbb{R}^n$ et vaut :

$$\nabla f_{j,T}(x) = (\nabla f_{j,i,T}(x))_i,$$

avec

$$\nabla f_{j,i,T}(x) = \begin{cases} \dfrac{-e^{x_j/T} e^{x_i/T}}{T(\sum_{k=1}^{n} e^{x_k/T})^2} & \text{si } i \neq j \\ \dfrac{e^{x_i/T}}{T \sum_{k=1}^{n} e^{x_k/T}} \left(1 - \dfrac{e^{x_i/T}}{\sum_{k=1}^{n} e^{x_k/T}}\right) & \text{sinon} \end{cases}.$$

On a alors $\|\nabla f_{j,j,T}\|_\infty \leq \frac{1}{2T}$, il suffit de reprendre ce qui a été fait en 2.3.2. Augmenter T permet donc d'améliorer la borne mathématique, tout en la contrôlant. Ce résultat ne vaut que si les poids W ont une norme comparable que ce soit pour $T = 1$ (le cas softmax classique) ou un T quelconque. En effet, il est naturel de se demander si les poids dans le cas softmax modifié ne sont pas de l'ordre de TW, avec W les poids dans le cas softmax classique. Dans ce cas ajouter une température importe peu et la réponse ne sera pas nuancée. C'est ce qui risque d'arriver d'ailleurs si le nombre d'étapes d'apprentissage est grand. Mais étant donné que le gradient est $1/T$ fois plus petit que dans le cas softmax, le nombre d'étapes nécessaire pour une étape classique correspond donc à T étapes dans le cas softmax modifié. Ainsi le nombre d'étapes d'apprentissage est kT avec k un entier pas trop grand.

Remarque. Si on prend un T très grand, alors le réseau de neurones renverra un vecteur de probabilité proche d'une loi uniforme. En effet :

$$\forall x, \quad \lim_{T \to \infty} e^{x/T} = 1,$$

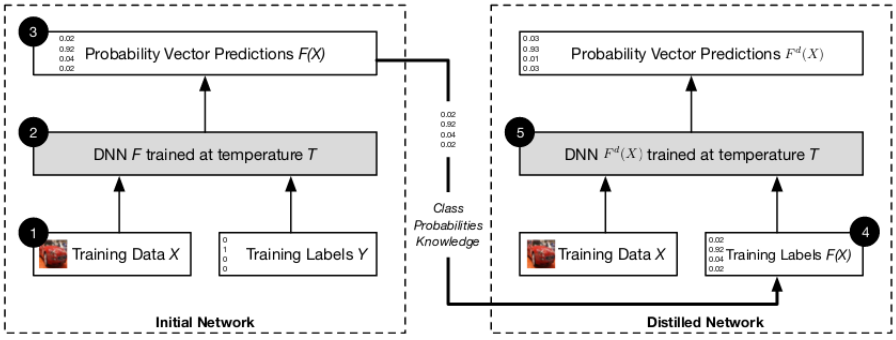
d'où :

$$\forall j, \quad \lim_{T \to \infty} \frac{e^{x_j/T}}{\sum_{k=1}^{n} e^{x_k/T}} = \frac{1}{n}.$$

Remarque. La température agit comme une régularisation. Hélas, la victoire n'est que partielle, les bosses et les creux sont préservés.

3.2.1 Defensive distillation

Telle que présentée précédemment la température n'a que pour impact de lisser la fonction, mais à aucun moment l'existence d'adversaire n'est plus difficile. En effet, l'important ce n'est pas la probabilité d'être ou de ne pas être, mais plutôt de connaître la classe qui a la plus grande probabilité. Or nous avons montré dans la partie précédente que la probabilité la plus grande restait la plus grande, malgré la température. C'est pour cette raison que Papernot et al. ont dans leur article [5], ajouté un deuxième réseau de neurones, dont l'architecture est identique au premier, qui utilise les résultats nuancés comme base de données d'entraînement. C'est ce deuxième réseau de neurones qui servira pour notre classifieur. Expérimentalement ils disent que c'est plus efficace. (N.D.A. je présente cette méthode, bien que je ne l'ai pas implémentée, car c'est une méthode classique).



Revenons un peu sur ce deuxième réseau de neurones. Le but de ce réseau est donc de ressembler au premier : on garde la même architecture, la même température, et on souhaite qu'il renvoie les mêmes résultats. Pourtant on a vu que le premier réseau ne rend pas le classifieur plus robuste, donc lui ressembler n'est pas un but en soi. L'astuce repose sur le fait que le second réseau n'apprend pas sur toutes les images possibles, ainsi les adversaires pour le premier réseau n'ont pas de raison de l'être pour le second (on ne s'est pas entraîné dessus donc il n'y a pas de raisons pour que le second réseau se trompe aussi).

Cette méthode a deux inconvénients majeurs que l'on a déjà mentionnés :

- il ne faut pas que le premier réseau soit trop entraîné (sinon le rôle de la température devient caduque),
- le but du second réseau est de ressembler au premier qui a les mêmes défauts qu'un classifieur classique.

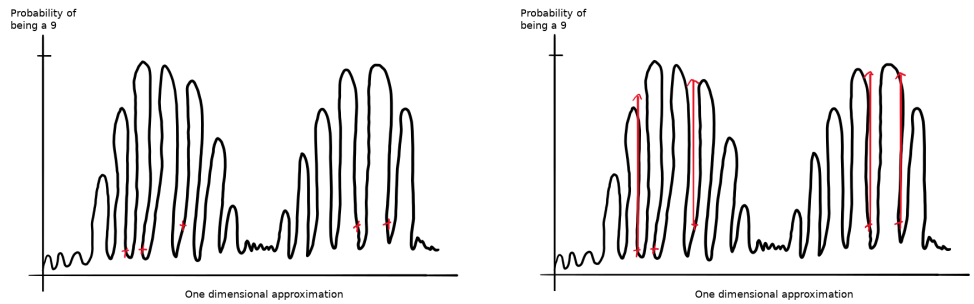


FIGURE 3.2 – *Dans cette figure on peut voir l'impact du raisonnement de l'algorithme de Madry et al. sur notre classifieur : on génère des adversaires (points rouges) et on ré-entraîne le réseau grâce à eux. Cette dernière étape a pour but de bien classifier nos adversaires, cela revient ainsi à remonter notre courbe en ces points, d'où les flèches.*

Finalement, il n'y a rien dans cette méthode qui corrige ces deux défauts. C'est pour ces raisons que je reste très critique vis-à-vis de cette méthode (même si nous ne l'avons pas testée).

3.3 Les stratégies de type Madry

Un autre raisonnement assez naturel pour répondre à ce problème consiste à se dire que finalement le réseaux de neurones sont très efficaces. Ne suffit-il donc pas d'augmenter la base de données avec des exemples adverses ? Le réseau de neurones ayant également des adversaires dans sa base d'apprentissage se robustifiera (cf Figure 3.2).

Il y a deux manières de procéder : soit en faisant de l'online training, soit offline. L'online learning consiste à combiner l'apprentissage aux expériences, il est ainsi possible d'adapter nos expériences pour rendre l'apprentissage plus efficace. Dans notre cas, cela se manifesterait en mélangeant création d'adversaire et entraînement de $f\theta$. L'offline learning, quant à lui, consiste à d'abord faire toutes les expériences avant de passer à l'apprentissage. Dans notre cas, l'offline learning serait : on génère un adversaire pour chaque image, et ensuite on adapte $f\theta$ grâce à cette base de données augmentée. Nous nous sommes intéressés à l'online learning pour deux raisons, la première car c'est celle que Madry et al. utilisent dans leur article [4] et la seconde parce que c'est en général plus efficace que l'offline training.

3.3.1 L'article de Madry et al.

Intéressons nous particulièrement à l'article de Madry et al. dont on s'inspire ([4]. L'article essaie d'étudier la robustesse en cherchant à résoudre pour un échantillon de N données :

$$\tilde{\theta} = \arg\min_{\theta} \left\{ \frac{1}{N} \sum_{i=1}^{N} \max_{\|r\|_\infty < \epsilon} \mathcal{L}(f_\theta(x_i + r), y_i) \right\}.$$

Pour trouver le $\tilde{\theta}$, ça se fait en deux temps. Dans un premier temps on va essayer de générer un adversaire à x dans $\mathcal{B}_\infty(x, \epsilon)$ et donc s'intéresser à :

$$\max_{\|r\|_\infty < \epsilon} \mathcal{L}(f_\theta(x + r), y),$$

Dans un second temps on s'attachera à trouver $\tilde{\theta}$.

Étape 1 : Pour cela ils s'inspirent de la stratégie proposée par Goodfellow et al. [3]. Faisons un peu d'histoire. Initialement Goodfellow et al. proposaient une attaque en une étape. Leur but était comme pour Madry et al. de minimiser l'écart entre l'adversaire et l'image originale grâce à la norme infinie. Pour cela Goodfellow et al. ont adapté la descente en gradient en :

$$x + \epsilon \operatorname{sign}(\nabla_x \mathcal{L}(f_\theta(x), y_0)).$$

Cette méthode, qui est alors utilisée pour réaliser une unique étape utilise la fonction $x \mapsto \epsilon \operatorname{sign}(x)$. Cette astuce a un double avantage, à la fois elle assure que l'adversaire reste dans $\mathcal{B}_\infty(x, \epsilon)$ et en même temps elle permet de garder l'esprit d'une descente en gradient classique.

Madry et al. vont ensuite vouloir étendre cette astuce en réalisant plusieurs étapes. Il faut alors s'assurer que l'adversaire reste bien dans $\mathcal{B}_\infty(x, \epsilon)$. La solution naturelle, et celle qui a été utilisée consiste à prendre la projection.

$$x_{t+1} = \Pi_{\mathcal{B}_\infty(x_0, \epsilon)} \left(x_t + \alpha \operatorname{sign}(\nabla_{x_t} \mathcal{L}(f_\theta(x_t), y_0)) \right).$$

Dans l'article de Madry on ne parle pas beaucoup de α, mais *a priori* on a : $\epsilon/\text{nombredetapes} \leq \alpha \leq \epsilon$. Cela nous permet bien d'avoir quelques chose qui ressemble à :

$$\max_{\|r\|_\infty < \epsilon} \mathcal{L}(f_\theta(x + r), y).$$

Étape 2 : Une fois qu'on a généré un échantillon d'adversaires, il s'agit de trouver un meilleur paramètre θ. Pour cela, il suffit de continuer l'apprentissage de notre classifieur avec le nouvel échantillon d'adversaires que l'on vient de générer. Cela revient alors à augmenter la base de données d'apprentissage. Une fois la phase de ré-apprentissage terminée on revient à l'étape 1 (c'est l'online learning).

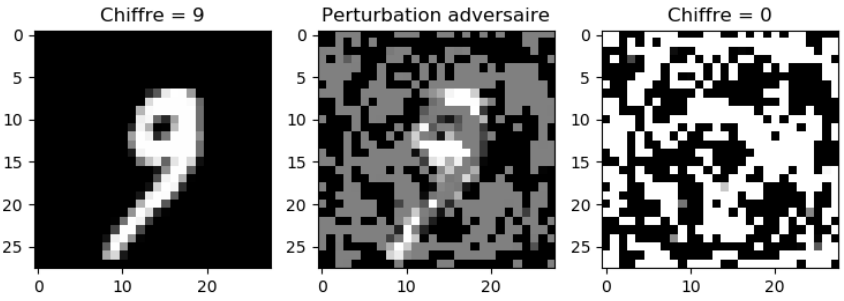
Remarque. Cette méthode a, à mes yeux, deux inconvénients majeurs. D'une part elle est assez longue, malgré un nombre d'étapes constant de descente en gradient. D'autre part, il suffit alors de localement avoir un gradient nul pour être battu, or ce que fait cette méthode c'est d'aplatir localement la courbe. C'est pour éviter ce problème que Madry et al. vont ajouter du bruit aléatoire à son image. On en reparle brièvement à la fin du chapitre.

3.3.2 Proposition pour accélérer la méthode de Madry

Comme il a été dit dans la remarque précédente, cette méthode est un peu longue. En effet pour chaque image il faut créer un adversaire (ou une image qui s'en approche). Donc si trouver un adversaire prend 1 seconde, quand on voudra que la base de données d'adversaires soient aussi grande que la base de données d'entraînement il aurait fallu 60 000 secondes (soit 16h40 et ce sans prendre en compte le temps d'entraînement et de réentraînement de notre classifieur). Ce calcul, bien qu'arbitraire, donne un bon ordre de grandeur du temps nécessaire pour utiliser la méthode précédente.

Rappelons que la méthode de descente en gradient sert à trouver le meilleur r, vérifiant $\|r\|_\infty < \epsilon$ qui maximise $\mathcal{L}(f_\theta(x+r), y)$. Naïvement on s'est alors demandés : et si au lieu d'utiliser un algorithme de descente en gradient on utilisait un réseau de neurones pour résoudre ce problème. Cette nouvelle méthode ne sera pas aussi efficace que celle proposée par Madry et al., mais elle a l'avantage de pouvoir générer 60 000 adversaires beaucoup plus rapidement (de l'ordre de la dizaine de minutes).

Dans un premier temps on a simplement cherché à maximiser $\mathcal{L}(f_\theta(x+r), y)$, indépendemment des contraintes. On obtenait des adversaires de cette forme :

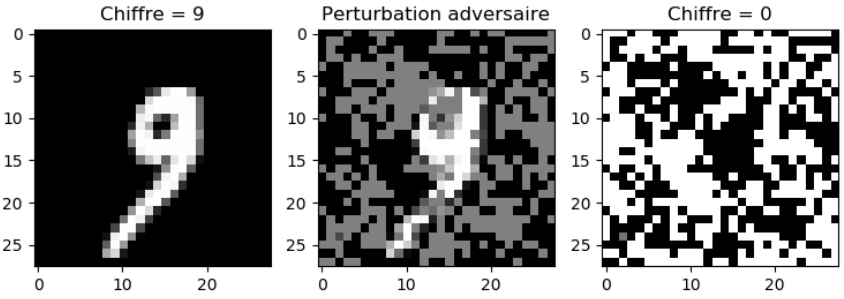


L'image adverse ne ressemble pas tant que ça à l'image initiale, ça ne va pas du tout ! Pour pallier ce problème on a alors simplement ajouter la contrainte sous forme de terme de régularisation. On a donc maximisé :

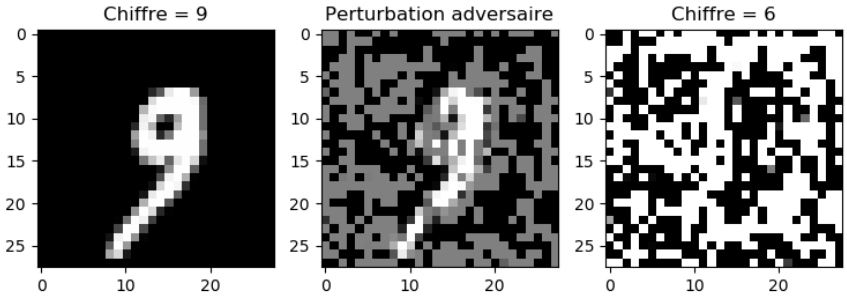
$$\mathcal{L}(f_\theta(x+r), y) + c\|r\|,$$

avec c la constante de régularisation.

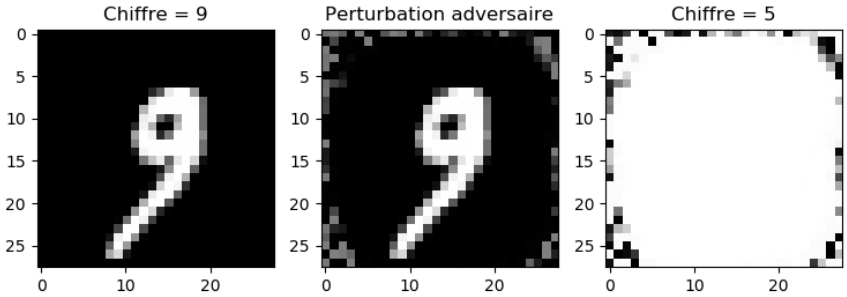
Contrairement à ce qui a été présenté dans la partie précédente la norme choisie, n'est pas la norme infinie. Ce choix est justifié pour deux raisons : je pense que cette norme est meilleure pour des images colorées, car chaque pixel est ainsi codé par un triplet RGB et donc il est plus facile d'avoir une norme infinie petite ; l'autre explication est plus importante et d'ordre pratique. Comme on peut le voir sur la figure ci-après les résultats sont très médiocres. (On a pris différentes valeurs pour le paramètre de régularisation, respectivement $10^{-4}, 1, 10^4$). Cela s'explique parce que la norme infinie n'a d'impact que sur un petit nombre de pixels.



Madry accéléré avec pour terme de régularisation : $10^{-4}\|r\|_\infty$

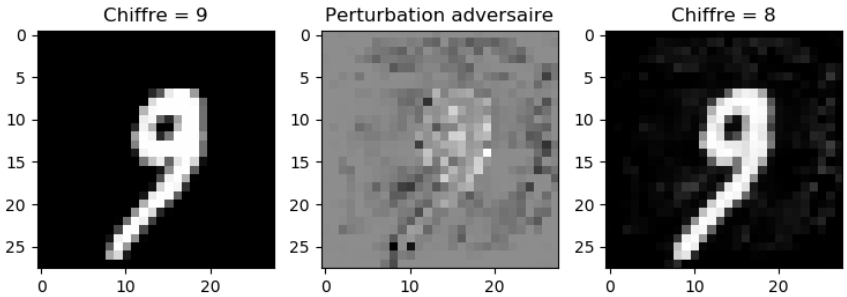


Madry accéléré avec pour terme de régularisation : $\|r\|_\infty$



Madry accéléré avec pour terme de régularisation : $10^4\|r\|_\infty$

On a donc choisit la norme 2. C'était le choix de Szegedy et al. dans leur article après tout. Les adversaires générés sont tout de suite meilleurs.



Avant d'atteindre ce réusltat il a fallu faire une étude pour savoir quel était le bon paramètre de régularisation. L'intuition nous dit qu'il faut choisir c de sorte que $c\|r\|$ soit de l'ordre de $\mathcal{L}(f_\theta(x+r), y)$, cela donne $c \simeq 100$. En revanche, on a souhaité s'assurer que notre générateur générait des adversaires, et la valeur $c = 100$ est un bon compromis. En effet, il y a 99% des images générées qui sont des adversaires (pour $c = 200$, il n'y en a plus

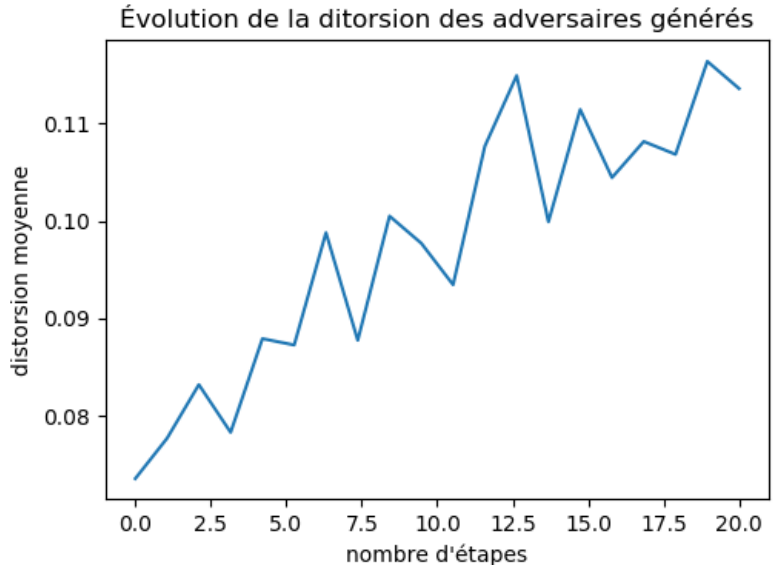


FIGURE 3.3 – *Cette figure illustre l'évolution de la distorsion moyenne des adversaires générés grâce à notre générateur. On remarque que la distorsion augmente avec le nombre d'étapes. Cela signifie, que pour générer des adversaires il faut aller les chercher de plus en plus loin, ie : notre classifieur est de plus en plus robuste*

que 95%, et pour $c = 50$ il y en a $99,5\%$).

Ce qui est long c'est d'entraîner notre générateur, mais une fois que c'est fait, générer 60 000 adversaires est quasi instantané. On peut alors ajouter ces 60 000 images adverses à notre base de données d'images et modifier notre classifieur. On obtient ainsi une stratégie plus rapide et qui semble tout aussi efficace que celle proposée par Madry et al. [4] :

1. entraîner notre classifieur avec la base de donnée,
2. entraîner un générateur d'adversaire pour le classifieur,
3. augmenter la base de donnée avec les 60 000 adversaires générés,
4. revenir à l'étape 1.

Remarque. L'expérience qui a permis de renvoyer la figure (3.3) n'est pas optimale, car on ne teste pas la distorsion avec une descente en gradient (méthode classique), mais plutôt avec notre générateur. C'est d'autant plus vrai que la distorsion donnée par notre générateur à la première étape vaut 0.07 et non 0.02 (méthode classique). Il faudrait donc idéalement tester la robustesse avec des méthodes classiques, mais l'apparition des NaNs com-

promet un peu les résultats, ce qui dans notre cas, veut donc dire que les gradients s'aplatissent. On peut donc bien s'attendre à ce que le réseau se robustifie.

Madry et al. ont ajouté un terme de bruit à leurs images avant de leur chercher des adversaires. Cela permet plusieurs choses, d'une part trouver des adversaires malgré des gradients petits au niveau des données (problème rencontré au paragraphe précédent), d'autre part cela permet de proposer une diversité d'adversaire. En effet, il y a autant d'adversaires différents qu'il y a de bruits possibles pour une même image. C'est l'idée que le bruit pouvait être créateur de diversité qui nous a poussé à nous intéresser aux GANs

Chapitre 4

Les GANs et nos impasses

Les GANs pour Generative Adversarial Networks (ou Réseaux Adverses Génératifs en français) sont une classe d'algorithmes qui consiste en la mise en concurrence de deux réseaux de neurones : un discriminateur et un générateur. Le rôle du générateur est de générer des données, tandis que le rôle du discriminateur est de déterminer si les données sont réelles ou générées. Ces algorithmes sont notamment utilisés pour générer des images, par exemple des tableaux qui imitent le style de Rembrandt, ou faire vieillir un portrait. Dans ces deux cas le but est de générer des images (rôle du générateur) crédibles (rôle du discriminateur).

Dans notre cas le discriminateur correspondrait au classifieur et notre générateur aurait pour but de générer des adversaires. On s'attend à ce que la mise en compétition de ces deux réseaux permette une amélioration de la robustesse de notre classifieur. Finalement c'est exactement la stratégie de type Madry accélérée. On voit à peu près comment on veut utiliser les GANs, cela permettrait, entre autre, d'utiliser la littérature sur le sujet.

4.1 Présentation des GANs

Dans cette partie nous allons présenter les GANs comme ils ont été présentés la première fois par Goodfellow et al. en 2014 [2].

Le discriminateur est une fonction qui prend une image en entrée et renvoie la probabilité, d'après le réseau, d'être une image réelle. Le générateur, quant à lui, quelle est son entrée ? Goodfellow et al. proposent de générer des images à partir de bruits (si on veut générer des images ressemblant à celles présentes dans la base MNIST, on peut choisir comme entrée du générateur $x \sim \mathcal{U}([0,1]^{100})$). Notre générateur idéal vérifie alors pour tout

z, $D(G(z)) = 1$ (ie : d'après le discriminateur la probabilité que l'image générée soit réélle est de 1) et on peut choisir la fonction G qui minimise :

$$\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))],$$

ou de manière équivalente :

$$V(G, D) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))].$$

Le discriminateur essaie d'identifier les images réélles et générées. Il veut alors maximiser à la fois $\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)]$ et $\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$. On cherche alors la fonction D qui maximise $V(G, D)$.
Nous remarquons que, dans les deux cas la fonctions à minimiser ou maximiser dépend de l'autre réseau. On souhaite donc que les deux réseaux apprennent en parallèle. On va alors régulièrement alterner les phases d'apprentissage du générateur avec celles du discriminateur.

Nous remarquons que, dans les deux cas la fonction à minimiser ou maximiser dépend de l'autre réseau. On souhaite donc que les deux réseaux apprennent en parallèle. On va alors régulièrement alterner les phases d'apprentissage du générateur avec celles du discriminateur. On peut alors voir ce problème comme un problème de minimax de théorie des jeux, où l'on cherche à résoudre :

$$\min_G \max_D V(G, D).$$

Remarque. La formule précédente est une manière de mathématiser le problème. Cependant, d'un point de vue purement objectif elle ne semble pas parfaitement correspondre au problème des GANs. En effet, le principe des GANs repose sur le yo-yo permanent entre l'apprentissage du générateur et celui du discriminateur. Cela revient donc à intervertir régulièrement la recherche du maximum et du minimum de $V(G, D)$ et en général on n'a pas $\min\max f(x, y) = \max\min f(x, y)$. Nous allons malgré cette remarque poser le problème de la même manière.

Propriété 4.1. *Pour un G fixé, le discriminateur optimal est :*

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

avec $p_g(x)$ est la probabilité de trouver x avec G.

Idée de la preuve : Il s'agit de remarquer que la fonction $x \mapsto a\log(x) + b\log(1-x)$ atteint son maximum en $\frac{a}{a+b}$, et de reconnaître cette forme dans $V(G, D)$. □

Propriété 4.2. *Le minimum global de $V(G, D_G^*)$ est atteint si et seulement si $p_g = p_{data}$.*

Démonstration. Dans un premier temps remarquons que :

$$\begin{aligned}
V(G, D_G^*) &= \mathbb{E}_{x\sim p_{data}(x)}\left[\log \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right] + \mathbb{E}_{x\sim p_g(x)}\left[\log \frac{p_g(x)}{p_{data}(x)+p_g(x)}\right] \\
&= \mathbb{E}_{x\sim p_{data}(x)}\left[\log \frac{p_{data}(x)}{p_{data}(x)+p_g(x)} + \log(2)\right] - \log(2) \\
&\quad + \mathbb{E}_{x\sim p_g(x)}\left[\log \frac{p_g(x)}{p_{data}(x)+p_g(x)} + \log(2)\right] - \log(2) \\
&= -\log(4) + \mathbb{E}_{x\sim p_{data}(x)}\left[-\log \frac{p_{data}(x)+p_g(x)}{2} + \log(p_{data}(x))\right] \\
&\quad + \mathbb{E}_{x\sim p_g(x)}\left[\log \frac{p_{data}(x)+p_g(x)}{2} + \log(p_g(x))\right] \\
&= -\log(4) + KL\left(p_{data}\,\middle\|\,\frac{p_{data}}{p_{data}+p_g}\right) + KL\left(p_{data}\,\middle\|\,\frac{p_g}{p_{data}+p_g}\right) \\
&= -\log(4) + 2JSD(p_{data}\|p_g),
\end{aligned}$$

où KL est la divergence de Kullback-Leibler et JSD la divergence de Jensen-Shannon. La divergence de Jensen-Shannon est positive et s'annule si et seulement si $p_{data} = p_g$. □

Remarque. Ce théorème montre que le but des GANs est d'avoir $p_g = p_{data}$ et donc d'avoir un générateur qui génère des images qui semblent réelles. En pratique, les GANs ont besoin de beaucoup de temps d'apprentissage. En effet, il faut réussir à générer des données réalistes avec du bruit qui ne donne aucune information. En contrepartie, une fois les GANs bien entraînés les résultats sont impressionnants (figure 4.1).

4.2 Les Essais

Nous avons présenté les GANs telles qu'ils ont été la première fois proposés en 2014 [2], mais on remarque que l'enjeu n'est pas le même que dans notre cas. En effet, le but initial des GANs est de générer une image réelle à l'aide de bruits, or nous à l'inverse on veut générer une image qui a l'air réelle mais qui trompe notre classifieur. Il nous faudrait alors avoir un générateur (qui génère des adversaires), un discriminateur (qui essaie de deviner si l'image est générée ou non, ie : l'image semble réelle) et un classifieur qui

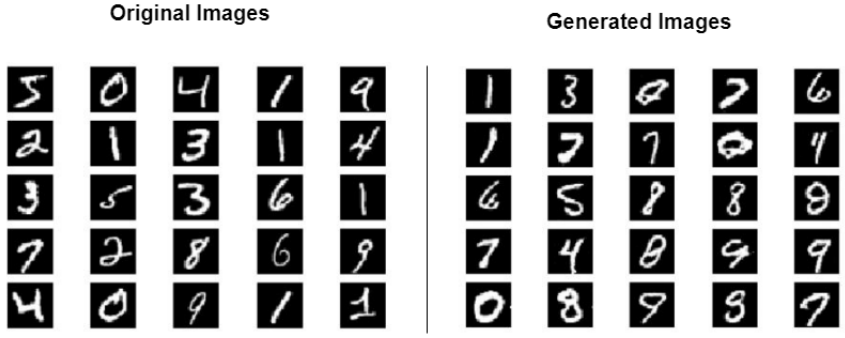


FIGURE 4.1 – *Cette figure illustre l'efficacité des GANs, une fois qu'elles ont été bien entraînées. En effet les images réelles ont l'air tout aussi vraies que les images originales.* (Source : https ://theailearner.com/wp-content/uploads/2019/09/real_vs_fake_gans.png)

se ferait trompé. Ce raisonnement naturel, a été vite mis de côté. Le but n'est pas de multiplier les réseaux, mais plutôt de mettre en compétition classifieur et générateur. Voilà pourquoi on s'est intéressés à d'autres types de GANs.

Il y a quelques idées des GANs classiques que nous souhaiterions conserver. Nous avons vu dans le bruit la possibilité de créer plusieurs adversaires pour une image donnée sans que ça augmente le temps d'apprentissage du générateur. En effet ce qui est long c'est de générer un grand nombre d'adversaire, donc si au lieu d'en exhiber un unique, on peut en générer 3 ou 4, on peut espérer aller 3 ou 4 fois plus vite. Le danger de cette méthode est que nos 3 ou 4 adversaires soient trop semblables pour finalement être considéré comme le même adversaire. En effet si nos trois adversaires n'ont qu'un seul pixel de différence qui change légèrement sa valeur, c'est presque comme si on avait généré le même adversaire. C'est dans ce but que nous nous sommes documentés sur différents types de GANs, notamment les Optimal Transport - GANs ([6]), mais nous n'avons pas réussi à trouver un moyen de contrôler le bruit pour nous assurer qu'il apporte de la diversité. Je m'explique. Si notre générateur prend en entrée l'image dont il essaie de créer un adversaire et du bruit, il a tout intérêt à mettre des poids nuls devant le bruit et de se concentrer uniquement sur l'image (en pratique c'est exactement ce qu'il fait). Il faut alors réussir à ajouter un critère de bruit dans le problème d'optimisation.

J'ai alors envisagé le problème à l'envers, et si au lieu de mettre le bruit en entrée, on mettait le bruit en sortie lors des étapes d'apprentissage. Cela permettrait d'ajouter un terme de bruit qui sera obligatoirement pris en compte lors de la retro-propagation sans avoir à l'ajouter à la fonction perte. Cette idée n'est restée qu'à l'état de prototype, car c'est à ce moment là que nous avons eu des résultats très encourageant avec notre méthode de Madry accélérée. Finalement grâce à cette méthode, générer des adversaires n'est plus très long (60 000 adversaires en moins de 30 minutes). En alternant les étapes d'entraînement du classifieur et celles du générateur, les adversaires générés sont ceux, qui, pour une étape de classifieur et une image données sont les plus efficaces. La diversité n'est plus un enjeu depuis que nous avons un générateur rapide, donc le rôle que l'on voulait prêter au bruit n'est plus un enjeu.

Cependant ce n'est pas parce que la diversité n'est plus un enjeu majeur que les GANs perdent totalement leur intérêt. En effet avec la méthode de Madry accélérée, la base de donnée initiale augmente à chaque étape. Donc au bout de neuf étapes, la base de donnée a vu sa taille multipliée par 10. Cela crée alors deux problèmes majeurs : il faut stocker une base de donnée toujours plus grande (problème de mémoire) et notre classifieur a un temps d'apprentissage toujours plus long. Or dans les GANs on ne stocke nulle part les images générées. On peut alors se poser la question suivante : Ne peut-on pas revoir notre étude des GANs mais cette fois dans le but de générer des adversaires sans les stocker ?

Chapitre 5

Conclusion

Nous nous sommes d'abord intéressés aux réseaux de neurones comme une méthode d'optimisation, avant de les utiliser pour résoudre le problème de classification de la base de données MNIST (chapitre 1.) Nous avons ensuite vu qu'il était très facile de générer des adversaires et donc que les réseaux de neurones n'étaient pas très robustes. Nous avons décidé d'étudier cette robustesse, dans un premier temps en exhibant une borne de robustesse mathématiques (chapitre 2) avant de nous intéresser aux méthodes classiques que la littérature sur le sujet propose. C'est ce qui nous a amené à étudier l'impact de la régularisation, la méthode dite de la Defensive Distillation et celle de Madry et al. Nous avons ensuite proposé une manière d'accélérer cette dernière méthode en utilisant un deuxième réseau de neurones pour générer les adversaires (chapitre 3). Cette stratégie semble assez proche des GANs (Generative Adversarial Networks), c'est ce qui nous a poussé à nous y intéresser (chapitre 4).

Ce sujet passionnant n'a été qu'effleuré ici, et il reste encore beaucoup à faire. Nous pouvons par exemple penser à la combinaison de plusieurs stratégies comme ajouter de la régularisation et du dropout à la Defensive Distillation ; à poursuivre la recherche du côté des GANs, se renseigner sur les auto-encodeurs, sur l'une des nombreuses méthodes qui allient plusieurs réseaux de neurones ; à s'intéresser à d'autres structures de réseaux comme le CNN ; ou tout simplement à contrôler et/ou étudier les poids W.

Vous trouvrez les codes des différentes méthodes présentées sur : `https://github.com/MatthieuPoyer/Stage-2020`.

Annexe A

Le dual de L_p s'identifie à L_q

On veut montrer que si on a $\frac{1}{p} + \frac{1}{q} = 1$ et $1 \leq p \leq +\infty$, alors le dual de L_p s'identifie (isométriquement isomorphe) à L_q.

L'inégalité de Hölder nous dit que pour $f \in L^p$, $g \in L^q$ et $\frac{1}{p} + \frac{1}{q} = 1$, alors $fg \in L^1$ et :

$$\|fg\|_1 \leq \|f\|_p \|g\|_q.$$

En choisissant un bon g, on peut déduire :

$$\|f\|_p = \sup_{g \in L^q \setminus \{0\}} \frac{| < f, g > |}{\|g\|_q}.$$

Soit Ψ l'application définie par :

$$\Psi : \begin{cases} L^p \longrightarrow (L^q)' \\ f \longmapsto \begin{cases} L^q \longrightarrow \mathbb{R} \\ g \longmapsto < f, g > . \end{cases} \end{cases}$$

Il apparaît alors que Ψ est une isométrie linéaire injective d'après l'égalité précédente. Il ne reste plus qu'à montrer que Ψ est surjective (pour $1 \leq p \leq +\infty$) et donc bijective.

Soit $\psi \in (L^q)'$, le but est donc de montrer qu'il existe une fonction f de L^p telle que $\Psi(f) = \psi$. On va restreindre l'étude à $L^p(X)$ et $(L^q(X))'$, avec $\mu(X) < +\infty$ où μ est la mesure de Lebesgue. (en effet dans notre cas on s'intéresse à $[-\epsilon, 1+\epsilon]$[784]). On définit pour $E \subset X$ un ensemble mesurable :

$$\lambda(E) = \psi(\mathbb{1}_E).$$

Soit E une réunion d'ensembles mesurables disjoints E_i. On note $A_k = \bigcup_i^k E_i$, pour $q \neq +\infty$ on trouve :

$$\|\mathbb{1}_E - \mathbb{1}_{A_k}\|_q = [\mu(E - A_k)]^{1/q} \xrightarrow[k\to+\infty]{} 0$$

Par continuité de ψ on a $\sum_i^k \lambda(E_i) = \lambda(A_k) \to \lambda(E)$ et $\lambda(\emptyset) = 0$, donc λ est une mesure. On remarque également que si $\mu(E) = 0$, alors $\lambda(E) = 0$. On vérifie alors $\lambda \ll \mu$, on peut donc utiliser le théorème de Radon-Nikodym qui assure l'existence d'une fonction $f \in L^1$ telle que pour tout ensemble mesurable E, on ait :

$$\psi(\mathbb{1}_E) =< f, \mathbb{1}_E > .$$

Par linéarité on a pour toute fonction mesurable étagée g :

$$\psi(g) =< f, g > .$$

On peut ensuite, dans un premier temps, étendre le résultat précédent à toute fonction $g \in L^\infty$, puis dans un second, à toute fonction $g \in L^q$. On souhaite alors avoir $f \in L^p$ pour terminer de montrer la surjectivité de Ψ.

Si $q = 1$:
Alors pour tout ensemble mesurable E de mesure non nulle :

$$\begin{aligned} & | < \mathbb{1}_E, f > | \leq \|\mathbb{1}_E\|_1 \|\psi\| = \mu(E)\|\psi\| \\ \iff \qquad & \frac{1}{\mu(E)} | < \mathbb{1}_E, f > | \leq \|\psi\|, \end{aligned}$$

avec $\|\psi\| = \sup |\psi(g)|/\|g\|_q < +\infty$ (car ψ est une forme linéaire continue). On a alors, $f(x) \leq \|\psi\|$ presque partout, d'où $\|f\|_\infty \leq \|\psi\|$ (et $f \in L^\infty$). En effet, soit $x \in X$ tel que $f(x) > \|\psi\|$, il existe $r > 0$ tel que $\mathcal{B}(f(x), r) \subset X \setminus \mathcal{B}_F(0, \|\psi\|)$. On va alors choisir $E = f^{-1}(\mathcal{B}(f(x), r))$, alors :

$$\begin{aligned} \frac{1}{\mu(E)} | < \mathbb{1}_E, f - f(x) > | &\leq \frac{1}{\mu(E)} \int_E |f - f(x)| d\mu \\ &\leq \frac{1}{\mu(E)} \int_E r d\mu \\ &\leq r, \end{aligned}$$

donc $\frac{1}{\mu(E)} | < \mathbb{1}_E, f > | \in X \setminus \mathcal{B}_F(0, \|\psi\|)$, or par hypothèse $\frac{1}{\mu(E)} | < \mathbb{1}_E, f > | \leq \|\psi\|$. On trouve une contradiction, donc $\mu(E) = 0$. donc $f(x) \leq \|\psi\|$ presque partout (ie : $f \in L^\infty$).

Si $1 < q < +\infty$:

On définit $E_n = \{x, |f(x)| \leq n\}$ et $g = \mathbb{1}_{E_n}|f|^{p-1}\frac{|f|}{f}$ quand $f(x) \neq 0$ et $g = 0$ sinon. Alors, $fg = |f|^p$ sur E_n, $g \in L^\infty$ et $|g|^q = |f|^{qp-q} = |f|^p$ (car $1/p + 1/q$), on en déduit alors :

$$\int_{E_n} |f|^p d\mu = \int_X fg d\mu = \psi(g) \leq \|\psi\|\|g\|_q = \|\psi\| \left(\int_{E_n} |f|^p d\mu\right)^{1/q},$$

de sorte que :

$$\forall n \geq 1, \quad \int_X \mathbb{1}_{E_n}|g|^p d\mu \leq \|\psi\|^p.$$

Il suffit alors d'utiliser le théorème de convergence monotone pour obtenir : $\|g\|_q \leq \|\psi\|$, donc $g \in L^q$ et $f \in L^p$.

Bibliographie

[1] Alexandre Araujo, Laurent Meunier, Rafael Pinot, and Benjamin Negrevergne. Robust neural networks using randomized adversarial training, 2019.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[3] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

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

[5] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks, 2015.

[6] Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving GANs using optimal transport. In *International Conference on Learning Representations*, 2018.

[7] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013.