RECENT ADVANCES IN MACHINE LEARNING

2023-2024

Théo Lopès-Quintas

Séance 1

Rien ne sert de courir, il faut partir à point

1.1 Partir du bon pied

1.1.1 Explosion et disparition des gradients

Pour mieux comprendre le problème d'explosion et de disparition des gradients, nous devons nous pencher sur l'apprentissage d'un réseau : l'algorithme de backpropagation. Commençons par formaliser un réseau de neurones.

On considère un réseau de neurones composé de L couches avec N_l neurones à la couche $l \leqslant L$. Chacun de ces neurones est caractérisé par trois objets :

- **Poids** : un vecteur de paramètre qui associe à chacun des coefficients du vecteur d'entré du neurones un poids
- **Biais** : un scalaire qui rend affine la transformation
- **Activation** : une fonction qui a partir de la transformation affine du vecteur d'entrée renvoie un nombre

La non-linéarité est introduite par la fonction d'activation. Résumons les notations introduites :
L'objectif d'un réseau de neurone, comme tous les algorithmes de Machine Learning, peut s'écrire sous forme d'un problème d'optimisation. Pour cela, on définit $\mathcal{C}$ la fonction de coût et on cherche à la minimiser. Par exemple dans un problème régression une fonction de coût possible est la MSE, et dans le cadre d'une classification la Cross-Entropy.

Puisque les notations sont déjà lourde, nous prendrons quelques liberté de notation pour privilégier des équations simple quand il n'y a pas d'ambiguïté. Ainsi, l'objectif de la back propagation est de calculer les quantités :

- $\dfrac{\partial C}{\partial w^l_{j,k}}$: pour calculer la mise à jour du k-ième poids du neurone $j \leqslant N_l$ à la couche $l \leqslant L$, avec $k \leqslant N_{l-1}$
- $\dfrac{\partial C}{\partial b^l_j}$: pour calculer la mise à jour du biais du neurone $j \leqslant N_l$ à la couche $l \leqslant L$

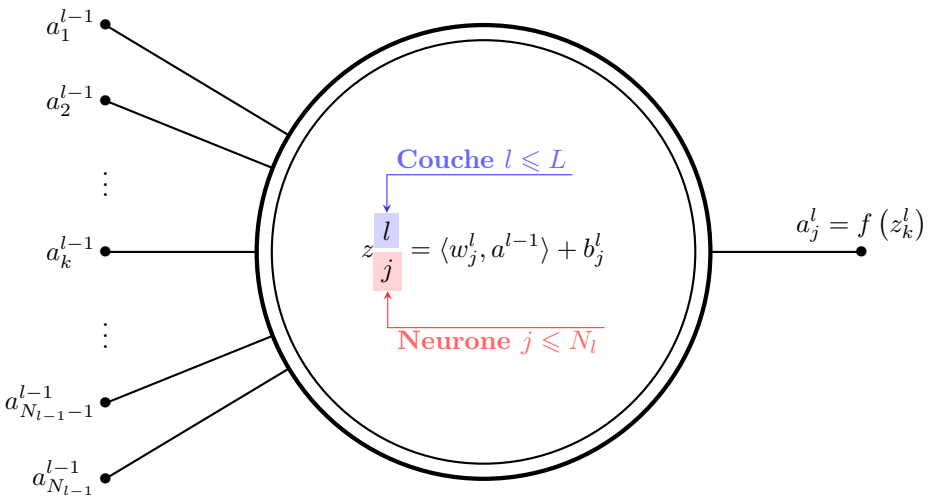


FIGURE 1.1 – Neurone j de la couche l paramétré par les poids $w_i^j \in \mathbb{R}^{N_{l-1}}$ et le biais $b_j^l \in \mathbb{R}$

Pour le faire, résolvons l'exercice suivant.

Exercice 1.1 (Back propagation)**.** *On reprend les notations définies précédemment.*

1. *Soit $l \leqslant L$, $j \leqslant N_l$ et $k \leqslant N_{l-1}$. Montrer que :*

$$\begin{aligned} \frac{\partial C}{\partial b_j^l} &= \frac{\partial C}{\partial z_j^l} \\ \frac{\partial C}{\partial w_{j,k}^l} &= a_k^{l-1} \frac{\partial C}{\partial z_j^l} \end{aligned}$$

2. *On note $\delta_j^l = \dfrac{\partial C}{\partial z_j^l}$ qui représente l'erreur du neurone $j \leqslant N_l$ à la couche $l \leqslant L$. Montrer que pour la dernière couche L, on a :*

$$\forall j \leqslant N_L, \quad \delta_j^L = f'(z_j^L) \frac{\partial C}{\partial a_j^L}$$

3. *On considère à présent la couche $l < L$. Montrer que l'on peut exprimer δ_j^l comme :*

$$\delta_j^l = \sum_{k=1}^{n_l} w_{j,k}^{l+1} \delta_k^{l+1} f'(z_j^l)$$

Solution. Nous conservons les notations précédentes et ferons des commentaires supplémentaires à chaque questions pour donner plus de liens à la résolution de l'exercice.

1. On considère le neurone $j \leqslant N_l$ à la couche $l \leqslant L$. Alors par le théorème de dérivation des

fonctions composées, on a :

$$\begin{aligned}\frac{\partial C}{\partial b_j^l} &= \frac{\partial C}{\partial b_j^l}\frac{\partial b_j^l}{\partial z_j^l} \quad \text{mais} \quad \frac{\partial b_j^l}{\partial z_j^l} = 1 \\ &= \frac{\partial C}{\partial z_j^l}\end{aligned}$$

On considère maintenant le k-ième poids du neurones. On a à nouveau avec le théorème de dérivation des fonctions composées :

$$\begin{aligned}\frac{\partial C}{\partial w_{j,k}^l} &= \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{j,k}^l} \quad \text{mais} \quad \frac{\partial z_j^l}{\partial w_{j,k}^l} = a_k^{l-1} \\ \frac{\partial C}{\partial w_{j,k}^l} &= a_k^{l-1}\frac{\partial C}{\partial z_j^l}\end{aligned}$$

On observe que pour le poids et le biais on a la même quantité $\dfrac{\partial C}{\partial z_j^l}$ qui apparait. Elle représente l'erreur du neurone $j \leqslant N_l$ à la couche $l \leqslant L$. L'exercice nous propose de la noter δ_j^l pour la suite.

2. On considère à présent spécifiquement la dernière couche. Soit $j \leqslant N_L$ un neurone de cette couche. Encore avec le théorème de dérivation des fonctions composées :

$$\begin{aligned}\delta_j^L &= \frac{\partial C}{\partial a_j^L}\frac{a_j^L}{\partial z_j^L} \quad \text{mais} \quad a_j^L = f(z_j^L) \\ &= f'(z_j^L)\frac{\partial C}{\partial a_j^L}\end{aligned}$$

Calculer la valeur de $\dfrac{\partial C}{\partial a_j^L}$ est aisé puisqu'il suffit de comparer la valeur prédite par le réseau à la valeur attendue. En combinant ce résultat à la question précédente, nous savons calculer la mise à jour des poids et des biais de la dernière couche.
Pour être capable de calculer la mise à jour des poids et des biais des couches précédentes, il faudrait être capable d'exprimer les δ_j^l en fonction de δ_j^L : probablement de proche en proche.

3. On considère à présent le j-ième neurone de la couche $l < L$. Par définition des notations, on a :

$$\begin{aligned}z_j^l &= \sum_{k=1}^{N_l} w_{j,k}^l a_j^{l-1} + b_j^l \\ z_k^{l+1} &= \sum_{i=1}^{N_{l+1}} w_{k,i}^{l+1} f(z_i^l) + b_k^{l+1}\end{aligned}$$

Ainsi, en appliquant plusieurs fois le théorème de dérivation des fonctions composées :

$$\begin{aligned}
\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\
&= \sum_{k=1}^{N_l} \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_k^l} \quad \text{et} \quad \frac{\partial z_k^{l+1}}{\partial z_k^l} = w_{k,j}^{l+1} f'(z_j^l) \\
&= \sum_{k=1}^{N_l} w_{k,j}^{l+1} f'(z_j^l) \frac{\partial C}{\partial z_k^{l+1}} \\
&= \sum_{k=1}^{N_l} w_{k,j}^{l+1} \delta_k^{l+1} f'(z_j^l) \quad \text{par définition des } \delta_j^l
\end{aligned}$$

Nous calculons donc l'actualisation des poids et des biais de proche en proche : d'où le terme back propagation !

□

En décrivant l'algorithme de back propagation, on comprend que chaque poids est calculés de proche en proche, en partant de la couche la plus proche à la plus profonde (celle juste après l'input). Puisque les calculs se font de proche en proche, alors il y a une possibilité qu'une suite géométrique apparaisse et fasse disparaître la magnitude du signal.

En effet, le signal de correction du j-ième neurones de la couche $l \leqslant L$ est :

$$\delta_j^l = \sum_{k=1}^{n_l} w_{j,k}^{l+1} \delta_k^{l+1} f'(z_j^l)$$

Dépendant clairement du précédent, on peut l'écrire sous la forme :

$$\begin{aligned}
\delta_j^l &= \sum_{k=1}^{n_l} w_{j,k}^{l+1} \delta_k^{l+1} f'(z_j^l) \\
&= \sum_{k=1}^{n_l} w_{j,k}^{l+1} f'(z_j^l) \times \left(\sum_{k=1}^{n_{l+1}} w_{j,k}^{l+2} \delta_k^{l+2} f'(z_j^{l+1}) \right) \\
&= \sum_{k=1}^{n_l} w_{j,k}^{l+1} f'(z_j^l) \times \left(\sum_{k=1}^{n_{l+1}} w_{j,k}^{l+2} f'(z_j^{l+1}) \times \left(... \left(\sum_{k=1}^{n_{L-1}} w_{j,k}^{L} f'(z_j^L) \delta_j^L \right) ... \right) \right)
\end{aligned}$$

Si l'on note $\gamma = \max_{x \in \mathbb{R}} f'(x)$ alors :

$$|\delta_j^l| \leqslant \gamma^{L-l+1} \left| \sum_{k=1}^{n_l} w_{j,k}^{l+1} \times \left(\sum_{k=1}^{n_{l+1}} w_{j,k}^{l+2} \times \left(... \left(\sum_{k=1}^{n_{L-1}} w_{j,k}^{L} \delta_j^L \right) ... \right) \right) \right|$$

Or, dans les premiers réseaux de neurones, la fonction sigmoid était utilisée. Dans ce cas : $\gamma = \frac{1}{4}$, d'où une probabilité forte que la force du signal des dernières couches soit faible et donc que le réseau n'apprennent pas correctement. Notons que la fonction tanh était également utilisée et que l'argument que nous avons construit ici ne suffit pas à conclure car $\gamma = 1$. Cependant, la même logique explique le problème : une composition répétée de nombreuse fois d'une quantité

de valeur inférieure à 1 conduit à une disparition des gradients. Le phénomène inverse s'explique de la même manière.

Ces phénomènes on fait perdre un temps considérable aux chercheurs : l'entraînement d'un réseau de neurones profond est lent, encore plus avec une puissance machine réduite. De plus, si l'on arrive pas à montrer que nos recherches peuvent fonctionner mieux que l'existant, on n'obtient pas de fonds supplémentaires pour continuer de mener les recherches. Face à ces deux problèmes, on pourrait s'en remettre à la chance.

> *Je ne crois pas qu'il y ait de bonne ou de mauvaise situation.*
> — Édouard Baer, jouant Otis dans Astérix et Obélix Mission Cléopâtre (2022)

En revanche, il existe un *bon* et un *mauvais* hasard. Voyons pourquoi.

1.1.2 Réponses Glorot et He

Une des manières de lutter contre ces deux phénomènes qui interviennent au cours de l'apprentissage du réseau est d'adopter une bonne initialisation des poids. Cela peut paraître surprenant : comment une initialisation peut-elle améliorer durablement un réseau de neurones ? Pour le comprendre, on considère un réseau de neurones à $L \in \mathbb{N}^*$ couches. On reprend la majorité des notations précédentes, mais on note z_k^l le résultat du neurone k à la couche l défini par :

Nombre de couche du réseau — Vecteur d'entrée de la couche l

$$\forall l \leqslant L,\ \forall k \in N_l,\quad z_k^l = \langle w_k^l,\ x^l \rangle$$

Nombre de neurones de la couche l — Matrice des poids du neurone k à la couche l

Nous avons omis ici le biais car notre analyse se situe à l'initialisation du réseau : il n'y a pas de raison d'introduire déjà un biais. Nous faisons les hypothèses supplémentaires :

- Indépendance et distribution : $\left(w^l\right)_{l\leqslant L}$, $\left(x^l\right)_{l\leqslant L}$ et $\left(z^l\right)_{l\leqslant L}$ sont indépendants et identiquement distribuées (respectivement)
- Indépendance : $\forall l \leqslant L,\ w^l \perp\!\!\!\perp x^l$
- Poids centrés : $\mathbb{E}\left[w^l\right] = 0$ et la distribution est symétrique autour de 0
- Résultats centrés : $\mathbb{E}\left[z^l\right] = 0$ et la distribution est symétrique autour de 0 : il n'y a pas de raison que les signaux qui se propage soient biaisé

Puisque l'on souhaite une bonne propagation des signaux dans les deux sens afin d'éviter une explosion ou une disparition des gradients, ce que l'on souhaite est que :

$$\forall l \leqslant L,\ \mathbb{V}\left[z^l\right] = \mathbb{V}\left[z^1\right] \tag{1.1}$$

Si c'est le cas, alors nous aurons lutter efficacement contre ces deux phénomènes. Calculons pour $l \leqslant L$ et $k \in N_l$:

$$
\begin{aligned}
\mathbb{V}\left[z_k^l\right] &= \mathbb{V}\left[\langle w_k^l, x^l\rangle\right] \\
&= \sum_{j=1}^{n_{\text{input}}} \mathbb{V}\left[w_{k,j}^l x_j^l\right] \text{ par indépendance} \\
&= n_{\text{input}} \mathbb{V}\left[w_{k,j}^l x_j^l\right] \text{ car identiquement distribué, pour } j \leqslant n_{\text{input}} \\
&= n_{\text{input}} \left(\mathbb{V}\left[w_{k,j}^l\right] \mathbb{V}\left[x_j^l\right] + \mathbb{E}\left[w_{i,j}^l\right]^2 \mathbb{V}\left[x_j^l\right] + \mathbb{E}\left[x_j^l\right]^2 \mathbb{V}\left[w_{k,j}^l\right]\right) \\
&= n_{\text{input}} \mathbb{V}\left[w_{k,j}^l\right] \left(\mathbb{V}\left[x_j^l\right] + \mathbb{E}\left[x_j^l\right]^2\right) \text{ car } \mathbb{E}\left[w_{i,j}^l\right] = 0 \\
&= n_{\text{input}} \mathbb{V}\left[w_{k,j}^l\right] \mathbb{E}\left[\left(x_j^l\right)^2\right]
\end{aligned}
$$

Rappelons que $x^l = f\left(z^{l-1}\right)$ avec f la fonction d'activation entre la couche $l-1$ et la couche l. Si f est impaire, alors $\mathbb{E}\left[x^l\right] = \mathbb{E}\left[f\left(z^{l-1}\right)\right] = 0$ donc $\mathbb{E}\left[\left(x_j^l\right)^2\right] = \mathbb{V}\left[z_k^{l-1}\right]$. Ainsi,

$$
\begin{aligned}
\forall l \leqslant L, \forall k \in N_l, \ \mathbb{V}\left[z_k^l\right] &= n_{\text{input}} \mathbb{V}\left[w_{k,j}^l\right] \mathbb{V}\left[z_k^{l-1}\right] \\
&= \prod_{i=1}^{l-1} \left(n_{\text{input}} \mathbb{V}\left[w_{k,j}^l\right]\right) \mathbb{V}\left[z_k^1\right]
\end{aligned}
$$

Une condition suffisante pour que l'on respecte l'équation (1.1) est que :

$$
\mathbb{V}\left[w_{k,j}^l\right] = \frac{1}{n_{\text{input}}}
$$

La condition précédente a été obtenue en considérant une passe forward avec une fonction d'activation impaire. On peut se convaincre que la passe backward abouti à la même forme d'équation sauf que cette fois n_{input} représente le nombre de neurones de la couche inférieure. Nous venons de reproduire le raisonnement de l'article *Understanding the difficulty of training deep feedforward neural networks* [Glorot and Bengio, 2010] écrit par Xavier Glorot et Yoshua Bengio en 2010. Pour résoudre le problème final, choisir la condition qui conviendra pour les deux passes, les auteurs proposent de prendre la moyenne entre les deux conditions. Plus tard, nous ferons références au nombre de neurones de la couche précédente (respectivement suivante) par *fan in* (respectivement *fan out*).

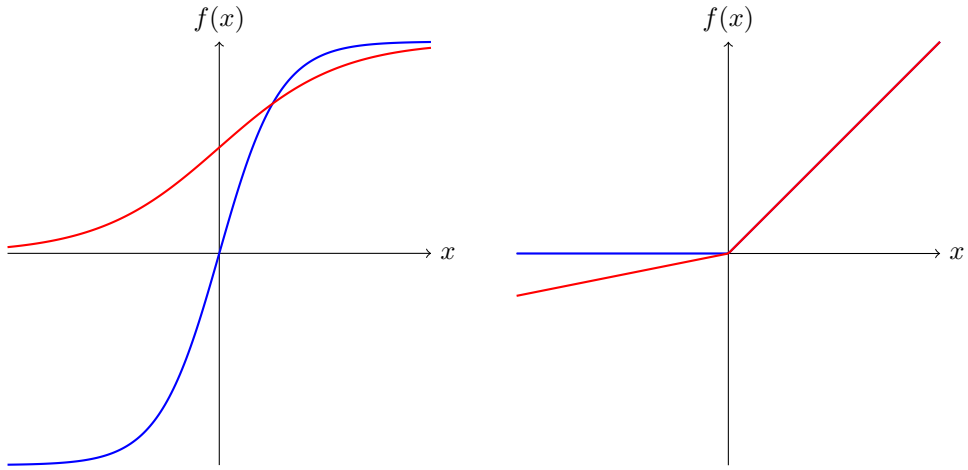
Notons que nous n'avons supposé aucune loi pour l'initialisation des poids à part des conditions d'indépendance, distribution et que la loi doit être symétrique autour de 0 et d'espérance 0.

Exercice 1.2 (Forme uniforme et normale)**.** *On considère un réseau de neurones avec une fonction d'activation impaire. Donner l'expression de la variance en respectant la condition précédente, en supposant que les poids suivent une loi normale puis une loi uniforme.*

Solution. Si les poids sont initialisé selon une loi normale $\mathcal{N}(0, \sigma^2)$: $\sigma = \sqrt{\dfrac{2}{\text{fan in} + \text{fan out}}}$

On rappelle que pour $U \sim \mathcal{U}([-a, a])$ on a $\mathbb{V}[U] = \dfrac{a^2}{3}$. Ainsi, si les poids sont initialisé selon une loi uniforme $\mathcal{U}([-a, a])$: $a = \sqrt{\dfrac{6}{\text{fan in} + \text{fan out}}}$ □

Le résultat précédent s'applique pour des réseaux qui utiliserons des fonctions d'activation impaire comme la fonction sigmoid ou tangente hyperbolique comme présenté à la figure (1.2a).



(a) Fonctions sigmoid et tangente hyperbolique (b) Fonctions ReLU et Leaky ReLU

FIGURE 1.2 – Différentes fonctions d'activations non-linéaire

Cependant, la majorité des réseaux moderne utilisent la fonction d'activation ReLU qui n'est pas impaire, comme visible dans la figure (1.2b). L'article *Delving deep into rectifiers : Surpassing human-level performance on ImageNet classification* [He et al., 2015] publié en 2015 par Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun répond à cette question.

Puisque cette fois f n'est pas impaire mais égale à $f(x) = \max\{0, x\}$, en simplifiant les notations, on a :

$$\begin{aligned}
\mathbb{E}\left[x^2\right] &= \mathbb{E}\left[f(z)^2\right] \\
&= \int_{-\infty}^{+\infty} f(t)^2 p(t)\mathrm{d}t \text{ avec } p \text{ la densité de probabilité associé à } z \\
&= \int_{0}^{+\infty} t^2 p(t)\mathrm{d}t \\
&= \frac{1}{2}\mathbb{V}\left[z\right]
\end{aligned}$$

Nous avons cette fois :

$$\begin{aligned}
\forall l \leqslant L, \forall k \in N_l, \quad \mathbb{V}\left[z_k^l\right] &= \frac{n_{\text{input}}}{2}\mathbb{V}\left[w_{k,j}^l\right]\mathbb{V}\left[z_k^{l-1}\right] \\
&= \prod_{i=1}^{l-1}\left(\frac{n_{\text{input}}}{2}\mathbb{V}\left[w_{k,j}^l\right]\right)\mathbb{V}\left[z_k^1\right]
\end{aligned}$$

A nouveau, une condition suffisante pour que la relation (1.1) soit vérifée pour la passe forward est :

$$\mathbb{V}\left[w_{k,j}^{l}\right] = \frac{2}{n_{\text{input}}}$$

C'est très proche de la condition quand l'on considère une fonction d'activation impaire, mais cela résulte en un écart significatif lors de l'entraînement. Contrairement à la résolution proposée par Xavier Glorot et Yoshua Bengio, la valeur de la variance pour les poids est obtenu en sélectionnant le *fan in* ou le *fan out*.

Exercice 1.3 (Forme uniforme et normale pour ReLU)**.** *On considère un réseau de neurones avec une fonction d'activation ReLU. Donner l'expression de la variance en respectant la condition précédente, en supposant que les poids suivent une loi normale puis une loi uniforme.*

Solution. Si les poids sont initialisé selon une loi normale $\mathcal{N}(0, \sigma^2) : \sigma = \sqrt{\frac{2}{\text{fan mode}}}$ avec *fan mode* valant soit *fan in* si l'on privilégie une conservation maximale des gradients en passe forward, *fan out* pour la passe backward.

Si les poids sont initialisé selon une loi uniforme $\mathcal{U}\left([-a, a]\right) : a = \sqrt{\frac{6}{\text{fan mode}}}$ avec la même définition de *fan mode*. □

Nous savons donc maintenant comment initialiser un réseau de neurones en fonction du choix de la fonction d'activation. Il nous reste à connaître une autre possibilité que l'on rencontre en pratique : le Leaky ReLU.

Exercice 1.4 (Forme uniforme et normale pour le Leaky ReLU)**.** *On considère un réseau de neurones avec une fonction d'activation Leaky ReLU de paramètre $\alpha \in [0, 1]$. Donner l'expression de la variance en respectant la condition précédente, en supposant que les poids suivent une loi normale puis une loi uniforme.*

Solution. En reprenant la démarche pour la fonction ReLU ainsi que les notations, nous avons :

$$\begin{aligned}
\mathbb{E}\left[x^2\right] &= \mathbb{E}\left[f(z)^2\right] \\
&= \int_{-\infty}^{+\infty} \max\left\{\alpha t, t\right\}^2 p(t)\mathrm{d}t \\
&= \int_{-\infty}^{0} \alpha^2 t^2 p(t)\mathrm{d}t + \int_{0}^{+\infty} t^2 p(t)\mathrm{d}t \\
&= \frac{\alpha^2 + 1}{2}\mathbb{V}\left[z\right]
\end{aligned}$$

Ainsi, si les poids sont initialisé selon une loi normale $\mathcal{N}(0, \sigma^2) : \sigma = \sqrt{\frac{2}{(1+\alpha^2) \times \text{fan mode}}}$ avec *fan mode* valant soit *fan in* si l'on privilégie une conservation maximale des gradients en

passe forward, *fan out* pour la passe backward.

Si les poids sont initialisé selon une loi uniforme $\mathcal{U}([-a, a]) : a = \sqrt{\frac{6}{(1+\alpha^2) \times \text{fan mode}}}$ avec la même définition de *fan mode*. □

Notons une certaine similarité entre le cas Leaky ReLU et ReLU : c'est capturé par la documentation PyTorch sous le concepts de *gains*. Pour chaque fonction d'activation est associé une valeur que l'on multiplie à l'expression de la variance pour la stratégie d'initialisation des poids que l'on choisit.

A travers ces problèmes et ses résolutions , nous comprenons l'importance d'une bonne initialisation des poids et donc d'un bon départ dans un réseau de neurone. Essayons de mesurer l'importance du début d'entraînement d'un réseau de neurone.

1.2 Un mauvais départ n'est pas rattrapé

Intuitivement, puisque un réseau peut être amené à s'entraîner longtemps, si l'on lui permet de propager correctement les gradients, il semblerait possible que l'algorithme puisse performer *quoi qu'il arrive*. C'est ce qu'explore l'article *Critical learning periods in deep neural networks* [Achille et al., 2017] publié en 2017 par Alessandro Achille, Matteo Rovere et Stefano Soatto.

Dans les années 1950, une des premières pistes suivi dans le domaine de l'intelligence artificielle était de s'inspirer du vivant. Plus particulièrement : inspirons-nous du fonctionnement du cerveau humain pour construire une machine capable elle aussi d'apprendre. Pour les raisons techniques et théoriques, comme évoqué plus tôt, les réseaux de neurones sont abandonnés pendant plusieurs décennies avant de devenir incontournable à partir des années 2010.

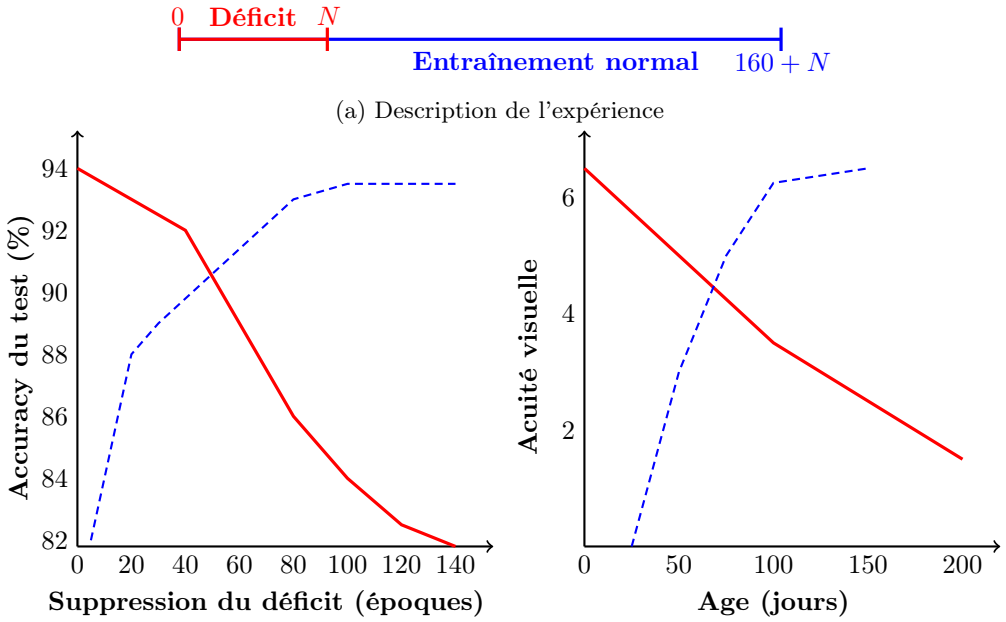
Le cerveau dans le monde animal exhibe des périodes d'apprentissage critique, notamment à la naissance. Plusieurs études se sont intéressées au sujet dont la mesure l'acuité visuelle des chatons en 1963. Cette étude est adapté au réseau de neurones dans cet article. Que se passerait-il si les premières époques d'apprentissage du réseau de neurones étaient réalisées avec des images floutées, tordues ou incomplète ? Il est montré qu'un exposition trop longue à ces images affecte à long terme les chats, qu'en est-il des réseaux de neurones ?

Nous savons que les enfants apprennent vite, et les adultes moins, grâce au concept de la plasticité cérébrale. Peut-on observer la même chose dans un réseau de neurones ? On peut reprendre notre intuition du début : puisque l'algorithme peut s'entraîner longtemps, il n'y a pas de raison qu'il ne puisse pas exhiber de plasticité, de plus sans être affecté par *l'âge*.

L'ensemble des résultats présentés dans cette section sont issu de l'article [Achille et al., 2017] mais certains on été reproduit à la main pour ces notes de cours pour des raisons de lisibilités.

La première expérience réalisée consiste à entraîner un réseau de neurones avec des images floues pendant N époques puis l'entraîner avec les images originelle pendant 160 époques supplémentaires comme décrit par la figure (1.3a). Pour comparer, on entraîne un réseau de neurones sans déficit

et on montre les performances pour chaque époques jusqu'à 140.
Plus le déficit sera supprimé tard, plus le réseau de neurones se sera entraîné longtemps : on peut imaginer que la performance finale sera donc meilleure. Notamment le réseau qui aura été entraîné avec le déficit pendant 140 époques : il aura 160 époques d'entraînement *en clair* de plus.



(a) Description de l'expérience

(b) Réseau de neurones

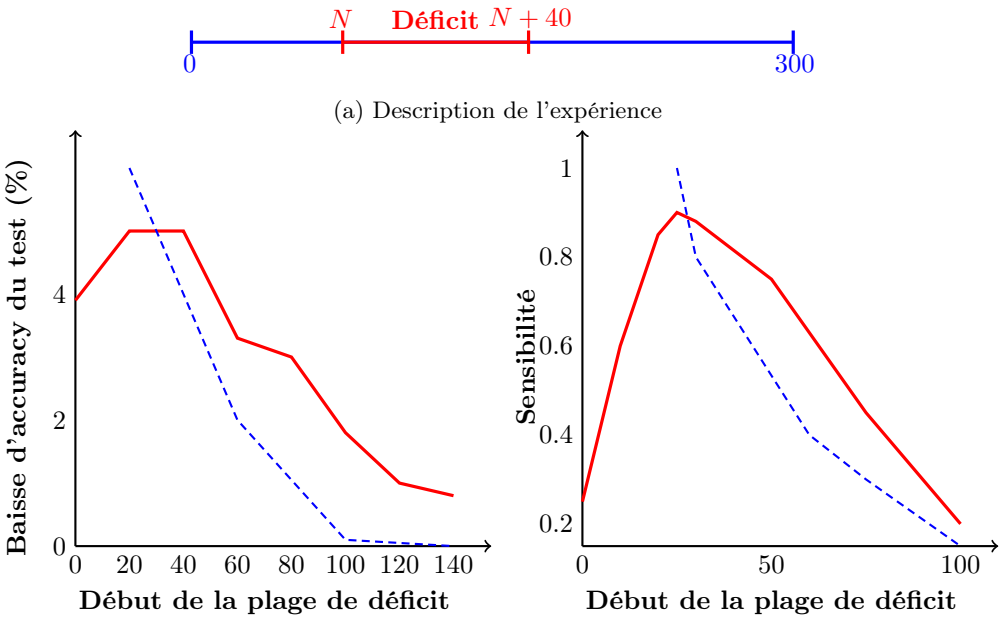
(c) Chaton [Giffin and Mitchell, 1978, Mitchell, 1988]

FIGURE 1.3 – Performance/acuité selon la durée du déficit par rapport à un développement normal

On observe l'inverse ! Si le déficit est supprimé *tôt* alors on peut obtenir une performance proche d'un entraînement sans déficit. Cependant, si le déficit dure, le temps d'entraînement supplémentaire ne semble pas permettre de rattraper ce retard.
Ce comportement est identique à celui décrit dans une expérience visuelle réalisée sur des chatons. On comprend donc que la période de départ est importante, portant l'emphase sur les premières époques.

Une deuxième expérience est réalisée en sélectionnant cette fois une fenêtre de taille fixe pour l'entraînement avec du déficit. Le nombre d'époque total est également fixe, c'est le début du déficit qui change cette fois, comme décrit dans (**??**). Avec l'expérience précédente, on conjecture que lorsque le déficit portera sur les premières époques il y aura un grand écart par rapport à un entraînement sain sur la performance sur le jeu de test. Mais qu'en est-il lorsque le déficit est présent en *fin* d'entraînement ?

Nous retrouvons bien l'importance du début d'entraînement avec une baisse jusqu'à 5% de la performance sur le jeu de test dans le déficit est appliqué sur les premières époques. Notons tout de même que lorsque le déficit est appliqué sur les toutes premières époques, la sensibilité est plus faible que lorsqu'il s'agit des époques 20/40. On observe également que lorsque le déficit est



(a) Description de l'expérience

(b) Réseau de neurones

(c) Chaton [Olson and Freeman, 1980]

FIGURE 1.4 – Sensibilité selon le début du déficit par rapport à un développement normal

porté sur les dernières époques, l'impact est beaucoup plus faible, mais présent.
Les conclusions sont identiques pour les chatons à nouveau. Ce qui montre à nouveau l'importance du début d'entraînement : un mauvais départ n'est pas rattrapable, même avec des époques supplémentaires. Qu'est-ce qui peut expliquer ce phénomène ? Pour répondre à cette question, l'article propose de mesurer l'apprentissage du réseau de neurones en étudiant la force de connexion entre les neurones.

1.2.1 Mesurer la force de connexion

On représente le résultat d'un réseau de neurones comme $\mathbb{P}_w(y \mid x)$ où w représente l'ensemble des poids qui paramètre le réseau de neurones. Si l'on note $w' = w + \delta w$ où δ représente une petite perturbation, alors on peut mesurer l'impact des poids modifiés par la perturbation dans le réseau de neurones comme *l'écart* entre $\mathbb{P}_w(y \mid x)$ et $\mathbb{P}_{w'}(y \mid x)$. Mais comment mesurer *l'écart* entre deux distributions ?

Cette question rejoint la théorie de l'information et plus particulièrement la divergence de Kullback-Leibler :

Distribution de densité p

$$D_{\mathrm{KL}}\left(P \,\|\, Q\right) = \int_{-\infty}^{+\infty} p(x) \ln\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x \qquad \text{(Kullack-Leibler)}$$

Distribution de densité q

Cette divergence mesure ce que l'on peut espérer perdre en remplaçant la vrai distribution par une distribution estimée. Comme son nom l'indique, la divergence de Kullback-Leibler est une divergence : donc elle n'est pas symétrique. Puisque c'est une divergence, pour toutes distributions P et Q, la divergence de Kullback-Leibler doit être positive : ce n'est pas évident tel quel. Le prouver est un exercice classique d'analyse présenté dans l'exercice (1.5).

Exercice 1.5 (Inégalité de Gibbs discrète)**.** *On considère* $P = \{p_1, \dots, p_n\}$ *et* $Q = \{q_1, \dots, q_n\}$ *deux distributions discrètes.*

1. *Montrer que :* $\forall x > 0, \ln x \leqslant x - 1$ *et que l'égalité est vérifié si, et seulement si,* x*=1.*
2. *Soit* $I = \{i \mid p_i \neq 0\}$*. En utilisant le résultat précédent, montrer que :*

$$-\sum_{i \in I} p_i \ln\left(\frac{q_i}{p_i}\right) \geqslant 0$$

3. *On rappelle que* $\lim_{x \to 0} x \ln(x) = 0$*. Montrer que :*

$$-\sum_{i=1}^{n} p_i \ln(q_i) \geqslant -\sum_{i=1}^{n} p_i \ln(p_i)$$

4. *Sous quelle condition y a-t-il égalité ?*
5. *Montrer donc que* $D_{KL}(P \| Q) \geqslant 0$

Puisque nous nous intéressons ici à des réseaux de neurones avec énormément de paramètre, l'estimation de la divergence de Kullback-Leibler va être difficile en l'état. Notons que puisque $\lim_{x \to 0} x \ln(x) = 0$, nous n'avons pas de problème pour des événements impossible pour P mais possible pour Q. Cependant l'inverse n'est pas vrai : nous aurons donc également un problème de définition puisque la divergence vaut, par convention, $+\infty$. En introduisant d'infimes quantités ε à chaque événements (dont ceux impossible pour l'une ou l'autre des distributions) nous sommes capables de calculer la divergence au prix d'un léger écart.
Cela ne résout pas notre problème de capacité à estimer la mesure pour un grand réseau de neurones. Il nous faudrait une expression plus simple pour la calculer.

On peut montrer, mais ce n'est pas l'objet de ce cours, que le développement de Taylor à l'ordre deux de la divergence de Kullback-Leibler s'écrit sous la forme suivante :

$$D_K L(\mathbb{P}_w(y \mid x) \| \mathbb{P}_{w'}(y \mid x)) = \delta w F(\delta w) + o\left(\delta w^2\right)$$

Avec F la métrique d'information de Fisher. On peut la comprendre dans ce contexte comme une métrique **locale** de la pertubation de la valeur d'un poids. Concrètement, si un poids a un information de Fisher faible c'est que le poids n'est pas très important pour la sortie du réseau de neurone.
À nouveau la matrice d'information de Fisher est trop grande pour être calculé systématiquement. Ce que l'article propose finalement, est de ne considérer que la force de connexion entre chaque couche : ne considérer que la trace de la matrice d'information de Fisher.

Nous sommes donc maintenant capables d'approcher, localement, la force de la connexion

entre les couches d'un réseau de neurone. Notons que puisque la mesure est approchée et largement imparfaite pour des raisons de puissance de calcul, ce sont des réseaux de neurones résiduels qui ont été utilisé : ils sont connu pour avoir des propriétés plus lisse que les autres type de réseau de neurones :

- [Balduzzi et al., 2017] montre que les ResNet ont des gradients plus lisses
- [Li et al., 2018] montre que les ResNet ont une surface de loss plus lisse également
- [Zhang et al., 2019] montre que la particularité des ResNet, les *skip connections*, peuvent être remplacé par une bonne initialisation et tout de même obtenir des résultats compétitif

Voyons comment cette notion peut nous aider à comprendre l'importance d'un bon départ. Puisque l'on peut interpréter la trace de la matrice de Fisher comme la quantité d'information *apprise* par le modèle, on devrait voir une augmentation continue au fur et à mesure de l'entraînement. Mais peut-être moins quand il y a un déficit.

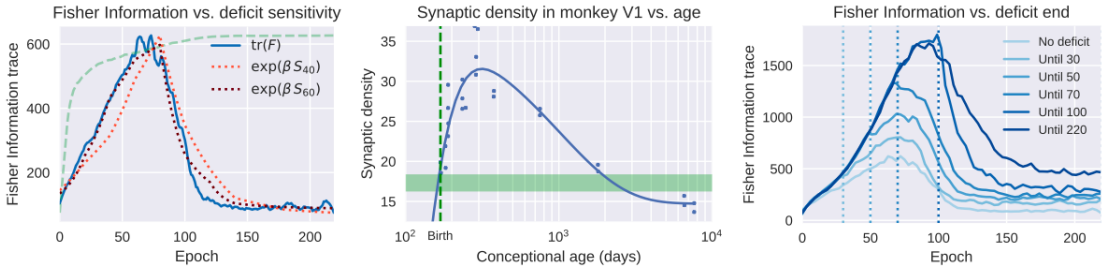


FIGURE 1.5 – Evolutions de la trace de la matrice d'information de Fisher (source : [Achille et al., 2017])

Dans le graphique de **gauche** est tracé en pointillé vert la performance du réseau sur le dataset de test. La courbe bleu correspond à la trace de la matrice d'information de Fisher. On observe deux phases : la trace augmente jusqu'à ce que la performance sur le jeu de test stagne puis s'écroule. On remarque que ce phénomène est commun peut importe qu'il y ait un déficit ou non (graphique *droit*). Cependant, on observe que plus le déficit est maintenu longtemps plus la quantité d'informations stockée est importante et que la phase de *stockage* d'information dure plus longtemps.
On a donc deux phases d'entraînement : une première où le réseau *apprend* et améliore ses performances sur le jeu de test, et une second de compression d'informations où les connexions *inutiles* sont supprimées. Remarquons que lorsque les images sont trop difficile pour être classifié à cause du déficit introduit, le réseau est forcé de mémoriser des observations.
Voyons comment les différentes couches participent à ce phénomène.

Pour un entraînement normal, le réseau semble *apprendre* avec les couches du milieu plutôt. Ce n'est pas le cas dès que l'on introduit un déficit : les dernières couches sont les plus mises à contributions et stocke énormément d'informations. Ce phénomène semble proportionnel au temps de maintient du définit.
La puissance de représentation du réseau de neurones n'est pas utilisé puisque les features générées naturellement par les couches intermédiaires n'ont pas lieu : le réseau est forcé de mémoriser les images. Une manière de s'assurer qu'un réseau ne *mémorise* pas les input est de le régulariser. Observons l'impact du *weight decay*[1].

1. La notion est présentée plus en détail dans la section (3.2.1)

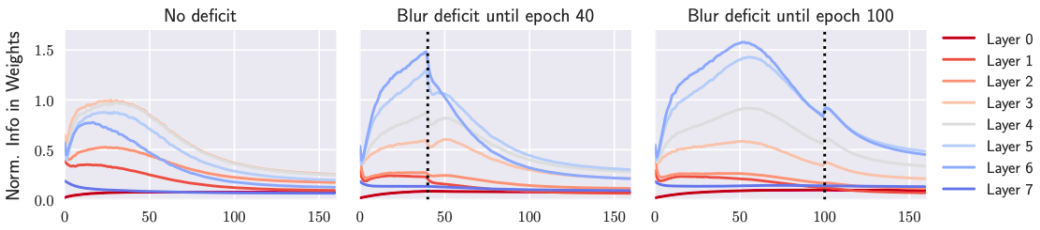


FIGURE 1.6 – Evolutions de la trace de la matrice d'information de Fisher par couche (source : [Achille et al., 2017])

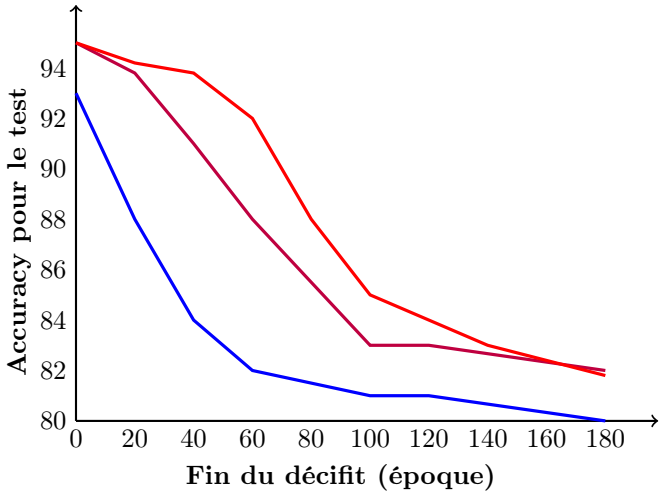


FIGURE 1.7 – Performance sur le dataset de test en fonction du nombre d'époque de maintien du déficit pour un weight decay de $\lambda = 0$, $\lambda = 5 \cdot 10^{-4}$ et $\lambda = 10 \cdot 10^{-4}$

Conformément à notre intuition, la régularisation rend la période critique du début d'entraînement un peu moins sensible au déficit. Cependant, si le déficit est maintenu trop longtemps on observe tout de même une baisse nette de performance.

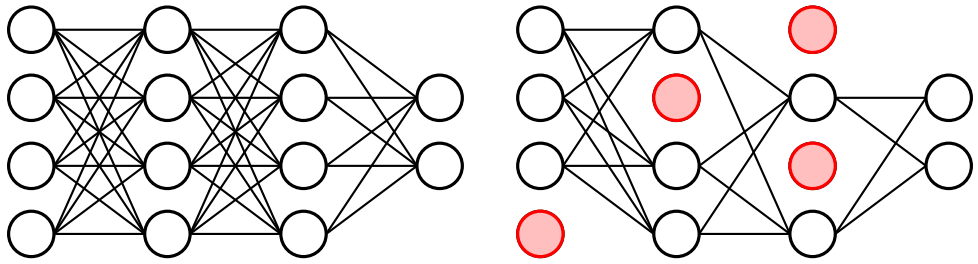
Ainsi, de manière surprenante nous comprenons qu'un mauvais départ n'est pas rattrapé et qu'en **contraignant** le réseau de neurones nous pouvons ralentir ce phénomène si le départ n'est pas très bon.

1.3 Bien se lancer pour mieux généraliser

Cette idée rejoins celle de Zhuang Liu, Zhiqiu XXu, Joseph Jin, Zhiqiang Shen et Trevor Darell développé dans l'article *Dropout Reduces Underfitting*[Liu et al., 2023] en 2023. Le dropout [Hinton et al., 2012b] et [Srivastava et al., 2014] est introduit en 2012 par Geoffrey Hinton Nitish Srivastava, Alex krizhevsky, Ilya Sutskever et Ruslan Salakhutinov. Cette méthode de régularisation est rapidement devenu une brique essentielle des réseaux de neurones. Un des facteurs du succès immédiat de la méthode est son utilisation dans le modèle AlexNet [Krizhevsky et al., 2012],

vainqueur de la compétition ImageNet en 2012.

Lors de l'entraînement, le dropout va *supprimer* de la chaîne d'entraînement des neurones avec une proportion $1 - p$ à chaque couche. Comme illustré avec la figure 1.8, le dropout s'applique à chaque couches, dont l'input, sauf l'output. On obtient donc à chaque passe forward un sous-ensemble du réseau initial.



(a) Entraînement classique (b) Entraînement avec dropout

FIGURE 1.8 – Réseau de neurones avec et sans dropout

Lors de l'exploitation du réseau de neurones en revanche nous ne pouvons pas avoir du hasard, il faut un comportement déterministe. Remarquons que ce que nous venons de décrire ressemble à du bagging ! Nous avons entraîné tout un ensemble de réseaux de neurones, tous différents (ou presque). Cependant, ils ne sont pas indépendant car ils partagent des poids. Nous devons trouver une manière d'exploiter la force des méthodes ensemblistes.
Classiquement, la prédiction finale d'un ensemble est construite en faisant la moyenne arithmétique des prédictions. Ici, cela coûterait trop en mémoire de conserver l'ensemble des réseaux, donc nous allons approcher ce comportement en faisant la moyenne géométrique. Pour approcher cela, chaque poids est multipliés par la proportion p de dropout.
Le dropout permet d'éviter une sur-spécifications de certain neurones, forçant le réseau à être plus redondant et robuste. De la même manière que chaque membres d'un équipe en entreprise doit être capable de se remplacer en cas de congé. L'intuition est la même pour le dropout.

L'amélioration quasi systématique des performances d'un réseau de neurones a rendu cette méthode très populaire mais pas incontournable. Dans l'entraînement des LLM par exemple, aucun dropout n'est ajouté. Cependant, il est fortement conseillé lors d'un fine-tuning. Son utilisation est principalement pour éviter le sur-apprentissage. Il est donc surprenant que le titre du papier [Liu et al., 2023] soit *Dropout Reduces Underfitting*. L'article propose de n'appliquer le dropout que lors du début de l'entraînement : cela s'appelle *l'early dropout*. En comparant à une baseline, il est noté que la norme des gradients lors de la phase d'early dropout est plus petite que ceux de la baseline. Ainsi, il est probable que la distance parcourue dans l'espace des paramètres soient également plus faible. Cependant, c'est l'inverse qui se produit !
Nous en déduisons que le dropout force le réseau à prendre une direction plus consistante que la baseline originale. C'est confirmé lorsqu'on calcule la variance dans la direction entre le réseau avec early dropout et la baseline. Finalement, contraindre le réseau de neurones permet de le placer dans de meilleurs conditions pour la suite de l'entraînement. On observe en effet que la réduction de loss est inférieur tant que le dropout est effectif, et cela s'inverse lorsque le dropout s'arrête.

On peut se poser la question de la régularisation en fin d'entraînement. Est donc introduit le *late dropout*. Il est montré que cette version n'est pas très efficace pour régulariser le réseau de neurones. Cependant la méthode *stochastic depth* l'est.

Cette méthode est introduite en 2016 [Huang et al., 2016] et ne s'applique qu'au réseau de neurones résiduels (ResNet) introduite un an plus tôt [2] [He et al., 2016].Les réseaux de neurones résiduels sont constitués de nombreux ResBlock dont la constitution est illustrée avec la figure (1.9).

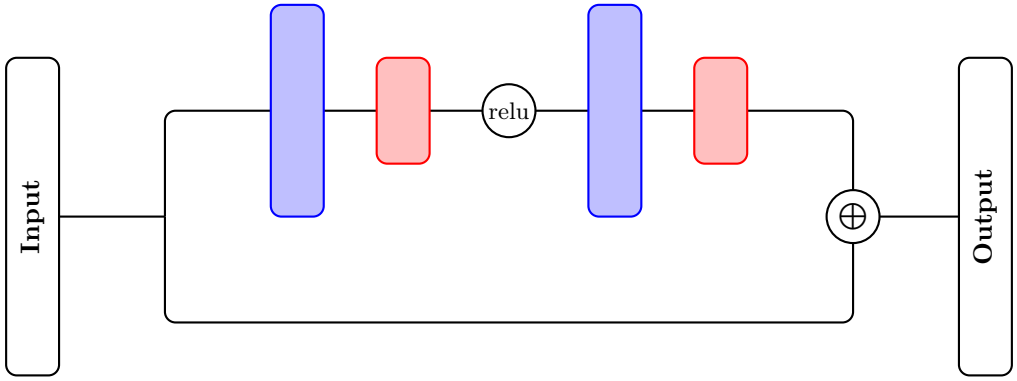


FIGURE 1.9 – ResBlock dans un ResNet avec des couches de convolutions et de Batch Normalisation

La partie *haute* correspond à une transformation de l'input tandis que la partie *basse* permet de ramener identiquement l'input pour former l'output du ResBlock. Nous étudions la couche de Batch Normalisation plus en détail dans la section 3.3
L'idée de la méthode stochastic depth est de ne pas systématiquement réaliser la partie *haute* d'un ResBlock. Il est proposé dans l'article d'associer une probabilité d'évitement du ResBlock de manière linéaire entre le premier ResBlock et le dernier, plutôt que ça soit la même valeur pour tous les blocks, comme dans le dropout.
Cette méthode est unique pour les ResNet, et il n'existe pas d'équivalent [3] pour les autres types de réseaux de neurones. On peut noter que GATO de Deepmind [Reed et al., 2022] utilise cette brique.

L'article exploite de nombreux outils que l'on définira dans les séances suivantes, et ne traite que des Vision Transformers. Nous verrons en TP si cela reste vrai pour d'autres types de réseaux de neurones [4]. Ce que l'on peut retenir de cet article, s'il est vérifié dans d'autres cas d'usage, est que le dropout devrait plutôt être utilisé comme régularisation du réseau de neurones en début d'entraînement. Si l'on travaille avec un ResNet il peut être intéressant d'ajouter la stochastic depth comme mesure de régularisation générale. Dans les deux cas, on peut voir ces méthodes comme des manières d'entraîner une multitude de réseaux de neurones de sorte à créer de la redondance et donc plus de robustesse. Ces deux aspects induisent une régularisation globale du réseau.

2. Cette méthode remporte la compétition ImageNet de 2015.
3. A notre connaissance.
4. Spoiler : Oui, mais il faut bien respecter les hyperparamètres d'architecture de l'article.

Nous avons montré dans cette séance l'importance d'un début d'entraînement. Il conditionne la bonne convergence du réseau à travers une bonne propagation des informations. Nous avons également montré que contraindre le réseau de neurones en début d'entraînement peut le placer dans les meilleurs conditions pour atteindre un meilleur optimum. L'importance de cette partie de l'entraînement est telle, que la baisse de performance induite par un mauvais départ ne sera pas rattrapé.

> *Rien ne sert de courir ; il faut partir à point*
> — Jean de la Fontaine (1668)

Annexes

Annexe B

Convexité : rappel et dualité avec les fonctions de perte bien définies

B.1 Définition et propriétés

B.1.1 Ensemble et fonction convexe

Dans l'ensemble de la section on travaillera toujours en dimension $d \in \mathbb{N}^*$. Commençons par définir ce que l'on appelle un ensemble convexe :

Définition B.1. *Soit A un sous-ensemble de $\mathbb{R}^d$. On dit que A est un ensemble convexe si :*

$$\forall a, b \in A, \forall t \in [0,1],\ ta + (1-t)b \in A$$

On appelle donc un ensemble convexe un ensemble dans lequel on peut relier deux points linéairement avec uniquement des points de l'ensemble. On peut illustrer cette définition avec la figure (B.1).

(a) Ensemble convexe (b) Ensemble non convexe

FIGURE B.1 – Exemple et contre exemple d'ensemble convexe

L'ensemble bleu répond parfaitement à l'exigence de la définition : chaque point de l'ensemble est joignable linéairement avec uniquement des points de l'ensemble. L'ensemble rouge n'est pas convexe parce qu'entre a et b le *chemin* entre les deux points n'est pas entièrement contenu dans

l'ensemble.

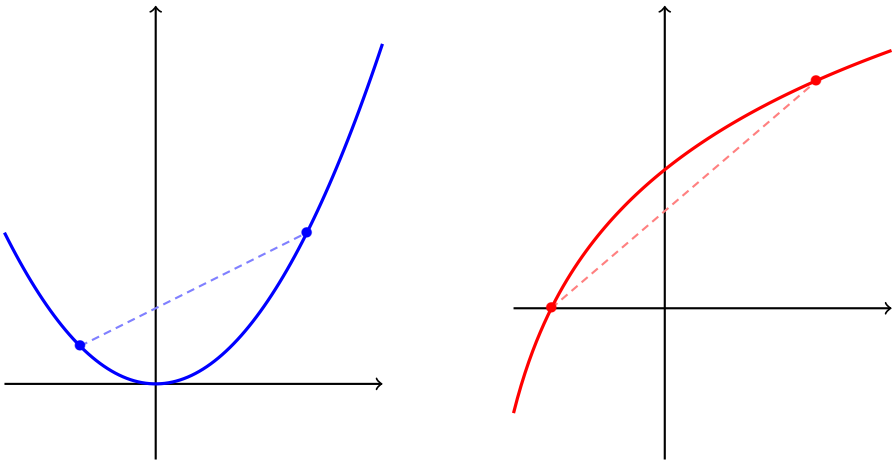
Avec cette notion, nous sommes capable de définir une fonction convexe :

Définition B.2. *Soit A un sous-ensemble convexe de $\mathbb{R}^d$. Une fonction $f : A \mapsto \mathbb{R}$ est convexe si et seulement si :*

$$\forall a, b \in A, \forall t \in [0,1],\ f\left(ta + (1-t)b\right) \leqslant tf(a) + (1-t)f(b)$$

On dira que f est strictement convexe si l'inégalité est stricte.

La définition de fonction convexe ressemble à la définition d'un ensemble convexe. Mais la différence réside dans l'utilisation d'une inégalité ici, et que l'on traite avec les images des points de l'ensemble convexe A. A nouveau, on peut visualiser cette définition avec la figure (B.2).



(a) Fonction convexe (b) Fonction non convexe

FIGURE B.2 – Exemple et contre exemple de fonction convexe

Exercice B.1. *Donner des exemples de fonctions qui répondent aux critères suivants.*

1. *Une fonction strictement convexe.*
2. *Une fonction convexe qui n'est pas strictement convexe.*
3. *Une fonction convexe qui n'est pas strictement convexe ni affine.*

Solution. On propose ici une possibilité, mais il y en a bien sur beaucoup plus.

1. La fonction $x \mapsto x^4$ est strictement convexe.
2. N'importe quelle fonction affine est convexe mais pas strictement convexe.
3. La fonction $x \mapsto \max\{0, x\}$ est convexe mais pas strictement convexe ni affine.

□

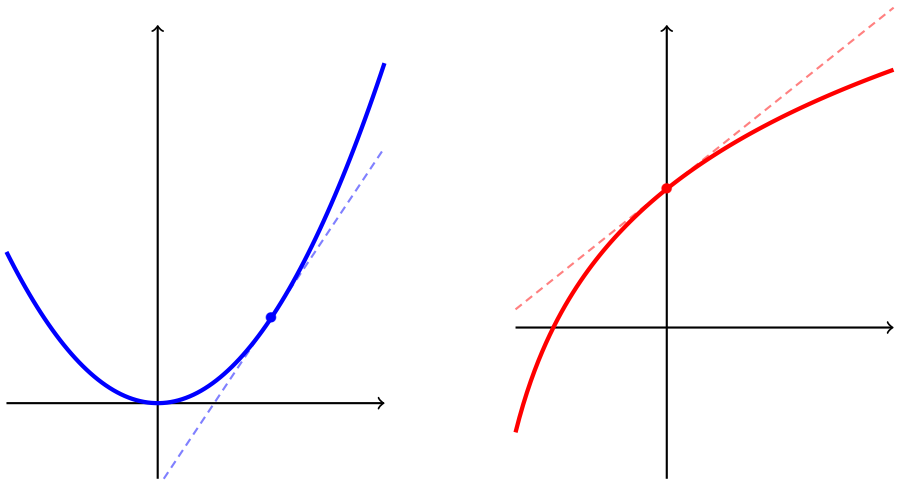
B.1.2 Caractérisation du premier et deuxième ordre

Il peut parfois être difficile de prouver qu'une fonction est convexe avec la définition que l'on vient de donner. Il nous faudrait une caractérisation plus simple d'utilisation :

Théorème B.1 (Caractérisation des fonctions convexes)**.** *Soit A un sous-ensemble convexe de $\mathbb{R}^d$ et soit $f : A \mapsto \mathbb{R}$ deux fois différentiable. Alors les propriétés suivantes sont équivalentes :*

1. *f est convexe*
2. $\forall x, y \in A,\ f(y) \geqslant f(x) + \nabla\langle f(x), y - x\rangle$
3. $\forall x \in A,\ \nabla^2 f(x) \succeq 0$

On sait déjà ce que la première propriété veut dire, il nous reste à comprendre les deux suivantes.
Dans la deuxième condition, on reconnait un développement de Taylor à l'ordre 1, ce qui nous dit que chaque tangente de la fonction f est un sous-estimateur global. On peut le visualiser avec deux exemples dans la figure (B.3).



(a) Fonction convexe (b) Fonction non convexe

FIGURE B.3 – Illustration de la propriété de sous-estimateur pour une fonction convexe et contre exemple pour une fonction non convexe

La troisième propriété veut dire qu'il n'y a pas de courbure négative dans la courbe de la fonction f. Autrement dit, que la dérivée de la fonction f est croissante. En dimension une, cela veut dire que la dérivée seconde est toujours positive ou nulle.
Il s'agit maintenant de prouver le théorème (B.1)

Démonstration. Commençons par montrer que si f est convexe, alors $\forall x, y \in A,\ f(y) \geqslant f(x) + \nabla f(x)(y - x)$.

Soit $x, y \in A$, par définition on a que :

$$\begin{aligned} \forall t \in [0,1],\ f(tx + (1-t)y) &\leqslant tf(x) + (1-t)f(y) \\ \forall t \in [0,1],\ f(x + t(y-x)) &\leqslant f(x) + t(f(x) - f(y)) \\ \forall t \in [0,1],\ f(y) - f(x) &\geqslant \frac{f(x + t(y-x)) - f(x)}{t} \end{aligned}$$

Ainsi, en prenant la limite pour $t \downarrow 0$, on a que :

$$\forall x, y \in A,\ f(y) \geqslant f(x) + \nabla f(x)(y - x)$$

D'où le résultat souhaité. Montrons maintenant que si on a le résultat précédent, alors f est convexe. Soit $x, y \in A$ et on définit $z = tx + (1-t)y$. Par la propriété que l'on suppose, on a :

$$\begin{aligned} f(x) &\geqslant f(z) + \nabla f(z)(x - z) \\ f(y) &\geqslant f(z) + \nabla f(z)(y - z) \end{aligned}$$

En multipliant la première équation par $t \in [0,1]$ et la seconde équation par $(1-t)$, on obtient :

$$\begin{aligned} tf(x) + (1-t)f(y) &\geqslant f(z) + \nabla f(z)(tx + (1-t)y - z) \\ &= f(z) \\ f(tx + (1-t)y) &\leqslant tf(x) + (1-t)f(y) \end{aligned}$$

On a donc montré que les deux premières propositions sont équivalentes. Montrons maintenant que les deux dernières propriétés sont équivalentes en dimension 1 pour simplifier les calculs.

Supposons que $\forall x \in A,\ f''(x) \geqslant 0$, alors par le théorème de la valeur moyenne de Taylor on a que :

$$\begin{aligned} \exists z \in [x, y], \qquad f(y) &= f(x) + f'(x)(y - x) + \frac{1}{2} f''(z)(y - x)^2 \\ \Rightarrow\quad f(y) &\geqslant f(x) + f'(x)(y - x) \end{aligned}$$

Finalement, supposons que $\forall x, y \in A,\ f(y) \geqslant f(x) + \nabla f(x)(y - x)$. Soit $x, y \in A$ tels que $y > x$. On a que :

$$\begin{aligned} f(y) &\geqslant f(x) + f'(x)(y - x) \\ f(x) &\geqslant f(y) + f'(y)(x - y) \end{aligned}$$

On en déduit donc que :

$$f'(x)(y - x) \leqslant f(y) - f(x) \leqslant f'(y)(y - x)$$

Donc en divisant par $(y - x)^2$ puis en prenant la limite pour $y \to x$, on obtient bien que la propriété souhaité. □

Ce résultat conclut notre section de présentation des notions de convexité. Intéressons-nous maintenant à l'utilisation de cette notion pour l'optimisation.

B.2 Résultats d'optimisations

On considère un problème d'optimisation sans contraintes avec une fonction $f : \mathbb{R}^d \mapsto \mathbb{R}$ différentiable :

$$x^* = \arg\min_{x \in \mathbb{R}^d} f(x) \tag{B.1}$$

Notons qu'ici nous n'avons pas encore spécifié que f est convexe. Dans le cadre général, on sait qu'une condition nécessaire pour que x soit une solution de ce problème est que $\nabla f(x) = 0$. Mais ce n'est pas une condition suffisante ! De plus, si cette condition nécessaire est vraie, et qu'il s'agit d'un minimum, alors on ne peut pas dire plus que "x est un minimum local" avec ces informations.

Exercice B.2. *Donner un exemple de fonction qui répond à chaque critère :*

1. *Une fonction où il existe $x \in \mathbb{R}$ tel que $f'(x) = 0$ et que x est un minimum local.*
2. *Une fonction où il existe $x \in \mathbb{R}$ tel que $f'(x) = 0$ mais que x n'est ni un minimum local ni un maximum local.*
3. *Une fonction où il existe une infinité de $x \in \mathbb{R}$ tel que $f'(x) = 0$ qui sont tous des minimum locaux.*

Solution. On propose ici une possibilité, mais il y en a bien sûr beaucoup plus.

1. La fonction $x \mapsto x^3 - x$ possède un minimum local (et un maximum local).
2. La fonction $x \mapsto x^3$ en $x = 0$.
3. La fonction $x \mapsto \cos(x)$ contient une infinité de point x tel que $f'(x) = 0$ (tous espacés de π) mais seulement *la moitié* sont des minimums.

□

On voit donc que nous n'avons pas de critères simples et clairs dans le cas général sur l'existence et l'unicité d'un minimum global. Voyons ce qu'il en est quand la fonction f est convexe.

Proposition B.1. *Soit un problème d'optimisation sans contraintes comme présenté dans (B.1) avec f une fonction convexe et différentiable. Alors, chaque point x qui vérifie $\nabla f(x) = 0$ est un minimum global.*

Pour une fonction convexe différentiable, la condition $\nabla f(x) = 0$ est une condition nécessaire et suffisante pour caractériser un minimum global.

Exercice B.3. *Prouver la proposition (B.1) à l'aide du théorème (B.1).*

Solution. Comme $f : A \mapsto \mathbb{R}$ est convexe et différentable, d'après le théorème (B.1) on a :

$$\forall x, y \in A,\ f(y) \geqslant f(x) + \nabla f(x)(y - x)$$

Donc en particulier pour $\overline{x}$ défini comme $\nabla f(\overline{x}) = 0$:

$$\begin{aligned} \forall y \in A,\ f(y) &\geqslant f(\overline{x}) + \nabla f(\overline{x})(y - \overline{x}) \\ \forall y \in A,\ f(y) &\geqslant f(\overline{x}) \end{aligned}$$

Qui est bien la définition d'un minimum global d'une fonction. □

Mais nous n'avons toujours pas l'unicité d'un minimum à ce stade. Pour l'obtenir, nous avons besoin d'avoir la stricte convexité.

Proposition B.2. *Soit un problème d'optimisation sans contraintes comme présenté dans (B.1) avec f une fonction strictement convexe et différentiable. Si $\nabla f(\overline{x}) = 0$, alors $\overline{x}$ est l'unique minimum global de f.*

Nous obtenons cette fois l'unicité à l'aide de la stricte convexité. Voyons comment.

Exercice B.4. *Prouver la proposition (B.2) en raisonnant par l'absurde. On suppose donc qu'il existe deux minimaux globaux et on aboutit à une absurdité en exploitant la stricte convexité.*

Solution. On suppose qu'il existe $a, b \in A$ qui minimisent f tel quel que $a \neq b$. Prenons $z = \dfrac{a+b}{2} \in A$ comme combinaison convexe de deux points du domaine $\left(\text{avec } t = \dfrac{1}{2}\right)$. Alors :

$$f(z) < \frac{1}{2}f(a) + \frac{1}{2}f(b) = f(a) = f(b)$$

On vient de trouver un nouveau nombre z qui minimise encore mieux la fonction f que les deux meilleurs minimiseurs : absurde, d'où l'unicité. □

Avec ces deux derniers résultats, nous comprenons pourquoi il est important de travailler avec des fonctions de perte convexes : elles nous garantissent qu'en suivant une descente de gradient, nous atteindrons bien un minimum global.
Voyons à présent à quelle vitesse.

B.3 Vitesse de convergence pour la descente de gradient

Dans cette section nous allons donner des preuves de vitesse de convergence pour deux hypothèses sur la fonction f.

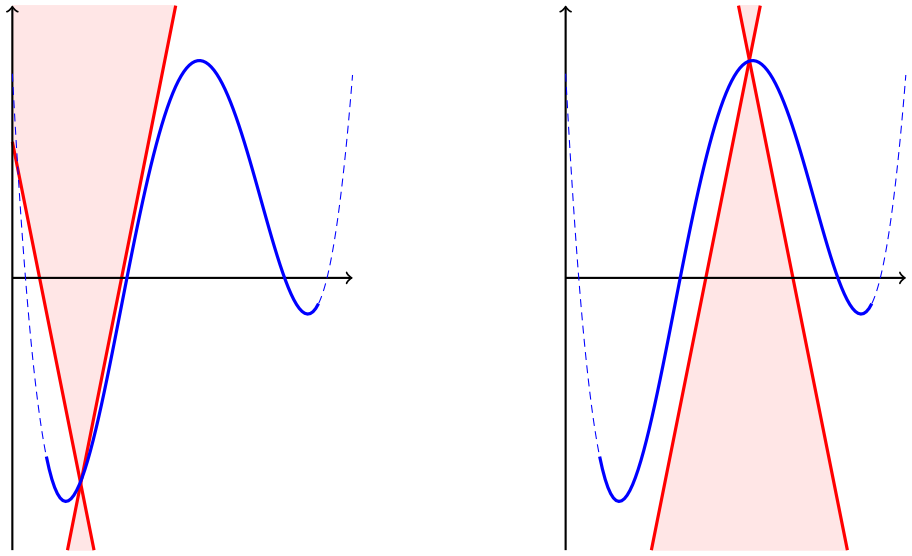
B.3.1 Pour une fonction Lipschitzienne

La plus petite supposition qui nous permet de garantir la convergence vers le minimum global est que la fonction soit Lipschitzienne.

Définition B.3 (Fonction L-Lipschitzienne)**.** *Soit $f : \mathcal{D} \to \mathbb{R}$ une fonction convexe. On dit que f est L-Lipschitzienne si il existe un nombre L tel que :*

$$|f(y) - f(x)| \leqslant L\|y - x\|$$

Pour comprendre visuellement cette définition, on peut s'appuyer sur la figure (B.4).



(a) Condition en $x = 1$ (b) Condition en $x = 2.7$

FIGURE B.4 – Illustration de la condition lipschitzienne pour une fonction

Dire qu'une fonction est L-lipschitzienne revient à dire que ses variations seront toujours hors des cônes rouge, autrement dit on contraint la progression de la fonction. Ici, on remarque que la fonction n'est pas lipschitzienne pour ce L-là sur $[0, 5]$ parce que la fonction passe dans les cônes rouge. En revanche, elle l'est pour l'intervalle $[0.5, 4.5]$. Remarquons également que nous avons une fonction qui peut être lipschitzienne sans être convexe.
Supposer les deux, permet d'avoir le résultat suivant.

Proposition B.3. *Soit $x^* \in \mathbb{R}^d$ le minimum de la fonction $f : \mathbb{R}^d \mapsto \mathbb{R}$ convexe et L-Lipschitzienne. Soit $\varepsilon > 0$ l'erreur que l'on accorde pour arrêter la descente de gradient. On choisit $\eta = \dfrac{\varepsilon}{L^2}$.*
Alors en $T = \dfrac{L^2\|x^* - x_0\|^2}{\varepsilon^2}$ itérations, l'algorithme converge vers x^* avec une erreur de ε.

Nous avons donc la garantie que nous sommes capables de converger vers le minimum de la fonction f avec un nombre d'itérations que l'on maitrise. Si l'on note $\widetilde{x} \in \mathbb{R}^d$ le point que l'on obtient à la suite de la descente de gradient, on a $f(\widetilde{x}) \leqslant f(x^*) + \varepsilon$.

Pour faire la preuve de ce résultat, nous avons besoin d'un résultat intermédiaire.

Lemme B.1. *Pour $(x_t)_{t\in\mathbb{N}}$ la suite définie pour une descente de gradient qui cherche à minimiser une fonction f convexe et L-Lipschitzienne, on a :*

$$\|x_{t+1} - x^*\|^2 \leqslant \|x_t - x^*\|^2 - 2\eta\left(f(x_t) - f(x^*)\right) + \eta^2 L^2$$

Exercice B.5. *Prouver le lemme (B.1).*

Solution.

$$\begin{aligned}\|x_{t+1}-x^*\|^2 &= \|x_t-x^*-\eta\nabla f(x)\|^2\\ &= \|x_t-x^2\|^2-2\eta\left(\nabla f(x)\right)^t(x_t-x^*)+\eta^2\|\nabla(f(x_t))\|^2\\ &\leqslant \|x_t-x^*\|^2-2\eta\left(f(x_t)-f(x^*)\right)+\eta^2L^2\end{aligned}$$

En exploitant le fait que f soit convexe et L-Lipschizienne pour la dernière inégalité. □

Nous avons donc maintenant tous les outils pour démontrer le résultat annoncé dans la proposition (B.3).

Démonstration. Soit $\Phi(t)=\|x_t-x^*\|^2$. Alors avec $\eta=\dfrac{\varepsilon}{L^2}$ et le précédent lemme on a :

$$\Phi(t)-\Phi(t+1)>2\eta\varepsilon-\eta^2L^2=\frac{\varepsilon}{L^2}$$

Puisque par définition, $\Phi(0)=\|x^*-x_0\|$ et $\Phi(t)\geqslant 0$, la précédente équation ne peut pas être vérifiée pour tous $0\leqslant t\leqslant\dfrac{L^2\|x^*-x_0\|^2}{\varepsilon^2}$ d'où le résultat. □

B.3.2 Fonction β-smooth

Une manière d'accélérer cette convergence est de faire plus d'hypothèse sur la fonction en demandant qu'elle soit β-smooth.

Définition B.4 (Fonction β-smooth)**.** *Soit $f:\mathcal{D}\mapsto\mathbb{R}$ une fonction convexe différentiable. On dit que f est β-smooth si :*

$$\forall x,y\in\mathcal{D},\ f(y)\leqslant f(x)+\langle\nabla f(x),y-x\rangle+\frac{\beta}{2}\|y-x\|^2$$

Avec l'exercice suivant, on peut découvrir une interprétation de cette nouvelle notion.

Exercice B.6 (Caractérisation d'une fonction β-smooth)**.** *Montrer que f est une fonction β-smooth si, et seulement si, le gradient de f est β-Lipschitz.*

Commençons par nous assurer que l'on aura bien toujours une *descente* :

$$\begin{aligned}f(x_{t+1}) &\leqslant f(x_t)-\langle\nabla f(x_t),x_{t+1}-x_t\rangle+\frac{\beta}{2}\|x_{t+1}-x_t\|^2\\ &\leqslant f(x_t)-\eta_t\langle\nabla f(x_t),\nabla f(x_t)\rangle+\frac{\beta}{2}\eta_t^2\|\nabla f(x_t)\|^2\\ &\leqslant f(x_t)-\left(\eta_t-\frac{\beta\eta_t^2}{2}\right)\|\nabla f(x_t)\|^2\end{aligned}$$

Ainsi, si $\eta_t\leqslant\dfrac{1}{\beta}$ on a bien une descente ! Voyons comment, avec cette condition supplémentaire, nous sommes en capacité d'avoir une convergence plus rapide.

Proposition B.4. *Soit $x^* \in \mathbb{R}^d$ le minimum de la fonction $f : \mathbb{R}^d \mapsto \mathbb{R}$ une fonction convexe et β-smooth. On choisit $\eta = \frac{1}{\beta}$.*
Alors pour toutes itérations $t \leqslant T$ avec $T \geqslant 1$, on a :

$$f(x_t) - f(x^*) \leqslant \mathcal{O}\left(\frac{1}{T}\right)$$

Nous ne prouverons pas ici le résultat et renvoyons vers le très complet livre de Sébastien Bubeck *Convex optimization : Algorithms and complexity* publié en 2015, pour avoir le détail de la preuve.
Dans ce même article, nous pouvons trouver d'autres hypothèses, comme par exemple que f soit fortement convexe, pour accélérer encore la convergence.

Connaître ces propriétés permet parfois de choisir une fonction plutôt qu'une autre quand elles remplissent le même rôle dans l'entraînement d'un modèle. C'est particulièrement vrai dans l'entraînement des réseaux de neurones, que nous ne traitons pas ici, où un très grand nombre de paramètres est à apprendre. Avoir une vitesse de convergence plus rapide parce que nous avons choisi une fonction à optimiser la plus régulière possible permet de faire gagner parfois des jours voire semaines de calculs. C'est ce qui explique le très grand nombre de différentes méthodes de descente de gradient qui ont été développées.

B.4 La fonction de perte des réseaux de neurones n'est pas en général convexe

L'ensemble de cette annexe est à propos des fonctions convexe, et nous avons en particulier traité de la vitesse de convergence pour des objectifs avec des fonctions convexe. Dans les réseaux de neurones, la descente de gradient [1] est clé pour l'algorithme de back-propagation. Ainsi, nous pourrions tirer des sections de cette annexes des garanties de vitesses de convergence pour des réseaux de neurones.
Pour cela, nous devons nous assurer que la fonction de perte d'un réseau de neurone est convexe. Cela semble plausible puisque chaque neurones est une combinaison affine de ses inputs, et que la plupart des fonctions d'activation (par exemple ReLU) sont convexes.

Exercice B.7 (Combinaison de fonctions convexes)**.** *Soient $U, V \subseteq \mathbb{R}$ et $f : U \to V$ et $g : V \to \mathbb{R}$ deux fonctions convexes.*

1. *Montrer que si g est croissante, alors $g \circ f$ est une fonction convexe.*
2. *Montrer à l'aide d'un contre-exemple que si g est strictement décroissante, alors $g \circ f$ n'est pas une fonction convexe.*

Solution. On reprend les notations de l'exercice.

1. Plus précisément ses variantes en mini-batch, avec momentum ou pas adaptatif.

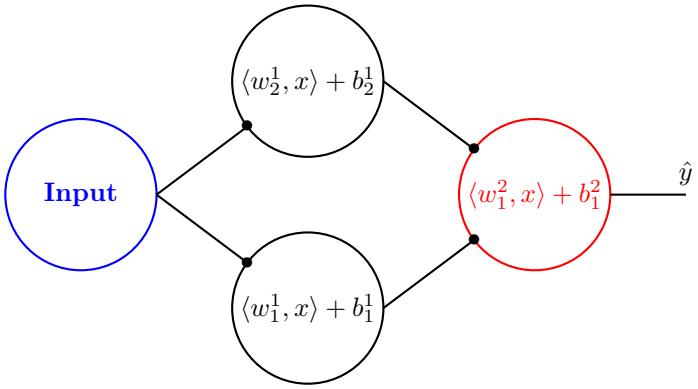


FIGURE B.5 – Réseau de neurone avec une couche caché de deux neurones, fonction d'activation ReLU

1. Soit $x, y \in U$ et $\lambda \in [0,1]$. Puisque f est convexe, on a :

$$\begin{aligned} f(\lambda x + (1-\lambda)y) &\leqslant \lambda f(x) + (1-\lambda) f(y) \quad \text{par définition} \\ (g \circ f)(\lambda x + (1-\lambda)y) &\leqslant g\left(\lambda f(x) + (1-\lambda) f(y)\right) \quad \text{car } g \text{ est croissante} \\ &\leqslant \lambda (g \circ f)(x) + (1-\lambda)(g \circ f)(y) \quad \text{car } g \text{ est convexe} \end{aligned}$$

   D'où $g \circ f$ est une fonction convexe.

2. Prenons $f(x) = x^2$ et $g(x) = e^{-x}$. Il s'agit bien de deux fonctions convexe puisque pour tout $x \in \mathbb{R}$, on a $f''(x) \geqslant 0$ et $g''(x) \geqslant 0$ avec le théorème B.1 de caractérisation des fonctions convexes. g est également strictement décroissante.
   Ici, $g \circ f(x) = e^{-x^2}$ et cette fonction n'est clairement pas convexe.

□

Avec l'exercice précédent, tout porte à croire que l'on a des chances d'avoir un réseau de neurones qui ait une fonction de perte convexe.

Exercice B.8. *On considère le réseau de neurones défini à la figure B.5. Pour simplifier les calculs, on fixe les valeurs de l'ensemble des biais* $b_1^1 = b_2^1 = b_1^2 = 0$.

On considère le dataset $\mathcal{D} = \{(-1,-1),(1,1)\}$ *et la fonction de perte* $\mathcal{L}(\theta) = \frac{1}{2}\sum_{i=1}^{2}(\hat{y}_i - y_i)^2$.

1. *Montrer que pour* θ_1 *:* $w_1^1 = 1, w_2^1 = -1, w_1^2 = (1,-1)$ *on a* $\mathcal{L}(\theta_1) = 0$
2. *Montrer que pour* θ_2 *:* $w_1^1 = -1, w_2^1 = 1, w_1^2 = (-1,1)$ *on a* $\mathcal{L}(\theta_2) = 0$
3. *Conclure sur la nature convexe ou non de la fonction de perte.*

Solution. Les deux premières questions sont immédiates. Si la loss est convexe, alors n'importe quelle combinaison convexe de ces paramètres donnera une loss inférieure ou égale à 0 (puisque $\mathcal{L}(\theta_1) = \mathcal{L}(\theta_2) = 0$).

En particulier, la moyenne arithmétique de θ_1 et θ_2 donne : $w_1^1 = 0, w_2^1 = 0, w_1^2 = (0,0)$. On obtient alors $\mathcal{L}\left(\frac{\theta_1 + \theta_2}{2}\right) = 1 > 0$. Donc la fonction de perte n'est pas convexe. □

Nous avons donc qu'avec un réseau de neurones aussi simple que le réseau de la figure B.5 la fonction de perte n'est pas convexe. Il est ainsi probable que pour des architectures de réseaux de neurones plus conséquent en taille, la fonction de perte soit encore plus non-convexe.
Nous ne pouvons donc plus nous appuyer sur les résultats théoriques obtenu pour la descente de gradient dans cette annexe.

Annexe D

Fléau de la dimension

Georg Cantor est un mathématicien allemand du 19e siècle qui est particulièrement connu pour son travail sur la théorie des ensembles et plus spécifiquement sur ses résultats concernant l'infini. L'ensemble des nombres entiers est infini par construction, mais l'ensemble des nombres entiers relatifs également. Et intuitivement, nous nous disons que ces infinis ne sont pas vraiment les mêmes, puisqu'il semblerait que $\mathbb{Z}$ soit plus grand que $\mathbb{N}$! Georg Cantor montre que ces deux infinis sont en fait les mêmes : il y a autant de nombres dans $\mathbb{Z}$ que dans $\mathbb{N}$. Plus fort encore, il démontre la puissance de l'infini qui nous donne un résultat encore plus contre intuitif : il y a autant de nombres dans l'intervalle $[0, 1]$ que dans $\mathbb{R}$ tout entier ! Alors qu'il venait de le démontrer, il a envoyé une lettre à son ami mathématicien Dedekind :

> *Tant que vous ne m'aurez pas approuvé, je ne puis que dire : je le vois mais je ne le crois pas.*
> — Georg Cantor (1877)

Il a prouvé quelque chose que l'ensemble de la communauté pensait intuitivement fausse, et lui-même n'y croyait pas. Nous sommes toujours mis en difficulté quand il s'agit de traiter avec l'infini, ou des grandes quantités. Cette remarque nous amène donc à nous questionner sur l'impact d'un grand nombre d'informations quand nous entraînons un modèle de Machine Learning.

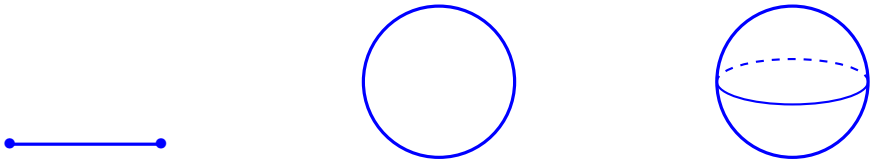
Le **fléau de la dimension** est une notion connue en statistiques et en Machine Learning. Ce terme rassemble tout un ensemble de phénomènes qui se produit en très grande dimension, mais pas dans une dimension plus petite. Nous proposons dans cette annexe d'illustrer quelques-uns des phénomènes étranges de la grande dimension et ses impacts en Machine Learning.

D.1 Volume d'une hypersphère

Pour essayer de sentir les problèmes de la très grande dimension, on s'intéresse au volume d'une hypersphère.

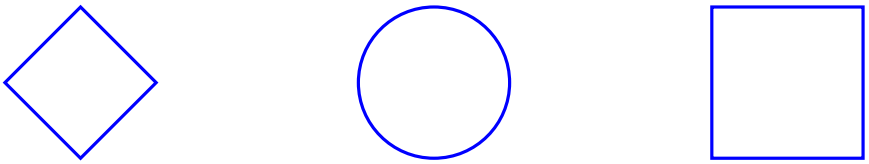
Avec la figure (D.1) nous avons l'intuition que le volume augmente avec la dimension. Donc pour une hypersphère de très grande dimension, on devrait avoir un très grand volume.
Nous avons tracé et mesuré le volume pour la distance euclidienne classique, mais nous pouvons aussi utiliser d'autres distances (autre norme) comme montré dans la figure (D.2).

Formalisons le problème et généralisons-le pour calculer le volume d'une hypersphère en n'importe quelle dimension et pour n'importe quelle norme. Soit $n \in \mathbb{N}^*$ la dimension de l'espace, on appelle *boule* ou hypersphère l'objet défini par :



(a) En dimension 1, volume = 2 (b) En dimension 2, volume = π (c) En dimension 3, volume = $\frac{4}{3}\pi$

FIGURE D.1 – Représentation et volume d'une hypersphère de rayon 1 dans 3 espaces de dimensions différentes



(a) Avec la norme 1 (b) Avec la norme 2 (c) Avec la norme infinie

FIGURE D.2 – Représentation d'une hypersphère de rayon 1 en dimension 2 pour 3 normes différentes

$$\begin{aligned} B_n^p(R) &= \{(x_1, \ldots, x_n) \in \mathbb{R}^n, \sum_{i=1}^{n} x_i^p \leqslant R^p\} \\ &= \{u \in \mathbb{R}^n, \|u\|_p^p \leqslant R^p\} \end{aligned}$$

Avec $\|u\|_p$ la norme p définie comme $\|u\|_p^p = \sum_{i=1}^{n} x_i^p$. $B_n^p(R)$ est la boule de dimension n avec une p-norme de rayon R. On définit $V_n^p(R)$ le volume de la boule $B_n^p(R)$ *i.e.* la mesure de $B_n^p(R)$ pour la mesure de Lesbegue dans $\mathbb{R}^n$. Formellement :

$$V_n^p(R) = \int_{B_n^p(R)} \bigotimes_{i=1}^{n} \mathrm{d}x_i$$

Proposition D.1 (Volume d'une hypersphere)**.** *Avec les notations précédentes, on a :*

$$\forall R > 0, \forall n \geqslant 2, \forall p \geqslant 1, \quad V_n^p(R) = \frac{\left(2R\Gamma\left(\frac{1}{p}+1\right)\right)^n}{\Gamma\left(\frac{n}{p}+1\right)}$$

Et son équivalent quand n tend vers l'infini :

$$V_n^p(R) \sim \sqrt{\frac{p}{2\pi n}}\left[2R\Gamma\left(\frac{1}{p}+1\right)\left(\frac{pe}{n}\right)^{\frac{1}{p}}\right]^n$$

Avec la fonction Γ définie comme :

$$\Gamma(x) = \int_0^{+\infty} e^{-t}t^{x-1}\mathrm{d}t$$

Démonstration. Soit la fonction ϕ définie comme :

$$\begin{array}{rccc} \phi & : & B_n^p(1) & \to & B_n^p(R) \\ & & (x_1,\ldots,x_n) & \mapsto & (Rx_1,\ldots,Rx_n) \end{array}$$

Par définition, si $u \in B_n^p(1)$, alors $\|u\|_p^p \leqslant 1 \iff \|Ru\|_p^p \leqslant R^p$ donc $\phi(u) \in B_n^p(R)$, en supposant que $R > 0$. Autrement dit, on ne considère pas le cas dégénéré. Par équivalence, on a que $\phi(B_n^p(1)) = \phi(B_n^p(R)$. ϕ est donc une bijection, ainsi par changement de variable : $V_n^p(R) = R^n V_n^p(1)$.

Par le théorème de Fubini :

$$\begin{aligned} V_n^p(1) &= \int_{-1}^{1} V_{n-1}^p\left(\sqrt[p]{1-|x|^p}\right)\mathrm{d}x \\ &= V_{n-1}^p(1)\int_{-1}^{1}(1-|x|^p)^{\frac{n-1}{p}}\,\mathrm{d}x \\ &= 2V_{n-1}^p(1)\int_{0}^{1}(1-|x|^p)^{\frac{n-1}{p}}\,\mathrm{d}x \\ &= \frac{2}{p}V_{n-1}^p(1)\int_0^1 y^{\frac{1}{p}-1}(1-y)^{\frac{n-1}{p}}\mathrm{d}y \end{aligned}$$

Rappelons la définition de la fonction Beta, définie pour $x, y \in \mathbb{R}_+^*$:

$$B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\mathrm{d}t$$

La fonction Beta est lié à la fonction Gamma par l'identité suivante :

$$B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

On peut alors écrire :

$$\begin{aligned} V_n^p(1) &= \frac{2}{p} V_{n-1}^p(1) \frac{\Gamma\left(\frac{1}{p}\right)\Gamma\left(\frac{n-1}{p}+1\right)}{\Gamma\left(\frac{n}{p}+1\right)} \\ &= V_{n-1}^p(1)\left[\frac{2}{p}\frac{\Gamma\left(\frac{1}{p}\right)\Gamma\left(\frac{n-1}{p}+1\right)}{\Gamma\left(\frac{n}{p}+1\right)}\right] \\ &= V_1^p(1)2^{n-1}\Gamma\left(\frac{1}{p}+1\right)^{n-1}\frac{\Gamma\left(\frac{1}{p}+1\right)}{\Gamma\left(\frac{n}{p}+1\right)} \\ &= V_1^p(1)2^{n-1}\frac{\Gamma\left(\frac{1}{p}+1\right)^n}{\Gamma\left(\frac{n}{p}+1\right)} \end{aligned}$$

Puisque $V_1^p(1) = 2$, on obtient finalement :

$$\forall n \geqslant 2, \forall p \geqslant 1, \quad V_n^p(1) = \frac{\left(2\Gamma\left(\frac{1}{p}+1\right)\right)^n}{\Gamma\left(\frac{n}{p}+1\right)}$$

Avec un rayon $R > 0$, plus généralement :

$$\forall R > 0, \forall n \geqslant 2, \forall p \geqslant 1, \quad V_n^p(R) = \frac{\left(2R\Gamma\left(\frac{1}{p}+1\right)\right)^n}{\Gamma\left(\frac{n}{p}+1\right)} \tag{D.1}$$

On déduit l'équivalent directement. □

Ce que ce résultat exhibe, c'est que le volume d'une hypersphère en grande dimension tend exponentiellement vite vers 0, c'est complètement contre intuitif ! Visualisons les courbes de cette fonction avec la figure (**??**).

On retrouve bien le comportement en hausse que nous avions observé, mais on comprend que le comportement ultime est que le volume tende vers 0 très rapidement. Avant de discuter de ce que ce résultat implique, regardons un autre résultat contre intuitif.

Exercice D.1 (Concentration dans l'hypersphère)**.** *Soit $\varepsilon > 0$. On considère une hypersphère de rayon R. Montrer que :*

$$\frac{V_n^p(R-\varepsilon)}{V_n^p(R)} = \left(1 - \frac{\varepsilon}{R}\right)^n$$

C'est encore plus étrange : les points semblent se concentrer proche des frontières de l'hypersphère, donc en ayant un *centre* vide. Cela veut dire que plus la dimension augmente, plus le volume tend vers 0 et que dans le même temps les données se rapprochent des frontières.
Donc si l'on distribue des points uniformément dans une sphère, la distribution des distances entre les points ne sera pas informative du tout. Ces intuitions sont confirmées par la figure (D.4).

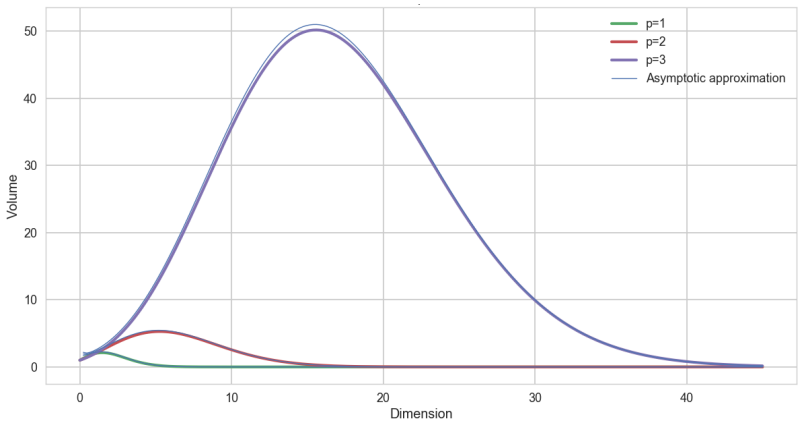


FIGURE D.3 – Volume de la boule unité en fonction de la dimension de son espace pour trois p-normes

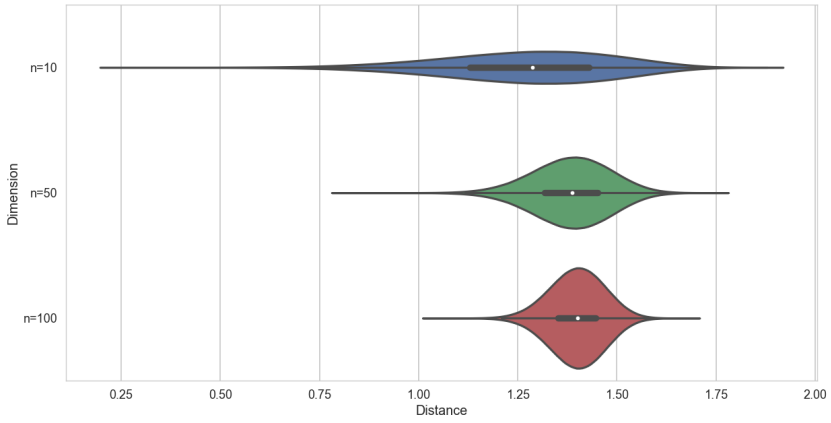


FIGURE D.4 – Distribution des distances entre chaque point en fonction de la dimension de l'espace

Ainsi, plus la dimension est grande, moins la notion de distance a du sens. Au-delà de l'aspect combinatoire et de stockage des données, avoir un modèle qui a moins d'indicateurs pour s'entraîner aura plus de chance d'être performant et *utile*. Par exemple, l'ensemble des méthodes de clustering par exemple seront largement impactées par une très grande dimension. Finalement, on peut remettre en cause l'exactitude d'une idée répandue : "*Avec plus d'informations les modèles sont meilleurs*". Ce qui est plus exact est qu'avec les informations utiles, les modèles sont meilleurs. C'est tout l'enjeu de la phase exploratoire et d'augmentation des données pour répondre à un problème de Machine Learning.

D.2 Orthogonalité à la surface d'une hypersphère

Nous avons donc montré que n'importe quelle distance issue d'une norme $\mathcal{L}_p$ était soumise au fléau de la dimension. En NLP *classique*, il est fréquent d'utiliser la distance cosinus, où le cadre est très souvent en très grande dimension. Les résultats semblent montrer que le fléau de la dimension n'affecte pas la distance cosinus. Vérifions.

On peut définir le produit scalaire entre $x, y \in \mathbb{R}^n$ comme :

$$< x, y >= \|x\|_2 \|y\|_2 \cos(\theta)$$

Avec θ l'angle entre le représentant de x et le représentant de y à l'origine comme défini dans la figure (D.5).

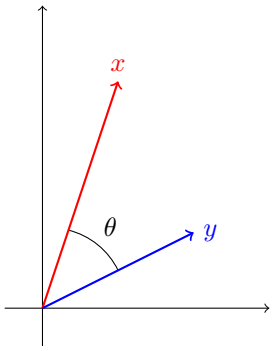


FIGURE D.5 – Définition de la métrique cosinus

Nous pouvons donc naturellement définir la métrique cosinus comme :

$$cosine(x, y) = \frac{< x, y >}{\|x\|_2 \|y\|_2}$$

On comprend que la distance cosinus sera bornée dans $[-1, 1]$ contrairement aux restes des distances qui ne sont pas bornées. Par définition, la métrique cosinus est liée à la distance euclidienne, et on remarque que si l'on prend x et y deux vecteurs unitaires, on obtient :

$$\begin{aligned} \|x - y\|_2^2 &= \|x\|_2^2 + \|y\|_2^2 - 2 < x, y > \\ &= \|x\|_2^2 + \|y\|_2^2 - 2cosine(x, y)\|x\|\|y\| \\ &= 2\left[1 - cosine(x, y)\right] \end{aligned}$$

Il serait donc très surprenant qu'une distance avec un lien aussi fort avec la distance euclidienne, ne souffre pas du fléau de la dimension. Nous avons un résultat intéressant :

Lemme D.1. *Soit x, y deux vecteurs choisis indépendamment à la surface d'une hypersphère. Alors, avec une probabilité supérieure à $1 - \frac{1}{n}$:*

$$|cosine(x, y)| = O\left(\sqrt{\frac{\log n}{n}}\right)$$

Autrement dit, en prenant deux vecteurs aléatoirement à la surface d'une boule en dimension n, on a avec une très grande probabilité que ces deux vecteurs sont orthogonaux. Cela rend inutilisable la métrique cosinus en très grande dimension.

Démonstration. Soit $a \in \mathbb{R}^n$ tel que $\|a\|_2 = 1$. Soit $x \in S_n^2$ avec $S_n^p = \{x \in \mathbb{R}^n \mid \|x\|_p = 1\}$ la surface de l'hyperboule unitaire. Soit $x \in \mathbb{R}^n$ un vecteur construit avec chacune de ses coordonnées sélectionnées aléatoirement entre -1 et 1. Puis on normalise le vecteur x.
Soit $X =< a, x >$, alors il est simple de montrer que $\mathbb{E}[X] = 0$ et $\mathbb{E}\left[X^2\right] = \frac{1}{n}$. Ainsi, en exploitant l'inégalité de Chernoff on a :

$$\mathbb{P}(|X| \geqslant t) \leqslant 2e^{-\frac{nt^2}{4}}$$

Donc pour $\varepsilon = 2e^{-\frac{nt^2}{4}} \iff t = \sqrt{\frac{-4\log\left(\frac{\varepsilon}{2}\right)}{n}}$ et si l'on choisit $\varepsilon = \frac{1}{n}$, on obtient :

$$\mathbb{P}\left(|cosine(x,y)| \geqslant \sqrt{\frac{4\log(2n)}{n}}\right) \leqslant \frac{1}{n}$$

□

Cette preuve complète la présentation du second comportement étrange que l'on mentionne concernant les hyperboules et hypersphères.

Un deuxième problème majeur en grande dimension est que le nombre de données à obtenir pour être capable d'avoir des garanties statistiques sur la qualités de l'apprentissage est colossal, c'est exponentiel. Ainsi, on peut se poser la question de la capacité des algorithmes à *apprendre* en grande dimension.

D.3 Interpolation et extrapolation

L'ensemble du Machine Learning tel qu'on l'a présenté correspond à de l'interpolation et à essayer de faire en sorte que cette interpolation puisse être capable d'extrapoler correctement. Randall Balestriero, Jerome Pesenti et Yann Le Cun ont publié en 2021 l'article *Learning in High Dimension always amount to extrapolation* [Balestriero et al., 2021] dont voici le résumé :

> *The notion of interpolation and extrapolation is fundamental in various fields from deep learning to function approximation. Interpolation occurs for a sample x whenever this sample falls inside or on the boundary of the given dataset ?s convex hull. Extrapolation occurs when x falls outside of that convex hull. One fundamental (mis)conception is that state-of-the-art algorithms work so well because of their ability to correctly interpolate training data. A second (mis)conception is that interpolation happens throughout tasks and datasets, in fact, many intuitions and theories rely on that assumption. We empirically and theoretically argue against those two points and demonstrate that on any high-dimensional (>100) dataset, interpolation almost surely never happens. Those results challenge the validity of our current interpolation/extrapolation definition as an indicator of generalization performances.*
>
> — Randall Balestriero, Jerome Pesenti et Yann Le Cun (2021)

L'objet de l'article est de montrer que l'on comprend et définit mal les notions d'interpolation et d'extrapolation en Machine Learning. Cela a des impacts théoriques et donc pratiques sur notre conception et les garanties mathématiques que l'on peut avoir sur les comportements des algorithmes présentés en très grande dimension. Nous invitons à lire en détail cet article pour en apprendre plus sur le sujet en lui-même, mais également pour voir qu'un domaine qui semble plutôt bien établi et en constante expansion se pose encore des questions sur ses fondements.

En résumé, le fléau de la dimension met en lumière les limites de notre intuition humaine et nous amène à nous questionner encore aujourd'hui sur les fondements communément acceptés. Être capable de répondre à ces questions nous permettrait d'être plus précis et plus complet sur notre approche de l'apprentissage en grande dimension.
De manière plus pragmatique, un data scientist doit être au courant que ces questions existent et que le fléau de la dimension va impacter son travail. D'où les techniques de réduction de dimension qui aident à résoudre le problème, mais ne le résolvent clairement pas par construction.

Bibliographie

[Achille et al., 2017] Achille, A., Rovere, M., and Soatto, S. (2017). Critical learning periods in deep neural networks. *arXiv preprint arXiv :1711.08856*.

[Anil et al., 2023] Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. (2023). Palm 2 technical report. *arXiv preprint arXiv :2305.10403*.

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv :1607.06450*.

[Balduzzi et al., 2017] Balduzzi, D., Frean, M., Leary, L., Lewis, J., Ma, K. W.-D., and McWilliams, B. (2017). The shattered gradients problem : If resnets are the answer, then what is the question ? In *International Conference on Machine Learning*. PMLR.

[Balestriero et al., 2021] Balestriero, R., Pesenti, J., and LeCun, Y. (2021). Learning in high dimension always amounts to extrapolation. *arXiv preprint arXiv :2110.09485*.

[Barron, 2017] Barron, J. T. (2017). Continuously differentiable exponential linear units. *arXiv preprint arXiv :1704.07483*.

[Barron, 2021] Barron, J. T. (2021). Squareplus : A softplus-like algebraic rectifier. *arXiv preprint arXiv :2112.11687*.

[Belkin et al., 2019] Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proceedings of the National Academy of Sciences*.

[Błasiok et al., 2023a] Błasiok, J., Gopalan, P., Hu, L., and Nakkiran, P. (2023a). A unifying theory of distance from calibration. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1727–1740.

[Błasiok et al., 2023b] Błasiok, J., Gopalan, P., Hu, L., and Nakkiran, P. (2023b). When does optimizing a proper loss yield calibration ? *arXiv preprint arXiv :2305.18764*.

[Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*.

[Chen et al., 2023] Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Liu, Y., Pham, H., Dong, X., Luong, T., Hsieh, C.-J., et al. (2023). Symbolic discovery of optimization algorithms. *arXiv preprint arXiv :2302.06675*.

[Chowdhery et al., 2022] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). Palm : Scaling language modeling with pathways. *arXiv preprint arXiv :2204.02311*.

[Dauphin et al., 2016] Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2016). Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR.

[Dawid, 1982] Dawid, A. P. (1982). The well-calibrated bayesian. *Journal of the American Statistical Association*.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*.

[Fan et al., 2018] Fan, A., Lewis, M., and Dauphin, Y. (2018). Hierarchical neural story generation. *arXiv preprint arXiv :1805.04833*.

[Giffin and Mitchell, 1978] Giffin, F. and Mitchell, D. E. (1978). The rate of recovery of vision after early monocular deprivation in kittens. *The Journal of Physiology*.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

[Gneiting and Raftery, 2007] Gneiting, T. and Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477) :359–378.

[Gopalan et al., 2022] Gopalan, P., Kim, M. P., Singhal, M. A., and Zhao, S. (2022). Low-degree multicalibration. In *Conference on Learning Theory*, pages 3193–3234. PMLR.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers : Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[Hendrycks and Gimpel, 2016] Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv :1606.08415*.

[Hinton et al., 2012a] Hinton, G., Srivastava, N., and Swersky, K. (2012a). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*.

[Hinton et al., 2012b] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv :1207.0580*.

[Hoffmann et al., 2022] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. (2022). Training compute-optimal large language models. *arXiv preprint arXiv :2203.15556*.

[Holtzman et al., 2019] Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. (2019). The curious case of neural text degeneration. *arXiv preprint arXiv :1904.09751*.

[Huang et al., 2016] Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *Computer Vision–ECCV 2016 : 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization : Accelerating deep network training by reducing internal covariate shift. *International conference on machine learning*.

[Ishida et al., 2020] Ishida, T., Yamane, I., Sakai, T., Niu, G., and Sugiyama, M. (2020). Do we need zero training loss after achieving zero training error ? *arXiv preprint arXiv :2002.08709*.

[Kakade and Foster, 2004] Kakade, S. M. and Foster, D. P. (2004). Deterministic calibration and nash equilibrium. In *International Conference on Computational Learning Theory*, pages 33–48. Springer.

[Kaplan et al., 2020] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv :2001.08361*.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam : A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

[Klambauer et al., 2017] Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-normalizing neural networks. *Advances in neural information processing systems*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*.

[Krogh and Hertz, 1991] Krogh, A. and Hertz, J. (1991). A simple weight decay can improve generalization. *Advances in neural information processing systems*.

[LeCun et al., 1998] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (1998). Efficient backprop. *Tricks of the Trade*.

[Lee and Cherkassky, 2022] Lee, E. H. and Cherkassky, V. (2022). Vc theoretical explanation of double descent. *arXiv preprint arXiv :2205.15549*.

[Li et al., 2018] Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31.

[Liu et al., 2018] Liu, W., Lin, R., Liu, Z., Liu, L., Yu, Z., Dai, B., and Song, L. (2018). Learning towards minimum hyperspherical energy. *Advances in neural information processing systems*, 31.

[Liu et al., 2023] Liu, Z., Xu, Z., Jin, J., Shen, Z., and Darrell, T. (2023). Dropout reduces underfitting. *arXiv preprint arXiv :2303.01500*.

[Loshchilov and Hutter, 2016] Loshchilov, I. and Hutter, F. (2016). Sgdr : Stochastic gradient descent with warm restarts. *arXiv preprint arXiv :1608.03983*.

[Loshchilov and Hutter, 2019] Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. *International Conference on Learning Representations (ICLR)*.

[Maas et al., 2013] Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*.

[Misra, 2019] Misra, D. (2019). Mish : A self regularized non-monotonic activation function. *arXiv preprint arXiv :1908.08681*.

[Mitchell, 1988] Mitchell, D. E. (1988). The extent of visual recovery from early monocular or binocular visual deprivation in kittens. *The Journal of physiology*.

[Nakkiran, 2021] Nakkiran, P. (2021). *Towards an Empirical Theory of Deep Learning*. PhD thesis, Harvard University.

[Olson and Freeman, 1980] Olson, C. R. and Freeman, R. (1980). Profile of the sensitive period for monocular deprivation in kittens. *Experimental Brain Research*.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn : Machine learning in python. *The journal of machine learning research*.

[Penedo et al., 2023] Penedo, G., Malartic, Q., Hesslow, D., Cojocaru, R., Cappelli, A., Alobeidli, H., Pannier, B., Almazrouei, E., and Launay, J. (2023). The refinedweb dataset for falcon llm : outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv :2306.01116*.

[Rae et al., 2021] Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., et al. (2021). Scaling language models : Methods, analysis & insights from training gopher. *arXiv preprint arXiv :2112.11446*.

[Ramachandran et al., 2017] Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv :1710.05941*.

[Reed et al., 2022] Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. (2022). A generalist agent. *arXiv preprint arXiv :2205.06175*.

[Savage, 1971] Savage, L. J. (1971). Elicitation of personal probabilities and expectations. *Journal of the American Statistical Association*, 66(336) :783–801.

[Schmidt et al., 2021] Schmidt, R. M., Schneider, F., and Hennig, P. (2021). Descending through a crowded valley-benchmarking deep learning optimizers. In *International Conference on Machine Learning*. PMLR.

[Shazeer, 2020] Shazeer, N. (2020). Glu variants improve transformer. *arXiv preprint arXiv :2002.05202*.

[Smith, 2017] Smith, L. N. (2017). Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout : a simple way to prevent neural networks from overfitting. *The journal of machine learning research*.

[Tan et al., 2022] Tan, C., Gao, Z., Wu, L., Li, S., and Li, S. Z. (2022). Hyperspherical consistency regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7244–7255.

[Thoppilan et al., 2022] Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., et al. (2022). Lamda : Language models for dialog applications. *arXiv preprint arXiv :2201.08239*.

[Touvron et al., 2023a] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023a). Llama : Open and efficient foundation language models. *arXiv preprint arXiv :2302.13971*.

[Touvron et al., 2023b] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. (2023b). Llama 2 : Open foundation and fine-tuned chat models. *arXiv preprint arXiv :2307.09288*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Xiong et al., 2020] Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. (2020). On layer normalization in the transformer architecture. In *International Conference on Machine Learning*. PMLR.

[Zhang and Sennrich, 2019] Zhang, B. and Sennrich, R. (2019). Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.

[Zhang et al., 2019] Zhang, H., Dauphin, Y. N., and Ma, T. (2019). Fixup initialization : Residual learning without normalization. *ICLR*.

[Zhou et al., 2023] Zhou, C., Liu, P., Xu, P., Iyer, S., Sun, J., Mao, Y., Ma, X., Efrat, A., Yu, P., Yu, L., et al. (2023). Lima : Less is more for alignment. *arXiv preprint arXiv :2305.11206*.