

---

# RECENT ADVANCES IN MACHINE LEARNING

---

2023-2024  
Théo Lopès-Quintas

# Cadre et approche du cours

Porté par des avancées techniques, théoriques et un nombre croissant de praticiens, le Machine Learning connaît un développement exponentiel depuis les années 2010 : le nombre de publications a plus que doublé en 10 ans [Maslej et al., 2024]. Plus spécifiquement, une tendance forte qui se dégage ces dernières années est de publier, sur arXiv par exemple, des articles avant qu'ils ne soient revus pour être publiés dans un journal. Cette dynamique nous apprend deux choses :

1. Le nombre d'articles publiés est trop important pour être capable de suivre l'ensemble des avancées du domaine.
2. Puisque de plus en plus d'articles sont publiés sans être vérifiés, il est possible que le nombre d'articles non qualitatifs soit en croissance.

Les avancées sont pour la majorité *open-source* : cela permet à tout le monde d'en bénéficier, de l'examiner et de l'améliorer, alimentant ainsi ce flux constant d'innovation. Cette ébullition nous invite à nous rappeler la remarque de Preetum Nakkiran dans l'introduction de sa thèse :

*Les praticiens et les théoriciens sont régulièrement surpris par les avancées en apprentissage profond : des cas où la modification des ingrédients conduit à des gains inattendus dans le système final.*

— Preetum Nakkiran (2021)

Nous proposons de nous étonner avec la dynamique de la science expérimentale où la théorie inspire l'expérimentation et inversement lors de ces quelques séances. Le contenu des articles que nous traiterons porte sur des idées *nouvelles* qui nous amènent à porter un regard neuf sur les idées *anciennes*. Nous devons donc apprendre à *désapprendre*.

*La nouvelle éducation doit enseigner à l'individu comment classer et reclasser l'information, comment évaluer sa véracité, comment changer de catégories quand cela est nécessaire, comment passer du concret à l'abstrait et inversement, comment regarder les problèmes sous un nouvel angle - comment s'enseigner à soi-même. L'illettré de demain ne sera pas l'homme qui ne sait pas lire ; il sera l'homme qui n'a pas appris à apprendre.*

— Herbert Gerjuoy (1970)

Tel est le programme du cours. Au-delà de l'objectif d'acquérir des connaissances techniques et théoriques dans ces domaines, nous visons à montrer comment se documenter, se questionner et exploiter les mathématiques pour démêler le vrai du faux dans un discours avec de plus en plus d'acteurs. Un défi supplémentaire, permis par les avancées que nous traiterons, est celui d'une utilisation éclairée des IA génératives dans l'apprentissage.

*L'IA amplifie l'intelligence humaine comme la machine amplifie la force.*  
— Yann Le Cun (2023)

Pour amplifier quelque chose, il faut que cette chose soit présente : commençons !

# 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  $l \leq L$ . Chacun de ces neurones est caractérisé par trois objets :

- **Poids** : un vecteur de paramètres qui associe à chacun des coefficients du vecteur d'entrée du neurone un poids
- **Biais** : un scalaire qui rend affine la transformation
- **Activation** : une fonction qui, à 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 neurones, 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 de régression, une fonction de coût possible est la MSE (Mean Squared Error), et dans le cadre d'une classification, la Cross-Entropy. Puisque les notations sont déjà lourdes, nous prendrons quelques libertés de notation pour privilégier des équations simples quand il n'y a pas d'ambiguïté. Ainsi, l'objectif de la backpropagation est de calculer les quantités :

- $\frac{\partial \mathcal{C}}{\partial w_{j,k}^l}$  : pour calculer la mise à jour du  $k$ -ième poids du neurone  $j \leq N_l$  à la couche  $l \leq L$ , avec  $k \leq N_{l-1}$
- $\frac{\partial \mathcal{C}}{\partial b_j^l}$  : pour calculer la mise à jour du biais du neurone  $j \leq N_l$  à la couche  $l \leq L$

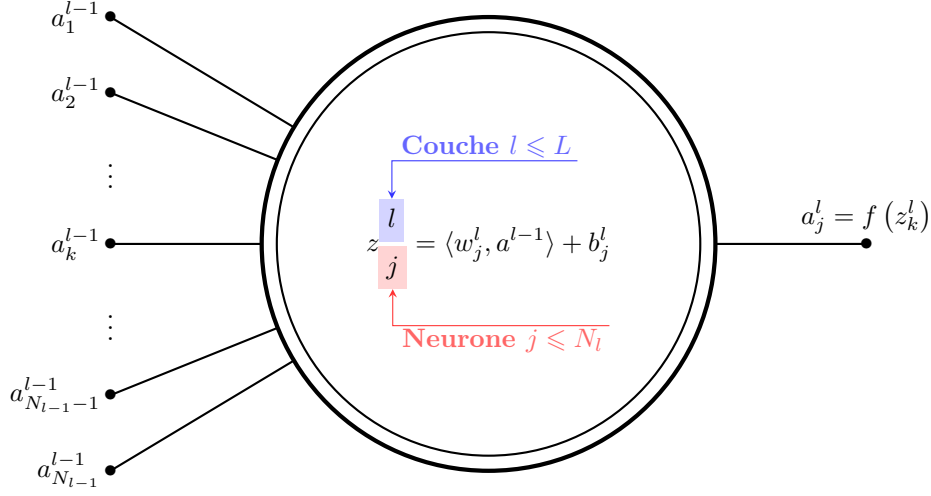


FIGURE 1.1 – Neurone  $j$  de la couche  $l$  paramétré par les poids  $w_i^j \in \mathbb{R}^{N_{l-1}}$  et le biais  $b_j^l \in \mathbb{R}$

Pour le faire, résolvons l'exercice suivant.

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

1. Soit  $l \leq L$ ,  $j \leq N_l$  et  $k \leq N_{l-1}$ . 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  $\delta_j^l = \frac{\partial C}{\partial z_j^l}$  qui représente l'erreur du neurone  $j \leq N_l$  à la couche  $l \leq L$ .

Montrer que pour la dernière couche  $L$ , on a :

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

3. On considère à présent la couche  $l < L$ . Montrer que l'on peut exprimer  $\delta_j^l$  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  $j \leq N_l$  à la couche  $l \leq L$ . 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  $\frac{\partial C}{\partial z_j^l}$  qui apparait. Elle représente l'erreur du neurone  $j \leq N_l$  à la couche  $l \leq L$ . L'exercice nous propose de la noter  $\delta_j^l$  pour la suite.

2. On considère à présent spécifiquement la dernière couche. Soit  $j \leq N_L$  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  $\frac{\partial C}{\partial a_j^L}$  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  $\delta_j^l$  en fonction de  $\delta_j^L$  : 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 :

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

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

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

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 backpropagation, on comprend que chaque poids est calculé de proche en proche, en partant de la couche la plus proche de la sortie à la plus profonde (celle juste après l'entrée). Puisque les calculs se font de proche en proche, 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 neurone de la couche  $l \leq L$  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 :

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

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

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

Or, dans les premiers réseaux de neurones, la fonction sigmoïde 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'apprenne 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 nombreuses 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 ont 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 n'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 (2002)

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 :

$$z_k^l = \langle w_k^l, x^l \rangle$$

$\forall l \leq L, \forall k \in N_l$

Diagram illustrating the notation for the output of a neuron  $k$  at layer  $l$ :

- Nombre de couche du réseau** (Number of layers of the network) points to  $L$ .
- Vecteur d'entrée de la couche  $l$**  (Input vector of layer  $l$ ) points to  $x^l$ .
- Matrice des poids du neurone  $k$  à la couche  $l$**  (Weight matrix of neuron  $k$  at layer  $l$ ) points to  $w_k^l$ .
- Nombre de neurones de la couche  $l$**  (Number of neurons in layer  $l$ ) points to  $N_l$ .

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

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 :

$$\forall l \leq L, \mathbb{V}[z^l] = \mathbb{V}[z^1] \quad (1.1)$$

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



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

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

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

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

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

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 aboutit à la même forme d'équation sauf que cette fois  $n_{\text{input}}$  représente le nombre de neurones de la couche inférieure. Nous venons de reproduire le raisonnement de [Glorot and Bengio, 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 initialisés selon une loi normale  $\mathcal{N}(0, \sigma^2)$  :  $\sigma = \sqrt{\frac{2}{\text{fan in} + \text{fan out}}}$

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

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).

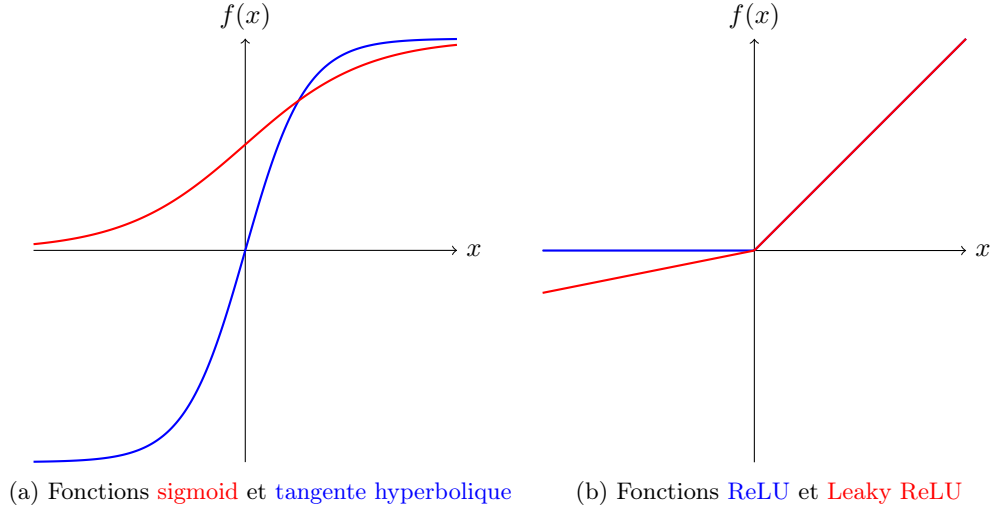


FIGURE 1.2 – Différentes fonctions d'activations non-linéaire

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] 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 :

$$\begin{aligned}
 \mathbb{E}[x^2] &= \mathbb{E}[f(z)^2] \\
 &= \int_{-\infty}^{+\infty} f(t)^2 p(t) dt \text{ avec } p \text{ la densité de probabilité associé à } z \\
 &= \int_0^{+\infty} t^2 p(t) dt \\
 &= \frac{1}{2} \mathbb{V}[z]
 \end{aligned}$$

Nous avons cette fois :

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

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

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

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  $\mathcal{U}([-a, a])$  :  $a = \sqrt{\frac{6}{\text{fan mode}}}$  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. On peut la visualiser avec la figure (1.2b).

**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 :

$$\begin{aligned} \mathbb{E}[x^2] &= \mathbb{E}[f(z)^2] \\ &= \int_{-\infty}^{+\infty} \max\{\alpha t, t\}^2 p(t) dt \\ &= \int_{-\infty}^0 \alpha^2 t^2 p(t) dt + \int_0^{+\infty} t^2 p(t) dt \\ &= \frac{\alpha^2 + 1}{2} \mathbb{V}[z] \end{aligned}$$

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*.  $\square$

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 distribution pour l'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, puisqu'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 [Achille et al., 2017].

Dans les années 1950, une des premières pistes suivies 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 des raisons techniques et théoriques, comme évoqué plus tôt, les réseaux de neurones sont abandonnés pendant plusieurs décennies avant de devenir incontournables 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 de l'acuité visuelle des chatons en 1963. Cette étude est adaptée 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 incomplètes ? Il est montré qu'une 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 issus de l'article [Achille et al., 2017] mais certains ont été reproduits à la main pour ces notes de cours pour des raisons de lisibilité.

La première expérience réalisée consiste à entraîner un réseau de neurones avec des images floues pendant  $N$  époques puis l'entraîner avec les images originales pendant 160 époques supplémentaires comme décrit par la figure (??). Pour comparer, on entraîne un réseau de neurones sans déficit et on montre les performances pour chaque époque 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.

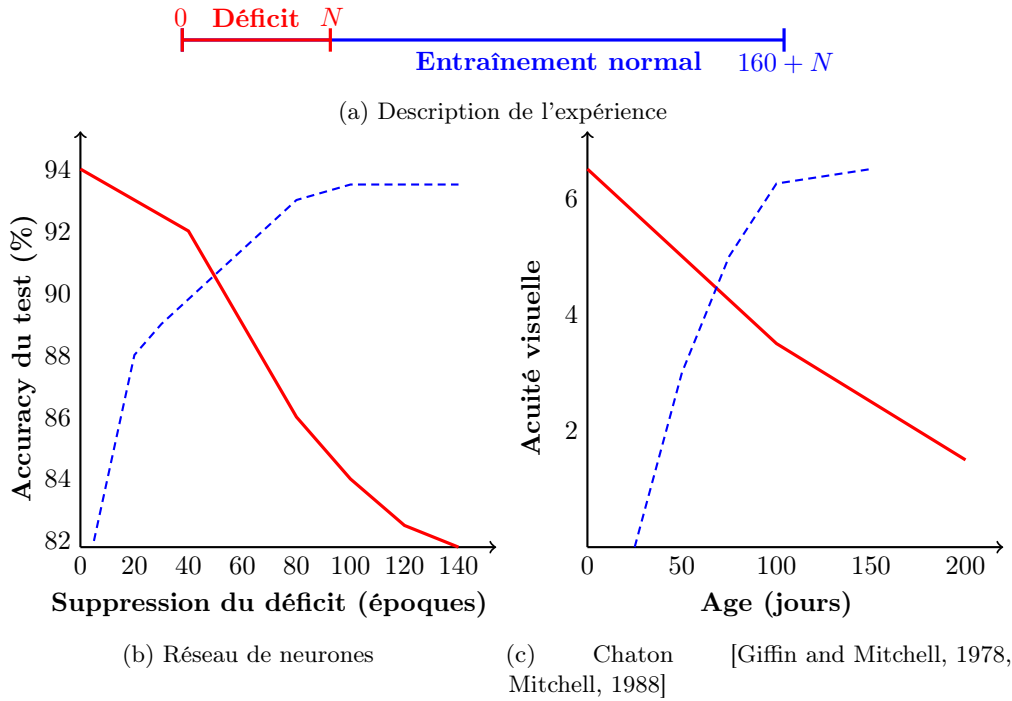


FIGURE 1.3 – Performance/acuité selon la durée du **déficit** par rapport à un développement **normal**

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 lorsque 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 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,

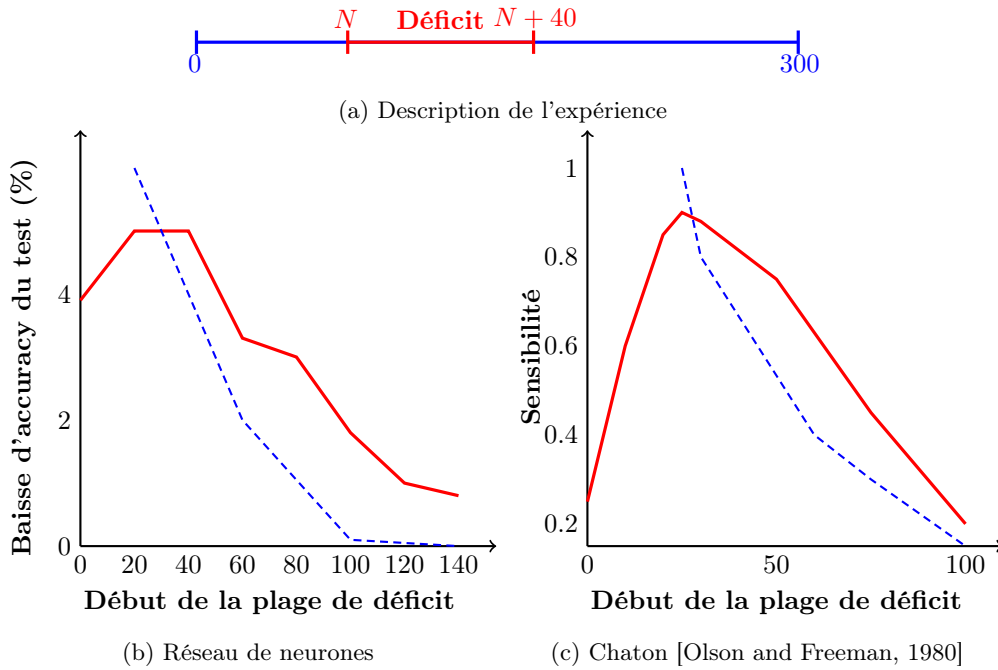


FIGURE 1.4 – Sensibilité selon le début du **déficit** par rapport à un développement **normal**

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  $\mathbb{P}_w(y | x)$  où  $w$  représente l'ensemble des poids qui paramètre le réseau de neurones. Si l'on note  $w' = w + \delta w$  où  $\delta$  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  $\mathbb{P}_w(y | x)$  et  $\mathbb{P}_{w'}(y | x)$ . Mais comment mesurer *l'écart* entre deux distributions ?

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

$$D_{\text{KL}} \left( \overset{\text{Distribution de densité } p}{\underset{\text{Distribution de densité } q}{P \parallel Q}} \right) = \int_{-\infty}^{+\infty} p(x) \ln \left( \frac{p(x)}{q(x)} \right) dx \quad (\text{Kullack-Leibler})$$

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

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

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

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

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

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  $\lim_{x \rightarrow 0} x \ln(x) = 0$ , 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  $\varepsilon$  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 :

$$D_{KL}(\mathbb{P}_w(y \mid x) \parallel \mathbb{P}_{w'}(y \mid x)) = \delta w F(\delta w) + o(\delta w^2)$$

Avec  $F$  la métrique d'information de Fisher. On peut la comprendre dans ce contexte comme une métrique **locale** de la perturbation de la valeur d'un poids. Concrètement, si un poids a une information de Fisher faible, c'est que le poids n'est pas très important pour la sortie du réseau de neurones.

À nouveau, la matrice d'information de Fisher est trop grande pour être calculée systématiquement. Ce que l'article propose finalement, c'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 neurones. 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 utilisés. Ils sont connus pour avoir des propriétés plus lisses que les autres types de réseaux 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 remplacées par une bonne initialisation et tout de même obtenir des résultats compétitifs

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.

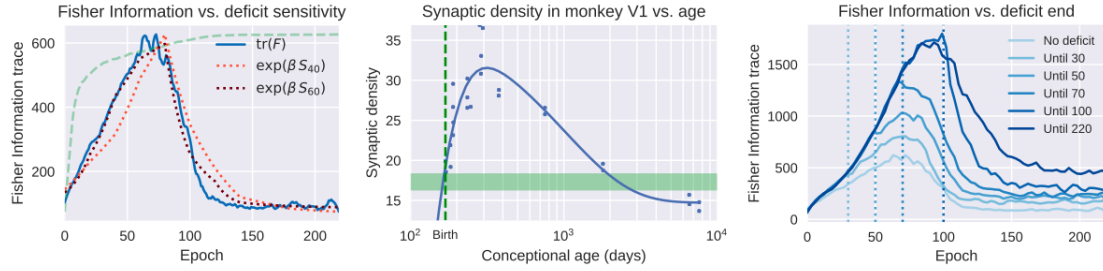


FIGURE 1.5 – Evolutions de la trace de la matrice d'information de Fisher (source : [Achille et al., 2017])

Dans le graphique de **gauche** est tracé en pointillé vert la performance du réseau sur le dataset de test. La courbe bleue 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 peu 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 seconde de compression d'informations où les connexions *inutiles* sont supprimées. Remarquons que lorsque les images sont trop difficiles pour être classifiées à cause du déficit introduit, le réseau est forcé de mémoriser des observations.

Voyons comment les différentes couches participent à ce phénomène.

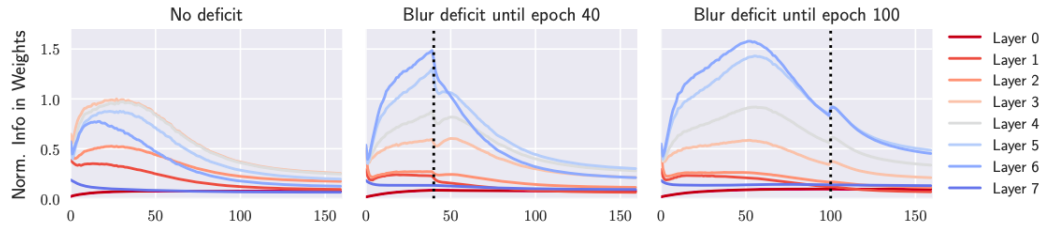


FIGURE 1.6 – Evolutions de la trace de la matrice d'information de Fisher par couche (source : [Achille et al., 2017])

Pour un entraînement normal, le réseau semble *apprendre* avec les couches du milieu principalement. Ce n'est pas le cas dès que l'on introduit un déficit : les dernières couches sont les plus



misés à contribution et stockent énormément d'informations. Ce phénomène semble proportionnel au temps de maintien du déficit.

La puissance de représentation du réseau de neurones n'est pas utilisée 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 inputs est de le régulariser. Observons l'impact du *weight decay*<sup>1</sup>.

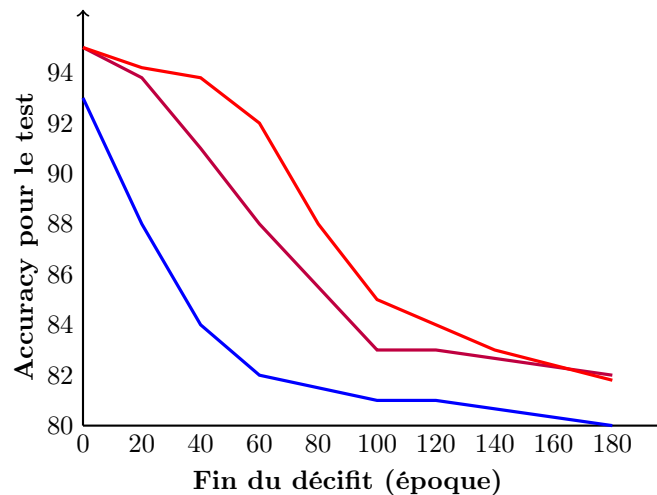


FIGURE 1.7 – Performance sur le dataset de test en fonction du nombre d'époque de maintien du déficit pour un weight decay de  $\lambda = 0$ ,  $\lambda = 5 \cdot 10^{-4}$  et  $\lambda = 10 \cdot 10^{-4}$

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.

Précisons que les graphiques et résultats présentés sont ceux pour des images floutées. Pour des *dégradations* moins fortes de l'image comme des rotations ou des symétries, des comportements similaires ne sont pas rapportés par l'article.

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 rejoint celle de l'article *Dropout Reduces Underfitting*[Liu et al., 2023] en 2023. Le dropout [Hinton et al., 2012b] et [Srivastava et al., 2014] est une méthode de régularisation qui est rapidement devenue 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

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

à chaque couche, dont l'entrée, sauf la sortie. On obtient donc à chaque passe forward un sous-ensemble du réseau initial.

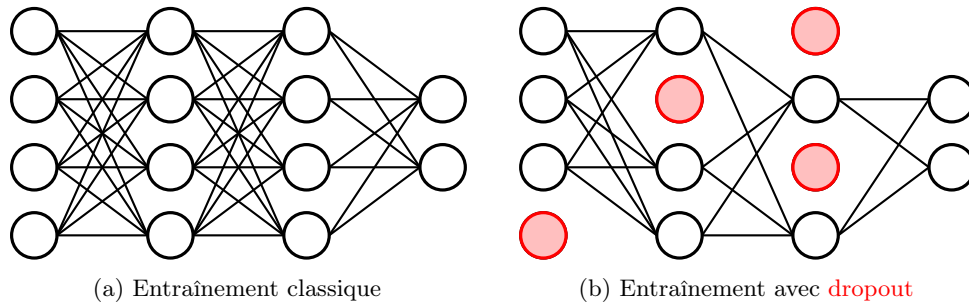


FIGURE 1.8 – Réseau de neurones avec et sans dropout

Lors de l'exploitation du réseau de neurones en revanche, nous ne pouvons pas avoir de 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 indépendants 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 cela, chaque poids est multiplié par la proportion  $p$  de dropout.

Le dropout permet d'éviter une sur-spécialisation de certains neurones, forçant le réseau à être plus redondant et robuste, de la même manière que chaque membre d'une équipe en entreprise doit être capable de se remplacer en cas de congé.

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 et le nomme *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 soit 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 meilleures conditions pour la suite de l'entraînement. On observe en effet que la réduction de loss est inférieure tant que le dropout est effectif, et cela s'inverse lorsque l'early dropout s'arrête.

On peut se poser la question de la régularisation en fin d'entraînement. Est donc introduit le *late dropout*, mais il 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'aux réseaux de neurones résiduels (ResNet) introduits un an plus tôt<sup>2</sup> [He et al., 2016]. Les réseaux de neurones résiduels sont constitués de nombreux ResBlock dont la constitution est illustrée avec la figure (1.9).

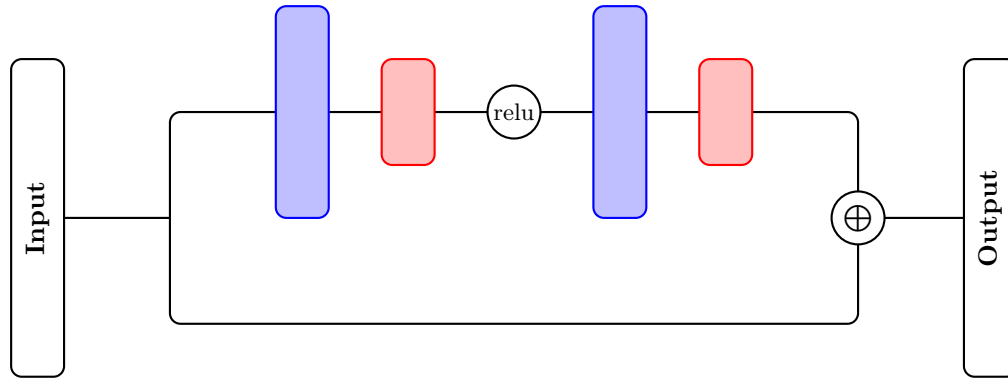


FIGURE 1.9 – ResBlock dans un ResNet avec des couches de **convolutions** et de **Batch Normalisation**

La partie *haute* correspond à une transformation de l'entrée tandis que la partie *basse* permet de ramener identiquement l'entrée 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<sup>3</sup> pour les autres types de réseaux de neurones. On peut noter que GATO de Deepmind [Reed et al., 2022] utilise cet outil.

L'article exploite de nombreuses techniques 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<sup>4</sup>. 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. À 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 meilleures 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 rattrapée.

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

## Séance 2

# Introduction à la Calibration

Les modèles de Machine Learning sont employés chaque jour un peu plus, en particulier pour des prises de décisions. La performance en termes d'optimisation de la fonction de perte est indispensable. Cependant, nous avons également besoin que le modèle soit capable de renseigner un niveau de confiance **fiable** : prendre une décision avec un niveau de confiance de 99% est plus aisé que de prendre une décision avec un niveau de confiance de 51%. À condition que l'on puisse se fier à ces valeurs !

Cette notion s'appelle la calibration. On dit qu'un prédicteur est calibré quand la probabilité prédite par le modèle est celle que l'on observe réellement. Concrètement, pour un modèle parfaitement calibré, si la probabilité qu'il y ait un chat sur une photo est estimée à 0.9, dans 90% des photos où la probabilité était de 0.9, il y avait effectivement un chat. La performance d'optimisation et la calibration sont deux concepts différents, mais pas nécessairement opposés. Il est tout à fait possible d'avoir un modèle très performant mais pas calibré et inversement.

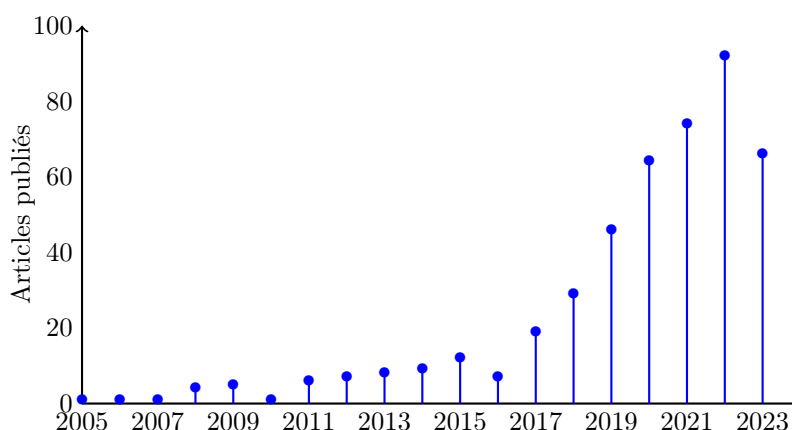


FIGURE 2.1 – Nombres d'articles Arxiv contenant le terme *calibration* dans le titre pour les catégories *cs.AI*, *cs.CL*, *cs.LG*, *math.ST*, *stat.AP*, *stat.CO*, *stat.ML* calculés en août 2023

La demande croissante de garantie sur les modèles déployés a mis petit à petit l'accent sur les travaux autour de la calibration, comme on peut l'observer avec la figure 2.1. Nous avons besoin

de comprendre plus en profondeur ce que cette notion *est* et ce qui l’engendre. La recherche s’articule donc en deux axes majeurs :

1. Comment mesurer la distance à la calibration ?
2. Quel est le lien entre la minimisation d’une fonction de perte et la calibration ?

Nous allons nous intéresser à ces deux questions, mais ne traiterons pas l’ensemble des idées proposées. Plus précisément, la séance s’inspire du travail théorique publié en mars et mai 2023 par Jarosław Błasiok, Pariskshi Gopalan, Lunjia Hu et Preetum Nakkiran. Ils proposent un cadre mathématique rigoureux pour répondre à la première question, puis l’exploitent pour répondre à la seconde. Nous nous limiterons au cadre d’une classification binaire et ne traiterons pas l’ensemble des métriques proposées pour mesurer la distance à la calibration. Ainsi, nous ne présenterons pas toutes les notions introduites dans les papiers que nous citerons. Nous ne travaillerons que sur celles qui sont utiles pour comprendre les résultats que nous proposons d’étudier. Nous prendrons soin cependant de noter ces écarts à chaque fois que nous le jugerons nécessaire.

## 2.1 Comment mesurer la distance à la calibration ?

La question de la définition de la distance à la calibration a été le sujet de nombreux articles, qui ont chacun proposé des métriques différentes. Cependant, il est difficile de les relier et donc de les comparer entre elles. De plus, nombre d’entre elles ne parviennent pas à respecter des conditions que l’on souhaiterait intuitivement avoir d’une telle mesure.

L’objectif de l’article *A Unifying Theory of Distance from Calibration* [Błasiok et al., 2023a] publié en mars 2023 par Jarosław Błasiok, Pariskshi Gopalan, Lunjia Hu et Preetum Nakkiran est, comme son titre l’indique, de résoudre ces problèmes et définir un formalisme clair sur ces questions.

Avant de décrire le cheminement de l’article, commençons par formaliser le contexte. Nous avons accès à des observations  $(x, y)$  définies comme :

$$(x, y) \sim \mathcal{D} \quad \text{avec } \mathcal{D} \text{ une distribution sur } \mathcal{X} \times \{0, 1\}$$

Domaine discret, espace des inputs

On demande que  $\mathcal{X}$  soit discret, éventuellement très grand, pour éviter des difficultés théoriques<sup>1</sup>. Cette hypothèse est toujours vérifiée dans la pratique.

On appelle un *prédicteur* une fonction  $f : \mathcal{X} \rightarrow [0, 1]$  que l’on interprète comme une estimation de  $\mathbb{P}(y = 1|x)$ . C’est le résultat de l’entraînement de l’algorithme dont on souhaite mesurer la calibration. On considère la distribution induite par la paire prédiction-label  $(f(x), y) \in [0, 1] \times \{0, 1\}$  que l’on notera  $\mathcal{D}_f$ . Il ne nous reste plus qu’à définir la parfaite calibration comme [Dawid, 1982].

---

1. Les auteurs précisent que supposer que  $f$  soit mesurable devrait suffire pour traiter le cas où  $\mathcal{X}$  est infini.

**Définition 2.1** (Parfaite calibration). *On dit qu'une distribution  $\Gamma$  sur  $[0, 1] \times \{0, 1\}$  est parfaitement calibrée si, et seulement si, on a :*

$$\mathbb{E}_{(v,y) \sim \Gamma} [y|v] = v$$

*On dit qu'un prédicteur  $f$  est parfaitement calibré par rapport à  $\mathcal{D}$  si  $\mathcal{D}_f$  est parfaitement calibrée. On note  $\text{cal}(\mathcal{D})$  l'ensemble des prédicteurs parfaitement calibrés par rapport à  $\mathcal{D}$ .*

Remarquons que l'on peut également définir la parfaite calibration comme l'impossibilité de distinguer  $(f(x), y)$  et  $(f(x), y')$  où  $y' \sim \text{Ber}(f(x))$  quand on ne regarde que les prédictions ! En définissant la parfaite calibration, nous avons également défini l'ensemble des prédicteurs parfaitement calibrés. En sachant que l'on a imposé que  $\mathcal{X}$  soit fini, il est légitime de se poser la question pour  $\text{cal}(\mathcal{D})$ .

**Proposition 2.1.** *L'ensemble des prédicteurs parfaitement calibrés pour une distribution  $\mathcal{D}$  est fini.*

*Démonstration.* Pour montrer que  $\text{cal}(\mathcal{D})$  est fini, montrons qu'il existe une injection entre  $\text{cal}(\mathcal{D})$  et l'ensemble des partitions de  $\mathcal{X}$ .

Soit  $\mathcal{S} = \{S_i \mid \forall i \leq m\}$  une partition de  $\mathcal{X}$ . On considère :

$$\forall x \in S_i, g_{\mathcal{S}}(x) = \mathbb{E}[y \mid x \in S_i]$$

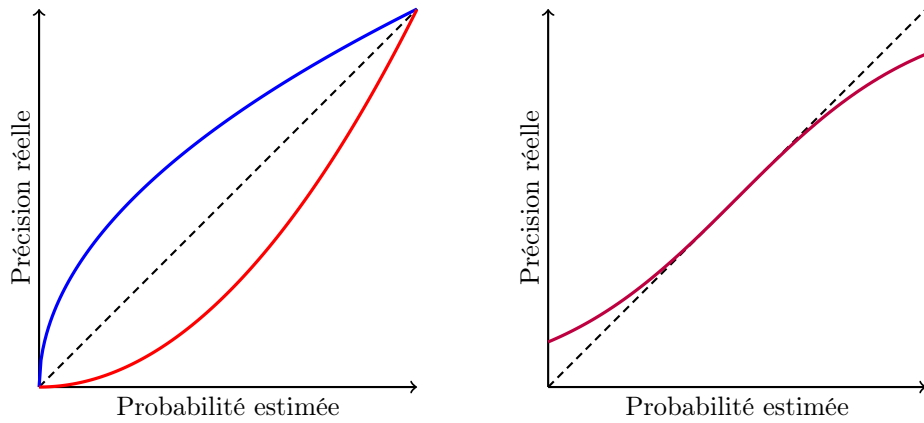
Autrement dit,  $g_{\mathcal{S}}$  renvoie la valeur moyenne de  $y$  pour chaque partitions  $(S_i)_{i \leq m}$  considérées. Par définition (2.1),  $g_{\mathcal{S}} \in \text{cal}(\mathcal{D})$ .

Soit  $f$  un prédicteur. Notons  $\text{level}(f)$  les niveaux de la fonction  $f$ , et partitionnons  $\mathcal{X}$  par cet ensemble. Alors,

$$f \in \text{cal}(\mathcal{D}) \iff f = g_{\text{level}(f)}$$

Nous venons de montrer que l'on peut lier un élément de  $\text{cal}(\mathcal{D})$  à un seul élément des partitions de  $\mathcal{X}$ , d'où l'injectivité. Puisque  $\mathcal{X}$  est fini, par injection,  $\text{cal}(\mathcal{D})$  est également fini.  $\square$

Il existe une fonction dans scikit-learn [Pedregosa et al., 2011] qui permet de visualiser la calibration d'un algorithme. En traçant la courbe de la probabilité réelle par rapport à celle estimée, avec en ligne de mire la droite  $y = x$  qui correspond à un prédicteur parfaitement calibré. Nous avons une manière visuelle de qualifier le comportement de notre algorithme. Voyons schématiquement les différents cas que l'on peut obtenir :



(a) Prédicteur **trop confiant** et **pas assez confiant**      (b) Prédicteur **non parfaitement calibré**

FIGURE 2.2 – Comparaison de prédicteurs par rapport à un prédicteur parfaitement calibré (en pointillés)

Il est important lorsqu'on publie un modèle qui sera amené à aider à la décision de connaître le comportement (confiant ou non) pour assurer une prise de décision saine. Il faut noter que ce graphique est présenté ici sans contexte : c'est une courbe générale qui peut se décliner dans autant de sous-domaines du dataset que l'on souhaite. Ainsi, quand on est amené à utiliser un algorithme qui opère sur des domaines variés, il peut être judicieux de présenter la calibration sur chaque domaine. Cela s'appelle la multicalibration, mais nous ne traiterons pas de cette notion dans ce cours.

### 2.1.1 Vrai distance à la calibration

Notre objectif est d'être capable de mesurer la distance à la calibration sans avoir à le faire visuellement. Nous serons donc amené à comparer les prédicteurs possibles entre eux. On considère la distance  $\ell_1$  sur cet ensemble :

Ensemble des prédicteurs  $f : \mathcal{X} \rightarrow [0, 1]$

$$\forall f, g \in \mathcal{F}_{\mathcal{X}}, \quad \ell_1(f, g) = \mathbb{E}_{\mathcal{D}} [|f(x) - g(x)|]$$

Notons que dans la définition précédente, nous calculons l'espérance par rapport à la distribution  $\mathcal{D}$  des observations  $(x, y)$  et non les distributions induites par  $f$  ou  $g$ . Si  $f$  et  $g$  sont distant alors  $D_f$  et  $D_g$  le sont aussi ?

**Exercice 2.1.** Soit  $\mathcal{X} = \{0, 1\}$  et on considère une distribution uniforme sur  $\mathcal{X}$ .

1. Quel est le meilleur prédicteur que l'on puisse construire ?
2. On considère  $f(x) = x$  et  $g(x) = 1 - x$  comme prédicteurs. Que dire de  $D_f$  et  $D_g$  ?
3. Calculer  $\ell_1(f, g)$ . Commentez.

*Solution.* On reprend les mêmes notations.



1. Le prédicteur constant valant 0.5 est l'estimateur bayésien.
2. Les prédicteurs  $f$  et  $g$  étant uniformes sur  $\mathcal{X}$ , les labels sont uniformes une fois conditionnés par  $f$  et  $g$ , donc  $\mathcal{D}_f$  et  $\mathcal{D}_g$  sont identiques.
3. On a par définition que  $\ell_1(f, g) = 1$  donc que  $f$  et  $g$  sont distants, bien qu'ils engendrent la même distribution paire-label.

□

La distance ainsi définie permet de vraiment cibler la qualité d'un prédicteur, et non la distribution induite, bien que la définition de calibration se fasse par celle-ci. Nous avons donc un moyen bien défini pour comparer deux prédicteurs. Ce qui veut dire que nous avons une manière de comparer tous les prédicteurs entre eux.

**Définition 2.2** (Vrai distance à la calibration). *Pour une distribution  $\mathcal{D}$  donnée et une famille  $\mathcal{F}_{\mathcal{X}}$  donnée, on définit la vrai distance à la calibration comme :*

$$\text{dCE}_{\mathcal{D}}(f) = \inf_{g \in \text{cal}(\mathcal{D})} \ell_1(f, g)$$

Cette distance serait une bonne candidate pour être une mesure de la distance à la calibration. Pourtant nous ne pouvons pas l'utiliser en pratique, voire la définir à cause du résultat suivant :

**Proposition 2.2.** *Soit  $\alpha \in [0, \frac{1}{2}]$ . Il existe un domaine  $\mathcal{X}$ , un prédicteur  $f \in \mathcal{F}_{\mathcal{X}}$  et deux distributions  $\mathcal{D}_f^1$  et  $\mathcal{D}_f^2$  telles que :*

- Les distributions  $\mathcal{D}_f^1$  et  $\mathcal{D}_f^2$  sont identiques
- $\text{dCE}_{\mathcal{D}_f^1}(f) \leq 2\alpha^2$  tandis que  $\text{dCE}_{\mathcal{D}_f^2}(f) \geq \alpha$

Nous ne rédigerons pas la preuve de cette proposition dans ce cours, mais elle est présentée dans l'article [Błasiok et al., 2023a]. Si l'on analyse les bornes, visualisées avec la figure (2.3), on s'aperçoit que nous n'avons pas des borne de dCE mais des prédictions contraires.

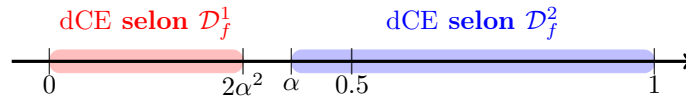


FIGURE 2.3 – Inégalités de la proposition (2.2)

Le coeur du problème soulevé est qu'un même prédicteur, pour des distributions indiscernables, peut avoir des distances à la calibration incohérentes si nous n'avons pas accès à  $x$ . Or nous avons fait le choix de travailler avec uniquement la paire prédiction-label  $(f(x), y)$ .

Comme dCE est intellectuellement simple à comprendre et correspond à ce que l'on cherche à obtenir, nous allons proposer des manières de calculer des approximations de cette distance dans le cadre que l'on s'est défini.

### 2.1.2 Mesure de calibration consistante

Pour cela, commençons par formaliser ce que l'on attend d'une mesure de calibration : une fonction qui renvoie un nombre entre 0 et 1 à partir d'une distribution  $\mathcal{D}$  d'observations et d'un prédicteur  $f$ . On note cette valeur  $\mu_{\mathcal{D}}(f)$ . Au minimum, on souhaite avoir les propriétés suivantes :

$$\begin{aligned} \mu_{\mathcal{D}}(f) &= 0 & \text{si } f \in (\mathcal{D}) & & (\text{Complétude}) \\ \mu_{\mathcal{D}}(f) &> 0 & \text{si } f \notin (\mathcal{D}) & & (\text{Correction}) \end{aligned}$$

Autrement dit, on demande à la mesure de calibration de valoir 0 quand le prédicteur  $f$  est parfaitement calibré par rapport à  $\mathcal{D}$ , et d'être strictement positif sinon. Ces termes viennent de la logique mathématiques : *Completeness* et *Soundness* en anglais. Concrètement, la complétude garantit que l'on peut prouver ce qui est vrai, et la correction que l'on ne peut pas prouver prouver quelque chose qui est faux.

Cela dit, on préférerait que la valeur soit petite si on est proche de la calibration parfaite, et grande si on en est loin, plutôt qu'une réponse binaire.

**Définition 2.3** (Mesure consistante de calibration). *Soit  $c \in \mathbb{R}_+$ , on dit qu'une mesure de calibration  $\mu$  satisfait la  $c$ -robuste complétude si :*

$$\exists a \in \mathbb{R}_+, \forall \mathcal{D} \sim [0, 1] \times \{0, 1\}, \forall f \in \mathcal{F}_{\mathcal{X}}, \quad \mu_{\mathcal{D}}(f) \leq a (\text{dCE}_{\mathcal{D}}(f))^c$$

*Soit  $s \in \mathbb{R}_+$ , on dit qu'une mesure de calibration  $\mu$  satisfait la  $s$ -robuste correction si :*

$$\exists b \in \mathbb{R}_+, \forall \mathcal{D} \sim [0, 1] \times \{0, 1\}, \forall f \in \mathcal{F}_{\mathcal{X}}, \quad \mu_{\mathcal{D}}(f) \geq b (\text{dCE}_{\mathcal{D}}(f))^s$$

*On dit que  $\mu$  est  $(c, s)$ -consistante si les deux conditions précédentes sont vérifiées.*

Si un prédicteur  $f$  est  $\varepsilon$  proche d'être parfaitement calibré alors par la robuste complétude on a que  $\mu_{\mathcal{D}}(f)$  est plus petit que  $\varepsilon^c$  et tend vers 0 quand  $\varepsilon$  tend vers 0, de même pour la robuste correction.

Quand la mesure de calibration de  $f$  vaut  $\eta \gg 0$  alors par la robuste correction est de l'ordre de  $\eta^s \gg 0$ .

Notons que dans la définition de consistance, nous n'avons pas pris en compte le fait que la mesure de calibration puisse être calculable en ayant accès uniquement aux prédictions et aux labels. Voyons ce que cela impose comme conditions.

**Corollaire 2.1.** *Soit  $\mu$  une mesure de calibration  $(c, s)$ -consistante que l'on peut calculer à l'aide des prédictions et des labels. Alors  $s \geq 2c$*

*Démonstration.* Il s'agit du corollaire de la proposition (2.2), nous reprenons donc ses notations. Par définition de la complétude robuste pour  $\mathcal{D}^1$  et la correction robuste pour  $\mathcal{D}^2$ , on a :

$$\begin{aligned} \exists a \in \mathbb{R}_+, \quad \mu_{\mathcal{D}^1}(f) &\leq a(2\alpha^2)^c \\ \exists b \in \mathbb{R}_+, \quad \mu_{\mathcal{D}^2}(f) &\geq b\alpha^s \end{aligned}$$

Puisque  $\mathcal{D}_f^1 = \mathcal{D}_f^2$  et que  $\mu$  est calculable à l'aide des prédictions et des labels on a que  $\mu_{\mathcal{D}^1}(f) = \mu_{\mathcal{D}^2}(f)$  donc :

$$b\alpha^s \leq a(2\alpha^2)^c \iff \left(\frac{b}{a}\alpha\right)^{\frac{s}{c}} \leq 2\alpha^2$$

On obtient clairement une contradiction quand  $\alpha$  tends vers zéro si  $\frac{s}{c} < 2$ .  $\square$

Nous avons restreint les mesures de calibration éligibles à travers les différents souhaits que nous avons décrit : consistance et capacité d'être calculées dans le cadre prédiction-label. Il nous faut maintenant en exhiber une ! Commençons par la manière la plus intuitive de mesurer la distance à la calibration.

**Définition 2.4** (Expected Calibration Error). *Soit  $\mathcal{D}$  une distribution sur  $[0, 1] \times \{0, 1\}$  et  $f : \mathcal{X} \rightarrow [0, 1]$  un prédicteur. On appelle Expected Calibration Error :*

$$ECE_{\mathcal{D}}(f) = \mathbb{E}_{\mathcal{D}} \left[ \left| \mathbb{E}_{\mathcal{D}}[y \mid f(x)] - f(x) \right| \right]$$

On saisit ce que l'ECE veut faire : mesurer la distance moyenne entre la valeur moyenne de  $y$  sachant  $f$  et la valeur de  $f$ . Si l'ECE est égale à zéro, alors le prédicteur est parfaitement calibré. Voyons si l'ECE est consistante.

| **Lemme 2.1.** *Soit  $\mathcal{S} = \text{level}(f)$ . Alors  $ECE(f) = \ell_1(f, g_{\mathcal{S}}) \geq \text{dCE}_{\mathcal{D}}(f)$*

*Démonstration.* On a que pour tout  $x \in f^{-1}(v)$ ,  $f(x) = v$  donc  $g_{\mathcal{S}}(x) = \mathbb{E}[y \mid v]$ . Ainsi, on a que :

$$\begin{aligned} \mathbb{E} \left[ \left| \mathbb{E}[y \mid f(x)] - f(x) \right| \right] &= \mathbb{E} [|f(x) - g_{\mathcal{S}}(x)|] \\ ECE(f) &= \ell_1(f, g_{\mathcal{S}}) \end{aligned}$$

Par définition de l'ECE et  $\ell_1$ . On obtient l'inégalité par la définition de dCE qui minimise la distance  $\ell_1$  parmi l'ensemble des prédicteurs calibré.  $\square$

Donc l'ECE vérifie la 1-robuste correction. Ce qui veut dire qu'avec l'ECE on ne peut pas prouver quelque chose de faux. Cependant elle ne vérifie pas la robuste complétude.

En effet, on considère une distribution uniforme sur  $\mathcal{X} = \{0, 1\}$  où le meilleur prédicteur est  $f^*(0) = 0$  et  $f^*(1) = 1$ . On considère le prédicteur  $f_{\varepsilon}$  avec  $\varepsilon \geq 0$  défini par  $f_{\varepsilon}(0) = \frac{1}{2} - \varepsilon$  et  $f_{\varepsilon}(1) = \frac{1}{2} + \varepsilon$ .

Notons que  $f_0(x) = \frac{1}{2}$  qui est parfaitement calibré. Cependant, on a par la définition de l'ECE que  $ECE_{\mathcal{D}}(f_{\varepsilon}) = \frac{1}{2} - \varepsilon$  donc  $ECE_{\mathcal{D}}(f_0) = \frac{1}{2}$  alors que  $f_0$  est parfaitement calibré !

Nous n'avons donc pas dans ce cas de continuité en 0 en plus d'une non fiabilité dans les valeurs par incomplétude. Autrement dit, il ne nous est pas possible de toujours prouver quelque chose qui est vrai. Nous venons de montrer que l'ECE n'est pas une mesure de calibration consistante bien qu'elle soit largement adopté par la communauté Machine Learning. Nous devons trouver une alternative qui réponde à nos *désiderata*.

### 2.1.3 Plus petite distance à la calibration

Avant de se lancer dans l'étude d'une nouvelle mesure de calibration, peut-être qu'il serait utile de travailler un peu plus autour de la distance dCE. Nous savons que nous ne pouvons pas l'exploiter directement, mais cette notion est centrale pour la consistance. Nous devons donc obtenir des garanties sur les estimations que nous pourrions faire. Pour cela nous allons définir la distance inférieure à la calibration<sup>2</sup>.

**Définition 2.5.** Soit  $\Gamma$  une distribution sur  $[0, 1] \times \{0, 1\}$ . On définit l'ensemble  $\text{ext}(\Gamma)$  comme l'ensemble des distributions  $\Pi$  des triplets  $(u, v, y) \in [0, 1] \times [0, 1] \times \{0, 1\}$  tel que :

- La distribution marginale de  $(v, y)$  est  $\Gamma$
- La distribution marginale de  $(u, y)$  est parfaitement calibré i.e.  $\mathbb{E}_{\Pi}[y|u] = u$ .

Cet ensemble de distributions n'est pas évident à sentir. Commençons par noter que les deux points ne sont *liés* que par  $y$ . Le premier représente une distribution que l'on observe, et le second représente une distribution que l'on souhaiterait obtenir. Finalement, nous aimerions que  $v$  et  $u$  soient le plus proche possible. Cette intuition est celle de la notion centrale de cette sous-section.

**Définition 2.6** (Distance inférieure à la calibration). On définit la distance inférieure à la calibration par :

$$\underline{\text{dCE}}(\Gamma) = \inf_{\Pi \in \text{ext}(\Gamma)} \mathbb{E}_{(u,v,y) \sim \Pi} [|u - v|]$$

Pour une distribution  $\mathcal{D}$  et un prédicteur  $f$ , on définit  $\underline{\text{dCE}}_{\mathcal{D}}(f) = \underline{\text{dCE}}(\mathcal{D}_f)$

Ce qui différencie cette notion de la définition de dCE est que la distributions est *liée* par  $y$  comme nous l'avons noté précédemment. Donc nous avons *accès* à toutes les informations possibles dans le cas où l'on observe  $f(x)$  et  $y$ . Cela règle le problème de dCE. Avec la notation, il semblerait que l'on puisse lier  $\underline{\text{dCE}}$  et dCE.

Puisque dCE traite de prédicteurs calibrés, remarquons que tout prédicteur  $g \in \text{cal}(\mathcal{D})$  génère une distribution  $\Pi \in \text{ext}(\mathcal{D}_f)$  où de  $(x, y) \sim \mathcal{D}$  on obtient  $(g(x), f(x), y)$ . En remarquant que  $\mathbb{E}_{(u,v,y) \sim \Pi} [|u - v|] = \ell_1(f, g)$ , il suffit de minimiser pour  $g \in \text{cal}(\mathcal{D})$  pour obtenir :

$$\underline{\text{dCE}}_{\mathcal{D}}(f) \leq \text{dCE}_{\mathcal{D}}(f)$$

Nous venons de justifier la notation et le nom de  $\underline{\text{dCE}}$ . Mais nous n'avons pas un lien suffisamment clair encore entre dCE et  $\underline{\text{dCE}}$ . Dans l'article [Błasiok et al., 2023a], via des raisonnements que nous ne traiterons pas ici, est obtenue l'inégalité suivante :

**Proposition 2.3.** Pour toute distribution  $\mathcal{D}$  et tout prédicteur  $f$ , on a :

$$\underline{\text{dCE}}_{\mathcal{D}}(f) \leq \text{dCE}_{\mathcal{D}}(f) \leq 4\sqrt{\underline{\text{dCE}}_{\mathcal{D}}(f)}$$

2. Dans l'article est également défini la distance supérieure à la calibration, mais nous n'en avons pas besoin dans le cadre de ce cours.

Cet encadrement ne permet pas une estimation précise de dCE mais informe que l'écart est au plus quadratique entre dCE et  $\underline{\text{dCE}}$ . Dans la théorie générale sur la calibration, cela reste vrai dans un grand nombre de cas. De plus, si une mesure de calibration est encadrée par la distance inférieure à la calibration, alors on sera capable de prouver la consistance de cette mesure en exploitant ce résultat. L'intérêt de la proposition (2.3) est plutôt dans la compréhension générale des notions que dans un but concret, pour le moment.

### 2.1.4 Smooth Calibration

Une alternative proposée à l'ECE est la smooth-calibration. Cette notion est un cas particulier d'une famille plus générale introduite dans l'article *Low-degree multicalibration* [Gopalan et al., 2022].

**Définition 2.7** (Calibration pondérée). *Soit  $W$  une famille de fonctions  $w : [0, 1] \rightarrow \mathbb{R}$ . La calibration pondérée d'une distribution  $\Gamma$  sur  $[0, 1] \times \{0, 1\}$  est définie comme :*

$$\text{wCE}^W(\Gamma) = \sup_{w \in W} \left| \mathbb{E}_{(v,y) \sim \Gamma} [(y - v)w(v)] \right|$$

*Étant donné une distribution  $\mathcal{D}$  sur  $\mathcal{X} \times \{0, 1\}$  et un prédicteur  $f : \mathcal{X} \rightarrow [0, 1]$ , on appelle la calibration pondérée de  $f$  par rapport à  $\mathcal{D}$  :*

$$\text{wCE}_{\mathcal{D}}^W(f) = \text{wCE}^W(\mathcal{D}_f) = \sup_{w \in W} \left| \mathbb{E}_{(x,y) \sim \mathcal{D}} [(y - f(x))w(f(x))] \right|$$

La définition précédente est cohérente avec la complétude : si  $\Gamma$  est parfaitement calibrée alors la valeur de  $\text{wCE}^W$  sera zéro. En effet si l'on conditionne par  $v$  :

$$\begin{aligned} \mathbb{E}_{\Gamma} [(y - v)w(v)] &= \mathbb{E}_{\Gamma} \left[ \mathbb{E} [(y - v)w(v) \mid v] \right] \\ &= \mathbb{E}_{\Gamma} \left[ w(v) \mathbb{E} [y \mid v] - v(w(v)) \right] \\ &= 0 \quad \text{parce que } \Gamma \text{ est parfaitement calibrée, par définition } \mathbb{E} [y \mid v] = v \end{aligned}$$

Une famille importante est celle des fonctions 1-Lipschitzienne bornées. Elle est pour la première fois introduite dans ce contexte dans l'article *Deterministic calibration and Nash equilibrium* [Kakade and Foster, 2004]. Nous suivrons cependant les notations de [Gopalan et al., 2022]. Pour rappel, une fonction  $w : [0, 1] \rightarrow [-1, 1]$  est 1-Lipschitzienne si elle vérifie :

$$\forall x, y \in [0, 1], \quad |w(x) - w(y)| \leq |x - y|$$

Cela contraint la vitesse d'évolution de la fonction  $w$  selon  $x$ . Autrement dit, on demande que  $w$  croisse au plus linéairement. Visuellement :

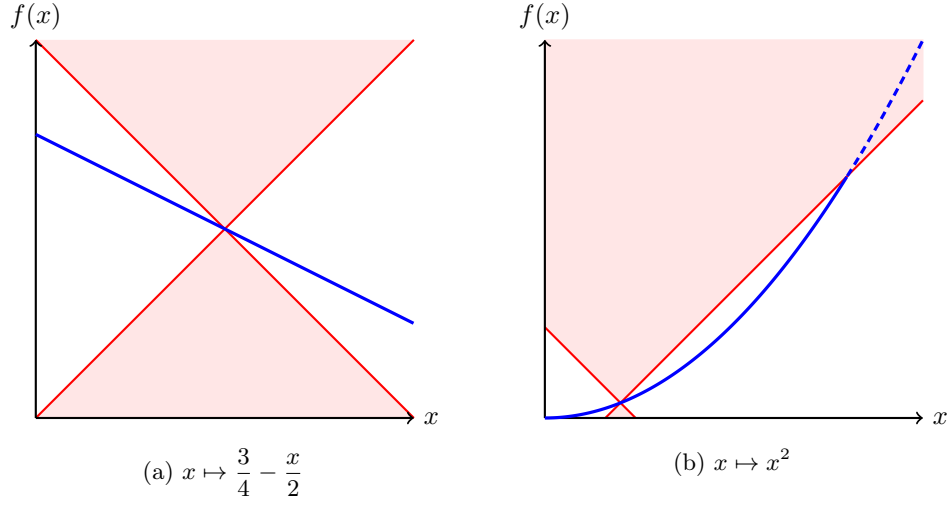


FIGURE 2.4 – Exemple et contre exemple de fonction 1-Lipschitzienne

La première fonction est même  $\frac{1}{2}$ -Lipschitzienne, alors que la seconde n'est pas 1-Lipschitzienne. Nous n'avons présenté ici que des fonctions à valeurs dans  $[0, 1]$ , mais l'intervalle  $[-1, 1]$  est autorisé par la définition<sup>3</sup>. Le rappel étant fait, nous pouvons définir la mesure smooth calibration comme cas particulier de la calibration pondérée.

**Définition 2.8** (Smooth calibration). *Soit  $L$  la famille des fonctions 1-Lipschitzienne  $w : [0, 1] \rightarrow [-1, 1]$ . L'erreur de smooth calibration d'une distribution  $\Gamma$  sur  $[0, 1] \times \{0, 1\}$  est définie comme :*

$$\text{smCE}(\Gamma) = \text{wCE}^L(\Gamma)$$

*De manière similaire, pour une distribution  $\mathcal{D}$  et un prédicteur  $f$  on définit :*

$$\text{smCE}_{\mathcal{D}}(f) = \text{smCE}(\mathcal{D}_f) = \text{wCE}^L(\mathcal{D}_f)$$

Il reste à se poser toujours la même question : smCE est-elle consistante ? Nous avons déjà montré que la calibration pondérée en général vérifie la complétude. Il reste à identifier avec quel coefficient et savoir si elle vérifie la correction robuste. Pour cela, lions la smooth-calibration à la plus petite distance à la calibration.

**Théorème 2.1.** *Soit une distribution  $\Gamma$  sur  $[0, 1] \times \{0, 1\}$ . Alors,*

$$\frac{1}{2} \underline{\text{dCE}}(\Gamma) \leq \text{smCE}(\Gamma) \leq 2 \underline{\text{dCE}}(\Gamma)$$

Nous avons écrit plus tôt que si nous étions en capacité de borner une mesure de calibration avec dCE alors nous serons capable de prouver la consistance. Nous avons un corollaire immédiat de cet encadrement :

3. Cette distinction fera encore plus sens dans la seconde partie du chapitre.

**Corollaire 2.2.** *La mesure de calibration  $\text{smCE}$  est  $(1, 2)$ -consistante.*

*Démonstration.* Soit  $\Gamma$  une distribution sur  $[0, 1] \times \{0, 1\}$ . Avec la partie droite de l'inégalité du théorème 2.1 :

$$\text{smCE}(\Gamma) \leq 2\text{dCE}(\Gamma) \leq 2 \times \text{dCE}(\Gamma)$$

D'où la 1-robuste complétude. Avec la partie gauche de l'inégalité du théorème 2.1 :

$$\text{smCE}(\Gamma) \geq \frac{1}{2}\text{dCE}(\Gamma)$$

En exploitant la partie droite de l'inégalité de la proposition 2.3 :

$$\text{dCE}_{\mathcal{D}}(f) \leq 4\sqrt{\text{dCE}_{\mathcal{D}}(f)}$$

En combinant les deux précédentes inégalités, on a :

$$\begin{aligned} \text{smCE}(\Gamma) \geq \frac{1}{2}\text{dCE}(\Gamma) &\implies \sqrt{\text{smCE}(\Gamma)} \geq \sqrt{\frac{1}{2}}\sqrt{\text{dCE}(\Gamma)} \\ &\implies 4\sqrt{\text{smCE}(\Gamma)} \geq \sqrt{\frac{1}{2}}\text{dCE}(\Gamma) \\ &\implies \text{smCE}(\Gamma) \geq \frac{1}{32}\text{dCE}(\Gamma)^2 \end{aligned}$$

D'où la 2-robuste correction et donc la  $(1, 2)$ -consistance.  $\square$

Remarquons que la condition imposée par le corollaire 2.1 est bien vérifiée et même optimale puisque  $\frac{s}{c} = 2$ . Il nous reste à prouver le théorème 2.1. Nous ne prouverons que la borne supérieure ici, la borne inférieure nécessitant beaucoup plus de matériel technique pour être traitée dans ce cours.

*Démonstration du théorème 2.1, borne supérieure.* Soit  $\Pi$  une distribution sur  $[0, 1] \times [0, 1] \times \{0, 1\}$  et soit  $w : [0, 1] \rightarrow [-1, 1]$  une fonction 1-Lipschitzienne. On a :

$$\begin{aligned} \left| \mathbb{E}_{(u,v,y) \sim \Pi} [(y-u)w(u)] - \mathbb{E}_{(u,v,y) \sim \Pi} [(y-v)w(v)] \right| &= \left| \mathbb{E}_{(u,v,y) \sim \Pi} [(y-u)w(u) - (y-v)w(v)] \right| \\ &\leq \mathbb{E}_{(u,v,y) \sim \Pi} [| (y-u)(w(u) - w(v)) |] + \mathbb{E}_{(u,v,y) \sim \Pi} [| (u-v)w(v) |] \\ &\leq 2 \mathbb{E}_{(u,v,y) \sim \Pi} [| (u-v) |] \end{aligned}$$

Parce que  $|y - u| \leq 1$  et par définition de  $w$ . Ainsi si  $\Pi \in \text{ext}(\Gamma)$  :

$$\begin{aligned} \left| \mathbb{E}_{(u,v,y) \sim \Pi} [(y-v)w(v)] \right| &\leq 2 \mathbb{E}_{(u,v,y) \sim \Pi} [|u - v|] + \left| \mathbb{E}_{(u,v,y) \sim \Pi} [(y-u)w(u)] \right| \\ &\leq 2 \mathbb{E}_{(u,v,y) \sim \Pi} [|u - v|] \end{aligned}$$

Puisque par définition de  $\text{ext}(\Gamma)$  la distribution de  $(u, y)$  est parfaitement calibrée. En prenant l'infimum sur  $\Pi \in \text{ext}(\Gamma)$  on obtient l'inégalité souhaitée.  $\square$

Au terme de cette section, nous savons définir une mesure de calibration consistante et en avons exhibé une que nous avons étudiée plus en détail. Il est important de noter que plusieurs autres mesures de calibration existent et que les principales ont été étudiées dans l'article référence de ce chapitre [Błasiok et al., 2023a]. Si l'on poursuit l'étude de cet article, on comprendra que le cadre théorique introduit permet d'étudier en profondeur les mesures proposées dans la littérature, et de les améliorer quand elles présentent des faiblesses.

Cependant, après ce travail pour qualifier la calibration, nous ne savons pas comment *l'obtenir*. Est-on capable d'avoir des garanties sur la calibration d'un algorithme ? Et si oui, quels sont les critères qui permettent de s'en assurer ? La question est vaste, et nous nous concentrerons dans la deuxième partie de ce chapitre sur le critère le plus intuitif qui pourrait être lié à la calibration.

## 2.2 Comment induire la calibration ?

Les algorithmes de classifications sont souvent interprétés comme des estimateurs de  $\mathbb{P}(y|x)$ . Ainsi, si l'optimisation de la fonction de perte est optimale, on devrait être proche de la calibration. Pour la régression logistique, qui a une interprétation naturelle avec les probabilités, ce lien semble évident comme on peut le lire dans le guide utilisateur de scikit-learn :

*Properly regularized logistic regression is well calibrated by default thanks to the use of the log-loss*

— Guide utilisateur scikit-learn (2007)

Cette affirmation n'est pas prouvée dans le guide, mais semble plausible. Si l'on généralise, l'utilisation d'une fonction de perte, peu importe l'algorithme, doit induire la même qualité de calibration à condition que l'optimal soit atteint. Cependant, en pratique, on observe qu'un réseau de neurones est souvent mieux calibré qu'une régression logistique sur les mêmes tâches alors qu'ils s'entraînent avec la même fonction de perte.

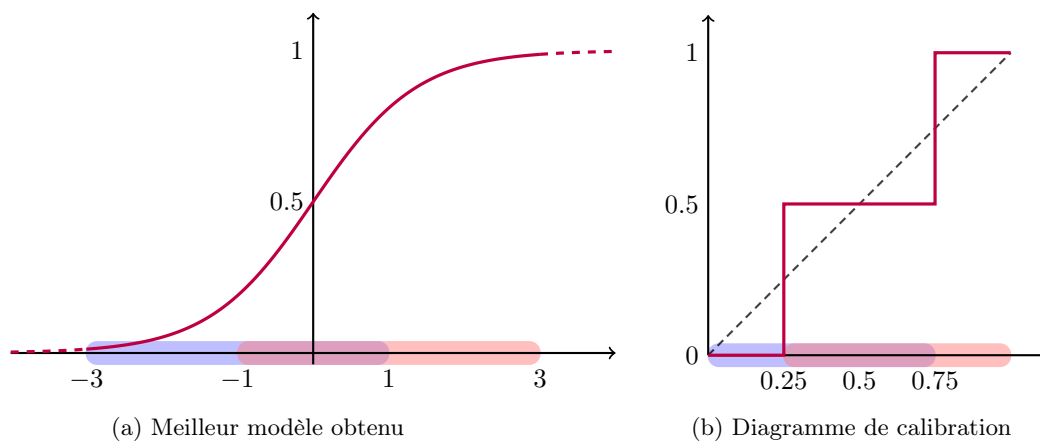


FIGURE 2.5 – Régression logistique pour la distribution  $(x, y) \sim \mathcal{D}$  avec  $y = 0$  pour  $x \sim \mathcal{U}([-3, 1])$  et  $y = 1$  pour  $x \sim \mathcal{U}([-1, 3])$

Avec l'exemple (2.5) on voit que l'on a réussi à obtenir le meilleur prédicteur en terme de minimisation de loss et pourtant, l'estimateur n'est pas calibré. Notons que le prédicteur constant de valeur 0.5 est parfaitement calibré. Notre intuition de départ serait plus correcte si nous



optimisations parmi l'ensemble des prédicteurs possibles. Or ce n'est pas ce que nous faisons : nous avons développé des algorithmes qui sont une partie des prédicteurs possibles. Dans l'exemple (2.5) nous avons choisi la régression logistique, mais il se pourrait qu'un autre algorithme réussisse à mieux combiner calibration et optimisation de la fonction de perte.

Le lien entre performance d'optimisation de la fonction de perte et calibration n'est finalement pas si évident et contre-intuitif. C'est ce travail qui a été réalisé avec l'article *When Does Optimizing a Proper Loss Yield Calibration?* [Błasiok et al., 2023b] publié en mai 2023, qui suit l'article *A Unifying Theory of Distance from Calibration* publié en mars 2023.

Comme dans la première section, nous ne présenterons pas l'ensemble de la théorie développée dans l'article, mais nous nous concentrerons sur les concepts centraux qui permettent d'avoir une piste solide pour apprécier le lien entre performance en fonction de perte et la calibration.

Commençons par éclaircir ce qu'on appelle *proper loss*. Dans l'ensemble de la section on considère une distribution  $\mathcal{D}$  sur  $\mathcal{X} \rightarrow \{0, 1\}$  que nous omettrons de préciser pour simplifier la lecture, même dans la notation de smCE par exemple. On considère un prédicteur  $f : \mathcal{X} \rightarrow [0, 1]$ , sauf indication contraire.

**Définition 2.9** (Fonction de perte bien définie). *Soit  $V \subseteq [0, 1]$  un interval non vide. On dit qu'une fonction de perte  $\mathcal{L} : \{0, 1\} \times [0, 1]$  est bien définie si pour tout  $v \in V$ , on a que :*

$$v \in \arg \min_{v' \in V} \mathbb{E}_{y \sim \text{Ber}(v)} [\mathcal{L}(y, v')]$$

Autrement dit, on demande qu'une fonction de perte permette effectivement d'atteindre le minimum que l'on cherche. On peut montrer que la *Squared loss*  $\mathcal{L}_{\text{sq}}$  est bien définie.

$$\forall y \in \{0, 1\}, \forall v \in V \subseteq [0, 1], \quad \mathcal{L}_{\text{sq}}(y, v) = (y - v)^2$$

Comme souvent en recherche, continuons d'étudier ce cas particulier pour voir ce que l'on pourrait obtenir quand on généralisera.

### 2.2.1 Cas particulier de la *Squared loss*

La notion centrale qu'introduit l'article est le *post-processing gap* :

**Définition 2.10** (Post-processing gap). *Soit  $K$  une famille de fonctions de post-processing  $\kappa : [0, 1] \rightarrow [0, 1]$  telle que la fonction de mise à jour  $\eta(v) = \kappa(v) - v$  soit 1-Lipschitzienne. On appelle post-processing gap de  $f$  par rapport à  $\mathcal{D}$  :*

$$\text{pGap}_{\mathcal{D}}(f) = \mathbb{E} [\mathcal{L}_{\text{sq}}(y, f(x))] - \inf_{\kappa \in K} \mathbb{E} [\mathcal{L}_{\text{sq}}(y, \kappa \circ f(x))]$$

Le post-processing gap mesure le gain que l'on peut obtenir en performance sur la loss si l'on modifiait les prédictions par :  $f(x) \mapsto f(x) + \eta(f(x))$  avec  $\eta$  une fonctions 1-Lipschitziennes. Le choix de la famille de fonction 1-Lipschitzienne, nous fait penser qu'on pourrait relier le post-processing gap avec la smooth calibration définie plus haut : c'est l'objet du théorème 2.2.

**Théorème 2.2.** Pour la fonction de perte  $\mathcal{L}_{\text{sq}}$  :

$$\text{smCE}(f)^2 \leq \text{pGap}_{\mathcal{D}}(f) \leq 2 \text{smCE}(f)$$

Ce résultat permet de lier la performance en terme de calibration à la performance en terme de fonction de perte. Plus précisément, à quel point on peut gagner si l'on modifie un peu les prédictions que l'on fait. Si l'on modifie très peu la performance alors pGap est faible et donc smCE l'est aussi donc nous avons une bonne calibration ! Inversement, si pGap est grand, c'est qu'en modifiant les prédictions on peut obtenir un vrai gain de performance. En utilisant la borne droite de l'inégalité de (2.2) on obtient une calibration de faible qualité.

Intuitivement, si nous donnions plus de *puissance* à notre algorithme, il pourrait être en mesure *d'apprendre* lui-même cette fonction de post-processing et ainsi optimiser encore plus ses performances. Cela nous amène à nous poser la question du choix de l'algorithme, paramètre que nous n'avons pas pris en compte pour le moment.

Avant d'intégrer cette réflexion dans l'analyse, le résultat précédent doit être plus général. En effet, cela ne fonctionne que pour la fonction de perte précisée. Prenons une autre fonction de perte, l'entropie croisée :

$$\mathcal{L}_{\text{xent}}(y, v) = - \left[ \underbrace{y \ln(v)}_{\text{Observation positive}} + \underbrace{(1-y) \ln(1-v)}_{\text{Observation négative}} \right]$$

On considère  $\mathcal{X} = \{x_0, x_1\}$  et  $\mathcal{D}$  une distribution sur  $\mathcal{X} \rightarrow \{0, 1\}$  telle que :

- $\mathbb{E}_{(x,y) \in \mathcal{D}} [y \mid x = x_0] = 0.1$
- $\mathbb{E}_{(x,y) \in \mathcal{D}} [y \mid x = x_1] = 0.9$

Soit  $f_\varepsilon : \mathcal{X} \rightarrow [0, 1]$  le prédicteur défini par  $f_\varepsilon(x_0) = \varepsilon$  et  $f_\varepsilon(x_1) = 1 - \varepsilon$  pour  $\varepsilon \in ]0, 0.1[$ .

Par la définition, on a que  $\lim_{\varepsilon \rightarrow 0} \text{smCE}(f_\varepsilon) = 0.05$ . Pourtant, puisque  $\lim_{\varepsilon \rightarrow 0} \mathcal{L}_{\text{xent}}(1, \varepsilon) = \lim_{\varepsilon \rightarrow 0} \mathcal{L}_{\text{xent}}(0, 1 - \varepsilon) = +\infty$  invalidant le résultat obtenu précédemment pour  $\mathcal{L}_{\text{sq}}$ .

Nous ne pouvons donc pas *simplement* remplacer la fonction de perte. Il faut trouver une approche plus générale, en regardant la notion de fonction de perte d'une manière *différente*.

## 2.2.2 Généralisation du lien entre pGap et smCE : par la dualité

Les travaux de Leonard Savage en 1971 [Savage, 1971] et ceux de Tilmann Gneiting et Adrian Raftery en 2007 [Gneiting and Raftery, 2007] ont construit un pont entre les fonctions de pertes bien définies et les fonctions convexes. Ce genre de pont est un exemple d'un grand concept : la dualité. On peut penser à la dualité onde-particule en physique quantique par exemple. Ici, plus modestement, nous allons traduire l'ensemble des notions précédentes dans le monde dual adapté. Le pont entre les deux mondes que nous considérerons se réalise via le résultat suivant :

**Proposition 2.4.** Soit  $V \subseteq [0, 1]$  un intervalle non vide. Soit  $\mathcal{L} : \{0, 1\} \times V \rightarrow \mathbb{R}$  une fonction de perte bien définie. On a :

$$\forall y \in \{0, 1\}, \forall v \in V, \quad \mathcal{L}(y, v) = \psi(\text{dual}(v)) - y \text{dual}(v)$$

*Fonction convexe  $\psi : \mathbb{R} \rightarrow \mathbb{R}$*

$\forall v \in V, \text{dual}(v) = \mathcal{L}(0, v) - \mathcal{L}(1, v)$

De plus, si  $\psi$  est différentiable, alors  $\nabla \psi(t) \in [0, 1]$

Cette proposition peut faire penser au théorème de représentation de Riesz : on explicite comment lier l'espace primal au dual. Dans notre cas, ce résultat indique que d'une fonction de perte bien définie s'induit une fonction convexe ainsi qu'une fonction qui relie une prédiction  $v \in V$  à sa prédiction duale  $\text{dual}(v) \in \mathbb{R}$ . Nous verrons plus concrètement avec l'exercice 2.2 comment cela se décline, mais avant traduisons une fonction de perte dans le monde dual.

**Définition 2.11** (Fonction de perte duale). Soit une fonction  $\psi : \mathbb{R} \rightarrow \mathbb{R}$ . On définit une fonction de perte  $\mathcal{L}^{(\psi)} : \{0, 1\} \times \mathbb{R} \rightarrow \mathbb{R}$  comme :

$$\forall y \in \{0, 1\}, \forall t \in \mathbb{R}, \quad \mathcal{L}^{(\psi)}(y, t) = \psi(t) - yt$$

Ainsi, si une fonction de perte  $\mathcal{L} : \{0, 1\} \times V \rightarrow \mathbb{R}$  satisfait les conditions de la proposition 2.4 pour  $V \subseteq [0, 1]$  un intervalle non vide et  $\text{dual} : V \rightarrow \mathbb{R}$ , alors :

$$\forall y \in \{0, 1\}, \forall v \in V, \quad \mathcal{L}(y, v) = \mathcal{L}^{(\psi)}(y, \text{dual}(v))$$

La définition de la fonction de perte duale met en lumière le rôle de la fonction dual introduite dans la proposition 2.4. Cette fonction permet de faire la traduction entre la prédiction  $v$  dans le monde réel et la prédiction  $t$  dans le monde dual. Notons que si  $v \in [0, 1]$ ,  $t$  peut lui valoir l'ensemble des valeurs réelles. Manipulons ces notions sur l'entropie croisée.

**Exercice 2.2** (Entropie croisée). On considère la fonction de perte entropie croisée définie comme :

$$\mathcal{L}_{\text{xent}}(y, v) = -y \ln(v) - (1 - y) \ln(1 - v)$$

On souhaite obtenir l'ensemble des informations nécessaires pour expliciter le mapping vers le monde dual. Identifier la fonction de perte duale,  $\psi$  et dual.

*Solution.* Par la définition de dual dans la proposition 2.4, on obtient que  $v = \sigma(t) = \frac{1}{1 + e^{-t}}$ . La fonction de perte duale de l'entropie croisée est la fonction de perte logistique :

$$\mathcal{L}^{(\psi)}(y, t) = \ln(1 + e^t) - yt$$

On a par définition de la fonction de perte duale que  $\psi(t) = \ln(1 + e^t)$ . On peut visualiser ces changements avec la figure (2.6). □

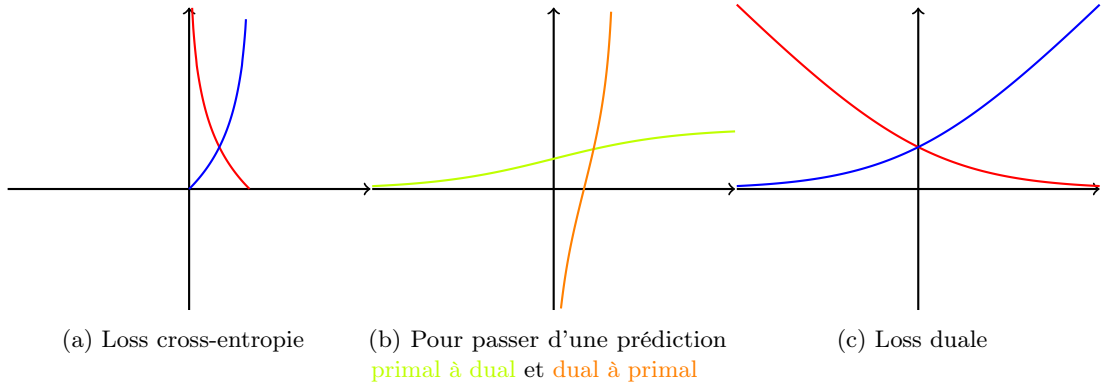


FIGURE 2.6 – Courbes des fonctions de pertes pour  $y = 0$  et  $y = 1$  ainsi que les fonctions de transferts

Rappelons que notre objectif est de généraliser les conclusions obtenue sur le cas particulier de la square loss. Puisque ce résultat impliquait la smooth calibration et le post processing gap du monde primal, nous devons définir ces notions dans le monde dual. Cette fois, les notions feront intervenir les prédictions duales. Pour un prédicteur  $f : \mathcal{X} \rightarrow V$ , ses prédictions duales sont :  $g(x) = \text{dual}(f(x))$  par rapport à la fonction de perte  $\mathcal{L}^{(\psi)}$ .

**Définition 2.12** (Gap de post-processing dual). On note  $K_\lambda$  pour  $\lambda > 0$  l'ensemble des fonctions de post-processing  $\kappa : \mathbb{R} \rightarrow \mathbb{R}$  telle qu'il existe une fonction  $\eta : \mathbb{R} \rightarrow [-\frac{1}{\lambda}, \frac{1}{\lambda}]$  qui est 1-Lipschitzienne et qui vérifie pour tout  $t \in \mathbb{R}$ ,  $\kappa(t) = t + \eta(t)$ .

On définit le gap de post-processing de  $g : \mathcal{X} \rightarrow \mathbb{R}$  :

$$\text{pGap}^{(\psi, \lambda)}(g) = \mathbb{E} \left[ \mathcal{L}^{(\psi)}(y, g(x)) \right] - \inf_{\kappa \in K_\lambda} \mathbb{E} \left[ \mathcal{L}^{(\psi)}(y, \kappa \circ g(x)) \right]$$

*Fonction de perte duale*

Notons que pour définir le gap de post-processing dual, nous n'avons eu qu'à décliner et revoir les ensembles de définitions des fonctions de la définition du gap de post-processing primal. L'adaptation de chaque ensemble de définition peut se deviner avec la figure (2.6) précédente.

**Définition 2.13** (Smooth calibration dual). On note  $H_\lambda$  pour  $\lambda > 0$  l'ensemble des fonctions  $\eta : \mathbb{R} \rightarrow [-1, 1]$  qui sont  $\lambda$ -Lipschitzienne. Soit  $\psi : \mathbb{R} \rightarrow \mathbb{R}$  une fonction différentiable avec pour tout  $t \in \mathbb{R}$ ,  $\nabla \psi(t) \in [0, 1]$ .

Pour une fonction  $g : \mathcal{X} \rightarrow \mathbb{R}$ , on définit le prédicteur  $f : \mathcal{X} \rightarrow [0, 1]$  tel que pour tout  $x \in \mathcal{X}$ ,  $f(x) = \nabla \psi(g(x))$ .

On appelle smooth calibration duale de  $g$  :

$$\text{smCE}^{(\psi, \lambda)}(g) = \sup_{\eta \in H_\lambda} \left| \mathbb{E} [(y - f(x)) \eta \circ g(x)] \right|$$

Nous pouvons faire le même commentaire que pour la définition du gap de post-processing

dual pour cette définition. Cette fois cependant nous apprenons comment d'un prédicteur dual on construit un prédicteur primal. Préciser que  $\nabla\psi(t) \in [0, 1]$  systématiquement permet à ce sous-gradient de faire le lien entre le prédicteur  $f$  du monde *réel* à  $g$  du monde dual. Avant de continuer les développements, nous devons nous assurer que nous sommes capables de relier la définition classique de la smooth calibration à celle du monde dual.

**Proposition 2.5.** Soit  $\psi : \mathbb{R} \rightarrow \mathbb{R}$  une fonction convexe différentiable qui vérifie pour tout  $t \in \mathbb{R}$ ,  $\nabla\psi(t) \in [0, 1]$ . Soit  $\lambda > 0$ , on suppose que  $\psi$  est  $\lambda$ -smooth :

$$\forall x, y \in \mathbb{R}, \quad |\nabla\psi(x) - \nabla\psi(y)| \leq \lambda|x, y|$$

Pour  $g : \mathcal{X} \rightarrow \mathbb{R}$  on définit  $f : \mathcal{X} \rightarrow [0, 1]$  telle que  $f(x) = \nabla\psi(g(x))$  pour  $x \in \mathcal{X}$ . On a :

$$\text{smCE}(f) \leq \text{smCE}^{(\psi, \lambda)}(g)$$

Ainsi, avoir une petite valeur de smooth calibration dans le monde dual garantit une plus petite valeur (ou égale) dans le monde réel. Il ne nous reste plus qu'à obtenir une inégalité dans le monde dual que l'on pourra exploiter dans le monde réel.

**Théorème 2.3.** Soit  $\psi : \mathbb{R} \rightarrow \mathbb{R}$  une fonction convexe différentiable qui vérifie pour tout  $t \in \mathbb{R}$ ,  $\nabla\psi(t) \in [0, 1]$ . Soit  $\lambda > 0$ , on suppose que  $\psi$  est  $\lambda$ -smooth. Alors pour toute fonction  $g : \mathcal{X} \rightarrow \mathbb{R}$  et toute distribution  $\mathcal{D}$  sur  $\mathcal{X} \times \{0, 1\}$  :

$$\frac{1}{2} \text{smCE}^{(\psi, \lambda)}(g)^2 \leq \lambda \text{pGap}^{(\psi, \lambda)}(g) \leq \text{smCE}^{(\psi, \lambda)}(g)$$

L'écart entre les deux notions que l'on observe est à nouveau au plus quadratique, comme pour  $\text{dCE}$  et  $\underline{\text{dCE}}$ . Il est précisé que les constantes sont optimales, nous ne présenterons pas les exemples qui le montrent.

Maintenant que nous avons une borne dans le monde dual, développons-là dans le monde réel :

Qualité de la calibration

$$\text{smCE}(f)^2 \leq 2\lambda \text{pGap}^{(\psi, \lambda)}(g)$$

Gain d'un post-processing

Nous avons lié la distance à la calibration d'un prédicteur à un gain potentiel de fonction de perte à la suite d'un post-processing par une fonction 1-Lipschitzienne. Ainsi, nous comprenons pourquoi il est difficile pour une régression logistique ou un arbre d'être parfaitement calibré : en plus d'apprendre la structure des données pour bien prédire, elle doit être capable de simuler une fonction 1-Lipschitzienne ! Cela est en revanche plus simple pour un réseau de neurones puisqu'une couche supplémentaire permettrait de le faire. Nous revenons à notre discussion sur le choix de l'algorithme dans la performance de calibration.

Cette remarque met encore plus l'accent sur l'importance de la profondeur dans un réseau de neurones ainsi que du rôle de la composition. Mais si l'on pousse la réflexion un peu plus loin, on peut se questionner sur l'impact de la méthode de descente de gradient. S'il suffit d'une nouvelle couche, peut-être suffit-il également d'une mise à jour supplémentaire des gradients par

SGD ? À ce stade nous n'avons pas la réponse, mais cela reste une piste de recherche.

Également, nous savons que l'implémentation de la régression logistique dans scikit-learn est accompagnée par défaut d'une régularisation  $\mathcal{L}_2$ . Il est montré dans l'article pour la fonction de perte *loss square* que l'on peut conserver les conclusions précédentes à partir du moment où la famille de prédicteurs est fermée pour la composition par les fonctions 1-Lipschitziennes et que la mesure de complexité n'évolue pas trop fortement avec la composition. Ce n'est pas le cas des arbres, SVM ou régressions logistiques par exemple alors que c'est le cas pour un réseau de neurones. Ce qui expliquerait pourquoi, même avec de nombreuses régularisations structurelles, ils peuvent être calibrés.

Nous avons présenté dans ce chapitre deux nouveaux exemples où l'intuition faillit à comprendre en profondeur une notion. À travers ce travail théorique autour de la définition de la distance à la calibration, nous avons compris en quoi l'ECE, bien qu'elle semble être la solution idoine, est incomplète. Partant de ce constat, nous avons proposé une version plus robuste. Nous avons également montré que la relation entre performance et calibration est plus complexe qu'elle n'y paraît. Cependant, comme discuté rapidement, la performance de calibration ne semble pas se limiter à la seule performance d'optimisation.

Cette question reste en suspens, comme les autres questions soulevées sans réponses dans ce cours. Elles, et probablement d'autres encore, nous guideront vers une compréhension plus globale et fine de ce domaine. Cela passera également par la notion de multi-calibration que nous avons rapidement évoquée : être calibré selon plusieurs sous-ensembles du domaine. Cette notion n'est pas traitée dans ce cours, déjà dense et pourtant incomplet, signe d'un domaine de recherche riche en enseignements.

## Séance 3

# Mise à jour moderne des poids

Dans le cadre supervisé, nous avons accès à un dataset  $\mathcal{D}$  définie comme :

$$\mathcal{D} = \left\{ (x_i, y_i) \mid \forall i \leq \underbrace{n}_{\text{Nombre d'observations}}, x_i \in \mathbb{R}^{\underbrace{d'}_{\text{Nombre d'informations}}}, y_i \in \mathcal{Y} \right\}$$

Avec  $\mathcal{Y} \subseteq \mathbb{R}$  pour un problème de régression et  $\mathcal{Y} \subset \mathbb{N}$  dans le cadre d'une classification. Nous suivrons ces notations, et les suivantes, sans rappeler systématiquement leurs significations tout au long du chapitre.

### 3.1 Quel schéma d'optimisation choisir ?

Les problèmes de Machine Learning supervisé peuvent souvent s'écrire sous la forme d'une optimisation d'une fonction de perte  $\mathcal{L} : \mathbb{R}^d \rightarrow \mathbb{R}_+$  comme suit :

$$\underbrace{\theta^*}_{\text{Vecteur des paramètres optimaux}} = \arg \min_{\theta \in \mathbb{R}^{\underbrace{d}_{\text{Dimension du vecteur de paramètres}}}} \mathcal{L}(\theta; X, y)$$

Dans la suite, pour simplifier les notations, nous omettrons la dépendance de  $\mathcal{L}$  en  $X$  et  $y$ . Notons qu'en général, nous avons  $d \neq d'$  et dans le cas du deep learning, très souvent  $d \gg d'$ .

La méthode la plus utilisée pour résoudre ce genre de problème est la descente de gradient :

$$\theta_{t+1} = \theta_t - \underbrace{\eta_t}_{\text{Learning rate}} \nabla \mathcal{L}(\theta_t)$$

Le learning rate est strictement positif et peut dépendre du temps : on dit que le learning rate suit un échancier<sup>1</sup>. Le learning rate est un hyperparamètre sensible : trop élevé on peut avoir

---

1. Nous avons détaillé quelques exemples dans la section ??

une divergence<sup>2</sup> et trop faible une convergence trop lente. Puisqu'il est fréquent de travailler avec des datasets volumineux, la base est découpée en plusieurs *batch* de taille fixe et on réalise une descente de gradient par batch : c'est ce qu'on appelle une descente de gradient stochastique. De multiples variantes de la descente de gradient existe, dont le but principal est d'accélérer la convergence. Prenons par exemple la descente de gradient avec *momentum* :

Paramètre du *momentum*

$$\begin{cases} v_{t+1} &= \beta v_t + (1 - \beta) \nabla \mathcal{L}(w_t) \\ w_{t+1} &= w_t - \eta v_{t+1} \end{cases} \quad (3.1)$$

Vecteur de vitesse

Avec cette version, on conserve dans l'update des poids la *tendance* de déplacement des poids dans l'espace des paramètres pour accélérer la descente. On peut le visualiser avec la figure (3.1).

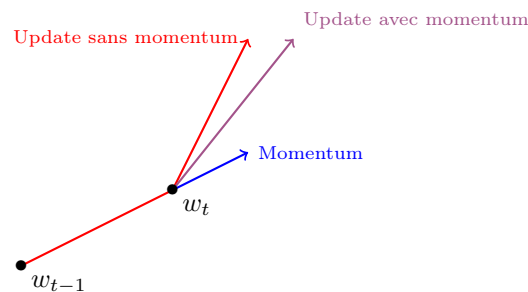


FIGURE 3.1 – Effet du momentum sur la mise à jour des poids

Cependant, accéléré ou non, la descente de gradient présente un problème indépendant de la valeur prise par le learning rate.

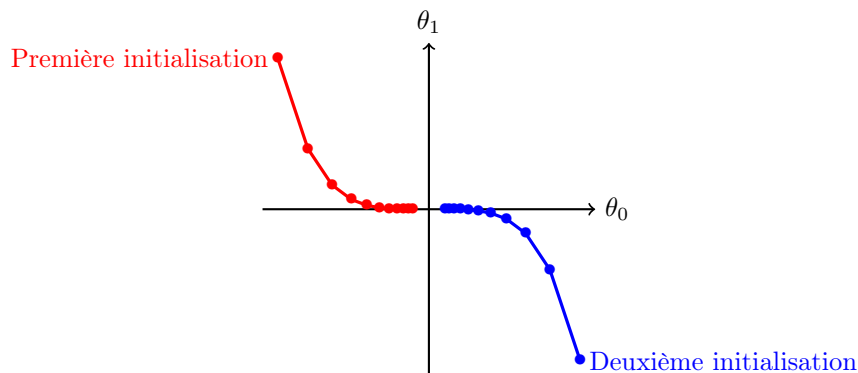


FIGURE 3.2 – 10 étapes de descente de gradient pour la fonction  $f(x, y) = x^2 + 3y^2$

Dans l'exemple de la figure (3.2), le gradient varie plus fortement avec  $y$  qu'avec  $x$  : cela entraîne une mise à jour des paramètres biaisés dans le sens de variation de  $y$ . Nous sommes

2. Exemple intéressant :  $f(x) = x^2$  avec  $\eta = 1$  et  $x_0 = 1$ .



arrivés à l'optimal pour  $y$  bien plus rapidement que pour  $x$ . Pour corriger ce comportement, est présenté en 2010 et publié en 2011 *Adaptive subgradient methods for online learning and stochastic optimization* [Duchi et al., 2011] par John Duchi, Elad Hazan et Yoram Singer. L'algorithme AdaGrad repose sur un suivi des gradients aux carrés pour normaliser la surface. La mise à jour des poids se fait donc comme suit, avec toutes les opérations vectorielle termes à termes :

$$\begin{cases} g_{t+1} = g_t + \nabla \mathcal{L}(\theta_t)^2 & \text{avec } g_0 = 0 \\ \theta_{t+1} = \theta_t - \eta \frac{\nabla \mathcal{L}(\theta_t)}{\sqrt{g_{t+1}} + \varepsilon} \end{cases} \quad (\text{AdaGrad})$$

Nombre pour éviter des problèmes numériques

L'intérêt de cette correction est qu'elle normalise l'apport de la mise à jour selon toutes les coordonnées. Voyons comment sur l'exemple précédent cela se comporte :

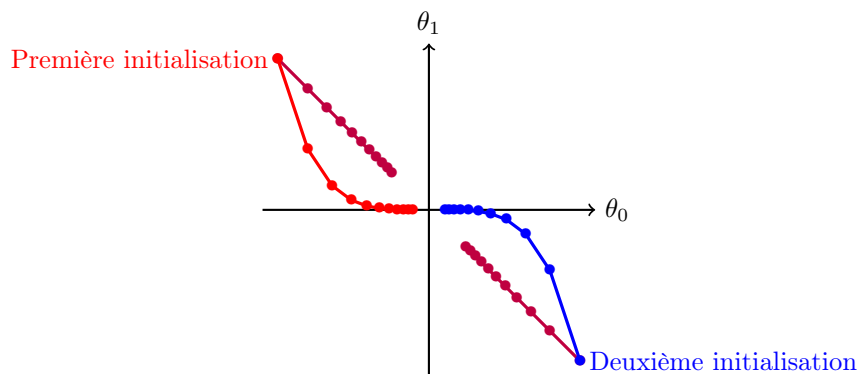


FIGURE 3.3 – Comparaison sur l'exemple précédent avec AdaGrad

La direction est cette fois non biaisé, comme prévu. Cependant il y a un ralentissement de la convergence alors que nous avons fournis à AdaGrad un learning rate deux fois supérieur à celui fourni à la descente de gradient classique pour cette simulation.

Si l'on rassemble le learning rate fixe que l'on fournit et la normalisation, on obtient bien un learning rate adaptatif, mais qui va de manière monotone décroître au fur et à mesure de l'entraînement. A la fin des dix époques précédentes, le learning rate adaptatif d'AdaGrad était divisé par 20 par rapport au départ : ce qui explique le ralentissement au fil de l'entraînement.

Ce défaut est corrigé en 2012 par Geoffrey Hinton, Nitish Srivastava et Kevin Swersky dans une slide du cours Deep Learning sur Coursera [Hinton et al., 2012a]. La méthode s'appelle RMSProp et vise à modifier AdaGrad pour être capable de faire décroître le cumul des gradients au carrés. Elle se décrit par :

$$\begin{cases} g_{t+1} = \alpha g_t + (1 - \alpha) \nabla \mathcal{L}(\theta_t)^2 & \text{avec } g_0 = 0 \\ \theta_{t+1} = \theta_t - \eta \frac{\nabla \mathcal{L}(\theta_t)}{\sqrt{g_{t+1}} + \varepsilon} \end{cases} \quad (\text{RMSProp})$$

Contrôle la mémoire des précédents gradients,  $\alpha \geq 0$

La méthode est très similaire à AdaGrad, seule la manière de conserver l'information des gradients précédents changent.

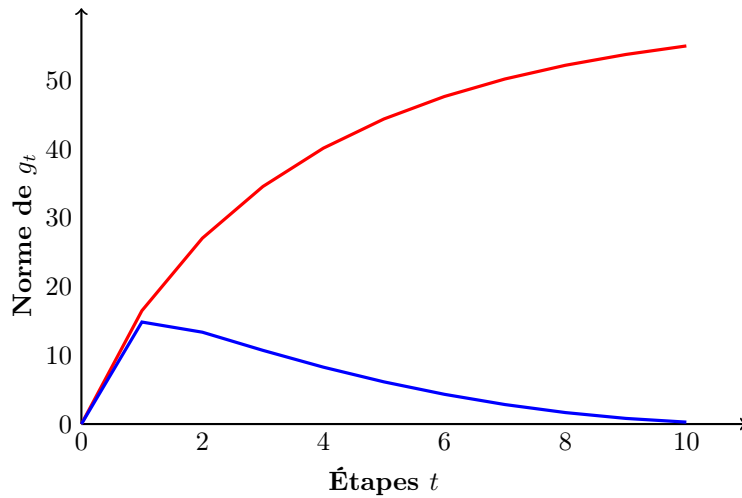


FIGURE 3.4 – Norme de  $g_t$  pour **RMSProp** et **AdaGrad**

Le comportement décrit dans les équations se visualise effectivement avec la figure (3.4) : pour RMSProp  $g_t$  est autorisé à *décroître* alors que pour AdaGrad la croissance est monotone. Continuons avec le même exemple que celui de la figure (3.2).

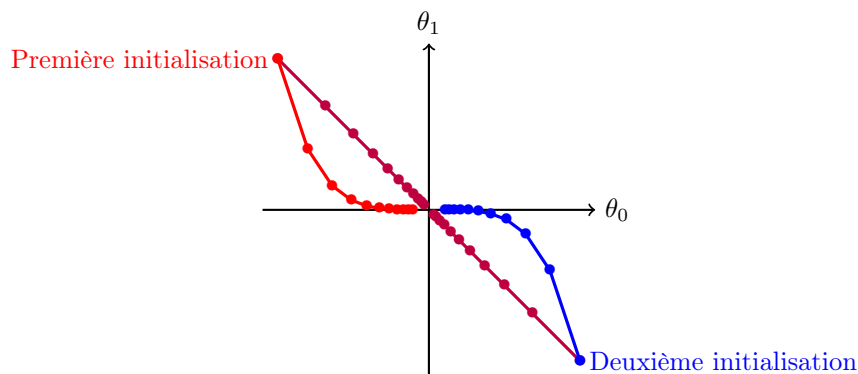


FIGURE 3.5 – Comparaison sur l'exemple précédent avec **RMSProp**

La convergence est cette fois beaucoup plus rapide pour le même learning rate que la descente de gradient classique. C'est essentiellement dû à cette correction de la direction. A nouveau, nous avons un learning rate adaptatif dans un schéma très proche d'AdaGrad. Ce qui change est la manière dont est conservé l'information.

Ces idées sont poussées à nouveau en 2014 par Diederik Kingma et Jimmy Ba en développant Adam [Kingma and Ba, 2015], l'optimiseur le plus utilisé dans le deep learning. Cette fois la mise à jour nécessite plus d'étapes :

$$\begin{cases} m_{t+1} = \beta_1 m_t + (1 - \beta_1) \nabla \mathcal{L}(\theta_t) & \text{avec } m_0 = 0 \\ \hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^{t+1}} \\ v_{t+1} = \beta_2 v_t + (1 - \beta_2) \nabla \mathcal{L}(\theta_t)^2 & \text{avec } v_0 = 0 \\ \hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^{t+1}} \\ \theta_{t+1} = \theta_t - \eta \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1} + \epsilon}} \end{cases} \quad (\text{Adam})$$

On conserve le momentum sur le carré du gradient et on l'applique au gradient simple également. De cette manière, nous avons une version accélérée de RMSProp. Cependant, puisque nous avons forcé l'initialisation des compteurs  $(m_t)_{t \leq T}$  et  $(v_t)_{t \leq T}$  nous avons un biais que l'on corrige avec les compteurs  $(\hat{m}_t)_{t \leq T}$  et  $(\hat{v}_t)_{t \leq T}$

**Exercice 3.1** (Correction du biais). *On reprend les notations précédentes. On s'intéresse ici à la correction que l'on doit apporter à la suite  $(v_t)_{t \leq T}$ . On suppose que les gradients sont identiquement distribués selon une loi de probabilité.*

1. Soit  $t \leq T$  une étape. Écrire  $v_t$  en fonction uniquement de  $\beta_2$  et des  $\mathcal{L}(\theta_i)$  pour  $i \leq t$ .
2. Calculer  $\mathbb{E}[v_t]$  avec l'expression obtenue précédemment.
3. En comparant  $\mathbb{E}[v_t]$  avec  $\mathbb{E}[g_t^2]$ , conclure sur la correction à apporter à  $v_t$ .

*Solution.* Soit  $t \leq T$ , on reprend les notations et hypothèses précédentes.

1. Par définition de  $v_t$ , on a :

$$\begin{aligned} v_t &= \beta_2 v_{t-1} + (1 - \beta_2) \mathcal{L}(\theta_t)^2 \\ &= \beta_2^2 v_{t-2} + (1 - \beta_2) [\beta_2 \mathcal{L}(\theta_{t-1})^2 + \mathcal{L}(\theta_t)^2] \\ &= \beta_2^t v_0 + (1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} \mathcal{L}(\theta_i)^2 \end{aligned}$$

2. En prenant l'expression précédente, et parce que les gradients sont identiquement distribués, on obtient :

$$\begin{aligned} \mathbb{E}[v_t] &= (1 - \beta_2) \mathbb{E}[\mathcal{L}(\theta_t)^2] \sum_{i=1}^t \beta_2^{t-i} \\ &= \mathbb{E}[\mathcal{L}(\theta_t)^2] (1 - \beta_2^t) \end{aligned}$$

D'où la correction présentée.

□

La preuve est analogue pour  $m_t$ . En intégrant cette correction de biais de l'initialisation, on s'assure que les premières itérations de l'algorithme seront correcte : nous avons montré à quel

point une initialisation de réseau de neurones compte dans la performance finale de l'algorithme. Avec Adam, nous avons cette fois deux hyperparamètres à régler, chacun ayant une signification claire. L'un contrôle la mémoire que l'on veut avoir pour des précédentes direction de gradient et l'autre la mémoire que l'on veut avoir des précédentes valeurs du gradient au carré. La méthode semble intuitive mais requiert trois hyperparamètres principaux en comptant le learning rate. Pour guider la valeur de ce dernier, on peut s'appuyer sur une interprétation probabiliste.

**Exercice 3.2** (Déplacement dans l'espace des paramètres). *On reprend les notations précédentes. On considère de plus une variable aléatoire  $X$  telle que  $\mathbb{E}[X]$  existe et  $\mathbb{E}[X^2]$  existe et n'est pas nulle.*

1. A l'aide de l'inégalité de Jensen, montrer que l'on a :  $\frac{|\mathbb{E}[X]|}{\sqrt{\mathbb{E}[X^2]}}$
2. On rappelle que  $\hat{m}_t$  est un estimateur non biaisé de  $\mathbb{E}[\nabla \mathcal{L}(\theta_t)]$  et que  $\hat{v}_t$  est un estimateur non biaisé de  $\mathbb{E}[\nabla \mathcal{L}(\theta_t)^2]$ . On définit  $\Delta_t = \theta_t - \theta_{t-1}$ . Exploiter le résultat précédent pour déduire une interprétation de la valeur du learning rate pour Adam avec une heuristique.

*Solution.* Nous reprenons les notations de l'exercice.

1. Puisque la fonction  $x \mapsto x^2$  est une fonction convexe, avec l'inégalité de Jensen on a :

$$\mathbb{E}[X^2] \geq \mathbb{E}[X]^2 \iff \frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]} \leq 1 \iff \frac{|\mathbb{E}[X]|}{\sqrt{\mathbb{E}[X^2]}}$$

2. Par définition de  $\Delta_t$  et en suivant un raisonnement approximatif :

$$\begin{aligned} \Delta_t &= \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}} \quad \text{donc} \quad |\Delta_t| = \eta \frac{|\hat{m}_t|}{\sqrt{\hat{v}_t}} \\ |\Delta_t| &\sim \eta \frac{|\mathbb{E}[\mathcal{L}(\theta_t)]|}{\sqrt{\mathbb{E}[\mathcal{L}(\theta_t)]^2}} \quad \text{ainsi} \quad |\Delta_t| \lesssim \eta \end{aligned}$$

□

Avec l'exercice précédent, on comprend que la distance parcourue dans l'espace des paramètres par une mise à jour de  $\theta_t$  est approximativement bornée par le learning rate. Dans la pratique, les valeurs de learning rate choisi sont de l'ordre de  $10^{-4}$ , c'est une bonne valeur pour commencer les essais. Adam est l'optimiseur utilisé pour entraîner les modèles GPT d'Open AI<sup>3</sup>[Brown et al., 2020], LaMDA de Google [Thoppilan et al., 2022] et GOPHER de Deepmind [Rae et al., 2021] par exemple. En dehors des transformers, Adam reste une valeur sûre pour l'entraînement de réseau de neurones.

---

3. Du moins jusqu'à la version 3 puisque nous n'avons pas d'informations sur la version 4.

De nombreuses autres possibilités existent, et chaque année plusieurs optimiseurs sont proposés. C'est pourquoi en 2021 Robin Schmidt, Frank Schneider et Philipp Hennig propose de comparer équitablement quinze optimiseurs sur différentes tâches [Schmidt et al., 2021]. Il en ressort deux informations :

1. Adam est l'optimiseur le plus performant sur le plus de tâches, sans pour autant être clairement supérieur. RMSProp et l'accélération de Nesterov restant des alternatives très intéressantes.
2. Choisir les meilleurs hyperparamètres est tout aussi efficace voire plus efficace que de changer d'optimiseur. Une bonne intuition de l'effet que pourra avoir l'optimiseur sur le dataset et l'architecture construite peut donc vraiment changer la donne.

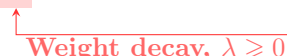
Le plus important selon eux est le constat suivant :

*Perhaps the most important takeaway from our study is hidden in plain sight : the field is in danger of being drowned by noise. Different optimizers exhibit a surprisingly similar performance distribution compared to a single method that is re-tuned or simply re-run with different random seeds. It is thus questionable how much insight the development of new methods yields, at least if they are conceptually and functionally close to the existing population.*

— Robin Schmidt, Frank Schneider et Philipp Hennig (2021)

Nous ne traiterons donc pas l'ensemble des variantes existantes et développées récemment. Nous pouvons cependant présenter une approche nouvelle qui fait écho à l'approche présenté dans la section ?? . En 2023 une équipe de chercheur de Google et UCLA propose Lion dans l'article *Symbolic discovery of optimization algorithms* [Chen et al., 2023]. Cet optimiseur s'écrit avec nos notations sous la forme :

$$\begin{cases} c_{t+1} &= \beta_1 m_t + (1 - \beta_1) \nabla \mathcal{L}(\theta_t) \quad \text{avec } m_0 = 0 \\ m_{t+1} &= \beta_2 m_t + (1 - \beta_2) \nabla \mathcal{L}(\theta_t) \\ \theta_{t+1} &= \theta_t - \eta (\text{sign}(c_t) + \lambda \theta_t) \end{cases} \quad (\text{Lion})$$


 Weight decay,  $\lambda \geq 0$

Lion ne travaille pas avec l'amplitude des gradients mais seulement le signe. On a également deux hyperparamètre  $\beta_1$  et  $\beta_2$  qui n'ont pas un rôle aussi clair que pour Adam. L'approche globale reste très proche de celle d'Adam.

Finalement on remarque la présence d'un terme  $\lambda$  nommé *Weight decay* : cette notion est l'objet de la prochaine section.

## 3.2 Comment régulariser les poids ?

En 1991 Anders Krogh et John Hertz publie *A simple weight decay can improve generalization* [Krogh and Hertz, 1991]. La conclusion est qu'ajouter une pénalisation à la fonction de perte qu'on minimise va forcer le réseau de neurones à choisir une solution avec un *petit* vecteur. Dans ce cas, le réseau de neurones sera plus à même de ne pas sur-apprendre ! C'est avec cette idée en tête que de nombreuses techniques de régularisations ont été développées. Les auteurs faisaient référence à la régularisation  $L_2$ , qui est traditionnellement la première solution enseignée pour régulariser un réseau de neurones. L'idée vient de la régression Ridge : en ajoutant cette contrainte, on

augmente le biais mais on diminue la variance. C'est exactement ce que l'on cherche. L'utilisation de la pénalité  $L_2$  devient donc incontournable dans la boîte à outil des régularisations d'un réseau de neurones. Nous avons déjà cité le dropout [Srivastava et al., 2014].

Dans cette section nous présentons un autre incontournable de la régularisation en étudiant le *weight decay*, le gradient clipping et la régularisation hypersphérique. Nous verrons que le *weight decay* énoncé dans le titre de l'article de 1991 est différent de la notion moderne.

### 3.2.1 Weight Decay

En 2017 Ilya Loshchilov et Frank Hutter publie l'article *Decoupled weight decay regularization* [Loshchilov and Hutter, 2019] et montre que l'on n'obtient pas toujours le comportement attendu par la régularisation  $L_2$ . Nous avons présenté à la section précédente plusieurs manières de réaliser la descente de gradient. Chacune de ces méthodes exploitent des informations du passé. On comprend donc que la régularisation  $L_2$  va donc se propager également la tendances. Vérifions.

**Exercice 3.3** (Régularisation  $L_2$  et momentum). On reprend les notations définies précédemment. On note la fonction de perte régularisée  $\mathcal{L}_\lambda(w) = \mathcal{L}(w) + \frac{\lambda}{2}\|w\|_2^2$  avec  $\lambda \geq 0$ .

1. Donner l'équation d'actualisation des poids d'un réseau de neurones avec une descente de gradient classique pour la fonction de perte  $\mathcal{L}_\lambda$  en fonction de la fonction de perte  $\mathcal{L}$ .
2. Montrer que l'équation d'actualisation des vecteurs  $(v_t)_{t \in \mathbb{N}}$  dans le cadre d'une descente de gradient avec momentum  $\beta$  pour la fonction de perte  $\mathcal{L}_\lambda$  en fonction de la fonction de perte  $\mathcal{L}$  est :

$$v_{t+1} = \beta v_t + (1 - \beta) [\nabla \mathcal{L}(w_t) + \lambda w_t]$$

3. En déduire l'équation de mise à jour des poids d'une descente de gradient avec momentum pour la fonction de perte  $\mathcal{L}_\lambda$  en fonction de la fonction de perte  $\mathcal{L}$ .

*Solution.* On reprend les mêmes notations :  $\lambda \geq 0$  le paramètre de la régularisation  $L_2$  et  $\beta \geq 0$  le paramètre du momentum.

1. On a pour une descente de gradient classique :

$$\begin{aligned} w_{t+1} &= w_t - \eta_t \nabla \mathcal{L}_\lambda(w_t) && \text{par définition de la descente de gradient (??)} \\ &= w_t - \eta_t \nabla \mathcal{L}(w_t) - \eta_t \lambda w_t && \text{par définition de } \mathcal{L}_\lambda \end{aligned}$$

2. Pour une descente de gradient avec momentum, on a cette fois :

$$\begin{aligned} v_{t+1} &= \beta v_t + (1 - \beta) \nabla \mathcal{L}_\lambda(w_t) && \text{par définition (3.1)} \\ &= \beta v_t + (1 - \beta) [\nabla \mathcal{L}(w_t) + \lambda w_t] && \text{par définition de } \mathcal{L}_\lambda \end{aligned}$$

3. En utilisant la question précédente, on a :

$$\begin{aligned} w_{t+1} &= w_t - \eta_t (\beta v_t + (1 - \beta) [\nabla \mathcal{L}(w_t) + \lambda w_t]) \\ &= w_t - \eta_t \beta v_t - \eta_t (1 - \beta) \nabla \mathcal{L}(w_t) - \eta_t (1 - \beta) \lambda w_t \end{aligned}$$

Au lieu de régulariser de manière classique avec le terme  $\eta_t \lambda w_t$ , quand on ajoute une accélération nous avons le terme  $\eta_t(1 - \beta)\lambda w_t$  qui prend en compte l'accélération. Ainsi la régularisation induite est complètement différente de celle que l'on souhaitais avoir.  $\square$

En observant donc ce problème, la régularisation  $L_2$  pour des schéma adaptatif régularise *trop*. L'article d'Ilya Loshchilov et Frank Hutter propose donc la solution *weight decay* : rajouter le terme  $\eta_t \lambda w_t$  avec  $\lambda \geq 0$  lors de l'actualisation des poids. C'est équivalent à une régularisation  $L_2$  dans le cas d'une descente de gradient classique (voir question 1 de l'exercice 3.3). Voyons ce que cela change lorsqu'on a un schéma adaptatif.

En prenant la stratégie de l'exercice précédent, on a que la mise à jour des vecteurs  $(v_t)_{t \in \mathbb{N}}$  est inchangé dans notre cadre puisque la fonction de perte n'est pas modifié. Ainsi, le seul changement a lieu pour l'apprentissage des poids.

$$\begin{aligned} w_{t+1} &= w_t - \eta_t v_{t+1} - \eta_t \lambda w_t && \text{par définition du } \textit{weight decay} \\ &= w_t - \eta_t [\beta v_t + (1 - \beta) \nabla \mathcal{L}(w_t)] - \eta_t \lambda w_t && \text{par définition de } v_{t+1} \\ &= w_t - \eta_t \beta v_t - \eta_t (1 - \beta) \nabla \mathcal{L}(w_t) - \eta_t \lambda w_t \end{aligned}$$

Cette dernière équation est à mettre en comparaison avec la mise à jour des poids pour une régularisation  $L_2$  d'une descente de gradient avec momentum :

$$w_{t+1} = w_t - \eta_t \beta v_t - \eta_t (1 - \beta) \nabla \mathcal{L}(w_t) - \eta_t (1 - \beta) \lambda w_t$$

En découplant la régularisation de la fonction de perte pour modifier directement la mise à jour des poids, on évite de **propager l'accélération** induite par les schéma adaptatif à la régularisation. Ainsi, on a un comportement plus cohérent de descente de gradient. Donc dès que l'on utilise les méthodes adaptatives, il ne faut pas utiliser de régularisation  $L_2$  mais préférer le *weight decay* pour atteindre le même objectif.

Nous avons étudié l'optimiseur Adam, couplé au weight decay nous obtenons l'optimiseur nommé AdamW qui n'est rien d'autre qu'Adam avec la régularisation weight decay ! Cette version est centrale dans les architectures des LLM, citons par exemple les modèles LLaMa de Meta [Touvron et al., 2023a, Touvron et al., 2023b, Dubey et al., 2024], Chinchilla et GATO de DeepMind [Hoffmann et al., 2022, Reed et al., 2022]. AdamW est l'optimiseur qui a servit d'input à la recherche symbolique d'un nouvel optimiser par l'équipe de Google qui a découvert Lion. On comprend donc mieux que le terme de weight decay soit apparu dans son expression.

Cependant, à ce jour les gains de Lion ne sont pas systématique par rapport à AdamW. Il est noté une dépendance selon le batch size (plus il est grand plus Lion serait performant). Si l'augmentation<sup>4</sup> de la data n'est pas trop forte il semblerait également qu'il y ait un gain à exploiter Lion.

Finalement, il n'y a pas à ce jour d'utilisation générale de Lion, il s'agit plutôt de chercheurs qui teste cette nouvelle méthode. En revanche, l'adoption du weight decay, elle, est bien présente !

---

4. Data augmentation classique : rotations, zoom, ...

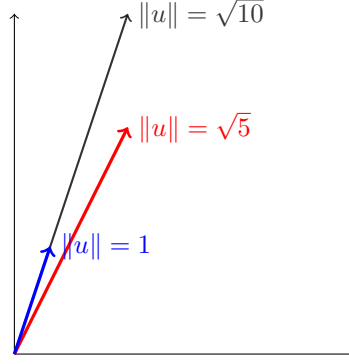


FIGURE 3.6 – Gradient initial, **gradient** après clipping et **gradient** après clipping par norme

### 3.2.2 Régularisation hyper-sphérique

Le weight decay cherche à régulariser chaque neurone séparément, en assurant que chacun trouve la meilleure balance entre petite norme et performance globale du réseau. Nous pouvons donc nous poser la question de régulariser les neurones *ensemble* ! Une approche possible est celle de la régularisation hyper-sphérique. [Liu et al., 2018] [Tan et al., 2022]

### 3.2.3 Gradient clipping

Comme exploré dans le chapitre 1, le phénomène d’explosion et de disparition des gradients n’a pas de réponse parfaite, mais on peut prendre un ensemble de mesure pour en prévenir ces effets. Puisque nous travaillons avec des réseaux de neurones dont la surface de perte est hautement non convexe, il est possible que l’on propage des gradients de très grande ampleurs qui peuvent conduire à nouveau à ces effets indésirable. Jusqu’ici nous avons présenté des techniques qui vont contrôler la norme des vecteurs de poids. Une autre manière de limiter l’explosion des gradients serait de contraindre les gradients à ne pas dépasser une certaine valeur : c’est l’idée du *gradient clipping*. On peut le faire de manière brute en imposant que chaque coordonnées, indépendamment des autres, ne dépasse un seuil  $\alpha$ . Si c’est le cas, alors on lui impose de valoir  $\alpha$ . Cela comporte un problème évident visualisé avec la figure (3.6) :

Si l’on impose que les coordonnées soient comprise entre  $[-\alpha, \alpha]$  indépendamment des autres coordonnées, on risque de ne plus conserver la *direction* du vecteur. Alors que si l’on modifie le gradient de la manière suivante :

$$\tilde{g} = \begin{cases} \alpha \frac{g}{\|g\|} & \text{si } \|g\| \geq \alpha \\ g & \text{sinon} \end{cases}$$

Alors on réalise un gradient clipping par norme qui préserve la direction du vecteur. C’est cette version du gradient clipping qui est proposé en 2013 par Razvan Pascanu, Tomas Mikolov et Yoshua Bengio dans l’article *On the difficulty of training recurrent neural network* [Pascanu et al., 2013]. L’article se basant avant tout sur les réseaux de neurones récurrent qui sont très sensible aux phénomènes d’explosion des gradients<sup>5</sup>.

5. On trouve une preuve de cette affirmation en fin d’article, après la bibliographie.



Cependant, si l'on a une coordonnée qui est très grande par rapport au reste, alors on risque d'avoir le reste des coordonnées qui deviennent très petite. Ainsi, le vecteur résultant aura du mal à faire progresser dans chaque direction indiquent l'apprentissage des différents poids. Il s'agit d'un problème que nous avons résolu avec RMSProp (et donc Adam).

Il n'y a pas une approche qui est meilleure que l'autre, chacune a ses défauts et avantages, et il faut explorer les deux méthodes.

Voyons sur un exemple très simple l'intérêt du gradient clipping avec la figure 3.7

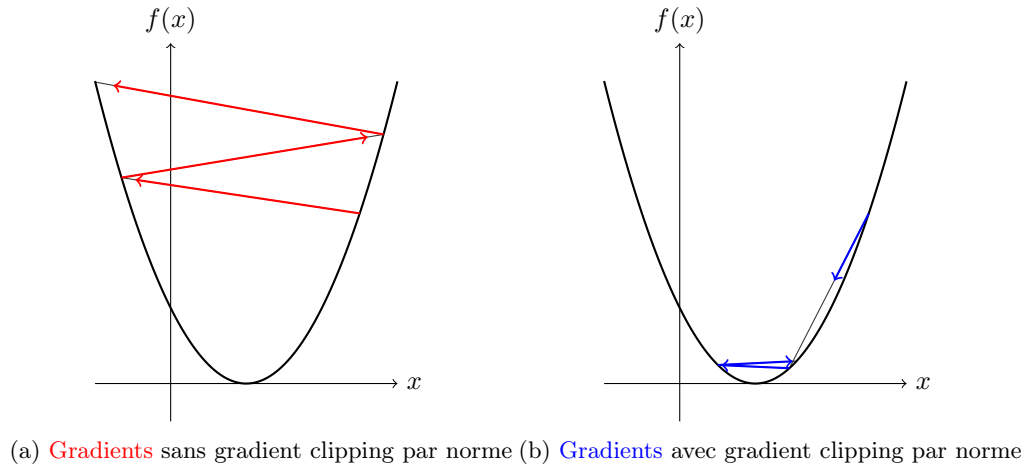


FIGURE 3.7 – Descente de gradient pour  $f(x) = (x - 1)^2$  et un learning rate de 1.05

Si l'on prend un learning rate supérieur à 1, nous allons avoir des gradients de plus en plus grand parce que l'on diverge. C'est ce qu'on observe dans la figure (3.7a). En ajoutant la technique du gradient clipping, on observe dans la figure (3.7b) que nous n'avons plus d'explosion. Mais nous n'avons pas non plus convergence! Le gradient clipping est un outil pour limiter l'explosion des gradients, mais ne garanti pas une convergence.

Le phénomène d'explosion des gradients est connu et très fréquent dans les réseaux de neurones récurrent. C'est ce qui explique son omni présence dans les descriptions techniques des Large Language Model comme les modèles GPT [Brown et al., 2020] d'OpenAI, LLaMa (1 et 2) [Touvron et al., 2023b, Touvron et al., 2023a] de Meta ou encore Chinchilla [Hoffmann et al., 2022] de Deepmind.

Finalement, dans la première partie nous cherchions à faire converger le réseau vers une solution le plus rapidement possible et dans cette deuxième partie nous le contraignons à ne pas atteindre son optimal au profit d'une version qui aura plus tendance à généraliser. Nous l'avons même contraint dans sa vitesse en modifiant la valeur des gradients. Ces deux efforts sont souvent couplé à un troisième : la normalisation d'un réseau de neurones.

### 3.3 Comment normaliser un réseau de neurones ?

En 1998, Yann Le Cun, Léon Bottou, Genevieve Orr et Klaus-Robert Müller publie l'article *Efficient BackProp* [LeCun et al., 1998b] où est montré l'importance de normaliser les input d'un réseau de neurone. Pour s'en convaincre suivons la même réflexion que dans l'article.

On considère le cas extrême où toutes les valeurs de l'input sont positive. Avec la sous-section (1.1.1) où l'on décrit l'algorithme de backpropagation, on comprend que l'ensemble des gradients d'update seront du même signe : donc un vecteur de poids ne pourra changer de direction qu'en zigzaguant. Cela ralentit fortement la convergence du réseau de neurone.

Toujours dans le chapitre 1 à la sous-section (1.1.2) nous avons montré qu'une manière d'éviter l'explosion et la disparition des gradients avec des initialisations bien définies : nous avons supposées que les inputs étaient effectivement normalisé. Nous savons donc garantir un début d'entraînement sain. Mais est-on certain qu'il va le rester ?

#### 3.3.1 Batch Normalization

C'est en partant de cette question que Sergey Ioffe et Christian Szegedy publie en 2015 l'article *Batch normalization : Accelerating deep network training by reducing internal covariate shift* [Ioffe and Szegedy, 2015] où l'on peut lire :

*Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities*

— Sergey Ioffe, Christian Szegedy (2015)

Ce phénomène est appelé l'*internal covariate shift* et fait partie de la famille des *shifts* qui existent en Machine Learning<sup>6</sup>. Pour essayer de contrer ses effets est proposé une nouvelle couche dans un réseau de neurones : Batch Normalization.

L'idée est de pousser l'idée de normaliser les inputs à l'ensemble des couches, pas seulement la première. Si nous le faisons systématiquement avec l'ensemble du dataset, alors nous devons calculer les nouvelles valeurs pour normaliser pour chaque couche à chaque époques. Donc calculer des matrices de variance-covariance ainsi que leurs racine carré inverse : c'est très coûteux.

Pour tout de même approcher l'idée de normaliser chaque couche, l'article propose une première simplification. La première est que chaque dimensions du vecteur d'input sera normalisée indépendamment :

$$\hat{x}^{(k)} = \frac{x^{(k)} - \overline{x^{(k)}}}{\sigma_{x^{(k)}}}$$

---

6. Il s'agit de l'ensemble des changements qui peuvent se produire pendant l'entraînement ou l'utilisation d'un algorithme. Par exemple le changement de distribution entre le dataset de train et le dataset de test.

Cela permet d'accélérer la convergence, même si les features sont encore corrélées. Ce genre de transformation, appliqué à chaque couche, peut changer ce que chaque couche *représente*. Normaliser les inputs d'une fonction d'activation sigmoïd les contraints à être proche de 0 donc du régime presque linéaire de sigmoïd. Nous devons nous assurer que la transformation que l'on introduit correspond à une transformation identité : on ne doit pas avoir de changement en terme de *shift*.

Pour ne pas le perdre, on introduit deux paramètres pour chaque couche  $l \leq L$  que le réseau va apprendre pour permettre de conserver le pouvoir de représentation du réseau. Si l'on note  $z$  l'output de la couche Batch-Normalization, on a :

$$z^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Puisque les paires  $(\gamma^{(k)}, \beta^{(k)})$  sont apprises avec le modèle, si l'on utilise une descente de gradient stochastique alors on ne peut plus utiliser l'ensemble du dataset pour normaliser. C'est ici qu'intervient la seconde simplification : chaque mini-batch  $\mathcal{B}$  produit une estimation de la moyenne et de la variance. La normalisation se calcule donc comme :

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}} \quad (3.2)$$

$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{j=1}^m x_j$   
 $\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{j=1}^m (x_j - \mu_{\mathcal{B}})^2$   
 $\varepsilon > 0$  pour prévenir les instabilités numériques

Et le résultat de la couche de batch-normalization comme :

$$z_i = \gamma \hat{x}_i + \beta$$

L'introduction de cette nouvelle brique dans l'architecture d'un réseau de neurones permet d'accélérer sa convergence, de manière saine, si l'on modifie également le reste des paramètres. Dans l'article, et dans la pratique, il est conseillé quand on utilise Batch Normalization d'augmenter le learning rate et de supprimer ou réduire le dropout par exemple.

Si l'article est clair sur la position de cette couche par rapport à l'activation (fonction d'activation puis Batch Normalization), dans la pratique ce n'est pas ce qui semble fonctionner le mieux. En témoigne le commentaire de François Chollet<sup>7</sup> suite à une question sur GitHub :

*I haven't gone back to check what they are suggesting in their original paper, but I can guarantee that recent code written by Christian [Szegedy] applies relu before BN. It is still occasionally a topic of debate, though.*

— François Chollet (2016)

**Ecart-type de  $x^{(k)}$  calculé sur le training set**

7. François Chollet est un chercheur chez Google qui a, entre autre, écrit Keras et un des livres références (si ce n'est le) sur le Deep Learning.

Par expérience, nous conseillons de suivre cette recommandation. Il faut cependant garder en tête que nous travaillons à présent en estimant la moyenne et la variance par batch : ainsi une petite taille de batch peut donner lieu à des estimations erronées. De plus, il n'est pas évident comment nous pouvons appliquer cette technique à des réseaux de neurones récurrents.

### 3.3.2 Layer Normalization et RMSNorm

Partant de ce constat est proposé la couche *Layer Normalization* un an plus tard par Jimmy Lei Ba, Jamie Ryan Kiros et Geoffrey Hinton dans l'article éponyme [Ba et al., 2016]. Pour supprimer la dépendance de la taille du batch, au lieu de normaliser chaque features indépendamment, normalisons chaque observations du batch indépendamment ! Nous faisons *l'inverse*.

Formellement, si l'on considère un vecteur avec  $d$  features, Layer Normalization normalise comme suit :

$$\hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2 + \varepsilon}} \quad (3.3)$$

$\mu_l = \frac{1}{d} \sum_{j=1}^d x_j$   
 $\sigma_l^2 = \frac{1}{d} \sum_{j=1}^d (x_j - \mu_l)^2$   
 $\varepsilon > 0$  pour prévenir les instabilités numériques

Puis, de manière identique à Batch-Normalization pour conserver la puissance de représentation du réseau, l'output de la couche est définie comme :

$$z_i = \gamma \hat{x}_i + \beta$$

Avec  $\gamma$  et  $\beta$  des paramètres appris en même temps que les autres paramètres du modèle. Notons que l'équation (3.2) de normalisation pour Batch-Normalization et (3.3) de normalisation pour Layer-Normalization sont très similaire. On peut visualiser ces différences avec la figure (3.8).

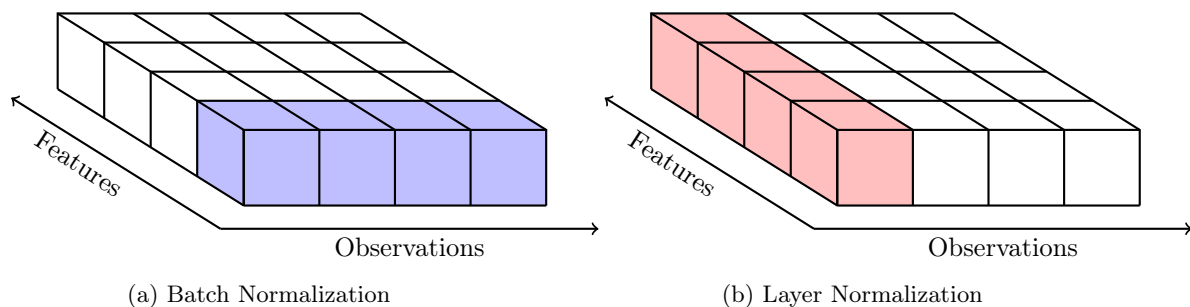


FIGURE 3.8 – Pour un dataset donné, comparaison de la normalisation entre les deux méthodes décrites

Notons que puisque Layer Normalization normalise selon les features, nous pouvons avoir des batch de taille 1 comme c'est le cas dans l'online learning par exemple.

Résolvant l'un des problèmes de Batch Normalization pour les réseaux de neurones récurrents et garantissant un entraînement plus sain, Layer Normalization fait partie intégrante de l'architecture Transformers des modèles GPT d'Open AI [Brown et al., 2020]. Google l'a également adopté en 2022 pour son LLM LaMDa [Thoppilan et al., 2022].

Cependant, Layer Normalization induit un surplus de calcul qui peut devenir coûteux quand on entraîne des LLM. Biao Zhang et Rico Sennrich propose une alternative à Layer Normalization baptisé RMSNorm dans l'article *Root mean square layer normalization* [Zhang and Sennrich, 2019].

L'hypothèse de départ de l'article est que la stabilisation offerte par les techniques de normalisation ne vient pas du retrait de la moyenne, mais de la mise à l'échelle. Ainsi, il propose de simplifier LayerNorm en supprimant la dépendance en la moyenne et en modifiant l'input comme :

$$\hat{x}_i = \frac{x_i}{\sqrt{\frac{1}{d} \sum_{j=1}^d x_j^2}} \quad (3.4)$$

En comparant aux équations (??) et (??) on observe la simplification croissante de la normalisation. Cependant, les performances sont comparables ! Ainsi, les modèles LLaMa de Meta [Touvron et al., 2023a, Touvron et al., 2023b, Dubey et al., 2024], les familles de modèles PaLM et Gemma de Google [Chowdhery et al., 2022, Anil et al., 2023, Team et al., 2024] et les familles de modèles Qwen [Yang et al., 2024, Team, 2024] utilise RMSNorm.

La discussion que nous avons concernant le placement de la couche de BatchNormalization se pose également pour la couche LayerNorm et RMSNorm. C'est l'objet de l'article *On layer normalization in the transformer architecture* [Xiong et al., 2020] publié en 2020. Il est préconisé de faire précéder directement les blocs d'attention et de Feed Forward Network par la couche de normalisation comme l'illustre la figure 3.9.

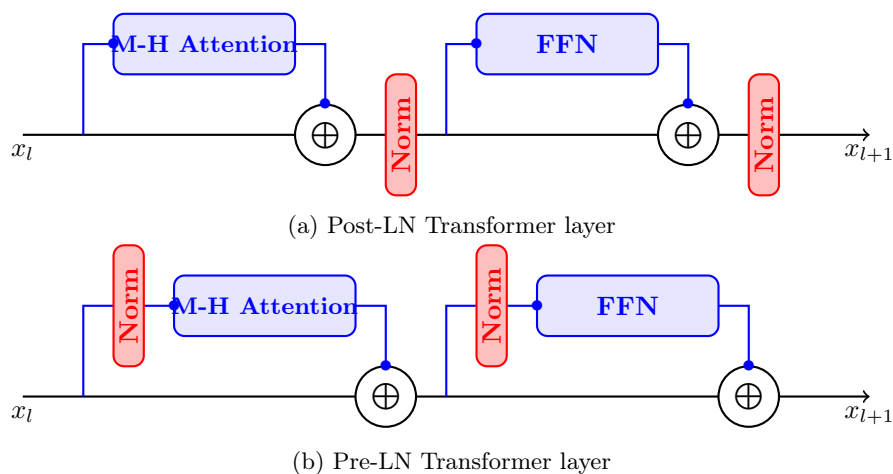


FIGURE 3.9 – Placement de la couche de normalisation dans l'architecture Transformers

Ce nouveau placement a été adopté par le modèle GPT-3 d’OpenAI [Brown et al., 2020] puis les modèles LLaMa de Meta [Touvron et al., 2023a, Touvron et al., 2023b, Dubey et al., 2024].

## Séance 4

# Avancées des *Large Language Models* pour les réseaux de neurones

Le début des années 2020 a récolté les fruits de la recherche théorique mondiale des trois décennies précédentes, accompagnées par les progrès techniques (induits ou non). En tête de pont se trouvent les modèles de langage. En 2017, un article révolutionne le domaine et des variantes toujours plus perfectionnées sont présentées chaque jour.

Cette effervescence du domaine de la recherche et de l'industrie sur le même sujet a amené de nombreux cerveaux à réfléchir aux mêmes problématiques et les avancées théoriques et techniques qui en découlent bénéficient à l'ensemble du deep learning et pas seulement au domaine du langage. La plupart du temps, les avancées étaient même déjà présentes mais pas encore sous le feu des projecteurs.

Nous proposons dans ce chapitre de parcourir quelques-unes de ces avancées techniques, sélectionnées par leur omniprésence ou leurs différences, simplicités ou complexités. Nous nous poserons des questions basiques et classiques en deep learning qui, à la lumière des LLM, nécessitent des réponses plus subtiles qu'il n'y paraît.

### 4.1 Comment modifier la valeur du learning rate au cours de l'entraînement ?

Le learning rate compte parmi les hyperparamètres les plus critiques du Machine Learning. Mal réglé, il peut soit faire diverger soit converger extrêmement doucement le modèle. Si la plupart des praticiens de Deep Learning travaillent avec des learning rate constants dans des schémas adaptatifs, les échéanciers se sont montrés d'une redoutable efficacité pour améliorer les performances des réseaux de neurones, en particulier ceux des LLM. Commençons par comprendre pourquoi il faut modifier cette valeur.

### 4.1.1 Un échancier est nécessaire pour la descente de gradient stochastique

Étant donnée un dataset  $\mathcal{D}$  défini comme dans (4.1), on cherche à minimiser une fonction de perte  $\mathcal{L} : \mathbb{R}^d \rightarrow \mathbb{R}$ .

$$\mathcal{D} = \left\{ (x_i, y_i) \mid \forall i \leq n, x_i \in \mathbb{R}^{d'}, y_i \in \mathcal{Y} \right\} \quad (4.1)$$

Nombre d'informations  
Nombre d'observations

Cette fonction de de perte s'écrit souvent comme la somme d'une fonction qui s'applique à chacun des observations de  $\mathcal{D}$ . Par exemple pour la MSE pour un problème de régression et pour la cross-entropie pour un problème de classification, on a :

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 & \mathcal{L}(\theta) &= -\frac{1}{n} \sum_{i=1}^n y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i) \\ &= \frac{1}{n} \sum_{i=1}^n \ell_i(\theta) & &= \frac{1}{n} \sum_{i=1}^n \ell_i(\theta) \end{aligned}$$

Ainsi, pour une unique étape de descente de gradient pour la fonction  $\mathcal{L}$ , on applique  $n$  fonctions  $\ell$ . La descente de gradient *stochastique* propose de sélectionner aléatoirement un index  $i_t$  à l'étape  $t$  et mettre à jour les poids uniquement avec cette observation. La descente de gradient devient alors :

$$\theta_{t+1} = \theta_t - \eta_t \nabla \ell_{i_t}(\theta_t) \quad (\text{Descente de gradient stochastique (SGD)})$$

Notons que par le côté aléatoire de la sélection de l'index  $i_t$ , nous n'avons pas vraiment une *descente* : on ne peut garantir qu'une descente en espérance.



**Exercice 4.1** (Descente en espérance). Soit  $\mathcal{L}$  une fonction de perte pouvant s'exprimer comme précédemment. On suppose de plus que  $\mathcal{L}$  est différentiable et  $\beta$ -smooth (voir définition A.4), qu'il existe un minimum pour cette fonction et que chaque  $\ell_i$  soit différentiable continue.

1. Montrer que :

$$\mathbb{E}[\mathcal{L}(\theta_{t+1})] \leq \mathcal{L}(\theta_t) - \eta_t \langle \nabla \mathcal{L}(\theta_t), \mathbb{E}[\nabla \ell_{i_t}(\theta_t)] \rangle + \frac{\beta \eta_t^2}{2} \mathbb{E}[\|\nabla \ell_{i_t}(\theta_t)\|^2]$$

A-t-on la garantie d'une descente ?

2. On suppose qu'un index  $i_t$  d'une mise à jour  $t$  vérifie :

- $i_t$  ne dépend pas des index précédents
- $\ell_{i_t}(\theta_t)$  est un estimateur non biaisé de  $\nabla \mathcal{L}(\theta_t)$
- $\mathbb{E}[\|\nabla \ell_{i_t}(\theta_t)\|^2] \leq \sigma^2 + \|\mathcal{L}(\theta)\|^2$

Comment obtenir  $i_t$  pour garantir les deux premiers points ?

3. Montrer qu'en supposant les trois points précédents, on a :

$$\mathbb{E}[\mathcal{L}(\theta_{t+1})] \leq \mathcal{L}(\theta_t) - \left( \eta_t - \frac{\beta \eta_t^2}{2} \right) \|\nabla \mathcal{L}(\theta_t)\|^2 + \frac{\beta \eta_t^2}{2} \sigma^2$$

A quelle condition a-t-on une descente en espérance ?

*Solution.* On reprend les notations précédentes.

1. Par définition d'une fonction  $\beta$ -smooth, on a :

$$\begin{aligned} \mathcal{L}(\theta_{t+1}) &\leq \mathcal{L}(\theta_t) + \langle \nabla \mathcal{L}(\theta_t), \theta_{t+1} - \theta_t \rangle + \frac{\beta}{2} \|\theta_{t+1} - \theta_t\|^2 \\ \mathbb{E}[\mathcal{L}(\theta_{t+1})] &\leq \mathcal{L}(\theta_t) - \eta_t \langle \nabla \mathcal{L}(\theta_t), \mathbb{E}[\nabla \ell_{i_t}(\theta_t)] \rangle + \frac{\beta \eta_t^2}{2} \mathbb{E}[\|\nabla \ell_{i_t}(\theta_t)\|^2] \end{aligned}$$

Nous n'avons pas une garantie de descente parce qu'il n'est pas certain que  $\mathbb{E}[\mathcal{L}(\theta_{t+1})]$  décroisse.

2. En tirant uniformément les  $i_t$ , on respecte les deux premières hypothèses. Pour respecter la dernière, il suffit qu'il existe  $M > 0$  tel quel pour tout itération  $t$  on ait  $\|\ell_{i_t}(\theta_t)\| \leq M$ , ce qui sera vrai si on travail dans un compact.
3. En utilisant le troisième point, on obtient l'inégalité souhaitée. Pour s'assurer une descente, il faut que pour tout  $t$  on ait  $\eta_t \leq \frac{1}{\beta}$

□

Nous travaillons avec des fonctions de perte convexe afin d'obtenir des garanties de convergence vers le minimum global. Cependant, dans un réseau de neurones la fonction de perte n'est pas convexe<sup>1</sup>.

**Théorème 4.1** (Convergence SGD avec learning rate fixe). *Soit  $\mathcal{L}$  une fonction de perte non convexe mais  $\beta$ -smooth. On note  $\theta^* = \arg \min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta)$ . On considère une descente de gradient stochastique avec  $\eta_t = \eta \in \left]0, \frac{1}{\beta}\right]$ . Alors pour tout  $T \geq 1$  :*

$$\mathbb{E} \left[ \frac{1}{T} \sum_{t=0}^{T-1} \|\nabla \mathcal{L}(\theta_t)\|^2 \right] \leq \eta \beta \sigma^2 + \frac{2(\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*))}{\eta T}$$

On obtient que  $\lim_{T \rightarrow +\infty} \mathbb{E} \left[ \min_{0 \leq t \leq T-1} \|\nabla \mathcal{L}(\theta_t)\|^2 \right] \in [0, \eta \beta \sigma^2]$  : le bruit nous empêche de converger vers un point de gradient nul. Ce n'est pas non plus une garantie de convergence vers le minimum global. Mais que se passerait-il si nous imposions d'avoir un learning rate décroissant ?

**Théorème 4.2** (Convergence SDG avec learning rate décroissant). *Soit  $\mathcal{L}$  une fonction de perte non convexe mais  $\beta$ -smooth. On note  $\theta^* = \arg \min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta)$ . On considère une descente de gradient stochastique avec  $(\eta_t)$  une suite décroissante telle que  $\eta_t \in \left]0, \frac{1}{\beta}\right]$  et :*

$$\sum_{t=0}^{+\infty} \eta_t = +\infty \quad \text{et} \quad \sum_{t=0}^{+\infty} \eta_t^2 < +\infty$$

Alors, pour tout  $T \geq 1$  :

$$\lim_{T \rightarrow +\infty} \mathbb{E} \left[ \frac{1}{\sum_{t=0}^{T-1} \eta_t} \sum_{t=0}^{T-1} \eta_t \|\nabla \mathcal{L}(\theta_t)\|^2 \right] = 0$$

Nous n'avons pas non plus une convergence vers un minimum global. Mais cette fois, au lieu d'atteindre un intervalle autour du point de gradient nul, on atteint le point de gradient nul ! On comprend donc la nécessité d'avoir un échancier adapté dans une SGD.

Observons à présent comment les échanciers des learning rate ont été défini pour les Transformers.

#### 4.1.2 Échancier des Transformers originels

De nombreux *Large Language Model* (LLM) se sont inspirés des hyperparamètres décrit dans l'article qui a lancé la nouvelle vague de traitement du langage *Attention is all you need* [Vaswani et al., 2017]. L'architecture, les fonctions d'activations, les régularisations mais aussi l'échancier pour la valeur du learning rate a été copié. Il est construit de la manière suivante :

---

1. Voir la section A.4

$$\eta_t = \frac{1}{\sqrt{d}} \min \left\{ \frac{1}{\sqrt{t}}, t \tau^{-1.5} \right\}$$

Époque Époques de warmup = 4000  
Dimension du modèle = 512

On peut le visualiser avec la figure (4.1) :

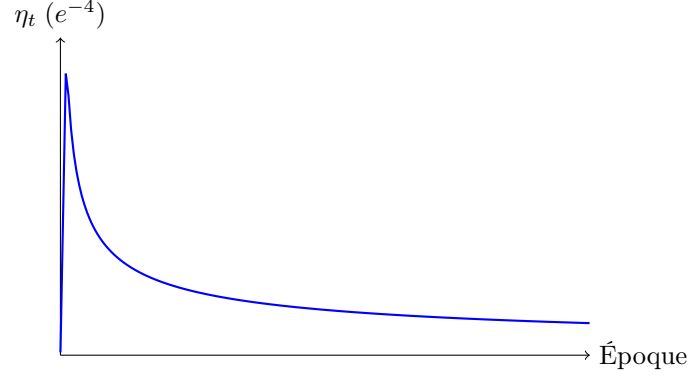


FIGURE 4.1 – Valeur du learning rate en fonction de l'époque

Pour mieux le comprendre, entrons un peu plus dans le détail de l'équation, soit  $t \in \mathbb{N}^*$ . On a :

$$\frac{1}{\sqrt{t}} \leq t \tau^{-1.5} \iff 1 \leq \sqrt{\left(\frac{t}{\tau}\right)^3} \iff 1 \leq \frac{t}{\tau} \quad \text{car } x \mapsto \sqrt{x^3} \text{ est croissante sur } \mathbb{R}_+$$

Ainsi, pendant la période de chauffe on augmente linéairement la valeur du learning rate puis on décroît selon la fonction  $x \mapsto \frac{1}{\sqrt{x}}$ . Cette décroissance est plus douce qu'une fonction inverse classique ce qui permet de conserver un learning rate relativement grand pendant plusieurs époques.

Ce genre d'échéancier est assez classique, mais nous n'avions pas avant 2017 de pratique unanime dans le domaine du deep learning. Dans le domaine des LLM en revanche, un nouveau type d'échéancier va faire l'unanimité : les échéanciers cycliques.

### 4.1.3 Échéanciers cycliques

[Smith, 2017] propose une nouvelle manière de définir un échéancier : il sera cyclique. A noter que cette proposition est antérieure à *Attention is all you need*, elle s'applique donc à l'ensemble des réseaux par construction.

*The essence of this learning rate policy comes from the observation that increasing the learning rate might have a short term negative effect and yet achieve a longer term beneficial effect*

— Leslie Smith (2015)

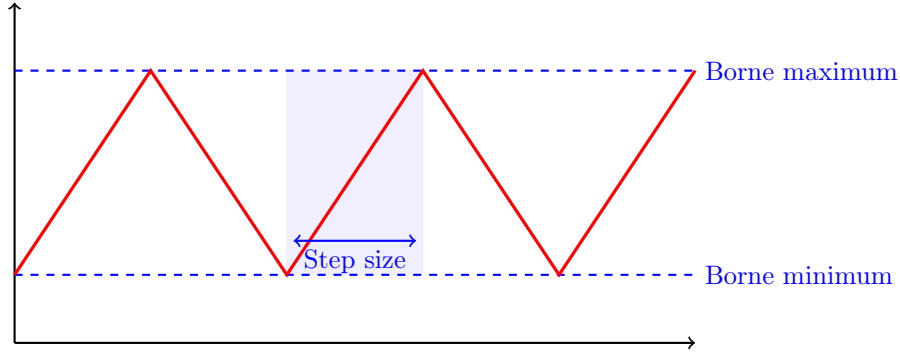


FIGURE 4.2 – Échéancier triangulaire pour le learning rate

Partant de cette remarque, il est proposé de faire varier le learning rate entre des bornes *raisonnables*. La première manière de faire varier la valeur est triangulaire :

Nous avons 3 hyperparamètres à régler : les deux bornes et la longueur d'un cycle. L'article propose de fixer la longueur d'un cycle comme 2 à 10 fois le nombre d'itérations qui seront réalisées à chaque époque. Concernant les deux bornes, c'est plus expérimental. En testant plusieurs valeurs sur plusieurs époques, on peut identifier les bornes comme les valeurs qui exhibent un début d'apprentissage et un début de stagnation/régression de la performance du réseau de neurones. Il est cependant rappelé de comparer les performances du réseau entraîné avec cet échancier cyclique et les performances d'un réseau de neurones entraîné avec un learning rate fixe.

Nous pouvons imaginer beaucoup de manières de réaliser ces cycles et on peut ajouter des bornes qui décroissent dans le temps également. Une manière est devenue standard dans tous les modèles fondamentaux : l'échéancier cosinus [Loshchilov and Hutter, 2016]. L'échéancier se définit par plusieurs paramètres :

$$\eta_t = \eta_{\min}^i + \frac{1}{2} (\eta_{\max}^i - \eta_{\min}^i) \left( 1 + \cos \left( \frac{T_{\text{cur}}}{T_i} \pi \right) \right)$$

Nombre d'itérations réalisées dans le cycle  $i$  Nombre d'itérations à réaliser dans le cycle  $i$

On peut le visualiser avec la figure (4.3) où l'on a décidé de prendre  $T_i$ ,  $\eta_{\min}^i$  et  $\eta_{\max}^i$  fixe. On peut tout à fait décider de valeurs différentes pour chaque cycle.

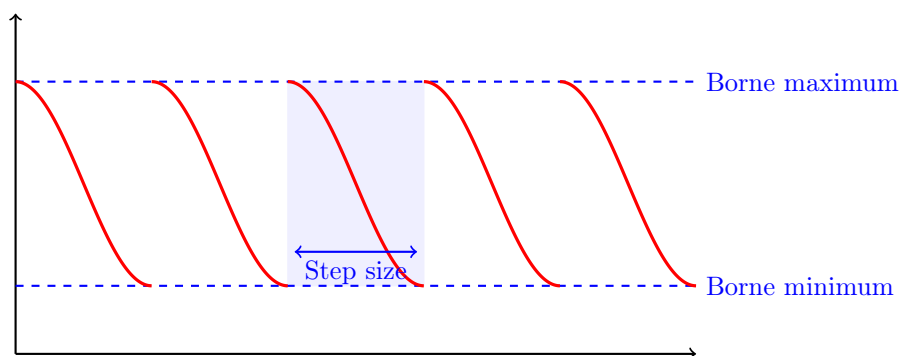


FIGURE 4.3 – Échéancier cosinus pour le learning rate

Il est montré que les *performances* des réseaux de neurones sur de multiples tâches, pour de multiples architectures, sont améliorées. On entend par *performances* une meilleure convergence plus rapide, à condition que l'on attende la fin d'un cycle (comme précédemment).

En pratique, les Large Language Models fondamentaux s'entraînent sur une seule époque. Autrement dit, la totalité du dataset n'est *vue* qu'une seule fois. Il paraît donc difficile de réaliser des *restarts* et donc d'avoir plusieurs cycles dans une même époque.

Il reste à décider de la valeur la plus haute et la plus basse du learning rate ainsi que la longueur du cycle. Ces questions d'architectures optimales ont été explorées par Deepmind en 2022 avec le modèle Chinchilla [Hoffmann et al., 2022]. L'objectif de Deepmind était de mieux comprendre comment mettre à l'échelle les LLM pour un budget alloué de calcul. Pour en faire la démonstration, ils ont créé un nouveau modèle (Chinchilla) qui répond aux lois d'échelles que ce papier et les précédents édictent. La conclusion est que les modèles étaient largement sur-paramétrés par rapport à la quantité de texte auquel ils avaient accès. Partant de ce constat, Deepmind est alors passé de Gopher [Rae et al., 2021] avec 280 milliards de paramètres à Chinchilla avec 70 milliards de paramètres avec de meilleures performances sur un grand ensemble de tâches.

Les questions du nombre de paramètres et du nombre de *training steps* pour un LLM sont spécifiquement étudiées dans ce papier. Plusieurs lois sont issues des précédents travaux, notamment *Scaling laws for neural language models* [Kaplan et al., 2020] publié en 2020. Les conclusions sont relativement similaires, mais diffèrent sur plusieurs points dont la taille optimale de modèle et, pour ce qui nous intéresse, l'utilisation de l'échéancier.

L'optimal (selon Deepmind) semble être d'adopter une période de chauffe du modèle en passant d'un learning rate très faible (de l'ordre de  $10^{-7}$ ) au plus haut learning rate que l'on souhaite, puis de le réduire de 10% avec un cosinus. La taille d'un cycle étant celle de l'entraînement, voire légèrement plus. Pour l'article de 2020, une décroissance jusqu'à zéro n'est pas vraiment un problème et ne semble pas changer énormément l'apprentissage tant que l'évolution du learning rate n'est pas trop brusque et que les valeurs ne sont pas trop petites.

Il est montré que réduire la valeur du learning rate jusqu'à 10% semble optimal car aller au-delà amène un gain faible. Cependant, réduire en dessous de 10% n'est pas une bonne pratique. Ainsi, la majorité des LLM adopte aujourd'hui l'échéancier décrit précédemment et visualisé avec

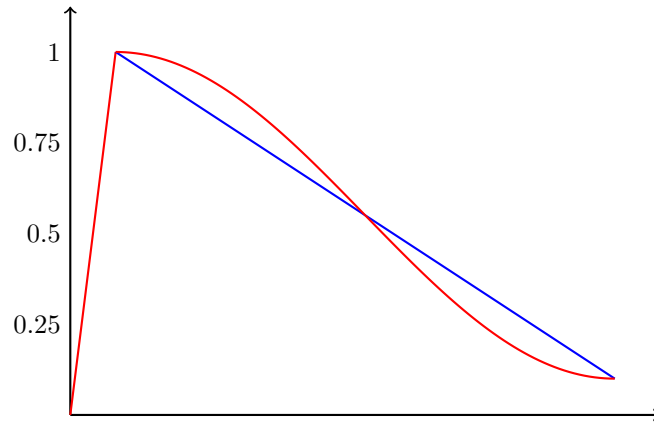


FIGURE 4.4 – Rapport maximum LR / LR pour l'échéancier **décroissement linéaire** et **cosinus** avec période de chauffe

la figure (4.4).

Il est cependant noté dans [Popel and Bojar, 2018] que la période de warmup est critique. À la fois la longueur et le maximum atteint ont une grande importance pour la suite de l'entraînement. Ces hyperparamètres sont donc sensibles mais très coûteux à tester. Ainsi, des articles comme [Xiong et al., 2020] en 2020 proposent de supprimer cette partie à condition de revoir le placement de la couche de normalisation. À ce jour, nous n'avons pas connaissance de LLM majeur suivant cette recommandation. En revanche, le placement lui l'est, comme évoqué dans la section (3.3).

## 4.2 Comment amener une bonne non-linéarité dans un réseau de neurones ?

Les fonctions d'activation sont le coeur d'un réseau de neurones : ce sont elles qui amènent la non-linéarité. Le choix est central et conditionne à plusieurs niveaux le réseau de neurones. Nous avons vu dans le chapitre 1 qu'en fonction de la fonction d'activation, il faut changer la manière dont on initialise les poids d'un réseau de neurones. De même, l'utilisation de la dérivée de cette fonction d'activation apparaît systématiquement dans la mise à jour des poids dans les équations de la back-propagation.

Ces fonctions sont amenées à être composées largement et certaines se comportent bien avec cette opération, d'autres moins.

**Exercice 4.2** (Fonctions d'activation et composition). *On dit qu'on compose deux fonctions quand on applique une fonction à l'image d'une autre fonction. On dit qu'on compose  $n$  fois une fonction avec elle-même quand on répète  $n$  fois le procédé précédent avec une même fonction. On notera le résultat :  $f^{(n)}(x)$ .*

1. Comment se comporte la fonction  $\text{ReLU}(x) = \max\{0, x\}$  avec la composition ?
2. Montrer que pour la fonction  $\text{softplus}(x) = \ln(e^x + 1)$ , on a :

$$\forall n \in \mathbb{N}^*, \forall x \in \mathbb{R}, \text{softplus}^{(n)}(x) = \ln(e^x + n)$$

3. Montrer que la fonction  $x \mapsto \sqrt{x^2 + b}$  se comporte également bien pour la composition.
4. En quoi est-ce une propriété importante ?

*Solution.* On reprend les notations définies précédemment.

1. La fonction ReLU est identique pour n'importe quel nombre de composition.
2. Par récurrence triviale.
3. Par récurrence à nouveau, en notant  $f(x, b) = \sqrt{x^2 + b}$ , on a :

$$\forall n \in \mathbb{N}^*, \forall x \in \mathbb{R}, f^{(n)}(x, b) = \sqrt{x^2 + nb}$$

4. Cela permet d'avoir une certaine stabilité dans la transmission d'informations et des garanties simples sur les différents gradients. Le cas de ReLU peut expliquer, entre autre, son succès dans le domaine.

□

Depuis son introduction, la fonction ReLU est devenue le standard des réseaux de neurones profonds. Malgré des atouts théoriques des fonctions d'activation comme Leaky ReLU par rapport à ReLU, aucune preuve empirique majeure n'a justifié un changement dans la pratique des réseaux de neurones. Pour plus de détails, on peut se référer à l'annexe ?? qui traite des avancées récentes pour corriger les problèmes de ReLU.

Cependant, dans les développements récents des modèles fondamentaux LLM, la fonction ReLU qui était utilisée dans l'architecture Transformer a été remplacée par plusieurs nouvelles fonctions.

#### 4.2.1 GELU et SiLU

[Hendrycks and Gimpel, 2016] introduit une nouvelle fonction d'activation : GELU. La volonté de combiner le comportement de ReLU (multiplier par 0 ou 1 l'input) et le comportement du Dropout (multiplier par 0 aléatoirement) est à l'origine de ce papier. Les deux idées sont fusionnées en multipliant l'input d'un neurone par  $m \sim \text{Bernoulli}(\phi(x))$  avec  $\phi(x) = \mathbb{P}(X \leq x)$  et  $X \sim \mathcal{N}(0, 1)$ . Le choix de la distribution normale est justifié par la tendance qu'ont les inputs d'un neurone à suivre cette distribution<sup>2</sup>.

2. Nous avons vu cela dans les chapitres 1 et 3.

Cependant, ce comportement n'est applicable que pendant la phase d'apprentissage : nous souhaitons un comportement déterministe dans la phase d'exploitation du modèle. Ainsi, pour approcher au mieux le comportement de cette régularisation, on définit la fonction d'activation GELU comme :

$$\text{GELU}(x) = x\mathbb{P}(X \leq x) = x\phi(x)$$

Etudions cette fonction plus en détail.

**Exercice 4.3** (Fonction GELU). *On reprend les notations précédentes, et on définit la fonction d'erreur comme :*

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

1. Montrer que :

$$\text{GELU}(x) = \frac{1}{2}x \left[ 1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right) \right]$$

2. Calculer sa dérivée. Quelle amélioration identifiez-vous par rapport à ReLU ?

*Solution.* En conservant les notations précédentes, nous avons que :

$$\begin{aligned} \frac{1}{2} \left( 1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right) \right) &= \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-t^2} dt \\ &= \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{t^2}{2}} dt \quad \text{par changement de variable} \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^0 e^{-\frac{t^2}{2}} dt + \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{t^2}{2}} dt \\ &= \phi(x) \end{aligned}$$

D'où on peut déduire le lien entre GELU et la fonction d'erreur, via le lien entre la fonction de répartition de la loi normale et la fonction d'erreur.

La dérivée se calcule directement comme :

$$\text{GELU}(x) = \phi(x) + \frac{x}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

Elle est continue, donc on a que GELU est  $C^1$  sur  $\mathbb{R}$  contrairement à ReLU. En particulier, elle est  $C^1$  dans le voisinage de zéro donc elle permet de transmettre correctement les informations quand on travaille avec des valeurs proches de zéro lorsque ReLU aura un comportement moins lisse.  $\square$

Cette fonction d'activation améliore quasiment systématiquement les performances des réseaux de neurones testées dans le papier de recherche pour différentes tâches (vision et NLP). Une autre fonction est également proposée, mais non testée, dans le papier : SiLU. Au lieu de choisir la distribution normale, nous aurions pu choisir la distribution logistique et donc définir la fonction :

$$\text{SiLU}(x) = x\sigma(x) \quad \text{avec} \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$



Cette fonction a été découverte indépendamment par une équipe de chercheurs de Google Brain l’année suivante dans [Ramachandran et al., 2017]. L’objectif de cet article était d’explorer de manière systématique un ensemble de formes de fonctions d’activation pour trouver<sup>3</sup> de nouvelles fonctions d’activation plus performantes que ReLU. GeLU n’a pas été trouvée dans cette recherche, mais la fonction SiLU oui. Elle a été d’abord nommée Swish par les auteurs, mais à la suite de la remarque des auteurs d’origine de SiLU, les chercheurs de Google Brain ont conservé le nom Swish et le justifient par la présence d’un paramètre  $\beta$  qui n’est pas introduit<sup>4</sup> par les auteurs originels.

Les fonctions GeLU et SiLU sont assez proches visuellement de la fonction ReLU :

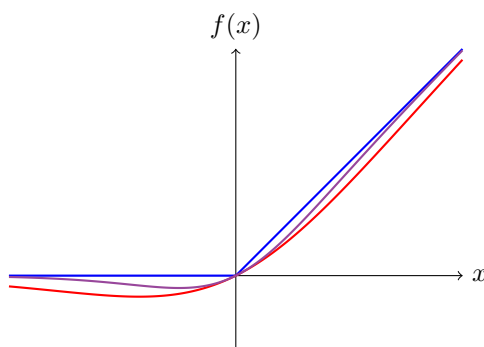


FIGURE 4.5 – Fonctions ReLU, SiLU et GeLU

Le comportement asymptotique des trois fonctions est similaire, mais ce qui change est la différentiabilité partout des deux nouvelles fonctions. De plus, elles ne sont pas monotones et n’auront pas le problème de mort de neurone que peut avoir ReLU.

La fonction GeLU est l’un des changements majeurs apportés par la famille des modèles GPT d’OpenAI au transformer original. À la suite de cette utilisation, de nombreux LLM se sont basés sur cette fonction d’activation. En pratique, à partir de la publication de ce résultat, seul le modèle T5+ utilise encore la fonction ReLU comme fonction d’activation. Plus récemment, notons que Mistral 7B [Jiang et al., 2023] exploite la fonction d’activation SiLU, moins répandue que GeLU. Les modèles PaLM [Chowdhery et al., 2022, Anil et al., 2023] de Google et LLaMa [Touvron et al., 2023b, Touvron et al., 2023a] de Meta exploitent une variation de la fonction SiLU : c’est l’objet de la section suivante.

#### 4.2.2 Gated Linear Unit

Parmi les grandes architectures des réseaux de neurones, on pense à la cellule LSTM qui contient trois *gate* (Input, Forget et Output) que l’on peut visualiser dans la figure (4.6).

L’une des idées de cette architecture est de contrôler à plusieurs niveaux la quantité d’informations que l’on transmet : cette idée est reprise dans [Dauphin et al., 2016]. Remarquons déjà que par rapport aux approches classiques pour traiter un problème de langage (RNN, LSTM, GRU), les réseaux convolutionnels sont cette fois utilisés. Un des gains importants de ce choix

3. L’approche proposée rappelle celle décrite dans la section 3.1 pour l’optimiseur Lion.

4. En pratique, le paramètre  $\beta$  est valorisé à 1 par les implémentations, et le nom choisi est SiLU.

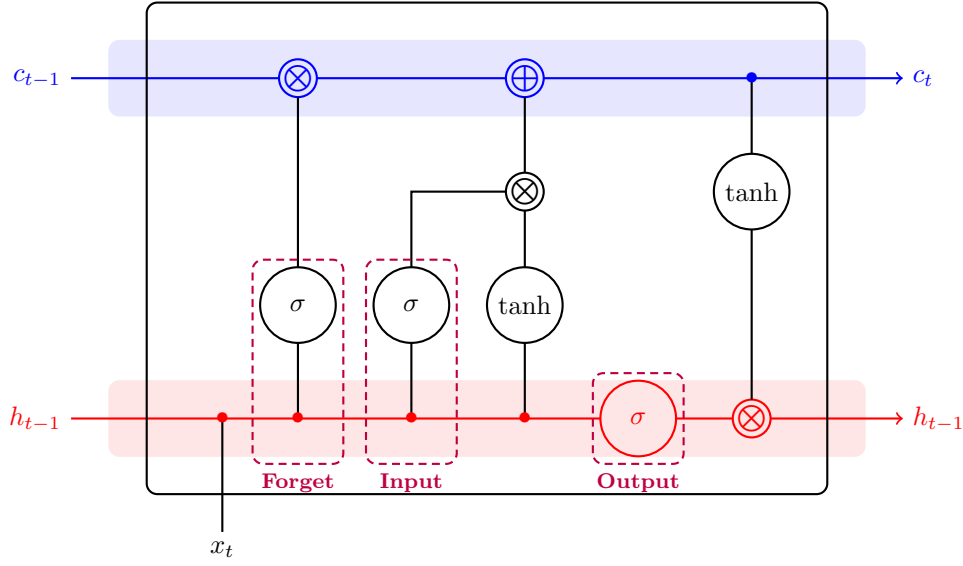


FIGURE 4.6 – Schéma d'un LSTM avec état de cellule  $(c_t)_{t \in \mathbb{N}}$  et état caché  $(h_t)_{t \in \mathbb{N}}$

est que l'on peut mieux paralléliser l'apprentissage, là où les réseaux récurrents doivent attendre l'état précédent, rendant impossible la parallélisation.

Dans ce travail, la nouveauté vient de la manière dont sont calculées les couches cachées  $h_l$  avec  $l \leq L$  la  $l$ -ième couche cachée du réseau de neurones à  $L$  couches :

**Matrice input**

$$\forall l \leq L, \quad h_l(X) = (XW + b) \otimes \sigma(XV + c)$$

**Poids couche 1**
**Poids couche 2**

On a noté  $\otimes$  le produit terme à terme et  $\sigma$  la fonction sigmoid. Le terme *gated* du titre de l'article prend alors son sens : pour chaque observation, les poids  $(V, c)$  ont pour objectif, une fois passés par la fonction sigmoid, de mesurer l'importance de cette observation. On peut se convaincre de cela avec une rapide étude des dimensions dans la formule.

Noam Shazeer pousse la réflexion plus loin en proposant en 2020 dans [Shazeer, 2020] de remplacer la fonction  $\sigma$  par d'autres fonctions d'activation classiques. Naît ainsi toute une nouvelle famille de *fonctions d'activation*. En pratique, ce ne sont pas vraiment des fonctions d'activation mais plutôt de nouveaux blocs d'architecture. Parmi celles-ci, notons :

$$\begin{aligned} \text{GEGLU}(X) &= \text{GELU}(XW + b) \otimes (XV + c) \\ \text{SwiGLU}(X) &= \text{SiLU}(XW + b) \otimes (XV + c) \end{aligned}$$

Ces deux nouveaux blocs font partis des modifications récentes apportées aux modèles fondamentaux. Comme écrit précédemment, les familles de modèles PaLM [Chowdhery et al., 2022, Anil et al., 2023], LLaMa [Touvron et al., 2023b, Touvron et al., 2023a] et Qwen [Yang et al., 2024,

Team, 2024] utilisent la fonction SwiGLU tandis que le modèle GATO [Reed et al., 2022] de Deepmind, LaMDa [Thoppilan et al., 2022] et la famille de modèle Gemma [Team et al., 2024] de Google utilisent GEGLU. On peut résumer le fonctionnement avec la figure (4.7).

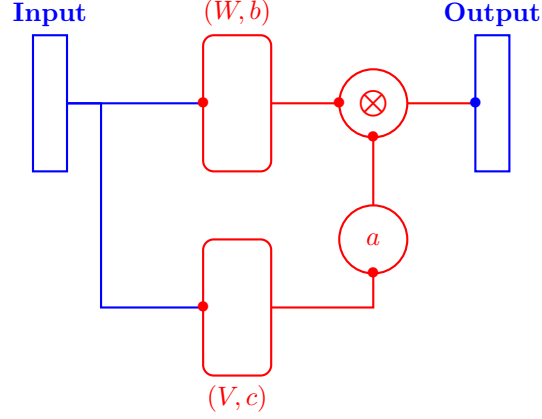


FIGURE 4.7 – Architecture GLU avec une fonction d’activation  $a$

Plus précisément, dans l’architecture Transformers les blocs *position-wise feed-forward networks* (FFN) sont les blocs impactés par cette modification. Initialement avec la fonction ReLU, maintenant avec SiLU ou GELU. A noter que de plus en plus de LLM suppriment les vecteurs de biais, pour atteindre les blocs FFN suivants :

$$\begin{aligned}\text{FFN}_{\text{GEGLU}}(X, W_1, V, W_2) &= (\text{GELU}(XW_1) \otimes (XV))W_2 \\ \text{FFN}_{\text{SwiGLU}}(X, W_1, V, W_2) &= (\text{SiLU}(XW_1) \otimes (XV))W_2\end{aligned}$$

Ainsi, un bloc FFN (figure 4.8) a maintenant trois matrices de poids à apprendre au lieu de deux :  $W_1$  et  $W_2$  pour chaque couche et  $V$  pour l’unité de *gate*. Donc pour conserver un ordre de grandeur similaire en termes de paramètres, il faut réduire le nombre de neurones que l’on place sur chaque couche.

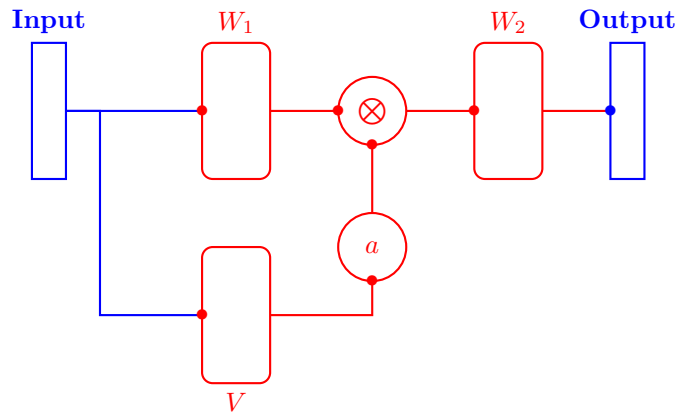


FIGURE 4.8 – Architecture FFN avec une fonction d’activation GLU  $a$

L'article de Noam Shazeer montre que ces variantes sont utiles à nombre de paramètres équivalents. Cependant, bien que cette architecture soit utilisée dans les modèles à l'état de l'art et que plusieurs articles convergent sur l'utilité, la seule explication pour éclaircir le gain de performance est donnée par l'article originel :

*We offer no explanation as to why these architectures seem to work; we attribute their success, as all else, to divine benevolence.*

— Noam Shazeer (2020)

## 4.3 Application à la vision

Nous avons jusqu'ici traité de techniques ou briques techniques popularisées par les LLM. On s'intéresse à présent au domaine qui a popularisé les réseaux de neurones : la vision. Depuis son introduction, l'utilisation des réseaux convolutionnels [LeCun et al., 1998a] fait consensus dans tous les domaines traitant avec des images. La structuration autour des datasets communs MNIST et FashionMNIST [LeCun et al., 2010, Xiao et al., 2017] ou la compétition ImageNet qui a été source d'architecture et briques élémentaires des réseaux de neurones, on peut citer :

- **AlexNet** [Krizhevsky et al., 2012] : remporte très largement la compétition de 2012 en s'appuyant sur un réseau d'environ 60 millions de paramètres. Pour le faire, c'est la première fois que l'on exploite les possibilités des GPU et qu'on utilise la technique du Dropout pour régulariser le réseau.
- **Inception v1** [Szegedy et al., 2015] : remporte la compétition de 2014 en utilisant presque 9 fois moins de paramètres qu'AlexNet mais avec 22 couches alors qu'il y en avait 8 pour AlexNet. Le module Inception est introduit : il consiste à travailler à plusieurs échelles en parallèle pour avoir une vision locale et plus globale en même temps.
- **VGG** [Simonyan and Zisserman, 2014] : talonne Inception v1 en 2014 avec une philosophie différente : plutôt que de faire varier la taille des convolutions, il suffit d'en placer plus de petite taille pour réduire la taille du modèle sans changer la capacité de représentation.
- **ResNet** [He et al., 2016] : la compétition est remportée en 2015 par l'introduction d'un réseau de neurones de 50 couches composé de ResBlock<sup>5</sup>.

Suite à la proposition du mécanisme d'attention, des travaux avec les réseaux convolutionnels ont été menés. On peut citer :

- [Bello et al., 2019] adresse le problème de localité des réseaux convolutionnels en augmentant les features avec le mécanisme d'attention. De cette manière, le modèle peut traiter d'informations distantes dans l'image.
- [Wu et al., 2020] propose le *Visual Transformer* (VT). Une image est d'abord traitée par des couches de convolutions, et ce résultat est placé dans 16 catégories sémantiques, traitées par la suite par un transformer. Le résultat peut être utilisé pour classifier par exemple.

### 4.3.1 Vision Transformers

[Dosovitskiy et al., 2020] propose en octobre 2020 les Vision Transformers (ViT) qui s'appuie uniquement sur l'architecture transformers, sans couches de convolution. Cependant, une image

---

5. Voir figure (??) page ??

n'est pas une séquence de vecteur. Ainsi, l'article propose de brutalement de transformer une image en patch qui ensemble formeront une séquence, donc une entrée pour un transformer.

**Exercice 4.4.** On considère une image de taille  $(H, W)$  pixels. On souhaite obtenir des patches de taille  $(P, P)$ .

1. On note  $N$  le nombre de patch. Que vaut  $N$  dans notre cas ?
2. Chaque patch est aplati (flattened). Quelle est la taille de la séquence total pour l'image de départ ?
3. Quelle est la taille de la séquence totale pour une image de taille  $(H, W, C)$  où  $C$  représente le nombre de canal ?
4. Faire l'application numérique pour une image de taille  $(6, 8, 3)$  avec  $P = 2$

- Solution.*
1. On doit découper une image d'aire  $H \times W$  par des patches d'aire  $P^2$  donc  $N = \frac{HW}{P^2}$
  2. Une image étant transformé en  $N$  patch de taille  $P^2$  qui sont tous aplati, on a une séquence de longueur  $N \times P^2$
  3. Si on ajoute le nombre de canal  $C$ , cela revient à multiplier par  $C$  le nombre de patch, d'où la séquence sera de longueur  $N \times (P^2 \times C)$
  4. On obtient  $N = 12$ , d'où la séquence de longueur 144. C'est cohérent puisque  $6 \times 8 \times 3 = 144$

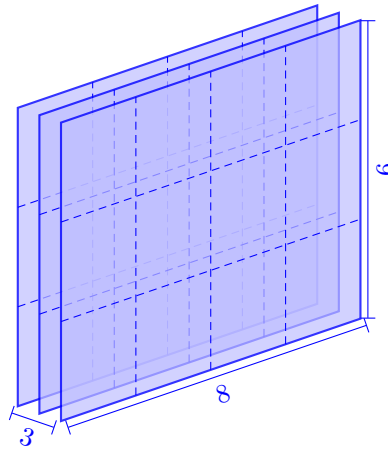


FIGURE 4.9 – Image de taille  $(6, 8, 3)$  découpée en patch de taille  $(2, 2)$

□

L'objectif d'aplatir l'image via des patches est de conserver une idée de *localité* : si l'on aplatit l'image brutalement <sup>6</sup>, on perd la structure. C'est là que les réseaux convolutionnels deviennent performants en intégrant mieux la dépendance des pixels spatialement proches. Cependant, c'est aussi un de leur défaut : un pixel en haut à gauche ne sera pas mis en regard

<sup>6</sup>. Comme dans les premières séances de Deep Learning avec MNIST par exemple

d'un pixel en bas à droite. C'est l'idée de l'attention dans un transformer qui permet de faire des liens plus *lointains* au prix d'un coût quadratique en calcul. Nous savons donc que les Vision Transformers fonctionnent pour des tâches de classifications avec des images de taille raisonnables, mais la vision ne se limite pas qu'à ça.

[Liu et al., 2021] propose une solution similaire pour traiter d'une *localité globale* en utilisant une architecture transformer (pour la globalité) et des patches (pour la localité). Sa spécificité est que les patches ne sont pas fixés mais évolutifs de tailles différentes pour pouvoir, comme dans les réseaux convolutifs avec différentes tailles de filtre, capturer des schémas plus locaux. De cette manière, on ré-intègre un mécanisme de fenêtre glissante inhérent à la vision pour être performant dans un plus grand nombre de tâches différentes.

Notons que dans les deux cas, les blocs Transformer exploitent la fonction d'activation GELU (voir section ??) et la couche de Layer Normalization (voir section 3.3.2). Il y a également plusieurs tailles de modèles qui sont disponibles pour pouvoir traiter différents usages. [Liu et al., 2021] met en particulier l'accent sur le temps d'inférence qui est crucial dans une détection d'objet pour une voiture autonome par exemple.

Il est important de préciser que pour obtenir des performances compétitives avec ces réseaux, il semble nécessaire empiriquement, d'après les deux articles, de traiter avec de très larges volumes de données tant en nombre qu'en classes à prédire.

Modèle	Année	Paramètres (M)	Accuracy (%)
ResNet-101	2015-12	45	78.2
ViT-B/16	2020-10	87	85.4
ViT-L/16	2020-10	305	86.8
Swin-B	2021-08	88	86.4
Swin-L	2021-08	197	87.3

TABLE 4.1 – Comparaison des performances d'accuracy sur ImageNet-1K, pré-entraîné sur ImageNet-22K pour des images de taille 384<sup>2</sup>

On note une performance significative des Vision Transformers par rapport à la performance historique du ResNet-101. Il faut cependant noter que les comparaisons ne sont pas tout à fait correctes : le nombre de paramètres est plus important et les ViT sont pré-entraînés sur ImageNet-22K. [Smith et al., 2023] montre que les réseaux convolutionnels s'ils sont entraînés sur autant de données que les ViT sont tout aussi performants. Autrement dit : nous comparions deux méthodes sans les mettre au même niveau. Nous voyons également que le transformer Swin est plus performant à taille comparable, et même à taille inférieure, au ViT classique. Cela démontre l'importance du biais importé de la vision version convolution.

### 4.3.2 ConvNeXt

Partant de ce constat, [Liu et al., 2022] déroule un plan de travail pour moderniser les réseaux convolutionnels en utilisant uniquement des couches de convolutions, mais en se nourrissant des avancées des Transformers. Plus précisément, l'état de l'art concernant les réseaux convolutifs était les ResNet avec le ResNet50 qui remporte la compétition ImageNet [He et al., 2016]. C'est donc ce réseau que l'on va chercher à améliorer.

Les premiers changements, avant de toucher à l'architecture, portent sur les méthodes d'entraînement avec entre autres :

- Entraînement plus long (de 90 époques à 300 époques) avec une data augmentation plus complète
- L'optimiseur AdamW (voir section 3.2.1) est exploité avec un batch size de 4096 au lieu de 256
- L'échéancier du learning rate suit un échancier cosinus (voir section 4.1.3)

On obtient un gain de performance de 2.7 points (selon l'article) en *mettant à jour* la procédure d'entraînement. Plusieurs autres changements sont présentés pour améliorer l'efficacité du modèle : nous ne les présenterons pas tous. Notons-en quelques-uns :

- **Plus grande taille de filtre** : la taille  $3 \times 3$  popularisée par VGG [Simonyan and Zisserman, 2014] est remplacée par une taille  $7 \times 7$
- **Moins de fonction d'activation** : à l'image d'un transformer, une seule fonction d'activation par bloc permet de gagner 0.7 point de performance. ReLU est également remplacé par GELU, mais ce changement ne change pas la performance.
- **Moins de couche de normalisation** : on passe de deux couches de Batch-Normalization à une seule couche de Layer-Normalization. Ces deux changements permettent de gagner chacun 0.1 point de performance.

L'ensemble de ces changements d'entraînement, macro et micro sur l'architecture ont permis d'obtenir un nouveau type de réseau nommé ConvNeXt. L'accent était mis sur le gain en performance métrique mais également en performance de vitesse de traitement d'images. Si l'on reprend la table de comparaison 4.1 :

Modèle	Année	Paramètres (M)	Accuracy (%)
ResNet-101	2015-12	45	78.2
ViT-B/16	2020-10	87	85.4
ViT-L/16	2020-10	305	86.8
Swin-B	2021-08	88	86.4
Swin-L	2021-08	197	87.3
ConvNeXt-T	2022-03	29	82.9
ConvNeXt-S	2022-03	50	85.8
ConvNeXt-B	2022-03	89	86.8
ConvNeXt-L	2022-03	198	87.5
ConvNeXt-XL	2022-03	350	87.8

TABLE 4.2 – Comparaison des performances d'accuracy sur ImageNet-1K, pré-entraîné sur ImageNet-22K pour des images de taille  $384^2$

À taille équivalente, on obtient un petit gain de performance ! Ainsi, un ajustement moderne apporté par un autre domaine du Machine Learning permet de gagner en performance sur la

vision. Notons également que l'accuracy de ConvNeXt-XL est meilleure que celle de ConvNeXt-L. Si il peut sembler trivial que plus de paramètres donnent de meilleures performances pour les Transformers, ce n'était pas un consensus pour les réseaux convolutionnels.

Nous avons présenté dans cette séance des techniques pour la plupart non destinées en premier lieu aux LLM, mais qui sont devenues standards grâce à eux. Nous avons pu faire converger les remarques des précédentes séances à celle-ci pour concrètement améliorer n'importe quel réseau de neurones. Cela démontre l'importance d'une recherche diversifiée, et l'importance de faire des liens entre des domaines à priori *trop* différents.

Si toutes les améliorations ne sont pas forcément concrètes ni bien comprises, nous voyons comment la recherche est devenue plus opérationnelle. L'emphasis dans les papiers de recherche sur l'efficacité de calcul et la possibilité d'arbitrage de sélection de modèle parmi une famille de taille différente montre l'impact positif de la mise en production de telles techniques.



# Annexes

## Annexe A

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

### A.1 Définition et propriétés

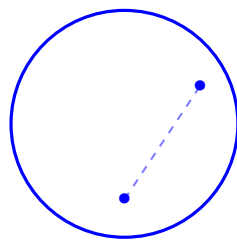
#### A.1.1 Ensemble et fonction convexe

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 :

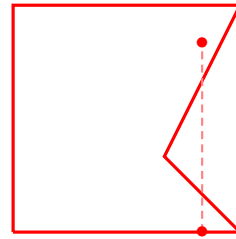
**Définition A.1.** Soit  $A$  un sous-ensemble de  $\mathbb{R}^d$ . On dit que  $A$  est un ensemble convexe si :

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

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 (A.1).



(a) Ensemble convexe



(b) Ensemble non convexe

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

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

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 (A.2).

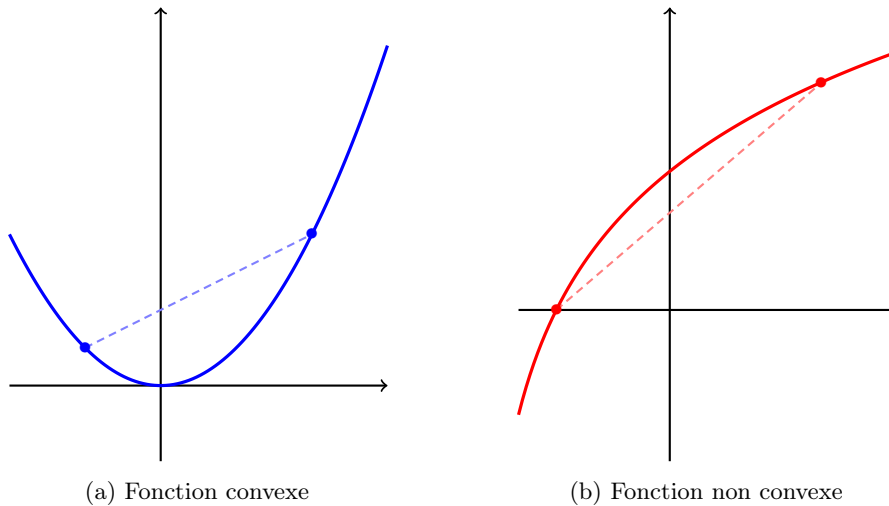


FIGURE A.2 – Exemple et contre exemple de fonction convexe

**Exercice A.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.

□

### A.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 A.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.  $\forall x, y \in A, f(y) \geq f(x) + \nabla \langle f(x), y - x \rangle$
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 reconnaît 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 (A.3).

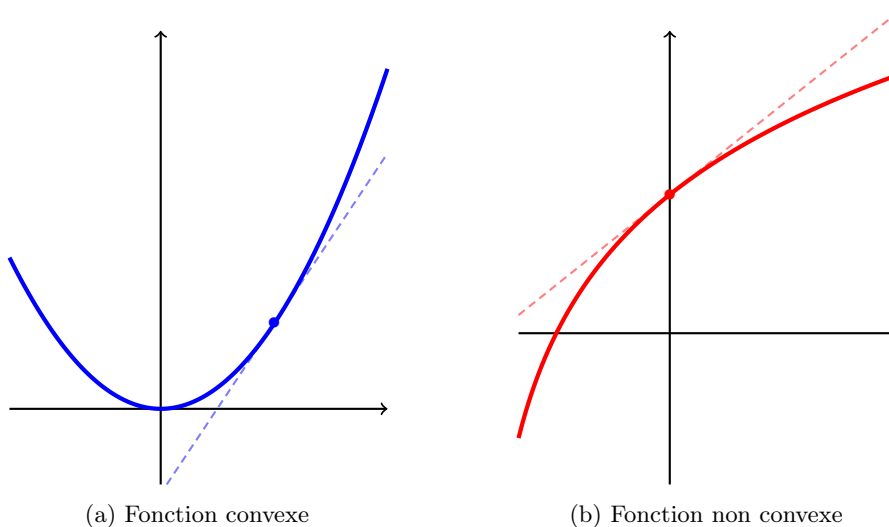


FIGURE A.3 – Illustration de la propriété de sous-estimateur pour une fonction **convexe** et contre exemple pour une fonction **non convexe**

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 (A.1)

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

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

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

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

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

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 :

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

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

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

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

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

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

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

On en déduit donc que :

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

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

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

## A.2 Résultats d'optimisations

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

$$x^* = \arg \min_{x \in \mathbb{R}^d} f(x) \quad (\text{A.1})$$

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 A.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  $\pi$ ) 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 A.1.** Soit un problème d'optimisation sans contraintes comme présenté dans (A.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 A.3.** Prouver la proposition (A.1) à l'aide du théorème (A.1).

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

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

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

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

Qui est bien la définition d'un minimum global d'une fonction.  $\square$

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

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

**Exercice A.4.** *Prouver la proposition (A.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  $z = \frac{a+b}{2} \in A$  comme combinaison convexe de deux points du domaine (avec  $t = \frac{1}{2}$ ). 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.  $\square$

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.

## A.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  $f$ .

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

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

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

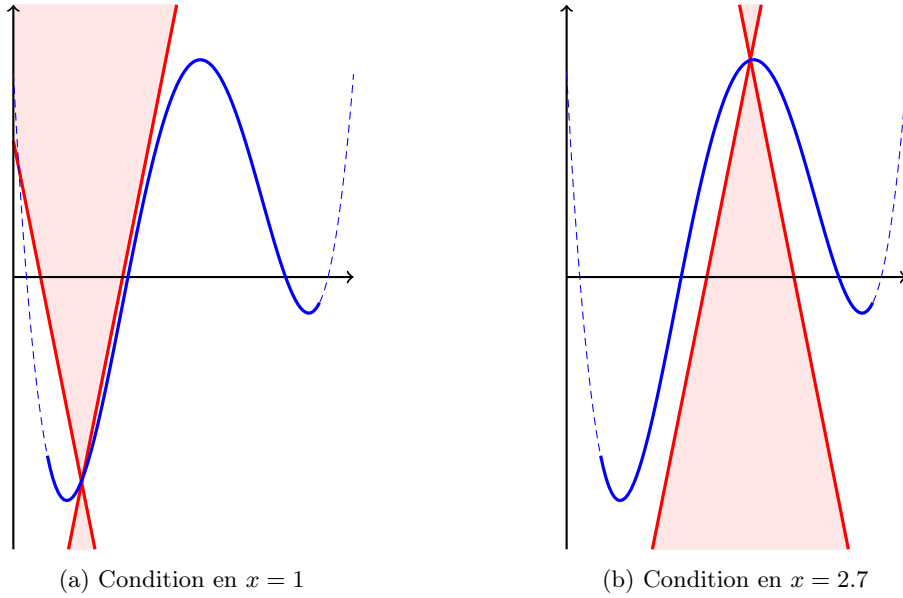


FIGURE A.4 – Illustration de la condition lipschitzienne pour une fonction

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 A.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  $\eta = \frac{\varepsilon}{L^2}$ . Alors en  $T = \frac{L^2\|x^* - x_0\|^2}{\varepsilon^2}$  itérations, l'algorithme converge vers  $x^*$  avec une erreur de  $\varepsilon$ .

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 maîtrise. Si l'on note  $\tilde{x} \in \mathbb{R}^d$  le point que l'on obtient à la suite de la descente de gradient, on a  $f(\tilde{x}) \leq f(x^*) + \varepsilon$ .

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



**Lemme A.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 :

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

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

*Solution.*

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

En exploitant le fait que  $f$  soit convexe et  $L$ -Lipschitzienne pour la dernière inégalité.  $\square$

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

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

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

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

### A.3.2 Fonction $\beta$ -smooth

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

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

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

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

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

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

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

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

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

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

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.

## A.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<sup>1</sup> 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.

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

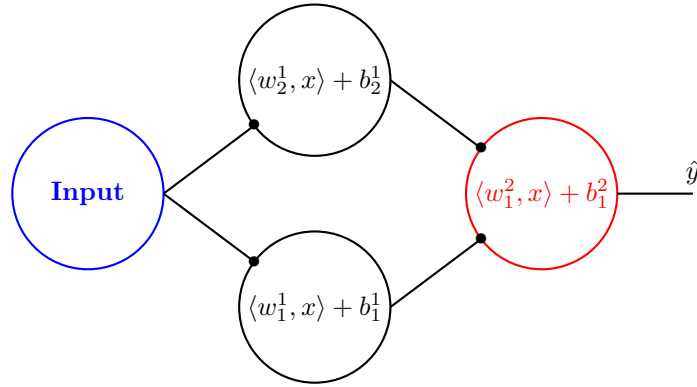


FIGURE A.5 – Réseau de neurone avec une couche caché de deux neurones, fonction d'activation ReLU

**Exercice A.7** (Combinaison de fonctions convexes). Soient  $U, V \subseteq \mathbb{R}$  et  $f : U \rightarrow V$  et  $g : V \rightarrow \mathbb{R}$  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. Soit  $x, y \in U$  et  $\lambda \in [0, 1]$ . Puisque  $f$  est convexe, on a :

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

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  $f''(x) \geq 0$  et  $g''(x) \geq 0$  avec le théorème A.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 A.8.** On considère le réseau de neurones défini à la figure A.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  $\theta_1 : w_1^1 = 1, w_2^1 = -1, w_1^2 = (1, -1)$  on a  $\mathcal{L}(\theta_1) = 0$
2. Montrer que pour  $\theta_2 : w_1^1 = -1, w_2^1 = 1, w_1^2 = (-1, 1)$  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  $\theta_1$  et  $\theta_2$  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.  $\square$

Nous avons donc qu'avec un réseau de neurones aussi simple que le réseau de la figure A.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 B

# Dualité entre les fonctions convexes et les fonctions de pertes bien définies

La notion de dérivée est largement utilisé dans la partie précédente, pour caractériser une fonction convexe par exemple. Ce concept est largement utilisé en mathématiques plus généralement pour être capable de quantifier le comportement locale d'une fonction. Cependant, une fonction convexe n'est pas forcément différentiable partout. Il nous faudrait un outil plus général que la dérivation mais qui permette d'atteindre un but similaire. C'est la notion de sous-différentiel.

**Définition B.1** (Sous-différentiel). Soit  $\varphi : V \rightarrow \mathbb{R}$  une fonction définie pour  $V \subseteq \mathbb{R}$ . On dit que  $t \in \mathbb{R}$  est un sous-gradient de  $\varphi$  en  $v \in V$  si :

$$\forall v' \in V, \quad \varphi(v') \geq \varphi(v) + (v' - v)t$$

Ou de manière équivalente :

$$vt - \phi(v) = \max_{v' \in V} v't - \varphi(v')$$

On appelle l'ensemble des sous-gradients le sous-différentiel.

Pour comprendre la définition, considérons la fonction valeur absolue. Elle est convexe, mais pas différentiable en 0.

Avec la figure B.1 on a l'exemple de deux sous-gradient qui appartiennent au sous-différentiel. On a aussi un contre-exemple : la droite tracé n'est pas toujours une minorante comme demandé dans la première définition. Ainsi, le sous-différentiel de la valeur absolue en 0 est l'intervalle  $[-1, 1]$ .

On comprend que si le sous-différentiel est réduit à un singleton, alors c'est que la fonction considéré est différentiable en ce point et que la valeur du singleton est la dérivée en ce point. De même, si zéro appartient au différentiel d'un point, alors c'est que la fonction atteint son minimum en ce point. Nous avons donc bien une généralisation de la notion de différentiabilité !

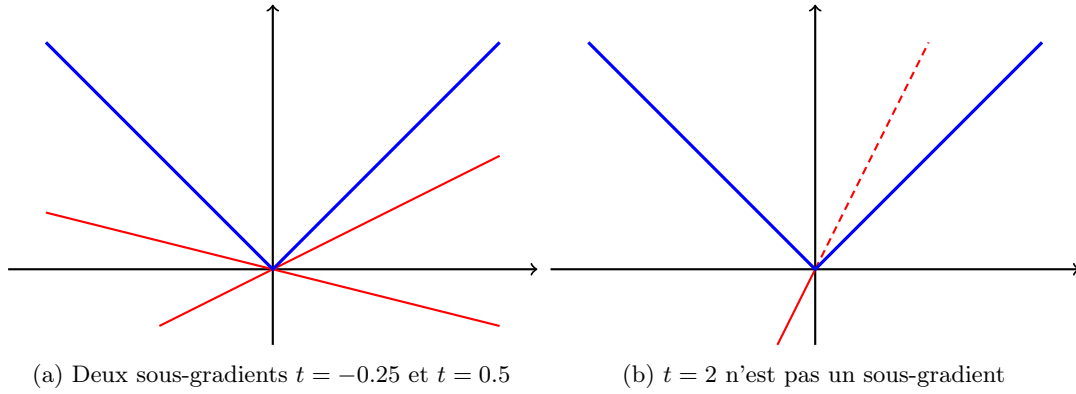


FIGURE B.1 – Fonction  $x \mapsto |x|$  avec exemple et contre-exemple de sous-gradient

**Définition B.2** (Divergence de Fenchel-Young). Soit  $V, T \subseteq \mathbb{R}$ . On définit la divergence de Fenchel-Young  $D_{\varphi, \psi} : V \times T \rightarrow \mathbb{R}$  pour une paire de fonction  $(\varphi, \psi)$  par :

$$\forall v \in V, \forall t \in T, \quad D_{\varphi, \psi}(v, t) = \varphi(v) + \psi(t) - vt$$

$\varphi : V \rightarrow \mathbb{R}$  (blue arrow from  $V$  to  $\varphi$ )  
 $\psi : T \rightarrow \mathbb{R}$  (red arrow from  $T$  to  $\psi$ )

Considérons une fonction  $\varphi : V \rightarrow \mathbb{R}$  et  $\psi : T \rightarrow \mathbb{R}$  définie avec  $V, T \subseteq \mathbb{R}$ . Essayons de lier les deux précédentes notions en commençant par supposer que  $t \in T$  est un sous-gradient de  $\varphi$  en  $v \in V$  :

$$\begin{aligned}
 vt - \varphi(v) &= \max_{v' \in V} (v't - \varphi(v')) \iff vt - \varphi(v) - \psi(t) = \max_{v' \in V} (v't - \varphi(v') - \psi(t)) \\
 &\iff \varphi(v) + \psi(t) - vt = \min_{v' \in V} (v't - \varphi(v') - \psi(t)) \\
 &\iff D_{\varphi, \psi}(v, t) = \min_{v' \in V} D_{\varphi, \psi}(v', t)
 \end{aligned}$$

Ainsi,  $t \in T$  est un sous-gradient de  $\varphi$  en  $v \in V$  si, et seulement si,  $D_{\varphi, \psi}(v, t) = \min_{v' \in V} D_{\varphi, \psi}(v', t)$ .

**Exercice B.1.** En s'inspirant de la démonstration précédente, montrer que  $v$  est un sous-gradient de  $\psi$  en  $t \in T$  si, et seulement si,  $D_{\varphi, \psi}(v, t) = \min_{t' \in T} D_{\varphi, \psi}(v, t')$

Avec le résultat précédent et celui de l'exercice, on comprend que si l'on suppose que  $D_{\varphi, \psi}(v', t') \geq 0$  pour tout  $v' \in V$  et  $t' \in T$  mais qu'il existe  $v \in V$  et  $t \in T$  tels que  $D_{\varphi, \psi}(v, t) = 0$  alors  $t$  est un sous-gradient de  $\varphi$  en  $v$  et  $v$  est un sous-gradient de  $\psi$  en  $t$ .

On saisit l'intérêt de l'introduction de la divergence de Fenchel-Young : elle permet de lier la notion de sous-gradient d'une paire de fonction. C'est ce lien que l'on va exploiter pour montrer comment on peut induire une fonction convexe depuis une fonction de perte bien définie.

**Proposition B.1.** Soit  $V \subseteq [0, 1]$  un intervalle non vide. Soit  $\mathcal{L} : \{0, 1\} \times V \mapsto \mathbb{R}$  une fonction de perte bien définie. Pour tout  $v \in V$  on définit  $\text{dual}(v) = \mathcal{L}(0, v) - \mathcal{L}(1, v)$ . Alors, il existe une fonction convexe  $\varphi : V \rightarrow \mathbb{R}$  et une fonction convexe  $\psi : \mathbb{R} \rightarrow \mathbb{R}$  telles que :

$$\begin{aligned} \forall y \in \{0, 1\}, \forall v \in V, & \quad \mathcal{L}(y, v) = \psi(\text{dual}(v)) - y \text{dual}(v) \\ \forall v \in V, \forall t \in \mathbb{R}, & \quad D_{\varphi, \psi}(v, t) \geq 0 \\ \forall v \in V, & \quad D_{\varphi, \psi}(v, \text{dual}(v)) = 0 \\ \forall (t_1, t_2) \in \mathbb{R}, t_1 \neq t_2, & \quad 0 \leq \frac{\psi(t_2) - \psi(t_1)}{t_2 - t_1} \leq 1 \end{aligned}$$

La démonstration de ce résultat montre comment sont construite les fonctions  $\varphi$  et  $\psi$ . Elle s'appuie également sur une réécriture de la fonction de perte à l'aide de la fonction duale. Nous proposons de nous intéresser à une autre manière décrire la fonction de perte  $\mathcal{L}$  avec les deux fonctions induites  $\varphi$  et  $\psi$ .

**Exercice B.2** (Réécriture d'une fonction de perte bien définie). Montrer, avec les mêmes hypothèses et notation que la proposition (B.1), que l'on peut écrire la fonction de perte  $\mathcal{L}$  bien définie comme :

$$\forall y \in \{0, 1\}, \forall v \in V, \quad \mathcal{L}(y, v) = -\varphi(v) + (v - y) \text{dual}(v)$$

*Solution.* Soit  $v \in V$ . Avec la proposition (B.1), et les mêmes notations, on a :

$$\begin{aligned} D_{\varphi, \psi}(v, \text{dual}(v)) &= 0 \\ \varphi(v) + \psi(\text{dual}(v)) - v \text{dual}(v) &= 0 \end{aligned}$$

Mais on sait aussi que pour tout  $y \in \{0, 1\}$  :

$$\mathcal{L}(y, v) = \psi(\text{dual}(v)) - y \text{dual}(v) \iff \psi(\text{dual}(v)) = \mathcal{L}(y, v) + y \text{dual}(v)$$

Finalement, on a :

$$\begin{aligned} \varphi(v) + (\mathcal{L}(y, v) + y \text{dual}(v)) - v \text{dual}(v) = 0 &\iff \varphi(v) + \mathcal{L}(y, v) + (y - v) \text{dual}(v) = 0 \\ &\iff \mathcal{L}(y, v) = -\varphi(v) + (v - y) \text{dual}(v) \end{aligned}$$

□

Il nous reste à voir comment à partir de fonctions convexes on peut induire une fonction de perte bien définie.

**Proposition B.2.** Soit  $V \subseteq [0, 1]$  un interval non vide. Soit  $\varphi : V \rightarrow \mathbb{R}$ ,  $\psi : \mathbb{R} \rightarrow \mathbb{R}$  et  $\text{dual} : V \rightarrow \mathbb{R}$  des fonctions telles que :

$$\begin{aligned}\forall v \in V, \forall t \in \mathbb{R}, D_{\varphi, \psi}(v, t) &\geq 0 \\ \forall v \in V, D_{\varphi, \psi}(v, \text{dual}(v)) &= 0\end{aligned}$$

On définit la fonction de perte  $\mathcal{L} : \{0, 1\} \times V \rightarrow \mathbb{R}$  qui vérifie

$$\forall y \in \{0, 1\}, \forall v \in V, \quad \mathcal{L}(y, v) = \psi(\text{dual}(v)) - y \text{dual}(v)$$

Alors, on a :

1.  $\mathcal{L}$  est une fonction de perte bien définie
2.  $\varphi$  est convexe
3. Pour tout  $v \in V$ ,  $\text{dual}(v)$  est un sous-gradient de  $\varphi$  en  $v$ , et  $v$  est un sous-gradient de  $\psi$  en  $\text{dual}(v)$
4.  $\varphi(v) = v \text{dual}(v) - \psi(\text{dual}(v)) = - \mathbb{E}_{y \sim \text{Ber}(v)} [\mathcal{L}(y, v)]$
5. Si on définit  $\psi'(t) = \sup_{v \in V} vt - \varphi(v)$  alors  $\psi'$  est convexe et  $\psi'(\text{dual}(v)) = \psi(\text{dual}(v))$  pour tout  $v \in V$ . De plus, pour  $t_1, t_2$  deux réels distincts on a :

$$0 \leq \frac{\psi(t_2) - \psi(t_1)}{t_2 - t_1} \leq 1$$

De ces deux propositions nous voyons la correspondance entre une fonction de perte bien définie  $\mathcal{L} : \{0, 1\} \times V \rightarrow \mathbb{R}$  et  $(\varphi, \psi, \text{dual})$  qui vérifient  $D_{\varphi, \psi}(v, t) \geq 0$  et  $D_{\varphi, \psi}(v, \text{dual}(v)) = 0$ . D'après la proposition (B.2) en supposant que  $\psi$  est différentiable, on a que  $v = \nabla \psi(\text{dual}(v))$ .

Ceci conclut le lien qui permet de mieux comprendre le lien entre performance d'optimisation et qualité de calibration d'un algorithme de Machine Learning.



## Annexe C

# 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.

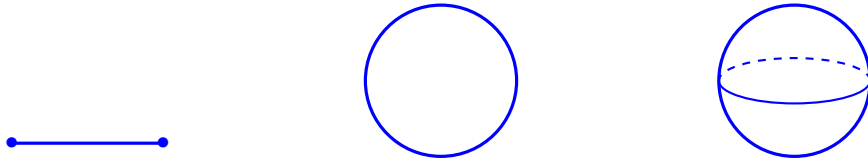
Le **fléau de la dimension** est une notion connue en statistiques et en Machine Learning. Ce terme rassemble tout un ensemble de phénomènes qui se produit en très grande dimension, mais pas dans une dimension plus petite. Nous proposons dans cette annexe d'illustrer quelques-uns des phénomènes étranges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

### C.1 Volume d'une hypersphère

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

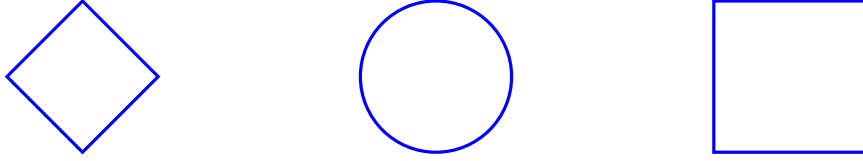
Avec la figure (C.1) nous avons l'intuition que le volume augmente avec la dimension. Donc pour une hypersphère de très grande dimension, on devrait avoir un très grand volume. Nous avons tracé et mesuré le volume pour la distance euclidienne classique, mais nous pouvons aussi utiliser d'autres distances (autre norme) comme montré dans la figure (C.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 =  $\pi$  (c) En dimension 3, volume =  $\frac{4}{3}\pi$

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

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

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 Lebesgue dans  $\mathbb{R}^n$ . Formellement :

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

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

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

*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  $\Gamma$  définie comme :*

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

*Démonstration.* Soit la fonction  $\phi$  définie comme :

$$\begin{aligned} \phi : \quad B_n^p(1) &\rightarrow B_n^p(R) \\ (x_1, \dots, x_n) &\mapsto (Rx_1, \dots, Rx_n) \end{aligned}$$

Par définition, si  $u \in B_n^p(1)$ , alors  $\|u\|_p^p \leq 1 \iff \|Ru\|_p^p \leq R^p$  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  $\phi(B_n^p(1)) = \phi(B_n^p(R))$ .  $\phi$  est donc une bijection, ainsi par changement de variable :  $V_n^p(R) = R^n V_n^p(1)$ .

Par le théorème de Fubini :

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

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

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

La fonction Beta est liée 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 :

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

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

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

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 avons 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 C.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 (C.4).

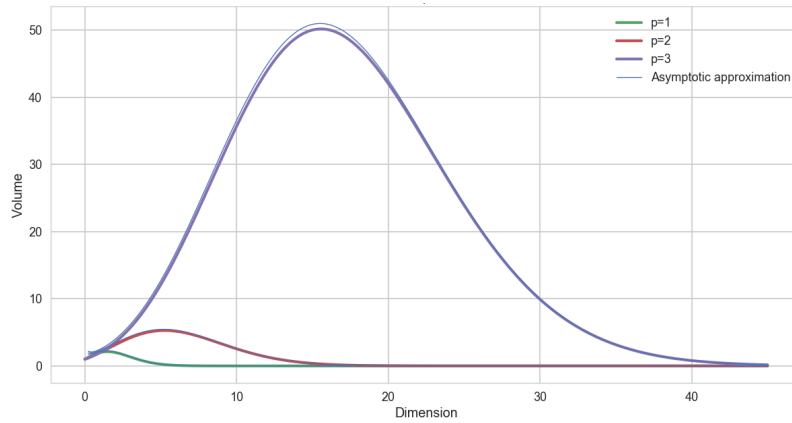


FIGURE C.3 – Volume de la boule unité en fonction de la dimension de son espace pour trois  $p$ -normes

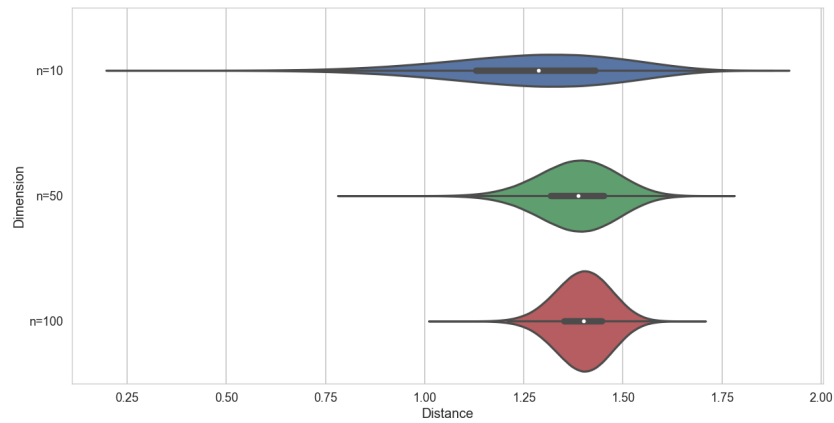


FIGURE C.4 – Distribution des distances entre chaque point en fonction de la dimension de l'espace

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.

## C.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 :

$$\langle x, y \rangle = \|x\|_2 \|y\|_2 \cos(\theta)$$

Avec  $\theta$  l'angle entre le représentant de  $x$  et le représentant de  $y$  à l'origine comme défini dans la figure (C.5).

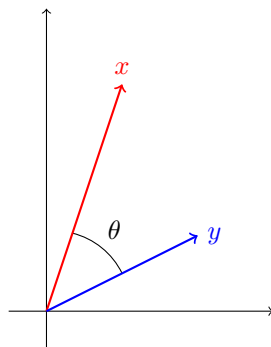


FIGURE C.5 – Définition de la métrique cosinus

Nous pouvons donc naturellement définir la métrique cosinus comme :

$$\text{cosine}(x, y) = \frac{\langle x, y \rangle}{\|x\|_2 \|y\|_2}$$

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 :

$$\begin{aligned} \|x - y\|_2^2 &= \|x\|_2^2 + \|y\|_2^2 - 2 \langle x, y \rangle \\ &= \|x\|_2^2 + \|y\|_2^2 - 2 \text{cosine}(x, y) \|x\|_2 \|y\|_2 \\ &= 2 [1 - \text{cosine}(x, y)] \end{aligned}$$

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

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

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  $X = \langle a, x \rangle$ , alors il est simple de montrer que  $\mathbb{E}[X] = 0$  et  $\mathbb{E}[X^2] = \frac{1}{n}$ . Ainsi, en exploitant l'inégalité de Chernoff on a :

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

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

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

□

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 qualité 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.

### C.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 Balestrieri, Jerome Pesenti et Yann Le Cun ont publié en 2021 l'article *Learning in High Dimension always amount to extrapolation* [Balestrieri 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 Balestrieri, 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*.
- [Ali et al., 2023] Ali, M., Fromm, M., Thellmann, K., Rutmann, R., Lübbering, M., Leveling, J., Klug, K., Ebert, J., Doll, N., Buschhoff, J. S., et al. (2023). Tokenizer choice for llm training : Negligible or crucial? *arXiv preprint arXiv :2310.08754*.
- [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.
- [Balestrierio et al., 2021] Balestrierio, R., Pesenti, J., and LeCun, Y. (2021). Learning in high dimension always amounts to extrapolation. *arXiv preprint arXiv :2110.09485*.
- [Bello et al., 2019] Bello, I., Zoph, B., Vaswani, A., Shlens, J., and Le, Q. V. (2019). Attention augmented convolutional networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3286–3295.
- [Bender et al., 2021] Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S. (2021). On the dangers of stochastic parrots : Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 610–623.
- [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*.
- [Broder, 1997] Broder, A. Z. (1997). On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*.
- [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*.

- [Carlini et al., 2022] Carlini, N., Ippolito, D., Jagielski, M., Lee, K., Tramer, F., and Zhang, C. (2022). Quantifying memorization across neural language models. *arXiv preprint arXiv :2202.07646*.
- [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*.
- [Computer, 2023] Computer, T. (2023). Redpajama : an open dataset for training large language models.
- [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*.
- [Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert : Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv :1810.04805*.
- [Dosovitskiy et al., 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words : Transformers for image recognition at scale. *arXiv preprint arXiv :2010.11929*.
- [Dubey et al., 2024] Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. (2024). The llama 3 herd of models. *arXiv preprint arXiv :2407.21783*.
- [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*.
- [Gage, 1994] Gage, P. (1994). A new algorithm for data compression. *The C Users Journal*.
- [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.

- [Gunasekar et al., 2023] Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Del Giorno, A., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., et al. (2023). Textbooks are all you need. *arXiv preprint arXiv :2306.11644*.
- [Hayase et al., 2024] Hayase, J., Liu, A., Choi, Y., Oh, S., and Smith, N. A. (2024). Data mixture inference : What do bpe tokenizers reveal about their training data? *arXiv preprint arXiv :2407.16607*.
- [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*.
- [Jiang et al., 2023] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. (2023). Mistral 7b. *arXiv preprint arXiv :2310.06825*.
- [Joulin et al., 2016] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv :1607.01759*.
- [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)*.
- [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*.
- [Kudo, 2018] Kudo, T. (2018). Subword regularization : Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv :1804.10959*.
- [Kudo and Richardson, 2018] Kudo, T. and Richardson, J. (2018). Sentencepiece : A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv :1808.06226*.
- [LeCun et al., 1998a] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.
- [LeCun et al., 2010] LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]*. Available : <http://yann.lecun.com/exdb/mnist>, 2.
- [LeCun et al., 1998b] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (1998b). Efficient backprop. *Tricks of the Trade*.
- [Lee et al., 2021] Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C., and Carlini, N. (2021). Deduplicating training data makes language models better. *arXiv preprint arXiv :2107.06499*.
- [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., 2024] Liu, Y., Cao, J., Liu, C., Ding, K., and Jin, L. (2024). Datasets for large language models : A comprehensive survey. *arXiv preprint arXiv :2402.18041*.
- [Liu et al., 2021] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer : Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*.
- [Liu et al., 2022] Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. (2022). A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986.
- [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)*.
- [Maslej et al., 2024] Maslej, N., Fattorini, L., Perrault, R., Parli, V., Reuel, A., Brynjolfsson, E., Etchemendy, J., Ligett, K., Lyons, T., Manyika, J., Niebles, J. C., Shoham, Y., Wald, R., and Clark, J. (2024). The ai index 2024 annual report. *AI Index Steering Committee, Institute for Human-Centered AI, Stanford University*.
- [Mitchell, 1988] Mitchell, D. E. (1988). The extent of visual recovery from early monocular or binocular visual deprivation in kittens. *The Journal of physiology*.
- [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., 2024] Penedo, G., Kydlíček, H., von Werra, L., and Wolf, T. (2024). Fineweb.
- [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*.
- [Popel and Bojar, 2018] Popel, M. and Bojar, O. (2018). Training tips for the transformer model. *arXiv preprint arXiv :1804.00247*.
- [Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*.
- [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*.
- [Raffel et al., 2019] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*.
- [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.
- [Schuster and Nakajima, 2012] Schuster, M. and Nakajima, K. (2012). Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*.
- [Sennrich et al., 2015] Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv :1508.07909*.
- [Shazeer, 2020] Shazeer, N. (2020). Glu variants improve transformer. *arXiv preprint arXiv :2002.05202*.
- [Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556*.
- [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.
- [Smith et al., 2023] Smith, S. L., Brock, A., Berrada, L., and De, S. (2023). Convnets match vision transformers at scale. *arXiv preprint arXiv :2310.16764*.
- [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*.
- [Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.
- [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.
- [Team et al., 2024] Team, G., Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivière, M., Kale, M. S., Love, J., et al. (2024). Gemma : Open models based on gemini research and technology. *arXiv preprint arXiv :2403.08295*.
- [Team, 2024] Team, Q. (2024). Qwen2.5 : A party of foundation models. <https://qwenlm.github.io/blog/qwen2.5/>.
- [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.
- [Wu et al., 2020] Wu, B., Xu, C., Dai, X., Wan, A., Zhang, P., Yan, Z., Tomizuka, M., Gonzalez, J., Keutzer, K., and Vajda, P. (2020). Visual transformers : Token-based image representation and processing for computer vision. *arXiv preprint arXiv :2006.03677*.
- [Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google’s neural machine translation system : Bridging the gap between human and machine translation. *arXiv preprint arXiv :1609.08144*.
- [Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist : a novel image dataset for benchmarking machine learning algorithms. *CoRR*.
- [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.
- [Yang et al., 2024] Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., et al. (2024). Qwen2 technical report. *arXiv preprint arXiv :2407.10671*.
- [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*.