Réseaux de neurones

Rapport de stage 2024-2025

Quentin Behague

École normale supérieure de Rennes

Encadré par François Malgouyres
Institut de Mathématiques de Toulouse (IMT)

Table des matières

1 Introduction

À l'aube de l'informatique moderne, certains problèmes, comme la classification automatique d'images, se sont rapidement révélés difficiles à résoudre avec les approches classiques. Reconnaître un chiffre manuscrit, distinguer un visage ou identifier un objet dans une image nécessitait une compréhension du contexte, une tolérance aux variations, et une flexibilité dont les algorithmes traditionnels, trop déterministes, étaient dépourvus. Ces limitations mettent en lumière la nécessité de méthodes capables d'apprendre à partir des données.

C'est dans ce contexte qu'émergent, au milieu du XXe siècle, les premiers travaux sur les réseaux de neurones artificiels. Inspirés par le fonctionnement du cerveau humain, Warren McCulloch et Walter Pitts proposent dès 1943 un premier modèle de neurone formel, créant ainsi l'unité élémentaire des réseaux de neurones artificiels modernes. Toutefois, les réseaux de neurones connaissent une évolution lente, freinée par l'incapacité des perceptrons simples à résoudre des problèmes non linéaires. C'est seulement en 1986 que David Rumelhart met au point l'algorithme de rétropropagation du gradient dans les réseaux multicouches.

Les réseaux de neurones commencent ainsi à démontrer un réel potentiel dès 1986. Mais c'est surtout au tournant des années 2010, avec l'essor du *deep learning* —dû à la puissance de calcul des ordinateurs qui ne cesse d'augmenter et à l'abondance de données numériques —que ces modèles révèlent toute leur puissance, en particulier pour des tâches longtemps considérées comme inaccessibles, comme la classification d'images.

Le fil conducteur de ce document est de comprendre les fondements mathématiques de l'apprentissage supervisé et d'analyser le fonctionnement des modèles de type réseaux de neurones. Notre objectif est, dans un premier de présenter les notions de base (fonction de risque, classes de prédicteurs, phénomènes de sous-apprentissage et sur-apprentissage, descente de gradient).

Dans un second temps, nous introduisons les réseaux de neurones, en commençant par le neurone formel et le perceptron, pour ensuite aborder les architectures multicouches. Nous détaillons également la rétropropagation du gradient, qui permet d'entraîner efficacement ces modèles. Nous abordons brièvement la question de la régularisation.

Enfin, une partie est consacrée aux réseaux ReLU. Nous mettons en lumière certaines de leurs propriétés géométriques et analytiques (non-coercivité, non-convexité, motifs d'activation, structure en régions polyédrales), et montrons comment ces caractéristiques influencent leur optimisation et leur capacité d'approximation.

Les codes Python ayant permis d'établir les figures de ce document et de réaliser les expériences durant le stage sont disponibles à l'adresse suivante :

https://github.com/QuentinBehagueENS/Neural-Networks

2 Apprentissage supervisé

L'apprentissage supervisé est une méthode d'apprentissage automatique dans laquelle un modèle est entraîné à partir de données étiquetées, c'est-à-dire des exemples pour lesquels la réponse attendue est connue. Chaque donnée d'entrée est associée à une sortie cible, l'objectif de l'algorithme est alors d'apprendre à généraliser et à prédire correctement sur de nouvelles données.

Dans cette section, nous introduisons la notion d'algorithme d'apprentissage ainsi que les outils permettant d'évaluer leur performance.

2.1 Généralités

Dans toute la suite, on considère deux espaces probabilisés $(\mathcal{X}, \mathcal{P}(\mathcal{X}), \mathbb{P}_X)$ et $(\mathcal{Y}, \mathcal{P}(\mathcal{Y}), \mathbb{P}_Y)$, on se donne (X, Y) un couple de variables aléatoires sur $\mathcal{X} \times \mathcal{Y}$. Les univers $\mathcal{X}$ et $\mathcal{Y}$ correspondent respectivement aux entrées et sorties possibles de l'algorithme d'apprentissage.
On supposera connu dans la suite un échantillon fini de données labélisées $(x_i, y_i)_{i \in [\![1,n]\!]} \in (\mathcal{X} \times \mathcal{Y})^n$. Ce jeu de donnée correspond aux observations disponibles à partir desquelles on souhaite associer un nouveau y (une classe) à un élément quelconque $x \in \mathcal{X}$.

Définition : (Algorithme d'apprentissage)
On appelle **algorithme d'apprentissage** une fonction $\mathcal{A} : (\mathcal{X}, \mathcal{Y})^n \longrightarrow \mathcal{Y}^{\mathcal{X}}$. Le résultat d'un algorithme d'apprentissage est une **fonction de prédiction** $g : \mathcal{X} \longrightarrow \mathcal{Y}$ mesurable.

L'objectif d'un algorithme d'apprentissage est de déduire une fonction de prédiction à partir d'un jeu de données $(x_i, y_i)_{i \in [\![1,n]\!]} \in (\mathcal{X} \times \mathcal{Y})^n$. L'efficacité d'un tel algorithme se mesure par l'écart entre la prédiction $g(x)$ et la valeur attendue y, pour tout couple $(x, y) \in \mathcal{X} \times \mathcal{Y}$. Pour quantifier cet écart, on suppose connue une fonction de coût $L : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$ mesurable et telle que $(x, y) \longmapsto L(g(x), y)$ soit $\mathbb{P}_{(X,Y)}$-intégrable sur $\mathcal{X} \times \mathcal{Y}$, c'est-à-dire :

$$\int_{\mathcal{X} \times \mathcal{Y}} |L(g(x), y)| \, \mathrm{d}\mathbb{P}_{(X,Y)}(x, y) < +\infty$$

Cela garantit l'existence de son espérance et permet de définir la notion de risque en population :

Définition : (Risque en population)
On appelle **risque en population** associé à la fonction de prédiction g la quantité :

$$\mathcal{R}(g) := \mathbb{E}[L(g(X), Y)] = \int_{\mathcal{X} \times \mathcal{Y}} L(g(x), y) d\mathbb{P}_{(X,Y)}(x, y)$$

où $\mathbb{P}_{(X,Y)}$ désigne une loi de probabilité sur $\mathcal{X} \times \mathcal{Y}$.

On cherche alors à minimiser le risque, c'est-à-dire réduire l'erreur moyenne entre la prédiction et la valeur attendue. En pratique, on ne connaît pas les lois de X et Y, nous sommes donc incapables de calculer explicitement ce risque. C'est pourquoi on introduit la notion de **risque empirique**, une approximation du risque basée sur les données observées.

Définition : (Risque empirique)
Soit $(x_i, y_i)_{i \in [\![1,n]\!]} \in \mathcal{X} \times \mathcal{Y}$ un jeu de données, on appelle **risque empirique** associé à la fonction de prédiction g la quantité :

$$\hat{\mathcal{R}}(g) := \frac{1}{n} \sum_{i=1}^{n} L(g(x_i), y_i)$$

Selon de faibles hypothèse, il existe une fonction qui minimise cette le risque empirique, on l'appelle le prédicteur de Bayes. Ce prédicteur permet alors de définir un risque optimal.

Définition : (Prédicteur de Bayes)
Si $y \longrightarrow L(y, Y)$ est $\mathbb{P}_Y$-intégrable sur $\mathcal{Y}$ et si, pour tout $x \in \mathcal{X}, y \longmapsto \mathbb{E}[L(y, Y)|X = x]$ admet un minimiseur, alors, on pose $g_* : \mathcal{X} \longrightarrow \mathcal{Y}$ une fonction telle que :

$$\forall x \in \mathcal{X} : g_*(x) \in \underset{y \in \mathcal{Y}}{\operatorname{argmin}}\, \mathbb{E}[L(y, Y)|X = x]$$

Elle vérifie alors :

$$\forall g : \mathcal{X} \longrightarrow \mathcal{Y} : \mathcal{R}(g) \geq \mathcal{R}(g_*)$$

Démonstration : Tout d'abord, pour tout $x, y \in \mathcal{X} \times \mathcal{Y}$, la quantité $\mathbb{E}[L(y, Y)|X = x]$ est bien définie, car L est $\mathbb{P}_Y$-intégrable. De plus, une telle fonction g_* existe, on peut considérer $g_* : x \longmapsto y_x^*$.
Soit $g : \mathcal{X} \longrightarrow \mathcal{Y}$ une fonction de prédiction :

$$\begin{aligned} \mathcal{R}(g) &= \mathbb{E}(L(g(X), Y)) \\ &= \int_{\mathcal{X}\times\mathcal{Y}} L(g(x), y)\, \mathrm{d}\mathbb{P}_{(X,Y)}(x, y) \end{aligned}$$

On peut maintenant appliquer le théorème de Fubini car, par hypothèse, L et g sont mesurables, et $(x, y) \longmapsto L(g(x), y)$ est $\mathbb{P}_{(X,Y)}$-intégrable sur $\mathcal{X} \times \mathcal{Y}$, enfin, $\mathbb{P}_{(X,Y)}$ est une mesure de probabilité sur l'espace produit $\mathcal{X} \times \mathcal{Y}$, elle est donc en particulier σ-finie. Ainsi, pour tout x, $L(g(x), \cdot)$ est $\mathbb{P}_{Y|X=x}$-intégrable sur $\mathcal{Y}$, et en posant :

$$I(x) = \int_{\mathcal{Y}} L(g(x), y)\mathbb{P}_{Y|X}(y|x)$$

On sait que I est $\mathbb{P}_X$-intégrable sur $\mathcal{X}$, de plus

$$\begin{aligned} \mathcal{R}(g) &= \int_{\mathcal{X}} \left[\int_{\mathcal{Y}} L(g(x), y)\, \mathrm{d}\mathbb{P}_{Y|X}(y|x)\right] \mathrm{d}\mathbb{P}_X(x) \\ &= \int_{\mathcal{X}} \mathbb{E}[L(g(x), Y)|X = x]\mathrm{d}\mathbb{P}_X(x) \\ &\geq \int_{\mathcal{X}} \mathbb{E}[L(g_*(x), Y)|X = x]\mathrm{d}\mathbb{P}_X(x) \\ &= \mathcal{R}(g_*) \end{aligned}$$

□

Définition : (Risque optimal)
On appelle **risque optimal**, et on note $\mathcal{R}^*$, la quantité :

$$\mathcal{R}^* := \inf_{g \in \mathcal{Y}^{\mathcal{X}}} \mathcal{R}(g)$$

Remarque : Lorsque le prédicteur de Bayes existe, $\mathcal{R}^* = \mathcal{R}(g_*)$

Définition : (Excès de risque)
On appelle **excès de risque** d'une fonction de prédiction $g : \mathcal{X} \longrightarrow \mathcal{Y}$ la quantité $\mathcal{R}(g) - \mathcal{R}^* \geq 0$

L'excès de risque d'une fonction de prédiction g permet de déterminer si g est proche ou non d'une solution optimale. Cette définition est nécessaire car on ne dispose pas nécessairement d'une fonction g_* qui annule le risque. Ce phénomène est évident pour des classes de fonctions n'ayant pas assez de paramètres. On se retrouve en situation de sous-apprentissage (voir Section 2.3), et $\mathcal{R}^*$ devient grand, mais on peut toujours faire tendre l'excès de risque vers zéro.

2.2 Classes de fonctions de prédiction

Étant en pratique impossible de considérer toutes les fonctions de $\mathcal{Y}^{\mathcal{X}}$, on se restreint à des classes de fonctions dépendantes de certains paramètres. On cherche alors à faire varier ces paramètres pour diminuer le risque empirique.
Dans la suite on considère Θ un ensemble de paramètres tel que, pour tout $\theta \in \Theta, g_\theta : \mathcal{X} \longrightarrow \mathcal{Y}$ est une fonction de prédiction. On notera $\mathcal{F}_\Theta = \{g_\theta : \mathcal{X} \longrightarrow \mathcal{Y} \mid \theta \in \Theta\} \subset \mathcal{Y}^{\mathcal{X}}$. Dans le cadre des réseaux de neurones, Θ sera l'ensemble des valeurs que peuvent prendre les poids et biais du réseau.

L'objectif est désormais de minimiser le risque empirique en faisant varier θ :

$$\hat{\mathcal{R}}(g_\theta) = \frac{1}{n}\sum_{i=1}^{n} L(g_\theta(x_i), y_i)$$

En pratique, même pour des réseaux de neurones de profondeur $H = 2$, la minimisation du risque empirique n'a pas de solution en général. Si on note g_θ la prédiction du réseau, alors $\inf_{\theta\in\Theta} \hat{\mathcal{R}}(g_\theta)$ n'est pas nécessairement atteint [4]. Dans ce cas, on se contenteras d'un $\hat{\theta} \in \Theta$ pour lequel la quantité $\hat{\mathcal{R}}(\hat{\theta})$ est petite.

2.3 Sous-apprentissage, sur-apprentissage

2.3.1 Définition

Le sous-apprentissage (*underfitting*) et le sur-apprentissage (*overfitting*) sont deux problèmes courants en apprentissage automatique. Lors de l'entraînement d'un modèle, il est essentiel de trouver un bon équilibre entre sa capacité à apprendre les données d'entraînement et sa capacité à généraliser à de nouvelles données. C'est dans ce contexte qu'interviennent ces notions.

Le sur-apprentissage se produit lorsqu'un modèle apprend "trop bien" les données d'entraînement, au point de capturer le bruit, les exceptions ou les particularités non représentatives des données générales. Un modèle surentraîné présente généralement une excellente performance sur l'ensemble d'entraînement mais échoue à généraliser sur de nouvelles données. En d'autres termes, il minimise très bien la quantité $\hat{\mathcal{R}}(g_\theta)$, mais peine à minimiser $\mathcal{R}(g_\theta)$ ce qui mène à une mauvaise performance sur les ensembles de validation. Ce phénomène peut avoir pour cause un modèle trop complexe, une durée d'entraînement trop grande, ou encore une trop faible quantité de données d'entraînement.

Le sous-apprentissage survient lorsque le modèle est incapable d'apprendre correctement les relations présentes dans les données. Il ne parvient pas à obtenir de bonnes performances, ni sur les données d'entraînement, ni sur les données de test ($\hat{\mathcal{R}}(g_\theta)$ et $\hat{\mathcal{R}}(g_\theta)$ sont élevés). Cela provient souvent d'un modèle trop simple, ou d'une durée d'apprentissage trop courte.

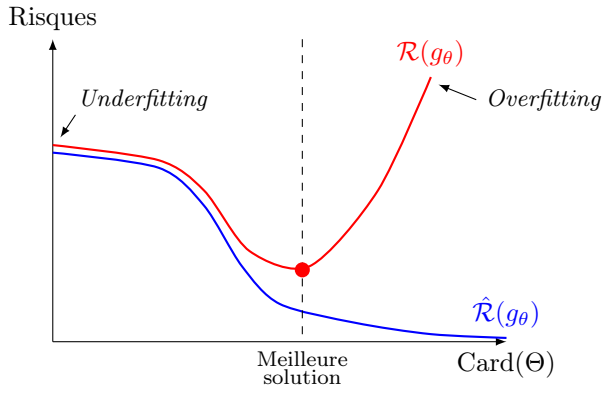

FIGURE 1 – Évolution typique des risques en fonction du nombre de paramètres

L'objectif est donc de trouver le bon équilibre entre ces deux extrêmes pour obtenir un modèle qui généralise bien. En pratique, les réseaux de neurones ne sont que peu, voire pas affectés par le problème de sur-apprentissage en raison du phénomène de régularisation implicite (Voir Section 5.1).

2.3.2 Exemple polynomial

Pour illustrer comment l'évolution du nombre de paramètres du modèle affecte ses performances, c'est-à-dire sa capacité à généraliser, nous allons employer des polynômes de degré croissant pour approximer un nuages de points. Les données d'entraînement sont de la forme $(x_i, y_i)_{i\in[\![1,20]\!]} \in ([-5,5]\times\mathbb{R})^{20}$. Ce sont des observations suivant la loi du couple de variable aléatoire (X,Y), avec $X \sim U([-5,5])$ et $Y \sim X^2 + U([-\epsilon,+\epsilon])$, avec $\epsilon = 3.3$.
Avec les notations précédentes, on a $\Theta = \mathbb{R}^{d+1}$ et $\mathcal{F}_\Theta = \mathbb{R}_d[X]$. On obtient les polynômes de la figure 2 de la manière suivante, on pose :

$$X = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^d \\ 1 & x_1 & x_1^2 & \cdots & x_1^d \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{20} & x_{20}^2 & \cdots & x_{20}^d \end{bmatrix} \in \mathcal{M}_{21,d+1}(\mathbb{R}) \quad \text{et} \quad y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{20} \end{bmatrix} \in \mathbb{R}^n$$

On fera l'hypothèse qu'on a choisit les données d'entraînement de sorte à ce que X soit de rang plein. On veut alors trouver $\tilde{\theta} \in \Theta$, i.e. des coefficients du polynôme, tel que :

$$\|X\tilde{\theta} - y\|^2 = \min_{\theta\in\mathbb{R}^{d+1}} \|X\theta - y\|^2$$

Comme X est de rang plein, $X^\top X$ est inversible, et la solution à ce problème est unique et est donnée par (voir annexe 7.2) :

$$\tilde{\theta} = (X^\top X)^{-1} X^\top y$$

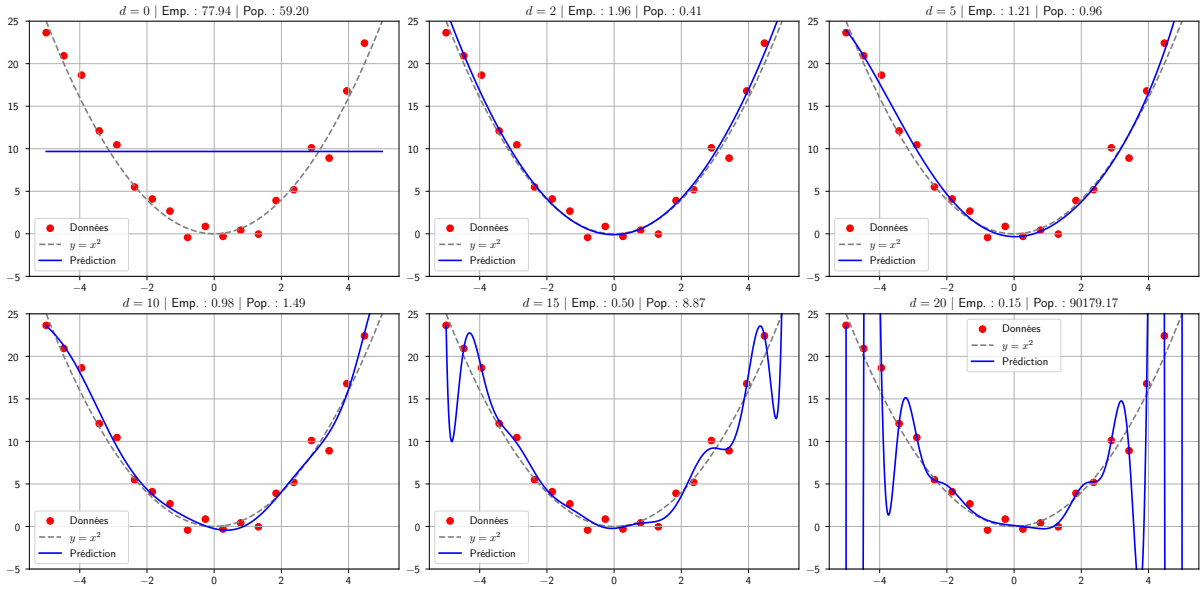

FIGURE 2 – Polynôme de différents degrés approximant un nuage de points

Pour $d = 0$ et $d = 1$, le modèle est en sous-apprentissage, il est incapable d'approximer correctement les données, en conséquence le risque empirique et le risque en population restent élevés.

Le modèle $d = 2$ correspond sans surprise à la meilleure paramétrisation (voir figure 3), au-delà, à partir de $d = 14$, on se retrouve dans une situation de sur-apprentissage évident, le risque empirique est presque nulle, mais la généralisation du modèle est mauvaise. Le nombre de paramètre utilisés est trop important pour des données nécessitant un modèle plus simple.

On peut ainsi observer dans un cas pratique, en figure 3, le comportement annoncé par la figure 1.

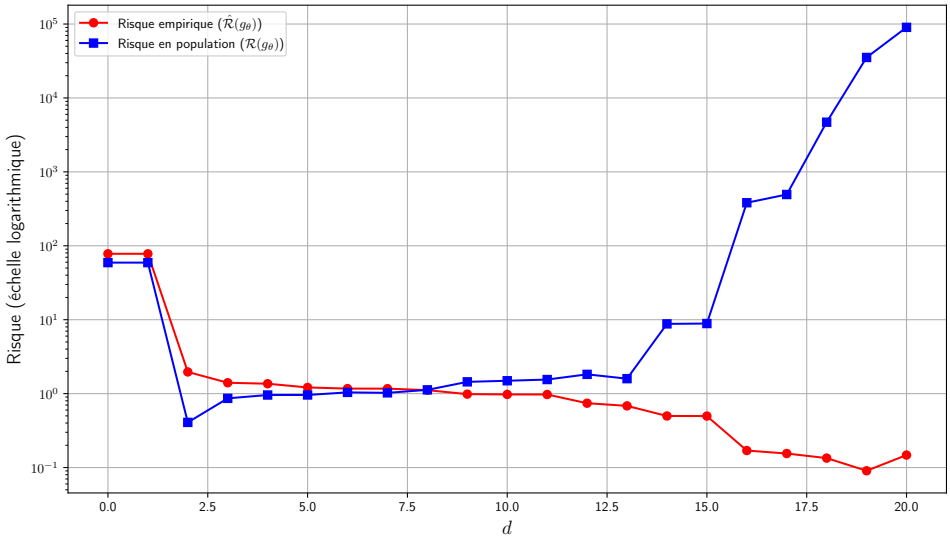

FIGURE 3 – Évolution du risque (après entrainement) pour des polynômes de degré croissant

3 Optimisation

3.1 Généralités

On se donne dans toute cette section $(E, \langle\cdot|\cdot\rangle)$ un espace hilbertien, $f : E \longrightarrow \mathbb{R}$ une fonction différentiable sur E, $x \in E$ et $X \subset E$.

Définition : (Problème d'optimisation)
Soit $\varepsilon > 0$, on appelle problème d'optimisation un problème consistant à déterminer $x^* \in X$ tel que :

$$\inf_{x \in X} f(x) - f(x^*) \leq \varepsilon$$

La fonction f est alors appelée fonction de coût. Un point $x \in E$ est dit admissible si $x \in X$. Quand la fonction admet un minimum le problème d'optimisation consistera simplement à trouver $\min_{x \in X} f(x)$. Dans tous les cas, X est appelé la contrainte.

Remarque : Dans ce document, nous ne traiterons que de l'optimisation numérique, c'est-à-dire $X \subset \mathbb{R}^n$. De plus, lorsqu'on se référera à un problème d'optimisation, on notera $\inf_{x \in X} f(x)$ par soucis de simplification, en négligeant ε.

Pour s'assurer qu'on ne sort pas du sous-ensemble X étudié, on définit la notion de direction admissible de la manière suivante.

Définition : (Direction admissible)
Soit $x \in X$. Une direction $d \in \mathbb{R}^n$ est admissible en x si :

$$\exists \varepsilon > 0, \forall \eta \in [0, \varepsilon], x + \varepsilon d \in X.$$

Définition : (Dérivée directionnelle)
On appelle dérivée directionnelle de f en $x \in E$ selon $v \in E$ la quantité :

$$D_v f(x) := \lim_{h \longrightarrow 0} \frac{f(x + hv) - f(x)}{h}.$$

Comme on a supposé f différentiable sur E, on a le résultat suivant :

Propriété 3.1.1 :

$$\forall x \in E, D_v f(x) = \langle \nabla f(x) | v \rangle$$

Propriété 3.1.2 :
Soit $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ différentiable sur $\mathbb{R}^n$ et X un ouvert de $\mathbb{R}^n$. Si x^* est un point de minimum local de f sur X alors :

$$\nabla f(x^*) = 0$$

.

3.2 Conditions suffisantes pour l'existence de solution

3.2.1 Coercivité

Définition : (Coercivité)
Une application $f : E \longrightarrow \mathbb{R}$ est dite **coercive** si :

$$\forall A \in \mathbb{R}, \exists R > 0, \forall x \in X, \|x\| \geq R \Longrightarrow f(x) \geq A.$$

Théorème 3.2.1 : Soient F un fermé non vide de $\mathbb{R}^n$ et $f : F \longrightarrow \mathbb{R}$ une application continue et coercive sur F. Alors :

$$\exists x_0 \in F, \forall x \in F, f(x_0) \leq f(x).$$

Démonstration : Soit $A \in \mathbb{R}$ suffisamment grand de sorte que :

$$K := f^{-1}(]-\infty, A])$$

soit non vide. On remarque que K est fermé (car image d'un fermé par une application continue) et borné (sinon, il existe une suite $(x_n)_{n\in\mathbb{N}}$ telle que $\|x_n\| \longrightarrow +\infty$ mais $f(x_n) \leq A$, ce qui contredit la coercivité). Donc K est compact et f admet un minimum global x_0 sur K. De plus :

$$\forall x \in F \backslash K, f(x) > A > f(x_0)$$

Donc x_0 est un minimum global de f sur F.

□

3.2.2 Convexité

Définition : (Convexité)
Soit $C \subset \mathbb{R}^n$ un ensemble convexe, la fonction $f : C \longrightarrow \mathbb{R}$ est dite **convexe** si :

$$\forall x, y \in C,\ \forall \lambda \in [0,1],\ f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$$

Théorème 3.2.2 : Soient $C \subset \mathbb{R}^n$ convexe et $f : C \to \mathbb{R}$ une fonction convexe continue sur C. Alors tout minimum local de f sur C est aussi un minimum global sur C. Autrement dit :

$$(\exists x_0 \in C, \forall x \in B(x_0, r) \cap C, f(x_0) \leq f(x)) \Rightarrow (\forall x \in C, f(x_0) \leq f(x))$$

Démonstration : Soit $x_0 \in C$ un minimum local de f, soit $x \in C$. Comme C est convexe, pour tout $\lambda \in [0,1]$:

$$x_\lambda := \lambda x + (1-\lambda) x_0 \in C.$$

Soit λ tel que $x_\lambda \in B(x_0, r) \cap C$ (qui existe car $x_\lambda \longrightarrow x_0$). Alors $f(x_0) \leq f(x_\lambda)$. Puis, par convexité de :

$$f(x_0) \leq f(x_\lambda) \leq \lambda f(x) + (1-\lambda) f(x_0).$$

Enfin :

$$\lambda f(x_0) \leq \lambda f(x) \Longrightarrow f(x_0) \leq f(x).$$

Ceci est vrai pour tout $x \in C$, donc x_0 est un minimum global de f sur C.

□

3.3 Algorithme de descente de gradient

3.3.1 Algorithme

Définition : (Direction de descente)
On appelle direction de descente de f en x tout vecteur $v \in E$ tel que :

$$D_v f(x) < 0$$

Remarque : Si $\nabla f(x) \neq 0$, la quantité $v = -\nabla f(x)$ est une direction de descente :

$$D_v f(x) = \langle \nabla f(x) | -\nabla f(x) \rangle = -\|\nabla f(x)\| < 0$$

Le fait que $-\nabla f(x)$ soit une direction de descente justifie intuitivement l'algorithme suivant.

Algorithme 1 : Descente de gradient

Entrées : $x_0 \in \mathbb{E}$, $\varepsilon > 0$, $f : \mathbb{E} \to \mathbb{R}$, $(\eta_k) \in \mathbb{R}_+^*$
Sortie : Un point approchant un minimum local de f
$k \leftarrow 0$
$x \leftarrow x_0$
tant que $\|\nabla f(x)\| > \varepsilon$ **faire**
  Calculer $\nabla f(x)$
  $x \leftarrow x - \eta_k \cdot \nabla f(x)$
  $k \leftarrow k + 1$
fin
retourner x

3.3.2 Décroissance et convergence de la descente de gradient discrète

Nous détaillons dans cette partie des conditions suffisante sur la suite $(\eta_k)_{k\in\mathbb{N}}$ et f pour que l'algorithme de descente de gradient fournisse une suite décroissante, ou convergente vers un minimum de f.

Définition : (Méthode de descente)
On dit qu'un algorithme est une méthode de descente, ou un algorithme de descente s'il fournit une suite $(x_k)_{k\in\mathbb{N}}$ telle que : $\forall k \in \mathbb{N}, f(x_{k+1}) \leq f(x_k)$

Théorème 3.3.1 : Inégalité de Taylor-Lagrange avec gradient lipschitzien
Soit $f : E \longrightarrow \mathbb{R}$ une fonction dérivable sur E, avec gradient L-lipschitzien. alors :

$$f(y) \leq f(x) + \langle \nabla f(x) | y - x \rangle + \frac{L}{2}\|x - y\|^2$$

Si de plus f est convexe, alors, on a :

$$f(x) + \langle \nabla f(x) | y - x \rangle \leq f(y) \leq f(x) + \langle \nabla f(x) | y - x \rangle + \frac{L}{2}\|x - y\|^2$$

Démonstration : Voir annexe 7.3

Propriété 3.3.1 :
Si $f : E \longrightarrow \mathbb{R}$ de classe $\mathcal{C}^1$, dont le gradient est L-lipschitzien, et si pour tout $k \in \mathbb{N}, \eta_k < \frac{2}{L}$, alors,

$$\begin{cases} x_0 \in E \\ x_{k+1} = x_k - \eta_k \nabla f(x_k) \end{cases}$$

est une méthode de descente.

Démonstration : Voir annexe 7.5.1.

Propriété 3.3.2 :
Si f est convexe, à gradient L-lipschitzien et admet un minimiseur x^*, alors l'algorithme de descente de gradient, pour $(\eta_k)_{k\in\mathbb{N}} = \left(\frac{1}{L}\right)_{k\in\mathbb{N}}$ converge vers un minimum de f, et, pour tout $k \in \mathbb{N}$:

$$f(x_n) - \min f \leq \frac{2L\|x_0 - x^*\|^2}{n}$$

Démonstration : Voir annexe 7.5.2

Remarque : On peut faire un autre choix que la direction de descente $d_k = -\eta_k \cdot \nabla f(x)$ (méthode de Cauchy).

On a démontré que dans le cas $\eta_k \leq \frac{L}{2}$, l'algorithme était une méthode de descende. Idéalement, pour le choix de η_k, il faut concilier deux objectifs contradictoires, trouver le meilleur pas η_k, i.e. minimisant la quantité de $f(x_k + \eta_k \nabla f(x_k))$, et effectuer peu de calcul. Pour cela, on effectue une recherche linéaire (voir [6], page 46 à 52).

3.3.3 Convergence de la descente de gradient continue

Théorème 3.3.2 : Si f est continue et différentiable sur E, convexe, coercive, que son gradient est L-lipschitz et si θ est solution de :

$$\begin{cases} \theta(0) &= \theta_0 \in E \\ \dot{\theta}(t) &= -\nabla f(\theta(t)) \end{cases}$$

Alors, $\lim\limits_{t \to +\infty} f(\theta(t))$ existe et :

$$\lim_{t \to +\infty} f(\theta(t)) \in \operatorname*{argmin}_{\theta \in E} f(\theta)$$

Démonstration : Tout d'abord, le minimum de f existe, soit $\theta^* \in \operatorname*{argmin}\limits_{\theta \in E} f(\theta)$ car f est coercive et convexe (voir 3.2). De plus $f(\theta(t))$ est décroissante :

$$\frac{\mathrm{d}f(\theta(t))}{\mathrm{d}t} = \nabla f(\theta(t))^\top \dot{\theta}(t) = \nabla f(\theta(t))^\top (-\nabla f(\theta(t))) = -\|\nabla f(\theta(t))\|^2 \leq 0$$

Comme f admet un minimum, $t \longmapsto f(\theta(t))$ est bornée inférieurement et décroissante, donc tend vers une limite f_∞. Ainsi, en posant $\theta_0 = \theta(0) < +\infty$, on en déduis que le gradient de $t \longmapsto f(\theta(t))$ est carré intégrable :

$$\frac{\mathrm{d}f(\theta(t))}{\mathrm{d}t} = -\|\nabla f(\theta(t))\|^2 \Longrightarrow \int_0^{+\infty} \|\nabla f(\theta(t))\|^2 \mathrm{d}t = f(\theta_0) - f_\infty < +\infty$$

Or, $t \longmapsto |\nabla f(\theta(t))|$ est positive, carré intégrable et lipschitzienne, donc elle converge vers $0 = |\nabla f(\theta^*)|$. Ainsi, par continuité de ∇f, $t \longmapsto \theta(t)$ converge vers l'unique minimum global θ^*.

□

3.3.4 Algorithme du gradient stochastique

Dans de nombreux problèmes d'optimisation, notamment en apprentissage automatique et plus particulièrement dans l'entraînement des réseaux de neurones profonds, le calcul exact du gradient de la fonction de coût est très coûteux, car il nécessite de parcourir l'ensemble du jeu de données à chaque itération.

Pour pallier ce problème, on utilise l'algorithme de descente de gradient stochastique. L'idée principale consiste à approximer le gradient en le calculant non pas sur l'ensemble des données, mais sur un sous-ensemble aléatoire, appelé mini-lot. À chaque itération, un nouveau mini-lot est échantillonné, ce qui rend l'algorithme bien plus rapide, tout en conservant une direction de descente convenable.

Cette approche introduit une forme de bruit aléatoire dans la direction de descente, c'est-à-dire qu'on descend plus rapidement, mais de manière plus incertaine. En pratique, ce bruit peut être bénéfique, notamment pour éviter de rester bloqué dans des minima locaux peu profonds ou des plateaux, en particulier dans les problèmes non convexes.

Tout comme pour l'algorithme de descende de gradient, on peut démontrer la convergence vers un minimum local sous certaines hypothèses.

3.4 Problème inverse

3.4.1 Généralité

En optimisation, un problème inverse désigne un problème où, au lieu de chercher directement une solution optimale à partir d'un modèle, on cherche les paramètres du modèle à partir d'observations ou contraintes sur le résultat optimal. Cela se traduit mathématiquement par l'ajout d'une pénalité $\lambda\tilde{f}(x)$ au problème classique de minimisation sans contrainte $\min f(x)$. Cette pénalité permet de tenir compte d'une certaine propriété qu'on souhaite que notre solution $x \in E$ vérifie.

Dans la plupart des cas, l'objectif du modèle étant la généralisation, on cherche à ce que ce terme force une forme de régularisation de la variable. Par exemple, pour un réseau de neurone, on transformerait le problème classique de minimisation du risque empirique (par soucis de simplification, on notera min en admettant qu'il est atteint) :

$$\min_{\theta\in\Theta} \hat{\mathcal{R}}(f_\theta)$$

En un problème :

$$\min_{\theta\in\Theta} \Big(\hat{\mathcal{R}}(f_\theta) + \lambda\mathcal{R}eg(f_\theta)\Big)$$

Dans lequel $\mathcal{R}eg \geq 0$ est une fonction de régularisation qui prend des valeurs élevées quand f_θ est à variation élevé, et prend des valeurs basses quand f_θ est plus "lisse", ou contient peu de zones affines par exemple.

3.4.2 Équivalence entre forme régularisée et forme contrainte

Définition : (Forme régularisée, forme contrainte)
Soit $\lambda \geq 0$, un problème d'optimisation est sous forme régularisé lorsqu'il est sous la forme :

$$(P_\lambda) \qquad \theta^* \in \operatorname*{argmin}_{\theta\in\Theta} \Big(\hat{\mathcal{R}}(f_\theta) + \lambda\mathcal{R}eg(f_\theta)\Big)$$

Soit $\tau \in \mathbb{R}$, un problème d'optimisation est sous forme contrainte lorsqu'il est sous la forme :

$$(\overline{P}_\tau) \qquad \theta^* \in \operatorname*{argmin}_{\substack{\theta\in\Theta \\ \mathcal{R}eg(f_\theta)\leq\tau}} \hat{\mathcal{R}}(f_\theta)$$

Propriété 3.4.1 :
Sous les hypothèses suivantes, en posant, pour tout $\theta \in \Theta$, pour tout $\lambda \geq 0$:

$$\mathcal{Q}(\lambda, \theta) := \hat{\mathcal{R}}(f_\theta) + \lambda\mathcal{R}eg(f_\theta)$$

1. $\theta \longrightarrow \mathcal{Q}(\lambda, \theta)$ est une fonction convexe continue et admet un unique minimiseur global.
2. $\theta \longmapsto \hat{\mathcal{R}}(\theta)$ est une fonction continue et convexe.
3. $\theta \longmapsto \mathcal{R}eg(f_\theta)$ est une fonction continue positive.

$$\exists\lambda > 0, \theta \text{ soit solution de } (P_\lambda) \iff \exists\tau \in \mathbb{R}, \theta \text{ soit solution de } (\overline{P}_\tau)$$

Démonstration :
Soit $\lambda > 0$ et $\theta_\lambda \in \Theta$ solution de (P_λ) (qui existe par hypothèse sur $\mathcal{Q}$) et posons $\tau = \mathcal{R}eg(f_{\theta_\lambda})$. Alors, on a, par définition de θ_λ pour tout $\theta \in \Theta$:

$$Q(\lambda, \theta_\lambda) \leq Q(\lambda, \theta)$$

En particulier, pour tout $\theta \in \Theta$ tel que $\mathcal{R}eg(f_\theta) \leq \tau$:

$$\hat{\mathcal{R}}(f_{\theta_\lambda}) + \lambda\mathcal{R}eg(f_{\theta_\lambda}) \leq \hat{\mathcal{R}}(f_\theta) + \lambda\mathcal{R}eg(f_\theta) \leq \hat{\mathcal{R}}(f_\theta) + \lambda\tau$$

$$\implies \hat{\mathcal{R}}(f_{\theta_\lambda}) - \hat{\mathcal{R}}(f_\theta) \leq 0$$

Donc, θ_λ est bien solution de $(\overline{P}_\tau)$ pour $\tau = \mathcal{R}eg(f_{\theta_\lambda})$.

Réciproquement, soit $\tau \in r$, intéressons nous en premier lieu au cas où une solution de $(\overline{P}_\tau)$, notée θ_τ, vérifie $\mathcal{R}eg(f_{\theta_\tau}) < \tau$. Dans ce cas, par continuité de $\mathcal{R}eg$, il existe un voisinage $\mathcal{V}_{\theta_\tau}$ de θ_τ tel que, pour tout $\theta \in \mathcal{V}_{\theta_\tau}, \mathcal{R}eg(f_\theta) < \tau$. Par définition, sur ce voisinage, θ_τ minimise $\hat{\mathcal{R}}(f_\theta)$. Il est donc solution du problème (P_λ) pour $\lambda = 0$ sur ce voisinage, de plus par hypothèse sur $\mathcal{Q}$, ce minimiseur local est l'unique minimiseur global, donc il est solution de (P_λ) sur Θ.

Traitons enfin le cas où $\mathcal{R}eg(f_{\theta_\tau}) = \tau$, il suffit de montrer qu'il existe $\lambda > 0$ et un voisinage $\mathcal{V}_{\theta_\tau}$ tel que θ_τ soit solution de (P_λ) sur ce voisinage. Posons g la fonction définie par :

$$\forall \sigma \in \mathbb{R}, g(\sigma) = \inf \left\{ \hat{\mathcal{R}}(f_\theta) \;\middle|\; \theta \in \Theta, \mathcal{R}eg(f_\theta) \leq \sigma \right\}$$

La fonction g est bien définie et convexe car son épigraphe est convexe, comme projection de l'ensemble convexe $\mathcal{E} := \left\{ (\theta, t, \sigma) \in \Theta \times \mathbb{R} \times \mathbb{R} \;\middle|\; \hat{\mathcal{R}}(f_\theta) \leq t, \mathcal{R}eg(f_\theta) \leq \sigma \right\}$). De plus, par hypothèse, $g(\tau) = \hat{\mathcal{R}}(f_{\theta_\tau})$. Par convexité, il existe $\lambda_\tau \in \mathbb{R}$ tel que :

$$\forall \sigma \in \mathbb{R}, g(\sigma) \geq g(\tau) + \lambda_\tau(\sigma - \tau)$$

En particulier, pour tout $\theta \in \Theta$:

$$\hat{\mathcal{R}}(f_\theta) \geq g(\mathcal{R}eg(f_\theta)) \geq g(\tau) + \lambda_\tau(\mathcal{R}eg(f_\theta) - \tau)$$

Donc :

$$\hat{\mathcal{R}}(f_\theta) + \lambda_\tau \mathcal{R}eg(f_\theta) \geq g(\tau) + \lambda_\tau \tau$$

On remarque alors qu'on a égalité pour $\theta = \theta_\tau$, donc θ minimise $\theta \longmapsto \hat{\mathcal{R}}(f_\theta) + \lambda_\tau \mathcal{R}eg(f_\theta)$, avec $\lambda_\tau \geq 0$ par décroissance de g.

□

4 Réseau de neurones

Les réseaux de neurones artificiels sont des modèles d'apprentissage inspirés du fonctionnement du cerveau humain. Ils sont composés de couches de neurones interconnectés. Grâce à leur capacité à modéliser des relations complexes et non linéaires, les réseaux de neurones sont devenus des outils puissants dans de nombreuses applications, comme la reconnaissance d'images ou la traduction automatique.

Dans cette section, nous introduisons la notion de réseau de neurones en présentant le fonctionnement du neurone formel, du perceptron et du perceptron multicouche. Nous étudions ensuite par quelle méthode un réseau apprend depuis un ensemble de données.

4.1 Neurone formel

Le neurone formel, introduit par McCulloch et Pitts en 1943, est un modèle informatique répliquant le fonctionnement d'un neurone biologique. Il constitue la base des réseaux de neurones modernes. Il prend en entrée $(x_i)_{1\leq i\leq n} \in \{0,1\}^n$ des valeurs booléennes, correspondant à l'activation de n précédents neurones, selon lesquelles il s'active ou non.

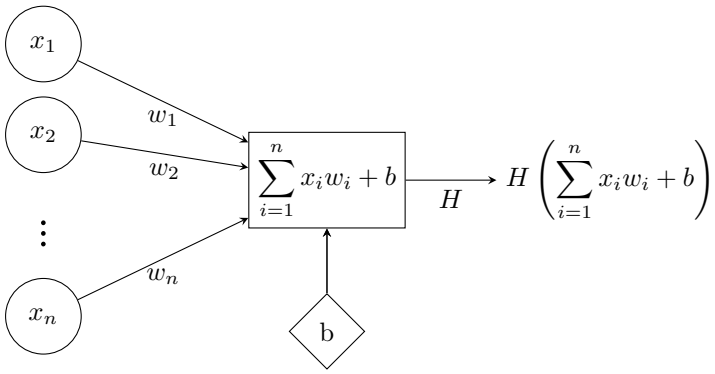

FIGURE 4 – Neurone formel

Pour $i \in [\![1, n]\!]$, le terme w_i représente l'influence du neurone i sur le neurone actuel. Le terme $b \in \mathbb{R}$ peut quant à lui être vu comme le seuil que doit dépasser la somme pondérée des activations des neurones précédents pour que le neurone s'active. L'activation du neurone peut uniquement prendre les valeurs 0 ou 1, on applique donc la fonction d'activation de Heaviside :

$$H : \begin{array}{ccc} \mathbb{R} & \longrightarrow & \mathbb{R} \\ x & \longmapsto & \begin{cases} 0 & \text{si } x < 0, \\ 1 & \text{sinon.} \end{cases} \end{array}$$

4.2 Perceptron

4.2.1 Principe

Le perceptron est le modèle le plus simple de réseau de neurones, inventé en 1957 par Frank Rosenblatt. Il s'agit d'un neurone formel muni d'une méthode d'apprentissage qui lui permet d'adapter automatiquement ses poids. Il est constitué d'un unique neurone, qui prend en entrée des valeurs réelles. Le perceptron est un classifieur linéaire capable de séparer deux classes à l'aide d'un hyperplan. Malgré sa simplicité, le perceptron a joué un rôle fondamental dans le développement de l'apprentissage automatique, en posant les premiers jalons des algorithmes d'apprentissage supervisé.

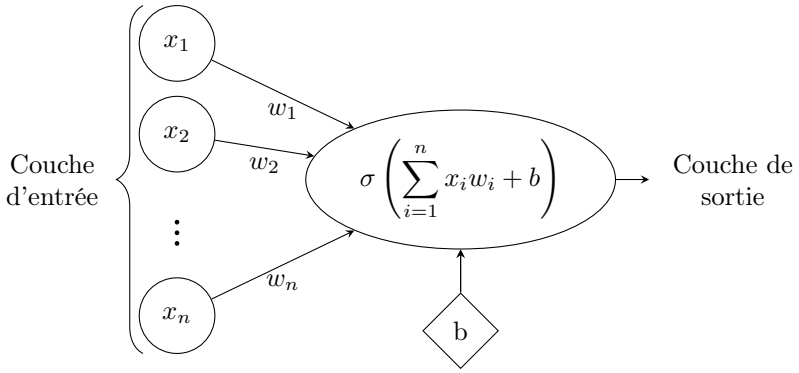

FIGURE 5 – Perceptron

Définition : (Fonction de prédiction associée à un perceptron)
On se donne $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ une fonction d'activation, $W = (w_i)_{i\in[\![1,n]\!]} \in \mathbb{R}^n$ une matrice de poids et on note g la fonction de prédiction (ou l'inférence) définie par :

$$g : \begin{array}{ccc} \mathbb{R}^n & \longrightarrow & \mathbb{R} \\ X = (x_i)_{i\in[\![1,n]\!]} & \longmapsto & \sigma(\langle W, X\rangle + b). \end{array}$$

4.2.2 Algorithme de Rosenblatt

Algorithme 2 : Algorithme du perceptron (apprentissage des poids)

Input : Un échantillon fini de données labellisées linéairement séparable
$\mathcal{S} = (x_i, y_i)_{i\in[\![1,n]\!]} \in (\mathcal{X} \times \mathcal{Y})^n$, $x_i \in \mathbb{R}^d$, $y_i \in \{0, 1\}$
Output : Les poids $\tilde{W} = (W, b) \in \mathbb{R}^{d+1}$ tel que, pour tout $(x_i, y_i) \in \mathcal{S}$:

$$\begin{cases} \langle W, x_i\rangle + b \geq 0 & \text{si } y_i = 1 \\ \langle W, x_i\rangle + b \leq 0 & \text{si } y_i = 0 \end{cases}$$

Initialiser $\tilde{W} \leftarrow 0_{\mathbb{R}^{n+1}}$
pour $i \leftarrow 1$ **to** n **faire**
  si $y_i = 1$ *et* $\langle \tilde{W}, x_i\rangle \leq 0$ **alors**
    $\tilde{W} \leftarrow \tilde{W} + x_i$
  fin
  si $y_i = 0$ *et* $\langle \tilde{W}, x_i\rangle \geq 0$ **alors**
    $\tilde{W} \leftarrow \tilde{W} - x_i$
  fin
fin
retourner $\tilde{W}$

Propriété 4.2.1 :
L'algorithme du perceptron de Rosenblatt converge si et seulement si l'échantillon $\mathcal{S}$ est linéairement séparable.

Démonstration Voir [5].

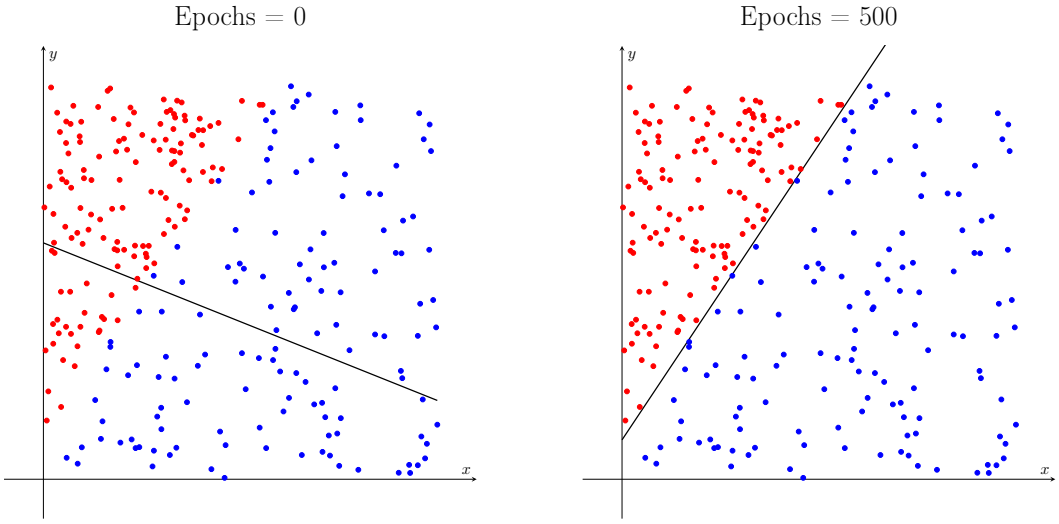

FIGURE 6 – Évolution de la prédiction du perceptron ($n = 2$)

En figure 6 est représenté un échantillon linéairement séparable (points rouge et bleu). La droite représente quant à elle la frontière entre les valeurs positives et négatives renvoyée par le perceptron avant et après entraînement.

4.3 Perceptron multicouches

Le perceptron multicouche (ou *MLP*, pour *Multi-Layer Perceptron*) est une extension du perceptron simple, qui permet de surmonter ses limites en matière de classification linéaire. Alors que le perceptron simple ne peut résoudre que des problèmes linéairement séparables, le perceptron multicouche introduit une ou plusieurs couches cachées de neurones, rendant possible l'apprentissage de relations non linéaires complexes. Cette architecture constitue la base des réseaux de neurones modernes.

Dans la suite, nous adopterons les notations suivantes :

- H désigne le nombre de couches (sans la couche d'entrée).
- Pour tout $h \in [\![0, H]\!], n_h$ désigne le nombre de neurone de la couche h. On appellera architecture du réseau la suite $(n_h)_{1 \leq h \leq H}$.
- Pour tout $h \in [\![1, H]\!], W^{(h)} \in \mathcal{M}_{n_h, n_{h-1}}(\mathbb{R})$ et $b^{(h)} \in \mathbb{R}^{n_h}$ désignent respectivement la matrice contenant les poids des arcs entre la couche h et $h-1$ et le vecteur contenant les biais ajoutés à la couche h.
- Enfin, pour tout $h \in [\![1, H]\!]$, on note σ_h la fonction d'activation appliquée à la couche h.

Un réseau de neurones est représenté en figure 7 pour l'architecture : $(n_0, n_1, n_2, n_3) = (3, 4, 4, 2)$.

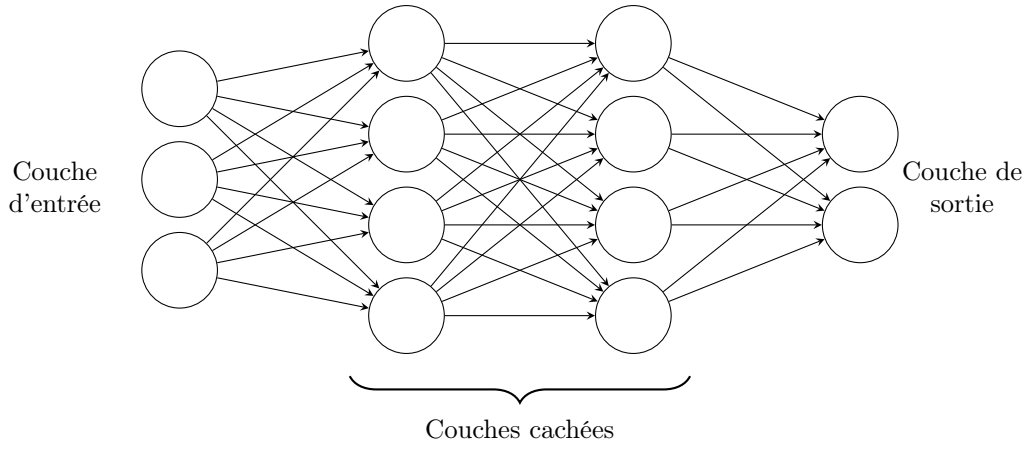

FIGURE 7 – Perceptron multicouche

Remarque : On dit que le réseau est :

- **linéaire** si la fonction d'activation de toute couche est l'identité,
- **totalement connectée** quand aucune matrice du réseau ne contient la valeur zéro.

Dans un perceptron multicouche, chaque neurone, à l'exception de celles de la couche d'entrée, fonctionne comme un perceptron simple. De la même manière, on peut associer une fonction à un réseau de neurones MLP.

Définition : (Fonction de prédiction associée à un réseau de neurone)
On note, pour tout $h \in [\![0, H]\!]$, g_h la fonction définie par récurrence par :

$$\forall x \in \mathbb{R}^{n_0} : g_h(x) = \begin{cases} x & \text{si } h = 0, \\ \sigma_h(W^{(h)} g_{h-1}(x) + b^{(h)}) & \text{si } h \in [\![1, H]\!] \end{cases}$$

La fonction $g_H : \mathbb{R}^{n_0} \longrightarrow \mathbb{R}^{n_H}$ est alors appelée la prédiction, ou l'inférence du réseau.

Les paramètres associés à un réseau sont alors réunis en un vecteur θ composé des coefficients des matrices $W^{(h)}$ et $b^{(h)}$, pour $1 \leq h \leq H$, on note alors $g_H = g_\theta$ (suivant le contexte, nous utiliserons les deux notations dans la suite de ce document). Pour choisir les poids et les biais les plus adaptés, on utilise l'algorithme de rétropropagation du gradient.

4.4 ReLU

La ReLU (Rectified Linear Unit) est une fonction d'activation très utilisée dans les réseaux de neurones. Elle est définie par :

$$\text{ReLU} : \begin{array}{ccc} \mathbb{R} & \longrightarrow & \mathbb{R}_+ \\ x & \longmapsto & \max(0, x). \end{array}$$

Autrement dit, elle renvoie zéro si l'entrée est négative, et l'entrée elle-même si elle est positive. ReLU est populaire car elle est simple, rapide à calculer et simplifie l'étude théorique des réseaux de neurones. Elle introduit aussi de la non-linéarité tout en maintenant une forme très efficace pour l'apprentissage. Dans la suite on dira qu'un réseau de neurones est un réseau ReLU si :

$$\sigma_H = Id \quad \text{et} \quad \forall h \in [\![1, H-1]\!], \sigma_h = \text{ReLU}$$

4.5 Rétropropagation du gradient

L'algorithme de rétropropagation du gradient (ou backpropagation) est une adaptation de l'algorithme de descente de gradient utilisée pour entraîner les réseaux de neurones. Il permet d'ajuster les paramètres (poids et biais) du réseau afin de réduire l'erreur entre la sortie prédite et la sortie attendue, c'est-à-dire

minimiser le risque empirique $\hat{\mathcal{R}}(g_\theta)$ en agissant sur $\theta \in \Theta$.

Voici un résumé rapide de son fonctionnement, pour une entrée (x, y) :

- **Propagation avant** : Les données d'entrée traversent le réseau couche par couche pour effectué le calcul de $g_H(x)$,
- **Calcul de l'erreur** : On compare la sortie du réseau et la sortie attendue, *i.e.* on détermine $C(g_H(x), y)$ où $C : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$ est une fonction de coût.
- **Rétropropagation** : On calcule le gradient de la fonction de coût par rapport aux poids du réseau, en appliquant la règle de la chaîne, *i.e.* on détermine ∇C.
- **Mise à jour des poids** : On se sert du calcul précédent pour mettre à jour les poids et biais du réseau. En notant $\eta > 0$ le taux d'apprentissage, on effectue les opérations :

$$\forall h \in [\![1, H]\!], \begin{cases} W^{(h)} = W^{(h)} - \eta \dfrac{\partial C}{\partial W^{(h)}} \\ \\ b^{(h)} = b^{(h)} - \eta \dfrac{\partial C}{\partial b^{(h)}} \end{cases}$$

Propriété 4.5.1 :
En notant, pour tout $h \in [\![1, H]\!]$, $z_h(x) = W^{(h)} g_{h-1}(x) + b^{(h)}$, on a, pour tout $(x, y) \in \mathcal{X} \times \mathcal{Y}$:

$$\begin{cases} \forall (i,j) \in [\![1, n_h]\!] \times [\![1, n_{h-1}]\!] : \dfrac{\partial C}{\partial W_{i,j}^{(h)}}(\theta) = (g_{h-1}(x))_j (\sigma_h'(z_h(x))_i \Delta_i^{(h)} \\ \\ \forall i \in [\![1, n_h]\!] : \dfrac{\partial C}{\partial b_i^{(h)}}(\theta) = (\sigma_h'(z_H(x))_i \Delta_i^{(h)} \end{cases}$$

Avec, pour tout $h \in [\![1, H-1]\!], i \in [\![1, n_h]\!]$:

$$\Delta_i^{(h)} = \begin{cases} \left(\dfrac{\partial C}{\partial x}(g_\theta(x), y) \right)_i & \text{si } h = H, \\ \displaystyle\sum_{j=1}^{n_{h+1}} \left(W_{j,i}^{(h+1)} \sigma_{(h+1)}'(z_{h+1}(x))_j \Delta_j^{(h+1)} \right) & \text{sinon.} \end{cases}$$

Démonstration : Ayant posé, pour tout $h \in [\![1, H]\!]$, $z_h(x) = W^{(h)} g_{h-1}(x) + b^{(h)}$, on applique la règle de la chaine, qui nous donne :

$$\frac{\partial C}{\partial W_{i,j}^{(h)}}(\theta) = \frac{\partial (z_h(x))_i}{\partial W_{i,j}^{(h)}} \frac{\partial (g_h(x))_i}{\partial (z_h(x))_i} \Delta_i^{(h)}$$

Or :

- $\forall h \in [\![1, H]\!], z_h(x) = W^{(h)} g_{h-1}(x) + b^{(h)} \Longrightarrow \dfrac{\partial (z_h(x))_i}{\partial W_{i,j}^{(h)}} = (g_{h-1}(x))_j$
- $\forall h \in [\![1, H]\!], g_h(x) = \sigma(z_h(x)) \Longrightarrow \dfrac{\partial (g_h(x))_i}{\partial (z_h(x))_i} = \sigma_h'(z_h(x))_j$

De même, par la règle de la chaine, on obtient, pour tout $h \in [\![1, H-1]\!], i \in [\![1, n_h]\!]$:

$$\Delta_i^{(h)} = \sum_{j=1}^{n_{h+1}} \left(\frac{\partial (z_{h+1}(x))_j}{\partial (g_h(x))_i} \frac{\partial (g_{h+1}(x))_j}{\partial (z_{h+1}(x))_j} \Delta_j^{(h+1)} \right)$$

Or :

- $\forall h \in [\![1, H-1]\!], z_{h+1}(x) = W^{(h+1)} g_h(x) + b^{(h+1)} \Longrightarrow \dfrac{\partial (z_{h+1}(x))_j}{\partial (g_h(x))_i} = W_{j,i}^{(h+1)}$

- $\forall h \in [\![1, H-1]\!], g_{h+1}(x) = \sigma(z_{h+1}(x)) \implies \dfrac{\partial(g_{h+1}(x))_i}{\partial(z_{h+1}(x))_i} = \sigma'_h(z_{h+1}(x))_j$

D'où le résultat.

□

4.5.1 Différentiation automatique

L'apprentissage d'un réseau de neurones repose sur l'optimisation d'une fonction de perte. Cela implique de calculer les gradients de cette fonction par rapport à des milliers, voire des millions de paramètres pour des réseaux de neurones profonds. Dans ce cadre, la différentiation symbolique telle qu'effectuée dans la sous-section précédente à beaucoup de désavantage. Tout d'abord, il faut recalculer manuellement à chaque changement de fonction de coût C, le calcul peut de plus devenir fastidieux pour des fonctions complexes. Pire encore, cette méthode échoue à calculer rapidement les dérivées partielles d'une fonction dépendant de nombreuses variables.

De même, la différentiation numérique, ou par méthode des différences finies possède des inconvénients. Bien qu'on ne soit plus contraint de différentier manuellement, cette méthode est coûteuse pour une fonction coûteuse à évaluer, de plus elle introduit des erreurs d'arrondi.

La différentiation automatique est en pratique utilisée pour des modèles complexes. Elle fonctionne en décomposant la fonction en opérations élémentaires, puis en appliquant systématiquement la règle de la chaîne pour propager les dérivées. Elle existe sous deux formes, l'accumulation directe et l'accumulation indirecte. Dans tout les cas, cette méthode permet de calculer rapidement les dérivées partielles d'une fonction, même si elle dépend d'un nombre important de paramètres. (Voir [2] pour plus de détail). Le transformer représentée en figure 8 est un exemple de réseau pour lequel le calcul d'une dérivée partielle est impossible à réaliser manuellement en pratique.

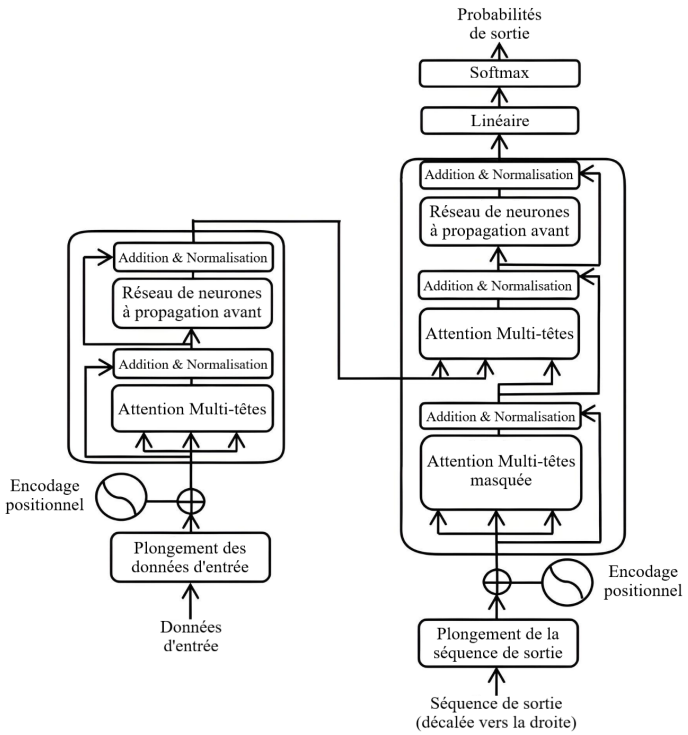

FIGURE 8 – Architecture d'un transformer

5 Régularisation

5.1 Phénomène de régularisation implicite

Dans l'espace des solutions au problème de minimisation du risque empirique, il existe potentiellement plusieurs solutions, à première vue, rien ne semble indiqué lesquelles sont privilégiées. Pourtant, on observe en pratique que la solution vers laquelle tendent les réseaux de neurones durant l'entraînement est une solution "régulière", sans avoir recours à des méthodes de régularisation. Ce phénomène est particulièrement observable dans le cadre de l'approximation de fonctions continues, affines par morceaux.

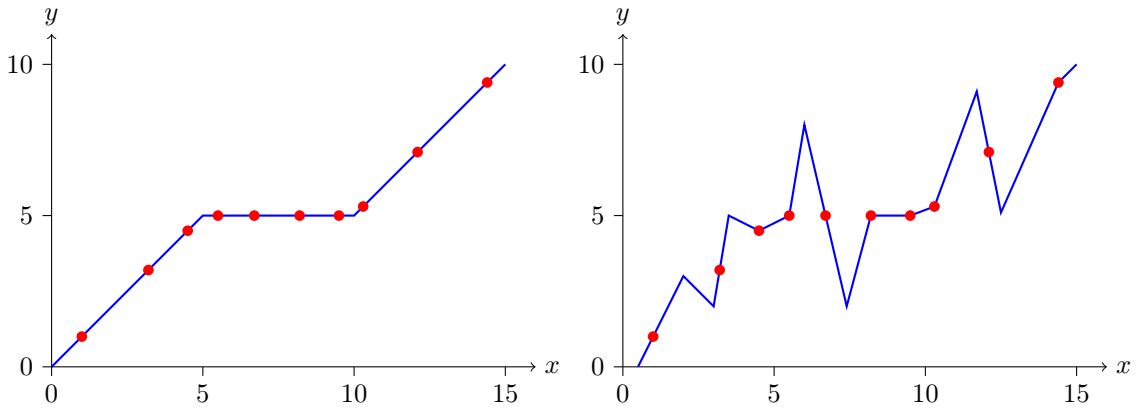

FIGURE 9 – Deux prédictions (en bleu) minisant le risque empirique ($\hat{\mathcal{R}}(g_\theta) = 0$) sur un même échantillon d'apprentissage (en rouge)

La figure 9 illustre deux solutions à un problème de minimisation du risque empirique. La solution de gauche est une courbe avec peu de zones affines, tandis que que la figure de droite correspond à une solution à fortes variations. Une solution ayant un nombre plus faible de zones affines (à minimisation égale du risque empirique) sera considérée comme meilleure qu'une solution à forte variations locale car elle se généralisera mieux sur des données non-observée. En pratique, ce sont de telles solutions qui sont fournies par l'entraînement des réseaux de neurones (descente de gradient et ses variantes).

5.2 Exemple

Cette sous-section illustre le phénomène de régularisation implicite dans le cas d'un réseau d'architecture $(1, 50, 50, 1)$. Le réseau est entraîné sur un échantillon $(x_i, y_i)_{i\in[\![1,80]\!]} \in ([0, 20] \times [0, 20])^{80}$ suivant la loi du couple de variable aléatoire $(X, f(X))$, avec $X \sim U([0, 20])$, et f la fonction cible représente en figure 9.

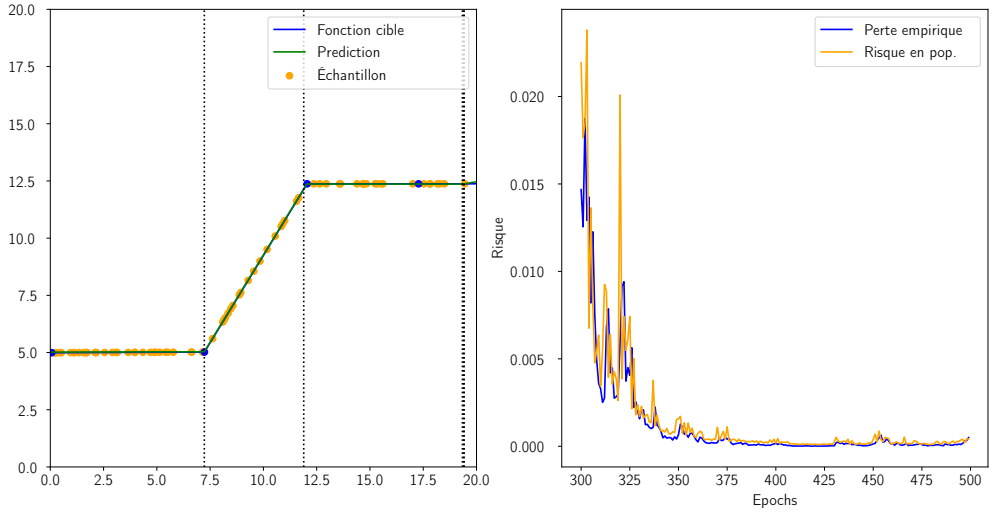

FIGURE 10 – Entraînement avec taux d'apprentissage 0.05 sur 500 epochs

On observe sur la figure 10 que la fonction de prédiction coïncide avec la fonction cible : le risque empirique est nul et le risque en population est très faible. Ceci révèle un phénomène de régularisation implicite pour les réseaux ReLU.

Remarque : Les droites verticales de la figure 10 représentent les changement de motif d'activation, détaillés en section 6.2.

5.2.1 Comportement imprévisible

Le phénomène de régularisation implicite n'a lieu que sur des zones affines qui sont couvertes par l'échantillon de données. En l'absence de données, notamment au niveau des changements de zones affines de la fonction à approximer, le comportement du réseaux semble alors imprévisible (voir figure 11).

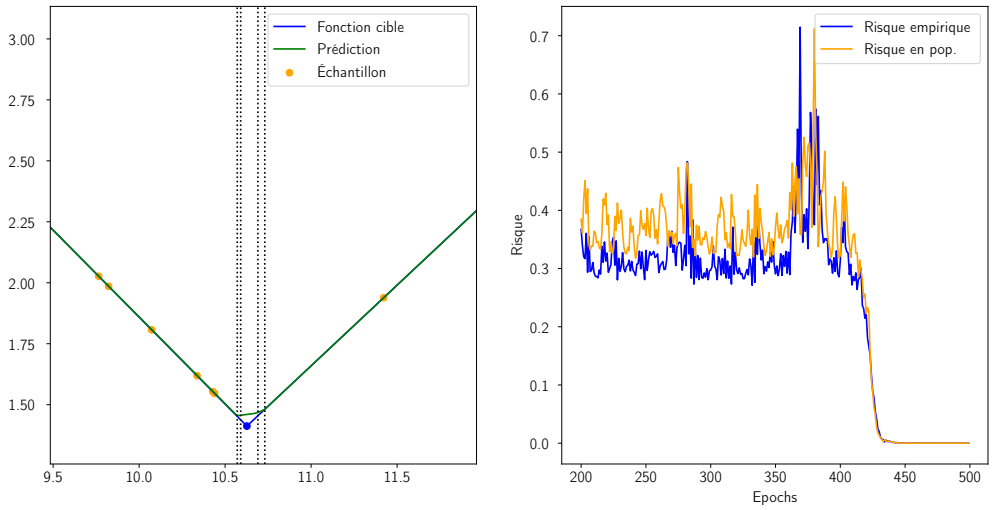

FIGURE 11 – Comportement imprévisible sur un changement de zone affine peu couvert par les données

On constate, en figure 11, que lors de l'entraînement, les changements de zones d'activation s'accumulent au niveau des changements de zones affines de la fonction cible. Ceci favorise l'apparition de

nombreuses petites zones affines indésirables.

5.3 Méthodes de régularisation

Pour forcer un réseau de neurones, ou, de manière plus générale, un algorithme d'apprentissage à fournir des résultats convenables, la régularisation implicite que le modèle offre n'est pas toujours suffisante. On applique alors des méthodes de régularisation comme celles énoncées ci-après.

5.3.1 *Early stopping*

Comme expliqué précédemment, entraîner le modèle trop longtemps sur le même échantillon de données peut le mener à capturer les particularités des données d'entraînement, qui ne sont pas représentatives des données générales. Pour éviter cela, on arrête l'entraînement dès que le risque empirique augmente.

5.3.2 *Dropout*

Le *dropout* est une technique de régularisation utilisée dans l'entraînement des réseaux de neurones pour réduire le sur-apprentissage.
Lors de chaque itération d'entraînement, le *dropout* consiste à désactiver aléatoirement (c'est-à-dire assigner temporairement la valeur zéro) un certain nombre de neurones d'une couche donnée. Cela signifie que, pour chaque descente de gradient dans le réseau, une sous-architecture différente du réseau est utilisée. Cette procédure empêche le réseau de devenir trop dépendant de certains neurones ou de certaines adaptations trop spécifiques. Le *dropout* force le réseau à apprendre des représentations plus robustes et plus réparties, ce qui améliore sa capacité à généraliser sur de nouvelles données.

6 Réseaux ReLU

Dans cette section, nous détaillons l'étude de l'un des types de réseau de neurones les plus étudiés : les réseaux ReLU.

6.1 Non-coercivité, non-convexité

6.1.1 Non-coercivité

Comme vu dans la section 2.2, on peut parfois obtenir des garanties sur un problème d'optimisation grâce aux propriété des fonctions étudiées. Malheureusement, pour les réseaux de neurones, nous ne disposons jamais de la coercivité du risque empirique, ni de la convexité de manière générale.

Propriété 6.1.1 :
Pour toute architecture $(n_1, \cdots, n_H)$ et pour tout échantillon $(x_i, y_i)_{i\in[\![1,n]\!]} \in (\mathbb{R}^{n_0} \times \mathbb{R}^{n_H})^n$, $\theta \longmapsto \hat{\mathcal{R}}(\theta)$ n'est pas coercive.

Démonstration : Soit $\theta = (W^{(1)}, \cdots, W^{(H-1)}, W^{(H)}) \in \Theta\backslash\{0\}$ tel que $b^{(h)} = 0$ pour tout $h \in [\![1, H]\!]$. On note, pour $\lambda > 0$ $\theta^\lambda = (\lambda^{-1}W^{(1)}, \cdots, \lambda^{-1}W^{(H-1)}, \lambda^{H-1}W^{(H)}) \in \Theta$.

Or par homogénéité de ReLU, on a :

$$\begin{aligned} g_{\theta^\lambda}(x) &= \lambda^{H-1}W^{(H)} \cdot \mathrm{ReLU}(\lambda^{-1}W^{(H-1)}\mathrm{ReLU}(\cdots \lambda^{-1}W^{(1)}x)) \\ &= W^{(H)} \cdot \mathrm{ReLU}(W^{(H-1)}\mathrm{ReLU}(\cdots W^{(1)}x)) \\ &= g_\theta(x) \end{aligned}$$

On a ainsi, pour tout $\lambda > 0$: $\hat{\mathcal{R}}(g_\theta) = \hat{\mathcal{R}}(g_{\theta^\lambda})$, mais $\lim\limits_{\lambda\longrightarrow+\infty} \|\theta^\lambda\| = +\infty$ Ce qui contredit la coercivité du risque empirique.

□

Remarque : On ne peut donc pas appliquer le théorème 3.3.1, de manière générale, le problème n'admet pas toujours de solution pour les réseaux ReLU.

6.1.2 Exemple non-convexe

De la même manière, en général, on ne dispose pas de la convexité du risque empirique pour les réseaux ReLU. Il suffit, pour s'en rendre de compte, de considérer le réseau d'architecture $(1, 1, 1)$ sur l'exemple $(x, y) = (1, 0)$ et la fonction de perte $l(x, y) = (x - y)^2$.

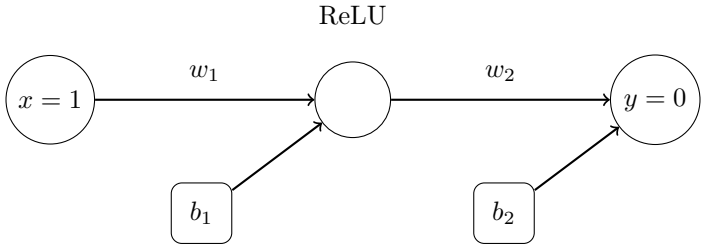

FIGURE 12 – Réseau $(1, 1, 1)$ avec poids et biais entourés

On peut calculer le risque empirique (ici pour $w_1 > 0$ et $b_1 = b_2 = 0$) :

$$\begin{aligned} \hat{\mathcal{R}}(\theta) &= \hat{\mathcal{R}}(w_1, w_2, b_1, b_2) \\ &= l(w_2\mathrm{ReLU}(w_1x + b_1) + b_2, y) \\ &= (w_2\mathrm{ReLU}(w_1x + b_1) + b_2 - y)^2 \\ &= (w_1w_2)^2 \end{aligned}$$

Or, la restriction de $f : (x, y) \longmapsto (xy)^2$ à $\mathbb{R}_+ \times \mathbb{R}$ n'est pas convexe (voir figure 13).

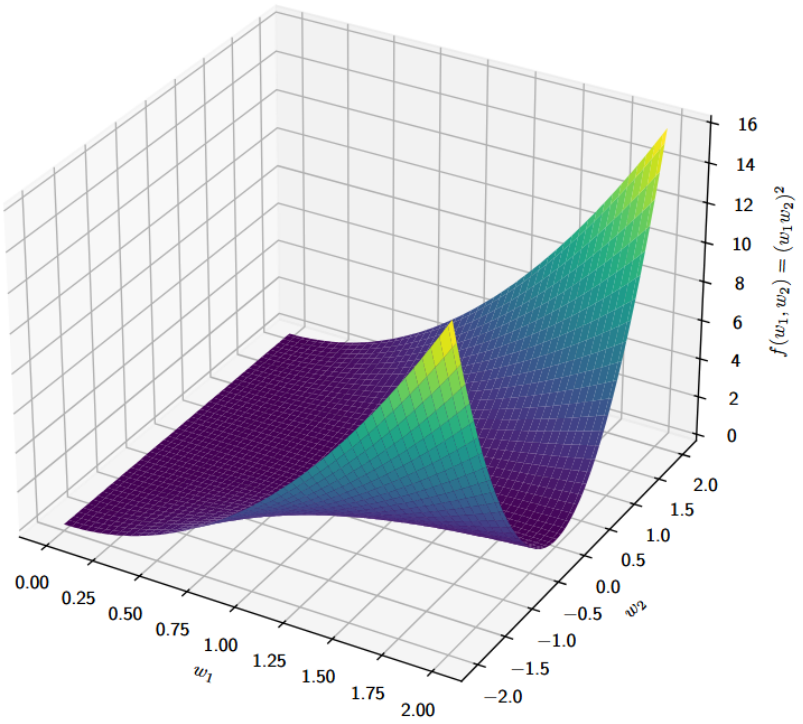

FIGURE 13 – Risque empirique non-convexe

6.2 Motif d'activation

Dans la suite on note, pour tout $\theta \in \Theta$, $h \in [\![1, H-1]\!]$, $x \in \mathbb{R}^{n_0}$, $a_h(x, \theta) \in \{0,1\}^{n_h}$ le vecteur défini par :

$$\forall i \in [\![1, n_h]\!], (a_h(x,\theta))_i = \begin{cases} 1 & \text{si } (W^{(h)} g_{h-1}(x) + b^{(h)})_i > 0 \\ 0 & \text{sinon.} \end{cases}$$

Définition : (Motif d'activation)
On appelle motif d'activation du réseau la fonction :

$$a(x, \cdot) : \begin{array}{ccc} \Theta & \longrightarrow & \{0,1\}^{n_1} \times \cdots \times \{0,1\}^{n_{H-1}} \\ \theta & \longmapsto & (a_h(x,\theta))_{1 \le h \le H}. \end{array}$$

Le motif d'activation peut être vu comme des booléens indiquant quels neurones sont activés pour une entrée x fixée.

Remarque : Avec cette notation, on a, pour un réseau ReLU :

$$\forall x \in \mathbb{R}^{n_0} : g_h(x) = \begin{cases} x & \text{si } h = 0, \\ \operatorname{diag}(a_h(x,\theta))\,(W^{(h)} g_{h-1}(x) + b^{(h)}) & \text{pour } \ h \in [\![1, H]\!]. \end{cases}$$

Définition : (Ensembles $D_\delta(\theta)$, $A_\delta(x)$)
Soit $\theta \in \Theta$ fixé, et soit $\delta \in \{0,1\}^{n_1} \times \cdots \times \{0,1\}^{n_H - 1}$ un motif d'activation, on note :

$$D_\delta(\theta) := \{x \in \mathbb{R}^{n_0} \mid a(x,\theta) = \delta\}$$

De la même manière, pour $x \in \mathbb{R}^{n_0}$ fixé, on définit :

$$A_\delta(x) := \{\theta \in \Theta \mid a(x,\theta) = \delta\}$$

Définition : (Activations atteignables)
On désigne, pour $\theta \in \Theta$, l'ensemble des motifs d'activations atteignables par :

$$A(\theta) = \{\delta \in \{0,1\}^{n_1} \times \cdots \times \{0,1\}^{n_{H-1}} \mid \text{Int}(D_\delta(\theta)) \neq \emptyset\}$$

Remarque : On a évidemment :

$$\mathbb{R}^{n_0} = \bigcup_{\delta \in A(\theta)} D_\delta(\theta)$$

6.3 Propriétés de g_H

Définition : (Fonction affine, affine par morceaux)
Soit $f : \mathbb{R}^m \longrightarrow \mathbb{R}^n$.

- On dit que f est affine ou linéaire s'il existe $A \in \mathcal{M}_{n,m}(\mathbb{R})$ et $b \in \mathbb{R}^n$ tels que : $\forall x \in \mathbb{R}^m, f(x) = Ax + b$
- On dit que f est affine par morceaux s'il existe une partition du domaine de définition de f en polyhèdres sur lesquels f est affine.

Propriété 6.3.1 :
Si $f : \mathbb{R}^l \longrightarrow \mathbb{R}^m$ et $g : \mathbb{R}^m \longrightarrow \mathbb{R}^n$ sont deux fonctions continues affines par morceaux, alors $f \circ g$ est continue affine par morceau.

Démonstration : Voir annexe 7.4

Propriété 6.3.2 :
Pour un réseau ReLU, avec $\theta \in \Theta$ fixé, et $\delta \in \{0,1\}^{n_1} \times \cdots \times \{0,1\}^{n_{H-1}}$ un motif d'activation, les fonctions $x \longmapsto g_h(x)$ sont continues et affines par morceaux.

Démonstration : On note, pour tout $h \in [\![1, H]\!]$, l_h la fonction :

$$l_h : \begin{array}{ccc} \mathbb{R}^{n_{h-1}} & \longrightarrow & \mathbb{R}^{n_h} \\ x & \longmapsto & \begin{cases} W^{(H)}x + b^{(H)} & \text{si } h = H, \\ \text{ReLU}(W^{(h)}x + b^{(h)}) & \text{sinon.} \end{cases} \end{array}$$

De telle sorte que, pour tout $h \in [\![1, H]\!] : \forall x \in \mathbb{R}^{n_0}, g_h(x) = (l_h \circ l_{h-1} \circ \cdots \circ l_1)(x)$.
Les fonctions l_h sont continues affines par morceau car ReLU est continue affine par morceau. De plus, par la propriété 6.3.1, la composition de deux fonctions continues affines par morceau est continue affine par morceaux. Par une récurrence finie immédiate, pour tout $h \in [\![1, H]\!]$, g_h est continue affine par morceaux.

□

Propriété 6.3.3 :
Les ensembles $D_\delta(\theta)$ sont des polyèdres, et, pour tout h la restriction de g_h à $D_\delta(\theta)$ est affine pour tout motif d'activation δ.

Démonstration : Soit $\delta \in \{0,1\}^{n_1} \times \cdots \times \{0,1\}^{n_H}$ un motif d'activation tel que $D_\delta(\theta) \neq \emptyset$. En reprenant les notations de la démonstration précédente, on a, pour tout $x \in D_\delta(\theta)$ et pour tout $h \in [\![1, H]\!]$:

$$l_h(x) = \begin{cases} W^{(H)}x + b^{(H)} & \text{si } h = H, \\ A^{(h)}(W^{(h)}x + b^{(h)}) & \text{sinon.} \end{cases}$$

Avec, pour tout $h \in [\![1, H]\!]$, $A^{(h)} = \text{diag}(a_h(x, \theta))$. Or la quantité $(a_h(x, \theta))_i$ est constante sur $D_\delta(\theta)$. Donc, la restriction de l_h à $D_\delta(\theta)$ est affine pour tout h et, par composition, la restriction de g_h à $D_\delta(\theta)$ est affine pour tout h.

Montrons maintenant que cet ensemble est un polyèdre. On a, en notant $A^{(h)} = \text{diag}(\delta_h)$:

$$\begin{aligned} x \in D_\delta(\theta) &\iff a(x,\theta) = \delta \\ &\iff \forall h \in [\![1, H]\!],\ a_h(x,\theta) = \delta_h \\ &\iff \forall h \in [\![1, H]\!],\ A^{(h)}(W^{(h)} g_{h-1}(x) + b^{(h)}) \geq 0 \end{aligned}$$

De plus, on vient de démontrer que la restriction de g_h est affine sur $D_\delta(\theta)$, on a donc un ensemble fini d'inégalité linéaire, ce qui définit bien un polyèdre.

□

Remarque : Comme $\mathbb{R}^{n_0} = \bigcup_{\delta \in A(\theta)} D_\delta(\theta)$, cela démontre une nouvelle fois que g_θ est affines par morceaux.

6.3.1 Exemple

Dans cette partie nous illustrons la proposition 6.3.3 à l'aide d'un réseau de neurone entraîné pour effectué une classification binaire. Dans la partie gauche de la figure 14, les segment noirs partitionnent l'espace en deux régions. On se donne alors un réseau de neurone d'architecture $(2, 3, 1)$, et on considère qu'un point $X = (x, y) \in \mathbb{R}^2$ et dans la classe bleue si $g_\theta(X) \leq 1$, sinon, il est dans la classe rouge.

Les droites vertes représentent les changements d'activation δ et donc les changements de zones affines $D_\delta(\theta)$. À gauche, on observe que $f(x, y)$ est bien affine sur les ensemble $D_\delta(\theta)$.

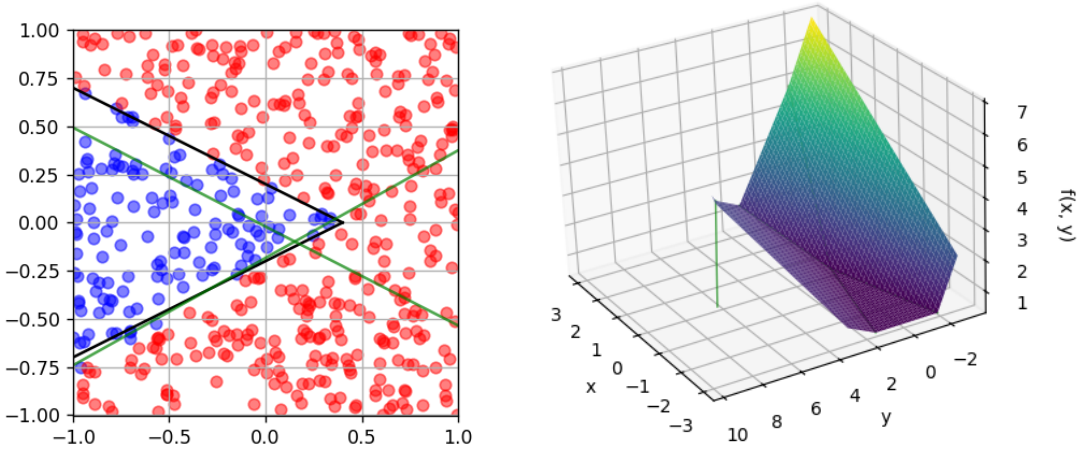

FIGURE 14 – Représentation des zones affines $D_\delta(\theta)$ pour un réseau entraîné à la classification binaire

Remarque : Le choix d'un réseau de neurone à peu de paramètres était utile ici pour représenter un petit nombre de changements d'activation. Cependant, en pratique, les réseaux de neurones ont beaucoup plus de paramètres, et lorsqu'on cherche à effectuer la même expérience à l'aide d'un tel réseau (comme ce qui est effectué en figure 10 pour $n_0 = 1$), on peut observer une accumulation des changements de motif d'activation au niveau des raccordements des zones affines ainsi que dans les zones peu couvertes par les données d'entraînement.

6.4 Géométrie des réseaux ReLU

Dans toute cette sous section, on fixe $X \in \mathbb{R}^{n_0 \times n}$, un échantillon de n entrées. Sachant que le motif d'activation du réseau est à image dans un ensemble fini, notons $A_1, \cdots, A_m$ les valeurs prises par $a(X, \cdot)$.

Définition : $(\mathcal{U}_i^X,\ \tilde{\mathcal{U}}_i^X)$
On définit $\tilde{\mathcal{U}}_i^X$, pour tout $i \in [\![1, m]\!]$, par :

$$\tilde{\mathcal{U}}_i^X := \mathrm{Int}\left\{\theta \in \Theta \mid a(x, \theta) = A_i\right\}$$

On définit $\mathcal{U}_i^X$, pour tout $i \in [\![1, m]\!]$, par :

$$\mathcal{U}_i^X := \left\{ \theta \in \tilde{\mathcal{U}}_i^X \;\middle|\; \mathrm{rank}(\mathrm{d}g_\theta(X)) = \max_{\theta \,\in\, \tilde{\mathcal{U}}_i^X} \left(\mathrm{rank}(\mathrm{d}g_\theta(X))\right) \right\}$$

Quitte à réindexer, on supposera dans la suite que les $\mathcal{U}_i^X$ sont tous non-nuls.

Remarque : Par définition des ensembles $\mathcal{U}_i^X$, on a :

- Les ensembles $\mathcal{U}_i^X$ sont non-vide et disjoints deux à deux.
- Le motif d'activation est constant sur les $\mathcal{U}_i^X$, il prend en particulier m valeurs distinctes sur l'union de ces ensembles.
- $\theta \longmapsto \mathrm{rank}(\mathrm{d}g_\theta(X))$ est constante sur les $\mathcal{U}_i^X$.

Définition : (Ensembles $\mathcal{T}_{h,i}^X, \mathcal{T}^X$)
Soit $x \in \mathbb{R}^{n_0 \times n}$, pour tout $h \in [\![1, H]\!]$, pour tout $i \in [\![1, n_h]\!]$:

$$\mathcal{T}_{h,i}^x = \left\{ \theta \in \Theta \;\middle|\; \sum_{k=1}^{n_{h-1}} \left(W_{i,k}^{(h)} (g_{h-1}(x))_k + b_i^{(h)} \right) = 0 \right\}$$

On définit aussi, pour $X \in \mathbb{R}^{n_0 \times n}$: $\mathcal{T}_{h,i}^X$ On pose de plus :

$$\mathcal{T}^x = \bigcup_{h=1}^{H-1} \bigcup_{i=1}^{n_h} \mathcal{T}_{h,i}^x$$

Lemme 6.4.1 :
Soit $x \in \mathbb{R}^{n_0}$ fixé, alors :

- La fonction : $\begin{array}{ccc} \Theta & \longrightarrow & \{0,1\}^{n_1} \times \cdots \times \{0,1\}^{n_{H-1}} \\ \theta & \longmapsto & a(x, \theta) \end{array}$ est surjective.
- Pour tout schéma d'activation $\delta \in \{0,1\}^{n_1} \times \cdots \times \{0,1\}^{n_{H-1}}$, $\theta \longmapsto f_\theta(x)$ est analytique sur $A_\delta(x)$.
- La mesure de Lebesgue de $\mathcal{T}^x$ est nulle et :

$$\mathcal{T}^x = \bigcup_{\delta \in \{0,1\}^{n_1} \times \cdots \times \{0,1\}^{n_{H-1}}} \partial A_\delta(x)$$

Lemme 6.4.2 :
Pour tout $n \in \mathbb{N}^*$, pour tout $X \in \mathbb{R}^{n_0}$, les ensembles $\tilde{\mathcal{U}}_1^X, \cdots, \tilde{\mathcal{U}}_m^X$ sont non vides, ouverts et disjoints, et ils vérifient les propriétés suivantes :

- $\left(\bigcup_{i=1}^{m} \tilde{\mathcal{U}}_i^X\right)^c = \mathcal{T}^X$, et en particulier, cet ensemble est fermé de mesure de Lebesgue nulle.
- Pour tout $j \in 1, \ldots, p_X$, la fonction $\theta \mapsto a(X, \theta)$ est constante sur chaque $\tilde{\mathcal{U}}_i^X$ et prend m valeurs distinctes sur $\bigcup_{j=1}^{m} \tilde{\mathcal{U}}_i^X$;
- Pour tout $i \in [\![1, m]\!]$, la fonction $\theta \mapsto f_\theta(X)$ est analytique sur $\tilde{\mathcal{U}}_i^X$.

Théorème 6.4.1 : (J. Bona-Pellissier, F. Bachoc, F. Malgouyres 2024) [3]

- Les ensembles $\mathcal{U}_i^X$ sont ouverts.
- L'ensemble fermé $\left(\bigcup_{i=1}^{m} \mathcal{U}_i^X\right)^c$ est de mesure de Lebesgue nulle.
- $\theta \longmapsto g_\theta(x)$ est polynomiale sur les $\mathcal{U}_i^X$.

Démonstration : Voir [3], page 33 à 38.

6.4.1 Exemple

Dans cette section, nous illustrons les propriétés des ensembles $\mathcal{U}_i^X$ dans le cas d'un réseau N d'architecture $(1, 1, 1)$. (Voir figure 13) et des entrées $X = (0, 1, 2)$, et $\theta = (w_1, b_1, w_2, b_2)$. On a :

$$f_\theta(X) = \begin{pmatrix} w_2 \mathrm{ReLU}(b_1) + b_2 \\ w_2 \mathrm{ReLU}(w_1 + b_1) + b_2 \\ w_2 \mathrm{ReLU}(2w_1 + b_1) + b_2 \end{pmatrix}^\top$$

Ainsi, on peut déterminer explicitement les ensembles $\mathcal{U}_i^X$. Les activations possibles sont $\Delta \in \{0, 1\}^3$, seules 6 sont atteintes dans l'exemple. Dans la figure 14, on effectue l'entraînement du réseau N pour des paramétrisations choisies aléatoirement suivant une loi normale $\mathcal{N}(0, 2)$. Le réseau N est entraîné sur les entrées $X = (0, 1, 2)$ et les sorties $Y = (1.5, 1.1, 0.4)$ (qui est indiqué par un point bleu). La coloration des points correspond au risque calculé en ces points.

Posons $\mathcal{P}$ le plan d'équation $x + y + z = 0$, plan orthogonal à $(1, 1, 1)$. On fait ce choix pour mieux visualiser les phénomènes de rang et car les ensembles $f_{\tilde{\mathcal{U}}_i^X}(X)$ sont invariants par translation selon $\lambda \in \mathrm{Vect}((1, 1, 1))$. On note $p_{\mathcal{P}}^{\perp}$ la projection orthogonale sur $\mathcal{P}$ parallèlement à $\mathrm{Vect}((1, 1, 1))$ et $\tilde{\Theta} \subset \Theta$ les choix aléatoires de paramètres du réseau. Alors, les points verts correspondent à $p_{\mathcal{P}}^{\perp}(f_{\tilde{\Theta}}(X))$.

Les droites rouges du graphique de droite correspondent aux frontières des ensembles $\tilde{\mathcal{U}}_i^X$. Les droites rouges du graphique de gauche correspond aux ensembles $\mathcal{P} \cap (f_{\tilde{\mathcal{U}}_i^X}(X))$ pour $i = 6$ (droite $y = 0$), $i = 5$ (droite $y = -\sqrt{3}x$) et $i = 4$ (droite $y = -1/\sqrt{3}x$). La zone hachurée en rouge correspond à $\mathcal{P} \cap (f_{\tilde{\mathcal{U}}_3^X}(X))$ La zone hachurées en bleu correspond à $\mathcal{P} \cap (f_{\tilde{\mathcal{U}}_5^X}(X))$ Enfin à $\mathcal{P} \cap (f_{\tilde{\mathcal{U}}_1^X}(X)) = 0$

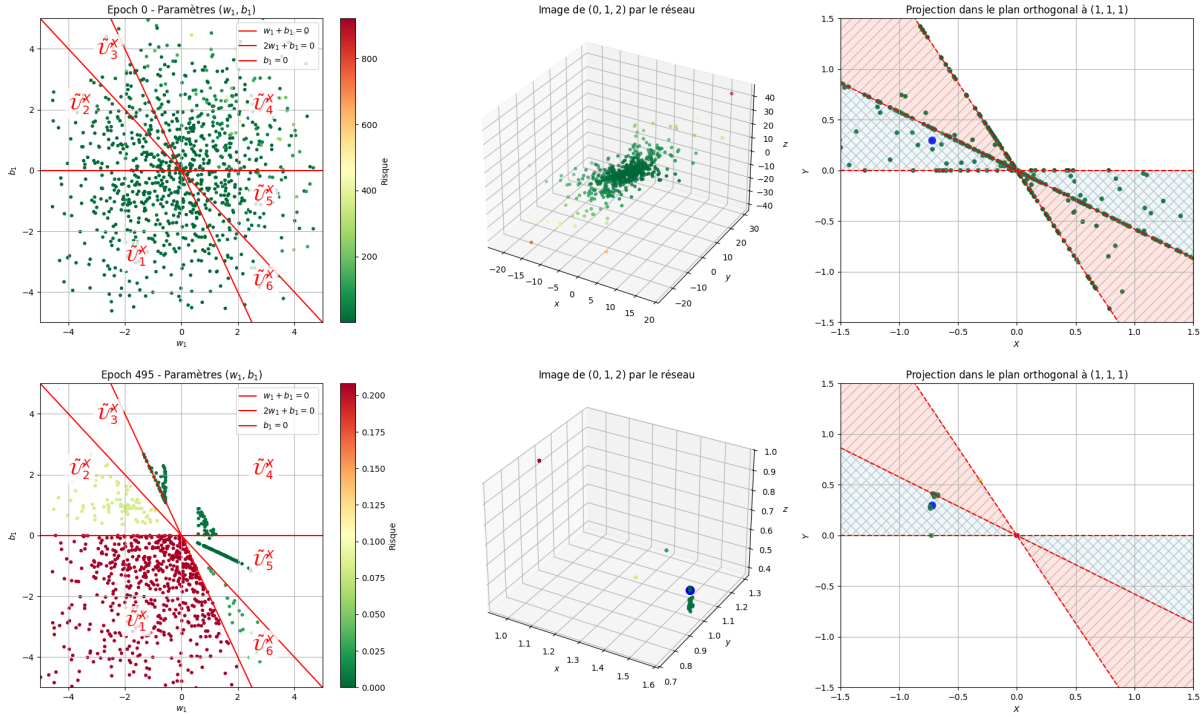

FIGURE 15 – Ensembles $\tilde{\mathcal{U}}_i^X$ et résultat de l'entraînement pour différents paramètres initiaux

On remarque dans un premier temps que les réseaux dont le paramétrage initial est effectué en $\tilde{\mathcal{U}}_1^X$ ont conservé les mêmes valeurs (w_1, b_1), en effet, pour tout les exemples fournit dans X, on a :

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial b_1} = 0$$

Donc leur valeur reste inchangée. De manière générale :

$$\frac{\partial}{\partial w_1} L(f_\theta(X), Y) = 2(f_\theta(X) - Y) \frac{\partial}{\partial w_1} f_\theta(X)$$

Avec :

$$\frac{\partial}{\partial w_1} f_\theta(X) = \begin{pmatrix} 0 \\ w_2 H(w_1 + b_1) \\ 2w_2 H(2w_1 + b_1) \end{pmatrix} \quad \text{et} \quad \frac{\partial}{\partial b_1} f_\theta(X) = \begin{pmatrix} w_2 H(b_1) \\ w_2 H(w_1 + b_1) \\ w_2 H(w_2 + b_1) \end{pmatrix}$$

7 Annexes

7.1 Expression matricielle du gradient d'un réseau ReLU

De même, il est possible d'exprimer plus simplement ΔC :

Propriété 7.1.1 :
Pour un réseau ReLU, ayant pour fonction de coût $C : (X, Y) \longmapsto \|X - Y\|^2$ les mêmes notations, on a, pour tout $(x, y) \in \mathcal{X} \times \mathcal{Y}$:

$$\forall h \in H, \begin{cases} \dfrac{\partial C}{\partial W^{(h)}}(\theta) = g_{h-1}(x) \cdot \left(a_h(x, \theta) \odot \Delta^{(h)}\right)^\top \\ \\ \dfrac{\partial C}{\partial b^{(h)}}(\theta) = a_h(x, \theta) \odot \Delta_i^{(h)} \end{cases}$$

Avec :

$$\Delta^{(h)} = \begin{cases} 2(x - y) & \text{si } h = H, \\ \Delta^{(h)} = W^{(h+1)} \cdot \left(a_{(h+1)}(x, \theta) \odot \Delta^{(h+1)}\right) & \text{sinon.} \end{cases}$$

7.2 Polynôme minimiseur du risque empirique

Démonstration : On cherche $\tilde{\theta} \in \Theta$ tel que :

$$\|X\tilde{\theta} - y\|^2 = \min_{\theta \in \mathbb{R}^{d+1}} \|X\theta - y\|^2$$

Pour X de rang plein. Notons $L : \theta \longmapsto \|X\theta - y\|^2$. Alors, pour tout $\theta \in \Theta$:

$$L(\theta) = \theta^\top X^\top X\theta - 2y^\top X\theta + y^\top y$$

La fonction L est $\mathcal{C}^\infty$ et son gradient est donné par : $\nabla L(\theta) = 2X^\top X\theta - 2X^\top y$ Ainsi :

$$\begin{aligned} \nabla L(\theta) = 0 &\implies 2X^\top X\theta - 2X^\top y = 0 \\ &\implies X^\top X\theta = X^\top y \\ &\implies \theta = (X^\top X)^{-1} X^\top y \end{aligned}$$

□

7.3 Taylor-Lagrange avec gradient lipschitz

Démonstration : Notons $g : t \in [0, 1] \mapsto f((1 - t)x + ty)$. C'est une fonction dérivable, et on a :

$$g'(t) = \langle \nabla f((1 - t)x + ty), y - x \rangle.$$

Ainsi,

$$f(y) = g(1) = g(0) + \int_0^1 g'(t)\,\mathrm{d}t = f(x) + \int_0^1 \langle \nabla f((1 - t)x + ty), y - x \rangle\,\mathrm{d}t.$$

Pour tout $t \in [0, 1]$, par hypothèse sur la L-lipschitzianité du gradient :

$$\|\nabla f((1 - t)x + ty) - \nabla f(x)\| \leq Lt\|x - y\|,$$

ce qui implique :

$$\langle \nabla f((1 - t)x + ty), y - x \rangle \leq \langle \nabla f(x), y - x \rangle + Lt\|x - y\|^2.$$

En intégrant, on obtient :

$$f(y) \leq f(x) + \int_0^1 \left(\langle \nabla f(x), y - x \rangle + Lt\|x - y\|^2\right) dt = f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|x - y\|^2.$$

□

7.4 Composition de fonction continue et linéaire par morceaux

Démonstration : Par définition, il existe deux famille de polyèdre fermés de $\mathbb{R}^l$ (notés $C_1, \ldots, C_r$) et $\mathbb{R}^m$ (notés $D_1, \ldots, D_s$) telles que $\bigcup_{i=1}^{r} C_i = \mathbb{R}^l$, $\bigcup_{j=1}^{s} D_j = \mathbb{R}^m$, f est linéaire sur chaque C_i et g est linéaire sur chaque D_j.

Soient $i \in 1, \ldots, r$ et $j \in 1, \ldots, s$. La fonction f coïncide avec une application linéaire $\tilde{h} : \mathbb{R}^l \longrightarrow \mathbb{R}^m$ sur C_i, et l'image réciproque d'un polyèdre fermé par une application linéaire est un polyèdre fermé donc $\tilde{h}^{-1}(D_j)$ est un polyèdre fermé. Ainsi, l'ensemble $h^{-1}(D_j) \cap C_i = \tilde{h}^{-1}(D_j) \cap C_i$ est un polyèdre fermé comme intersection de deux polyèdres fermés.

La fonction h est linéaire sur C_i, et g est linéaire sur D_j, donc $g \circ h$ est linéaire sur $h^{-1}(D_j) \cap C_i$.On a donc une famille de polyèdres fermés sur lesquels $f \circ g$ est linéaire :

$$(h^{-1}(D_j) \cap C_i)_{\substack{1 \leq i \leq r \\ 1 \leq j \leq s}}$$

On peut conclure que $g \circ h$ est une fonction continue par morceaux linéaire, car l'union de ces polyèdres est $\mathbb{R}^l$.

□

7.5 Descente de gradient discrète

7.5.1 Propriété 3.3.1

Démonstration : Soit k tel que $\nabla f(x_k) \neq 0$. D'après l'inégalité de Taylor-Lagrange avec gradient lipschitzien :

$$\forall x, y \in E, f(y) \leq f(x) + \nabla f(x)^\top (y - x) + \frac{L}{2}\|y - x\|^2$$

En l'appliquant à $x = x_k$ et $y = x_{k+1}$, il vient :

$$\begin{array}{rcl} f(x_{k+1}) & \leq & f(x_k) - \nabla f(x_k)^\top (\eta_k \nabla f(x_k)) + \frac{L}{2}\|\eta_k \nabla f(x_k)\|^2 \\ & \leq & f(x_k) - \eta_k \|\nabla f(x_k)\|^2 + \frac{L\eta_k^2}{2}\|\nabla f(x_k)\|^2 \\ & \leq & f(x_k) - \eta_k \left(1 - \frac{L\eta_k}{2}\right) \|\nabla f(x_k)\|^2 \\ & \leq & f(x_k) \end{array}$$

□

7.5.2 Propriété 3.3.2

Démonstration : Soit $k \in \mathbb{N}$:

$$\begin{array}{rcl} \|x_{k+1} - x^*\|^2 & = & \left\langle x_k - \frac{1}{L}\nabla f(x_k) - x^* \middle| x_k - \frac{1}{L}\nabla f(x_k) - x^* \right\rangle \\ & = & \|x_k - x^*\|^2 - \frac{2}{L}\langle \nabla f(x_k) | x_k - x^* \rangle + \frac{1}{L^2}\|\nabla f(x_k)\|^2 \end{array}$$

Or, on a, par inégalité de Taylor-Lagrange à gradient convexe :

$$f(x_{k+1}) \leq f(x_k) - \frac{1}{2L}\|\nabla f(x)\|^2 \Longrightarrow \|\nabla f(x)\|^2 \leq 2L(f(x_k) - f(x_{k+1}))$$

Et, par convexité :

$$f(x) + \langle \nabla f(x) | y - x \rangle \leq f(y) \Longrightarrow \langle \nabla f(x_k) | x^* - x_k \rangle \leq f(x^*) - f(x_k)$$

Ainsi :

$$\begin{array}{rcl} \|x_{k+1} - x^*\|^2 & \leq & \|x_k - x^*\|^2 - \frac{2}{L}(f(x_k) - \min f) + \frac{2}{L}(f(x_k) - f(x_{k+1})) \\ & \leq & \|x_k - x^*\|^2 \end{array}$$

Il en résulte que la suite $(\|x_k - x^*\|^2)_{k\in\mathbb{N}}$ est décroissante. De plus :

$$\begin{aligned} & f(x_n) + \langle \nabla f(x_n), x^* - x_n \rangle \leq f(x^*) = \min f \\ \implies \quad & f(x_n) - \min f \leq \langle \nabla f(x_n), x_n - x^* \rangle \leq \|\nabla f(x_n)\| \cdot \|x_n - x^*\|, \\ \implies \quad & \frac{f(x_n) - \min f}{\|x_n - x^*\|} \leq \|\nabla f(x_n)\|. \end{aligned}$$

En combinant avec l'inégalité précédente :

$$f(x_{n+1}) - \min f \leq f(x_n) - \min f - \frac{1}{2L} \cdot \frac{(f(x_n) - \min f)^2}{\|x_n - x^*\|^2} \leq f(x_n) - \min f - \frac{1}{2L} \cdot \frac{(f(x_n) - \min f)^2}{\|x_0 - x^*\|^2}$$

donc

$$\frac{1}{f(x_{n+1}) - \min f} \geq \frac{1}{f(x_n) - \min f} \cdot \frac{1}{1 - \dfrac{f(x_n) - \min f}{2L\|x_0 - x^*\|^2}}.$$

On utilise alors l'inégalité $\dfrac{1}{1-u} \geq 1 + u$ pour $u < 1$, ce qui donne :

$$\frac{1}{f(x_{n+1}) - \min f} \geq \frac{1}{f(x_n) - \min f} \left(1 + \frac{f(x_n) - \min f}{2L\|x_0 - x^*\|^2}\right) = \frac{1}{f(x_n) - \min f} + \frac{1}{2L\|x_0 - x^*\|^2}.$$

Par récurrence, on obtient alors pour tout $n \in \mathbb{N}^*$:

$$\frac{1}{f(x_n) - \min f} \geq \frac{n}{2L\|x_0 - x^*\|^2},$$

ce qui équivaut à :

$$f(x_n) - \min f \leq \frac{2L\|x_0 - x^*\|^2}{n}.$$

□

[1]

Références

[1] Francis Bach. *Learning theory from first principles*. MIT Press, décembre 2024.

[2] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning : a survey. *arXiv*, 2018.

[3] Joachim Bona-Pellissier, François Malgouyres, and François Bachoc. Geometry-induced Implicit Regularization in Deep ReLU Neural Networks. working paper or preprint, February 2024.

[4] Mateusz Michalek Lek-Heng Lim and Yang Qi. Best k-layer neural network approximations. *Constructive approximation*, 55(2), 2021.

[5] Albert Novikoff. On convergence proofs for perceptrons. *Office of Naval Research et Stanford Research Institute*, 1963.

[6] Aude Rondepierre. *Méthodes numériques pour l'optimisation non linéaire déterministe*. Département Génie Mathématique et Modélisation, 2017-2018.