

Notes et commentaires au sujet des conférences de S. Mallat du Collège de France (2019)

Les réseaux de neurones multi-couches: le comment et le pourquoi.

J.E Campagne

Avril 2019; rév. 6 Mai 2025

Table des matières

1 Introduction de la série de conférences	7
1.1 Les domaines des mathématiques	7
1.2 Analyses de données	9
1.3 Apprentissage supervisé	10
1.3.1 Les 2 types de problèmes: Classification/Régression	10
1.3.2 Algorithme d'apprentissage	11
1.4 Algorithme non-supervisé	12
1.5 Retour sur la malédiction de la dimension	13
1.6 Les classificateurs linéaires	14
1.7 Réseau de Neurones/Neural Network (RN/NN)	16
1.8 Les CNN ou Réseaux convolutionnels profonds (RCp)	18
1.9 Plan du cours 2019	21
1.9.1 Temps 1 : on se fait une intuition et on se pose des questions	21
1.9.2 Temps 2: réseau à 1 couche cachée	22
1.9.3 Temps 3: réseaux multicouches	22
1.9.4 Temps 4: réseaux convolutifs (convolutionels)	22
2 Les Applications	23
2.1 Vision par ordinateur	23
2.1.1 Classification d'images	23
2.1.2 Classification en video	29
2.1.3 Autre type de problème: la segmentation d'images	30

2.2	Reconnaissance de la parole	31
2.2.1	Approche classique	32
2.2.2	La révolution (très récentes) des RNP	33
2.3	Le traitement du langage naturel	36
2.3.1	Vision du problème antérieure à 1990	36
2.3.2	Vision du problème postérieure à 1990	38
2.3.3	Traduction automatique	40
2.3.4	Raconter une vidéo	41
2.4	La Physique: interaction à N-corps	41
2.5	Lien avec la Neurophysiologie	42
2.6	Apprentissage par renforcement	43
2.7	Apprentissage non-supervisé	44
2.8	Generative Adversarial Networks (GAN)	45
2.9	Limites et opportunités	48
2.9.1	Face obscure à éclairer	48
2.9.2	Face qui motive	49
3	Point de vue mathématique	49
3.1	Introduction	49
3.2	Problème d'approximation	52
3.3	Problème d'estimation et optimisation	54
3.4	Autres questions	55
3.5	Approximation/Régularité, quel type de régularité?	56
3.6	Émergence des symétries et groupes locaux	57

3.6.1	Du global au local	57
3.6.2	Quel impact sur le réseau de neurones?	60
3.7	Séparation d'échelles, hiérarchie multiéchelles	61
3.8	La notion de parcimonie	65
3.9	Bilan sur le point de vue mathématique	67
4	D'où viennent les idées des réseaux de neurones?	67
4.1	La cybernétique	67
4.2	Le Perceptron (1957)	71
4.2.1	Introduction	71
4.2.2	L'algorithme de descente de gradient	73
4.2.3	La régularisation	78
4.2.4	SVM: Support Vector Machine	79
4.2.5	Bilan sur le Perceptron	80
5	Architecture multi-couches: Partie I	81
5.1	Introduction	81
5.2	L'expression de la sortie du réseau	82
5.2.1	Le cas des fonctions Booléennes	84
5.2.2	Usage de la régularité de la fonction	86
5.3	Théorème d'Universalité d'un réseau à 1-couche cachée	88
5.3.1	Base de Fourier	90
5.3.2	Approximation des sinus par σ	95

6 Architecture multicouches: Partie II	97
6.1 Introduction	97
6.2 Rappels sur le Théorème d'Universalité	98
6.3 Convergence de l'approximation \tilde{f}	99
6.4 Définition(s) de la Régularité	100
6.4.1 Régularité au sens des dérivées (Sobolev/Hilbert): Théorème d'optimalité de Maiorov	100
6.4.2 Autres types de régularité	102
6.4.3 Augmenter le nombre de couches: c'est mieux!	108
7 Optimisation des Réseaux de Neurones	109
7.1 Introduction	109
7.2 L'approche de Bayes et le principe du maximum de vraisemblance	110
7.2.1 Transformation du problème via Bayes	110
7.2.2 Maximum de vraisemblance	111
7.2.3 Divergence de Kullback-Leibler	112
7.2.4 Relation avec les modèles bayésiens	113
7.3 Mise en œuvre pour un réseau de neurones (classification)	116
7.3.1 Introduction	116
7.3.2 Introduction du softmax	117
7.3.3 Optimisation: cas particulier de la classification par régression logistique	118
7.3.4 Pour un réseau de neurones MLP	119

8 Algorithmes d'optimisation des MLP	119
8.1 Calcul du gradient dans les MLP (Back-prop)	119
8.1.1 Flux Forward & Backward	121
8.1.2 L'initialisation: le calcul $\nabla_{x_j} \ell$?	123
8.1.3 Rétropropagation des gradients	124
8.1.4 Les Jacobiens des F_j	125
8.1.5 Représentation graphique de l'algorithme	127
8.2 L'étude de la convergence du GD	127
8.2.1 GD par Batch ou Stochastique	127
8.2.2 Exemple de la fonction quadratique: convergence du GD	129
8.2.3 Normalisation par mini-batch	133
9 Gradient Stochastique	134
9.1 Introduction	135
9.2 Accélération du GD et SGD à pas variable	136
9.3 Cadre mathématique: fortement convexe et régulier	137
9.3.1 Méthode Batch	137
9.3.2 Méthode SGD	140
9.4 Généralisations ?	142
9.5 Problèmes d'optimisation	143

Avertissement: dans la suite, vous trouverez mes notes au style libre prises au fil de l'eau et remises en forme avec quelques commentaires (NDJE ou bien sections dédiées). Il est clair que des erreurs peuvent s'être glissées et je m'en excuse par avance. Je vous souhaite une bonne lecture.

1. Introduction de la série de conférences

Durant cette année 2019, et les suivantes, on va s'intéresser à **Pourquoi** ça marche, plutôt uniquement se focaliser sur **Comment** ça marche. En effet, il y a beaucoup de tutoriels qui donnent accès aux softwares de mise en œuvre des Réseaux de Neurones (acronymes usités dans ce cours: RN ou NN ou MLP pour Multi Layer Perceptron) pour résoudre des cas concrets. Donc, en faire un autre ici n'apporterait pas de valeur ajoutée. En revanche, le *Pourquoi* est du domaine de la recherche fondamentale que nous voulons promouvoir. Nous verrons d'ailleurs que bien des questions resteront en suspend.

1.1 Les domaines des mathématiques

Bien entendu, il y a un aspect algorithmique que nous étudierons (détails, origines...) mais au-delà pour comprendre un Réseau de Neurones profonds (RNp), cela nous demande des outils qui couvrent un très large champ des mathématiques:

- **statistiques**: on va collecter des données donc on part de mesures empiriques dont on calcule des estimateurs;
- **probabilités**: pour que les statistiques donnent des résultats précis, il faut faire des hypothèses sur les distributions de probabilités;
- **algorithmique** dont **l'optimisation** des paramètres. Au début des années 2000-10, on s'est concentré sur l'optimisation convexe, car il y a des théorèmes de convergence. Mais les RNp fourmillent de minima locaux, donc on est dans des situations **non convexes** et donc *a priori* les méthodes ne devraient pas marcher. Et pourtant *Ça marche!* et même bien. De nouvelles questions se posent: pourquoi ça marche? et dans quelles situations ça marche?
- Il y a d'autres branches qui vont intervenir, car on va s'apercevoir que ces RNp ne vont **bien marcher** que s'ils sont **bien structurés**. Depuis 2010, on a utilisé des

réseaux de neurones convolutionnels (CNN), et quand on veut savoir (et analyser) pourquoi ils fonctionnent si bien, on tombe dans le champ de **l'analyse harmonique** qui est liée à la *Transformée de Fourier* et aussi aux *analyses multiéchelles* de type *Ondelettes* déjà introduites en 2018.

- Ces réseaux (RNp) sont évolutifs par exemple au cours de l'apprentissage, et donc on peut les voir comme des **systèmes dynamiques**. C'est une nouvelle branche des mathématiques qui adresse ce problème de la dynamique et de la convergence de systèmes complexes.
- Enfin, un dernier domaine des mathématiques qui va nous servir, c'est la **géométrie**. En effet, ces RNp font de l'apprentissage en **très grande dimension**. Il faut comprendre quelles sont les structures dans ces espaces, c'est-à-dire apprêhender les **symétries** (les représentations de **groupes**) de ces systèmes.

Finalement, tous ces champs vont interagir et nous serons à la frontière de la connaissance actuelle des mathématiques.

Un autre aspect bien entendu qui est derrière les RNp est “**l'intelligence artificielle**” (IA). Bien que la chaire n'en fasse pas état, il est clair que les RNp posent les problèmes de relations/modélisations de l'interaction de notre cerveau avec son environnement par le biais des sens (vision, audition) mais aussi la communication (langage). Cela est très étroitement lié à la notion d'intelligence. Il y a par ailleurs une très longue histoire de l'étude de l'intelligence et l'on peut dire que la réponse des RNp est au moins surprenante, en tous les cas pas du tout intuitive.

Enfin, un dernier aspect qui va être sous-jacent à la série de cours, c'est **l'architecture de la complexité**. C'est le titre d'un article daté de 1962 de Herbert A. Simon (1916-2001), prix “Nobel” d'économie en 1978, mais surtout prix Turing en 1975 pour ses travaux en IA dont il est un des pionniers aux USA. Pourquoi cette notion est-elle importante?

Le premier cours (en 2018) portait sur la **Malédiction de la dimensionnalité** dans lequel on s'apercevait qu'une fonction à très grand nombre de variables (pensez aux millions de pixels d'une image) qui veut répondre à une question, doit faire face à une **extrêmement grande combinatoire**. Et *a priori* le nombre d'échantillons qu'il nous faudrait acquérir pour mener à bien un apprentissage est une fonction exponentielle de la dimension, et donc sans espoir. Il faut bien avoir cela en tête pour comprendre la complexité du problème. Et pourtant les **RN/RNp apprennent avec un nombre raisonnable d'échantillons** par

exemple pour reconnaître des sons, des images, un langage. Pourquoi? La raison en est que la fonction sous-jacente n'est pas arbitraire, elle a beaucoup de régularités. Mais **de quelle régularité s'agit-il?**

En fait si le monde (problème) est très complexe, il y a une forte structuration, et les RNP sont capables d'appréhender cette structuration, pour peu qu'ils puissent approximer la fonction originale. Donc, en étudiant les architectures des RNP, on en apprend sur la nature des régularités des fonctions qu'ils approximent. Et on se pose la question: alors que le problème est très complexe, pourquoi les fonctions sont *aussi* simples (cf. régulières)? On fera le lien avec la physique et les notions d'architectures des RNP.

1.2 Analyses de données

On aborde des notions qui sont **génériques**, c'est-à-dire qui s'appliquent aussi bien dans l'analyse de signaux audio ($d \approx 10^6$ échantillons temporels), ou d'images ($d \approx 10^6$ pixels), de textes ($d \approx 10^6$ mots), des agents d'un réseau social ($d \approx 10^9$ de personnes), voire des molécules de quelque chose en chimie ($d \approx 10^{24}$ entités). On a donc énormément de données à traiter et se pose la question de savoir comment trouver des méthodes sans trop se spécialiser dans tel ou tel domaine. Deux grandes questions viennent alors à l'esprit:

- **La modélisation** des données: c'est-à-dire la capture de leurs natures et leurs variabilités, comprendre leurs structures. On peut associer des méthodes d'apprentissage non supervisé (voir 1er cours de 2018), nous y reviendrons. Dans l'espace potentiellement très grand dans lequel évoluent les données, la question est: quel est en fait le sous-espace restreint qui est balayé? On va mettre en œuvre des modèles probabilistes. Les applications de la modélisation sont diverses: si on veut faire de la **compression** par exemple, il faut comprendre la structure, élaborer des modèles pour déterminer le plus petit nombre de bits possible qui permet la restauration du signal; si on veut **restaurer** des images prises par exemple en imagerie médicale, on a besoin de définir des modèles également; idem si l'on veut faire des **synthèses** d'images.
- **La prédition**: plus en lien avec l'IA, il s'agit à partir de données de connaître la réponse à une question (en général) par exemple: quel est le type d'animal dans telle ou telle image? Également, dans ce type de problématique, on trouve le diagnostic médical, l'analyse de textes (reconnaitre l'auteur, les champs/catégories du

texte...) et la traduction, etc. Tout cela fait partie du domaine de **l'apprentissage statistique** et c'est en somme à travers les travaux dans ce domaine qu'il y a eu une sorte de *révolution* dans les dix dernières années.

1.3 Apprentissage supervisé

1.3.1 Les 2 types de problèmes: Classification/Régression

On se place en grande dimension d comme on l'a vu précédemment, on note le vecteur d'un échantillon par

$$x = (x(1), \dots, x(d)) \in \mathbb{R}^d \quad (1)$$

Le premier type de problème est la **Classification**: si on donne x , que va être $f(x)$ la classe à laquelle il appartient? Mettons pour fixer les idées que j'ai des images: différents arbres, animaux, fleurs,... Le domaine des classes est très grand, qui plus est dans chaque classe il y a une **grande variabilité**. Dans l'apprentissage supervisé, on suppose que l'on a un certain nombre d'images/échantillons d'entraînement: c'est-à-dire que l'on dispose de n échantillons **étiquetés** $\{x_i, y_i = f(x_i)\}_{i \leq n}$. L'énorme variabilité à l'intérieur d'une classe est une façon de visualiser la **malédiction de la dimension**. Pour s'en sortir, il faut arriver à trouver ce qu'il y a de commun à tous les éléments d'une classe (révéler leur structure), autrement dit trouver les **invariants d'une classe**, et trouver parmi les invariants ceux qui **discriminent les différentes classes**. En sous-jacent qui dit *invariants* dit symétries et donc la théorie des groupes (cf. la géométrie).

Le second type de problème est la **Régression**: c'est-à-dire que l'on se pose la question de savoir que vaut $f(x)$ en tant de fonction par exemple à valeurs dans \mathbb{R} (et non comme index discret de classe). Un exemple de ce type de problème est la *Physique*. La question (à cent balles) est de savoir si l'on peut apprendre la Physique sans connaître les "lois" sous-jacentes¹, mais "simplement à partir des données". En chimie quantique, x va décrire la position de chaque atome et on aimerait calculer l'énergie quantique du système $f(x)$. En astrophysique, connaissant la position, la vitesse, la masse d'objets (planètes, étoiles, galaxies...) peut-on connaître l'énergie du système? On sait que si on connaissait $f(x)$ alors en la dérivant (gradient) on aurait accès aux "forces/interactions" du problème et

1. NDJE: ou bien peut-on trouver de nouvelles lois pertinentes?



FIGURE 1 – Vu schématique d'un algorithme qui partant de l'entrée x fournit une estimation \tilde{y} .

donc on apprendrait de la Physique... Mais si je rajoute, une molécule, une galaxie, serait-on capable de calculer son énergie sans connaître l'équation de Schrödinger, ou celle de la gravitation? Oui, mais! Il faut un apprentissage avec de **l'information *a priori*** et c'est un sujet d'étude de savoir comment cet *a priori* est capturé par les RNP.

Le cas de la Physique est intéressant, car on connaît beaucoup de principes qui contraignent $f(x)$, ce qui n'est pas le cas en classification des chats/chiens par exemple où on ne connaît rien *a priori* sur ce qui relie une image à un chien ou à un chat ou à un autre objet. En Physique, on peut se poser des questions beaucoup plus pertinentes sur le *Pourquoi ça marche*, surtout que la notion de **symétries** est fondamentale. Notons que la Physique au sens large (incluant la Chimie) était traditionnellement la discipline de la grande dimension, or *l'apprentissage statistique* est devenu également une discipline de la grande dimension, ainsi, il n'est donc pas étonnant que la première serve à guider le chemin pour comprendre la seconde.

1.3.2 Algorithme d'apprentissage

Prenons le cas d'une image x : on se pose la question “quel est l'animal dans cette image?” La réponse y en l'occurrence est “chien”. Un algorithme d'apprentissage en apprentissage statistique est **paramétré** (figure 1), avec potentiellement un très grand nombre de paramètres comme les poids du RNP. La phase d'apprentissage permet **d'optimiser l'algorithme** de façon à ce que l'estimation **sur les exemples** \tilde{y}_i approxime la bonne réponse y_i . L'enjeu évidemment, c'est la **généralisation**. C'est-à-dire si on donne un x

inconnu de la base d'entraînement, il faut que la réponse \tilde{y} de l'algorithme soit proche de la vraie valeur y . Dans l'idéal, elle devrait même être égale à celle-ci. La question sous-jacente est: quelles sont les fonctions qui vont pouvoir se prêter favorablement à l'apprentissage? Quelle est la **régularité** à laquelle doit satisfaire ces fonctions pour qu'à partir de quelques échantillons on puisse interpoler une réponse qui soit une bonne approximation de la solution. **L'algorithme paramétré ci-dessus peut être vu ainsi comme un algorithme d'interpolation.**

Point de vue mathématique: on attaque par le biais de la régularité qui révèle la structuration du problème. Or, on l'a vu lors du 1er cours (2018) que les notions de régularité par la **continuité/dérivabilité** ne sont pas du tout suffisantes et **ne fonctionnent pas du tout en grande dimension**.

Point de vue Informatique: tous les algorithmes qui sont à l'œuvre dans les RN sont très jolis, ils marchent bien, mais on les comprend mal. C'est donc un sujet de recherche fondamental.

1.4 Algorithme non-supervisé

On a des données et l'on veut comprendre quelles sont les structures sous-jacentes: vortex dans un fluide, filaments dans les gaz hélium/hydrogène interstellaires. Ce sont des problèmes de modélisation, et on aimerait connaître la densité de probabilité $p(x)$ de chaque réalisation x et, par exemple différentier entre un champ de turbulences d'un autre type de champ aléatoire.

Ce type de problème (par ex. la turbulence) n'est pas nouveaux en Physique, voir les papiers de Kolmogorov des années 1940, et pourtant leur résolution est toujours d'actualité. Notons que Kolmogorov introduit un modèle d'invariance d'échelle en lien avec l'équation de Navier-Stockes. Mais ce modèle, même s'il donne une approximation, n'est finalement pas adapté à la description fine des phénomènes turbulents.

Le problème de synthétisation/modélisation devient plus obscur quand on a à faire non pas à des "textures" comme des champs turbulents, mais à des visages ou des objets, voire des scènes complètes mélangeant des objets du quotidien. Déjà le problème est beaucoup plus mal posé que dans le cas de champs de turbulence, car il n'y a pas d'équations sous-jacentes (Navier-Stockes ou autres). Pourtant, il est manifeste qu'empiriquement les RNP

sont capables d'adresser ce type de problèmes (en Physique ou en dehors) sans que l'on sache vraiment pourquoi.

Dans l'apprentissage non supervisé, on ne dispose pas de *labels*, or, il faut reconnaître que s'il n'y a pas de coût associé à l'acquisition de données (quoique cela dépende du domaine), par contre, il est couteux dans le monde industriel d'avoir les labels. Donc, il est naturel d'envisager des algorithmes de **clustering** et c'est une thématique qui revient de temps en temps. La question est alors: est-ce un fantasme de pouvoir faire de la classification sans label? Dans certain cas "simples" en effet ça va marcher. En fait, il faut que **le nombre de degrés de liberté soit faible** (cf. la dimensionnalité du problème). Ça ne marche pas en grande dimension (**malédiction**) car les points sont loin des uns des autres, et il faut une "sphère" qui couvre la quasi-totalité de l'espace pour collecter un échantillon semblable (voir 1er cours de 2018 et la section suivante).

1.5 Retour sur la malédiction de la dimension

C'était le sujet du cours de 2018, mais on y revient ici, car il faut bien assimiler le problème. En apprentissage supervisé, on a donc des échantillons étiquetés $\{x_i, y_i = f(x_i)\}_{i \leq n}$ et l'on cherche à approximer $f(x)$. On peut représenter x_i comme 1 point en dimension d et $f(x_i)$ comme un indice de couleur par exemple. Ainsi, on peut étudier la répartition de ces points dans l'espace.

Maintenant, soit x un nouveau point: que vaut $f(x)$? Il vient immédiatement l'idée d'interpoler au point x la valeur de f . Pour cela, il faut collecter les plus proches voisins de x qui sont étiquetés. Le problème est qu'en général en **grande dimension** $\|x - x_i\|$ est très grand! Pour s'en convaincre prenons l'exemple suivant. On se place en dimension d dans un espace $[0, 1]^d$, et l'on veut garantir que pour tout x , on trouve un voisin à une distance inférieure à $1/10$, c'est-à-dire, il faut que les x_i satisfassent:

$$\forall x \in [0, 1]^d, \exists x_i \in [0, 1]^d \quad / \quad \|x - x_i\| \leq 1/10 \quad (2)$$

Si les x_i sont répartis uniformément (cf. on ne privilégie pas un x particulier), il faut 10^d points qui couvrent tout l'espace. Or, il faut bien penser que $d \approx 10^6$ dans les cas usuels de grande dimension, donc le nombre d'échantillons est plus qu'astronomique. En pratique, dès que $d \geq 20$, **on est dans un régime de grande dimension**.

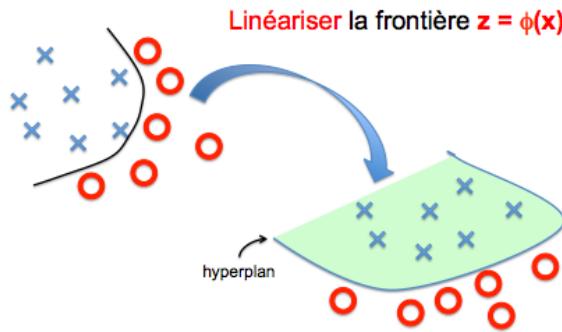


FIGURE 2 – Illustration de l’usage d’une transformation $x \rightarrow \Phi(x)$ pour se placer dans un espace, où le problème de classification binaire est séparable par un hyperplan.

Donc, il est assez rare que l’on puisse utiliser des algorithmes de plus proches voisins; en tous les cas, il ne faut le réserver qu’aux petites dimensions. En revanche, en grande dimension, si l’on veut que ça marche, il va falloir soit découvrir des formes de **régularité beaucoup plus globales**, soit faire de la **réduction de dimensionnalité** qui s’interprète aussi comme la découverte de régularités sous-jacentes qui permettent par exemple l’identification de classes d’objets (fleurs, chaises, lampes, visages...). Moralité, avoir des labels *a priori* est très important en grande dimension et pourtant, c’est le plus couteux.

1.6 Les classificateurs linéaires

Ces notions ont été abordées dans le cours de 2018, elles datent des années 90. L’idée est de rendre le problème le plus simple possible en le linéarisant. Ce que l’on veut, c’est trouver une transformation (ou changement de variables) Φ telle qu’on puisse **linéariser la frontière** (figure 2). Prenons pour illustrer le cas d’une classification à 2 classes:

$$x = (v_1, \dots, v_d) \xrightarrow{\Phi} \Phi(x) = (v'_1, \dots, v'_d) \quad (3)$$

Si la technique de *linéarisation* est courante en math/physique, ici le x se trouve dans un espace de grande dimension et comprendre la nature de ce changement de variables est plus complexe. Sinon une fois la transformation faite la surface de séparation est un hyperplan de vecteur normal w (figure 3).

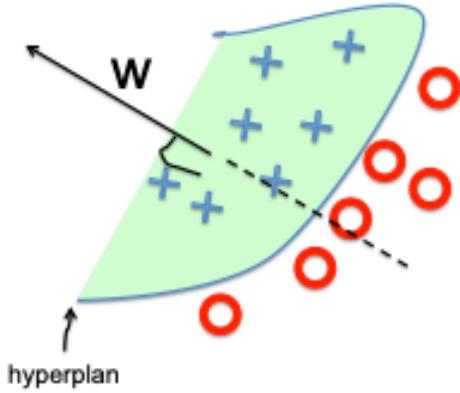


FIGURE 3 – Schéma de la projection normale à l'hyperplan séparateur.

L'apprentissage dans ce cas de figure est la détermination du vecteur w . Le classificateur a une expression simple:

$$\tilde{f}(x) = \text{sign} \{ \langle w, \Phi(x) \rangle + b \} = \text{sign} \left(\sum_k w_k v'_k + b \right) \quad (4)$$

Un exemple trivial de ce type de classificateur est celui qui vous indique sur une image s'il y a un camion de pompiers ou une voiture. La diversité à l'intérieur de ces 2 classes peut être très grande (cf. d est le nombre de pixels de l'image) et pourtant compter le nombre de pixels de couleur rouge est très discriminant. Dans des cas moins triviaux, les v'_k sont souvent appelés **des patterns ou features**, et bien que cela ne soit pas la seule interprétation possible la somme $\sum_k w_k v'_k$ peut être vue comme un *vote* entre les patterns. Donc, cette procédure de linéarisation rend le problème très simple, mais dans quelle situation peut-elle être appliquée? ou bien dans quel cas peut-on trouver la transformation $\Phi(x)$?². Deux cas de figures se présentent alors:

- soit on dispose de connaissances *a priori* (cf. les camions de pompiers sont gros et rouges) et dans ce cas la transformation $\Phi(x)$ est connue;
- soit on ne dispose **pas de connaissances a priori** (notons que c'est la majorité des cas de figure) alors, il va falloir “**apprendre**” le $\Phi(x)$. Les RN donnent une stratégie

2. Notons en passant que trouver w n'est pas le problème, car il y a beaucoup d'algorithmes en algèbre linéaire (voir SVM, la Régression Logistique...) qui traitent de ce sujet.

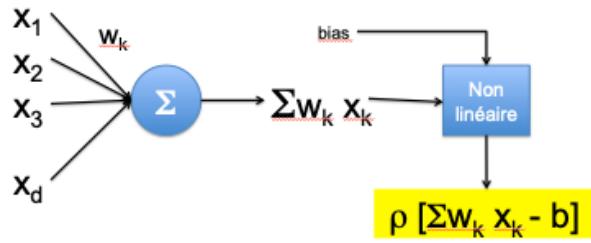


FIGURE 4 – Schéma du fonctionnement d'un neurone avec une partie d'*agrégation linéaire* et une partie d'*activation non linéaire*.

dans ce cas de figure où à la fois le $\Phi(x)$ et le w sont **déterminés en même temps** lors de la phase d'apprentissage. Ceci dit, nous verrons qu'on ne part pas sans *a priori* totalement, mais c'est bien avec toutes les variables disponibles en dimension d que l'on travaillera³.

1.7 Réseau de Neurones/Neural Network (RN/NN)

Les réseaux de neurones informatiques datent des années 50, ils ont été modelés par analogie de cerveaux de petits animaux. Le modèle à 1 couche a été donné dans le cours de 2018, il correspond au schéma⁴ de la figure 4. Comme boite “non-linéaire”, on peut penser à un **rectificateur (ReLU)** qui nous le verrons est très utilisé pour les Réseaux de Neurones profonds (RNP):

$$\begin{cases} 0 & \text{si } \sum_k w_k x_k < b \\ \sum_k w_k x_k - b & \text{sinon} \end{cases} \quad (5)$$

Le seuillage par le biais b permet d'activer un neurone avec **parcimonie**⁵.

On remarque que ce modèle à 1 couche est du même type que le classificateur linéaire avec la frontière déterminée par un hyperplan caractérisé par les w_k et le biais b . Mainte-

3. NDJE: Cependant, les méthodes par Kernel permettent de se passer de la transformation en tant que telle puisque dans ce cas, il "suffit" d'appliquer une fonction de similitude K ; mais il n'en reste pas moins que K est un choix *a priori*.

4. nb. le signe de b a changé par rapport aux illustrations de 2018.

5. nb. c'est un effet du même type que l'on retrouve en Analyse d'Ondelettes.

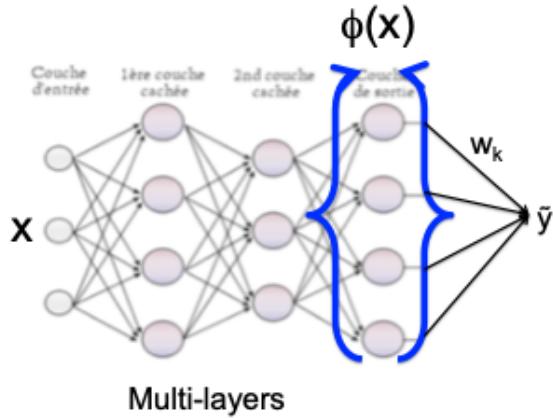


FIGURE 5 – Schéma simplifié d'un réseau de neurones multicouches où la représentation Φ est le résultat de l'ensemble des sous-couches; elle est utilisée en fin de réseau pour procéder par exemple à une classification.

nant, on peut **enchaîner des couches de neurones pour les mettre en réseau** (RN ou MLP pour Multi Layer Perceptron) comme sur la figure 5 (2 couches cachées). On constate alors que la **fonction $\Phi(x)$** que nous cherchions auparavant est le résultat de **l'apprentissage de la partie amont du réseau** de neurones; la dernière opération est semblable à celle d'un seul neurone de la figure 4.

À quoi correspond **l'apprentissage d'un RN**? Il consiste à la détermination de ses paramètres (les poids entre les différentes couches) en minimisant l'erreur entre \tilde{y}_i et la vraie valeur étiquetée y_i . Pour cela, on définit des **fonctions de coût** dont on cherche le minimum. C'est un **problème d'optimisation difficile**, car il y a pléthore de minima locaux dont il faut se sortir pour trouver le minimum global ou tout du moins s'en approcher⁶.

Ce qu'il faut avoir en tête et qu'il y a de **l'information a priori**, nous l'évoquions dans la section précédente, elle réside dans **l'architecture du réseau**. *Elle n'est pas pour le moment du moins découverte au cours de l'apprentissage* bien que les poids peuvent en quelque sorte donner une indication des neurones les plus actifs et ceux dont l'activité est quasi-nulle. Est-ce que cette artillerie marche en général: la réponse est non! C'est-à-dire que si

6. NDJE: Bien que des études très récentes montrent que les solutions trouvées par les méthodes de Descente de Gradient ne sont pas n'importe quelles solutions et que les RN ainsi obtenus donnent des résultats similaires

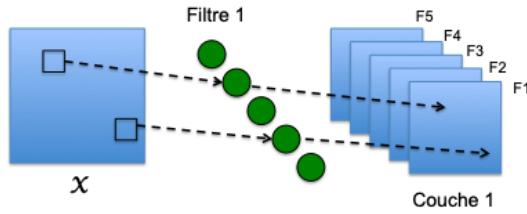


FIGURE 6 – Schématisation d'un premier étage de convolution avec 5 filtres différents. Pour chaque filtre, ex. le filtre F1, chaque neurone s'occupe d'une petite partie de l'image d'origine et tous les poids associés à chaque neurone sont identiques d'un neurone à l'autre.

vous mettez **plein de neurones et plein de couches** inter-connectées, et vous essayez de faire une optimisation en “croisant les doigts”, et bien **ça ne donne rien du tout!**. Donc, ce ne seront que des architectures bien particulières (*information a priori*) qui vont permettre de sortir quelque chose qui donne satisfaction. D'une manière empirique, on a découvert des **architectures pertinentes** pour certains problèmes et en sous-jacent cela veut dire que la complexité du problème a bien été capturée (réduite) par ces architectures.

1.8 Les CNN ou Réseaux convolutionnels profonds (RCp)

Une avancée significative a été réalisée dans les années 1990 par Y. LeCun (AT&T Bell Lab.) & J. Bengio (Univ. Montreal) dans la façon de concevoir l'architecture de RN⁷. Ils ont capturé dans l'architecture des symétries du problème comme **l'invariance par translation** (*nb. on se prévaut aussi de distorsions locales*) par l'usage de **filtres par convolution** qui capturent cette connaissance *a priori* du problème et réduisent considérablement les connections du/des premier(s) étages du RN, facilitant du même coup l'optimisation des poids par l'apprentissage.

Voyons de quoi il retourne progressivement. On va prendre pour fixer les idées une image à $n \times n = d$ pixels, cependant il faut bien avoir en tête que, comme l'indiquait d'ailleurs l'article de LeCun et Bengio, la procédure s'adresse également à des échantillons

7. NDJE: voir <http://yann.lecun.com/exdb/lenet/index.html> qui retrace les premiers réseaux convolutionnels, lesquels ont servi à reconnaître automatiquement l'écriture des chiffres pour les chèques. Notons que Y. LeCun, J. Bengio et G. Hinton ont reçu le Prix Turing 2019.

vocaux et à des séries temporelles. En premier lieu, **chaque neurone** de la première couche (les bulles vertes dans la figure 6) n'est **connecté qu'à** (ou responsable de) une sous-partie de l'image initiale typiquement un **patch** de petite taille 3×3 ou 5×5 par exemple. **Chaque neurone** de ce type fait donc une **combinaison linéaire locale** des valeurs de pixels auxquels il est connecté (x_i avec i contraint dans une région) et on lui associe aussi un **rectificateur** (ou autre fonction non linéaire).

Or, imaginons que dans l'image, on veuille reconnaître un petit objet (ex. un bâton), on ne connaît pas *a priori* où il se trouve dans l'image, donc il n'y a aucune raison de privilégier la réponse “à un bâton” dans l'un ou l'autre des 2 patchs de la figure 6. **Conséquence: les poids des w_k des 2 neurones doivent être les mêmes, ce qui traduit l'invariance par translation.** Ainsi, on a une **réduction considérable de la complexité**: primo chaque neurone s'occupe d'une dizaine d'entrées et secundo la matrice de poids est la même pour tous les neurones. Si on a maintenant une **famille de filtres** avec chacun une spécialité pour tel ou tel traitement, les résultats se traduisent par autant de sorties, lesquelles sont notés F_1, F_2, \dots, F_5 sur la figure 6,

Ensuite (figure 7), les neurones de la couche suivante s'occupent toujours d'un petit patch, mais cette fois, ils considèrent la même zone dans les différentes images filtrées (cf. le petit tube en pointillé). **Chaque neurone fait la combinaison linéaire des résultats du premier étage dans un petit cube.** Or, par le même argument que précédemment, on en déduit que pour **un autre neurone** qui s'occupe d'un autre petit cube **les poids sont les mêmes**. Au passage, on perd la localisation spatiale dans l'image originale. Mais on construit, une petite image de la couche 2, et en faisant varier les poids (cf. en changeant le filtre) on obtient une série de petites images qui constituent la couche 2 (*nb. à chaque combinaison linéaire on associe un rectificateur*).

On peut continuer **en empilant les couches de filtrage pour construire l'architecture du réseau (CNN)**. À la fin, on termine ici par un neurone connecté (on a en général plutôt un réseau “classique” fully connected) qui donne l'estimation \hat{y} et la dernière couche donne en fait la transformation (changement de variables) $\Phi(x)$ (la figure 8). En définitive, on a réduit considérablement la complexité initiale du problème car les neurones sont spécialisés et on a mis en œuvre l'invariance par translation en minimisant le nombre de poids à chaque couche. Cependant, en empilant les couches, on aboutit rapidement à des **RCp de plusieurs centaines de millions de paramètres** à optimiser.

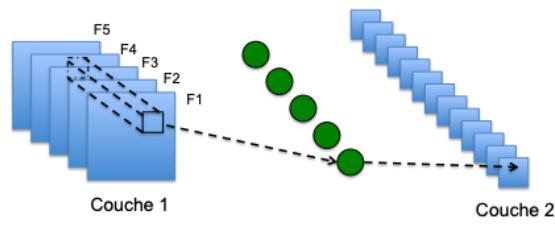


FIGURE 7 – Schématisation de la seconde couche convolutionnelle: les résultats de chaque filtre du premier étage sont associés dans un nouveau processus de convolution.

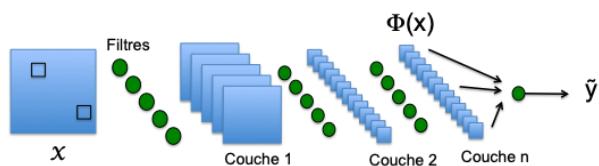


FIGURE 8 – Vue d'ensemble du réseau convolutionnel avec en dernière partie l'étape d'agrégation pour sortir \tilde{y} à partir de la représentation $\Phi(x)$ apprise lors du processus d'entraînement.

Le nombre de paramètres est beaucoup plus grand (en général) que la dimensionnalité d de l'espace d'origine dans lequel évolue x . On entraîne ensuite le réseau avec typiquement des milliers, voire des millions d'échantillons, et normalement ça ne devrait pas marcher. **Or, la grosse surprise est que cela marche:** les résultats sur des benchmarks “classiques” sont tout à fait remarquables et *génériques*, à savoir que la même architecture est capable d'attaquer des problèmes très divers comme la classification d'images, la reconnaissance de la parole, des problèmes de traduction, des problèmes de physique. Nous analyserons dans les cours **le comment ça marche**, car il y a un certain art algorithmique avec les méthodes de minimisations qui ont été mises au point comme la descente de gradient stochastique; mais on essayera de s'attacher surtout au **pourquoi ça marche**.

Voyons empiriquement ce qu'il se passe. Les neurones sont capables de **réduire considérablement la dimension du problème**, en effet si on part typiquement d'un échantillon d'un espace à 10^6 dimensions (ex. le nombre de pixels d'une image), **la dernière couche à typiquement 100 variables**. D'autre part, quand on analyse ce qu'il se passe dans les couches, **les frontières entre classes s'aplatissent** (linéarisent) et à la fin, on a un problème à hyperplans pour faire la classification. Mais pourquoi cet aplatissement se réalise-t-il? **L'architecture bien entendu joue un rôle crucial et c'est ce lien que nous étudierons via l'analyse harmonique et la géométrie (théorie des groupes)**.

1.9 Plan du cours 2019

1.9.1 Temps 1 : on se fait une intuition et on se pose des questions

On va **commencer par les applications**, car s'il y a eu une période de questionnement durant laquelle on a pu se demander si tout cela n'était que du *buzz*, *un effet de mode* et n'allait pas durer, maintenant on en n'est plus là. Les résultats sont impressionnantes, les enjeux industriels sont considérables. Il faut donc se forger une intuition de ces objets complexes en voyant comment ils fonctionnent. Et dans la foulée, on posera les **questions mathématiques que cela soulève**. S. Mallat indique qu'il va se *balader* dans un univers assez vaste où chaque expert d'une discipline peut contribuer à la compréhension du problème (*taper avec son marteau*). Donc, on verra un peu tous les points de vue qui éclairent le problème de la compréhension de ces RNP. On reviendra à l'origine des questions de l'Intelligence Artificielle avec *La cybernétique* (Wiener 1947, Herbert Simons 1960), les

premiers réseaux de Franck Roseblatt (Perceptron de 1957). En fait, les questions dès cette époque sont fondamentales.

1.9.2 Temps 2: réseau à 1 couche cachée

Ensuite, on va passer aux réseaux à 2 couches (c'est-à-dire à **1 couche cachée**) car c'est le cas que l'on arrive assez bien à comprendre maintenant. On connaît un fameux **théorème d'approximation universelle** de Cybenko (1988)⁸. Ce théorème en résumé vous dit que **ce type de réseau peut approximer un peu n'importe quelle fonction**. Mais on sait que ce **résultat est trompeur**, car si la **convergence** est garantie par le théorème, alors si elle est **exponentiellement lente** en pratique, on n'est pas avancé. Donc, il va falloir étudier la théorie de l'approximation, parce que le problème n'est pas que l'erreur tende vers 0 simplement, mais bien qu'elle tende vers 0 **rapidement**, et suffisamment pour qu'elle soit effective avec le nombre d'exemples étiquetés dont on dispose typiquement.

1.9.3 Temps 3: réseaux multicouches

Ensuite, on passera aux **réseaux multicouches** (MLP: multi-layer perceptrons, ou RNP: réseaux de neurones profonds). On regardera les performances d'approximation, mais on sera vite **limité par les outils mathématiques**. On passera alors sur le point de vue **algorithmique**. Ainsi, on verra l'optimisation par **gradient stochastique**. L'idée est ancienne (Robbins et Monro 1951) mais il y a eu énormément de recherche dans ce domaine (voir d'ailleurs le séminaire de 2018 sur des versions *lazy*). Ces algorithmes sont particulièrement efficaces pour entraîner des réseaux multicouches à travers la méthode de la **rétro-propagation** (Rumelhart, Hinton, Geoffrey; 1986).

1.9.4 Temps 4: réseaux convolutifs (convolutionnels)

Ensuite, nous étudierons les réseaux convolutifs (CNN ou RCp) introduits par Y. LeCun en 1990 et dont les résultats dans la reconnaissance de l'écriture humaine, puis dans d'autres domaines sont spectaculaires. On rentre dans un monde de *mathématiques*,

8. NDJE: en 2018 S. Mallat l'a énoncé à la fin de sa série de cours repoussant sa démonstration à cette année.

algorithmique et *applications* et on va prendre son temps pour avancer, identifier les questions sous-jacentes et tenter d'y répondre. Donc, pour les personnes pressées qui veulent programmer bille en tête, ce cours n'est pas fait pour elles, il y a beaucoup de "tutos" sur le Web pour manier Torch, Keras, TensorFlow, etc.

NDJE: ce quatrième volet a été repoussé à 2020.

2. Les Applications

Une expérience réalisée par Thorpe *et al.* en 1996⁹ a montré que pour un humain, savoir s'il y a un animal dans une image se fait en 150 ms, malgré la diversité et la complexité des images. Cela signifie que la réponse est certes efficaces mais relativement lente pour passer typiquement 10 couches neuronales. D'ailleurs, les observations tendent à montrer qu'il n'y a pas de boucles de feed-back (rétroaction) car cela serait encore plus lent.

2.1 Vision par ordinateur

2.1.1 Classification d'images

La première application qui a été vraiment significative est celle de la **vision par ordinateur** qui a démarré dans les années 70. Typiquement sur une image du type de celle de la figure 9, on veut pouvoir d'écrire la scène: par exemple reconnaître que l'on joue au football, reconnaître la position du joueur, découvrir la position du ballon... Les réponses à ces questions dans les années 70-90 passaient par le raisonnement suivant. Il faut reconnaître des *patterns*, donc comprendre la nature spécifique d'un ballon (ex. forme), d'un joueur (ex. bras, jambes...), et ensuite essayer d'extraire ces informations de l'image. Comment s'y prend-on? Par exemple, on va extraire **les contours** (ici, on peut en appliquer des filtres) comme sur la figure 9 (droite) traitée par Mathematica EdgeDetect. Il y a eu beaucoup de papiers dans les années 70-80 décrivant comment détecter au mieux les contours. Et on était assez "content", car on avait réduit de beaucoup le nombre de points pertinents, et ça devait être plus "facile".

9. Nature, Jun 6;381(6582):520-2.

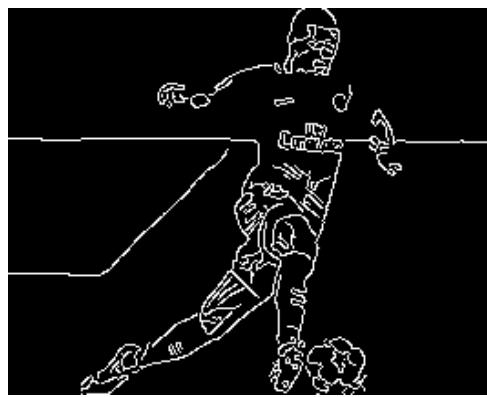


FIGURE 9 – Illustration de l'usage de techniques "classiques" d'analyse des structures d'une image: ici détection des contours.

Maintenant, il faut **reconnaitre les objets** (patterns) et comment ils sont **agencés les uns par rapport aux autres** dans l'image. Et là ça commence à coincer, voire, on ne peut répondre que dans des cas simples. En effet, comment définir des *métriques* pour comparer des contours? Les problèmes étaient par exemple: qu'est-ce qu'une forme, comment la décrire, comment les "objets" au sens large interagissent entre eux? Et on a regardé ces problèmes selon l'angle des **représentations symboliques** et **la grammaire des images** (le ciel est plutôt bleu et en haut d'une image, les pieds plutôt en bas, la tête ronde plutôt en haut de l'image, etc). En fait, toute cette approche du problème de reconnaissance d'image a été fortement **influencée par la grammaire du langage naturel** (*mouvement cognitivisme*, S. Mallat cite Noam Chomsky, le fondateur de la linguistique générative). Cependant, si ces méthodes fonctionnent dans des cas simples (ex. la reconnaissance d'un visage peut se faire à l'aide de quelque points), par contre, elles ne sont pas du tout adaptées dès que l'image est un tant soit peu complexe.

Il y a eu un **échec** de cette approche qui n'a pas été évident de mettre en évidence surtout que sociologiquement la quasi-totalité de la communauté des chercheurs dans ce domaine ont tous travaillé dans le même *moule méthodologique* pendant une trentaine d'années. Et il faut bien se rendre compte que l'approche par Réseaux de Neurones pour attaquer ces problèmes est radicalement différente et qu'elle n'a pas été acceptée tout de suite.

Les RN vont montrer une très nette avancée dans ce domaine en 2012, donc après

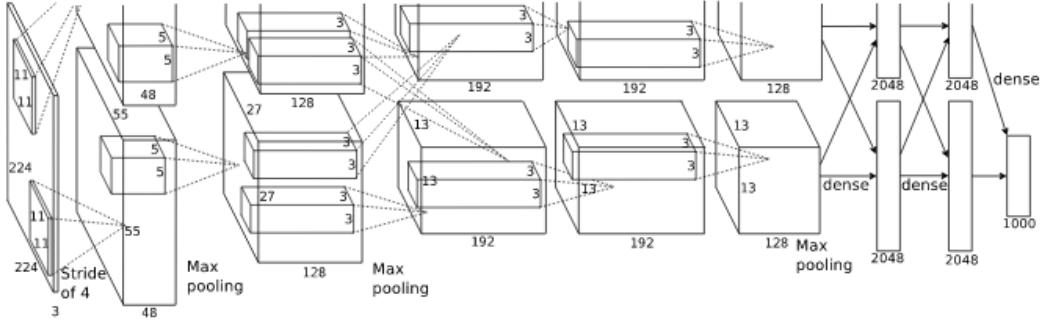


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

FIGURE 10 – Architecture d’AlexNet (2012) le premier réseau de neurones à battre les méthodes à noyau sur ImageNet data challenge.

quasiment 25 ans de recherche. Il s’agissait de résultats d’une compétition **ImageNet**, une base de données de 1 million d’images et 2000 classes d’objets constituée par des chercheurs à Princeton. La compétition avait pour but de **savoir si on peut faire de la classification** en apprenant avec ces images? On était à cette époque dans une période où les algorithmes d’apprentissages étaient de type **à noyau** (Kernel methods: Kernel PCA, Kernel SVM...), c’est-à-dire que les **features** étaient établis *empiriquement* par un pré-processing des images et extraits pour être donnés aux algorithmes. Le domaine décolle entre 2000-2010 avec ces techniques qui donnaient de bons résultats. Mais le problème **ImageNet restait très difficile**.

Puis, une équipe canadienne autour de Goeffrey Hinton (Alex Krizhevsky et Ilya Sutskever) fait part de leur score avec une architecture de réseau présentée en 2012 (soumis à NIPS 2012) dont la structure issue de leur papier (**AlexNet**) est présentée sur la figure 10. C’est un **réseau convolutionnel** à 5 couches entraîné avec 1.2 millions d’images (224 x 224) contenant 1000 classes. La première couche de filtrage (48 filtres différents) est composée de neurones dont chacun prend en charge des patches de 11x11, la seconde couche fait un sous-échantillonnage, et on enchaîne avec 128 filtres, et ainsi de suite. À la fin, il y a un classificateur dense qui se termine par un vecteur à 1000 variables qui fournit le résultat de la classification.

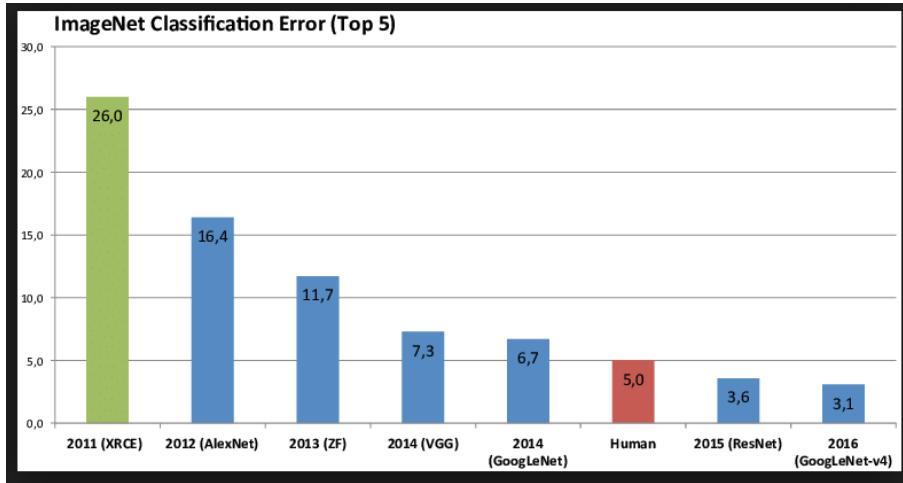


FIGURE 11 – Évolution du taux d’erreur sur ImageNet des différents algorithmes: AlexNet en 2012 marque le début de l’ère des Réseaux de Neurones.

Il a fallu beaucoup d’essais avant d’arriver à cette architecture. Mais le résultat remarquable, c’est que ça marche!, car ils ont obtenu $\sim 17\%$ d’erreur¹⁰, ce qui était un gain d’un facteur d’environ 30% par rapport à la concurrence (voir figure 11). **AlexNet** est devenu un cas d’école et une architecture simple à reproduire avec les librairies actuelles. Notons que les meilleurs réseaux actuels tournent avec 150 couches et arrivent à un 3% d’erreur soit en dessous de la performance humaine (voir plus bas). Pourquoi faut-il augmenter la taille du réseau? Ce n’est pas forcément pour capturer plus d’information, mais c’est plutôt lié à des problèmes d’optimisation, nous y reviendrons.

Pourquoi ça marche? et pourtant, les images sont compliquées! On pense que si ça marche aussi bien, c’est que le réseau a pu capturer (apprentissage) des structures sous-jacentes, des *invariants*. Quels sont-ils? **Quelle est la nature de la connaissance extraite par le RNP?** Ces questions sont importantes, car si l’on ne comprend pas le fonctionnement de ce type de réseau, on l’utilise comme une **boîte noire** sans être sûre si oui ou non la réponse est correcte. Qui plus est dans certains cas **on doit être capable d’expliquer le résultat**: ex. lors de sélection de candidats à l’embauche, il peut y avoir des contentieux, lors d’un diagnostic médical, on peut orienter vers le bon traitement, ou pas... .

Évolution des erreurs: sur la figure 11 sont collectés les erreurs Top 5 (*c'est-à-dire*

10. NDJE: S. Mallat cite un chiffre de 15.3% qui vaut pour une variante de l’architecture présentée.

que le l'on compare le vrai label aux 5 labels du classifieurs de plus grande probabilité) au cours des années depuis 2011. Avant AlexNet, le meilleur résultat de l'ordre de 26% était obtenu avec un algorithme de type k-NN (k-nearest neighbor) ou de type NCM (nearest class mean classifier) avec des descripteurs locaux très bien structurés. Puis, AlexNet lance le top départ des réseaux de neurones avec des raffinements sur l'architecture et sur les algorithmes d'apprentissage¹¹.

On remarque que **les récents résultats montrent une performance “supérieure” à celle d'un humain.** C'est un peu rapide somme toute car la nature des erreurs est différente. Il est clair qu'un humain qui découvre pour la première fois l'image d'un animal ou d'une fleur exotique ne va pas la reconnaître, par contre si on lui représente le même animal ou la même fleur dans une autre image, il a une aptitude à apprendre bien supérieure au RNp à qui il faut “beaucoup” d'exemples pour apprendre. D'un autre côté, dans certains cas, les erreurs des RNp sont surprenantes, donc **la nature de leurs erreurs est au moins différente de celle d'un humain.**

Maintenant comprendre pourquoi un **système complexe fait des erreurs** est très important (à part l'aspect intellectuel), car cela est connecté **en ingénierie à sa stabilité/instabilité.** Cette problématique, on la voit pour les RNp avec ce qu'on appelle les **exemples adversaires**. Sur la figure 12, la colonne de gauche sont des images d'ImageNet correctement étiquetées (x), au milieu ce sont des images d'erreur que l'on ajoute ϵ pour “tromper” le RNp (ici AlexNet) avec $\|\epsilon\| < 10^{-2}\|x\|$ (*les défauts sont imperceptibles à l'oeil nu et il faut les magnifier d'un facteur 10 pour “voir” qq chose dans la colonne du milieu*). Les images de **la colonne de droite** constituées par $x + \epsilon$ sont toutes identifiées comme une **espèce d'autruche!** Il y a donc une **instabilité que l'on ne veut absolument pas avoir** et donc qu'il va falloir comprendre pour trouver une parade.

En fait, on comprend la nature de ces instabilités. D'ailleurs dans l'exemple ci-dessus le bruit a été particulièrement choisi pour faire planter le réseau. Une première réaction serait de se dire que tomber sur un bruit qui reproduit ce type de défauts particuliers à une probabilité extrêmement très faible: on oublie... Mais, **imaginons qu'au lieu de classifier des images, ce type de RNp soit aux commandes d'un avion de ligne: a-t-on le droit de laisser passer ce type de défaut qui feraient "crasher" l'appareil?** La réponse va de soi.

11. NDJE: nb. et le hardware qui suit...

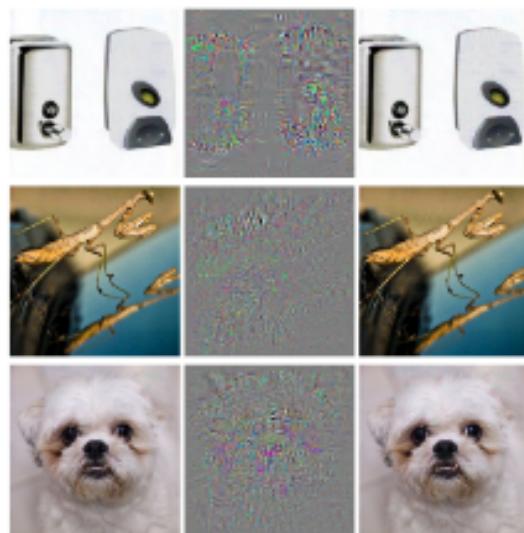


FIGURE 12 – Illustration de ce que sont des exemples adversaires: ici c'est AlexNet qui est pris en défaut de reconnaître les images "bruitées" de droite, classées comme "autruche", alors que sans bruit, il les classe très bien. Ledit bruit est quand même très spécial, et sa structure est révélée (images du milieu) quand elle est magnifiée d'un facteur 10. En norme L2 ce bruit ne représente que 1% du signal.



FIGURE 13 – Arrêt sur image d'une vidéo (https://www.youtube.com/watch?v=qrzQ_AB1DZk) qui montre le résultat d'un CNN avec en haut à gauche les 3 types de sport ayant les scores les plus élevés. Extrait de la publication <https://cs.stanford.edu/people/karpathy/deepvideo/>: *classification using a new dataset of 1 million YouTube videos belonging to 487 classes.*

Cependant, éviter ces défauts nuit en fait à la performance du réseau: il y a une balance entre performance et stabilité. Donc, l'idée serait au moins de s'assurer que **la probabilité d'occurrence de ces défauts “catastrophiques” soit extrêmement faible**. Notons au passage que ces “cas pathologiques” sont contre-intuitifs, car ils viennent **contredire la très bonne aptitude de ces RNP à procéder à la généralisation une fois entraînés**

2.1.2 Classification en video

Sur la vidéo en ligne dont une image arrêtée est montrée sur la figure 13, on peut apprécier le résultat d'un classifieur à base de RNP Convolutif (compétition CVPR 2014) agissant en temps réel pour reconnaître le type de sport pratiqué. En fait la classification est faite image par image. Les images sont complexes et le nombre de classes est élevé et pourtant, ça marche remarquablement jusqu'à 80% pour le Top 5 des classes.



FIGURE 14 – Le type d’image qui pose problème: faut-il partir de "contours" pour deviner ce qu’il y a dans l’image, ou bien partir de "formes" *a priori* et les chercher dans l’image?

2.1.3 Autre type de problème: la segmentation d’images

Trouver le contour d’un objet inconnu dans une image, par exemple un dalmatien dans une image noir et blanc tachetée (figure 14). On voit que pour résoudre ces images, il faut avoir une connaissance *a priori* de ce qu’est un chien … ou tel organe pour le cas d’une image médicale. Ce problème dit de *bas niveau* qu’est la segmentation est **à l’interface avec les problèmes de reconnaissance de structure**. Ce type de problème est crucial dans certains domaines comme l’interprétation d’images médicales et bien sûr la “voiture autonome”. Cela marche bien avec les réseaux de neurones comme sur l’exemple de la figure 15.

Le réseau reconnaît à la fois **les structures et les contours**: l’information de **bas niveau (le contour)** est en interaction — dans le réseau — avec l’information de **haut niveau (le type de structure)**. Ce problème a été un sujet de recherche depuis les années 60 (cf. Roberts cross operator) où on se posait la question de savoir de quel bout faut-il prendre le problème: ex. faut-il reconnaître des “transitions” pour définir un contour qui par la forme donne la structure de l’objet? Ou bien faut-il partir d’un type d’objet que l’on essaye de reconnaître pour déterminer son contour? En fait le RNP permet des boucles de rétroactions qui mélangent les 2 points de vue.

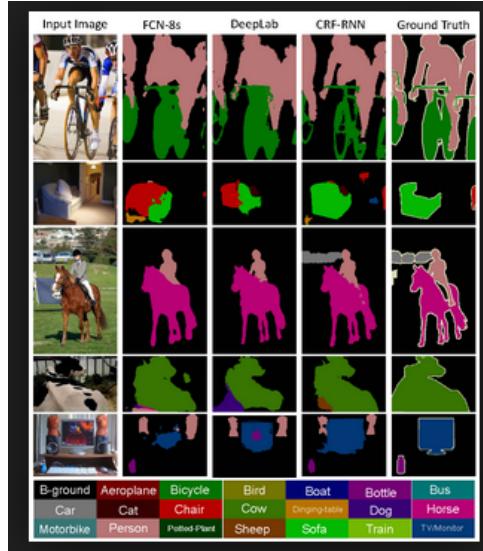


FIGURE 15 – Exemple de segmentation avec un CNN.

Ce type d’algorithme de segmentation est implanté dans des chips qui permettent des utilisations temps-réel pour la voiture autonome. On peut trouver des vidéos sur le sujet. Si les RNP sont très longs à entraîner une fois cette étape passée, la réponse à un nouveau stimulus est très rapide.

Maintenant, dans le cas de la voiture autonome, il y a **des cas rares de plantages**: “l’enfant qui court après son ballon et qui surgit sur la chaussée”. Les industriels se doivent de les prendre en compte. Typiquement ce sont des cas qui n’ont pas fait partie des vidéos d’entraînement. Il faut trouver une stratégie: **il va falloir structurer le problème**. Non seulement en structurant la base de données d’entraînement, mais aussi l’architecture du réseau (ex. le découper en parties spécialisées).

Donc, il ne suffit pas de fournir à n’importe quel RNP des données en croisant les doigts pour que cela converge au bout de 3 jours.

2.2 Reconnaissance de la parole

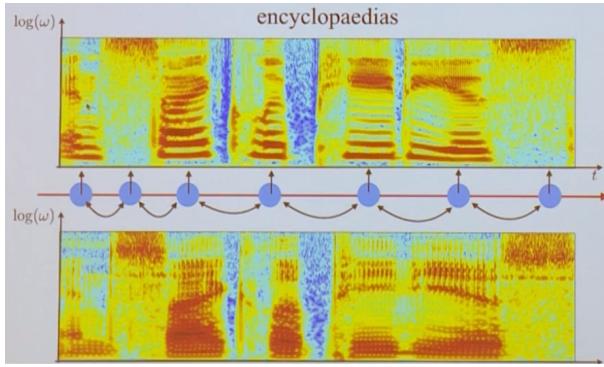


FIGURE 16 – Exemples de deux spectrogrammes de deux locuteurs anglais (une femme pour celui du haut et un homme pour celui du bas) enregistrés au cours de la prononciation du mot "encyclopedias". Une analyse par chaîne de Markov va mettre en relation des phonèmes qui correspondent à l'état de la chaîne.

2.2.1 Approche classique

C'est un domaine qui date des débuts des années 60, et où les techniques employées ont assez peu évolué jusque dans les années 2010. Typiquement, on commençait par prendre le spectrogramme du signal temporel comme sur la figure 16. Dans le spectrogramme du haut, on remarque les phonèmes (du mot encyclopaedias prononcé en anglais) avec des structures " fondamentales + harmoniques", et d'autres qui ne sont pas "voisés" comme le "cy". Donc, localement en temps, on tente la reconnaissance d'un phonème (ou son élémentaire) que l'on représente par un état d'une **chaîne de Markov** (blobs bleus ci-dessus dans la figure 16). Pour apprendre ces états, on fait un apprentissage par Mélange de Gaussiennes (Gaussian Mixture). Puis, on optimise cette chaîne de Markov en prenant en compte la temporalité de l'apparition des phonèmes: cf. il y a des successions de phonèmes plus probables que d'autres selon le corpus lexical de la langue du locuteur. Donc, **on optimise les probabilités de transition** (flèches courbes) entre un son à $t = i$ et un autre son au temps suivant $t = i + 1$.

La technique employée est un algorithme de programmation dynamique (**Viterbi**). En bref, cet algorithme date de 1967 et permet de corriger les erreurs dans un canal bruité. Si n est la taille du message bruité et a le nombre de possibilités par lettre alors, l'arbre des possibilités croît en a^n . A. Viterbi a trouvé une astuce pour simplifier l'arbre au fur et à mesure de sa construction. **Cette méthode est le cadre dans lequel les gens ont travaillé**

durant une cinquantaine d'années et le domaine était devenu un problème industriel et plus tellement celui de la recherche académique.

Notons au passage que les deux spectrogrammes de la figure 16 sont en fait le même mot (encyclopedias) prononcé par un locuteur féminin (haut) et un locuteur masculin (bas). Donc, on voit bien la difficulté de la reconnaissance vocale, car la diversité des voix est très grande et donne des spectrogrammes très différents.

2.2.2 La révolution (très récentes) des RNp

Vers 2013-14, la méthodologie change. On se rend compte que les premières étapes (Mixture gaussienne, chaîne de Markov) peuvent être remplacées par des RN et petit-à-petit toutes les étapes ont été introduites dans des architectures de RNp. Les applications mobiles sont pourvues de ces types de RNp embarqués dans les chips. La structure est la même que celle pour l'analyse d'images, c'est-à-dire **des cascades de convolutions, sous-échantillonnages**. La différence est que les filtres travaillent non plus dans l'espace d'une image, mais dans celui de **séries temporelles** où la notion de **causalité** est aussi un ingrédient important.

L'évolution a été brutale, et cela continue. Un des problèmes classiques par exemple est **la séparation de sources**. Ces développements sont importants, en particulier si l'on considère les appareils auditifs pour malentendants. En effet, si l'appareil se contente d'augmenter la puissance sonore, non seulement il augmente le bruit global, mais surtout ne permet pas de mieux cerner ce que l'interlocuteur vous dit.

Un humain “normalement constitué” avec ses deux oreilles et fixant son attention sur l'interlocuteur fait naturellement de la séparation de composantes et de l'amplification selective. Pour le cas de la figure 17, le problème est au départ un peu plus complexe, car il n'y qu'un seul micro et il y a un mélange des deux locuteurs. En fait, les méthodes mises au point avant 2018 marchaient plus ou moins avec des techniques pourtant très sophistiquées de parcimonies et de séparation comme la **Non-negative matrix factorisation** ... Le problème est que le spectrogramme de son recueilli est un mélange comme on peut s'en rendre compte par exemple avec 2 locuteurs (rouge et bleu) de la figure 17. Idéalement chaque point dans le plan **temps-fréquence** (ici codée sur 256 canaux) est associé soit au locuteur “bleu” soit au locuteur “rouge”, comme dans un problème de **classification**. Mais

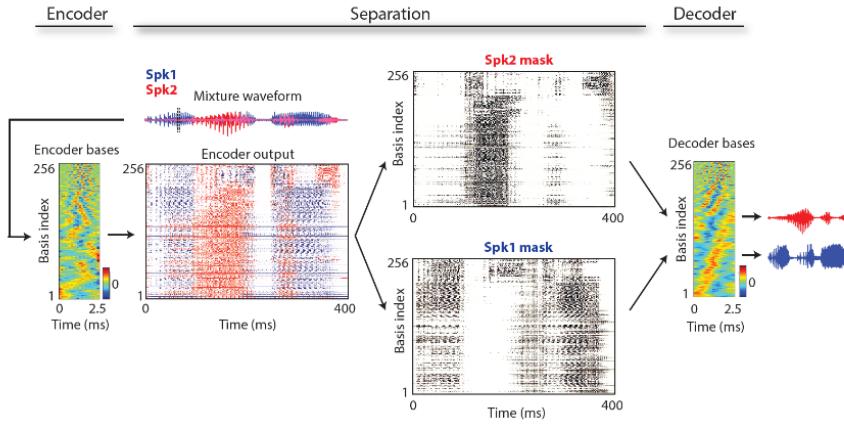


FIGURE 17 – Illustration de la façon dont on peut procéder pour séparer le mélange du son de deux locuteurs.

on ne connaît ni le nombre de locuteurs ni les spectrogrammes typiques de chacun(e)s. Si par contre, on sait faire cette séparation entre locuteurs et avec un peu d'information sur la phase alors, on peut reconstruire les voix de chacun comme sur la figure 17.

Cependant, l'article de Yi Luo et Nima Mesgarani de Sept 2018 (arXiv:1809.07454v2) décrit l'architecture d'un réseau CNN (figure 18). Les auteurs ont pu montrer que **le réseau convolutif est bien supérieur aux méthodes de séparation de mélanges de spectrogrammes de chaque locuteur dans l'espace Temps-Fréquence**¹². Dans ce réseau, en fait les "masques" des locuteurs sont appris en même temps que les sons qu'ils prononcent. On est dans le même schéma de ce que font les réseaux de neurones profonds: à savoir apprendre la représentation en même temps que la classification. Notons que S. Mallat ne pensait pas qu'on arriverait à ce niveau de qualité d'écoute.

Bon, on comprend l'algorithme certes, mais savoir quelles sont les structures qui ont été capturées, ce n'est pas encore compris. On a des idées basées sur les méthodes antérieures qui essayaient de prendre en compte le timbre, les rigidités fréquentielles inhérentes à un locuteur particulier, mais en utilisant ces spécificités personne n'a été capable et de très loin, d'arriver au niveau des RNP.

Notons que les auteurs ont également essayé de se passer totalement de spectrogrammes, c'est-à-dire, ils ont employé les enregistrements bruts. Ils ont constaté que les

12. Un réseau LSTM avait été mis au point par les auteurs, mais celui de la figure 18 est encore meilleur

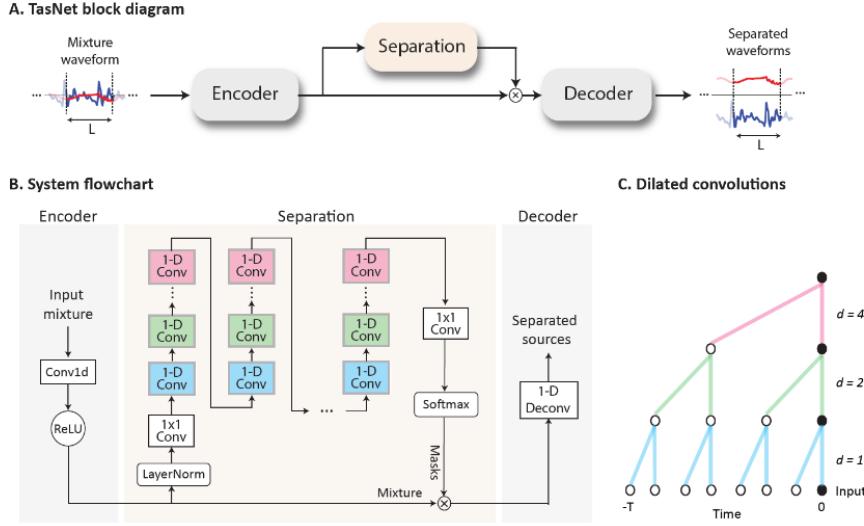


FIGURE 18 – Architecture CNN de arXiv:1809.07454v2 pour résoudre le problème de la séparation des voix de plusieurs locuteurs parlant en même temps. Voici la légende du papier: (A): the block diagram of the TasNet system. An encoder maps a segment of the mixture waveform to a high-dimensional representation and a separation module calculates a multiplicative function (i.e., a mask) for each of the target sources. A decoder reconstructs the source waveforms from the masked features. (B): A flowchart of the proposed system. A 1-D convolutional autoencoder models the waveforms and a dilated convolutional separation module estimates the masks based on the nonnegative encoder output. (C): An example of causal dilated convolution with three kernels of size 2

premières couches du réseau reproduisaient l'équivalent d'un spectrogramme. Mais, au-delà de ces premières couches, on a beaucoup de mal à se représenter ce que fait ou ce qu'apprend le RNp.

2.3 Le traitement du langage naturel

2.3.1 Vision du problème antérieure à 1990

Il s'agit d'un domaine qui couvre: la traduction, l'analyse de texte, la conversation entre personnes, la réponse à des questions, les commentaires descriptifs.... C'est-à-dire tout ce qui touche à la **linguistique**. Avant de prendre un point de vue mathématique/algorithmique sur le sujet, ayons un regard totalement différent qui nous apporte un éclairage sur les notions qui somme toute regardent la façon dont le langage a émergé et comment il est structuré. On pourra ainsi avoir un regard sur les algorithmes "armé" de ces connaissances.

Les premières grammaires datent du 16e siècle et notons que la *Grammaire générale de Port Royal*¹³ publiée en 1660 est inspirée des *Règles pour la direction de l'esprit* de R. Descartes. La linguistique quant à elle date du XXe siècle avec Ferdinand de Saussure (1857-1913) précurseur du **structuralisme en linguistique**, et l'apparition de la science des systèmes de signes de communication (la **sémiotique** ou sémiologie) laquelle est inventée par Charles Sanders Peirce à la même époque (1839-1914). **Le structuralisme** (par ex. Claude Lévi-Strauss en anthropologie, Roland Barthes en littérature, Jacques Lacan en psychanalyse, Michel Foucault et Louis Althusser en philosophie...) va distinguer le *signifiant* (ce qui supporte le signe: ex. le son) et le *signifié* (ex. l'idée, le concept sous-jacent), et tente d'analyser ces *signes comme un système et tente de mettre en évidence des structures*¹⁴. **Donc on a une succession (hiérarchique) de structures:** sons, phonèmes, morphèmes, morphologie, syntaxe... Un exemple: *chanteurs* peut être découpé en *chant* (l'action), *eur* (celui qui fait) et *s* porteur du sens pluriel; *chant* est plutôt du champ lexical, tandis que le *s* du champ grammatical. Toutes ces notions ont émergé et ont été guidées par l'empirisme en segmentant et structurant des discours. **Cette approche**

13. son titre complet: *Grammaire générale et raisonnée contenant les fondements de l'art de parler, expliqués d'une manière claire et naturelle*

14. d'ailleurs en faisant à l'extrême disparaître les locuteurs/auteurs, mais ceci est un autre débat

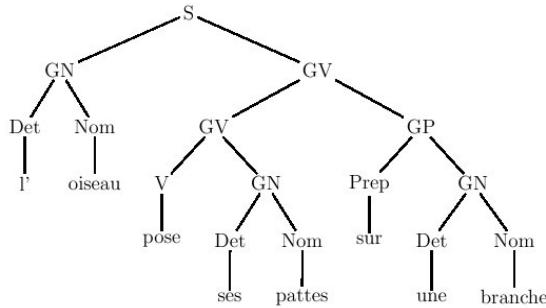


FIGURE 19 – Analyse par un arbre syntaxique de la phrase: “l’oiseau pose ses pattes sur une branche”.

structuraliste a dominé la linguistique jusque dans les années 60, mais elle a imprégné d’autres champs disciplinaires comme le traitement d’images (découpage en structures élémentaires, contours, et relations entre patterns...). Ensuite, le post-structuralisme des années 60-70 (ex. Jacques Derrida, Michel Foucault, Gilles Deleuze, etc) généralise ces notions et décortique les phénomènes sociaux à travers la structure du contexte d’où ils émergent (ex. structure de la langue).

En 1957 émergent **les grammaires formelles de Chomsky** (Noam Chomsky (1928-), le fondateur de la linguistique générative, “Structures Syntaxiques”) c’est-à-dire qu’une **vision plus rationaliste vient remplacer la vision empirique antérieure**. Les notions dégagées par Chomsky ont un lien très profond avec la programmation, les compilateurs, la théorie de la calculabilité (logique mathématique) et le traitement des langages naturels. **L’hypothèse de Chomsky est qu’une partie des structures du langage humain, sa grammaire et syntaxe, est liée à la transmission biologique du code génétique**. Du point de vue de la syntaxe (ou de la grammaire), on peut distinguer des phrases *correctes* de phrases *incorrectes*. Or, ces structures grammaticales sont au cœur, selon Chomsky, de la capacité pour un humain à structurer tout le langage. Donc, la question qui se pose est: quelles sont les structures des grammaires en général et à quel type correspondent les grammaires du *langage naturel*. Il y a selon Chomsky des **structures universelles** (cf. indépendantes de la culture), et il a proposé des classifications de grammaires connues sous le vocable de **Hiérarchie de Chomsky**. Ces développements formels sont entrés en résonance avec l’émergence de l’informatique.

En 1965 paraît le livre intitulé *Aspects de la théorie syntaxique* dans lequel N. Chomsky

propose les structures profondes universelles et les structures périphériques. On y trouve l'apprentissage de la langue comme un développement inné. Une notion importante est la **récursivité** qui permet la génération des phrases, à travers la génération des **arbres syntaxiques** où apparaît une **hiérarchie**. Or, c'est beaucoup trop **rigide**, car si on prend l'arbre syntaxique qui analyse la phrase "l'oiseau pose ses pattes sur une branche" comme sur la figure 19. Les mots "pattes" et "sur" qui se suivent sont "très loin" l'un de l'autre dans l'arbre. Donc, cette méthode a beaucoup de mal à capturer les **composantes horizontales des interactions**. Cependant, cette base théorique a été centrale pour l'étude du langage naturel: traduction, interactions entre locuteurs.

Dans les années 70-90, c'est le développement de la **sémantique formelle** avec comme outil la **logique mathématique** et le **langage théorique formel** (λ -calcul premier langage à définir et formalisé les fonctions récursives). Richard Montague (1930-71) montre en 1970 que **le langage naturel peut être traité comme un langage formel** (d'abord l'anglais puis la généralisation en a suivi). La *connaissance* peut être capturée par un ensemble de *symboles* qui vont se structurer en *propositions*. En informatique, on voit apparaître **les systèmes experts** et le langage **Prologue**. En 1985, les cours d'Intelligence Artificielle étaient basés sur ces concepts et outils. Lorsque l'on était immergé dans le domaine, il était très difficile de penser que cela pouvait être autre chose. En effet, si on s'intéresse à ce qu'est l'intelligence, on pense rapidement au langage, lequel met en jeu des structures linguistiques et l'on se pose la question de savoir comment s'agencent ces structures. Et donc, "c'est évident que cela doit être que ça!"¹⁵ En arrière-plan des développements de cette époque, la **Machine de Turing** définissait le cadre qui "simulait" l'esprit humain. Et, le cadre étant défini, il fallait remplir les "cases".

2.3.2 Vision du problème postérieure à 1990

Nous allons porter maintenant un regard **totalement différent** qui apparaît grossièrement vers 1990 avec le Machine Learning Statistique puis les RNP. Il faut cependant avoir à l'esprit les anciennes notions, car si elles s'effondrent ce n'est pas totalement comme on le verra.

15. NDJE: C'est précisément ce qu'en histoire des sciences, on nomme un paradigme kuhnien, du nom de Thomas Kuhn contemporain de K. Popper, qui a décrit les conditions de révolutions scientifiques entre des phases "normales" où un "paradigme" prévaut.

La raison d'un changement de paradigme a été la confrontation **avec le mur de la complexité**. C'est-à-dire que tant que l'on fait face à des petits problèmes: ex. phrases très simples "John aime Marie", "John va à la soirée avec Marie", etc, l'ancienne méthode marche. Mais s'il y a plusieurs "acteurs" qui interagissent, alors il y a une explosion de la complexité à laquelle les systèmes à petits corpus linguistiques très spécialisés ne peuvent faire face. Un petit détour en philosophie que l'on peut avoir en tête avec les deux pôles que sont **l'Empirisme et le Rationalisme**¹⁶. **L'empirisme**, notion plutôt anglo-saxonne proposée par Francis Bacon (1561-1626), John Lock (1632-1704), David Hume (1711-1776) considère que **la connaissance se fonde sur l'accumulation d'observations et de faits mesurables, dont on peut extraire des lois générales par un raisonnement inductif, allant par conséquent du concret à l'abstrait**. Il s'oppose au **rationalisme** (et l'innéisme) qui pose **la raison discursive comme seule source possible de toute connaissance réelle**. **Le rationalisme** de René Descartes (1556-1650), Gottfried Wilhelm Leibniz (1646-1716)... postule, **l'existence en la raison de principes logiques universels** (principe du tiers exclu, principe de raison suffisante) et d'idées a priori, c'est-à-dire indépendantes de l'expérience et **précedant toute expérience**.

Si l'empirisme tire ses racines de la "**tabula rasa**" d'Aristote (l'image de la cire qui reçoit les impressions), il a eu des prolongations importantes: la philosophie analytique, l'empirisme logique du Cercle de Vienne... et la linguistique. Le rationalisme puise ses sources **chez Platon avec l'exercice des mathématiques qui nous détache des sens; et chez Aristote qui s'appuie sur l'observation concrète de la nature (physis) pour poser par la raison les base de la logique formelle (Organon), de la métaphysique, et de l'éthique**. Toutes ces approches de la connaissance se retrouvent entre autres en Physique et en Mathématique et, **Emmanuel Kant** (1724-1804) critique l'empirisme de Hume, car **pour lui c'est le sujet qui donne ses règles à l'objet pour le connaître** (il y a une forme d'*a priori*). **Il fixe les limites de l'entendement humain qui n'a accès qu'aux phénomènes** (via l'expérience), et **il les filtre par la raison à travers un a priori**. C'est la synthèse kantienne.

L'empirisme va remettre en cause la causalité: on n'a accès qu'à des événements successifs, et la relation causale n'est qu'une relation de succession temporelle, sans garantie

16. NDJE: c'est une partie que j'ai réécrite en discutant avec des philosophes. Bien entendu, ce n'est qu'un résumé.

que la même chose puisse se reproduire¹⁷. **La connaissance ne s'obtient pas par inférence logique, mais plutôt par association.**

Ces deux perspectives sont au cœur de deux natures des mathématiques complètement différentes. **La sémantique formelle antérieure à 1990 est plutôt de type rationnaliste (cf. la logique), tandis que l'approche ML/RNp est plutôt basée sur des notions d'analogie (géométrie, distance).** Qu'on le veuille ou non, nous sommes soumis à des **biais culturels** et si en France, on dira “demain, il va faire beau ou il va neiger”, aux USA, on dira plutôt “il y a 30% de chance pour qu'il fasse beau ou qu'il neige” donc beaucoup plus probabiliste. Nous verrons en quoi cela se manifeste dans le domaine qui nous intéresse dans ce cours.

2.3.3 Traduction automatique

Il y a eu une évolution, dans les années 90, on a une approche statistique avec les chaînes de Markov liées à l'analyse du son. Mais, petit à petit, il y a un **fossé qui s'est creusé** dans le monde de la linguistique: d'un côté, on a des personnes qui font de la **théorie** (structuraliste, grammaires formelles), et de l'autre côté, des personnes qui font de **l'expérience et des statistiques** et mettent en évidence des corrélations dont ils se servent pour faire de la production de phrases. Les derniers n'étant pas bien vus par les premiers...

Les RNp appartiennent à la seconde catégorie. **En 2010, ils ont eu des résultats saisissants**, et ils sont à la base de tous les softwares actuels pour faire de la traduction en ligne (Google Translate, DeepL,...). Il y a des cas où ce genre de traducteur fait des erreurs, mais la syntaxe et la sémantique sont respectées. Il y a toujours besoin d'énormément de données pour faire l'apprentissage. Pourtant, l'architecture a été étendue aux **langues rares!** En fait, on garde les couches les plus internes du réseau appris avec du français et de l'anglais, et on n'entraîne à nouveau que les dernières couches avec la langue cible. Donc, c'est comme si les **couches internes du réseau** capturaient des structures **universelles** (générique à la manière de Chomsky) indépendantes du langage.

En définitive, on voit disparaître le fossé cité plus haut. Car ces **réseaux** empiriquement (statistique) ont été capables de **capturer des structures du langage** (conceptuel). Ces

17. NDJE: a l'extrême la science est impossible

structures ont-elles été apprises à partir des données (d'entraînement) **ou** bien étaient-elles là au départ, c'est-à-dire dans **l'architecture du réseau** qui est un *a priori*? En quelque sorte: “**l’élément inné**” c’est **l’architecture du réseau (rationalisme, innéisme)** et “**l’élément appris**” ce sont les **poids (empirisme, apprentissage par l’expérience)**.

Dans ce débat empirisme/rationalisme, le problème va donc être de comprendre en quoi l’architecture du réseau est fondamentale (rationalisme) ou non, car alors ce serait l’apprentissage qui ferait le boulot (empirisme).

2.3.4 Raconter une vidéo

Au-delà d’une application centrée uniquement sur le langage, on commence à voir des applications **multimodales**: vidéo et description textuelle, mais plus généralement son-image/vidéo-texte-langage. Voir par exemple la vidéo <https://vimeo.com/146492001> qui donne une illustration de NeuralTalk2 de la Google Brain Team de 2016 (un réseau constitué d’un CNN: convolutional NN suivi d’un RNN: récurrent NN).

2.4 La Physique: interaction à N-corps

Rappelons que la Physique jusqu’à très récemment était la “seule” science de la très grande dimension avec énormément d’interaction entre les particules, pensez par exemple à l’Astrophysique, la Dynamique des Fluides, la Chimie Quantique. Or, au fil des siècles, on a pu bâtir un corpus solide d’équations qui régissent ces phénomènes: Newton/Einstein, Boltzmann, Maxwell, Navier-Stokes, Schrödinger/Dirac… On pourrait associer à cette démarche de découverte des interactions entre particules, une approche de type “rationaliste”. On pourrait croire que connaissant ces lois fondamentales de la dynamique, de l’électromagnétisme et autres forces entre particules élémentaires, cela suffirait pour comprendre toute la Physique car “il n’y a plus qu’à dérouler le formalisme”. Or, force est de constater que le problème est extrêmement complexe à cause de sa dimensionnalité. **Fondamentalement, tous les champs cités (Astrophysique, Mécanique des Fluides, Chimie Quantique) sont des problèmes à N-corps.** Les simulations peuvent dans certains cas aider, mais la plupart des cas, il faut faire beaucoup d’approximations!

Mais “empiriquement”, on voit apparaître des **structures**: galaxies, turbulence bidimensionnelle, et des molécules. Donc, plutôt que de partir du “simple” + équations pour

construire du complexe, ne peut-on pas partir d'observations pour capturer les structures complexes. Ce qui peut se formuler par la question suivante: **peut-on prédire une solution par régression à partir d'une base de données de solutions et quelques informations a priori sur le système étudié?** En fait, la réponse est oui!, et encore une fois **avec les RNP ça se fait de mieux en mieux.** Et **à chaque fois** (même pour le langage) on voit que c'est le même type de réseau qui fonctionne, à savoir **les réseaux convolutionnels.** Et par exemple en mécanique quantique, les fonctions d'ondes anti-symétriques des fermions semblent être capturées par ce type de réseau.

Les applications sont très importantes, car faire par exemple des **simulations rapides sur des systèmes complexes** sont au cœur de beaucoup d'industries: ex. l'industrie pharmaceutique pour la fabrication de nouvelle molécule (ex. calculer la stabilité); la physique des matériaux; la modélisation de l'écoulement dynamique autour du profil d'une voiture/avion. Tous ces domaines sont en plein boom depuis 3 ans. Notons que ces applications ne nécessitent pas une prédiction à 10^{-12} près, mais veulent au moins obtenir les comportements de ces solutions.

2.5 Lien avec la Neurophysiologie

Historiquement vers 1950, les neurones informatiques ont émergé de modèles biologiques. Bien entendu, les neurones du cerveau sont beaucoup plus complexes que les neurones formels, de plus dans le cerveau, on sait maintenant que les cellules gliales sont au moins aussi importantes que les neurones. Ceci dit dans le cerveau humain en fonctionnement, on voit apparaître des zones “spécialisées” (**structuration à grande échelle**), et on a mis en évidence la plasticité neuronale (Marian Diamond, 1964) qui vient expliquer la ‘reconfiguration’ (**par apprentissage**) de neurones qui prennent alors les fonctions de neurones déficients.

Quelles sont les opérations (mathématiques) qui sont prises en charge par ces zones spécialisées? Notons que s'il y a un lien entre toutes ces zones, on peut penser aux “briques élémentaires” du cerveau. Ainsi, si les CNN fonctionnent pour la vue “artificielle”, il est peut-être naturel qu'ils fonctionnent aussi pour l'ouïe “artificielle”, car ces 2 fonctions dans le cerveau sont prises en charge certes par 2 zones différentes, mais fondamentalement ces zones sont composées de neurones (et autres cellules). Et on peut généraliser aux sphères du langage, de la mémoire etc.

Dans les cinq dernières années, il y a eu beaucoup de travaux sur les liens entre **neurophysiologie et réseaux de neurones**. Par exemple, la réponse d'un neurone dans l'air secondaire de la vision (arrière du cerveau) à un stimulus d'entrée: peut-on construire un modèle mathématique de ce lien? Pour ce qui concerne le modèle mathématique, on n'y arrive pas, mais alors peut-on construire un réseau de neurone qui rempli la même fonction? En fait, on ne peut pas entraîner de tels réseaux avec des stimuli physiologiques, alors ce qui est fait, c'est que l'on reprend des neurones de réseaux entraîner sur des images d'ImageNet par exemple, puis en diminuant les entrées, on essaye de voir quelles sont les capacités de prédictions de ces neurones artificiels. Ce que l'on constate, c'est qu'ils sont bien supérieurs à tous les modèles élémentaires obtenus avec des modèles linéaires auto-régressifs ou avec une simple non-linéarité, etc.

Il y a donc similarité entre neurones artificiels et neurones physiologiques. On la voit aussi au niveau des **invariants**. Dans les années 1981, on a découvert des groupes de neurones au sein du cortex visuel qui ne sont sensibles qu'à la reconnaissance de simples barres claires sur fond sombre. Ensuite, certains neurones du système nerveux central seraient spécialisés dans la détection de stimuli complexes (théorie du neurone "grand-mère", cf. qui reconnaîtrait le visage de votre grand-mère). Or, ces fonctions ont également été découvertes dans les réseaux de neurones profonds (voir Google Deep Mind de 2012).

Finalement, on ne peut pas éviter d'aborder **le sujet dans toute sa généralité**. Si au niveau mathématique, on jongle avec des propriétés géométriques, statistiques, probabiliste..., on voit bien que ce sujet touche énormément de domaines déjà cités précédemment, mais on peut y ajouter les sciences sociales, sciences humaines, car il y a **par derrière une conception de la connaissance**.

2.6 Apprentissage par renforcement

On a vu qu'il y a **l'apprentissage supervisé** et **l'apprentissage non supervisé**. Il y a un troisième type, c'est **l'apprentissage par renforcement**. Il est différent, car il est basé sur le couple "**essai-erreur**". Au lieu de donner des exemples étiquetés (exemple + réponse connue), on a un système dynamique avec une boucle de rétroaction (récompense si succès). L'idée est qu'il faut adapter au fur et à mesure le système pour optimiser le renforcement (la récompense) pour qu'il soit le plus positif.

Une des applications est l'apprentissage pour les jeux, ex. AlphaGo qui a battu le champion du monde en titre du jeu de Go. Dans ce cas, il faut distinguer dans AlphaGo des RNP convolutionnels (*nb. il y a une invariance par translation locale et il y a des structures multi-échelles*) qui ont été réadaptées d'ailleurs pour jouer aux échecs. Il y a une **forme de créativité** indéniable par rapport à Deep Blue qui avait battu Kasparov. Deep Blue était basé sur une stratégie de "gain" d'une fonction à plusieurs coups de profondeur (fonction d'utilité) par recherche de la solution optimale dans un arbre des décisions. Ce qui contraignait Deep Blue, c'est la programmation de la fonction d'utilité qui vous dit "garder la dame ou le fou dans telle ou telle situation". Par contraste (pour ne pas dire "par contre"), **l'apprentissage par renforcement n'utilise pas du tout ces *a priori***. Mais attention les réseaux n'apprennent pas "à partir de rien", car il y a quand même un *a priori* sous-jacent: c'est leur architecture. Cependant, à part donc l'architecture, on ne dit pas au réseau de ne pas sacrifier la dame par exemple. Et ce que l'on a vu apparaître ce sont de nouvelles stratégies (de sacrifices) aux échecs qui révolutionnent la discipline.

En dehors des aspects "ludiques", dans le **monde industriel, c'est très important par exemple pour entraîner un robot à faire une tâche particulière**: exemple le fameux jeu pour les enfants de mettre les bonnes formes dans les bons trous! En appliquant ce type d'apprentissage, les robots arrivent à partir des essais-erreurs d'accomplir des tâches très compliquées. Voir par exemple: <https://www.youtube.com/watch?v=JCjTQfy0h8w> Et il y a beaucoup d'autres vidéo qui montrent des batteries de robots qui apprennent ensemble...

2.7 Apprentissage non-supervisé

Vraisemblablement, nous n'aurons pas le temps d'aborder ce type d'apprentissage cette année, mais plutôt l'année prochaine, cependant il faut l'avoir en tête. Donc, l'idée est qu'il faut établir **des modèles à partir des données**: ex. à partir de données sur des écoulements fluides, on veut établir les densités de probabilités. Là aussi, il y a des résultats spectaculaires obtenus à partir des RNP.

Prenons la figure 20 sur laquelle on a de gauche à droite: un nuage interstellaire turbulent, un autre type de fluide, un tas de pierres, et des bulles. On prend une image que l'on met à l'entrée d'un réseau convolutionnel entraîné sur ImageNet (animaux, objets

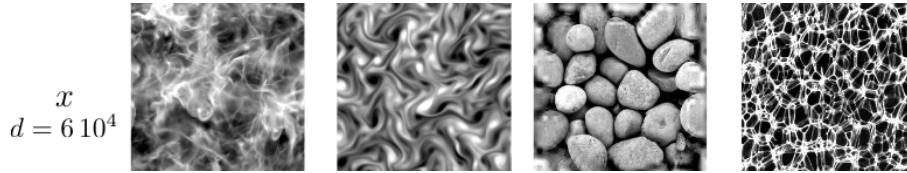


FIGURE 20 – Exemple de différentes textures: un nuage interstellaire turbulent, un autre type de fluide, un tas de pierres, et des bulles.

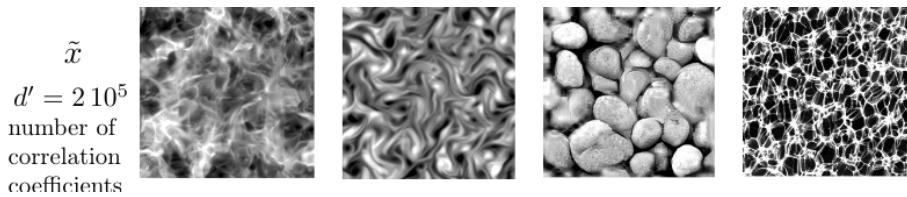


FIGURE 21 – Génération de nouvelles textures à partir de celles de la figure 20.

du quotidien...) sans aucun rapport avec la turbulence. A l'intérieur du réseau, **on peut calculer des coefficients de corrélations entre les sous-images produites par les neurones “stimulés” par cette nouvelle image**. Et à partir des coefficients de corrélation, on peut régénérer de nouvelles images (figure 21) à partir d'un bruit blanc qui petit-à-petit reproduit les matrices de corrélation (**variational auto-encoder**)¹⁸.

Si on avait déterminé des matrices de corrélations des images elles-mêmes pour produire de nouvelles images, on aurait complètement détruit les structures¹⁹. Donc, comment cela se fait que ça marche? La nature de ces corrélations est un domaine actuel de recherche en statistique, car **cela définit de nouveaux processus aléatoires qui ont des propriétés très sophistiquées**.

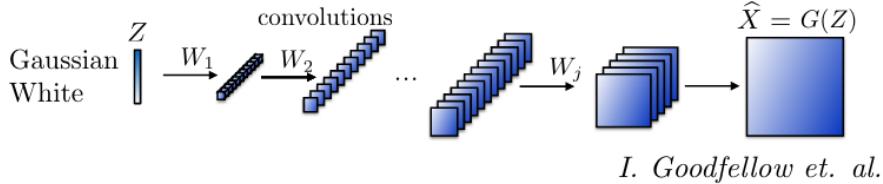
2.8 Generative Adversarial Networks (GAN)

Dans les exemples de la section précédente, les images sont **stationnaires**, c'est-à-dire que les propriétés statistiques locales sont invariantes par translation dans l'image. Peut-

18. **NDJE**. Ici le réseau est entraîné avec ImageNet et les coefficients de corrélations non donc *a priori* aucune information capturant la nature des images de turbulence, tas de cailloux ou ensemble de bulles.

19. **NDJE**: pensons à ce qui se passe quand on produit des cartes CMB à partir des C_ℓ

- Generative network for non-stationary processes:



- Discriminative network:

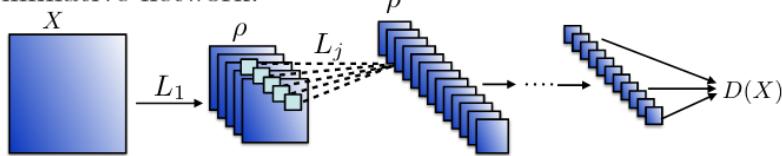


FIGURE 22 – Architecture schématisée des Generative Adversarial Networks.

on faire la même chose sur des images **non stationnaires**? Par exemple si je prends l'image d'un (ou plusieurs) visage, peut-on faire de la **synthèse** de nouveaux visages?

Il y a 6 ans la réponse aurait été NON, on n'était pas capable de faire de la synthèse de turbulence (image stationnaire) alors que les équations de Navier-Stokes datent du milieu du XIXe siècle et que la théorie de Kolmogorov date de 1941, donc la communauté était particulièrement sceptique. La réponse actuelle est OUI: Ian J. Goodfellow et al. dans le groupe de Yoshua Bengio en 2014. Ce sont des réseaux convolutionnels particuliers que l'on appelle des “auto-encodeurs génératifs” qui sont constitués de 2 réseaux (figure 22). Il est clair que si nous n'avions qu'un seul réseau, en lui donnant un bruit blanc ça n'a aucune chance de marcher. **Les deux réseaux agissent en adversaires:** le réseau génératif (G), produit de nouvelles images, et le discriminant (D) fournit la probabilité que l'image qu'on lui présente vienne de la base de données plutôt qu'une image générée.

On utilise une fonction de coût qui va optimiser tous les paramètres des deux réseaux simultanément:

$$\min_G \max_D \{ \mathbb{E}[\log D(X)] + \mathbb{E}[\log(1 - D(G(Z)))] \} \quad (6)$$

Cette fonction dit que le réseau “discriminatif” doit se tromper le moins possible, mais le réseau “génératif” doit le tromper au maximum (c'est l'équivalent d'un minimax two-player game). C'est-à-dire, D est entraîné à maximiser la probabilité de donner une classification correcte d'une image soit de la base de données soit générée par le modèle G ; tandis que G est entraîné à minimiser $\mathbb{E}[\log(1 - D(G(Z)))]$ ou plutôt en pratique à maxi-

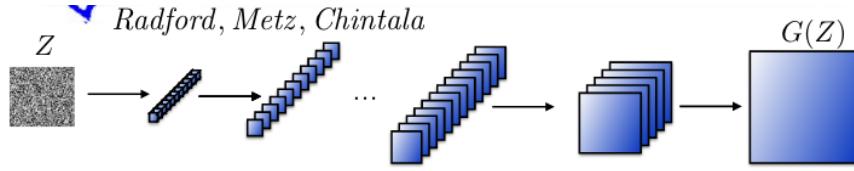


FIGURE 23 – Génération d'une nouvelle image à partir d'un bruit blanc qui passe à travers un GAN entraîné.

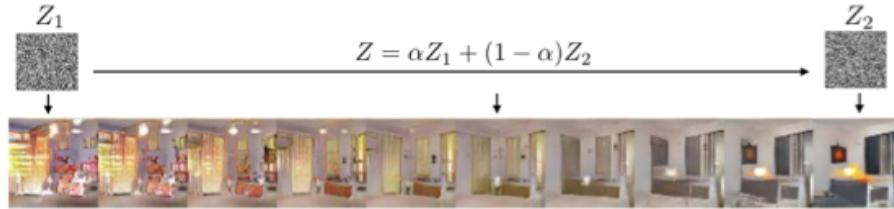


FIGURE 24 – Illustration de la propriété des combinaisons linéaires des GAN.

miser $\mathbb{E}[\log D(G(Z))]$. **A la convergence**, $D(X) = D(G(Z)) = 1/2$ c'est-à-dire que le discriminateur ne fait pas la différence entre une image de base de données et une image générée.

Quelques exemples. Prenons un GAN entraîné sur des images de chambre, on présente donc un vecteur gaussien blanc z d'environ 100 paramètres, ici représenté sous forme d'une petite image, mais ça peut être un vecteur (figure 23). Il va générer en sortie une image de 10^5 à 10^6 pixels $G(z)$. Donc, on présente z_1 et G donne $G(z_1) = g_1$, puis un nouveau z_2 produit $G(z_2) = g_2$. Une propriété intéressante est que si l'on fait des combinaisons linéaires de z_1 et z_2 alors le résultat est une image de chambre résultant du mélange des deux images g_1 et g_2 (figure 24).

On peut faire ce genre d'exercice avec une base de données de visages (des exemples fourmillent sur le Web), etc. À chaque fois, on obtient un G différent: $G_{chambre}$, G_{visage} ... Or, il n'y a aucune information dans le bruit blanc à l'entrée (les z), donc c'est forcément dans les **filtres** des G_i qu'il y a un codage (de l'information) de ce qu'est une chambre à coucher, un visage... **Le générateur a capturé les structures importantes pour reconstituer un visage, une chambre, etc.** Si on se place d'un point de vue rationaliste pour le dire vite, on décomposerait un visage en ses éléments comme la bouche, les yeux, le nez,

et on mettrait en place des relations (grammaire) entre eux. **La question sous-jacente: comment cette structuration est imprégnée dans le réseau?** Finalement, quel **type de mémoire, d'organisation** est incorporée dans ces réseaux? On sait les programmer mais on n'est pas capable actuellement de décrire la façon dont ils fonctionnent.

Notons qu'il y a des applications en art graphique: peindre telle ou telle toile à la façon de tel ou tel peintre, voire interpoler des styles artistiques (voir Gatys, Ecker, Bethdge 2015; <https://arxiv.org/pdf/1508.06576.pdf>). Ne parlons pas de génération "d'œuvres" musicales qui en laisse plus d'un sur leur fin, mais ça se fait.

2.9 Limites et opportunités

2.9.1 Face obscure à éclairer

Actuellement, on est dans une **phase essentiellement empirique**, avec des travaux portant sur les **algorithmes**. On a **besoin d'énorme volume de données labellisées** et souvent ça ne marche pas à cause du manque de données d'apprentissage. **Quelles sont les structures encodées dans les filtres?** En effet, si on entraîne le même réseau avec des valeurs initiales différentes, les poids "individuellement" vont converger vers des valeurs différentes, et pourtant les performances du réseau sont principalement les mêmes! **Donc ce ne sont pas les poids individuels qui comptent mais bien l'ensemble du réseau.** Quelle est la régularité des fonctions apprises?

On **comprend très mal** comment ça marche un réseau, quid de leurs performances, leurs capacités de généralisation et leurs limites. Finalement, **on n'a donc pas de contrôle a priori sur le résultat**. Les seuls contrôles que l'on puisse avoir sont établis à partir de **tests statistiques**. Or, souvent ces tests sont **biaisés**. En effet, reprenons la méthode classique qui partage la base d'entraînement en 3 lots; **training set, cross-validation set** et **test set**. Tout se passe(rait) bien si la base d'entraînement est (était) représentative de l'ensemble des cas de figure que le réseau va devoir traiter. Or, il y a de fortes probabilités que la **base d'entraînement soit biaisée**, qu'on le veuille ou non.

Enfin, les **architectures des réseaux** sont la plupart du temps **optimisées de façon empirique**, et donc c'est long et couteux. On aimerait pouvoir disposer d'un "guide" d'architecte.

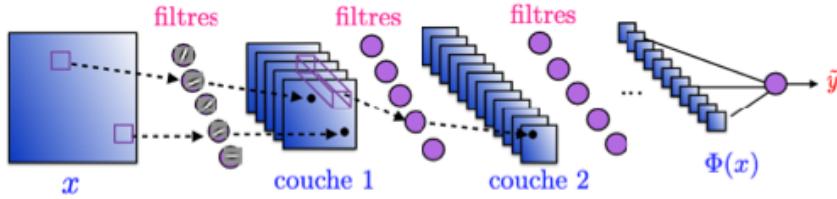


FIGURE 25 – Schéma d'un réseau convolutionnel CNN.

2.9.2 Face qui motive

Le fait que le même type d'architecture puisse donner d'excellents résultats dans des domaines très différents, c'est-à-dire l'aspect **générique** de ces réseaux, est certes une énigme, mais cela participe à l'engouement pour comprendre **le Pourquoi**, et à répondre à la question: quelle forme de *connaissance* est apprise?

Indépendamment de ces questions que certains qualifieront de philosophiques, les perspectives scientifiques et les applications sont considérables. En effet, si on est capable d'apprendre à prédire l'évolution de **systèmes dynamiques à N-corps**, alors on devient capable de faire de la **physique statistique** beaucoup plus fine que celle qui était basée sur l'étude de systèmes désorganisés tels que les gaz, les cristaux. L'enjeu, c'est de faire de la Physique au niveau **mésoscopique** (chimie, biologie).

En math, bien entendu, il y a beaucoup de questions ouvertes, et d'une manière générale il y a beaucoup d'opportunités de recherche: et cela va très vite! Notons qu'il n'y a pas beaucoup de personnes qui travaillent sur les maths en dehors de l'algorithme. Pourquoi?: parce que c'est difficile pard!

3. Point de vue mathématique

3.1 Introduction

Abordons dans ce cours (et les suivants) les questions qui sont soulevées par les réseaux de neurones du point de vue du mathématicien. On va se focaliser sur **l'apprentissage su-**

pervisé. Donc, on va s'intéresser à la structure de la figure 25, où l'entrée x (ici une matrice) passe à travers le réseau où les filtres (W_i) sont des opérateurs linéaires (matrices) associés avec une non-linéarité σ comme un ReLU (rectified logical unit), et par empilement successif, on construit l'architecture du réseau jusqu'à la dernière couche $\Phi(x)$ qui est agrégée pour donner $\tilde{y} = \tilde{f}(x)$ l'approximation de la fonction que l'on veut calculer.

L'ensemble des paramètres du réseau W est constitué par l'ensemble des matrices W_i ainsi que les biais des fonctions non-linéaires, donc pour J couches:

$$W = \{W_i, b_i\}_{i \leq J} \quad (7)$$

Le nombre de paramètres est assez conséquent, à savoir plusieurs millions. Pour chaque jeu de paramètres W , on obtient un \tilde{f} différent, il est donc naturel d'utiliser la notation f_w . Donc, en faisant varier W , on obtient toute une classe de fonctions f_w , notée \mathcal{H}_w :

$$\mathcal{H}_w = \{f_w / \forall W\} \quad (8)$$

\mathcal{H}_w est l'ensemble des fonctions que l'on peut potentiellement programmer avec un réseau de neurones.

Dans tous les exemples que nous avons vus lors des chapitres précédents, ce que l'on cherche, c'est trouver une approximation de f telle que $y = f(x)$, avec $x \in \Omega$ où Ω est par exemple un ensemble d'images structurées, c'est un sous-ensemble de toutes les images possibles de même dimension. On sait que $\Omega \subset \mathbb{R}^d$ avec d la dimensionnalité du problème (figure 26). On ne connaît ni f et *a fortiori* ni les (W_i, b_i) , et on ne connaît pas plus la topologie de Ω ... mais en apprentissage supervisé, on connaît des exemples: $\{x_i, y_i = f(x_i)\}_{i \leq n}$. Donc, on veut approximer \tilde{y} avec une fonction d'un réseau de neurones particulier, c'est-à-dire $f_w = \tilde{y}$.

Ce cadre général étant fixé, comment fait-on pour obtenir le bon réseau de neurones, ou comment déterminer les W_i et b_i ? Rappelons que l'on a une énorme classe de fonctions possibles (cf. \mathcal{H}_w), donc quelle est la bonne stratégie? En fait, on a deux sous-problèmes.

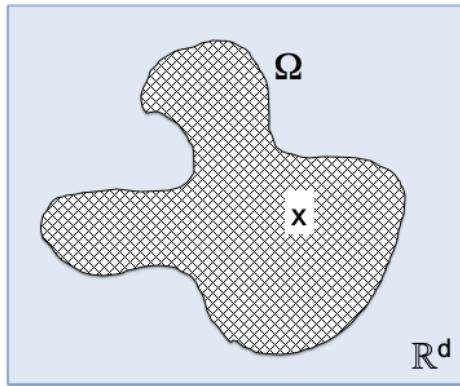


FIGURE 26 – Les images structurées font partie d'un ensemble $\Omega \subset \mathbb{R}^d$.

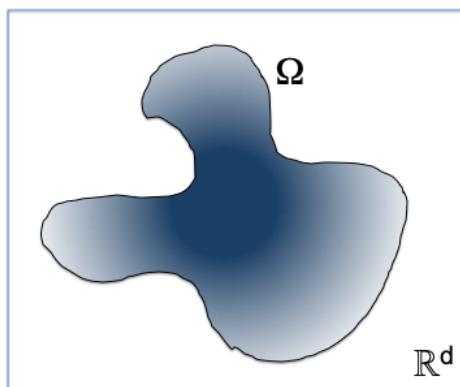


FIGURE 27 – Illustration de la densité de probabilité $p(x)$ d'une image dans l'ensemble Ω .

3.2 Problème d'approximation

Tout d'abord, nous avons **un problème d'approximation**, c'est-à-dire la détermination W^* d'un jeu de paramètres (*nb. même si non écrit il y a les biais aussi*) telle que l'approximation $\tilde{y} = f_{w^*}(x)$ soit la meilleure possible. On définit une notion de risque

$$R(W) = \mathbb{E}_x(r(f(x), f_w(x))) \quad (9)$$

avec $r(y, \tilde{y})$ une fonction de pénalité quand on fait une erreur. Typiquement en Régression et Classification, on utilise les risques suivants (liste non exhaustive)

Régression	Classification
$y \in \mathbb{R}$	$y \in \mathcal{A}$
$(y - \tilde{y})^2$	1 si $y \neq \tilde{y}$, 0 sinon

La solution W^* est idéalement le minimum du risque *en moyenne*, autrement dit

$$W^* = \underset{W}{\operatorname{argmin}} R(W) \quad (10)$$

Cela suppose que l'on connaît **la distribution de probabilité** $p(x)$ de la répartition des données x sur Ω : ex. sur la figure 27, où plus la couleur est sombre plus la densité de probabilité est élevée. À quelle condition l'erreur $R(W^*)$ est petite? C'est-à-dire, on voudrait s'assurer qu'étant donné ε alors

$$R(W^*) \leq \varepsilon \quad (11)$$

Or, en grande dimension ce problème est difficile. Car si l'on prend des hypothèses classiques sur la fonction f comme ayant une forme de régularité sur $x \in \Omega$ comme **lipchitzienne** (cf. la dérivée presque partout bornée) qui donne une régularité **locale**

$$\|f(x) - f(x')\| \leq C\|x - x'\| \quad (12)$$

pour avoir une toute petite erreur, il va falloir **avoir suffisamment d'exemples** qui pavent Ω pour que x soit proche de quelques-uns pour interpoler f (figure 28). Combien faut-il

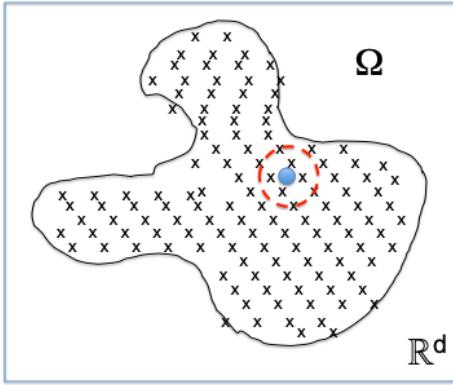


FIGURE 28 – Illustration du problème d'interpolation en grande dimension. On peut ne pas avoir d'échantillons connus proches d'un nouvel échantillon.

exemples? (cf. le cours de l'année 2018) On sait que typiquement

$$n \approx \varepsilon^{-d} \quad (13)$$

et il en sera de même pour le nombre de paramètres qu'il va falloir pour s'adapter sur ces exemples

$$W \approx \varepsilon^{-d} \quad (14)$$

Or, d est très grand! **C'est la fameuse malédiction de la dimension.** Donc, il va falloir avoir des formes de **régularité très fortes** au-delà de lipchitzienne, c'est-à-dire donnant des informations de l'évolution de f sur Ω **très globale**. Alors seulement la connaissance d'un **nombre limité d'échantillons** permettra de contraindre $f(x)$. Or, une régularité forte signifie en d'autres termes que les images x de Ω sont **très structurées**. Que sont ces (nouvelles) régularités?

C'est tout un champ des mathématiques qui est bien compris quand la dimension d est petite (≤ 3), mais en grande dimension, c'est autre chose, et nous l'aborderons par la suite. Mais avant voyons un second problème qui se pose à nous.

3.3 Problème d'estimation et optimisation

Dans la section précédente, on prenait pour acquis que l'on avait *la meilleure solution*, et on se posait la question si cette solution satisfaisait un critère de qualité (minimum de coût/risque). Maintenant, on se pose la question d'obtenir cette solution W^* . Pourquoi est-ce difficile?

Premièrement, il faut calculer l'**espérance** $R(W)$

$$R(W) = \mathbb{E}_x(r(f(x), f_w(x))) \quad (15)$$

$$= \int_{\mathbb{R}^d} r(f(x), f_w(x)) p(x) dx \quad (16)$$

Or, $p(x)$ est **totalement inconnue**, car ce que l'on en connaît, c'est à travers seulement les quelques exemples d'apprentissage. Ce que l'on peut faire alors c'est **calculer un estimateur du risque** $\tilde{R}(W)$. On se place donc dans le domaine des statistiques (empiriques). Mettons que j'ai constitué un réseau de neurones, alors on peut connaître le risque moyen empirique **sur les exemples** utilisés:

$$\tilde{R}(W) = \frac{1}{n} \sum_i r(y_i, f_w(x_i)) \quad (17)$$

et on aimerait que son comportement à grande valeur de n soit tel que

$$\tilde{R}(W) \xrightarrow{n \rightarrow \infty} R(W) \quad (18)$$

À quelle condition ceci est vrai? Il faut **contrôler la variabilité** de l'estimateur empirique; c'est-à-dire par exemple contrôler l'erreur maximum

$$\max_W |R(W) - \tilde{R}(W)| \quad (19)$$

Or, cela va dépendre du nombre de paramètres de W . Au bout du compte ce que l'on veut, c'est que

$$\tilde{R}(W^*) = \min_W \tilde{R}(W) \quad (20)$$

C'est-à-dire que si je mets un exemple x_i à l'entrée du réseau de neurones "optimisé" avec les paramètres W^* , la différence entre y_i et $f_{w^*}(x_i)$ soit la plus petite possible. Mais

plus on va vouloir ajuster un grand nombre de paramètres, plus il faudra d'exemples pour assurer une erreur petite. On retombe un peu dans le problème précédent. Typiquement, $n \sim O(10^{5-6})$ alors que le nombre de paramètres est plutôt $O(10^8)$. Or, si en pratique ça marche, c'est qu'il y a une **forme de régularisation** qui s'opère. En quelque sorte le réseau va s'adapter pour ne fixer/ajuster que les paramètres les plus pertinents pour donner la réponse. Cette notion de **régularisation** est donc l'autre thème des réseaux de neurones que nous verrons, et elle est souvent *implicite*. Ceci dit, on pourrait la fixer *a priori* explicitement par exemple en requérant que $\|W\|$ ne soit pas trop grand en ajoutant au coût une pénalité (voir les régularisations L1 et L2 introduites l'an dernier). Or, même sans mettre de norme, il y a une régularisation qui s'opère. **Il faut absolument comprendre la nature de cette régularisation.**

Maintenant, il faut également **trouver la solution** \tilde{W}^* qui va minimiser le coût empirique. Le cadre mathématique qui était bien connu, était celui de la **minimisation de fonction convexe** où la descente de gradient simple fonctionne parfaitement. Le problème avec les réseaux de neurones est que l'on n'est pas du tout dans ce cas de figure. **On a pléthore de minima locaux.** Donc, *a priori* les méthodes de descente de gradient ne vont pas donner la bonne solution. Or, **ça marche!** On va certes tomber dans des minima locaux, mais la “distance” par rapport au minima global que l'on estime est finalement assez petite: 1) **si on a bien configuré le réseau**, 2) **si on a bien choisi la bonne méthode descente de gradient** et 3) **on a bien choisi les paramètres de l'algorithme**.

Il y a là tout un champ empirique: c'est **l'optimisation non convexe** qui essaye d'obtenir quand même des théorèmes qui permettent de dégager des pistes pour constituer un bon réseau. **Cette optimisation va être centrale dans le cours de cette année**, car au moins si on ne s'intéresse pas aux maths, on aimeraient avoir des idées pour concevoir un réseau et savoir l'entrainer et que faire si ça ne marche pas.

3.4 Autres questions

Si on revient sur le problème d'approximation, on se demande **comment se fait-il que ces architectures** de réseaux, quand bien même on arrive à les optimiser, **pourquoi donnent-elles une bonne réponse?** On essaye de comprendre quelles sont les fonctions $f(x)$ qui peuvent être approximées par un réseau? Par exemple, vous partez d'un problème de

chimie quantique à N-corps qui doit satisfaire une équation de Schrödinger potentiellement très compliquée, pourquoi finalement, on arrive à approximer la solution avec un réseau fusse-t-il à 10^8 paramètres et très peu d'exemples? Mais plus généralement **pourquoi les images, le son, le langage sont structurés ou de quelles structures s'agit-il?** Ce type de question se pose quand on aborde le problème de l'approximation sous la forme: quelle est la **notion de régularité**. Une autre question est **en quoi tous ces problèmes sont-ils similaires?** Y-a-t-il une **notion commune de régularité?**

3.5 Approximation/Régularité, quel type de régularité?

Comme on l'a vu à plusieurs reprises, si on n'a qu'une régularité **locale** (lipchitzienne) **ce n'est pas suffisant**, car on n'est pas en mesure de fournir suffisamment d'échantillons au voisinage d'un x particulier afin d'inférer la valeur de $f(x)$. Donc, il nous faut des notions de régularités **globales** qui font appel aux **symétries**. On va introduire un **opérateur** (à la façon des physiciens...) g qui va déplacer x en x' :

$$x \rightarrow g.x = x' \quad (21)$$

Comment la fonction f va-t-elle réagir à cette transformation? C'est-à-dire au lieu de regarder $|f(x) - f(x')|$ avec x' au voisinage de x , ici, on regarde $f(g.x)$. Or, **si g est une symétrie de f alors**

$$\forall x \in \Omega, f(g.x) = f(x) \quad (22)$$

Remarquons au passage que $|g.x - x|$ n'a pas lieu d'être petit contrairement à la régularité lipschitzienne. Peut-on décrire les propriétés de f à partir de ses symétries? Or, intuitivement plus la fonction a de symétries, plus elle est régulière. Petit rappel: E. Gallois introduit les **symétries des racines d'équations algébriques** de degré d pour montrer qu'on ne peut pas les exprimer sous forme de radicaux dès lors que $d \geq 5$; c'est-à-dire qu'il a pu comprendre la nature des solutions.

Si g_1 et g_2 sont 2 symétries de f , alors

$$f(g_2.(g_1.x)) = f(g_1.x) = f(x) \quad (23)$$

c'est-à-dire que $g_2.g_1$ **est aussi une symétrie de f** . Et donc on peut étudier, l'ensemble

des symétries de f :

$$G_f = \{g/g \text{ symétrie de } f\} \quad (24)$$

qui a une structure de **groupe** dont les propriétés de base sont:

- la composée de 2 éléments est aussi un élément du groupe;
- l'existence d'un élément neutre;
- l'existence d'un inverse pour chaque élément;
- la loi de composition est associative.

3.6 Émergence des symétries et groupes locaux

3.6.1 Du global au local

Voyons dans le cas pratique de la classification d'images, quel genre de symétrie est en jeu et en quoi cela va nous servir de connaître les symétries du problème? Une image x est composée de pixels que l'on repère par u , $x(u)$ est la valeur du pixel. Si j'opère **une translation** (*nb. attention à la notation, mais ça se comprend...*):

$$x(u) \rightarrow g.x(u) = x(u - g) \quad (25)$$

la nature de l'objet contenu dans l'image ne change pas, ce qui impose que

$$f(g.x) = f(x - g) = f(x) \quad (26)$$

Si également, si on opère une **rotation**

$$x(u) \rightarrow g.x(u) = x(R_g.u) \quad (27)$$

dans certains cas cela ne va pas changer le problème (ex. reconnaître un chien, galaxies). Mais parfois cela peut poser des problèmes de reconnaissance (ex. ciel/mer, ou chiffre 6 vs 9). **Donc, l'invariance par rotation va dépendre du problème.**

Maintenant, en quoi connaître le type de symétrie nous aide-t-il? En fait, **on va pouvoir réduire la dimension du problème**. En effet, initialement, $x \in \Omega$ de dimension d énorme; or s'il existe un groupe de symétrie G , on peut **réduire (par quotiantage)** l'espace d'étude à

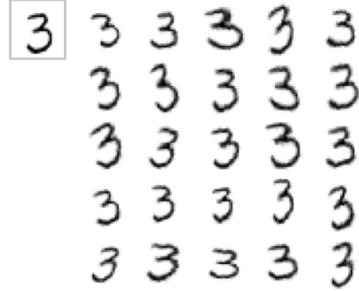


FIGURE 29 – Exemples de déformation du chiffre 3.

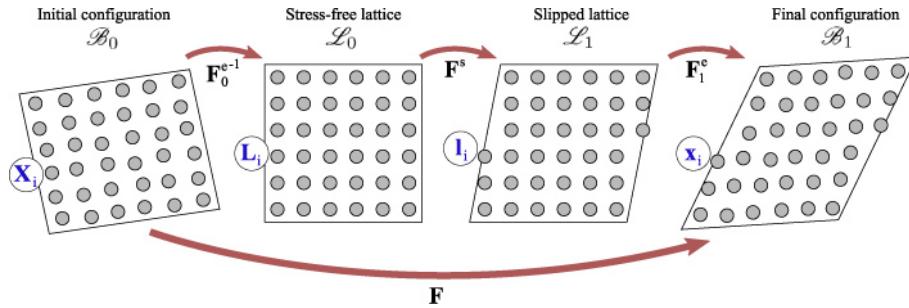


FIGURE 30 – Différents types de déformations que l'on peut opérer sur une image.

l'**ensemble** Ω/G (penser en arithmétique, aux congruences $\mathbb{Z}/p\mathbb{Z}$ qui permettent de réduire l'étude aux restes de la division par p). Pour la translation, on peut donc soustraire 2 paramètres (pour une image 2D), mais si d est très grand, $d - 2$ est toujours très grand! Donc, si on veut réduire sensiblement la dimension du problème, il faut obtenir de **très gros groupes de symétrie**. À quoi peut-on penser? Dans l'analyse des chiffres par exemple, on peut envisager des groupes de **déformations**. Par exemple sur la figure 29, on a différentes distorsions du chiffre 3.

On peut formaliser le résultatat d'une **déformation par une “translation locale”**:

$$x(u) \rightarrow g.x(u) = x(u - g(u)) \quad (28)$$

(c'est un groupe local). Remarquons que l'on utilise toujours la même notation $g.x(u)$ pour dégager le concept de transformation sinon on peut rentrer dans les détails comme sur la figure 30, mais cela alourdirait la notation. On voit alors qu'au lieu de **2 paramètres**

pour définir une translation globale, maintenant le groupe G de **translation locale** va être indexé par **2 fonctions régulières**, donc la **dimension de G** peut être très grande. Au final, si ce type de symétrie est présente dans le problème alors la dimension d s'en trouve réduite de beaucoup. On est passé de **groupes discrets** à des **groupes de Lie** de Marius Sophius Lie (1842-1899), bien connu en Physique Théorique et qui décrivent les régularités sous-jacentes des fonctions $f(x)$.

Cependant, il y a quelque chose à avoir en tête, c'est qu'il y a un élément de **basse dimension derrière** qui permet de s'attaquer au problème à savoir **la dimension de l'espace dans lequel évolue l'index u** . Dans le cas d'une image 2D u indexe la position au sein de l'image, donc on a 2 paramètres indépendants; pour un son, c'est le temps qui indexe soit 1 paramètre; de même pour un texte il y a un ordonnancement des lettres... même en Physique la dimensionnalité sous-jacente n'est que de 4 dimensions (sauf cas contraire). Et cela est **absolument fondamental**, car dans tous les réseaux convolutionnels, les convolutions agissent sur la variable u . Or, découvrir les symétries du problème, c'est faire agir sur la variable u des transformations et regarder comment se comporte f .

Regardons ce qu'il se passe **en audio**. Il est naturel d'étudier les fréquences et les harmoniques (cf. dans le spectrogramme). Or, s'il y a des régularités spectrales, on a envie de faire des **translations en fréquence** qui sont modélisées par

$$x(u) \rightarrow g.x(u) = x(u)e^{ig.u} \quad (29)$$

mais là aussi au lieu de prendre une **transformation globale**, on peut envisager des **transformations locales** de type

$$x(u) \rightarrow g.x(u) = x(u)e^{ig(u).u} \quad (30)$$

En pratique, on se poserait alors la question: peut-on reconnaître un son même s'il est déformé par une vocalise produite par une personne chanteuse? Si on veut s'attaquer à ce problème, alors la fonction doit être invariante par déformation locale dans l'espace des fréquences... Ce que l'on constate à travers cet exemple, c'est que par la connaissance de l'espace dans lequel évolue u et le type de problème posé, on a accès à différents types de symétries. Or, **comment les réseaux de neurones capturent-ils les symétries du problème**, lesquelles appartiennent à des groupes de très grande dimension? De plus, on

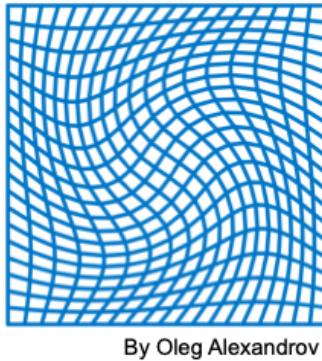


FIGURE 31 – Exemple de difféomorphisme appliqué à un carré.

ne maîtrise pas totalement le sujet du **groupe des déformations en grande dimension**, c'est-à-dire l'étude des **diffeomorphismes** dont un exemple est donné sur la figure 31 pour un carré en 2D.

3.6.2 Quel impact sur le réseau de neurones?

Un réseau fait les transformations successives

$$x \rightarrow \Phi_w(x) \rightarrow f_w(x) = \sigma(\langle \omega, \Phi_w(x) \rangle) \quad (31)$$

avec une transformation linéaire pour passer de $\Phi_w(x)$ à $f_w(x)$, suivi d'une non-linéarité σ . Donc, si l'on sait que le problème possède la symétrie g , alors

$$\Phi_w(g.x) = \Phi_w(x) \Rightarrow f_w(g.x) = f_w(x) \quad (32)$$

On dit que g est une symétrie de f (et de Φ), ce qui est équivalent à dire que f (et Φ) est invariante par action de g .

Ainsi, petit-à-petit le réseau se construit, au fur et à mesure de l'entraînement, une représentation Φ qui a la bonne symétrie. **Expérimentalement, on observe ces invariants dans les réseaux.** D'ailleurs, on les observe également dans les réseaux du cerveau dans les zones supérieures corticales (ex. pour le son, la vue). Et dans le cas du cerveau, on observe des symétries beaucoup plus subtiles puisqu'on peut reconnaître un visage dans

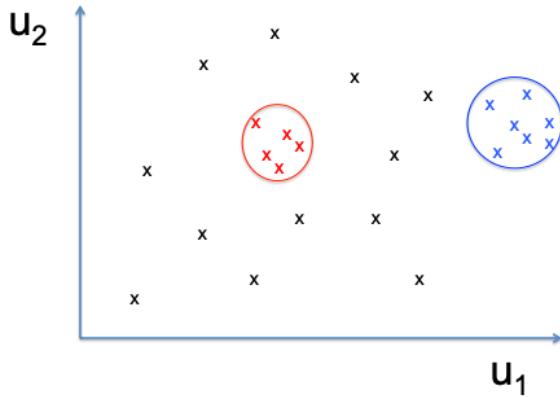


FIGURE 32 – Illustration des interactions *à courte distance* au sein d'un groupe

bien des conditions qui pourraient être qualifiées de “dégradées”. Ainsi, on en revient toujours à la même question: quels types d'invariants sont appris par le réseau? Car si l'on sait concevoir des réseaux ayant des propriétés de symétries *a priori* (ex. symétrie translation \Rightarrow filtre de convolution), comment sont-ils capables aussi d'apprendre d'autres types de symétrie? et qu'est-ce qu'ils apprennent? On sait que ce ne sont pas les paramètres eux-mêmes qui sont porteurs de sens. Il y a des invariants qui font que 2 solutions obtenues par 2 apprentissages différents sont équivalentes. L'étude des symétries peut tenter d'appréhender ces invariants.

3.7 Séparation d'échelles, hiérarchie multiéchelles

Quand on a **beaucoup de variables**, comme par exemple les pixels d'une image, les lettres dans un texte, les atomes en physique, les agents dans un réseau social, et que ce sont **les interactions** entre ces variables qui sont pertinentes pour répondre au problème étudié, ce que l'on constate, c'est que les **acteurs qui interagissent le plus sont ceux qui sont les plus proches les uns des autres** (figure 32).

Dans un réseau social, une personne va interagir avec sa famille, ses amis; dans une image un pixel va interagir avec ses voisins etc, par exemple les croix rouges dans la figure 32. Mais, potentiellement il faut également prendre en compte **les interactions à longue distance** (entre croix rouges et bleues sur la figure 33) furent-elles de bien plus

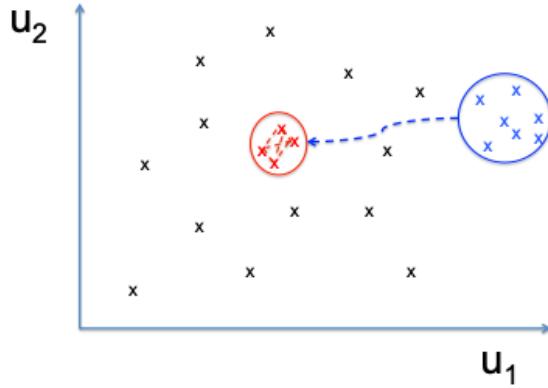


FIGURE 33 – Illustration des interactions *à longue distance* entre groupes.

faible amplitude: dans un réseau social ne faut-il pas considérer l'influence d'un pays lointain pour comprendre le vote des personnes près de chez soi? Dans une image, on sait que le fond par ailleurs peut avoir des composantes à grande échelle; en physique statistique, on connaît aussi l'importance des forces à longue distance dans les verres de spins, etc. Pourquoi ne faut-il pas négliger ces interactions à grande distance? La raison en est que comme on a beaucoup de variables (ex. plusieurs milliards dans un réseau social, etc), en négligeant les interactions avec beaucoup d'agents, on néglige la somme de ces interactions par rapport aux interactions locales, **or elles sont potentiellement de même ordre de grandeur** à cause du nombre élevé d'interactions entre agents à longue distance. Cependant, si à courte distance, on peut prendre en compte toutes les interactions entre plus proches voisins, plus on prend en compte des groupes éloignés, plus on peut considérer dans un premier temps de ne prendre en compte qu'une interaction "moyennée".

Donc, par cette **opération de moyenne** qui prend de plus en plus de variables (agents, particules, pixels,...) au fur et à mesure de l'éloignement, **on passe d'un problème en dimension d** (le nombre d'agents, de particules, de pixels...) **à un problème en dimension $O(\log d)$** . C'est un phénomène de **hiérarchisation, multiéchelles** sur lequel nous reviendrons et qui est très intuitif (citons même des racines dans le *Discours de la Méthode* de Descartes: "*Diviser chacune des difficultés que j'examinerais en autant de parcelles qu'il se pourrait et qu'il serait requis pour les mieux résoudre*"). C'est très important, car potentiellement **on s'affranchit de la malédiction de la grande dimension**.

Convolutional Neural Networks

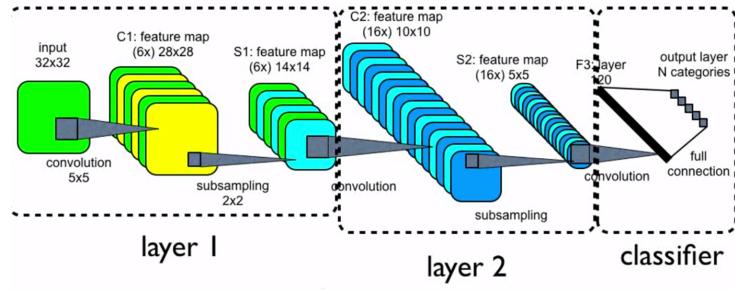


FIGURE 34 – Détail des opérations de convolution et sub-sampling dans un CNN qui font apparaître les interactions à plus ou moins grande distance entre pixels de l'image initiale.

Ceci se matérialise dans les réseaux de neurones convolutionnels comme celui de la figure 34: au fur et à mesure de la progression de l'entrée vers les couches profondes, on a une réduction de l'information (couches de convolution et de sub-sampling), cependant elle devient de plus en plus complexe et agrège des plages de pixels de l'image initiale de plus en plus grandes.

Ainsi, cette hiérarchie des échelles permet de faire une réduction massive du nombre de paramètres du réseau. Cependant, le problème est quand même ardu, car certes, on peut envisager de “moyenner” les interactions au fur et à mesure que l'on considère les grandes échelles. Mais les grands ensembles d'agents (pixels...) loin d'une région considérée ont aussi entre eux des interactions comme illustré sur la figure 35. On peut par exemple considérer un problème de géopolitique où ses propres actions (ex. achats) sont influencées par des pays tiers dont on voudrait globaliser le comportement pays par pays, mais ce sont peut-être les tensions entre ces pays "lointains" qui sont pertinentes pour nos affaires.

Quel outil mathématique va être mis en jeu pour capturer cette hiérarchie d'information? Dans les années 1980-90, il y a un outil, c'est la **Théorie des Ondelettes**²⁰. Notons que la première idée de hiérarchisation qui vient à l'esprit est celle d'une **structure en**

20. NDJE: S. Mallat en avait touché deux mots en 2018 et j'en avais fait une annexe assez détaillée.

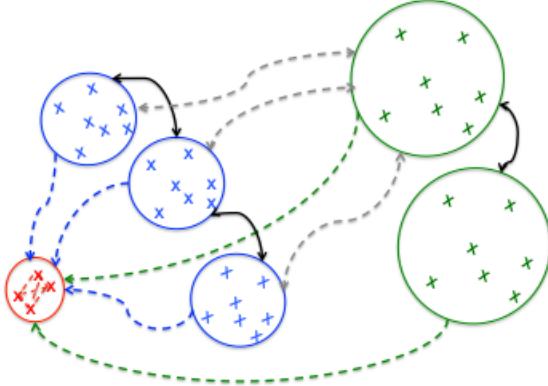


FIGURE 35 – Illustration du rôle des interactions entre groupes à différentes échelles.

arbre: ex. une entreprise qui est déclinée en pôles, services avec en bout de chaîne des humains; on a vu également ces arbres lors des grammaires de Chomsky. Dans ce type de formalisation, on n'a pas vraiment besoin de mathématiques même si on peut développer des algorithmes tels que ceux des *arbres de décision*. Mais cet outil somme toute simple et standard, **ne marche pas** dans notre problème, car même en entreprise, on le sait bien, on a 2 voisins de bureau qui ne se parlent pas. La raison en est que si on remonte la chaîne hiérarchique, ces deux personnes sont susceptibles d'être très éloignées l'une de l'autre. Le problème s'est concrètement posé aussi pour les grammaires de Chomsky quand on passe à l'échelle. **On a envie de mettre une structure horizontale à tous les “noeuds”** pour qu'on puisse échanger des informations, ou autrement dit pour rendre la structure plus souple. Les **Random Forest** sont des arbres de décisions, mais **il y en a beaucoup**, et on en fait des moyennes pour justement rendre compte à la fois de la structure hiérarchique et de la variance à toutes les échelles.

Dans les réseaux convolutionnels, les informations communiquent “naturellement” dans les filtres. **En quoi ces communications sont-elles reliées à la notion de symétrie est un sujet d'étude?**

3.8 La notion de parcimonie

La notion de "parcimonie" concerne en fait tous les algorithmes d'apprentissage, et elle est généralement utilisée en basse dimension. On l'a souvent identifiée à **la reconnaissance de formes** comme s'il y avait quelques **formes élémentaires** de base qu'il fallait reconnaître pour appréhender le problème. Or, intuitivement ceci nous fait sens: dans un visage il y a des yeux, un nez, une bouche; dans un texte, il y a des mots, des phrases, des paragraphes... On a donc envie de mettre en évidence des **patterns** qui structurent l'ensemble d'un visage, texte, etc. Cependant, selon le problème, il peut y avoir beaucoup de formes élémentaires que l'on a sans doute envie de stocker selon des **familles**. Mathématiquement, l'idée de décomposer une fonction selon des familles particulières est du ressort de la **Théorie de l'approximation**. On définit alors une **base** \mathcal{B} qui comprend un ensemble très grand de vecteurs orthogonaux ($N \sim e^d$)

$$\mathcal{B} = \{g_n\}_{n \leq N} \quad (33)$$

Ainsi, toute fonction se décompose selon

$$f(x) = \sum_{k=1}^N \alpha_k g_k(x) \quad (34)$$

Cependant, il est judicieux de faire apparaître ces **patterns/features** qui ne sont pas forcément orthogonaux, car potentiellement redondants, mais qui représentent mieux la fonction f . Alors, on prend un sous-ensemble des coefficients qui malgré tout donne une bonne approximation:

$$f(x) \approx \sum_{k \in I} \alpha_k g_k(x) \quad (35)$$

Or, en ne prenant qu'un sous-ensemble des coefficients, cela veut dire que les autres sont mis à 0. Donc, trouver une représentation **parcimonieuse** signifie trouver une représentation (base) dans laquelle **la plupart des coefficients sont nuls**, et les quelques-uns **non nuls sont les patterns élémentaires**.

Un outil auquel on pense naturellement dans ce domaine de parcimonie est **la base de Fourier** $e^{im.x}$ dans laquelle on décompose la fonction f . En effet, si **la fonction f est extrêmement régulière** (espace de **Sobolev**) alors les basses fréquences sont bien représentatives, tandis que **les hautes fréquences** seront peu présentes, voire **nulles**. Maintenant, s'il y a des **discontinuités**, il va falloir adapter la base. D'une manière générale, il faut

trouver des **dictionnaires** (pas forcément linéairement indépendants, on accepte n'importe quel pattern)

$$\mathcal{D} = \{g_n\}_n \quad (36)$$

et la représentation sera dite **parcimonieuse** si on peut approximer la fonction f par f_M avec un nombre restreint M de coefficients:

$$f_M(x) = \sum_{k \in I_M} \alpha_k g_k(x) \quad \text{et, } \text{card}(I_M) = M \quad (37)$$

On aimerait que l'approximation converge vite vers la fonction quand on augmente M , c'est-à-dire

$$\|f - f_M\| \leq CM^{-\alpha} \quad (38)$$

La parcimonie, c'est donc trouver une base, une famille, un dictionnaire qui permet ce type d'approximation en réduisant le nombre de coefficients.

Ce type de schéma, **on arrive à le mettre en œuvre en basse dimension**. Ici, on parle de la dimensionnalité de u , cf. dans une image x , on indexe les pixels par 2 paramètres si on veut garder la structuration ligne, colonne (u_1, u_2) , et la valeur du pixel est $x(u)$. Ainsi, dans le cas d'une image (idem pour le son), on a développé des algorithmes performants de compression. Mais dans le cas d'une fonction f à très grand nombre de paramètres, par exemple $f(x(u))$ comme fonction de x cette fois, alors **le nombre d'éléments dans le dictionnaire va croître exponentiellement**.

A priori, il va falloir donc **utiliser d'autres outils dès le départ**. On sent que malgré tout qu'il y a certes un aspect de très grande dimension dans certains problèmes (cf. chimie, physique...) mais d'une certaine manière si l'on travaille avec des molécules d'un certain type dont on connaît les interactions, on peut oublier l'aspect quantique et l'équation de Schrödinger. Ce sont des concepts qui sont à l'œuvre dans la théorie classique de l'approximation. La question est alors: **comment et où apparaissent ces structures dans les réseaux de neurones?** Dans les publications (récentes) sur les réseaux de neurones, les auteurs essayent **de faire apparaître ces patterns dans les filtres**. Par exemple en reconnaissance de chiffre, tel filtre aura capturé telle forme, coin...

Donc, il devient naturel dans une première approche d'un problème de se poser la question des patterns. **Mais ce n'est pas suffisant**, en effet, on ne peut pas capturer des phénomènes très complexes en grande dimension qui font émerger des propriétés/structures

nouvelles. Pensons, aux textures de bois, de pierres, de tissus etc, si l'on veut rendre compte de l'ensemble des différents types de texture que l'on va rencontrer dans la nature, il y en a une foultitude. Et pourtant l'humain capte très facilement les différentes textures dans son environnement.

Il faut donc avoir vraisemblablement une vision de la globalité. Ce qui se traduit par la **compréhension de la structure de l'ensemble** Ω dans lequel évolue x . Typiquement, c'est ce qu'on regarde en apprentissage non supervisé. Par exemple, si on travaille avec des molécules organiques (Ω) et que l'on commence à comprendre comment elles interagissent, puisque l'on entraîne un réseau avec ces molécules, il est clair que si on lui fournit une molécule d'un autre type (cristal ionique) ça ne va pas *a priori* bien se passer! Moralité, **le réseau va totalement se spécialiser sur les éléments de Ω** . Là, on sent bien la différence entre ce type d'apprentissage, et celui qui consiste à comprendre l'équation de Schrödinger, ou autre équation de la Physique qui fonctionne en dehors de Ω . En définitive, même en très grande dimension, en se restreignant à Ω on simplifie (un peu) le problème.

3.9 Bilan sur le point de vue mathématique

Ce qui a été décrit dans les sections précédentes est plutôt un programme de recherche. Dans la suite, on va aborder l'aspect algorithmique, optimisation, estimation. Quand on étudie un réseau à 1 couche cachée (dit à 2 couches) la théorie de l'approximation est bien comprise, mais au-delà, il y a une explosion de la complexité, et on a beaucoup plus de mal à comprendre ce qu'il se passe.

4. D'où viennent les idées des réseaux de neurones?

4.1 La cybernétique

L'histoire commence avec le mathématicien américain **Norbert Wiener** (1894-1964) dont les idées sont exposées dans son livre *Cybernetics or Control and Communication in the Animal and the Machine* (1948) qui a eu un retentissement bien au-delà du monde scientifique. Le père de Wiener était un russe immigré aux États-Unis, et semble-t-il, avait

une vision de l'enfant comme une pâte à modeler totalement malléable, et donc il avait comme projet de faire de son fils un génie. Avec un ami, également russe immigré qui partageait ses opinions éducatives, ils ont parcouru l'Europe pour donner à leurs enfants une éducation historique, littéraire, etc, tout en les abreuvant de sciences à la maison. Il se trouve que Norbert a développé une mémoire prodigieuse suite à des problèmes de vision qui le faisait se concentrer sur son audition. Et tout compte fait le petit Norbert et son ami sont devenus des petits génies... Les deux enfants ont eu leur doctorat vers 18 ans à Harvard, en logique mathématique pour Norbert. Cependant, son ami a craqué et a fini par se suicider.

Norbert quant à lui a créé et révolutionné: le domaine du traitement du signal (ex. le filtrage éponyme), le domaine de l'automatique et du contrôle (ex. il a introduit les boucles de *feedback* lesquelles étaient motivées par la poursuite des avions par la DCA), le mouvement brownien est une "mesure de Wiener", l'analyse de Fourier est revue par les analyses de Wiener... Donc, une créativité extraordinaire qui a largement dépassé le domaine des sciences.

La cybernétique est selon Wiener une « théorie entière de la commande et de la communication, aussi bien chez l'animal que dans la machine ». Dans ce cadre, **l'apprentissage est vu comme un système dynamique**. **L'intelligence** est vue quant à elle, comme une **adaptation au réel**, et cette adaptation ne peut pas être pensée sans **penser au temps**. Ainsi, on a un système qui évolue dans le temps et qui doit s'adapter. Une conséquence est que **l'on ne modélise pas le monde, mais la façon de réagir par rapport à l'extérieur**. Un exemple qui est devenu classique est celui *du bateau qui rentre au port* avec des conditions extérieures telles que: le vent, les courants, les vagues. Deux approches s'offrent alors: **soit on modélise tout** l'environnement extérieur et le fonctionnement du bateau lui-même, **soit on se fixe uniquement sur le cap et la vitesse du bateau** qu'il faut adapter pour remplir l'objectif. Sur la figure 36, les différentes phases d'une boucle d'asservissement (contrôle) sont schématisées. La commande est à prendre au sens large, c'est sûr quoi le marin peut agir sur son bateau (cap, vitesse), la boîte noire est la partie "non modélisable" (l'environnement extérieur à très grand nombre de paramètres) ou plutôt que l'on ne veut pas modéliser, le/les senseur(s) mesure(nt) l'état de la position du bateau et la réinjecte dans un comparateur (+/-) pour mesurer l'écart à l'objectif et induire une nouvelle commande si besoin.

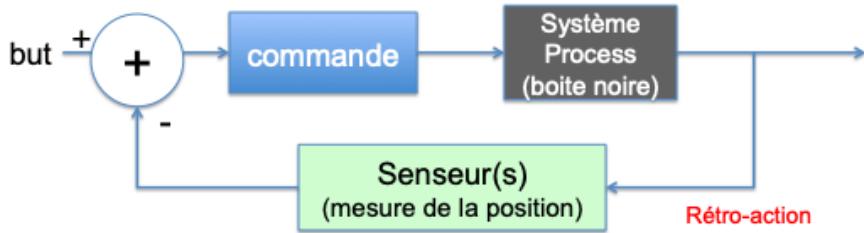


FIGURE 36 – Les différentes phases d'une boucle d'asservissement pour arriver à bon port.

Ce système très général est celui de la **rétroaction négative** où l'on agit en fonction de **l'erreur**. Cette idée a été très développée non seulement en ingénierie, mais aussi en science sociale et biologie. Il s'en est suivi le développement d'un projet de **robot cognitif** dont le fonctionnement (caricatural) est schématisé sur la figure 37. Le "but" ou "l'objectif" a réalisé, c'est une entrée de la partie "Planificateur" du système cognitif (version primitive du cerveau) qui comporte également une "Représentation" du Monde et un module de "Perception" pour interagir avec lui. Les gens ont beaucoup discuté ce schéma et on essayait de voir comment cela pouvait être implémenté dans des systèmes informatiques (et mécaniques). Cela a été fondamental pour le domaine du "**contrôle-commande**".

Les développements de la cybernétique ont été faits dans les années 1940-60. Il y a eu pas mal d'analyse du schéma (figure 36) par rapport à ce que l'on pouvait rencontrer dans la nature. Un article de **Herbert Simon "The Architecture of Complexity"** (1962) met en évidence que l'apprentissage est effectif quand **le monde est structuré** et pas trop complexe. La complexité va définir par exemple le nombre d'échantillons d'apprentissage, mais d'une manière générale va donner une limite à apprendre le monde. H. Simon montre qu'il y a également une notion **de hiérarchie** omniprésente que l'on observe (*réductionnisme*) en Physique (particule, molécule, atome..), en Biologie (cellule, tissu, organe,...), dans le domaine symbolique (ex. langage: lettre, phonème, mot, phrase, paragraphe, chapitre...), mais aussi en Histoire avec l'évolution des états (tribus, village, ville, région, état, empire...), et généralement des résolutions de problème de mathématique (hiérarchie de théorèmes..).

Robot Cognitif (modèle simplifié)

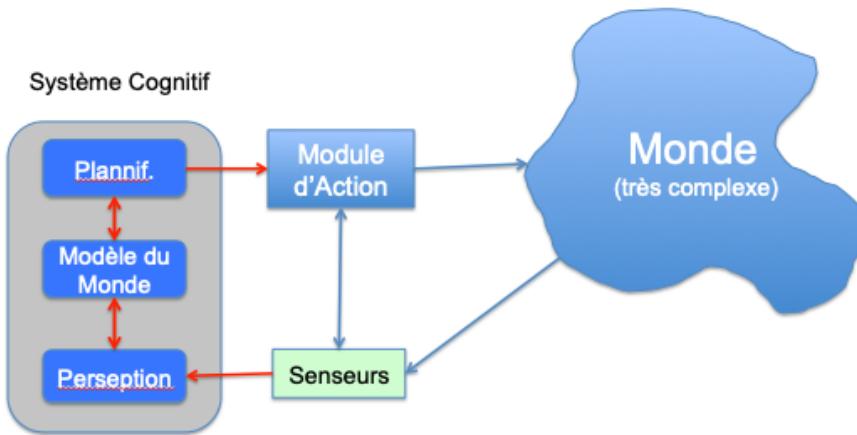


FIGURE 37 – Le projet de "Robot Cognitif".

Une des questions qui émergent est: *Pourquoi observe-t-on cette structuration hiérarchique un peu partout?* La thèse de H. Simon de l'existence de ces hiérarchies, c'est la nécessité d'avoir quelque chose qui soit **stable** tout en étant **adaptatif dans le temps**. Pour illustrer son propos, il prend l'image d'un horloger²¹ qui assemble une montre très complexe lors d'une étape durant laquelle il tient toutes les pièces ensemble, et qui est continuellement interrompu par des appels de clients. Le résultat est que ce pauvre horloger devra recommencer son assemblage très souvent, voire qu'il n'arrivera jamais à assembler une montre! L'horloger va alors sans doute changer de mode opératoire pour assembler des morceaux, puis assembler des sous-ensembles de plus en plus gros et pour finir, assembler les “grosses morceaux” entre eux. S'il est interrompu alors, il aura moins de travail à refaire et convergera vers l'assemblage de la montre. Cette image est celle de l'**évolution** darwinienne qui structure par hiérarchie des éléments stables à toutes les échelles. **H. Simon décrit la complexité à travers la hiérarchie des structures et cette hiérarchie permet de comprendre la complexité.** Cette idée est très importante, car elle est au cœur de tous les **réseaux de neurones profonds** dans lesquels la “profondeur” fait apparaître les **notions de structuration et de hiérarchie**.

²¹. NDJE: La construction de structures gigantesques pour les challenges de type Domino Day illustre également le propos.

La conséquence de la hiérarchie est qu'il y a une organisation en **sous-systèmes** qui sont **quasi séparables** et donc faiblement liés. Cette idée qualitative est assez naturelle et elle n'est pas nouvelle. Pourtant, cette approche est un échec! La véritable question est: comment représenter des interactions faibles sans pour autant les éliminer (voir la section 3.7). Comment représenter ces hiérarchies, comment représenter les différents états à toutes les échelles tout en intégrant la notion de parcimonie pour que l'on puisse "apprendre cette structure". C'est la question qui se retrouve dans tous les champs disciplinaires cités plus haut et à laquelle la communauté pendant très longtemps n'a pas été capable de répondre, rappelez-vous des grammaires de Chomsky. Et elle est au cœur de la compréhension du fonctionnement des réseaux de neurones profonds (RNp).

Donc les idées sous-jacentes des RNp ne sont pas nouvelles, **on en est au stade "algorithmique"**, c'est-à-dire que l'on sait les mettre en œuvre et leur faire apprendre des structures, mais "mathématiquement", on ne comprend pas leur fonctionnement.

4.2 Le Perceptron (1957)

4.2.1 Introduction

C'est l'idée de **Frank Rosenblatt**, un psychologue américain qui invente un réseau à 1 couche capable d'apprentissage²². Notons que Wiener, Chomsky, Rosembatt étaient au MIT, tout comme Marvin Lee Minsky et John McCarthy qui fondèrent le Groupe d'Intelligence Artificielle. Minsky et McCarthy ont développé la partie logique et symbolique.

Le problème posé est une classification $x \in \mathbb{R}^d$

$$f(x) = \begin{cases} -1 \\ 1 \end{cases} \quad (39)$$

et l'on veut pouvoir séparer des lots de x_i en trouvant une *frontière* entre les deux populations comme sur la figure 38. L'équation de la frontière est donnée par

^{22.} NDJE: En 1943 W. Pitts et W. McCulloch introduisent un "neurone formel" qui peut réaliser des fonctions logiques, 4 ans avant la réalisation du premier transistor.

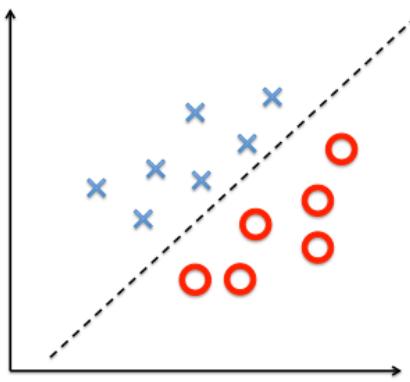


FIGURE 38 – Exemple de classification en 2 dimensions.

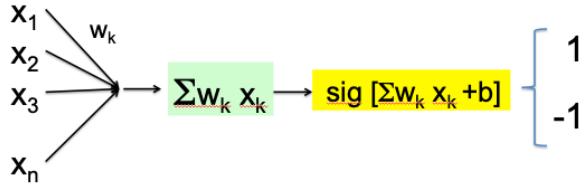


FIGURE 39 – Représentation graphique de ce que doit faire un classificateur linéaire.

$$\langle w, x \rangle + b = 0 \quad \text{et on choisit} \quad \|w\| = 1 \quad (40)$$

avec w la direction orthogonale et b la distance de l'origine des axes à la droite. Les "x" sont de la classe 1 et les "o" sont de la classe -1. On veut connaissant la classe y_i de tous les x_i du lot d'échantillons d'apprentissage que les x_i soient du "bon côté" de la frontière pour que

$$y_i(\langle w, x_i \rangle + b) > 0 \quad (41)$$

Ceci peut se représenter graphiquement selon le schéma figure 39 où la non-linéarité est ici la fonction "signe". La question est de trouver les composantes du vecteur w (cf. les w_k) pour obtenir la séparation de 2 classes, avec un algorithme itératif qui va faire évoluer les poids pour converger vers la solution. Les idées de bases des réseaux de neurones ont été définies déjà à ce stade très simple.

La première idée est d'introduire **un coût (risque) à la mauvaise classification**. Donc, on va prendre tous les points mal classés $i \in \mathcal{M}$ et le risque de Rosenblatt est donné par

$$R(w, b) = \sum_{i \in \mathcal{M}} (-y_i) (\langle w, x_i \rangle + b) \quad (42)$$

Dans l'expression ci-dessus, “ $-y_i$ ” donne la “mauvaise classe” et “ $\langle w, x_i \rangle + b$ ” donne la distance signée du point à la droite, c'est la pénalité à une mauvaise classification. En d'autres termes, plus le point est loin et du mauvais côté de la frontière, plus on le pénalise.

La seconde question est de trouver un algorithme itératif qui minimise $R(w, b)$. Le problème est que **la solution n'est pas unique**. En fait, Rosenblatt s'est posé la question d'en trouver au moins une. On va procéder par une descente de gradient par rapport aux paramètres libres $\Theta = (w, b)$.

4.2.2 L'algorithme de descente de gradient

L'algorithme de descente de gradient peut se décliner ainsi (voir la figure 40 où k indique la k-ième étape du calcul):

- On définit une valeur initiale Θ^0
- Ensuite, on calcule le gradient de la fonction à minimiser

$$(\nabla R(\Theta))_k = \frac{\partial R}{\partial \Theta_k} \quad (43)$$

La dérivée par rapport à une direction unitaire \vec{v} est alors le produit scalaire

$$\frac{\partial R}{\partial \vec{v}} = \vec{v} \cdot \vec{\nabla} R(\Theta) \quad (44)$$

Si on veut maximiser la vitesse de convergence alors \vec{v} doit être colinéaire et opposé au gradient.

En pratique, l'évolution des valeurs de Θ est schématisée comme sur la figure 41 à 1 dimension. Comme d'ailleurs, on peut le voir sur cet exemple, l'algorithme peut se trouver bloqué dans un minimum local, mais ici ce n'est pas le sujet, on verra des améliorations de cette méthode dans les cours par la suite²³⁾.

23. NDJE: nb. en 2018 il y a eu un séminaire dédié à des nouvelles méthodes de Gradient Descent de

	Gradient Standard
0. initialize \mathbf{x}_0	
1. compute $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$	
2. si $\ \nabla f(\mathbf{x}_k)\ \leq \epsilon$, stop	
3. trouver le pas α_k minimisant $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$	
4. update $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$	

FIGURE 40 – Schéma d'une descente de gradient simple.

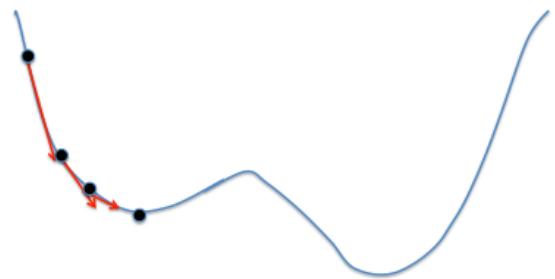


FIGURE 41 – Schématisation d'une descente de gradient. On peut se trouver "coincer" dans un minimum local.

Donc, calculons les gradients, il vient

$$(\nabla R(\Theta))_k = \begin{cases} \sum_{i \in \mathcal{M}} (-y_i) x_{i,k} & \text{pour } w_k \\ \sum_{i \in \mathcal{M}} (-y_i) & \text{pour } b \end{cases} \quad (45)$$

(nb. on peut omettre l'indice k pour une notation vectorielle.) Cependant, on peut soit directement à chaque “pas” de calcul calculer la somme $\sum_{i \in \mathcal{M}}$, soit **visiter les points les uns après les autres**. Cette dernière méthode avait été introduite déjà par Rosenblatt. C'est en fait le **gradient stochastique** qui n'avait pas été formalisée ainsi à l'époque. C'est-à-dire que chaque point mal classé a déjà un coût

$$r_i(w, b) = (-y_i) (\langle w, x_i \rangle + b) \quad (46)$$

avec un gradient

$$\nabla r_i(\Theta) = \begin{cases} (-y_i) x_i \\ (-y_i) \end{cases} \quad (47)$$

et donc **à chaque fois que l'on a une nouvelle donnée on peut faire une mise à jour des paramètres Θ** :

$$\boxed{\Theta^{(s+1)} = \Theta^{(s)} - \alpha^{(s)} \nabla r_i(\Theta)} \quad (48)$$

Ainsi, dans le cas qui nous occupe ici

$$\begin{aligned} w^{(s+1)} &= w^{(s)} + \alpha^{(s)} y_i x_i \\ b^{(s+1)} &= b^{(s)} + \alpha^{(s)} y_i \end{aligned} \quad (49)$$

Les deux algorithmes (standard et stochastique) sont schématiquement comparés dans la figure 42 dans le cas de la recherche d'un minimum en 2D. Une observation connue sous le nom de **Règle de Hebb** (Donald Hebb psychologue et neuropsychologue canadien 1904-1985) en biologie comme en informatique, fait remarquer que **le poids w est augmenté si l'entrée x et la sortie y** qui est aussi l'entrée d'un neurone d'une couche suivante **sont corrélées positivement**. Maintenant la question qui vient à l'esprit est: comment cela

type “Lazy”

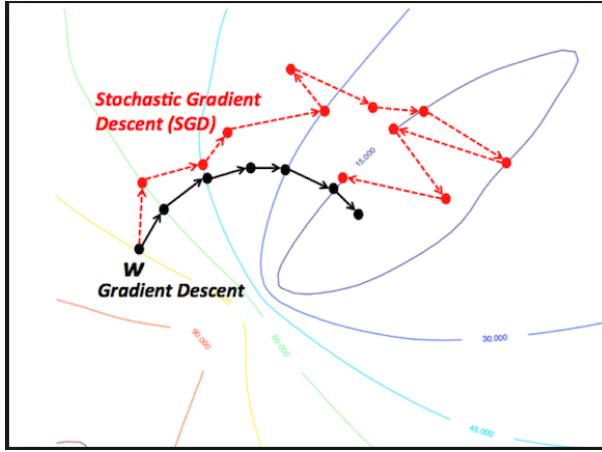


FIGURE 42 – Comparaison schématique d'une descente de gradient globale (ou Batch) et une descente de gradient stochastique.

converge-t-il?

Théorème 1. *Le Perceptron converge vers un hyperplan séparateur si les données sont séparables.*

Un exemple de **non-séparabilité** est la fonction **XOR**. Cela a été remarqué très rapidement pour affaiblir le résultat du Perceptron de Rosenblatt²⁴. Mettons que l'on soit dans un cas séparable, c'est-à-dire qu'il y ait une solution possible, est-ce que le Perceptron converge vers cette solution? Oui... mais.

Démonstration 1. Il existe donc une solution par hypothèse. Notons $x^* = (x, 1)$ et $\Theta = (w, b)$, il vient

$$\exists \Theta_* / \quad \forall i \quad y_i \langle \Theta_*, x_i^* \rangle > 0 \quad (50)$$

On peut normaliser $\|x^*\| = 1$, la condition reste identique. Maintenant, une fois Θ_* connu, je peux le normaliser tel que la condition soit changée en

$$\exists \Theta_* / \quad \min_i (y_i \langle \Theta_*, x_i^* \rangle) = 1 \quad (51)$$

24. NDJE: D'ailleurs dans les années 70, une grande désillusion s'empare de la communauté, car tous les classificateurs linéaires sont concernés (Fisher, Perceptron, etc): une impossibilité à résoudre des problèmes non-linéaires.

Si pour le moment α est constant, alors la règle de modification de Θ devient

$$\Theta^{(s+1)} = \Theta^{(s)} + \alpha y_i x_i^* \quad (52)$$

Alors

$$\|\Theta^{(s+1)} - \Theta_*\|^2 = \|\Theta^{(s)} - \Theta_* + \alpha y_i x_i^*\|^2 \quad (53)$$

$$= \|\Theta^{(s)} - \Theta_*\|^2 + \alpha^2 y_i^2 \|x_i^*\|^2 + 2\alpha y_i \langle (\Theta^{(s)} - \Theta_*), x_i^* \rangle \quad (54)$$

Notons à présent que l'itération se fait sur les échantillons (x_i, y_i) mal classés, c'est-à-dire qu'à l'étape s ,

$$y_i \langle \Theta^{(s)}, x_i^* \rangle < 0 \quad (55)$$

Donc, en utilisant la contrainte sur Θ_* et notant $y_i = \pm 1$ et prenant $\alpha = 1$ (le pas du gradient ou learning rate) pour minimiser la borne sup, alors

$$\|\Theta^{(s+1)} - \Theta_*\|^2 \leq \|\Theta^{(s)} - \Theta_*\|^2 + \alpha^2 y_i^2 - 2\alpha y_i \langle \Theta_*, x_i^* \rangle \quad (56)$$

$$\leq \|\Theta^{(s)} - \Theta_*\|^2 + \alpha^2 y_i^2 - 2\alpha \quad (57)$$

$$\leq \|\Theta^{(s)} - \Theta_*\|^2 + \alpha^2 - 2\alpha \quad (58)$$

$$\leq \|\Theta^{(s)} - \Theta_*\|^2 - 1 \quad (59)$$

Donc, à l'étape $s + 1$, on gagne sur la distance carrée à la limite de 1 unité. Il faut N itérations pour que

$$\|\Theta^{(N)} - \Theta_*\|^2 < 1 \quad (60)$$

avec N donné par

$$N = \lceil \|\Theta^{(0)} - \Theta_*\|^2 \rceil \quad (61)$$

et dans ce cas $\Theta^{(N)}$ définit un plan séparateur. En effet,

$$\forall i \quad y_i \langle \Theta^{(N)}, x_i^* \rangle = y_i \langle \Theta^{(N)} - \Theta_*, x_i^* \rangle + y_i \langle \Theta_*, x_i^* \rangle \quad (62)$$

Or

$$|y_i \langle \Theta^{(N)} - \Theta_*, x_i^* \rangle| = |\langle \Theta^{(N)} - \Theta_*, x_i^* \rangle| \leq \|x_i^*\| \|\Theta^{(N)} - \Theta_*\| \leq 1 \quad (63)$$

et

$$y_i \langle \Theta_*, x_i^* \rangle > 1 \quad (64)$$

donc

$$\forall i \quad y_i \langle \Theta^{(N)}, x_i^* \rangle > 0 \quad (65)$$

qui définit bien un plan séparateur.

Donc, $\Theta^{(N)}$ n'est pas forcément égal à Θ^* car il n'y a pas unicité de la solution à l'hyperplan séparateur, mais c'en est un obtenu en N étapes. Cependant, on n'a pas *a priori* de contrôle sur le choix du plan séparateur final, et comme on le verra ils ne sont pas tous équivalents. De plus, dans le cas non séparable, on peut montrer que les paramètres vont avoir un comportement cyclique, potentiellement très long, et le diagnostic que l'algorithme ne converge pas peut ne pas être évident.

4.2.3 La régularisation

On a mentionné à plusieurs reprises que les réseaux de neurones (sous-entendu *profonds*) ont un nombre considérable de paramètres, et étant donné le relatif faible nombre d'échantillons, on ne devrait pas être capable de leur faire apprendre les poids. Et pourtant, on s'aperçoit que l'on fait converger l'algorithme d'apprentissage vers une solution qui ne fait pas ou peu d'*overfitting*, c'est-à-dire qui est assez robuste à la généralisation. Ceci est dû à la **régularisation**, et c'est exactement ce qui manque à l'algorithme du Perceptron que l'on vient de mettre en œuvre dans la section précédente.

Introduisons la notion importante de **marge** qui va définir la notion de **régularisation** et nous allons voir la correspondance avec la **pénalité** sur les poids. On peut se reporter au cours *Classification par Support Vector Machine* de 2018. L'idée en 1990 de Vladimir Vapnik (1936-) mathématicien russe est la suivante: il faut prendre l'**hyperplan qui est le plus robuste vis-à-vis de petits changements des données d'entraînement** (figure 43). C'est-à-dire prendre l'hyperplan qui est le "plus loin" des deux classes: x_i^+ et x_i^- .

Notons que sur l'hyperplan $\langle w, x \rangle + b = 0$, et en dehors $\langle w, x \rangle + b = c \neq 0$. Et de part et d'autre du plan, on peut trouver les points x_1 et x_2 les plus proches de l'hyperplan des deux classes et tels que l'hyperplan se trouve au milieu (c'est-à-dire que le point $(x_1+x_2)/2$ est sur l'hyperplan) donc si on note $m\|w\|$ la distance de x_1 , x_2 au plan comme ils sont

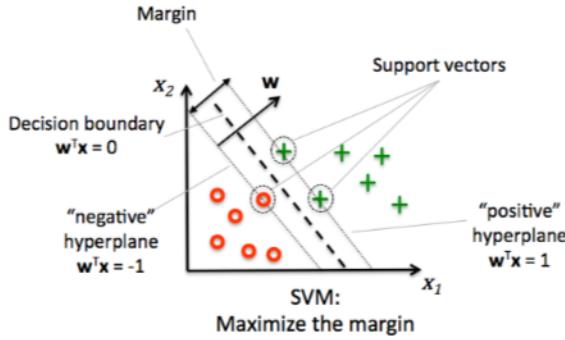


FIGURE 43 – Schématisation de la notion de marge pour trouver l’hyperplan le moins sensible à la variabilité des échantillons près de la frontière.

de part et d'autre, en toute généralité, on a:

$$\begin{cases} \langle w, x_1 \rangle + b = m\|w\| \\ \langle w, x_2 \rangle + b = -m\|w\| \end{cases} \quad (66)$$

Or, on peut rescaler les x tels que la distance entre x_1 et x_2 soit égale à 2, ainsi

$$\boxed{\|w\| = \frac{1}{m}} \quad (67)$$

Donc, **Maximiser la marge “ $2m$ ” (et donc m) revient à minimiser $\|w\|$.** On obtient ainsi, un résultat vraiment important: **quand on minimise la norme des poids du réseau, on a tendance à obtenir un réseau plus robuste à la généralisation.**

4.2.4 SVM: Support Vector Machine

Sachant que w est une combinaison linéaire des x_i , le problème du Perceptron se reformule ainsi: trouver (w, b) tel que

$$\boxed{y_i \times (\langle w, x_i \rangle + b) \geq 1; \text{ et } \min(\|w\|^2)} \quad (68)$$

Durant le cours de 2018, nous avions résolu ce problème constraint par l'introduction d'un lagrangien avec les multiplicateurs de Lagrange α_i

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 + \sum_i \alpha_i(1 - y_i \times (\langle w, x_i \rangle + b)) \quad (69)$$

Il faut résoudre le cas où le problème n'est pas strictement séparable à cause *d'outliers* (cas pathologique de mauvaise classification). C'est résolu par l'introduction d'une pénalité de type Perceptron pour les cas de mauvaise classification. Vapnik propose de définir l'erreur sur la marge comme: de combien faut-il déplacer les points/échantillons mal placés pour les ramener au-delà de la marge pour les reclassifier correctement. Donc, si on note $\xi_i \geq 0$ la distance pour l'échantillon i qu'il faut pour le déplacer alors,

$$y_i \times (\langle w, x_i \rangle + b) \geq 1 - \xi_i \quad (\text{Soft SVM}) \quad (70)$$

et la contrainte devient

$$\min(||w||^2 + C \sum_i \xi_i) \quad (71)$$

Donc, le SVM ressemble au Perceptron, mais la contrainte sur $||w||^2$ change toutes les propriétés du classificateur: il est **plus robuste** et comme la contrainte est convexe, il converge vers une **solution unique**.

Notons que l'on aurait pu choisir une contrainte $\sum_i \xi_i^2$, mais alors les points très mal classés ont une influence très importante sur la solution w trouvée; en prenant $\sum_i \xi_i$, on pénalise sans trop avoir d'influence des cas très mal classés.

4.2.5 Bilan sur le Perceptron

Avec ce simple exemple du Perceptron et de son extension au Support Vector Machine, on voit apparaître des concepts qui se sont avérés très importants:

- la descente de gradient (GD)
- sa variante gradient stochastique (SGD) (bien que dans le cas précis ici, le hasard ne soit pas apparu comme un ingrédient essentiel: c'est dû à la convexité du problème simple. On y reviendra dans un cours plus tard.)
- les problèmes de convergence

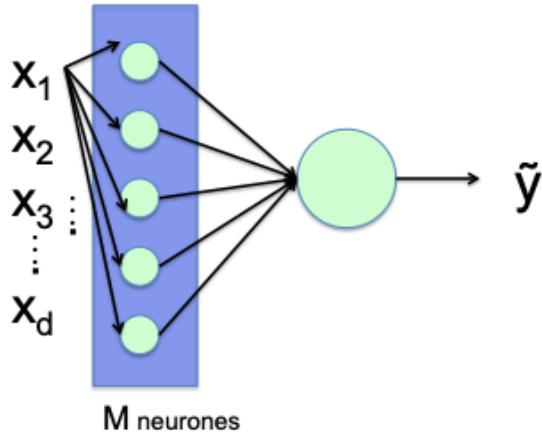


FIGURE 44 – Schématisation d'un réseau de neurones à 1 couche cachée ou bien à 2 couches (cf. la couche d'entrée + la couche cachée). Notez que le nombre de neurones cachés est aussi noté K .

— les problèmes de régularisation.

Mais il reste le **problème d'approximation**. On réussit à faire converger l'algorithme certes, **mais quelle est la taille de l'ensemble des fonctions que l'on arrive à approximer avec le Perceptron ?** On sait que le XOR ne peut rentrer dans cette classe.

5. Architecture multi-couches: Partie I

5.1 Introduction

En introduction, S. Mallat fait référence à la déprime de la communauté dans les années 70s²⁵. Dans ces années-là, les classificateurs linéaires ont été disqualifiés, puis il a été réalisé que la piste des multicouches devait résoudre le problème des classifications non-linéaires, et tout cela a été bien formalisé dans les années 80-90s. On va donc étudier le **Réseau à 1 couche cachée** ou comme le disent certains "le réseau à 2 couches" (voir la figure 44), et l'on va montrer que **ce réseau permet d'approximer n'importe quelle**

²⁵. NDJE: C'est le "1er âge glaciaire". Il y en a eu un second dans les années 90s à cause des gradients devenant nuls ou infinis quand on augmente le nombre de couches.

fonction (linéaire ou non): c'est le **Théorème d'Universalité** que nous allons démontrer sous deux angles différents. À travers la démonstration de ce théorème, on va voir que:

- si la démonstration de ce théorème peut être considérée comme une jolie pièce de maths, il n'en reste pas moins qu'en fin de comptes, il ne résout pas le problème comme on l'a esquissé l'an dernier, à savoir que la **malédiction de la dimensionnalité** n'est pas loin;
- cependant ce théorème est intéressant, car il fait comprendre la nature du problème. En particulier quand on fait de **l'approximation, on utilise la régularité de la fonction** et donc on voit par ce biais quel est l'outil mathématique que l'on utilise dans ce cas: **la théorie de l'approximation et l'analyse de Fourier absolument centrale ici** pour comprendre la capacité d'approximation de ce réseau.

5.2 L'expression de la sortie du réseau

Dans l'architecture de la figure 44²⁶,

$$\tilde{y} = \sigma \left[\sum_{k=1}^K c_k \sigma \left[\sum_{m=1}^d w_{k,m} x_m + b_k \right] + b'_k \right] = \tilde{f}(x) \quad (72)$$

avec $\sigma(x)$ une non-linéarité (ReLU, sigmoïde, tanh, signe, etc) (*nb. notez que par moment on a noté la fonction σ par ρ*). Ce que l'on constate maintenant, c'est que dans les RNP, si on change de fonction de non-linéarité (et il y a eu beaucoup d'expérimentations de faites), l'architecture si elle est bien configurée va toujours fonctionner si on l'entraîne de nouveau. Il faut juste veiller à ce qu'il y ait bien une non-linéarité non polynomiale. La fonction **ReLU est particulièrement adaptée**, car la dérivée est presque partout définie et bornée, et donc la descente de gradient se fait bien. On a ce faisant **abandonné toute sur-interprétation des types de non-linéarité** comme cela a été très souvent fait par le passé.

Notons que chaque neurone k de la couche cachée répond selon

$$x_k \equiv \sigma \left[\sum_{m=1}^d w_{k,m} x_m + b_k \right] \quad (73)$$

26. nb. la notation a un peu changé par rapport à l'année 2018, mais rien de fondamental

et cela n'est autre que l'application d'une **séparation par l'hyperplan** (w_k, b_k) des données en entrée. Ainsi, sur la couche cachée, on obtient la collection de K hyperplans séparateurs en 1 passe, ce qui vous laisse imaginer la puissance de sélection. On peut écrire de façon matricielle la relation précédente pour le neurone caché k

$$x_k = \sigma [W_k x + b_k] \quad (74)$$

et ceci se généralise dans un multicouches où la sortie d'une couche $j+1$ dépend de la sortie de la couche précédente j

$$\vec{x}_{j+1} = \sigma [W_j \vec{x}_j + \vec{b}_j] \quad (75)$$

ou bien en utilisant une autre notation

$$\boxed{\mathbf{x}_{j+1} = \sigma [\mathbf{W}_j \mathbf{x}_j + \mathbf{b}_j]} \quad (76)$$

Dans la suite des cours en général, on oublie la notation avec les flèches ou en **gras** pour les vecteurs (par moment, nous utiliserons celle-ci) mais il faut bien se représenter cette notion vectorielle/matricielle (*nb. la non linéarité est appliquée pour chaque composante du vecteur entre []*).

En notant Θ l'ensemble des paramètres du réseau, c'est-à-dire tous les $(W_k, b_k, c_k, b'_k)_{k \leq K}$, on s'intéresse à la **classe de toutes les fonctions** \tilde{f}_Θ :

$$\mathcal{H} = \left\{ \tilde{f}_\Theta / \forall \Theta \right\} \quad (77)$$

et on se demande si cette **classe est suffisamment grande pour approximer la fonction $f(x)$ qui m'intéresse?** Si tel n'est pas le cas, on aura une erreur de biais.

On dispose de n **échantillons étiquetés** $\{x_i, y_i = f(x_i)\}_{i \leq n}$ et comme on l'a vu précédemment, on a un problème qui se divise en 3 parties (**approximation, estimation et optimisation**):

- primo, on va essayer de chercher dans \mathcal{H} une fonction particulière \tilde{f}_{Θ^*} telle que $\tilde{f}_{\Theta^*} \approx f(x)$; c'est donc une question **d'approximation**;
- secundo, quand bien même \tilde{f}_{Θ^*} existerait, **peut-on l'estimer à partir de n exemples?**
- tertio, finalement quand bien même, il serait possible de l'estimer, on veut obtenir

une “bonne” approximation, on a donc un problème **d’optimisation**.

Dans ce qui suit, on va s’attaquer à la partie “approximation”, sachant que “l’estimation” est aisée dans ce cas (convexité), et il restera l’optimisation à traiter. En fait le problème très bloquant est bien celui de l’approximation.

5.2.1 Le cas des fonctions Booléennes

A priori, on a une fonction de type:

$$x \in \mathbb{R}^d \xrightarrow{f} y \in \mathbb{R} \quad (78)$$

mais, ici, on va approximer chaque composante de x par un q -bit (*nb. ici la notation d’un bit est ± 1*), idem pour y , donc dans un premier temps regardons les fonctions de type

$$x \in (\pm 1)^{q \times d} \xrightarrow{f} y \in (\pm 1)^q \quad (79)$$

Si q est suffisamment grand, on peut connaître les réels avec une bonne précision numérique. Notons, que l’on peut décomposer la fonction f en q fonctions qui donnent 1-bit de y , donc on simplifie le problème en étudiant les fonctions de type:

$$x \in (\pm 1)^{q \times d} \xrightarrow{f} y \in (\pm 1) \quad (80)$$

On peut voir ce problème comme une classification à 2 classes avec en entrée des vecteurs à $q \times d = d'$ dimensions quantifiées.

Théorème 2. *Soit la fonction f , telle que*

$$x \in (\pm 1)^{d'} \xrightarrow{f} y \in (\pm 1) \quad (81)$$

alors pour tout d' , il existe un réseau à 2 couches (ou 1 couche cachée), tel que $f \in \mathcal{H}$.

Démonstration 2. Nous allons procéder à la construction du réseau de neurones qui va bien et voir de combien de neurones cachés on a besoin. Essayons donc de construire un réseau en se concentrant sur l’ensemble des x définis par

$$\mathcal{A} = \{x / f(x) = 1\} \quad (82)$$

On note que x a $2^{d'}$ valeurs possibles. Prenons K éléments de cet ensemble que l'on note selon

$$\{w_k\}_{k \leq K} \quad \text{avec} \quad w_k = (\pm 1)^{d'} \quad \text{et} \quad f(w_k) = 1 \quad (83)$$

et pour les non-linéarités, on prendra la fonction “signe”. Considérons à présent la relation entre l’entrée x et la première couche:

$$\sigma(w_k \cdot x + b_k) \quad (84)$$

En analysant les bits de x et w_k , le produit scalaire est égal à

$$w_k \cdot x = \begin{cases} d' & \text{si } x = w_k \\ \leq d' - 2 & \text{si } x \neq w_k \end{cases} \quad (85)$$

et si l’on prend $b_k = 1 - d'$, alors

$$\sigma(w_k \cdot x + b_k) = \begin{cases} 1 & \text{si } x = w_k \\ -1 & \text{si } x \neq w_k \end{cases} \quad (86)$$

Donc, chaque neurone k détecte si l’entrée x est égale à w_k ou non. On peut ainsi noter $f(x)$ comme une suite de OU

$$f(x) = 1 \text{ si } (x = w_1) \parallel (x = w_2) \parallel \dots \parallel (x = w_K) \quad (87)$$

Il nous faut donc réaliser une disjonction logique (OU) de ces K égalités. En fait, on la réalise comme cela

$$f(x) = \sigma \left(\sum_{k=1}^K \sigma(w_k \cdot x + 1 - d') + K - 1 \right) = \begin{cases} 1 & \text{si } \exists k / x = w_k \\ -1 & \text{si } \forall k, x \neq w_k \end{cases} \quad (88)$$

On a en fait construit une représentation d’une fonction Booléenne. Or, pour représenter tous les x tels que $f(x) = 1$ comme des q -bits en dimension d , **il va falloir**

typiquement $2^{d'-1} = K$ **neurones** si on considère que la moitié des vecteurs sont de la classe 1, mais sinon on peut certainement dire que $K \approx O(2^{d'})$. Donc, on a besoin d'un **nombre exponentiel de neurones**.

Finalement, ce type de réseau est **une grosse mémoire** qui stocke des vecteurs (w_k) pour lesquels la fonction vaut 1. Pour effectuer la classification, il fait une simple comparaison entre l'entrée et ces vecteurs encodés. **Ces réseaux paraissent, donc assez grossiers**, voire exempts d'une quelconque forme de complexité. Or, on sait que ces neurones à 1 couche cachée sont assez satisfaisants pour un certain nombre de problèmes, mais pas trop complexes, c'est-à-dire on ne peut leur demander de faire de la classification d'images. Pour cela, il faut aller au-delà de cette description simple des fonctions booléennes.

5.2.2 Usage de la régularité de la fonction

Dans la construction précédente, on s'est contenté de stocker les vecteurs tels que $f(x) = 1$, **sans se préoccuper de la régularité de la fonction**. Donc, primo, il faut vraiment considérer $x \in \mathbb{R}^d$ comme une variable continue (cf. non quantifiée), et secundo analyser la régularité de $f(x)$. C'est dans ce cadre de continuité qu'apparaît le **Théorème d'Universalité**. On va simplifier l'expression

$$\tilde{f}(x) = \sigma \left[\sum_{k=1}^K c_k \sigma \left[\sum_{m=1}^d w_{k,m} x_m + b_k \right] + b'_k \right] \quad (89)$$

en enlevant la dernière non-linéarité non essentielle ici. Alors on essaye d'approximer $\tilde{f}(x)$ selon

$$\tilde{f}(x) = \sum_{k=1}^K c_k \sigma \left[\sum_{m=1}^d w_{k,m} x_m + b_k \right] = \sum_{k=1}^K c_k \sigma [w_k \cdot x + b_k] \quad (90)$$

Notons que les termes $\sigma [w_k \cdot x + b_k]$ sont des fonctions en anglais nommées des "**ridge functions**". D'une manière générale une fonction "ridge" est une fonction

$$f : \mathbb{R}^d \longrightarrow \mathbb{R} \quad tq \quad \exists g_a : \mathbb{R} \longrightarrow \mathbb{R} \quad tq \quad f(x) = g(a \cdot x) \quad (91)$$

Alors

$$\forall x \in \mathbb{R}^d / a \cdot x = c \in \mathbb{R} \Rightarrow f(x) = g(c) \quad (92)$$

c'est-à-dire **qu'une fonction ridge est constante sur l'hyperplan** $a.x = c$. Ainsi, $\sigma [w_k.x + b_k]$ est constante pour les x appartenant à des hyperplans parallèles à l'hyperplan (w_k, b_k) , et ne varie que si x a une composante colinéaire à w_k . Donc, le problème est de décomposer $f(x)$ (trouver une approximation) selon K fonctions (ridges) qui ne varient chacune que dans 1 direction. La question au bout du compte sera de savoir de combien de directions (cf. valeurs de K) faudra-t-il pour approximer $f(x)$?

Soit l'ensemble des fonctions²⁷

$$\mathcal{M}_\sigma = \text{Vect} \left\{ \sigma [w.x + b] / w \in \mathbb{R}^d, b \in \mathbb{R} \right\} \quad (93)$$

Est-il suffisamment grand pour pouvoir approximer la fonction f ? Autrement dit, **\mathcal{M}_σ est-il dense dans l'ensemble des fonctions continues $C(\mathbb{R}^d)$** . C'est le minimum de régularité que l'on impose.

Donc, on veut approximer $f(x)$ par une combinaison linéaire d'éléments de \mathcal{M}_σ aussi précisément que possible. Il faut donc se donner un critère, une distance. Pour cela, on va se placer sur des compacts Ω dans \mathbb{R}^d (ce sont essentiellement des ensembles bornés), et donc on veut

$$\forall \epsilon > 0, \exists \tilde{f} \in \mathcal{M}_\sigma / \forall \Omega \subset \mathbb{R}^d, \forall x \in \Omega \quad |f(x) - \tilde{f}(x)| \leq \epsilon \quad (94)$$

Il s'agit de la **métrique de convergence uniforme sur un compact** qui est une **topologie particulière**. Ces aspects ont été très étudiés entre 1987 et 1993:

- Hecht & Nielsen qui utilisaient un résultat de Kolmogorov qui en fait ne devait pas s'appliquer ont montré une nouvelle façon de poser le problème et ont redonné un certain renouveau au problème;
- Gallant & White ensuite se sont posés la question si on peut construire une fonction sigmoïde particulière qui pourrait satisfaire le résultat d'approximation;
- Cybenko en 89 montre que le résultat est vrai pour toute sigmoïde;
- Liensko, Lin, Pinkus et Schocken (93) montrent que le résultat se généralise à toute fonction non-linéaire σ pourvu qu'elle ne soit pas un polynôme.

Finalement, S. Mallat mentionne le papier de revue datant de 1999 qui présentait l'état de l'art à cette époque: **Allan Pinkus** “*Approximation theory of MLP model in Neural*

27. nb. "combinaison linéaire" = Vect

Network"²⁸.

5.3 Théorème d'Universalité d'un réseau à 1-couche cachée

Théorème 3. *Soit une fonction $\sigma \in C(\mathbb{R})$ alors \mathcal{M}_σ est dense pour la convergence uniforme sur un compact, ssi σ n'est pas un polynôme.*

Nous allons procéder à une démonstration **constructive**, car à travers cette construction, on veut comprendre: comment cela fonctionne; pourquoi c'est difficile; pourquoi cela se relie à la **régularité de la fonction** à approximer en particulier **plus la fonction sera régulière moins on aura besoin de neurones cachés; comment la malédiction de la dimensionnalité se manifeste sauf dans quelques cas particuliers**, etc.

Si σ est un polynôme de degré m , voyons ce qu'il se passe en $d = 1$ alors

$$\tilde{f}(x) = \sum_{k=1}^K c_k \sigma(w_k x + b_k) \quad (95)$$

est aussi un polynôme de degré m , or ce n'est pas suffisant pour approximer une fonction continue sur un intervalle. On peut le vérifier avec un polynôme de degré $m + 1$, si on veut l'approximer avec un polynôme de degré m on aura une erreur non négligeable. Par exemple sur l'intervalle borné $\Omega = [0, 1]$ prenons la fonction continue:

$$f(x) = x^5 \quad (96)$$

que l'on l'ajuste avec un polynôme de degré 4. Les résidus sont ceux de la figure 45. On ne peut obtenir une convergence uniforme d'erreur arbitrairement petite sur $[0, 1]$. Donc, on constate que **cela ne marche pas car on reste dans la classe des polynômes si σ est un polynôme**.

En fait, précisons bien le problème qui se pose: dans le cas d'usage d'un MLP, on ne connaît pas f en fait, on ne connaît que des échantillons étiquetés, et on veut fixer σ *a priori* (dans le cas ci-dessus, on fixe m le degré du polynôme) pour remplir des critères pour tout f appartenant à une certaine classe de fonctions, ici les fonctions continues sur

28. <http://www2.math.technion.ac.il/~{}pinkus/papers/acta.pdf>

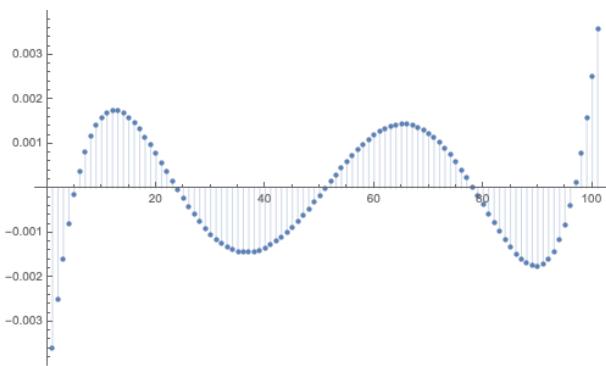


FIGURE 45 – Résidus de l'ajustement de la fonction x^5 par un polynôme de degré 4.

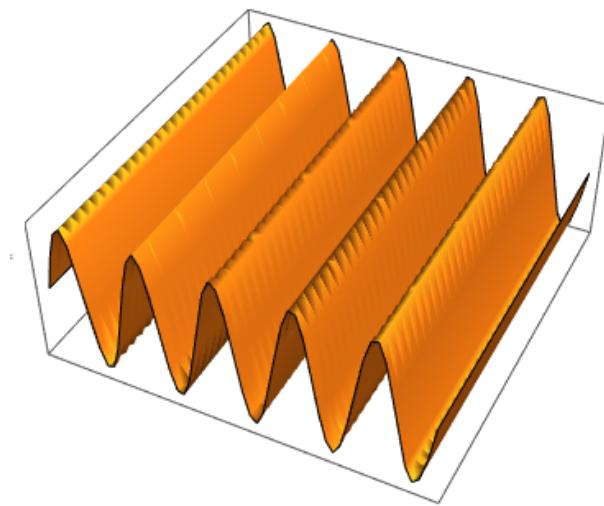


FIGURE 46 – Fonction ridge sinusoïdale qui varie uniquement selon une seule dimension.

\mathbb{R}^d , c'est-à-dire $C(\mathbb{R}^d)$. Donc, prenons **un** σ **non-polynomial**. S. Mallat mentionne qu'il y a beaucoup de stratégies de démonstration, celle qu'il propose passe par les *sinus/cosinus*, car les **fonctions ridges** que l'on maîtrise bien mathématiquement sont les bases de **Fourier**. On va choisir des fonctions sinus en \mathbb{R}^d qui ne vont varier que selon 1 dimension uniquement (figure 46).

5.3.1 Base de Fourier

Lemme 1. *On peut approximer n'importe quelle fonction élément de $C(\mathbb{R}^d)$ avec une précision arbitraire sur une base de Fourier, autrement dit*

$$\forall f \in C(\mathbb{R}^d), \forall \Omega(\text{compact}) \subset \mathbb{R}^d, \forall \epsilon > 0, \exists K \text{ et } \{w_k\}_{k \leq K}$$

$$\text{tq. : } \forall x \in \Omega \quad |f(x) - \sum_{k=1}^K (\alpha_k \cos(w_k \cdot x) + \beta_k \sin(w_k \cdot x))| \leq \epsilon \quad (97)$$

Par le biais de ce lemme, on fait la connexion avec **l'Analyse harmonique** qui va permettre de comprendre en quoi le nombre de fonctions ridges nécessaires (cf. K) est directement relié à la régularité de la fonction f . Imaginons un instant que ce lemme soit démontré, si par la suite, on peut écrire une décomposition des *sinus/cosinus* selon les fonctions ridges σ comme cela

$$\cos(t) = \sum_j \lambda_j^c \sigma(\gamma_j^c t + \delta_j^c) \quad (98)$$

$$\sin(t) = \sum_j \lambda_j^s \sigma(\gamma_j^s t + \delta_j^s) \quad (99)$$

alors, on aura gagné. Passons à la démonstration du lemme en reliant K à la régularité de f . Donc, on travaille sur un compact $\Omega \subset \mathbb{R}^d$. Comme il est **borné**, on peut l'immerger dans une grande boite telle que (figure 47)

$$\Omega \subset [-\pi\Delta, +\pi\Delta]^d \quad (100)$$

On définit ensuite une base de Fourier d'énergie finie sur la grande boite $L^2([-\pi\Delta, +\pi\Delta]^d)$.

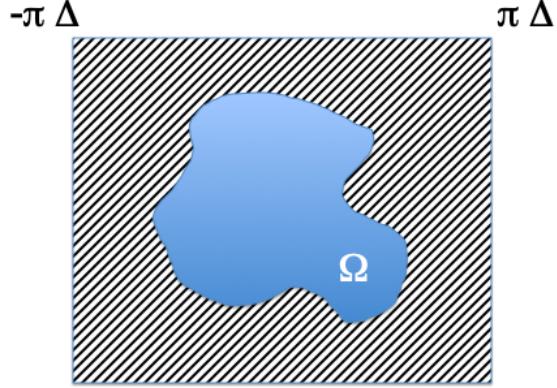


FIGURE 47 – Immersion du compact Ω dans une boîte cubique de taille $[-\pi\Delta, \pi\Delta]^d$.

Petit rappel en dimension $d = 1$

$$L^2([-\pi\Delta, +\pi\Delta]) = \left\{ f \mid \|f\|^2 = \int_{-\pi\Delta}^{\pi\Delta} |f(x)|^2 dx < \infty \right\} \quad (101)$$

et sur $L^2([-\pi\Delta, +\pi\Delta])$ on sait que la famille

$$\left\{ e^{inx/\Delta} \right\}_{n \in \mathbb{Z}} \quad (102)$$

est une base orthogonale, le produit scalaire étant défini par

$$\langle f, g \rangle = \int_{-\pi\Delta}^{\pi\Delta} f(x)g^*(x)dx \quad (103)$$

Maintenant en d dimensions, $x \in \mathbb{R}^d = (x_1, \dots, x_d)$, et l'ensemble des vecteurs

$$\left\{ e^{in_1 x_1 / \Delta} \cdot e^{in_2 x_2 / \Delta} \cdot (\dots) \cdot e^{in_d x_d / \Delta} \right\}_{\{n_1, \dots, n_d\} \in \mathbb{Z}^d} = \left\{ e^{iw \cdot x / \Delta} \right\}_{w \in \mathbb{Z}^d} \quad (104)$$

forment **une base orthogonale de $L^2([-\pi\Delta, +\pi\Delta]^d)$** ²⁹.

On veut approximer f sur Ω (figure 47) que l'on a plonger dans $[-\pi\Delta, +\pi\Delta]^d$, on va

29. nb. En deux mots la démonstration se fait 1) en montrant que la version 1d est orthogonale, 2) la densité se fait par le noyau de Poisson, 3) la version en dimension d se fait par le produit des bases dans les d dimensions 1d.)

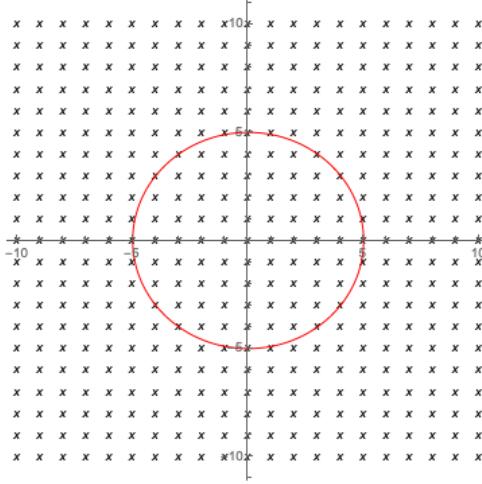


FIGURE 48 – Illustration des poids $w = \{n_1, \dots, n_d\} \in \mathbb{Z}^d$ et, se concentrer sur les poids $\|w\| \leq C$ est équivalent à utiliser une régularisation de type Perceptron.

donc prolonger f dans \mathbb{R}^d en imposant une condition de nullité en dehors de $[-\pi\Delta, +\pi\Delta]^d$, et en procédant par exemple par interpolation linéaire sur le complémentaire de Ω dans $[-\pi\Delta, +\pi\Delta]^d$. Donc, on sait d'après ce qui précède que $f \in L^2([-\pi\Delta, +\pi\Delta]^d)$ peut être décomposée selon la base de Fourier selon³⁰

$$f(x) = \frac{1}{(2\pi\Delta)^d} \sum_{w \in \mathbb{Z}^d} \hat{f}(w) e^{iw \cdot x / \Delta} \quad (105)$$

Or, cela nécessite **une infinité de fréquences**, donc on aimerait savoir si l'on peut **prendre un nombre limité** (cf. c'est une forme de parcimonie) et quand même obtenir une bonne approximation de f .

Les $w \in \mathbb{Z}^d$ sont en fait **les poids des neurones cachés**, en 2D, ils pavent l'espace selon la figure 48 (intersections entières). En **se limitant** aux plus petits poids (cercle rouge), on retrouve ainsi la **régularisation du Perceptron** :

$$\|w\| \leq C \quad (106)$$

Cette restriction est très naturelle, car on veut approximer des fonctions **régulières** dont

30. le coefficient de normalisation n'a pas d'importance.

les **basses fréquences** sont prépondérantes. On tente alors d'approximer la fonction f selon la décomposition sur la base de Fourier tronquée, et on va regarder la norme de l'erreur.

$$\begin{aligned}
 Err &= \left\| f(x) - \frac{1}{(2\pi\Delta)^d} \sum_{|w| \leq C} \hat{f}(w) e^{iw \cdot x / \Delta} \right\|^2 \\
 &= \left\| \frac{1}{(2\pi\Delta)^d} \sum_{|w| > C} \hat{f}(w) e^{iw \cdot x / \Delta} \right\|^2 \\
 &= \frac{1}{(2\pi\Delta)^{2d}} \sum_{|w| > C} |\hat{f}(w)|^2
 \end{aligned} \tag{107}$$

On voit ici le gros avantage d'être dans une base orthogonale, l'erreur est due à l'énergie à haute fréquence qui n'a pas été prise en compte. Or, l'hypothèse de régularité de f va jouer sur la vitesse de décroissance des coefficients de Fourier³¹. **En 1d, pour une fonction de l'espace de Sobolev dérivable α -fois ($\alpha > 1/2$)**, les coefficients de Fourier satisfont la relation de décroissance suivante:

$$|\hat{f}(\omega)| = o(|\omega|^{-\alpha}) \tag{108}$$

Donc, on garantit une décroissance des coefficients de Fourier, on peut donc fixer un seuil C pour lequel la somme des termes à haute fréquence est plus petite que ϵ . Ce qui va importer alors est la dépendance $C(\epsilon)$, c'est-à-dire de combien de fréquences a-t-on besoin? ou dit autrement pour notre réseau à 1 couche, de combien de neurones cachés a-t-on besoin? On voit bien la problématique et on va la dérouler à d dimensions.

Si on fixe C , alors le nombre K de vecteurs w tels que $|w| \leq C$ est à une constante de proportionnalité près égale à

$$K \propto C^d \tag{109}$$

Donc, si C par malheur explose avec la contrainte sur ϵ alors le nombre de neurones va aussi exploser, car d est potentiellement aussi très grand. Or, on va s'apercevoir que C est grand si la fonction est irrégulière. On voit ainsi poindre le triptyque:

régularité \leftarrow Malédiction de la dimension \rightarrow nombre de neurones

31. NDJE: revoir le cours de 2018 sur l'analyse de Fourier.

Montrons que si $f \in C(\mathbb{R}^d)$ bornée, alors on a un nombre fini d'éléments. On sait que $f \in L^2([-\pi\Delta, +\pi\Delta]^d)$ et que l'on peut la décomposer sur la base de Fourier, et à une constante près, l'énergie de la fonction est la somme des composantes de Fourier qui converge

$$\|f\|^2 = \frac{1}{(2\pi\Delta)^{2d}} \sum_{w \in \mathbb{Z}^d} |\hat{f}(w)|^2 < \infty \quad (110)$$

Donc,

$$\forall \epsilon > 0, \exists C(\epsilon) / \frac{1}{(2\pi\Delta)^{2d}} \sum_{|w| > C(\epsilon)} |\hat{f}(w)|^2 \leq \epsilon \quad (111)$$

Remarquons, que le lemme est basé sur une métrique différente, la norme infinie (appelée aussi "norme sup" ou "norme de la convergence uniforme") alors que le raisonnement ci-dessus est basé sur la norme L2. Or, nous sommes en dimension infinie (cf. usage des intégrales) et les deux normes ne sont pas équivalentes. Cependant,

- Si f est indéfiniment dérivable, et si l'on prend sa série de Fourier tronquée alors le résultat est vrai, c'est-à-dire qu'il y a convergence uniforme. En fait l'hypothèse de dérivabilité implique une décroissance des coefficients de Fourier très rapide.
- Si f est (**simplement**) **continue**, on ne peut tronquer brutalement la série de Fourier; mais on peut montrer qu'elle est approximable par des polynômes selon le théorème de Stone-Weierstrass. Exemple à 1d, sur le compact $[-\pi, \pi]$:

$$\forall \epsilon > 0 \ \exists p / \forall x, |f(x) - p(x)| \leq \epsilon \quad (112)$$

Or, les polynômes sont infiniment dérивables, donc la série de Fourier tronquée de p (notée $\hat{p}_C(w)$) va converger vers p , et donc la cascade $\hat{p}_C \rightarrow p \rightarrow f$ permet de conclure. Il y a cependant un point subtil: c'est que \hat{p}_C n'est pas la série de Fourier tronquée de f .

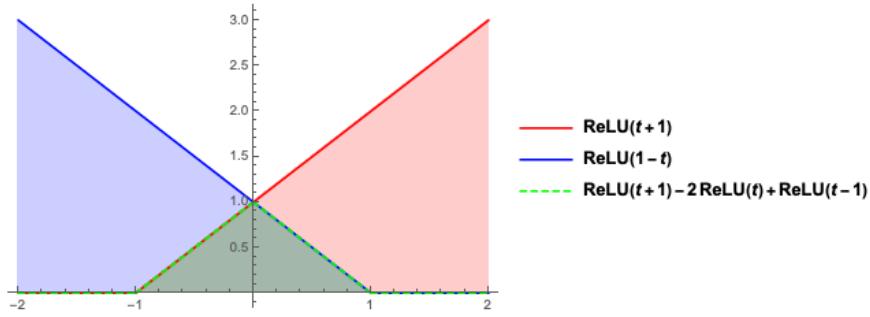


FIGURE 49 – Avec une combinaison linéaire de ReLU, on peut obtenir une fonction "triangle" (en vert).

5.3.2 Approximation des sinus par σ

Reste donc à approximer les sinusoïdes avec la non-linéarité. **On va le faire avec le ReLU.** Donc, on veut traiter le cas des

$$\begin{cases} \cos(w \cdot x) \\ \sin(w \cdot x) \end{cases} \quad \text{avec } |w| < C \text{ et } x \in [-\pi\Delta, +\pi\Delta]^d \quad (113)$$

Notons que le produit scalaire $w \cdot x$ (argument du cosinus ou sinus) est borné selon

$$|w \cdot x| < |w| \cdot \|x\|_{L2} < |w| \cdot \|x\|_{L1} < C\pi\Delta d \quad (114)$$

Prenons le cosinus (ici, on se retrouve avec des scalaires, donc en 1d), donc on veut

$$|\cos(t) - \sum_j \lambda_j^c \sigma(\gamma_j^c t + \delta_j^c)| \leq \epsilon/K \quad (115)$$

(la borne vient du fait que l'approximation de f par la somme des K cosinus et sinus est bornée à ϵ , on oublie le facteur 2) et combien faut-il de non-linéarité?

Observons qu'avec **un ReLU, on peut faire une fonction Triangle** par combinaison linéaire comme sur la figure 49. Or, on sait (cours de 2018) que **le théorème d'échantillonnage** nous dit que ce triangle engendre l'espace vectoriel des splines linéaires. Pour s'en convaincre, sur la figure 50, on peut facilement se convaincre que la somme des triangles

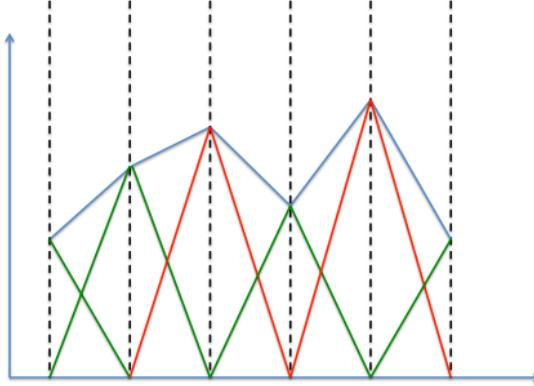


FIGURE 50 – Approximation d'une fonction affine par morceaux à l'aide de fonctions "triangle" correctement normalisées.

rouges et verts donnent la courbe bleue, donc des triangles approximent exactement une fonction linéaire par morceaux, laquelle peut approximer le cosinus à un certain niveau de précision fixée. **Donc, en utilisant le ReLU et ses translations on peut obtenir n'importe quelle approximation linéaire par morceaux du cosinus.**

Remarquons que **les biais servent dans cette opération à translater les ReLU**. Cependant, combien de ReLU faut-il? En fait, il en faut autant que des biais, c'est-à-dire autant que le nombre d'échantillons qu'il faut pour obtenir une approximation à $\epsilon' = \epsilon/K$ près³².

Il faut $2K$ sinus et cosinus, et chaque sinus/cosinus a besoin de $2\pi\Delta dC/\epsilon'$ échantillons (taille du domaine divisé par l'erreur), donc le nombre de ReLU est (ordre de grandeur)

$$N_\sigma \approx K^2 \Delta dC/\epsilon \approx C^{2d+1} \frac{d\Delta}{\epsilon} \quad (116)$$

Dans cette formule, finalement Δ est la taille du domaine et d la dimension, ces deux facteurs ne sont pas dramatiques. Par contre, on a le **facteur crucial C^{2d+1} avec $C(\epsilon)$ qui est régi par la régularité de la fonction laquelle fixe la coupure à haute fréquence.**

Dans la section suivante, nous analyserons comment $C(\epsilon)$ varie avec la régularité de

32. Une remarque en passant, la ReLU n'est pas cruciale, il faut pouvoir définir à partir de la non-linéarité, une fonction à support compact qui joue le rôle du Triangle

la fonction. Cependant, on ne va pas pouvoir le contraindre à décroître suffisamment rapidement pour contrebalancer l'exponentiation pour ne pas tomber dans **le malheur de la dimensionnalité**. Le seul cas qui peut marcher, c'est quand on peut utiliser **beaucoup moins** de fréquences, c'est-à-dire que l'on peut être **parcimonieux**. Il y a des cas où cela se produit: par exemple quand le problème est séparable dans les différentes dimensions. On verra aussi l'état de la connaissance en l'an 2000 sur ce que l'on peut apprendre quand on augmente le nombre de couches: on verra qu'on n'avait pas de grand espoir “ça ne sert à rien d'augmenter le nombre de couches”!

6. Architecture multicouches: Partie II

6.1 Introduction

On va revenir sur le **Théorème d'Universalité** dont la démonstration faite à la section précédente n'est pas tant qu'elle offre une construction du Réseau de Neurones à 1 couche cachée, mais plutôt qu'elle met en lumière les limites de ce type de réseau, et qu'elle donne un état de l'art de la compréhension du problème et des questions soulevées. Donc, on part sur le schéma de la figure 51 (*nb. la seconde non-linéarité ne sera pas nécessaire*) et l'on se demande: **si en fonction de K (le nombre de neurones cachés) peut-on approximer n'importe quelle fonction continue sur \mathbb{R}^d ?**

Le premier niveau de réponse est bien **oui**, on peut approximer n'importe quelle fonction, mais la plupart des cas de figures K va devenir énorme dès lors que l'on demande que l'approximation soit bonne. Le second niveau de réponse est **K va devoir être énorme**, mais cela va dépendre de la **régularité de la fonction $f(x)$** .

C'est le second niveau qui va nous intéresser, car **il fait le lien entre K , le nombre de neurones et la régularité de la fonction que l'on veut approximer f que l'on ne connaît toujours pas a priori**. Il peut être curieux de s'accrocher à la notion de régularité, parce qu'on a l'impression qu'elle est bien définie. Or, ce n'est pas le cas, **cette notion de régularité a plusieurs facettes que l'on peut voir sous différents angles**, avec certaines qui ne sont pas effectives pour le problème qui nous occupe ici, et d'autres plus adaptées. Notons pour finir cette introduction que la “régularité” est fondamentale pour comprendre le problème de “généralisation” pour les RNP.

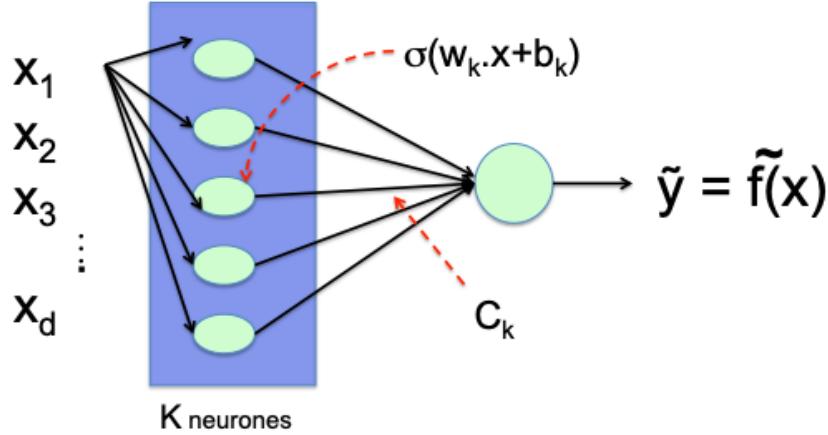


FIGURE 51 – Schéma légèrement modifié de la figure 44 avec les notations du Théorème d’Universalité.

6.2 Rappels sur le Théorème d’Universalité

Donnons au **Théorème d’Universalité** (section 5.3) une autre formulation:

Théorème 4. *Si $f \in C(\mathbb{R}^d)$, soit une fonction $\sigma \in C(\mathbb{R})$ non polynomiale, alors*

$$\forall \epsilon > 0, \exists \tilde{f} \in \mathcal{M}_\sigma / \forall \Omega \subset \mathbb{R}^d, \forall x \in \Omega \quad |f(x) - \tilde{f}(x)| \leq \epsilon \quad (117)$$

avec

$$\boxed{\tilde{f}(x) = \sum_{k=1}^{K(\epsilon)} C_k \sigma(w_k \cdot x + b_k)} \quad (118)$$

Rappel du principe de la démonstration qui se passe en deux temps:

1. on décompose $f(x)$ selon une famille de fonctions ridges de type sinus/cosinus. Décomposition en série de Fourier où l’on identifie les $\{w_k\}$ comme les index de fréquences (cf. le compact Ω étant borné, on peut le plonger dans une grande boîte de volume $(2\Delta)^d$ que l’on quantifie), et les $\{b_k\}$ comme les index de translations de la fonction σ d’échantillonnage³³. On limite aux basses fréquences $|w| < C$ pour

33. nb. vu avec un ReLU mais la démonstration complète montre que ce résultat est indépendant de σ

obtenir une approximation, ce qui donne la valeur de K .

2. Ensuite, on montre que les sinus et cosinus se décomposent sur les fonctions σ .

Au bilan le nombre de σ nécessaires N_σ pour que l'erreur sur $f(x)$ soit ϵ est obtenu selon

$$|w| < C_\epsilon \Rightarrow K = C_\epsilon^d \Rightarrow N_\sigma \approx K^2 \Delta d C_\epsilon / \epsilon \approx C_\epsilon^{2d+1} \frac{d\Delta}{\epsilon} \quad (119)$$

En fait, c'est bien **la contrainte sur K qui compte**. La question est: que vaut la constante C_ϵ ?

6.3 Convergence de l'approximation \tilde{f}

On sait que l'on a coupé les hautes fréquences pour obtenir l'approximation $\tilde{f}(x)$, donc

$$\tilde{f}(x) = \sum_{|w| < C_\epsilon} \hat{f}(w) e^{i 2\pi \frac{w \cdot x}{\Delta}} \quad (120)$$

Comme de plus, on a une base orthogonale, alors on obtient rapidement une erreur de l'approximation car:

$$f(x) - \tilde{f}(x) = \sum_{|w| \geq C_\epsilon} \hat{f}(w) e^{i 2\pi \frac{w \cdot x}{\Delta}} \quad (121)$$

et en norme L^2 cela se traduit par

$$\|f(x) - \tilde{f}(x)\|^2 = \frac{1}{(2\pi\Delta)^d} \int_{[-\pi\Delta, \pi\Delta]^d} |f(x) - \tilde{f}(x)|^2 dx \quad (122)$$

l'erreur n'est autre que l'énergie de tous les coefficients omis, à savoir

$$\|f(x) - \tilde{f}(x)\|^2 = \frac{1}{(2\pi\Delta)^d} \sum_{|w| \geq C_\epsilon} |\hat{f}(w)|^2 \quad (123)$$

La régularité de la fonction f va se traduire par la vitesse de décroissance des coefficients de Fourier³⁴. C'est la base de toute l'Analyse Harmonique et Fonctionnelle.

Notons que comme $f \in L^2([-\pi\Delta, \pi\Delta]^d)$ la série de Fourier converge. Donc, quand $C_\epsilon \rightarrow \infty$ alors on sait que l'erreur va tendre vers 0. De ce côté-là, il n'y a pas de problème.

34. NDJE: revoir le cours de 2018 sur les fonctions lipschitziennes.

Le vrai problème est encore une fois comment choisir la valeur de K ? On peut relier **la régularité uniforme de f avec la décroissance de $|\hat{f}(w)|$ qui va définir K .**

6.4 Définition(s) de la Régularité

6.4.1 Régularité au sens des dérivées (Sobolev/Hilbert): Théorème d'optimalité de Majorov

Prenons le cas à 1d pour illustration. On peut par exemple définir une régularité simplement par la dérivée première, par exemple, considérer que la fonction est **dérivable**, et qu'elle soit **d'énergie finie**, c'est-à-dire que

$$\int |f'(x)|^2 dx < \infty \quad (124)$$

Or, $\hat{f}'(w) = iw\hat{f}(w)$, et si on est sur un compact, alors la seconde contrainte donne

$$\sum_w |w|^2 |\hat{f}(w)|^2 < \infty \quad (125)$$

Donc, ce n'est pas uniquement l'énergie de la fonction f qu'il faut contraindre; de plus comme la somme converge

$$|w|^2 |\hat{f}(w)|^2 = o(1) \Rightarrow |\hat{f}(w)| = o(|w|^{-1}) \quad (126)$$

ce qui indique le type de décroissance à l'infini des coefficients de Fourier³⁵. On peut étendre cette notion en requérant des contraintes sur les dérivées jusqu'à l'ordre m , alors on définit la **régularité de Sobolev**:

$$\sum_w |w|^{2m} |\hat{f}(w)|^2 < \infty \Rightarrow |\hat{f}(w)| = o(|w|^{-m}) \quad (127)$$

35. NDJE: rappel $o(1) \ll 1$ pour un mathématicien.

Comme la fonction f est de carré sommable, on écrit plutôt la contrainte sous la forme³⁶.

$$\sum_w (1 + |w|^{2m}) |\hat{f}(w)|^2 = A < \infty \quad (128)$$

Ceci étant dit, la conclusion est que l'énergie à haute fréquence va décroître d'autant plus vite que m est grand. Donc, reprenons le calcul de l'erreur

$$\begin{aligned} \|f(x) - \tilde{f}(x)\|^2 &= \frac{1}{(2\pi\Delta)^d} \sum_{|w| \geq C_\epsilon} |\hat{f}(w)|^2 < \sum_{|w| \geq C_\epsilon} \frac{(1 + |w|^{2m})}{C_\epsilon^{2m}} |\hat{f}(w)|^2 \\ &< A/C_\epsilon^{2m} = \epsilon^2 \end{aligned} \quad (129)$$

La contrainte sur C_ϵ et K (le nombre de neurones) s'écrit alors

$$C_\epsilon = A^{1/2m} \epsilon^{-1/m} \Rightarrow K = A^{d/2m} \epsilon^{-d/m} \quad (130)$$

Le nombre de coefficients à ajuster lui croît selon

$$N_\sigma \propto K^2 \propto \epsilon^{-2d/m} \quad (131)$$

Rappelons-nous que l'on veut rendre ϵ le plus petit possible, donc pour que K n'explose pas en grande dimension (cf. d grand) l'on ne pourra approximer que des fonctions de très grande régularité (cf. m grand). Autrement dit, requérir des régularités sur les dérivées d'ordres m donne une dimension apparente du problème donnée par d/m .

Dans les années 1988-94, on a essayé d'obtenir de meilleures bornes supérieures de l'équation 129. Ici, S. Mallat est passé par l'intermédiaire des sinus/cosinus, au lieu de faire une analyse directement avec les ReLU pour échantillonner la fonction f , donc ce n'est pas optimal, mais intentionnel de sa part pour la clarté de l'exposé. La réponse de ces études est que OUI, au lieu de nécessiter K^2 coefficients, on peut très bien se contenter de K coefficients. Mais cela ne change pas conceptuellement la donne, car $d \approx 10^6$. Ceci dit donnons la forme définitive du théorème sur les fonctions de Sobolev.

Théorème 5. Maiorov (1999): Si f a m dérivées au sens de Sobolev alors, il existe des

36. NDJE: on trouve aussi pour des fonctions de L^2 la contrainte $\|f\|_{L^2,m}^2 = \sum_w (1 + |w|^2)^m |\hat{f}(w)|^2 = A < \infty$ qui indiquerait que f est un élément de l'espace de Hilbert H^m (cf. fonction de L^2 satisfaisant cette contrainte de Fourier), mais celle donnée par S. Mallat convient tout aussi bien.

sigmoïdes telles que le nombre total de coefficients à contrôler est de l'ordre de

$$K \approx \epsilon^{-(d-1)/m} \quad (132)$$

Mais ce qui est très joli, c'est qu'on **NE PEUT PAS FAIRE MIEUX!**, c'est-à-dire qu'il n'existe aucun schéma (linéaire ou non-linéaire) qui peut battre cette loi d'échelle. Donc, on peut se dire “**fin de l'histoire**”, on a trouvé la machine universelle d'approximation. **Mais le problème**, est toujours le même, si on veut **diminuer l'erreur par 2**, alors on doit multiplier le nombre de neurones par un facteur colossal selon la loi:

$$\epsilon \rightarrow \epsilon/2 \Rightarrow K \rightarrow 2^{(d-1)/m} K \quad (133)$$

Donc, ce résultat, certes joli de math. fonda., est totalement impraticable sauf dans des cas de dimensionnalité réduite! Il faut envisager d'autres types de régularité bien plus grande et au-delà de l'Analyse de Fourier pour les cas de très grande dimension qui rendrait le résultat moins pessimiste.

6.4.2 Autres types de régularité

Un commentaire de S. Mallat: il y a un aspect “piègeux” des maths, c'est que l'on aboutit à des théorèmes comme celui de Maiorov, et qu'on en oublie les hypothèses (non pas du théorème lui-même) du cadre conceptuel général dans lesquelles elles s'inscrivent. Ainsi, les conséquences du théorème de Maiorov ont eu pour effet pour la grande majorité des personnes travaillant dans le domaine, de considérer que les réseaux de neurones à 1 ou plusieurs couches ne seront pas un sujet d'avenir.

Donc, il faut plutôt se reposer les questions de base de l'origine du problème qui ont suggéré de poser l'hypothèse de régularité de Sobolev, pour de nouveau se pencher sur la notion de régularité elle-même. On peut citer dans ce nouveau cadre de réflexion, un papier dont la première version date de 2014 et revu en Oct. 2017 de **Francis Bach** (ENS/Inria): “*Breaking the curse of dimensionality with Convex Neural Networks*”³⁷.

Rappelons-nous que dans ce type de problème, il y a 3 facettes à considérer:

^{37.} source: 2014arXiv1412.8690B, voir une présentation https://www.di.ens.fr/~fbach/fbach_cifar_2014.pdf

- **l'Approximation** : si on a un oracle qui donne les poids, quelle est l'erreur que l'on fait en approximant la fonction $f(x)$?
- **l'Optimisation** : quel algorithme pour obtenir les poids?
- **l'Estimation**: on ne dispose que N exemples en nombre “raisonnable”, comment faire?

Ceci dit, on va se pencher sur le premier problème, celui de l'Approximation, car il conditionne le reste en définitive. Quelles sont les hypothèses sur f que l'on peut faire pour que le problème d'Approximation ait une solution raisonnable? F. Bach a refait la liste des hypothèses “classiques” intéressantes: il y en a 3.

Dans la première hypothèse (Réduction de la dimension): on considère, l'ensemble Ω tel que

$$x \in \Omega \subset \mathbb{R}^d \quad \text{avec } \dim(\Omega) = s \ll d \quad (134)$$

C'est-à-dire que Ω est une variété de \mathbb{R}^d . La version la plus simple est celle d'une variété linéaire, alors

$$f(x) = g(\mathbf{W}^T x) \quad \text{avec } \text{rang}(\mathbf{W}) = s < d \quad (135)$$

c'est-à-dire que \mathbf{W} est une matrice $d \times s$. Par ce biais, le nombre de variables pertinentes est en fait s . Dans ce cas de figure, comme on l'a esquissé dans la section précédente, les théorèmes d'Universalité et de Maiorov s'appliquent avec le remplacement de d par s (*nb. cela s'apparente à de la feature reduction*):

$$\epsilon \approx K^{-(m/(s-1))} \quad (136)$$

Il y a des types de problèmes qui se prêtent naturellement à cette réduction de dimensionnalité: ex. mesures à différents points d'un bras articulé, car les articulations apportent des contraintes structurelles.

La seconde hypothèse utilise “la Séparabilité des interactions”. Ceci se retrouve souvent dans les problèmes de Physique. Par exemple dans un cas probabiliste où l'on étudie des Modèles de Markov:

$$x \in \mathbb{R}^d, \quad f(x) = \sum_{j=1}^J f_j(x_i; i \in I_j) \quad (137)$$

c'est-à-dire que f se décompose en une somme de fonctions qui chacune ne dépendent

que d'un petit nombre de variables. On peut imaginer une image dans laquelle on fait des traitements locaux qui ne concerne qu'un petit nombre de pixels pour chaque traitement. En Physique, on retrouve ce type de découpage quand on peut négliger les interactions à longue portée. Pour un problème de probabilité de Markov, c'est comme si on n'avait besoin que de connaître les probabilités conditionnelles locales. Donc, cette hypothèse de séparabilité est cruciale, et se retrouve dans beaucoup de domaines. Elle permet de casser la malédiction de la dimensionnalité.

Si les I_j sont tous de dimension s , on a donc J problèmes de dimension s . Dans ce cas le nombre de "neurones" est de l'ordre de

$$\epsilon \approx J \times K^{-(m/s)} \Leftrightarrow K \approx (\epsilon/J)^{-m/s} \quad (138)$$

C'est une situation qui se retrouve dans beaucoup de problèmes. Ce que regarde Francis Bach, c'est le cas où

$$f(x) = \sum_{j=1}^J f_j(\langle x, w_j \rangle) \quad (139)$$

c'est-à-dire que l'on projette les d variables sur un produit scalaire, donc $s = 1$, et dans ce cas-là

$$\epsilon \approx J \times K^{-(m)} \Leftrightarrow K \approx (\epsilon/J)^{-m} \quad (140)$$

La troisième hypothèse utilise "la Sparsity ou Parcimonie" Dans ce cadre³⁸, on se pose la question de savoir si on peut réduire le nombre de descripteurs de la fonction $f(x)$. On définit un dictionnaire $\mathcal{D} = \{g_m\}_{m \leq M}$ avec des features/patterns/descripteurs g_m potentiellement en grand nombre; cependant pour obtenir une approximation \tilde{f} de f nous allons pouvoir en prendre **qu'un petit nombre**:

$$\tilde{f}(x) = \sum_{m \in I} \alpha_m g_m(x), \quad \|f - \tilde{f}\| \leq \epsilon \text{ avec } Card\{I\} = K \text{ tel que } \epsilon \sim K^{-\alpha} \quad (141)$$

c'est-à-dire, on prend le plus petit nombre de descripteurs (K) et en même temps, on veut une erreur d'approximation qui décroisse vite si on augmente ce nombre.

Rappelons que dans une analyse de Fourier, on avait la même attitude, à savoir, on ne garde pas toutes les fréquences, mais on se restreint aux basses fréquences $|w| < C$

38. NDJE: Notons que cette notion a été abordée en 2018: Analyse en Ondelettes.

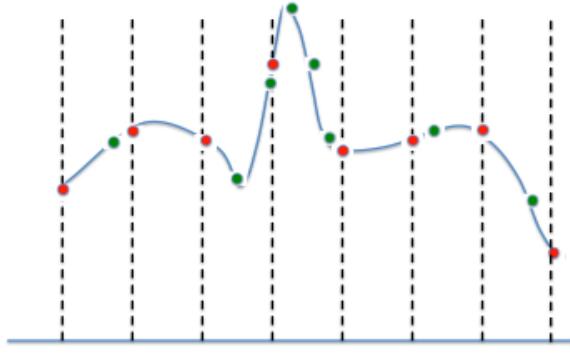


FIGURE 52 – Exemple de la différence entre un échantillonnage à pas régulier peu importe la fonction à traiter (points rouges), et un échantillonnage adaptatif qui tient compte des variations brutales de la fonction à approximer (points verts).

(cf. la régularisation du Perceptron). Cependant, la constante C dépend uniquement de la nature de la dérivable de f à travers m , mais elle ne dépend pas fondamentalement de la fonction f à proprement parlée: deux fonctions dérivables 3 fois auront quasiment la même valeur de C .

Dans l'hypothèse de parcimonie, on se laisse **la possibilité de s'adapter à la fonction f** , en particulier I dépend de f . Donc, on introduit de **l'adaptivité** et surtout les coefficients α_k vont dépendre de f , c'est-à-dire que l'on introduit naturellement de la **non-linéarité**. L'adaptabilité peut se comprendre assez facilement, si l'on prend le cas de la figure 52 où l'échantillonnage à pas constant est moins efficace que l'échantillonnage adaptatif pour capturer la variabilité brusque de la fonction qui lui est propre.

Un résultat de **A. R. Barron** a été proposé en 1991: au lieu de prendre la contrainte de Sobolev (Eq. 127) on prend une hypothèse plus faible

$$\sum_w |w| |\hat{f}(w)| < \infty \quad (142)$$

alors le nombre de termes (neurones) va décroître/croître selon

$$K \propto \frac{1}{\epsilon} \quad (143)$$

c'est-à-dire que l'on a **supprimé la malédiction de la dimensionnalité!** Ce résultat est maintes fois cité dans les livres sur le ML. Ce résultat paraît miraculeux, car rappelons l'hypothèse de Sobolev:

$$\sum_w |w|^{2m} |\hat{f}(w)|^2 < \infty \quad (144)$$

À première vue, on ne voit pas bien le changement. Or, **ce n'est pas du tout la même chose, parce qu'il faut se rappeler que $w \in \mathbb{Z}^d$** (voir le Théorème d'Universalité), et il y a une notion de parcimonie "cachée" qui en Fourier n'est quasiment jamais vérifiée. Voyons, comment la nouvelle hypothèse de A.R. Barron fonctionne. On va même faire plus simple pour montrer le lien avec la parcimonie. Rappelons que si $f \in L^2$ donc d'énergie finie, alors

$$\|f\|_{L^2}^2 = \sum_w |\hat{f}(w)|^2 < \infty \quad (145)$$

Il s'agit une contrainte forte.

Théorème 6. *Si on suppose*

$$A = \sum_{w \in \mathbb{Z}^d} |\hat{f}(w)| < \infty$$

alors

$$\exists \{w_k\}_{k \leq K} \text{ avec } K = (A/\epsilon)^2 / \|f_K\|^2 \quad / \quad f_K(x) = \sum_{k=1}^K \hat{f}(w_k) e^{-iw_k x}, \quad \|f - f_K\|^2 \leq \epsilon^2 \quad (146)$$

L'idée est de montrer que l'on se débarrasse de l'exposant d . En fait la contrainte est une **norme L1**, or dès que cette norme est utilisée, il y a de la **sparsité**³⁹. Pour la démonstration, on ordonne les coefficients de Fourier par ordre décroissant

$$\mathcal{F} = \{\hat{f}(w_k) / |\hat{f}(w_k)| \geq |\hat{f}(w_{k+1})|\} \quad (147)$$

Utilisons alors le résultat suivant:

Lemme 2.

$$A = \sum_{w \in \mathbb{Z}^d} |\hat{f}(w)| \Rightarrow \hat{f}(w_k) \in \mathcal{F}, \quad |\hat{f}(w_k)| \leq \frac{A}{k} \quad (148)$$

39. NDJE: voir les types de régularisation L1, L2, dans le cours de 2018: Classification/Régression en grande dimension

Si on a un mécanisme qui fait que l'on obtient une approximation en ne prenant que des fréquences appartenant à un ensemble I , alors l'erreur vient des fréquences non-prises en compte:

$$f_I = \sum_{w \in I} \tilde{f}(w) e^{-iw \cdot x} \Rightarrow \|f - f_I\|^2 = \sum_{w \notin I} |\tilde{f}(w)|^2 \quad (149)$$

Donc, maintenant pour sélectionner les bonnes fréquences, on va prendre celles pour lesquelles les coefficients de Fourier sont les plus grands, ainsi

$$f_K = \sum_{k=1}^K \tilde{f}(w_k) e^{-iw_k \cdot x} \quad (150)$$

c'est-à-dire que $I = \mathcal{F}_K \subset \mathcal{F}$ pour lequel on constraint $k \leq K$. Alors

$$\|f - f_K\|^2 = \sum_{k>K} |\tilde{f}(w_k)|^2 \leq \sum_{k>K} \frac{A^2}{k^2} \leq A^2 \int_K^\infty \frac{dx}{x^2} = \frac{A^2}{K} \quad (151)$$

et donc si on prend $K = A^2/\epsilon^2$ alors

$$\|f - f_K\|^2 \leq \epsilon^2 \quad (152)$$

Ce qui est le résultat du théorème. Donc, l'approximation parcimonieuse consiste à sélectionner les coefficients de Fourier les plus grands et si les coefficients décroissent assez vite alors la contrainte sur K est indépendante de la dimension d .

Il reste à démontrer le lemme 2 sur la norme L1 qui conditionne la décroissance des coefficients. On a donc (les w_k sont les fréquences dont les coefficients de Fourier sont ordonnées, cf. \mathcal{F}), $\forall P$

$$\begin{aligned} A &= \sum_{w \in \mathbb{Z}^d} |\hat{f}(w)| = \sum_{k=1}^P |\tilde{f}(w_k)| + \sum_{k>P} |\tilde{f}(w_k)| \geq \sum_{k=1}^P |\tilde{f}(w_k)| \\ &\geq P \times |\tilde{f}(w_P)| \end{aligned} \quad (153)$$

Donc, on a bien

$$\forall P \quad |\tilde{f}(w_P)| \leq \frac{A}{P} \quad (154)$$

ce qui est le résultat du lemme.

Ce résultat est plus général que le cas particulier de la transformée de Fourier; dès lors que les coefficients dans une base orthogonale sont contrôlés par une norme L1. En fait A.R Baron utilise la condition

$$\sum_w |w| |\hat{f}(w)| < \infty \Rightarrow K \propto 1/\epsilon \quad (155)$$

et non $1/\epsilon^2$ comme ci-dessus, mais l'idée est la même.

Cependant, pourquoi le théorème miraculeux de Baron ne s'applique pas aux vrais problèmes? Pour la simple et bonne raison qu'à de très rares cas près, les fonctions $f(x)$ (en classification d'images, de sons, etc) ne sont pas parcimonieuses en Fourier. Donc, c'est un très joli théorème mais qui ne s'applique pas. Au bilan, dans les années 2000, avec toutes les formes de régularité envisagées, le résultat d'augmenter le nombre de couches n'est pas très clair: est-ce que cela va changer la donne?

6.4.3 Augmenter le nombre de couches: c'est mieux!

Il y a un joli papier qui montre qu'au contraire qu'augmenter le nombre de couches est mieux. Il s'agit d'un article de **Ronen Eldan, Ohad Shamir** (2015-16) : “*The power of depth of feedforward networks*”⁴⁰. Les auteurs montrent qu'il existe une fonction radiale simple de \mathbb{R}^d exprimable avec un réseau à 3 couches (2 couches cachées), qui ne peut en aucun cas être exprimable avec un réseau à 2 couches (1 couche cachée) sauf à utiliser un nombre exponentiel de neurones. Cette fonction est la transformée de Fourier de la boule de volume unité de rayon R_d en dimension d :

$$\phi(x) = \left(\frac{R_d}{\|x\|} \right)^{d/2} J_{d/2}(2\pi R_d \|x\|) \quad (156)$$

Donc, par ce contre-exemple du théorème d'universalité (attention cela ne va pas dire que ce dernier est faux, il faut faire attention aux hypothèses.): **mieux vaut augmenter le nombre de couches que d'augmenter le nombre de neurones sur un réseau plus petit.** Pour construire leur fonction, Eldan et Shamir utilisent le fait que pour approximer leur fonction, il faut **beaucoup de puissance à haute fréquence**, ce qui tue l'approximation

40. <https://arxiv.org/abs/1512.03965>

$|w| < C$ dans la démonstration du réseau à 1 couche cachée. Et, ils se servent de la **symétrie de révolution à haute fréquence** pour qu'avec l'ajout de 1 couche cachée en plus cela se passe très bien.

Donc le résultat est intéressant pour concevoir l'idée que des réseaux multi-couches sont meilleurs. Cependant, les fonctions de 'leur "démonstration"' ne sont pas réalistes pour autant.

7. Optimisation des Réseaux de Neurones

7.1 Introduction

Souvenons-nous du schéma de la figure 1 représentant l'algorithme paramétré qui à x fait correspondre \tilde{y} la sortie du réseau de neurones. En fait \tilde{y} dépend de la paramétrisation θ , et on le note alors \tilde{y}_θ . Dans le cas de la **Classification**, on a des couples entrée-sortie (x, y) avec y un **index** qui est bien différent du cas de la **Régression** où $y \in \mathbb{R}^p$. Dans le cas des **index**, il n'y a pas de topologie naturelle, on peut indexer de multiples façons. Ainsi la **notion de continuité de la fonction $f(x)$ sous-jacente** que doit approximer $\tilde{y}_\theta = f_\theta(x)$ n'est **pas évidente du tout !**

En terme d'optimisation, on définit la notion de risque (voir cours de 2018)

$$r(y, \tilde{y}) = \begin{cases} 0 & y = \tilde{y} \\ 1 & y \neq \tilde{y} \end{cases} \quad (157)$$

L'erreur totale sur une base de données empirique qui sert pour l'apprentissage par exemple est définie à partir du risque

$$\tilde{R}(\theta) = \frac{1}{n} \sum_{i=1}^n r(f_\theta(x_i), y_i) \quad (158)$$

On espère que ce risque empirique converge vers le risque moyen quand n tend vers ∞ :

$$R(\theta) = E_{(x,y)}[r(y, f_\theta(x))] \quad (159)$$

Quel est le problème? Certes, on a des algorithmes de descente de gradient pour minimiser le risque, mais cela suppose que la fonction $\tilde{R}(\theta)$ soit **differentiable**. Cependant, en classification, le risque $r(y, \tilde{y})$ n'est pas du tout différentiable, et cela s'ajoute au fait que les y sont des **index**. Le schéma d'attaque du problème va donc de trouver des quantités qui sont régulières, et que l'on va estimer avec des f_θ régulières ne dépendant pas d'un trop grand nombre de paramètres. Puis, on définit un risque qui va se différentier selon θ .

7.2 L'approche de Bayes et le principe du maximum de vraisemblance

7.2.1 Transformation du problème via Bayes

Soit donc le risque moyen introduit à la section précédente:

$$R = E_{(x,y)}[r(y, \tilde{y}(x))] \quad (160)$$

on peut se poser la question de savoir quelle est la meilleure approximation $\tilde{y}(x)$ que l'on a de y ? À cette fin, on peut utiliser les probabilités conditionnelles pour écrire que

$$R = E_x \left(\sum_y r(y, \tilde{y}(x)) p(y|x) \right) \quad (161)$$

avec $\sum_y p(y|x) = 1$. Or, $y = \{0, 1\}$ et on veut minimiser le risque R , donc on voudrait $r(y, \tilde{y}(x)) = 0$ dans le cas où la probabilité $p(y|x)$ est maximale, donc

$r(y, \tilde{y}(x)) = 0 \text{ (càd : } y = \tilde{y} \text{)} \quad \text{qd} \quad p(\tilde{y}|x) = \max_y p(y|x)$

(162)

C'est le **Classificateur de Bayes**: $\tilde{y}(x)$ est la classe la **plus probable** étant donné l'observation x . Ce qui implique que l'on choisisse la classe qui maximise $p(y|x)$ et de ce fait $\tilde{y}(x)$ **minimise l'erreur en moyenne**. C'est le **classificateur idéal**, mais pour le mettre en œuvre, il faut pouvoir connaître la distribution de probabilité $p(y|x)$. **Donc, on transforme un problème d'approximation de y (index) en l'approximation des $p(y|x)$ fonction continue (pour tout y)**. Remarquer que si l'on a 10 valeurs de y possibles (cf. 10 classes), alors il y a 10 fonctions de x à approximer: les $p(y|x) = g_y(x)$. Or, ces $g_y(x)$ sont **des fonctions**

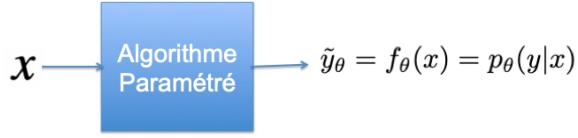


FIGURE 53 – Le réseau de neurones vu comme une "machine" qui approxime des densités de probabilité conditionnelles (normalisées).

bien régulières et bien structurées.

Le Réseau de Neurones est vu selon cet angle comme une "machine" qui approxime des densités de probabilité conditionnelles (normalisées). Le schéma qui se dégage est celui de la figure 53.

7.2.2 Maximum de vraisemblance

Donc, on veut faire en sorte que $p_\theta(y|x)$ soit une approximation de la vraie distribution de probabilité $p(y|x)$. Il nous faut donc une métrique qui nous donne une estimation de l'erreur d'approximation. C'est dans ce cadre que le **maximum de vraisemblance** va apparaître comme cas particulier de minimisation de l'erreur entre $p_\theta(y|x)$ et $p(y|x)$, à travers la distance particulière de la **divergence de Kullback-Leibler**.

Le cadre est le suivant, étant donné une famille de Data $= \{x_i, y_i\}_{i \leq n}$ comment approximer $p(y|x)$? Si la famille des échantillons est issue de $p_\theta(y|x)$ alors le principe du maximum de vraisemblance, c'est choisir $\theta = \theta^*$ tel que la probabilité des données soit maximum pour $p_{\theta^*}(y|x)$. On suppose en outre que les données sont toutes **indépendantes**.

La probabilité d'obtenir y_i sachant x_i selon p_θ est par définition $p_\theta(y_i|x_i)$, si les données sont toutes **indépendantes**, alors la probabilité d'obtenir les n échantillons de la famille, n'est autre que le produit des probabilités individuelles soit

$$\prod_{i=1}^n p_\theta(y_i|x_i) \tag{163}$$

et donc

$$\begin{aligned}\theta^* &= \operatorname{argmax}_{\theta} [\prod_{i=1}^n p_{\theta}(y_i|x_i)] = \operatorname{argmax}_{\theta} [\log (\prod_{i=1}^n p_{\theta}(y_i|x_i))] \\ &= \operatorname{argmax}_{\theta} \left[\frac{1}{n} \sum_{i=1}^n \log p_{\theta}(y_i|x_i) \right]\end{aligned}\quad (164)$$

(*nb. notons que le facteur de normalisation n'a pas d'importance*).

Or, la moyenne des “log” peut être exprimer comme l’espérance des “log” sur la distribution empirique de la famille d’échantillons, et donc le principe du maximum de vraisemblance se traduit par

$$\boxed{\theta^* = \operatorname{argmax}_{\theta} E_{\{x,y\} \sim \text{Data}} (\log p_{\theta}(y|x))} \quad (165)$$

Or, ce Principe de Vraisemblance peut être vu comme une minimisation d'une “distance”: la divergence de Kullback-Leibler.

7.2.3 Divergence de Kullback-Leibler

La divergence de Kullback-Leibler⁴¹ se définit par

$$D_{KL}(p||q) = E_p \left(\log \frac{p}{q} \right) = \int p(x) \log \frac{p(x)}{q(x)} dx \quad (166)$$

Ce n'est pas une “distance” stricto sensu, mais elle a d'importantes propriétés:

- $D_{KL}(p||q) \geq 0$ qui se démontre en notant que $\forall x > 0, \log x \leq x - 1$ (égalité pour $x = 1$);
- elle n'est pas symétrique (ce n'est donc pas une distance);
- une manière de la comprendre: imaginons des symboles produits par $p(x)$ et que l'on veuille coder ces symboles de manière optimale. Cependant, on vous donne la distribution $q(x)$ à la place de $p(x)$, donc vous allez avoir une inefficacité de codage: $D_{KL}(p||q)$ est exactement cette inefficacité. Pourquoi? parce que le code

41. NDJE: La divergence de Kullback-Leibler ou divergence K-L ou encore entropie relative, doit son nom à Solomon Kullback et Richard Leibler, deux cryptanalystes américains de la NSA qui ont inventé cette notion dans les années 50.

optimal de Shannon associé à $p(x)$ est de taille $-\log(p(x))$, or, on a $q(x)$ à la place donc on obtient le code optimal $-\log(q(x))$ donc, la différence entre les deux fait apparaître le $\log(p/q)$ et en moyenne cela donnera $p \log(p/q)$.

- il y a beaucoup d'autres interprétations de cette "distance" qui apparaît partout en théorie de l'information (ML compris).
- si $D_{KL}(p||q) = 0$ alors $p = q$ (c'est le cas particulier du premier point ci-dessus).

Donc, ça a du sens d'utiliser cette divergence pour optimiser un réseau de neurones qui tente d'approximer $p(y|x)$ par $p_\theta(y|x)$.

Théorème 7. *Le maximum de vraisemblance va minimiser*

$$D_{KL}(p_{\text{Data}}(y|x)||p_\theta(y|x))$$

La démonstration est assez simple, il suffit d'écrire ce que vaut la divergence ci-dessus:

$$\begin{aligned} D_{KL}(p_{\text{Data}}||p_\theta) &= E_{p_{\text{Data}}} \left(\log \frac{p_{\text{Data}}}{p_\theta} \right) \\ &= E_{p_{\text{Data}}} (\log p_{\text{Data}}(y|x)) - E_{p_{\text{Data}}} (\log p_\theta(y|x)) \end{aligned} \quad (167)$$

et on veut minimiser cela par rapport à θ . Or, seul le second terme dépend de θ , et le minimiser, revient à maximiser l'opposé ce qui est exactement la définition du maximum de vraisemblance.

Donc prendre le paramètre θ qui maximise la vraisemblance est équivalent à trouver la distribution de probabilité qui approxime au mieux la distribution empirique (c-à-d obtenue sur les données elles-mêmes) selon la métrique de la divergence de Kullback-Leibler.

7.2.4 Relation avec les modèles bayésiens

7.2.4.1 Un commentaire en préliminaire

Dans la littérature sur le ML, il y a constamment deux points de vue.

- **Un point de vue purement déterministe** qui part du constat que le réseau va fournir la réponse $y = \tilde{f}(x)$ laquelle doit au mieux approximer la fonction $f(x)$.

On est donc dans un domaine d'approximation de fonctions qui est totalement déterministe (cf. pas de probabilité).

- Et **un point de vue exclusivement probabiliste** et bayésien par essence qui dans ce cadre essaye d'approximer des densités de probabilité.

En fait **ces deux points de vue sont équivalents**, et l'on passe rapidement entre l'un et l'autre dans une même publication, pour pouvoir identifier le bon "objet" que l'on va estimer. Par exemple, dans le cas d'une régression avec des fonctions "bien" régulières, on n'a pas besoin du point de vue probabiliste; ce qui n'est pas le cas de la classification où rien n'est régulier au départ, et l'usage des distributions de probabilité est une manière de reformuler le problème dans un cadre régulier.

7.2.4.2 Approche bayésienne vs déterministe

Dans l'approche bayésienne, on a des **données**, $\text{Data} = \{x_i, y_i\}_{i \leq n}$, et des paramètres θ qui modélisent ces données. De plus, on aimerait obtenir θ^* qui maximise la probabilité de θ étant donné l'ensemble Data:

$$\theta^* = \operatorname{argmax}_{\theta} p(\theta|\text{Data}) \quad (168)$$

C'est le **maximum a posteriori**. La loi de Bayes nous dit

$$p(\theta|\text{Data}) = \frac{p(\text{Data}|\theta) \pi(\theta)}{p(\text{Data})} \quad (169)$$

avec $p(\text{Data}|\theta) \equiv \mathcal{L}(\theta)$ le **Likelihood**, $\pi(\theta)$ le **prior** sur θ , et $p(\text{Data})$ la probabilité d'obtenir les données. On voudrait calculer le max sur θ , donc seul compte le numérateur.

Si maintenant, on n'a aucune information⁴² *a priori* sur θ alors $\pi(\theta) = Cte$, et donc la valeur de θ est telle que

$$\theta^* = \operatorname{argmax}_{\theta} p(\text{Data}|\theta) \quad (170)$$

qui n'est autre que le **principe du maximum de vraisemblance qui maximise le Likeli-**

42. fNDJE: la notion de prior "non informatif" est très délicate. Le choix d'un prior "constant" qui semble aller de soit, n'est pas si évident que ça, puisque par exemple, il n'est pas invariant par changement de variables. Pour se faire une idée du problème, voir par exemple l'introduction des cours <https://people.eecs.berkeley.edu/~jordan/courses/260-spring10/lectures/lecture1.pdf> et la référence aux travaux de **Harold Jeffreys**. Ceci dit, on ne travaille jamais avec des priors constants et cela est bien montré dans la suite, parce que les priors sont une facette des termes de régularisation.

hood. La formulation bayésienne et le principe du maximum de vraisemblance ne coïncide que si $\pi(\theta) = cte$.

Si par contre $\pi(\theta) \neq Cte$, alors maximiser la probabilité revient à maximiser le log de la probabilité, ainsi

$$\theta^* = \operatorname{argmax}_{\theta} [\log p(\text{Data}|\theta) + \log \pi(\theta)] \quad (171)$$

Le terme $\log \pi(\theta)$ est une **pénalisation** qui se retrouve en fait dans les réseaux de neurones lors de la définition de la fonction de coût. En effet, on écrit

$$\tilde{R}(\theta) = \frac{1}{n} \sum_{i=1}^n r(f_{\theta}(x_i), y_i) + C(\theta) \quad (172)$$

et la fonction $C(\theta)$ vient de la connaissance *a priori* sur θ qui se formule sous forme de contrainte comme utiliser la norme L2 sur les poids (qui sont ici représentés par la notation générique θ). En d'autres termes imposer que la norme L2 sur les poids des paramètres ne soit pas trop grande **est une information a priori (cf. $\pi(\theta) \neq Cte$) pour contraindre la valeur de θ** .

Ainsi, **on voit ici qu'exprimer une fonction de coût dans un domaine purement déterministe avec une régularisation $C(\theta)$, peut se voir également comme une version purement bayésienne avec une probabilité a priori sur θ .**

7.2.4.3 Cas de la Régression

En fait, on retombe sur les mêmes idées que précédemment. En effet, dans ce cas on aurait $y = f(x)$ que l'on veut approximer avec $y_{\theta} = f_{\theta}(x)$, et on veut minimiser le risque empirique quadratique en moyenne

$$\min_{\theta} \sum_{i=1}^n |y_i - f_{\theta}(x_i)|^2 \quad (173)$$

Quel est le lien avec le Maximum de Vraisemblance? Le réseau calcule $f_{\theta}(x)$ et on voudrait lui associer une distribution de probabilité $p_{\theta}(y|x)$. En général, on utilise une gaussienne centrée sur la valeur calculée $f_{\theta}(x)$ avec une erreur σ :

$$p_{\theta}(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-f_{\theta}(x))^2}{2\sigma^2}} \quad (174)$$

Si maintenant, on applique un maximum de vraisemblance, ce qui revient à minimiser la distance de Kullback-Leibler

$$\begin{aligned} \min_{\theta} D_{KL}(p_{\text{Data}} || p_{\theta}) &= -\min_{\theta} E_{p_{\text{Data}}} (\log p_{\theta}(y|x)) \\ &= \min_{\theta} E_{p_{\text{Data}}} ((y - f_{\theta}(x))^2) \\ &= \min_{\theta} \sum_{i=1}^n |y_i - f_{\theta}(x_i)|^2 \end{aligned} \quad (175)$$

Donc, minimiser une erreur (cout) quadratique (cf. une norme L2), c'est aussi dire que l'on prend le maximum d'une vraisemblance gaussienne centrée sur la sortie du réseau.

Les deux approches sont donc équivalentes et ici cela n'apporte pas grand-chose, mais dans le cas de la classification, où il n'y a pas *a priori* de métrique, on a vu comment la divergence de Kullback-Leibler (ou le Maximum de Vraisemblance) apporte la solution.

7.3 Mise en œuvre pour un réseau de neurones (classification)

7.3.1 Introduction

La technique est assez générique et donc s'applique à une large classe de classificateurs. On se rappelle la figure 53 avec θ représentant tous les coefficients des matrices linéaires W_k et tous les biais b_k (et ici, on a 1 couche). Une des propriétés cependant des probabilités $p_{\theta}(y|x)$ est que

$$\sum_y p_{\theta}(y|x) = 1 \quad (176)$$

Or, les sorties du réseau n'ont aucune raison d'être normalisées, **on doit donc ajouter une étape de normalisation**. Une fois cette étape effectuée, alors le meilleur \tilde{y} , c'est-à-dire la meilleure classe (pour un x donné) est celui ou celle qui satisfait

$$\tilde{y} = \operatorname{argmax}_y [p_{\theta^*}(y|x)] \quad (177)$$

C'est une approche en tout point bayésienne.

7.3.2 Introduction du softmax

Maintenant, comment réaliser la normalisation? La fonction **softmax** va associer à la sortie, mettons $z_y(x)$, c'est-à-dire étant donné x la valeur numérique z pour l'index y , la valeur

$$z_y(x) \xrightarrow{\text{softmax}} \frac{e^{z_y(x)}}{\sum_{y'} e^{z_{y'}(x)}} = p_\theta(y|x) \quad (178)$$

On prend cette définition et non la simple expression $z_y / \sum_{y'} z_{y'}$ car les z_y ne sont pas forcément positifs, mais surtout s'il y a une valeur prépondérante, alors l'exponentielle renforce cet état. Il y a un autre avantage que nous allons examiner à présent. D'après la section précédente, on veut minimiser la divergence D_{KL} qui s'exprime selon

$$\sum_i -\log p_\theta(y_i|x_i) \quad (179)$$

avec l'expression de la probabilité $p_\theta(y_i|x)$ donnée par le **softmax**. Donc, on veut **minimiser** (*en fait z_y dépend de θ mais la notation devient lourde*)

$$L(\theta) = \sum_i -\log \left(\frac{e^{z_{y_i}(x_i)}}{\sum_{y'} e^{z_{y'}(x_i)}} \right) = -\sum_i \left[z_{y_i}(x_i) - \log \left(\sum_{y'} e^{z_{y'}(x_i)} \right) \right] \quad (180)$$

Cette fonction $L(\theta)$, pour un échantillon i donné, dépend par le premier terme directement de la sortie du réseau grâce au **softmax**, et d'un terme de normalisation. Le fait d'avoir $L(\theta)$ directement liée à $z_{y_i}(x_i)$ va permettre un conditionnement du problème de minimisation bien efficace. En effet, on veut minimiser $L(\theta)$ et donc maximiser $z_{y_i}(x_i)$ c'est-à-dire maximiser la probabilité de la réponse $y_\theta(x_i) = y_i$. L'autre terme ressemble au maximum des $z_{y'}(x_i)$ en particulier si ce maximum est bien détaché des autres valeurs

$$\log \left(\sum_{y'} e^{z_{y'}(x_i)} \right) \approx \max_{y'} [z_{y'}(x_i)] \quad (181)$$

Ainsi $L(\theta)$ revient à comparer $z_{y_i}(x_i)$ et $\max_{y'} [z_{y'}(x_i)]$:

$$L(\theta) \approx -\sum_i \left(z_{y_i}(x_i) - \max_{y'} [z_{y'}(x_i)] \right) \quad (182)$$

Donc, lors d'une itération de minimisation sur θ , si le maximum des $z_{y'}(x_i)$ est réalisé pour $y' = y_i$ alors la différence vaut 0, et on a gagné, car il n'y a pas besoin de faire d'update de θ . Le softmax permet de réaliser cela en ayant une fonction différentiable.

7.3.3 Optimisation: cas particulier de la classification par régression logistique

La régression logistique (en fait, il s'agit d'une classification) peut être vue comme un réseau de neurones à 1 seule couche non cachée dont la réponse z et une fonction linéaire de l'entrée x (cf. pas de fonction d'activation non-linéaire):

$$z = \mathbf{W}x + b \quad (183)$$

Si l'on prend la ligne y de cette matrice cela représente la réponse $z_y(x)$ selon

$$z_y(x) = \langle x, w_y \rangle + b \quad (184)$$

avec y un entier comme s'il était un index de classe. De plus notant que $\theta = \mathbf{W}$, on veut minimiser la fonction de coût suivante:

$$L(\mathbf{W}) = - \sum_i \left[(\langle x_i, w_y \rangle + b) - \log \left(\sum_{y'} e^{\langle x, w_{y'} \rangle + b} \right) \right] \quad (185)$$

L'interprétation géométrique est la suivante: première remarque w_y a la même dimension que x , ils évoluent dans le même espace. Donc les w_y représentent chacun une direction dans cet espace. Le classificateur va essayer de trouver la direction w_y qui coïncide pour la “classe” y_i avec la direction w_{y_i} autour de laquelle les x_i s'agrègent (voir figure 54), c'est l'effet du premier terme; l'autre terme de la fonction $L(\mathbf{W})$ va contraindre la direction w_y à ne pas être dans la direction des autres classes.

Que se passe-t-il si on a une erreur, c'est-à-dire un “outlier” pour l'échantillon i ? Par exemple, si on a 2 classes $\{-1, 1\}$ alors $w_1 = -w_{-1}$ (figure 55), et supposons que l'échantillon i soit classé -1 au lieu de 1 . La pénalité est égale à:

$$-(z_{y_i}(x_i) - \max_{y'} [z_{y'}(x_i)]) = -(z_1(x_i) - z_{-1}(x_i)) = -\langle x_i, w_1 \rangle + \langle x_i, w_{-1} \rangle = 2\langle x_i, w_{-1} \rangle \quad (186)$$

Or x_i est situé dans la direction de w_{-1} par rapport au plan séparateur, **on pénalise alors par (2 fois) la distance qu'il faut pour le ramener du bon côté du plan** (cf. on prend son symétrique par rapport au plan). C'est très similaire à ce que fait l'algorithme **SVM** pour le terme marge qui sert de régularisation (voir la section 4.2.4).

7.3.4 Pour un réseau de neurones MLP

Dans le cas d'un MLP, il y a plusieurs couches cachées pour lesquelles la transformation entre deux couches successives fait intervenir la relation suivante:

$$x_j = \sigma(\mathbf{W}_j x_{j-1} + b_j) \quad (187)$$

jusqu'à la couche J pour laquelle on obtient la représentation $x_J = \Phi(x)$, puis on agrège par une régression logistique pour obtenir un vecteur de classification par les **softmax**, puis enfin, on prend par exemple le **max** pour donner \tilde{y} (figure 56). On optimise **conjointement** la représentation des données ($\Phi(x)$) et le classificateur (logistique). L'information *a priori* est contenue dans l'architecture. Notons qu'il faut également beaucoup d'échantillons pour apprendre la représentation. Dans la section suivante, nous allons aborder la partie algorithmique des MLP.

8. Algorithmes d'optimisation des MLP

Nous allons aborder l'algorithme “Backprop” de rétro-propagation des gradients et la descente de gradients stochastique. Par la suite nous nous pencherons sur les démonstrations.

8.1 Calcul du gradient dans les MLP (Back-prop)

Voir l'article de 1986 Rumelhart, Hinton, Williams⁴³ qui certes exploitent les gradients de fonctions composées qui est une idée “simple”, mais qui dans le cadre des MLP s'avère

43. Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning representations by back-propagating errors. *Nature*, 323, 533–536.

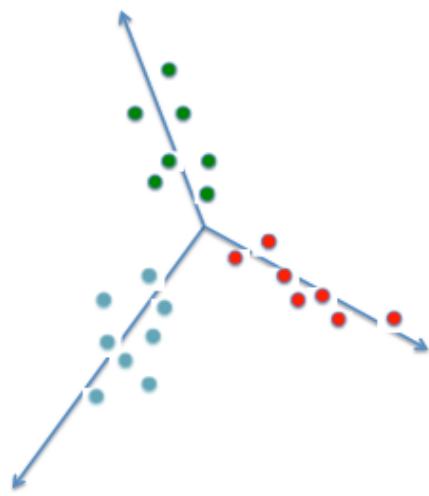


FIGURE 54 – Schématisation des vecteurs w selon lesquels les données s'agrègent.

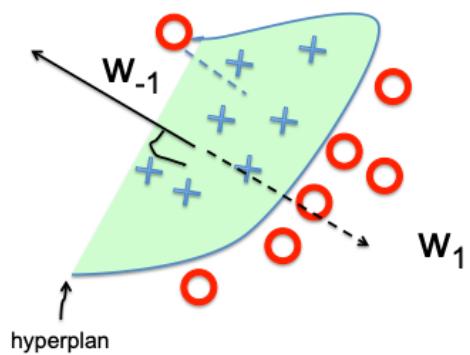


FIGURE 55 – Outlier dans le cas de 2 classes.

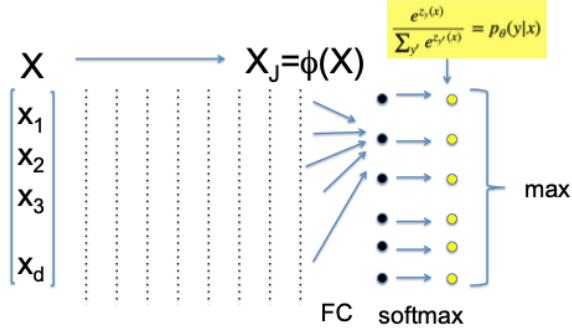


FIGURE 56 – Schématisation des différentes parties d'un MLP.

être très efficace⁴⁴.

8.1.1 Flux Forward & Backward

Le réseau multicouches peut être représenté comme une succession/empilement de couches ayant une entrée, une sortie et effectuant une opération F (voir figure 57). La première couche de neurones effectue l'opération (*nb. attention \mathbf{W} va souvent implicitement inclure le biais avec une convention d'ajout d'un 1 pour les x_i*):

$$x_1 = F_1(x_0, \mathbf{W}_1) = \sigma(\mathbf{W}_1 x_0 + b_1) \quad (188)$$

et ainsi de suite pour F_2, F_3 , etc, jusqu'à F_J qui donne la sortie x_J , laquelle est comparée par une fonction de coût (*loss*) à la sortie y d'un échantillon étiqueté. L'ensemble des paramètres du réseau (les \mathbf{W}_i, b_i) est noté θ dont certains peuvent être partagés éventuellement par plusieurs F_i . Donc, la fonction de perte globale moyenne qui dépend de θ , notée $\ell(\theta)$ est calculée sur un lot d'échantillons étiquetés $\{x_0^i, y^i\}_{i \leq N}$ comme suit

$$\ell(\theta) = \frac{1}{N} \sum_{i=1}^N \ell(x_J^i, y^i) \quad (189)$$

44. NDJE: J'avais signalé en 2017 que Werbos en 1974 avait discuté de cela, puis en 1982 il en a fait une application spécifique pour les réseaux de neurones. Puis LeCun et Parker en 85 avait publié sur des idées connexes. Mais les idées de minimisations des erreurs par descente de gradient datent de Cauchy 1847 et Hadamard 1908. Ainsi, trouver qui le premier à inventer l'algorithme de Back-Propagation est une histoire en soi.

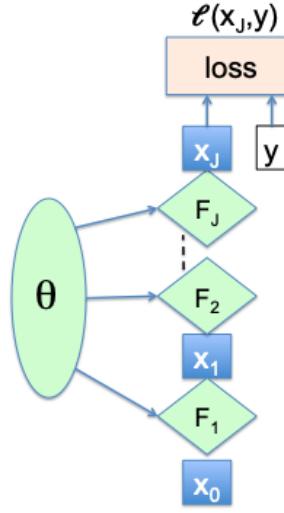


FIGURE 57 – Schématisation du flux "forward" (ici "montant") dans lequel on suit un échantillon x qui passe successivement dans les différentes couches du réseau, pour être comparé à y dans une fonction de coût $loss$. Les paramètres du réseau (notés θ) interviennent dans la description des opérations F à chaque couche.

où θ intervient dans le calcul des x_J^i . Pour trouver le minimum de cette fonction de coût, on utilise *la descente de gradient* qui à une étape n définit un jeu de paramètres, noté θ_n , et la valeur $n + 1$ est déduite en ajoutant une contribution qui est dirigée à l'opposé du gradient de $\ell(\theta)$ calculé avec θ_n . Ainsi

$$\begin{aligned} \theta_{n+1} &= \theta_n - \alpha \nabla_\theta \ell(\theta) \\ &= \theta_n - \alpha \frac{1}{N} \sum_{i=1}^N \nabla_\theta \ell(x_J^i, y^i) \end{aligned} \quad (190)$$

Donc, le problème est de calculer $\nabla_\theta \ell(x_J^i, y^i)$ pour n'importe quelle entrée $x_0^i \equiv x^i$. Le principe est le suivant: la sortie de la j -ième couche est calculée selon

$$x_j = F_j(x_{j-1}, \mathbf{W}_j) \quad (191)$$

Si maintenant, on connaît la variation de la $loss$ par rapport à cette sortie, c'est-à-dire que l'on connaît $\nabla_{x_j} \ell$, alors

$$\nabla_{\mathbf{W}_j} \ell = \nabla_{x_j} \ell \times \nabla_{\mathbf{W}_j} F_j \quad (192)$$

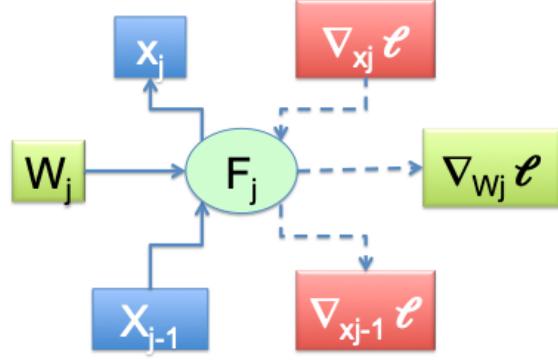


FIGURE 58 – Zoom sur les entrées-sorties de la couche j du réseau 57: on distingue à un flux “montant” ou Forward des entrées/sorties x_j et un flux “descendant” ou Backward des entrées/sorties $\nabla_{x_j} \ell$, c'est-à-dire la rétro-propagation de la variation des erreurs.

Donc, on voit qu'il est intéressant de calculer non seulement les $\nabla_{W_j} \ell$ mais aussi les $\nabla_{x_j} \ell$. Ceci est visualisé dans le schéma de la figure 58 où l'on distingue à l'étape j un flux “montant” ou **Forward** des entrées/sorties x_j et un flux “descendant” ou **Backward** des entrées/sorties $\nabla_{x_j} \ell$, c'est-à-dire **la rétro-propagation de la variation des erreurs**.

8.1.2 L'initialisation: le calcul $\nabla_{x_J} \ell$?

8.1.2.1 Dans le cas d'une régression

L'expression de la *loss* est donnée typiquement par

$$\ell(x_J, y) = \frac{1}{2}(x_J - y)^2 \quad (193)$$

et il vient donc aisément

$$\partial_{x_J} \ell = x_J - y \quad (194)$$

8.1.2.2 Dans le cas d'une classification

Il n'y a pas de différentiabilité *a priori* et x_J est un vecteur à K dimensions ayant pour signification que $x_J(k)$ donne la probabilité de la k -ième classe estimée par un **softmax**; tandis que les y est un **one-hot vector** dont tous les éléments sont nuls sauf pour une

classe pour laquelle $y(k) = 1$. Pour régler le problème de la différentiabilité, on a vu au cours précédent que l'on peut utiliser le maximum de vraisemblance de la cross-entropie, ou minimiser la *loss* suivante:

$$\begin{aligned}\ell(x_J, y) &= - \sum_{k'=1}^K y(k') \log(\text{softmax}(x_J(k'))) \\ &= -\log(\text{softmax}(x_J(k))) \\ &= -x_J(k) + \log \sum_{k'} e^{x_J(k')}\end{aligned}\tag{195}$$

avec

$$\text{softmax}(x_J(k)) = \frac{e^{x_J(k)}}{\sum_{k'} e^{x_J(k')}}\tag{196}$$

Le gradient se calcule alors facilement en séparant les cas où l'on considère la composante k de x_J et y pour laquelle $y(k) = 1$, ou bien les autres composantes pour lesquelles $y(k') = 0$:

$$\nabla_{x_J} \ell = \left| \begin{array}{ll} -1 + \frac{e^{x_J(k)}}{\sum_q e^{x_J(q)}} = \partial_{x_J(k')} \ell & \text{qd } k = k' \\ \frac{e^{x_J(k')}}{\sum_q e^{x_J(q)}} = \partial_{x_J(k')} \ell & \text{qd } k \neq k' \end{array} \right| = \text{softmax}(x_J) - y\tag{197}$$

L'expression est finalement très simple grâce au **softmax**. On se rend compte que ce gradient est l'erreur entre un maximum “doux” (ou adouci) de la composante de x_J qui donne la probabilité d'avoir la classe correspondante, et y qui est la “vraie” probabilité (ie. composante valant 1 pour la bonne classe). Le gros avantage du **softmax** est la différentiabilité.

8.1.3 Rétropropagation des gradients

Cela fait appel à la dérivée de composition de fonctions:

- 1d: $h(x) = g(f(x))$ alors $h'(x) = f'(x)g'(f(x))$ que l'on peut reformuler en terme des variables $z = h(x)$ et $y = f(x)$:

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}\tag{198}$$

— en dimension supérieure: $x = \{x_k\}_k$, $y = \{y_j\}_j$, alors

$$\frac{\partial z}{\partial x_k} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_k} \quad (199)$$

Si on introduit le Jacobien

$$\left(\frac{\partial y}{\partial x} \right) \equiv \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_K} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_M}{\partial x_1} & \cdots & \frac{\partial y_M}{\partial x_K} \end{pmatrix} \quad (200)$$

alors

$$\boxed{\nabla_x z = \left(\frac{\partial y}{\partial x} \right)^T \cdot \nabla_y z} \quad (201)$$

Ainsi, quand on veut calculer $\nabla_{x_{j-1}} \ell$ et $\nabla_{\mathbf{W}_j} \ell$, on fait intervenir les jacobiens de x_j par rapport à x_{j-1} et \mathbf{W}_j , à savoir

$$\nabla_{x_{j-1}} \ell = \left(\frac{\partial x_j}{\partial x_{j-1}} \right)^T \cdot \nabla_{x_j} \ell = \left(\frac{\partial F_j(x_{j-1}, \cdot)}{\partial x_{j-1}} \right)^T \cdot \nabla_{x_j} \ell \quad (202)$$

$$\nabla_{\mathbf{W}_j} \ell = \left(\frac{\partial x_j}{\partial \mathbf{W}_j} \right)^T \cdot \nabla_{x_j} \ell = \left(\frac{\partial F_j(\cdot, \mathbf{W}_j)}{\partial \mathbf{W}_j} \right)^T \cdot \nabla_{x_j} \ell \quad (203)$$

Donc on (rétro)propage le gradient (cf. on passe de la couche $j - 1$ à partir de la couche j) par multiplication des Jacobiens de la transformation dans chaque “boîte”/“couche” (cf. les fonctions F_j) qui est donnée dans le cas d’un réseau de neurones par une transformation associant l’entrée-sortie par combinaison linéaire et une non-linéarité ponctuelle comme un ReLU.

8.1.4 Les Jacobiens des F_j

On peut décomposer (*nb. attention au biais*)

$$F_j = F_j(x_{j-1}, \mathbf{W}_j) = \sigma(\mathbf{W}_j x_{j-1} + b_j) \quad (204)$$

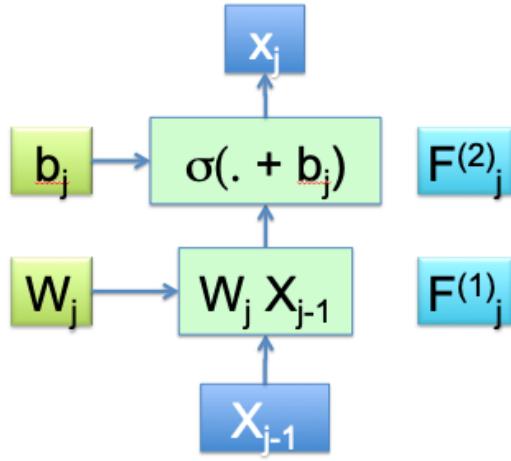


FIGURE 59 – Décomposition en deux phases F^1 et F^2 la transformation dans une couche.

en deux étapes (figure 59):

- une étape matricielle

$$x_{j-1} \xrightarrow{F_j^1} \mathbf{W}_j x_{j-1} \quad (205)$$

- une étape d'ajout de biais et application d'une non linéarité

$$x \xrightarrow{F_j^2} \sigma(x + b_j) \quad (206)$$

Donc, on peut calculer les Jacobiens de F^1 et F^2 assez facilement. D'une part

$$\frac{\partial F_j^1}{\partial \mathbf{W}_j} = x_{j-1} \quad , \quad \frac{\partial F_j^1}{\partial x_{j-1}} = \mathbf{W}_j^T \quad (207)$$

et d'autre part pour F_j^2 , il est bon de remarquer que les variables ne sont pas mélangées donc le Jacobien est diagonal et s'exprime selon

$$\frac{\partial F_j^2}{\partial x_{j-1}^1} = \text{Diag}(\sigma'(x_{j-1}^1 + b_j)) \quad (208)$$

$$\frac{\partial F_j^2}{\partial b_j} = \text{Diag}(\sigma'(x_{j-1}^1 + b_j)) \quad (209)$$

8.1.5 Représentation graphique de l'algorithme

On a vu par les figures 57, 58 et 59 déjà la structure en couches et sous-couches de cet algorithme de rétro-propagation (*backprop*) et l'on peut généraliser ces graphes dans le cas de partages de paramètres par exemple. Les graphes que l'on peut modéliser comme cela ont une contrainte: **ils n'incluent pas de récursion**.

Sinon, les bibliothèques de type Keras ou pyTorch permettent de définir les gradients des différentes couches (si elles ne sont pas de types déjà connus) et l'application des méthodes de descente de gradients par *backprop*.

8.2 L'étude de la convergence du GD

On a développé dans la section précédente les outils pour procéder à une *descente de gradient (GD)*, mais encore faut-il que celle-ci converge vers un minimum de l'erreur sur les données d'entraînement, sachant que l'on voudrait une erreur de généralisation la plus petite possible. Notons que cette dernière fait intervenir la classe de fonctions que l'on peut approximer avec l'architecture du réseau choisi *a priori*, ce qui conditionne le terme de *biais* de l'erreur. Or, dans quelle situation cela converge-t-il? Quel type de GD faut-il pour que cela converge efficacement? Finalement, tout l'art est de trouver l'architecture du réseau en adéquation avec la méthode GD utilisée et vice-versa. Envisageons à présent les deux méthodes de GD: la version *Batch* et la version *Stochastique*.

8.2.1 GD par Batch ou Stochastique

Si on appelle $z_i = (x_i, y_i)$, alors la *loss* globale s'écrit suivant les variables choisies

$$\ell(\theta) = \frac{1}{N} \sum_{i=1}^N \ell(f_\theta(x_i), y_i) = \frac{1}{N} \sum_{i=1}^N \ell(\theta, z_i) \quad (210)$$

L'étape $t + 1$ du GD s'écrit selon la méthode générale

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha \nabla_{\theta} \ell(\theta_t) \\ &= \theta_t - \alpha \left(\frac{1}{N} \sum_i \nabla_{\theta} \ell(\theta_t, z_i) \right) \quad (\text{BGD}) \\ &= \theta_t - \alpha \langle \nabla_{\theta} \ell(\theta_t, z_i) \rangle_{i \leq N} \end{aligned}\tag{211}$$

S'il faut D opérations pour calculer par rétro-propagation $\nabla_{\theta} \ell(\theta, z_i)$, alors pour calculer θ_{t+1} , il faut grossièrement $N \times D$ opérations.

La question qui est venue assez vite est la suivante: est-il nécessaire d'utiliser les N échantillons pour calculer une instance du gradient moyen? On peut répondre déjà en ne prenant qu'un sous lot M des N échantillons, c'est la version **Batch**. Cependant, il y a une version plus drastique encore qui ne retient **qu'un seul échantillon à la fois**, c'est la version **Stochastique**. Dans ce cas, $i_t \in \{1, \dots, N\}$ étant un échantillon pris au **hasard**

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} \ell(\theta_t, z_{i_t}) \quad (\text{SGD})\tag{212}$$

Le **hasard est important** dans cette opération, car alors

$$E[\nabla_{\theta} \ell(\theta_t, z_{i_t})] = \sum_{i_t=1}^N \nabla_{\theta} \ell(\theta_t, z_{i_t}) \times P(i_t) = \frac{1}{N} \sum_{i_t=1}^N \nabla_{\theta} \ell(\theta_t, z_{i_t}) = \nabla_{\theta} \ell(\theta_t)\tag{213}$$

ce qui fait ressortir que $\nabla_{\theta} \ell(\theta_t, z_{i_t})$ est un **gradient bruité** qui en moyenne redonne le gradient global de la *loss*.

La méthode SGD est beaucoup plus rapide que la méthode BGD, mais elle va converger beaucoup plus lentement que cette dernière, surtout vers la fin quand on est proche du minimum. Pour trancher, il faut pouvoir calculer les vitesses de convergence respectives. Dans le cas **convexe** la méthode par **Batch converge exponentiellement** alors que la méthode **Stochastique converge en $1/N$** .

Il est intéressant de noter que la méthode SGD a été décrite par Munro & Robbins en 1951⁴⁵, mais n'avait jamais été mise en pratique à cause de la trop lente convergence, et surtout à cause de la petitesse des bases de données. La donne a changé dès que 1)

45. Robbins, H. and Munro, S. "A Stochastic Approximation Method." Ann. Math. Stat. 22, 400-407, 1951.

on s'est attaqué à des problèmes **non convexes**, et 2) que l'on a pu utiliser des **bases d'entraînement de taille conséquente**.

8.2.2 Exemple de la fonction quadratique: convergence du GD

Définissons la matrice hessienne (dit le hessien) de la *loss* selon

$$H[\ell](\theta) \equiv \left(\frac{\partial^2 \ell}{\partial \theta_i \partial \theta_j} \right) \quad (214)$$

Si τ est un vecteur unitaire dans l'espace des θ , alors

$$\frac{\partial^2 \ell}{\partial \tau^2} = \tau^T H[\ell](\theta) \tau \quad (215)$$

En effet ceci peut se démontrer par un petit rappel de “dérivée directionnelle”. Si $t \in \mathbb{R}$, $x \in \mathbb{R}^n$ et τ unitaire de \mathbb{R}^n , alors définissons la fonction $g(t)$ selon

$$g(t) = f(x + t\tau) = f(x_1 + t\tau_1, \dots, x_n + t\tau_n) \quad (216)$$

Quand on calcule les dérivées successives de $g(t)$, il vient

$$g'(t) = \sum_{i=1}^n \tau_i \partial_{x_i} f(x_1 + t\tau_1, \dots, x_n + t\tau_n) = \tau^T \nabla_x f(x + t\tau) \quad (217)$$

et

$$g''(t) = \sum_{i=1}^n \tau_i \left\{ \sum_{j=1}^n \tau_j \partial_{x_j} \partial_{x_i} f(x_1 + t\tau_1, \dots, x_n + t\tau_n) \right\} = \tau^T \nabla_x^2 f(x + t\tau) \tau \quad (218)$$

Donc, quand on prend la limite quand $t \rightarrow 0$ alors, on obtient la définition des dérivées directionnelles de f selon la direction τ au point x :

$$\frac{\partial f}{\partial \tau} = \nabla_\tau f(x) = \tau^T \nabla_x f(x) \quad (219)$$

$$\frac{\partial^2 f}{\partial \tau^2} = \nabla_\tau^2 f(x) = \tau^T \nabla_x^2 f(x) \tau = \tau^T H[f](x) \tau \quad (220)$$

Maintenant, si on effectue un développement de Taylor de ℓ au voisinage d'un point θ_0 (*nb. pour alléger les notations $H[\ell](\theta)$ et noté $H(\theta)$*), il vient alors:

$$\ell(\theta) = \ell(\theta_0) + (\theta - \theta_0)^T \nabla \ell(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta_0)(\theta - \theta_0) + \dots \quad (221)$$

Si les termes d'ordres supérieurs à 2 sont négligeables, l'analyse est plus simple. En effet, une itération de descente de gradient quand on part de θ_0 donne alors

$$\theta = \theta_0 - \alpha \nabla \ell(\theta_0) \quad (\alpha > 0) \quad (222)$$

et si on note $\nabla \ell(\theta_0) = g$ alors

$$\ell(\theta) = \ell(\theta_0 - \alpha g) = \ell(\theta_0) - \alpha \|g\|^2 + \frac{\alpha^2}{2} g^T H(\theta_0) g \quad (223)$$

Donc, on remarque bien que l'étape de GD fait bien diminuer la fonction au premier ordre (cf. le signe “-”) et **les conditions de l'existence d'un minimum** s'écrivent alors

$$\|g\| = 0 \quad \text{et} \quad g^T H(\theta_0) g \geq 0 \quad (224)$$

C'est la notion de **convexité**, et si l'on veut que cette propriété soit partout définie alors, on impose que ces **2 conditions soient réalisées** $\forall \theta_0$, c'est-à-dire que le **hessien soit une matrice positive** (ie. toutes les valeurs propres positives). On peut même calculer le **saut optimal**, c'est-à-dire déterminer la valeur optimale de α , à savoir

$$\alpha^* = \frac{\|g\|^2}{g^T H(\theta_0) g} \quad (225)$$

mais il faut pouvoir calculer le hessien.

Maintenant, quel est l'ordre de grandeur de ce pas optimal? Si la condition de convexité

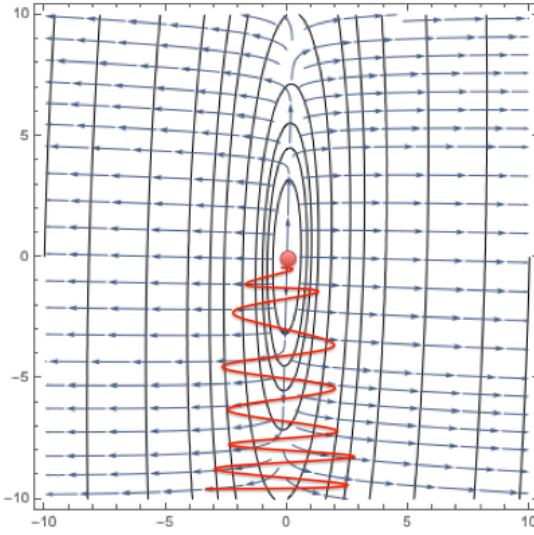


FIGURE 60 – Cas de figure où la fonction, dont on doit trouver le minimum, a une forme en "gouttière étroite". La direction du pas du GD étant opposé au gradient local, il est fort probable que partant d'un θ_0 , les valeurs successives des paramètres oscillent de par et d'autre des bords de la gouttière en convergeant très lentement vers le minimum.

est réalisée alors, on peut borner les valeurs propres du hessien. Si de plus, on impose que la plus petite valeur propre soit **non nulle**, alors on obtient l'encadrement suivant

$$0 < \mu \leq \frac{g^T H(\theta_0)g}{\|g\|^2} \leq L \quad (226)$$

On peut cependant se retrouver dans la situation déplaisante de la figure 60 où l'on passe son temps à faire des aller-retours de part et d'autre d'un couloir étroit. C'est-à-dire qu'il faut réaliser que le gradient indique la direction de “plus grande pente”. Calculons la **vitesse de la convergence**, pour ce faire, on veut pouvoir estimer étape après étape comment se comporte la distance entre θ_t et θ^* (le minimum). Donc, lors d'une étape $t \rightarrow t + 1$

$$\theta_{t+1} = \theta_t - \alpha \nabla \ell(\theta_t) \quad (227)$$

or

$$\nabla \ell(\theta^*) = 0 = \nabla \ell(\theta_t) + H(\theta_t)(\theta^* - \theta^t) \quad (228)$$

donc

$$\theta_{t+1} = \theta_t - \alpha H(\theta_t)(\theta_t - \theta^*) \quad (229)$$

et

$$\theta_{t+1} - \theta^* = \theta_t - \theta^* - \alpha H(\theta_t)(\theta_t - \theta^*) = (1 - \alpha H(\theta_t))(\theta_t - \theta^*) \quad (230)$$

Ainsi, par itérations successives, on obtient alors

$$\theta_{t+1} - \theta^* = (1 - \alpha H(\theta_t)) \dots (1 - \alpha H(\theta_0))(\theta_0 - \theta^*) \quad (231)$$

(*nb notez qu'ici α est une constante du processus alors que nous avons vu un peu plus haut que l'on peut l'adapter à chaque étape t .*) Pour qu'il y ait convergence, il faut que les matrices $(1 - \alpha H(\theta_i))$ soient contractantes, c'est-à-dire que la norme de ces matrices soient plus petites que 1, donc

$$\forall \theta \quad \|1 - \alpha H(\theta)\| < 1 \quad (232)$$

et si on sait que les valeurs propres de $H(\theta)$ sont bornées

$$\mu < \|H(\theta)\| < L \quad (233)$$

alors, on a au moins la contrainte que

$$\boxed{\alpha < \frac{1}{L}} \quad (234)$$

Mais le pire cas est quand le hessien devient tout petit

$$\max_\theta \|1 - \alpha H(\theta)\| = 1 - \alpha \mu \quad (235)$$

alors

$$\|\theta_t - \theta^*\| \leq (1 - \alpha \mu)^t \|\theta_0 - \theta^*\| \quad (236)$$

Or, on ne peut garantir que la condition sur α ci-dessus, ce qui contraint la vitesse de

convergence à:

$$\boxed{||\theta_{t+1} - \theta^*|| \leq \left(1 - \frac{\mu}{L}\right)^t ||\theta_0 - \theta^*||} \quad (237)$$

c'est-à-dire que **l'on voudrait que le rapport μ/L soit le plus proche de 1**, ce qui constitue **le conditionnement du hessien**.

Mais de toute évidence on constate, ne serait que sur la figure 60, que l'on ne va pas dans la “bonne” direction à chaque étape. Rappelons néanmoins que dans le cas quadratique ici, **si le hessien est inversible**

$$\nabla \ell(\theta_t) = H(\theta_t)(\theta_t - \theta^*) \Rightarrow \theta_t - \theta^* = H(\theta_t)^{-1} \nabla \ell(\theta_t) \quad (238)$$

et donc **en principe on peut “directement” aller en 1 étape au minimum** selon

$$\boxed{\theta^* = \theta_0 - H(\theta_0)^{-1} \nabla \ell(\theta_0)} \quad (239)$$

Cependant il faut 1) pouvoir calculer le Hessien, et 2) pouvoir l'inverser. C'est bien là où le bât blesse, car dans les réseaux de neurones primo, on a beaucoup de paramètres, secundo, on n'est pas dans le cas quadratique donc même l'équation précédente n'est pas exacte et il faudrait appliquer l'algorithme d'ordre 2 de Newton. **Or, en pratique ce n'est pas possible du tout de calculer le hessien et on ne peut se contenter que de l'algorithme d'ordre 1.**

8.2.3 Normalisation par mini-batch

C'est un résultat qui date de 4-5 ans qui est très efficace dans le cas des réseaux de neurones où on a typiquement

$$x \longrightarrow F(W.x + b) \quad (240)$$

et on est intéressé par le hessien par rapport au W . Si on note le hessien de F par rapport à une variable z , $F(z) \rightarrow H_z[F](z)$, alors le hessien par rapport au W (*attention, c'est une matrice*) s'écrit

$$F(W.x + b) \rightarrow x \ H_z[F](Wx + b) \ x^T \quad (241)$$

Imaginons que $H_z[F](Wx + b)$ soit à une constante près la matrice identité (*nb. rappelons que l'on veut que le rapport des min/max des valeurs propres soit proche de 1*), alors

le hessien par rapport au W est donné par $xx^T h$ et si maintenant, on somme sur tous les échantillons alors cela va faire intervenir

$$\sum_i F(W.x_i + b) \rightarrow \sum_i x_i x_i^T \quad (242)$$

On reconnaît **l'autocorrélation des données**. Or, cette autocorrélation peut se représenter comme un ellipsoïde qui “englobe” toutes les données. **Si cette ellipsoïde a des petites valeurs propres alors le hessien est mal conditionné.** Donc, on peut essayer de **pré-conditionner** en tentant de faire en sorte que l'autocorrélation soit la plus isotrope possible. Cependant, on n'a pas envie de faire une PCA pour trouver les axes principaux de la matrice d'autocorrélation, surtout qu'il faudrait répéter cette opération sur chaque couche du réseau. Ainsi, on va travailler sur les axes “originaux” des données pour calculer la moyenne et la variance d'ensemble

$$M = \frac{1}{N} \sum_i x_i \quad S^2 = \frac{1}{N} \sum_i (x_i - M)^2 \quad (243)$$

et pour procéder au rescaling

$$x_i \rightarrow x'_i = \frac{x_i - M}{S} \quad (244)$$

Cependant, dans ce cas, on change les variables à l'intérieur du réseau. Pour pallier ce défaut dû au conditionnement du hessien, ce qui est proposé, c'est qu'à la sortie d'un neurone: on procède au rescaling comme ci-dessus et on ajoute 2 variables α, β telles que $\alpha x'_i + \beta$ pour enlever ce rescaling.

Donc après ce rescaling, le hessien est bien conditionné, et la descente de gradient se fait bien. **Ce que l'on comprend pas bien c'est pourquoi ça marche si bien !** Avant d'avoir mis en place cette technique, on utilisait le **Drop-out** qui détruit une fraction aléatoire des connexions entre couches.

9. Gradient Stochastique

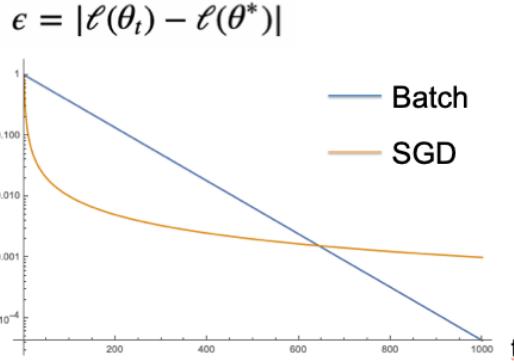


FIGURE 61 – Différence typique entre une descente de gradient par batch, et une version stochastique: si la version stochastique converge très rapidement dans les premières étapes, elle devient beaucoup moins efficace que la version batch au-delà d'un certain nombre d'itérations.

9.1 Introduction

Petit rappel, dans ce cas le gradient “moyen” est calculé sur 1 donnée prise aléatoirement

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} \ell(\theta, z_{i_t}) \quad (245)$$

et en moyenne

$$E_{i_t} [\nabla_{\theta} \ell(\theta, z_{i_t})] = \nabla_{\theta} \ell(\theta) \quad (246)$$

L’évolution en fonction du “temps” de la descente de gradient va être plus chaotique, et on risque de tourner en rond au voisinage du minimum, car l’erreur entre la direction optimale et la direction chaotique va s’accroître au fur et à mesure que l’on se rapproche du minimum.

Si on regarde la vitesse de convergence alors, on a les comportements suivants⁴⁶ (voir figure 61)

$$\epsilon = |\ell(\theta_t) - \ell(\theta^*)| = \begin{cases} O((1 - \alpha\mu)^t) & \text{Batch} \\ O(t^{-1}) & \text{SGD} \end{cases} \quad (247)$$

46. L’évolution en Batch est dite “linéaire” en jargon d’optimisation.

La convergence du SGD peut être plus rapide pour les premières itérations, mais il y a un point à partir duquel le Batch est plus efficace. Cependant, dans les équations ci-dessus, le pas α est constant. Envisageons dans la section suivante de faire varier étape par étape.

9.2 Accélération du GD et SGD à pas variable

Si on veut tenter d'accélérer la vitesse de convergence du SGD, il faut pouvoir réduire le “bruit”, c'est-à-dire les fluctuations du gradient d'une donnée à l'autre. Une façon, c'est d'utiliser des “mini-batch”, autrement dit de prendre une fraction des N échantillons: par exemple, on tire aléatoirement M variables z_i , sur lesquelles on moyenne le gradient. Si $B = 1$ alors, on est dans le cas SGD, et si $B \rightarrow N$ alors la convergence va tendre vers celle de la méthode Batch.

Mais en fait, on aimerait plutôt avoir une méthode qui est de type SGD au départ et qui au fur et à mesure converge vers la méthode Batch, c'est-à-dire, on aimerait des mini-batch de taille de plus en plus grande. Cependant, il y a une manière plus efficace, c'est de jouer sur le pas α du gradient en le diminuant au fur et à mesure. Voyons comment cela se met en œuvre dans le cas du SGD.

Intuitivement, si on passe de $t \rightarrow t + 1$ en 1 step de taille α , ou si on fait la même opération en 2 étapes de taille $\alpha/2$ alors, on compare les méthodes (1) et (2) ci-dessous:

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} \ell(\theta_t, z_{i_t}) \quad (1)$$

$$\begin{aligned} \theta_{t+1} &= \theta_t - \frac{\alpha}{2} \nabla_{\theta} \ell(\theta_t, z_{i_{t1}}) - \frac{\alpha}{2} \nabla_{\theta} \ell(\theta_t, z_{i_{t2}}) \\ &= \theta_t - \alpha \frac{1}{2} (\nabla_{\theta} \ell(\theta_t, z_{i_{t1}}) + \nabla_{\theta} \ell(\theta_t, z_{i_{t2}})) \end{aligned} \quad (2)$$

Par le changement $\alpha \rightarrow 2 \times \alpha/2$ on a réduit par $\sqrt{2}$ la variance sur l'estimation du gradient. Donc dans la démonstration qui va suivre, on va voir que dans le cas du SGD à pas fixe, cela ne converge pas ou plus précisément les valeurs vont converger dans une boule autour du minimum, alors que dans le cas du SGD à pas variable, la méthode converge, car la taille de la boule tend vers 0 si le pas tend lui aussi vers 0.

9.3 Cadre mathématique: fortement convexe et régulier

9.3.1 Méthode Batch

On est capable de démontrer le résultat "intuitif" de la section précédente dans le cas **fortement convexe**, alors qu'en pratique ça marche aussi dans les cas des réseaux de neurones où on n'a aucune garantie de convexité. On va cependant se placer dans un cadre bien précis édicté en 3 propositions.

- Prop 1: **L-Lipschitz**

On impose une propriété de régularité sur $\ell(\theta)$ qui est en lien avec les propriétés du hessien que l'on a vu précédemment. On impose donc que **le gradient soit L-Lipschitz**, c'est-à-dire

$$\boxed{\forall \theta, \theta' \quad \|\nabla_\theta \ell(\theta) - \nabla_\theta \ell(\theta')\| \leq L \|\theta - \theta'\|} \quad (250)$$

Ainsi, si ℓ est 2 fois différentiable alors L-Lipschitz est équivalent à $\|H(\theta)\| < L$. Cependant dans le cas d'un ReLU, sa dérivée est un Heaviside, donc la dérivée seconde est un Dirac. Donc, pour intégrer ce cas de figure, on ne va pas imposer la double différentiabilité mais rester avec celle de L-Lipschitz. Voici une propriété qui découle de la contrainte L-Lipschitz :

$$\boxed{\ell(\theta') \leq \ell(\theta) + (\theta' - \theta)^T \nabla \ell(\theta) + \frac{L}{2} \|\theta' - \theta\|^2} \quad (251)$$

Donc, par cette propriété on constraint en fait la variation de la *loss* quand on se déplace dans la direction du gradient.

- Prop 2: **Convexité**

Imposer la convexité de ℓ revient à dire que

$$\boxed{\forall t \in [0, 1] \quad \ell(t\theta + (1-t)\theta') \leq t\ell(\theta) + (1-t)\ell(\theta')} \quad (252)$$

c'est-à-dire que la valeur de la *loss* en un point sur la droite $\{(\theta, \ell(\theta)), (\theta', \ell(\theta'))\}$ est de valeur plus petite que la moyenne pondérée des valeurs $\ell(\theta)$ et $\ell(\theta')$. Si ℓ est double différentiable alors la convexité est équivalente à ce que le hessien soit

positif, c'est-à-dire que toutes ses valeurs propres sont positives et

$$\forall g, \quad g^T H(\theta)g \geq 0$$

C'est une propriété que l'on avait utilisée dans le cas de la *loss* quadratique, mais ici on est dans le cas de *loss* qui ne sont pas 2 fois différentiables. Donc, on va imposer quelque chose de plus fort.

— Prop 3: **Convexité forte**

On va éviter des cas où le rayon de courbure au minimum est très grand (c'est-à-dire faible courbure); cela va revenir à contraindre que la plus petite des valeurs propres du hessien (cf. μ) n'est pas nulle. Donc, envisageons

$$\boxed{\ell(\theta) - \frac{\mu}{2} \|\theta\|^2 \text{ soit convexe.}} \quad (253)$$

Dans le cas où ℓ est deux fois différentiable revient à imposer que

$$\mu.Id \leq H(\theta)$$

que nous avions aussi utilisé dans le cas quadratique.

La convexité forte implique une propriété de borne Inf:

$$\boxed{\ell(\theta') \geq \ell(\theta) + (\theta' - \theta)^T \nabla \ell(\theta) + \frac{\mu}{2} \|\theta' - \theta\|^2} \quad (254)$$

A l'aide de ces trois propriétés, on énonce le théorème suivant.

Théorème 8. (*Batch GD*):

Si le gradient de ℓ est L -Lipschitz (Prop. 1) et ℓ est μ -fortement convexe (Prop. 3) alors

$$\boxed{\forall \theta_0, \quad \forall \alpha \leq \frac{1}{L} \quad \|\theta_{t+1} - \theta^*\|^2 \leq (1 - \alpha\mu)^{t+1} \|\theta_0 - \theta^*\|^2} \quad (255)$$

c'est-à-dire que l'on retrouve le résultat du cas quadratique mais cette fois sans imposer cette forme de loss spécifique.

Corollaire:

On obtient facilement une contrainte de convergence vers le minimum de la *loss* :

$$\boxed{|\ell(\theta_{t+1}) - \ell(\theta^*)| \leq \frac{L}{2}(1 - \alpha\mu)^{t+1} \|\theta_0 - \theta^*\|^2} \quad (256)$$

En effet, ce corollaire se démontre facilement en utilisant la Prop. 1 et le Théorème 8 ci-dessus. En effet

$$\ell(\theta_{t+1}) \leq \ell(\theta^*) + \frac{L}{2}\|\theta_{t+1} - \theta^*\|^2 \leq \ell(\theta^*) + \frac{L}{2}(1 - \alpha\mu)^{t+1} \|\theta_0 - \theta^*\|^2 \quad (257)$$

Démonstration 8. En utilisant l'étape $t \rightarrow t + 1$ du GD on a

$$\begin{aligned} \|\theta_{t+1} - \theta^*\|^2 &= \|\theta_t - \alpha\nabla\ell(\theta_t) - \theta^*\|^2 \\ &= \|\theta_t - \theta^*\|^2 - 2\alpha\langle\nabla\ell(\theta_t), \theta_t - \theta^*\rangle + \alpha^2\|\nabla\ell(\theta_t)\|^2 \end{aligned} \quad (258)$$

Si on utilise la Prop. 3

$$\ell(\theta^*) \geq \ell(\theta_t) + \langle\nabla\ell(\theta_t), \theta^* - \theta_t\rangle + \frac{\mu}{2}\|\theta^* - \theta_t\|^2 \quad (259)$$

donc

$$\langle\nabla\ell(\theta_t), \theta_t - \theta^*\rangle \geq \ell(\theta_t) - \ell(\theta^*) + \frac{\mu}{2}\|\theta^* - \theta_t\|^2 \quad (260)$$

ou

$$-2\alpha\langle\nabla\ell(\theta_t), \theta_t - \theta^*\rangle \leq -2\alpha(\ell(\theta_t) - \ell(\theta^*)) - \alpha\mu\|\theta^* - \theta_t\|^2 \quad (261)$$

ainsi

$$\|\theta_{t+1} - \theta^*\|^2 \leq \|\theta_t - \theta^*\|^2(1 - \alpha\mu) - 2\alpha(\ell(\theta_t) - \ell(\theta^*)) + \alpha^2\|\nabla\ell(\theta_t)\|^2 \quad (262)$$

Si on est sûr que $\ell(\theta_t) \geq \ell(\theta^*)$ par contre le terme proportionnel à α^2 peut venir contre-balancer le terme linéaire en α , ce qui ne garantirait plus la convergence. C'est bien le problème! D'où l'addition d'une proposition qui découle de la Prop. 1 :

— Prop. 4:

On peut contraindre le gradient de ℓ et θ_t selon

$$\boxed{\|\nabla\ell(\theta_t)\|^2 \leq 2L(\ell(\theta_t) - \ell(\theta^*))} \quad (263)$$

En effet, en utilisant la Prop. 1 (cf. la régularité L-Lipschitz)

$$\ell(\theta') \leq \ell(\theta) + \langle \theta' - \theta, \nabla \ell(\theta) \rangle + \frac{L}{2} \|\theta' - \theta\|^2 \quad (264)$$

avec

$$\theta' = \theta - \frac{1}{L} \nabla \ell(\theta) \quad (265)$$

alors

$$\ell(\theta^*) \leq \ell(\theta') \leq \ell(\theta) - \frac{1}{2L} \|\nabla \ell(\theta)\|^2 \quad (266)$$

ce qui démontre la proposition 4. Par conséquent, on obtient la contrainte suivante

$$-2\alpha(\ell(\theta_t) - \ell(\theta^*)) + \alpha^2 \|\nabla \ell(\theta_t)\|^2 \leq -2\alpha(\ell(\theta_t) - \ell(\theta^*))(1 - \alpha L) \leq 0 \quad (267)$$

si $\alpha \geq 1/L$ (cf. hypothèse du théorème 8), et ainsi

$$\|\theta_{t+1} - \theta^*\|^2 \leq \|\theta_t - \theta^*\|^2(1 - \alpha\mu) \quad (268)$$

ce qui démontre le théorème 8 par itérations successives, et son corollaire sur la convergence sur la loss.

9.3.2 Méthode SGD

Théorème 9. SGD: *Dans le cas du gradient stochastique, on part avec une loss régulière et fortement convexe, et on va démontrer que*

$$E(\|\theta_t - \theta^*\|^2) \leq (1 - \alpha\mu)^{t+1} \|\theta_0 - \theta^*\|^2 + \frac{\alpha}{\mu} B^2 \quad (269)$$

où l'on a fait l'hypothèse

$$\forall t \quad E_{z_i}(\|\nabla \ell(\theta_t, z_i)\|^2) = \frac{1}{n} \sum_{i=1}^n \|\nabla \ell(\theta_t, z_i)\|^2 \leq B^2 \quad (270)$$

Démonstration 9. On va reprendre la démonstration comme dans le cas de Batch

$$\begin{aligned} \|\theta_{t+1} - \theta^*\|^2 &= \|\theta_t - \alpha \nabla \ell(\theta_t, z_{i_t}) - \theta^*\|^2 \\ &= \|\theta_t - \theta^*\|^2 - 2\alpha \langle \nabla \ell(\theta_t, z_{i_t}), \theta_t - \theta^* \rangle + \alpha^2 \|\nabla \ell(\theta_t, z_{i_t})\|^2 \end{aligned} \quad (271)$$

On veut calculer $E_{i_t}(\|\theta_{t+1} - \theta^*\|^2)$ c'est-à-dire l'espérance relativement au choix de z_{i_t} , or on sait que

$$E_{i_t}(\nabla \ell(\theta_t, z_{i_t})) = \frac{1}{n} \sum_{i=1}^n \nabla \ell(\theta_t, z_i) = \nabla \ell(\theta_t) \quad (272)$$

donc

$$E_{i_t}(\|\theta_{t+1} - \theta^*\|^2) = E(\|\theta_t - \theta^*\|^2) - 2\alpha \langle \nabla \ell(\theta_t), \theta_t - \theta^* \rangle + \alpha^2 E_{i_t}(\|\nabla \ell(\theta_t, z_{i_t})\|^2) \quad (273)$$

Le terme proportionnel à α est le même que dans le cas Batch, donc on peut le contraindre pareillement, ainsi on obtient (rappel $\ell(\theta_t) \geq \ell(\theta^*)$)

$$\begin{aligned} E_{i_t}(\|\theta_{t+1} - \theta^*\|^2) &\leq E(\|\theta_t - \theta^*\|^2)(1 - \alpha\mu) - 2\alpha(\ell(\theta_t) - \ell(\theta^*)) + \alpha^2 E_{i_t}(\|\nabla \ell(\theta_t, z_{i_t})\|^2) \\ &\leq E(\|\theta_t - \theta^*\|^2)(1 - \alpha\mu) + \alpha^2 E_{i_t}(\|\nabla \ell(\theta_t, z_{i_t})\|^2) \end{aligned} \quad (274)$$

Or, le terme quadratique en α n'a pas de raison de tendre vers 0, d'où l'introduction de l'hypothèse supplémentaire qui borne la norme du gradient, il vient

$$E_{i_t}(\|\theta_{t+1} - \theta^*\|^2) \leq E(\|\theta_t - \theta^*\|^2)(1 - \alpha\mu) + \alpha^2 B^2 \quad (275)$$

On a une équation de récurrence que l'on peut résoudre exactement, ainsi

$$\begin{aligned} E_{i_t}(\|\theta_{t+1} - \theta^*\|^2) &\leq E(\|\theta_0 - \theta^*\|^2)(1 - \alpha\mu)^{t+1} + \frac{\alpha B^2}{\mu}(1 - (1 - \alpha\mu)^{t+1}) \\ &\leq E(\|\theta_0 - \theta^*\|^2)(1 - \alpha\mu)^{t+1} + \frac{\alpha}{\mu} B^2 \end{aligned} \quad (276)$$

ce qui démontre le théorème 9.

Donc, le théorème nous dit juste que **le terme de gradient ne peut être contrôler et qu'il va sauf accident empêcher la convergence**, car quand $t \rightarrow \infty$ le terme en $(1 - \alpha\mu)^{t+1}$ tend vers 0 certes, mais on va tendre vers une asymptote pour l'erreur en moyenne de

$\|\theta_{t+1} - \theta^*\|^2$. Pour s'en sortir, il faut faire évoluer le α vers 0. Comment le choisir ?

- Prop 5. On va choisir $\alpha = \alpha(t)$ tel que

$$\boxed{\alpha(t) = \frac{\mu}{1 + \mu^2 t} \Rightarrow E_{i_t}(\|\theta_{t+1} - \theta^*\|^2) \leq \frac{C}{1 + t\mu} = O\left(\frac{1}{t}\right)} \quad (277)$$

En effet, si dans l'équation du théorème 9, on identifie α à $\alpha(t+1)$ et que l'on veuille que

$$(1 - \alpha(t+1)\mu)^{t+1} \sim \frac{\alpha(t+1)}{\mu} \quad (278)$$

alors si $\alpha(t)\mu \ll 1$ (mais pas fondamental) on a

$$\alpha(t) = \frac{\mu}{1 + \mu^2 t} \quad (279)$$

Donc

$$E_{i_t}(\|\theta_{t+1} - \theta^*\|^2) \leq E(\|\theta_0 - \theta^*\|^2)(1 - \alpha\mu)^{t+1} + \frac{\alpha}{\mu}B^2 = (E(\|\theta_0 - \theta^*\|^2) + B)\frac{\alpha}{\mu} \quad (280)$$

$$\leq \frac{C}{1 + \mu^2 t} = O(1/t) \quad (281)$$

Ce qui donne la vitesse de convergence du SGD.

9.4 Généralisations ?

Peut-on généraliser les idées des sections précédentes?

- Cadre non différentiable

Si on a des points non différentiables (pour $\ell(\theta)$) alors c'est comme si le gradient n'était pas unique (notion de "sous-gradient"). Mais ces gradients "restent" sous la courbe, laquelle est convexe, donc cela ne change rien à la démonstration précédente tant que l'on contraint que la moyenne des "sous-gradients" soit plus petite que B^2 .

- Nombre d'opérations p . Dans le cas SGD, l'erreur sur les θ est typiquement $\epsilon \sim O(1/p)$ alors que Batch $\epsilon \sim O(1 - \alpha\mu)^{p/n}$.

Si on se réfère à la figure 61, on voit qu'il y a un point "optimal". Si on a n grand et que l'on n'a pas besoin d'une précision extrême alors la méthode stochastique est votre amie; si n n'est pas trop grand par contre vaut mieux utiliser la méthode Batch.

Il y a des méthodes hybrides qui commencent en SGD et qui à partir d'un certain point obtiennent un scaling Batch (voir travail de Francis Bach dont un exposé en 2018 présentait les travaux: méthode SAG et SAGA).

Mais dans le cas des réseaux de neurones, ces méthodes n'adressent pas le fond du problème. Pourquoi? **principalement à cause des minima locaux, et on n'est pas dans un cadre fortement convexe.** Cependant, ce que l'on observe "expérimentalement", c'est que l'on n'est pas bloqué par la vitesse asymptotique du SGD, mais plutôt par les minima locaux. C'est donc un problème beaucoup plus difficile pour contrôler l'optimisation surtout quand on décide de l'architecture du réseau.

9.5 Problèmes d'optimisation

Listons en vrac un certain nombre de points qui restent en suspend (non exhaustif):

- 1) La *loss* $\ell(\theta)$ n'est pas convexe, et donc il faut comprendre la structure des minima locaux, c'est-à-dire par exemple quel est leur profil: sont-ils étroits ou larges? ce qui sous-tend la sensibilité à un step en θ . On peut essayer de connaître leur nombre, savoir s'ils sont profonds... En fait on rencontre des problèmes assez similaires à ceux de **la Physique Statistique** où quand on baisse la température d'un système on veut garantir qu'il tombe dans un état stable d'énergie minimale.
- 2) Les learning rates (α): on sait les contrôler dans le cadre fortement convexe, sinon il faut tâtonner expérimentalement.
- 3) Les mini-batch fonctionnent, mais on ne sait pas pourquoi quand on essaye de comprendre comment cela se cascade dans les réseaux profonds.
- 4) Il ne faut pas croire que l'on obtient une convergence à chaque coup, loin s'en faut et il y a des tas de contre-exemples. Ex. dans l'article arXiv:1812.01662 on montre que le réseau de neurones ne converge pas du tout pour une simple tâche de comparaison de pattern binaires qui *a priori* pourrait être une tâche simple. Mais il y a des croyances (sic) : par exemple "en très grande dimension il y a peu de chance de tomber dans un minima local" (FAUX).

Les stratégies pour aborder ces problèmes sont de 2 types:

- soit on veut (essaye) de “tout contrôler” à savoir le nombre, la structure des minima locaux, etc;
- soit on procède à la manière de l’étude de “système dynamique” qui s’intéresse à la trajectoire en “temps réel” de la minimisation pour que la probabilité de tomber dans un minimum local avec $\ell(\theta)$ éloignée de $\ell(\theta^*)$ soit faible.

Ces quelques réflexions terminent le cours de cette année 2019, il nous reste pas mal de questions en suspend. L’an prochain nous aborderons les réseaux convolutionnels qui semblent être adaptés à capturer des régularités très globales en grande dimension dans tous les domaines où ils ont été introduits.