Couches Dense et Conv2d “ sphériques ” par l’algèbre géométrique conforme

Julien de Saint Angel, Christophe Saint-Jean

Couches Dense et Conv2d « sphériques » par l'algèbre géométrique conforme

Julien de Saint Angel Christophe Saint-Jean

Laboratoire Mathématiques Image et Applications (MIA)
La Rochelle Université

{julien.de_saintangel,christophe.saint-jean}@univ-lr.fr

Résumé

Les couches de type Dense et Conv2d constituent parmi les briques élémentaires les plus populaires des réseaux de neurones actuels. Dans cet article, nous proposons d'en explorer une variante où les hyperplans sont remplacés par des hypersphères. Nous utilisons pour cela le modèle conforme défini par Hestenes et al. [6] où des hypersphères en dimension n peuvent être paramétrées par un vecteur de l'algèbre géométrique conforme $R^{n+1,1}$. En ce qui concerne la couche Dense, une approche similaire existe déjà dans Banarer et al. [1]. L'implémentation de ces deux nouveaux types de couche via des opérateurs d'algèbre linéaire/de convolution classiques est explicitée en même temps que les contraintes pour la mise à jour des poids. L'observation d'expérimentations réalisées sur des données synthétiques et des bases d'images conduit à conclure que les variantes « sphériques » des couches Dense et Conv2d améliorent la fonction de coût et convergent plus rapidement avec un nombre identiques de paramètres. Malheureusement, ce type de couches se révèlent parfois plus sensibles à l'initialisation ce qui peut conduire à une certaine instabilité. Quelques pistes pour remédier à ces problèmes sont présentées.

Mots Clef

Apprentissage profond, Algèbre géométrique conforme, Hypersphère, Convolution.

Abstract

Dense and Conv2d layers are among the most popular building blocks of current neural networks. In this paper, we propose to explore a variant where hyperplanes are replaced by hyperspheres. We use the conformal model defined by Hestenes et al. [6] where hyperspheres in dimension n can be parameterized by a vector of the conformal geometric algebra $R^{n+1,1}$. For the Dense layer, a similar approach already exists in Banarer et al. [1]. The implementation of these two new layer types via classical linear algebra/convolution operators is made explicit along with the constraints for updating the weights. The observation of experiments performed on synthetic data and image databases leads to the conclusion that the "spherical" variants of the Dense and Conv2d layers improve the cost function and converge faster with the same number of parameters. Unfortunately, these types of layers are sometimes more sensitive to initialization, which can lead to instability. Some ways to remedy these problems are presented.

Keywords

Deep learning, Conformal geometric algebra, Hypersphere, Convolution.

1 Introduction

Les couches de type Dense et Conv2d constituent parmi les briques élémentaires les plus populaires des réseaux de neurones actuels. Celles-ci sont simplement basées sur un produit scalaire $x.w$ pour Dense (resp. appliqué localement pour Conv2d) où x est l'entrée et w le vecteur de poids [1] (resp. le filtre de convolution). Pour un vecteur x de taille n, le nombre de paramètres de la couche est $n + 1$. Il est bien connu que l'on ajoute ensuite une fonction d'activation de type "ReLu", chaque neurone d'une couche réalise une partition de $\mathbb{R}^n$ en deux sous-espaces de taille infinie séparés par un hyperplan ; les valeurs positives du produit scalaire sont préservées alors que ses valeurs négatives sont atténuées voire annulées.

En s'inspirant la méthode d'indexation spatiale VP-trees [10], on a l'idée de partitionner $\mathbb{R}^n$ par des hypersphères plutôt que par des hyperplans. D'un point de vue théorique, la VC-dimension dans les deux cas étant de 2 pour $n \geq 2$, on ne doit pas s'attendre une performance significativement meilleure en faveur d'un classifieur basé sur des hypersphères. Toutefois dans la perspective de travaux futurs, on note que la partition induite dans ce cas est faite entre deux sous-espaces dont l'un est de taille finie.

Dans la communauté apprentissage, les fonctions à base radiales [2] permettant l'approximation de fonctions relèvent de la même idée :

$$f(x) \approx \sum_{i=1}^{n} w_i \rho_i(\|x - c_i\|) \tag{1}$$

La fonction f est approximée par une combinaison linéaire de fonctions univariées $\rho_i : \mathbb{R}^+ \mapsto \mathbb{R}$ où $\|.\|$ dénote gé-

1. Pour faciliter la lecture, on considère de x est augmenté pour intégrer le biais dans w

néralement la norme euclidienne. Dans le cas d'un noyau gaussien, $\rho_i(r) = e^{-s_i^2 r^2}$ avec s_i un paramètre d'échelle à estimer s'apparentant au rayon d'une sphère. Les réseaux à fonctions de base radiales étaient considérées dans les années 80-90 comme une alternative à la composition de fonctions (i.e. de couches) [9], le modèle dominant actuellement. Sans détailler plus ici, les machines à vecteur de support utilisant des noyaux de fonctions à base radiales relève de la même idée [3].

Une hypersphère peut être paramétrée classiquement en séparant l'apprentissage des centres et de celui des rayons (comme pour les fonctions à bases radiales). Bien que cette approche soit acceptable, elle cache le lien géométrique existant entre hyperplan et hypersphère : un hyperplan peut être vu comme une hypersphère dont le centre est "à l'infini" et le rayon infini. Pour répondre à ce besoin d'unification, nous avons adopté le formalisme des algèbres géométriques conformes dans lequel hyperplan et hypersphère sont représentés par un vecteur de dimension $n+2$. A l'intérieur de ce cadre mathématique, Banarer et al. [1] ont défini un modèle de neurone hypersphérique qui est la base de notre travail.

Contributions. Nos contributions sont les suivantes : (a) Reprendre le modèle de neurones décrites dans [1] en explicitant les détails de l'entrainement de ce type de neurones (optimiseur, contrainte, initialisation) et ainsi le rendre facilement applicable dans les frameworks modernes d'apprentissage profond. (b) Nous proposons une extension de ce type de neurones aux filtres à convolution 2D hypersphérique et (c) nous comparons ces différents types de couches sur des architectures et des jeux de données simples.

L'article est organisé en trois parties. Nous commencerons par décrire le cadre mathématique des algèbres géométriques conformes pour ensuite étudier les propriétés du modèle de Banarer *et al.* et en proposer une extension à la convolution par des filtres hypersphériques. La deuxième partie sera consacrée à l'implémentation de ces modèles via des opérations de l'algèbre linéaire. Nous y aborderons les questions d'optimisation et d'initialisation. Dans la troisième partie, nous terminerons par des expérimentations destinées à valider les deux modèles et à entrevoir les avantages et les faiblesses.

2 Modèle de neurones hypersphériques avec l'algèbre géométrique conforme

Nous allons commencer par la description du cadre mathématique des algèbres géométriques et du modèle conforme. Cela nous permettra ensuite de définir une sphère comme un vecteur particulier de cette algèbre et d'en lister les propriétés.

2.1 Cadre mathématique

Algèbre géométrique $\mathcal{G}_n$. Une algèbre géométrique $\mathcal{G}_n = \mathcal{G}(\mathcal{V}^n)$ est une algèbre construite à partir d'un n-espace vectoriel sur un corps $\mathbb{K}$ noté $\mathcal{V}^n$ munie d'une forme quadratique $\mathbf{Q}$. Plus précisément, il s'agit de l'algèbre tensorielle $\mathcal{T}(\mathcal{V}^n)$ quotienté par l'idéal engendré par les éléments de la forme $a \otimes a = \mathbf{Q}(a)\mathbf{1}$, a appartenant à $\mathcal{V}^n$. Le produit, défini sur cette algèbre quotient et appelé *produit géométrique*, est multilinéaire, associatif et satisfait la règle suivante [6] :

$$\forall a \in \mathcal{V}^n, \quad aa = a^2 = \mathbf{Q}(a) = \epsilon_a |a|^2$$

où $\epsilon_a \in \{-1, 0, 1\}$ est la signature de a et $|a|$ est un réel positif ou nul appelé magnitude de a.
Par suite, si l'on note $\mathbf{B}$ la forme bilinéaire symétrique associée à $\mathbf{Q}$, le produit géométrique de deux vecteurs a et b de $\mathcal{V}_n$ plongés dans cette algèbre vérifie nécessairement $ab + ba = 2\mathbf{B}(a, b)$.
Par application des propriétés précédentes, on peut donner une expression explicite du produit géométrique ab en le décomposant comme la somme de deux nouveaux produits :

$$ab = \underbrace{\frac{1}{2}(ab + ba)}_{\text{symétrique}} + \underbrace{\frac{1}{2}(ab - ba)}_{\text{anti-symétrique}}$$

- *produit interne* : $\quad a.b := \frac{1}{2}(ab + ba) = b.a = \mathbf{B}(a, b)$
- *produit externe* : $\quad a \wedge b := \frac{1}{2}(ab - ba) = -b \wedge a$

Par suite, $a.b$ un scalaire appartenant à $\mathbb{K}$ et $a \wedge b$ est analogue à un tenseur d'ordre 2. Plus généralement, dans cette algèbre de dimension 2^n, on appellera r-vecteur la combinaison linéaire entre produits externes de r vecteurs. L'ensemble des r-vecteurs forme un sous-espace $\mathcal{G}_n^r$ de $\mathcal{G}_n$ de dimension C_n^r (combinaison des vecteurs de la base de $\mathcal{V}^n$) si bien que

$$\mathcal{G}_n = \bigoplus_{r=0}^{n} \mathcal{G}_n^r$$

Les éléments de $\mathcal{G}_n$ sont appelés des multivecteurs. Pour le lecteur qui souhaite aller plus loin dans les définitions et les applications en traitement du signal et des images, nous recommandons la thèse de Ghina El Mir [4, chap. 1.2], l'article de référence [6] et les actes de la conférence AGACSE depuis 1999.

Algèbre géométrique conforme de $\mathbb{R}^{n+1,1}$. Dans le cadre applicatif nous concernant (vecteurs de $\mathbb{R}^n$), l'espace vectoriel $\mathcal{V}^n$ est choisi comme

$$\mathbb{R}^{n+1,1} = \mathbb{R}^n \oplus \mathbb{R}^{1,1}$$

où $\oplus$ désigne la somme directe et $\mathbb{R}^{1,1}$ le plan de Minkowski. ayant par base $\{e_+, e_-\}$ ($e_+^2 = 1$ et $e_-^2 = -1$). Toutefois, Li *et al.* [8] remplace cette base par

$$\{e_0 := \frac{1}{2}(e_- - e_+), e_\infty := (e_- + e_+)\}$$

de façon à vérifier les propriétés suivantes :

$$e_0^2 = e_\infty^2 = 0, \quad e_\infty . e_0 = -1$$

Pour résumer, si l'on dote $\mathbb{R}^n$ d'une base canonique $\{e_1, \ldots, e_n\}$, l'ensemble des produits internes entre les éléments de la base sont :

$$\begin{cases} e_i . e_j = \delta_{ij}, & \forall i, j \in \{1, ..., n\} \\ e_i . e_0 = e_i . e_\infty = 0 \\ e_\infty . e_0 = e_0 . e_\infty = -1 \end{cases} \tag{2}$$

Parmi les plongements possibles de $\mathbf{x} \in \mathbb{R}^n$ dans un vecteur $x \in \mathbb{R}^{n+1,1}$, les auteurs ont choisi celui vérifiant $x^2 = 0$ et $x.e_\infty = 0$ (cf. [8]) :

$$x = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0 \tag{3}$$

Dans la mesure où $\mathbf{x} \wedge \mathbf{x} = 0$ et suivant les règles données en Eq. 2, $\mathbf{x}^2 = \mathbf{x}.\mathbf{x}$ se réduit au produit scalaire usuel des vecteurs de $\mathbb{R}^n$.

Afin de définir l'équation d'une hypersphère dans ce modèle, on rappelle l'expression du produit interne entre deux points p, q donnés par Eq. 3 :

$$\begin{aligned} p.q = & \ (\mathbf{p} + \frac{1}{2}\|\mathbf{p}\|^2 e_\infty + e_0).(\mathbf{q} + \frac{1}{2}\|\mathbf{q}\|^2 e_\infty + e_0) \\ = & \mathbf{p}.\mathbf{q} + \frac{1}{2}\|\mathbf{q}\|^2 \underbrace{\mathbf{p}.e_\infty}_{0} + \underbrace{\mathbf{p}.e_0}_{0} + \\ & \frac{1}{2}\|\mathbf{p}\|^2 \left(\underbrace{\mathbf{q}.e_\infty}_{0} + \frac{1}{2}\|\mathbf{q}\|^2 \underbrace{e_\infty.e_\infty}_{0} + \underbrace{e_\infty.e_0}_{-1} \right) + \\ & \underbrace{e_0.\mathbf{q}}_{0} + \frac{1}{2}\|\mathbf{q}\|^2 \underbrace{e_0.e_\infty}_{-1} + \underbrace{e_0.e_0}_{0} \\ = & -\frac{1}{2}\|\mathbf{p} - \mathbf{q}\|^2 \end{aligned} \tag{4}$$

Nous disposons maintenant de tous les éléments pour rappeler le modèle de neurone hypersphérique défini par [1] .

.

2.2 Modèle de neurone hypersphérique

Dans $\mathbb{R}^n$, une hypershère de centre $\mathbf{c}$ et de rayon ρ est définie par l'ensemble des $\mathbf{x}$ vérifiant $\|\mathbf{x} - \mathbf{c}\|^2 = \rho^2$. Cette équation peut être reformulée par l'Eq. 4 comme

$$\begin{aligned} & x.c + \frac{1}{2}\rho^2 = x.c - \frac{1}{2}\rho^2 (x.e_\infty) = 0 \Leftrightarrow \\ & x.\underbrace{(c - \frac{1}{2}\rho^2 e_\infty)}_{s} = x.s = 0 \end{aligned}$$

En développant l'expression de s, on obtient la définition comme un vecteur de $\mathbb{R}^{n+1,1}$:

$$s = \mathbf{c} + \frac{1}{2}(\|\mathbf{c}\|^2 - \rho^2)e_\infty + e_0 \tag{5}$$

Ainsi, on peut voir que le vecteur s représente une hypersphère qu'à un coefficient multiplication car $x.\lambda s = \lambda x.s = 0$ pour tout réel λ non nul. Afin de rendre ce vecteur *unique*, on dispose de la contrainte de normalisation $s.e_\infty = -1$ qui garantit qu'un point x est

- à l'intérieur de l'hypersphère) si $0 < x.s < \frac{1}{2}\rho^2$.
- sur l'hypersphère si $x.s = 0$.
- à l'extérieur de l'hypersphère s'il vérifie $x.s < 0$.

La figure 1 illustre la valeur du produit interne $x.s$ pour le cas $n = 2$ dans le cas où $\mathbf{x} = (x_0, 0)$ avec $x_0 \in [-10, 10]$ pour une sphère de centre $(0, 0)$ et de rayon $\rho = 5$. Comme attendu, le produit interne est nul pour $x_0 \pm 5$ et atteint son maximum 12.5 pour $x_0 = 0$.

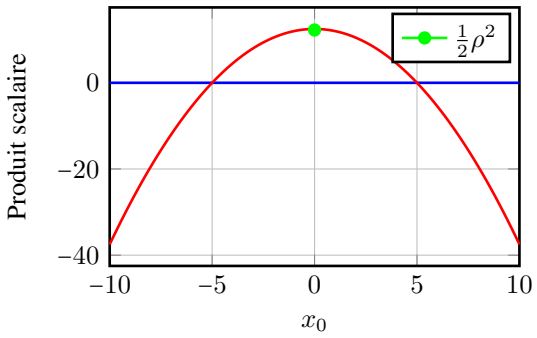


FIGURE 1 – Profil du produit scalaire $x.s$

Pour résumer ce paramétrage d'une hypersphère en dimension n, les points principaux sont les suivants :

1. Un vecteur s de $\mathbb{R}^{n+1,1}$ qui encode les paramètres du centre et du rayon.
2. La contrainte $s.e_\infty = -1$ qui implique que s est dans un sous-espace de $\mathbb{R}^{n+1,1}$ de dimension $n + 1$.
3. Le signe de produit interne $x.s$ qui permet de qualifier la position de $\mathbf{x}$ par rapport à l'hypersphère.

Comme le mentionne Li et al. [8], une hypersphère non normée peut être représentée de façon générale à partir des paramètres de rayon ρ et centre p qui s'écrivent de la façon suivante :

$$\rho^2 = \frac{s^2}{(s.e_\infty)^2} \qquad p = \frac{s}{-s.e_\infty} + \frac{1}{2}\rho^2 e_\infty$$

On constate donc que dans le cas limite où $s.e_\infty$ tend vers 0, le rayon de l'hypersphère est infini. De même lorsque le rayon est nul, l'hypersphère correspond à un hyperpoint c. Ce cas particulier assure théoriquement que les problèmes dont les données sont linéairement séparables, sont également séparables par des hypersphères.

Rappelons que la contrainte de normalisation $e_\infty.s = -1$, assure la représentation d'une hypersphère par un vecteur unique. Si l'on change le signe de $e_\infty.s$ en imposant $e_\infty.s = 1$, cela reviendrait à échanger les rôles d'intérieur et d'extérieur de la sphère. Le centre de la sphère se retrouverai ainsi à l'extérieur de celle-ci.

2.3 Modèle de filtre hypersphérique pour la convolution

L'extension du modèle de neurone hypersphérique vers celui d'un filtre de convolution hypersphérique ne requiert pas de nouveaux outils mathématiques. Pour la description de ce modèle, nous nous consacrerons au cas d'un filtre de convolution 2-d discret (pour les images) mais l'approche

reste valable en toute dimension, quelque soit le mode de *padding* et les valeurs de *strides* choisis.
Il est bien connu que la convolution standard peut se voir comme le produit scalaire entre les versions vectorisées du bloc d'image et du filtre F plus un biais :

$$\begin{aligned}(I * F)[i,j] &= \sum_{d_i}\sum_{d_j} I[i+d_i, j+d_j] * F[d_i, d_j] + b \\ &= vec(I[i-d_i : i+d_i, j-d_j : j+d_j]). \\ &\quad vec(F[-d_i : d_i, -d_j : d_j]) + b\end{aligned}$$

où d_i et d_j désignent les indices de parcours du filtre qui dépendent de la taille du filtre $d \times d$. De façon naturelle, on peut voir $vec(I[i-d_i : i+d_i, j-d_j : j+d_j])$ comme un vecteur $x_I(i,j)$ de $\mathbb{R}^{d^2}$ que l'on va plonger dans l'algèbre géométrique conforme $\mathbb{R}^{d^2+1,1}$. Identiquement, le filtre de convolution est remplacé par une sphère en dimension $\mathbb{R}^{d^2}$ paramétrée dans cette même algèbre. Si l'on note $\circledast$ cette "nouvelle" convolution, on en définit le résultat comme :

$$(I \circledast s)[i,j] = x_I(i,j).s \tag{6}$$

La valeur de sortie de ce filtre possède donc les mêmes propriétés que le produit interne $x.s$ défini dans la sous-section précédente (voir Fig. 1). L'extension de ce filtre aux images à k-canaux ne pose pas de difficulté :

— $(I \circledast s)[i,j] = \sum_k x_I(i,j,k).s$ (~ Conv2d)
— $(I \circledast s)[i,j] = \sum_k x_I(i,j,k).s_k$ où une hypersphère est définie par canal (~ DepthWiseConv2d)

Bien que non traité dans cet article, on peut imaginer qu'une "Conv2dSeparable" revient à paramétrer une hypersphère en dimension d dans $\mathbb{R}^{d^2}$.
Les détails d'implémentation sont donnés dans la section suivante.

3 Implémentation

On peut remarquer que l'ensemble des formules de la section précédente s'appliquent quelque soit la dimension n et s'écrivent sans la nécessité de définir une base pour $\mathbb{R}^n$. On appelle cela le *calcul sans coordonnées*. Toutefois, lors d'une mise en œuvre pratique, les $n+2$ valeurs permettant d'encoder les données et hypersphères doivent être explicitées pour se ramener à des opérations matricielles prenant en compte la forme quadratique $\mathbf{Q}$.

3.1 Formulation matricielle de $x.s$

Pour garantir l'efficacité du calcul du produit interne, celui doit être exprimé comme un produit matriciel $\check{x}^t M \check{s}$.
Dans la base $\{e_1, e_2, ..., e_n, e_0, e_\infty\}$, $\mathbf{x}$ et s de centre $\mathbf{c}$ et rayon ρ s'expriment comme des vecteurs à $n+2$ coordonnées :

$$\mathbf{x} \mapsto \check{x} := [x_1, x_2, \ldots, x_n, 1, \frac{1}{2}\|\mathbf{x}\|^2]^t \tag{7}$$

$$(\mathbf{c}, \rho) \mapsto \check{s} := [c_1, c_2, \ldots, c_n, 1, \frac{1}{2}(\|\mathbf{c}\|^2 - \rho^2)]^t \tag{8}$$

Pratiquement, l'équation 7 revient concaténer deux valeurs selon le dernier axe.

La matrice M qui exprime les relations données dans Eq. 2 s'écrit comme

$$M = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 & 0 \\ 0 & 0 & \ldots & 1 & 0 & 0 \\ 0 & 0 & \ldots & 0 & 0 & -1 \\ 0 & 0 & \ldots & 0 & -1 & 0 \end{pmatrix} \tag{9}$$

Le lecteur pourra aisément vérifier que :

$$\check{x}^t M \check{s} = \frac{1}{2}\left(\rho^2 - \|\mathbf{x}-\mathbf{c}\|^2\right) \text{ et } \check{p}^t M \check{q} = -\frac{1}{2}\left(\|\mathbf{p}-\mathbf{q}\|^2\right)$$

3.2 Calcul du filtre hypersphérique

Pour le filtre hypersphérique, nous conseillons de séparer en trois parties le calcul de $\check{x}^t M \check{s}$:

1. La convolution standard de l'image avec le filtre correspondant aux n premières coordonnées de $\check{s}$ redimensionnées à la taille $d \times d$.
2. L'équivalent de $-\frac{1}{2}\|x\|^2$ revient à convoluer l'image par un filtre constant de valeur $-\frac{1}{2}$ de taille $d \times d$. Le résultat est multiplié par l'avant dernière coordonnée de $\check{s}$ qui vaut 1 (cf : sous-section 3.3)
3. On effectue une multiplication termes à termes pour obtenir un tenseur contenant les coefficients équivalents au produit entre la partie e_0 de $\check{x}$ et e_∞ de $\check{s}$.

Le résultat de la convolution hypersphérique est la somme de ces trois termes. Par ailleurs, si l'image traitée contient plusieurs canaux, une nouvelle somme selon la profondeur permet reproduire le comportement de Conv2D.

3.3 Contraintes et initialisations

Comme nous l'avons dit à plusieurs reprises, une hypersphère est représentée de manière unique par un vecteur $\check{s}$ dont l'avant-dernière coordonnée est 1 (correspondant à la contrainte $s.e_\infty = -1$). L'optimisation *conjointe* de $\mathbf{p}$ et ρ par une méthode de descente de gradient doit garantir cette contrainte. Dans ce but, un re-normalisation de la sphère est effectuée après chaque mise à jour de $\check{s}$.
L'initialisation des paramètres d'une couche à plusieurs neurones hypersphérique est un problème complexe. Dans nos expérimentations, nous avons grossièrement choisi l'heuristique de Glorot et Bengio en effectuant un tirage aléatoire des centres suivant une loi normale centrée dont l'écart-type dépend diminue avec n et le nombre de neurones de la couche[5]. Cela correspond au choix par défaut des couches Dense [2]. Étant donné que nous avons centré-réduits les données en amont, les rayons des hypersphères ont été initialisés à 1. D'autres choix sont possibles et seront discutés dans les perspectives.

2. Il s'agit dans ce cas des coordonnées du vecteur normal à un hyperplan

4 Expérimentations

Les expérimentations ont pour objectif de valider les modèles de couches hypersphériques construits à partir d'un ensemble de neurones ou de filtre de convolution hypersphériques. Comme nous nous positionnons sur des briques élémentaires de bas-niveau, nous avons volontairement choisi des architectures de réseau simples afin de limiter l'écueil du sur-apprentissage et des difficultés d'interprétation. Nos expérimentations sont découpées en 2 parties suivant le type de neurone à étudier.

4.1 Couches hypersphériques vs Dense

Jeux de données. Deux jeux de données synthéthiques [3] ont été utilisés pour étudier la performance des réseaux à couches hypersphériques.
Le premier nommé *Easy* est construit de façon à avoir 3 classes quasiment linéairement séparables. Celui-ci contient 350 points en dimension 2 ce qui donne l'opportunité de illustrer le modèle par différentes images.
Le second nommé *Dif* contient également une série de 450 points en dimension 2 répartis en 3 classes non linéairement séparables avec un taux de chevauchement non négligeable. Ce dernier point nous permettra de voir et comparer la forme des frontières de décisions pour différentes architectures. Les deux jeux de données ont été séparés en ensemble d'apprentissage (66.66%) et de validation (33.33%).

Types de réseaux et entrainement. L'architecture de base que nous considérons est un perceptron multi-couches de faible profondeur. Dans un premier cas, les couches du réseau (noté PMC) seront des couches Dense classiques alors que le réseau sera nommé GeoPMC lorsque les neurones seront uniquement hypersphériques. Différents paramètres ont été évalués :

- Le nombre de couches cachées varie de 0 à 2.
- Le nombre de neurones par couches cachées prend les valeurs 2, 3 ou 10.
- La batch normalisation et la fonction d'activation ReLu qui s'applique ou non sur toutes les couches cachées.

Chaque réseau se termine par une couche de 3 neurones classiques ou hypersphériques avant de passer par la fonction d'activation softmax car il s'agit d'un problème de classification ; celle-ci n'est jamais précédée d'un ReLu.
Pour faciliter la lecture des résultats, chaque architecture du réseau est codée par une chaîne de caractères : un chiffre correspond au nombre de neurones par couche, "r" à l'utilisation de la fonction d'activation relu, "b" à une batch normalisation et "sf" à la fonction d'activation softmax.
La taille du batch est de 30, l'algorithme d'optimisation est Adam [7].

Résultats. Bien que nous ayons testé l'ensemble des configurations ci-dessus, nous avons sélectionné un ensemble de configurations efficaces. Les figures 2 et 3 reportent le taux de bonne prédiction (*accuracy*) sur les jeux de données de validation au cours des itérations. On constate que l'*accuracy* augmente plus rapidement pour les modèles GeoPMC à architecture identique. L'échec du modèle GeoPMC "b 2 r b 2 r b 3 sf" s'explique probablement par une mauvaise architecture qui contribue à l'inhibition de certains neurones (*dying-relu*) .

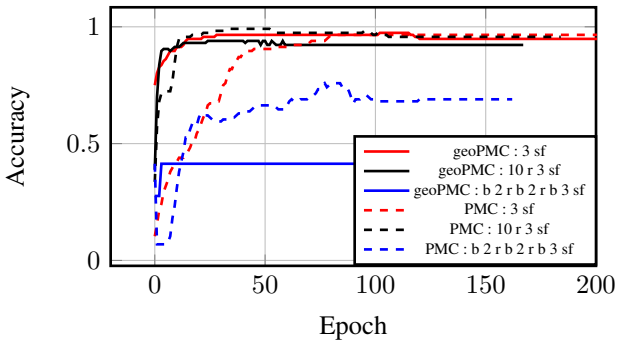


FIGURE 2 – easy : *accuracy* pour les données de validation

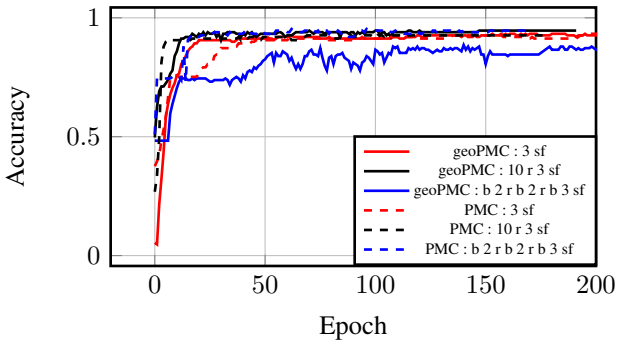


FIGURE 3 – *Dif* : *accuracy* pour les données de validation

Pour aller plus loin, la table 1 reporte l'*accuracy* en fin d'entrainement des 14 réseaux les plus performants. Excepté les trois premiers cas, l'ensemble des réseaux efficaces sur *Easy*, le sont également sur *Dif*.

	Easy		*Dif*	
Architectures	GeoPMC	PMC	GeoPMC	PMC
10 r : 3 sf	92%	99%	94.6%	92.6%
b 2 r : b 2 r : b 3 sf	41%	68%	86%	94%
3 r : 3 sf	67.2%	95%	93.3%	92.6%
3 sf	96.5%	97.4%	93.3%	92%
b 2 : b 2 : b 3 sf	98.3%	98.3%	91.3%	92%
10 : 3 sf	99%	96%	92%	93.3%
10 : 10 : 3 sf	99%	94%	91.3%	93.3%
2 : 2 : 3 sf	98.3%	95.7%	91.3%	93.3%
3 : 3 : 3 sf	98.3%	95.7%	90%	93.3%
b 10 : b 3 sf	98.3%	97.4%	93%	91.3%
b 10 r : b 3 sf	98.3%	98.3%	92%	92%
b 3 r : b 3 sf	98.3%	98.3%	92%	91.6%
b 3 sf	97.4%	98.3%	92.4%	92%
b 2 : b 3 sf	98.3%	98.3%	91.8%	91.6%

TABLE 1 – Accuracy pour différentes architectures

3. Lien de téléchargement `https://tinyurl.com/2xc8x4ct`

Hormis l'architecture "10 r : 3 sf" qui s'avère être plus performante pour les réseaux à couches hypersphériques et le réseau "b 2 r : b 2 r : b 3 sf" qui lui semble être meilleur que le modèle classique sur les données de *Easy*, l'ensemble des résultats d'*accuracy* ne sont pas significativement différents. Etant donnée la taille des jeux de données de validation, un écart de 1% représente donc entre 1 et 2 points mal classées.
Afin de mieux voir l'impact de la fonction d'activation ReLu et la batch normalisation, les tables 2 et 3 présentent les résultats de l'apprentissage par architecture.

GeoPMC				
Architectures	∅	ReLu	bn	bn + ReLu
10 : 3 sf	99%	92%	98.3%	98.3%
3 sf	95%	ND	98.3%	ND
3 : 3 sf	94%	67.2%	98.3%	98.3%
2 : 3 sf	92.2%	97.4%	98.3%	98.3%
3 : 3 : 3 sf	98.3%	99.1%	97.4%	68.9%
PMC				
Architectures	∅	ReLu	bn	bn + ReLu
10 : 3 sf	96%	95.6%	97%	98.3%
3 sf	97.4%	ND	98.3%	ND
3 : 3 sf	95.6%	94.8%	98.3%	98.3%
2 : 3 sf	96.5%	94.8%	98.3%	97.4%
3 : 3 : 3 sf	95.7%	94%	98.3%	98.3%

TABLE 2 – *Easy* : Comparaison de l'impact de ReLu et de la batch normalisation

GeoPMC				
Architectures	∅	ReLu	bn	bn + ReLu
10 : 3 sf	92%	94.6%	93%	92%
3 sf	93.3%	ND	92.4%	ND
3 : 3 sf	92.6%	93.3%	91.9%	92.6%
2 : 3 sf	92.6%	89.6%	92%	92%
3 : 3 : 3 sf	90%	86%	92.6%	90%
PMC				
Architectures	∅	ReLu	bn	bn + ReLu
10 : 3 sf	93.3%	92.6%	91%	92%
3 sf	92%	ND	92%	ND
3 : 3 sf	92.6%	92.6%	92%	92%
2 : 3 sf	92.6%	92.6%	92%	93%
3 : 3 : 3 sf	93.3%	48%	92.6%	92.2%

TABLE 3 – *Dif* : Comparaison de l'impact de ReLu et de la batch normalisation

De manière générale, la présence du ReLu semble dégrader davantage la qualité des prédictions sur les réseaux à couches hypersphèriques d'autant que le nombre de couches cachées croît. En conséquence, le ReLu ne semble pas être la fonction d'activation la plus adaptée sachant que $x.s$ peut être un scalaire très négatif (cf. Fig. 1). Une étude supplémentaire s'avère nécessaire pour en trouver une plus appropriée. En ce qui concerne la batch normalisation, elle permet dans le cas des données de *Easy*, d'améliorer considérablement les résultats. Ce qui est moins pertinent pour les données issue du fichier *Dif*.

Si l'on regarde maintenant les valeurs de la fonction de perte (*loss*) sur le jeu de données de validation (Fig. 4 et 5), on peut observer que pour les modèles GeoPMC convergent plus vite vers des valeurs plus petites.

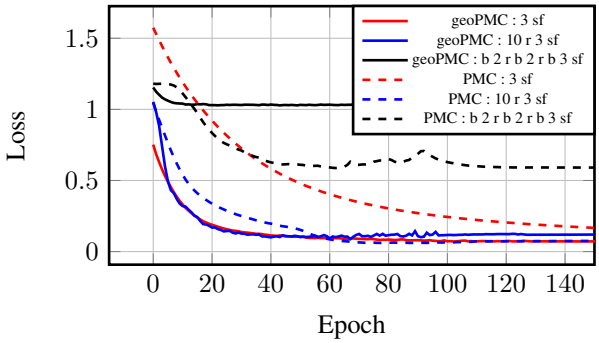


FIGURE 4 – *Easy* : Courbes des fonctions de pertes

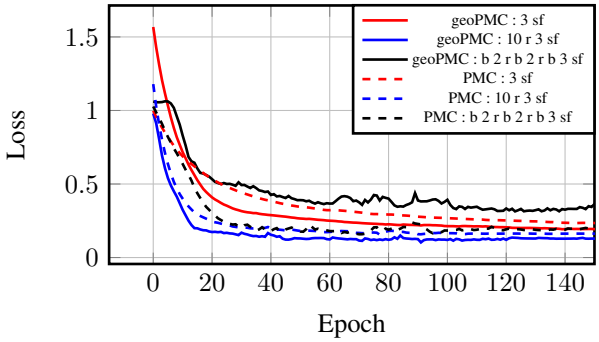
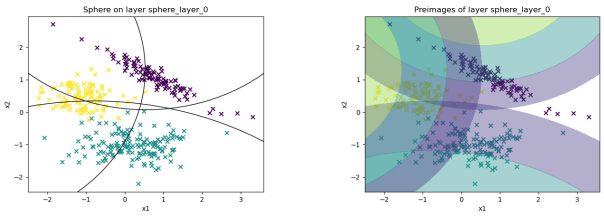


FIGURE 5 – *Dif* : Courbes des fonctions de pertes

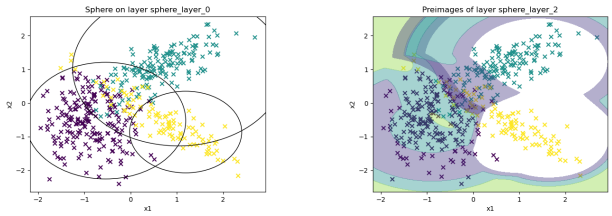
Comme les expérimentations sont effectuées sur des données 2d, il est possible de visualiser la pré-image d'un neurone hypersphèrique paramétré par s en affichant les bandes de niveau pour les valeurs positives de $\Phi(x).s$ (Fig. 6b et 7b). Pour chaque point $\mathbf{x} = (x_1, x_2)$ de grille d'affichage, on infère l'entrée $\Phi(x)$ de la couche à afficher pour $\mathbf{x}$ et on calcule la valeur de $\Phi(x).s$.



(a) $x.s = 0$ (b) $x.s > 0$

FIGURE 6 – *Easy* : Bandes de niveaux pour geomPMC "3 sf"

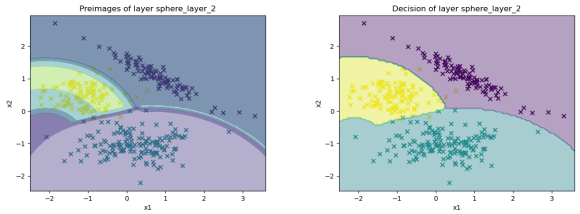
Si l'hypersphère à afficher est située dans la couche 0 (ie. pas de non-linearité) (cf. Fig. 7a) ou qu'il n'existe pas de couches cachées (voir Fig. 6a) , les bandes de niveau sont circulaires concentriques.



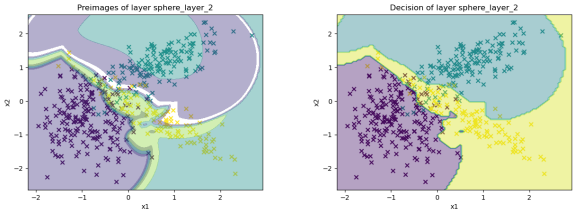
(a) Sphères de la couche 0 (b) Pré-images de la couche 2

FIGURE 7 – *Dif* : Bandes de niveaux pour geomPMC "b 3 r : b 3 r : b 3 sf"

La figure 8 permet d'illustrer la proximité entre les pré-images des hypersphères (situées avant la couche softmax) et les frontières de décision. Cela peut paraître évident au premier abord mais si l'on se réfère à la figure 6b, on sait que la frontière de décision entre deux hypersphères en l'absence de non-linéarité est un hyperplan... On peut noter également, que les centres des hypersphères (colonne de gauche en jaune) ne correspondent pas aux centres des classes (colonne de droite).



(a) *Easy* : geoPMC "3 r : 3 r : 3 sf"



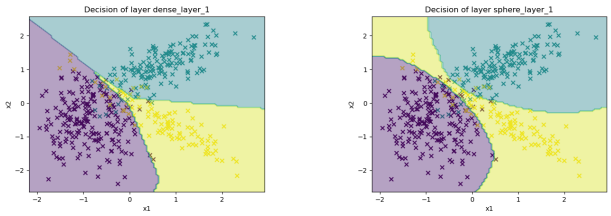
(b) *Dif* : geoPMC "10 r : 3 r : 3 sf"

FIGURE 8 – PréImages couche 2 (gauche) et frontières de décision (à droite)

La différence entre deux modèles GeoPMC et PMC de même architecture est flagrante si l'on compare leurs frontières de décision (cf. Fig. 9). Si l'on regarde leur forme, PMC correspond une succession de lignes brisées alors GeoPMC produit une succession d'arcs de cercle brisés.

4.2 Filtres hypersphériques vs Conv2d

Dans cette partie, nos proposons une comparaison entre les filtres hypersphériques et Conv2d sur le jeu de données MNIST. Il s'agit d'une base de 70000 mini-images labelisées, en niveaux de gris, de taille 28×28 représentant l'écriture manuscrite des chiffres de 0 à 9. C'est donc une problème de reconnaissance à 10 classes.
Dans cette expérimentation, nous testons une architecture



(a) PMC "3 r : 3 r : 3 sf" (b) GeoPMC "3 r : 3 r : 3 sf"

FIGURE 9 – *Dif* : Comparaison de frontières de décision

où la séquence de couches est la suivante :

1. une batch normalisation ou non.
2. une couche hypersphérique pour GeoPMC / une couche Dense pour PMC / une couche filtres hypersphériques pour GeoConv2d / une couche Conv2d pour Conv2d avec un certain nombre de filtres de taille 3×3.
3. une fonction ReLU ou non.
4. une batch normalisation ou non.
5. une couche Dense ou hypersphérique avec un certain nombre de neurones.
6. une activation softmax.

La table 4 donne les résultats d'*accuracy* pour l'ensemble des configurations testées.

	Dense		Convolution	
Architectures	GeoPMC	PMC	GeoConv2d	Conv2d
10 sf	92.7%	92.7%	ND	
32 : 10 sf	93%	92.6%	97.5%	97.5%
32 r : 10 sf	10%	96.8%	98%	98.2%
b 32 : b 10 sf	86.4%	91.6%	97.2%	97.2%
b 32 r : b 10 sf	10%	96%	97.8%	98.2%
512 : 10 sf	93%	93%	97%	97%
512 r : 10 sf	10%	98%	98%	98%
b 512 : b 10 sf	93%	93%	97.7%	97.1%
b 512 r : b 10 sf	10%	98%	97.7%	98.3%

TABLE 4 – MNIST : Comparaison de l'*accuracy* pour différents modèles

La première observation que l'on peut établir concerne l'incapacité de faire fonctionne GeoPMC en "grande" dimension avec un ReLU. Le taux d'erreur de 90% laisse à penser que le classifieur prédit toujours la même classe. Le problème provient selon nous d'une mauvaise initialisation des hypersphères qui fait tous les points sont à l'extérieur de toutes les sphères. Par la suite, ReLu annulera tout signal vers les couches suivantes du réseau. Ce problème ne se produit certainement pas pour GeoConv2d car les hypersphères sont définies en dimension 9 au lieu de 784. Plus généralement, on peut se demander si les couches hypersphériques sont sensibles à la malédiction de la dimension. On constate la même amélioration lors du passage des modèles GeoPMC vers GeoConv2d que celui de PMC vers Conv2d. D'après nos premières expériences, GeoConv2d et Conv2d obtiennent des résultats identiques.

5 Conclusions et perspectives

Dans cet article, nous avons repris le cadre mathématique des algèbres géométriques conformes pour étudier le modèle de neurone hypersphérique de Banarer *et al.* [1]. Nous avons étendu ce modèle à la définition d'un filtre de convolution hypersphérique pour les tenseurs d'ordre l à plusieurs canaux. Cela revient à transformer les coefficients de ce filtre et du biais en coordonnées d'une hypersphère dont on peut optimiser les paramètres.

Les expérimentations réalisées sur des cas simples ont eu pour objectif de valider ces modèles et les comparer sur des architectures simples à des couches Dense ou Conv2d. Nous avons observé que les résultats des taux de prédiction restent globalement similaires entre les réseaux classiques et les réseaux à couches hypersphériques pour une architecture de type Perceptron Multicouches. Nous avons pu constater dans nos expériences que les fonctions de pertes convergent plus rapidement et atteignent des valeurs plus petites. De même en adaptant des réseaux convolutifs avec des filtres hypersphériques, on observe que les résultats sur les prédictions restent analogues. Néanmoins,lors du passage d'un réseau à couche dense hypersphérique à un réseau convolutif à filtre hypersphérique, l'augmentation du taux de bonne prédiction est similaire au cas classique.

L'initialisation des paramètres des hypersphères est certainement un point à améliorer. On pourrait à l'image de ce qui se fait dans les réseaux à bases de fonctions radiales choisir les centres à partir d'un algorithme de clustering et les rayons à partir la dispersion autour des centres de ces clusters. Rappelons toutefois que, comme l'a montré les expérimentations, il ne faut pas confondre les centres des hypersphères avec les centres de clusters.

Une piste plus prometteuse serait d'initialiser les paramètres d'une hypersphère à partir des paramètres d'un hyperplan. Cela revient à choisir s de façon à ce que $x.s$ approxime $w.\mathbf{x} + b$. On imagine que dans ce cas s correspond à une sphère de rayon "grand" et dont le centre est positionné tel que $w.\mathbf{c} + b > 0$. La formalisation de cette intuition nécessite d'étudier plus en détails la représentation d'un plan par un vecteur de $\mathbb{R}^{n+1,1}$. L'avantage de cette approche est double. La première est de pouvoir mieux bénéficier des nombreux travaux qui concernent une meilleure initialisation des couches Dense et Conv2d. La seconde encore plus prometteuse est de transférer les poids appris des couches Dense et Conv2d vers des couches hypersphériques et poursuivre l'entrainement. De cette façon, nous pouvons substituer toutes couches denses et convolutives dans des modèles plus complexes (ex : ResNet152) par des couches hypersphériques. Par ailleurs, nous souhaitons étudier si les modèles hypersphériques décrits dans cet article peuvent avoir de l'intérêt dans une détection "controlée" d'anomalies.

Références

[1] Vladimir Banarer, Christian Perwass, and Gerald Sommer. The hypersphere neuron. In *In 11th European Symposium on Artificial Neural Networks, ESANN 2003, Bruges*, pages 469–474, 2003.

[2] Martin D Buhmann. *Radial basis functions : theory and implementations*, volume 12. Cambridge university press, 2003.

[3] Kai-Min Chung, Wei-Chun Kao, Chia-Liang Sun, Li-Lun Wang, and Chih-Jen Lin. Radius margin bounds for support vector machines with the rbf kernel. *Neural computation*, 15(11) :2643–2681, 2003.

[4] Ghina El Mir. *Algèbres de Clifford conformes et orbites de points de vue d'images*. Theses, Université de La Rochelle, July 2014.

[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[6] David Hestenes, Hongbo Li, and Alyn Rockwood. *New Algebraic Tools for Classical Geometry*, pages 3–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[7] Diederik P. Kingma and Jimmy Ba. Adam : A method for stochastic optimization, 2017.

[8] Hongbo Li, David Hestenes, and Alyn Rockwood. *Spherical Conformal Geometry with Geometric Algebra*, pages 61–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[9] Tomaso Poggio and Federico Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9) :1481–1497, 1990.

[10] Peter Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. volume 93, 01 1993.