Aix-Marseille Université

École Doctorale en Mathématiques et Informatique - ED 184

Laboratoire d'Informatique et Systèmes – UMR 7020

Thèse présentée pour obtenir le grade universitaire de docteur

Discipline : informatique
Spécialité : apprentissage automatique

Approximations parcimonieuses et méthodes à noyaux pour la compression de modèles d'apprentissage

Luc GIFFON

Soutenue le 18/12/2020 devant le jury composé de :

Prof.	Gilles GASSO	INSA Rouen	Rapporteur
Docteur	Nicolas USUNIER	Facebook AI Research	Rapporteur
Prof.	Frédéric BÉCHET	Aix-Marseille Université	Examinateur
Prof.	Marianne CLAUSEL	Université de Lorraine	Examinateur
DR	Rémi GRIBONVAL	INRIA	Examinateur
Prof.	Thierry ARTIÈRES	École Centrale Marseille	Directeur
MCF HDR	Hachem KADRI	Aix-Marseille Université	Co-Directeur
MCF	Stéphane AYACHE	Aix-Marseille Université	Co-Directeur

cette thèse est dédicacée à la
Bis'villa

Résumé

L'explosion de la puissance de calcul ainsi que la collecte massive de données ont conduit à l'actuel âge d'or de la recherche en intelligence artificielle. Cette surabondance de ressources a favorisé la conception d'algorithmes certes impressionnants du point de vue applicatif, mais aussi excessivement demandant en terme de données d'apprentissage et/ou de puissance calculatoire. Cette thèse a pour objectif d'étudier et de valider expérimentalement les bénéfices, en terme de quantité de calcul et de données nécessaires, que peuvent apporter les méthodes à noyaux et les méthodes d'approximation parcimonieuses à des algorithmes d'apprentissage existants.

Les réseaux de neurones convolutifs ont révolutionné le traitement de données structurées telles que les images ou le texte. Cependant, les nouvelles architectures neuronales, toujours plus performantes, contiennent toujours plus de paramètres à régler, ce qui nécessite un très grand volume de données. À l'inverse, les méthodes à noyaux, basées sur l'utilisation de fonctions dites «noyaux», sont moins performantes en pratique que les réseaux neuronaux mais demandent peu de données. On peut donc naturellement se demander s'il est possible de combiner ces techniques pour garder le meilleur des deux mondes. Dans une première partie de cette thèse, nous proposons un nouveau type d'architecture neuronale qui fait intervenir une fonction noyau afin d'en réduire le nombre de paramètres à apprendre, ce qui permet de la rendre robuste au sur-apprentissage dans un régime où peu de données annotées sont disponibles.

La puissance de calcul nécessaire pour faire tourner les algorithmes d'apprentissage récents est étroitement liée à la «complexité» de ces algorithmes. Une approche possible à la réduction de cette complexité est l'utilisation d'approximations parcimonieuses. Dans une seconde partie de cette thèse, nous cherchons à réduire la complexité des modèles d'apprentissage existants en y incluant des approximations parcimonieuses. D'abord, nous proposons un algorithme alternatif à l'algorithme des K-moyennes qui permet d'en accélérer la phase d'inférence grâce à l'expression des centroides sous forme d'un produit de matrices parcimonieuses. En plus des garanties de convergence de l'algorithme proposé, nous apportons une validation expérimentale de la qualité des centroides ainsi exprimés et de leur bénéfice en terme de coût calculatoire. Ensuite, nous explorons la compression de réseaux neuronaux par le remplacement des matrices qui le constituent avec des décomposition parcimonieuses. Enfin, nous détournons l'algorithme d'approximation parcimonieuse *Orthogonal Matching Pursuit* (OMP) pour faire une sélection pondérée des arbres de décision d'une forêt aléatoire, nous analysons l'effet des poids obtenus et proposons par ailleurs une alternative non-négative de la méthode qui surpasse toutes les autres techniques de sélection d'arbres considérées sur un large panel de jeux de données.

Mots clés : méthodes à noyaux, réseaux de neurones, approximations parcimonieuses, forêts aléatoires

Abstract

The explosion of computing power and massive data collection has led to the current golden age of artificial intelligence research. This overabundance of resources has favoured the design of algorithms that are certainly impressive from an application point of view, but also excessively demanding in terms of learning data and/or computing power. This thesis aims at studying and experimentally validating the benefits, in terms of amount of computation and data needed, that kernel methods and sparse approximation methods can bring to existing machine learning algorithms.

Convolutional neural networks have revolutionized the processing of structured data such as images or text. However, the new neural architectures, always more powerful, contain more and more parameters to be adjusted, which requires a very large volume of data. Conversely, kernel methods, based on the use of a function called « kernel», are less efficient in practice than neural networks but require little data. One can therefore naturally wonder whether it is possible to combine these techniques to keep the best of both worlds. In a first part of this thesis, we propose a new type of neural architecture that uses a kernel function to reduce the number of learnable parameters, thus making it robust to overfiting in a regime where few labeled observations are available.

The computational power required to run recent learning algorithms is closely related to the «complexity» of these algorithms. A possible approach to reducing this complexity is the use of sparse approximations. For example, one can try to decompose a matrix into a product of sparse matrices. The final operator obtained from this approximation contains fewer non-zero values than the initial matrix and can therefore be used instead of the initial matrix to calculate low-cost matrix products. Another example: a signal can be approximated by a linear sparse combination of vectors from a dictionary. In a second part of this thesis, we seek to reduce the complexity of existing machine learning models by including sparse approximations. First, we propose an alternative algorithm to the K-means algorithm which allows to speed up the inference phase by expressing the centroids as a product of sparse matrices. In addition to the convergence guarantees of the proposed algorithm, we provide an experimental validation of both the quality of the centroids thus expressed and their benefit in terms of computational cost. Then, we explore the compression of neural networks by replacing the matrices that constitute its layers with sparse matrix products. Finally, we hijack the *Orthogonal Matching Pursuit* (OMP) sparse approximation algorithm to make a weighted selection of decision trees from a random forest, we analyze the effect of the weights obtained and we propose a non-negative alternative to the method that outperforms all other tree selection techniques considered on a large panel of data sets.

Keywords: kernel methods, neural networks, sparse approximation, random forest

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué à la réalisation de cette thèse. Je ne peux nommer toutes les personnes qui m'ont aidé pendant cette thèse car elles sont trop nombreuses, je m'excuse par avance pour mes oublis.

Pour l'avoir rendue possible, je remercie tout le personnel de l'ED184, d' Aix-Marseille Universités, de l'ANR[1], du Ministère de l'enseignement supérieur et de la recherche ainsi que tous ceux qui ont participé à son financement en payant leurs taxes et impôts. Ces travaux ont bénéficié d'un accès aux moyens de calcul de l'IDRIS au travers de l'allocation de ressources 2020-AD011011766 attribuée par GENCI. Je remercie aussi les membres de mon jury de thèse de prendre le temps de lire et évaluer mon travail. Je remercie mes superviseurs, Hachem Kadri, Thierry Artières et Stéphane Ayache d'abord de m'avoir convaincu de commencer cette thèse, puis de m'avoir accompagné tout en me laissant la liberté suffisante pour que je puisse découvrir l'envers du décors du monde universitaire. Je remercie aussi Lionel Spinelli qui m'a fait découvrir le monde de l'informatique et sans qui je n'aurais jamais changé de filière. Merci aussi aux collègues du LIS pour leur accueil et particulièrement : merci à l'équipe QARMA pour leur bienveillance permanente. Ça a vraiment été plaisant de travailler avec toute l'équipe.

Je remercie les membres non-permanents avec qui j'ai passé le plus de temps : merci d'abord à Riikka Huusari pour toute son amitié, toute son aide et sa patience. Merci aussi à mes co-bureaux Ama Marina Krémé, Adrien Meynard, Hung Truong, Dominique Benielli, et Sokol Koço de m'avoir parfois aidé dans mon apprentissage des mathématiques, parfois contredit sur des sujets de société et en général, merci de m'avoir fait évoluer à la fois scientifiquement et humainement. Merci à Charly Lamothe, Léo Bouscarrat, Farah Cherfaoui et Paolo Milanesi avec qui j'ai aimé travailler sur le projet des non-permanents. Merci enfin à Baptiste Bauvin pour sa bonne humeur lors de ses passages parmi nous à Marseille. Je remercie ensuite les chercheurs permanents qui m'ont accompagné pendant le périple de la thèse. Je pense notamment à Valentin Émiya et son exemplarité, son énergie et sa bonne volonté pour animer la vie de l'équipe et du laboratoire. Merci à lui aussi d'avoir pris beaucoup de son temps pour travailler avec moi et participer autant à ma formation de chercheur. Par ailleurs, je remercie aussi Ronan Sicre pour son amitié, ainsi que Rémi Eyraud, Liva Ralaivola, Cécile Capponi, Paul Villoutreix et François-Xavier Dupé qui ont aussi participé à mon

[1]Numéro de dotation : ANR16-CE23-000

enseignement. Merci enfin à Manuel Bertrand pour son efficacité et sa patience en tant que support informatique. Je m'estime réellement chanceux d'avoir passé ces trois ans avec cette équipe dans ce laboratoire qui a grandement participé à mon épanouissement. Je leur souhaite à tous d'être heureux et de la réussite.

A titre plus personnel, j'aimerais aussi remercier les habitants de la Bis'Villa à qui je dédicace cette thèse. Merci de m'avoir toujours accueilli à bras ouverts en toutes circonstances. Bien-sûr, merci aussi à mon amoureuse Alicia, pour son affection indéfectible et toute la reconnaissance qu'elle me donne. Merci de m'accepter et de toujours me soutenir. Merci pour tous ses efforts et toute la confiance qu'elle a en moi. Même si nous ne nous voyons pas beaucoup, merci enfin à ma famille. Merci à mes parents qui m'ont bien éduqué, qui ont financé mes longues études sinueuses, qui m'ont toujours laissé énormément de liberté et surtout, qui m'ont fait confiance. J'espère qu'ils sont fiers de moi.

Table des matières

Introduction

La recherche en Intelligence Artificielle est née au cours d'une session de travail en 1956, à l'université de Darthmouth, au New Hampshire. Sur le document d'invitation à cette session, on peut lire :

> *To proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.*

Cette définition «originelle» de la recherche en Intelligence Artificielle intégrait déjà la notion d'apprentissage automatique. Il semble en effet assez intuitif qu'une machine supposée imiter l'intelligence humaine soit aussi capable d'assimiler de nouvelles connaissances à partir d'expérience.

Plus précisemment, nous envisageons un modèle d'apprentissage automatique comme capable d'apprendre, à partir d'une collection d'exemples d'apprentissage prédéfinis, la relation qui lie une donnée d'entrée à un résultat attendu en sortie. Les modèles d'apprentissage peuvent généralement appartenir à trois catégories : les modèles d'apprentissage par renforcement qui fonctionnent en optimisant la récompense rendue par un environnement en fonction des actions d'un agent, les modèles dits «supervisés» qui ont accès au résultat attendu pour chaque exemple d'entraînement – on dit que les exemples sont étiquetés – et les modèles dits «non-supervisés» qui peuvent apprendre à partir de données non-étiquetés. Par exemple, dans le cas de l'apprentissage supervisé, des images d'animaux peuvent être classifiées par espèces : dans ce cas, le modèle doit apprendre à partir d'un ensemble de couples *(image, espèce)*, quelles sont les informations contenues dans l'image d'un animal qui permettent de décider de son espèce. Dans le cas d'un apprentissage non-supervisé, on peut imaginer une tâche où il faudrait apprendre à regrouper entre elles les images d'animaux qui appartiennent à la même espèce, sans jamais avoir accès à l'espèce de l'animal d'une image ; on parle alors d'une tâche de *regroupement* (*clustering*) où la relation à apprendre est celle entre l'image et le groupe auquel elle appartient. Par «apprendre», nous entendons déterminer itérativement, à partir des données d'apprentissage, les paramètres qui permettront au modèle d'accomplir la tâche désirée sur de nouvelles données de test ; on dit que le modèle doit «généraliser» plutôt que «sur-apprendre» (apprendre par cœur) les données d'entraînement.

Parmi la grande variété des modèles d'apprentissage déjà existants, nous nous intéressons notamment aux modèles dits «non-linéaires», c'est à dire des modèles dont la fonction de prédiction ne peut pas être exprimée par un opérateur

linéaire. Ces modèles sont particulièrement intéressants car beaucoup de problèmes du monde réel nécessitent des fonctions de prédiction non-linéaires pour être résolus. On trouve par exemple les forêts aléatoires, composées d'une collection d'arbres de décisions dans cette catégorie de modèles. Les arbres de décision sont capables d'apprendre à discriminer les données par analyse dichotomique. Ils sont simples à mettre en œuvre, facilement interprétables, mais instables et sujets au sur-apprentissage. Pour compenser cette faiblesse, on entraîne habituellement une multitude d'arbres de décision simples sur la base d'échantillons aléatoires des données d'apprentissage, pour former ce qu'on appelle une forêt aléatoire. Par un effet de moyennage, cette collection d'arbres légèrement différents les uns des autres produit de bonnes performances mais au prix d'un coût calculatoire plus élevé.

Un autre exemple de classe de modèles d'apprentissage est la classe des méthodes à noyaux qui sont une famille de modèles aux fondements théoriques solides et qui reposent sur une fonction dite «noyau» capable de mesurer une forme de similarité entre deux observations. En fait, le résultat du calcul d'une fonction noyau de deux observations est égal au produit scalaire entre leurs représentations dans un espace de représentation, habituellement non-linéaire. Les fonctions noyaux peuvent ainsi être utilisées en lieu et place du produit scalaire dans des algorithmes d'apprentissage linéaire pour obtenir une fonction de prédiction non-linéaire ! C'est le choix de la fonction noyau qui détermine l'espace de représentation des données et donc la capacité de la méthode à résoudre des problèmes non-linéaires. Les méthodes à noyaux fonctionnent bien lorsque peu de données d'apprentissage sont disponibles. Toutefois, pour un ensemble de données de N observations, mesurer la valeur du noyau entre chacune d'elles et enregistrer les résultats dans une matrice nécessite un espace de stockage de l'ordre de $\mathcal{O}(N^2)$ ce qui les rend peu utilisables d'un point de vue pratique dans le contexte du «déluge de données» que nous connaissons actuellement.

Récemment, ce sont plutôt les réseaux de neurones, une autre famille de modèles d'apprentissage, qui ont le vent en poupe. Ces modèles reposent sur des combinaisons de modules élémentaires appelés neurones, dont le fonctionnement est analogue à celui des neurones biologiques. Les neurones sont organisés en couches capables de fournir une représentation non-linéaire de leurs entrées respectives. L'ensemble complet des couches de neurones forment ce qu'on appelle une architecture neuronale. L'efficacité des réseaux de neurones vient de leur capacité mettre au point les paramètres de chaque neurone afin d'apprendre d'eux même une représentation non-linéaire riche des données et accomplir la tâche d'apprentissage. Chaque neurone fait intervenir son propre ensemble de paramètres et donc une architecture neuronale qui contient des centaines voire des milliers de neurones peut rapidement accumuler une très grande quantité de paramètres. Cette sur-paramétrisation des réseaux neuronaux joue certainement un rôle dans leur étonnante capacité à résoudre des tâches, de traitement d'image notamment, lorsque de grandes bases de données sont utilisables. À

l'inverse, lorsque peu de données d'apprentissage sont disponibles, cette surparamétrisation des réseaux neuronaux peut conduire à de mauvais résultats en généralisation.

Cette idée qu'un modèle puisse être «trop complexe» par rapport à un problème d'apprentissage donné n'est pas nouvelle et est même fondamentale en théorie de l'apprentissage (VAPNIK, 1999). Elle peut être associée au principe de raisonnement philosophique qu'est «le rasoir d'Ockham», ou principe de parcimonie qui veut que les hypothèses les plus simples (les moins complexes) soient privilégiées. Ce principe nous amène à chercher des modèles d'apprentissage qui contiennent moins de paramètres à apprendre afin d'en améliorer la capacité de généralisation.

Un autre avantage, plus terre à terre, de ces modèles moins complexes dits «parcimonieux» est le fait qu'ils soient en général moins coûteux à mettre en œuvre. En effet, quoique ce ne soit pas nécessairement toujours le cas, exécuter un modèle qui contient moins de paramètres devrait par ailleurs nécessiter moins de calculs et moins de ressources matérielles, économiques ou environnementales. Par exemple, un produit matrice-vecteur nécessitera d'autant plus de multiplications et d'additions que la dimension est grande. Autre exemple, pour une dimension fixée, une matrice peut être «creuse» (contenir beaucoup de zéros) ou bien même être exprimée en un produit de matrices creuses. Dans ces deux cas, le produit matrice-vecteur associé sera d'autant moins cher qu'il y aura de zéros dans les matrices ou le vecteur.

C'est pour cette raison que tout un pan de la recherche en apprentissage consiste à produire des versions compressées des modèles existants. Typiquement, une direction de recherche consiste à trouver des solutions efficaces pour produire automatiquement des réseaux neuronaux contenant moins de connections, ou moins de neurones, sans impacter trop leur performances. Pour les méthodes à noyaux, il existe des méthodes d'approximations qui permettent d'éviter de calculer la matrice des similarités évoquée plus tôt, ou bien seulement une partie. Enfin, des travaux proposent d'élaguer les forêts aléatoires en ne sélectionnant qu'une partie des arbres qui la composent. Toutefois, ces techniques sont en général considérées séparément et on peut se demander si elles ne pourraient pas être combinées afin que chacune tire bénéfice des avantages des autres.

En vertu du principe de parcimonie, peut on imaginer construire des modèles qui contiendraient moins de paramètres tout en préservant leur efficacité ? Peut-on incorporer des méthodes parcimonieuses à des modèles existants afin de les comprimer ? Peut-on même espérer que ces modèles plus légers en paramètres, soient par ailleurs moins sensible au sur-apprentissage lorsque peu de données d'entraînement annotées sont disponibles ?

Ce sont les problématiques que nous abordons dans cette thèse. Nous proposons et étudions expérimentalement diverses approches de compression de modèles qui font intervenir la parcimonie et/ou des méthodes d'approximation de noyaux.

Aperçu des Chapitres

Chapitre 1 Dans ce chapitre, nous introduisons plus en détails deux familles de modèles d'apprentissage non-linéaire qui sont d'importance prépondérante dans cette thèse : les méthodes à noyaux et les réseaux de neurones. Nous essayons de couvrir les concepts clés de ces modèles pour comprendre le reste du document. Dans les deux cas, nous commençons par décrire le fonctionnement général des méthodes puis des approches existantes pour alléger leur coût calculatoire. De nombreuses références à ce chapitre seront faites tout au long de la thèse.

Chapitre 2 Dans ce chapitre, nous travaillons à l'interface des méthodes à noyau et réseaux de neurones pour construire un modèle appelé réseau Nyström adaptatif qui permet de tirer parti du meilleur des deux mondes. Nous proposons de remplacer les couches denses d'un réseau de neurones par une couche Nyström adaptative qui permet d'apprendre une représentation non-linéaire des données en sortie de convolution. Cette couche possède un nombre très réduit de paramètres et permet donc d'apprendre dans un contexte ou peu de données sont disponibles, en comparaison à des couches denses standards qui ont tendance à sur-apprendre.

Chapitre 3 Dans ce chapitre, nous proposons de remplacer les matrices denses de modèles d'apprentissage communs en produits de matrices parcimonieuses. Cette idée prend son origine dans le constat que des opérateurs linéaires rapides peuvent être exactement exprimés en produit de matrices parcimonieuses et donc peuvent être appris sous cette forme grâce à des algorithmes existants dans la littérature. Ce chapitre se divise en deux contributions : dans la première, nous proposons un algorithme analogue à l'algorithme des K-moyennes mais dont la matrice de centroïdes résultants s'exprime en un produit de matrices parcimonieuses. Nous illustrons l'intérêt de notre méthode avec des exemples d'utilisation de la matrice des centroïdes, notamment dans le cadre de l'approximation de Nyström ; dans la seconde, nous proposons un cadre général où les matrices denses des couches de réseaux neuronaux pré-entraînées sont distillées en des produits de matrices parcimonieuses. Nous avons mené une large gamme d'expériences sur différentes architectures et différents jeux de données pour étudier le fonctionnement de notre méthode.

Chapitre 4 Dans ce chapitre, nous nous intéressons à l'élagage de forêts aléatoires. Nous exprimons le problème de l'élagage d'une forêt sous la forme d'un problème d'optimisation avec contrainte l_0 et nous proposons d'utiliser l'algorithme `OMP` pour le résoudre. Cette approche permet non seulement de sélectionner les arbres d'une forêt aléatoires mais aussi de leur attribuer des poids

d'importance. Dans une série d'expériences, nous montrons l'intérêt de notre méthode et en particulier de l'utilisation de poids d'importance sur les arbres. Nous proposons aussi un variant de notre méthode avec une contrainte de non-négativité sur les poids ce qui permet d'améliorer encore les performances du modèle, le rendant ainsi meilleur que toutes les autres techniques concurrentes évaluées.

Liste des publications

Révisions en cours :

- Luc Giffon, Valentin Emiya, Hachem Kadri, Liva Ralaivola. *QuicK-means Accelerating Inference for K-Means by Learning Fast Transforms*. Soumis pour révision au *Machine Learning Journal* en Juillet 2020.

Publié en conférence internationale avec évaluation par les pairs :

- Luc Giffon, Stéphane Ayache, Thierry Artières, Hachem Kadri. *Deep Networks with Adaptive Nyström Approximation.* Publié dans *The International Joint Conference on Neural Networks* (IJCNN) en 2019 à Budapest, Hongrie. (conference niveau A d'après le classement ERA)

Publié en conférence francophone avec évaluation par les pairs :

- Luc Giffon, Stéphane Ayache, Thierry Artières, Hachem Kadri. *Emulsion de noyaux et d'apprentissage profond*. Publié dans la Conférence sur l'Apprentissage automatique (CAp), Rouen, France, 2018.
- Luc Giffon, Charly Lamothe, Léo Bouscarrat, Paolo Milanesi, Farah Cherfaoui, Sokol Koço. *Pruning Random Forest with Orthogonal Matching Trees*. Publié dans la Conférence sur l'Apprentissage automatique (CAp), Vannes, France, 2020.

Notations courantes

Nous donnons en Table 0.1 les conventions générales d'écriture mathématique utilisées dans cette thèse ainsi que quelques abréviations courantes en Table ??. Chaque chapitre a son propre lot d'abréviations qui lui sont propres mais nous introduisons aussi ici certaines notations utiles qui ne changent pas entre les chapitres.

Description	Notation	Sens
lettres latines minuscule	$i, j, k, ...$	indices
lettres latines majuscule	$N, M, W, ...$	constantes entières
lettres grecques minuscules	$\alpha, \beta, \gamma, ...$	scalaires
caractères minuscules avec indice	$a_i, b_i, \alpha_i, ...$	$i^{ème}$ élément du vecteur
caractères gras en minuscule	$\mathbf{a}, \mathbf{b}, \boldsymbol{\alpha}, ...$	vecteurs ou fonctions
caractères gras en minuscule avec indice	$\mathbf{a}_i, \mathbf{b}_i, \boldsymbol{\alpha}_i, ...$	$i^{ème}$ vecteur ligne de la matrice
caractères gras en majuscule	$\mathbf{A}, \mathbf{B}, \mathbf{C}, ...$	matrices
caractère gras en majuscule avec indice	$\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i, ...$	$i^{ème}$ matrice
exposant -1	$\mathbf{M}^{-1}$	inverse d'une matrice
dague	$\mathbf{M}^{\dagger}$	pseudo-inverse d'une matrice
chevrons	$\langle \cdot, \cdot \rangle_{\mathbf{M}}$	produit scalaire dans l'espace engendré par $\mathbf{M}$
barres verticales indice zéro	$\| \cdot \|_0$	norme zéro d'un vecteur ou d'une matrice
barres verticales indice 2	$\|\mathbf{v}\|_2$	norme deux pour un vecteur
barres verticales indice 2	$\|\mathbf{M}\|_2$	norme opérateur pour une matrice
barres verticales indice F	$\| \cdot \|_F$	norme de Frobenius
demi-crochets haut	$\lceil \cdot \rceil$	arrondi supérieur
demi-crochets bas	$\lfloor \cdot \rfloor$	arrondi inférieur
point encerclé	$\odot$	produit point à point (produit de Hadamard)
doubles crochets	$[\![N]\!]$	liste des entiers de 1 à N
Grand O calligraphié	$\mathcal{O}(\cdot)$	Mesure de complexité
Grand N calligraphié	$\mathcal{N}(\mu, \sigma^2)$	Loi Normale de moyenne μ et écart-type σ

TABLE 0.1 : Notations courantes.

1 Représentations non-linéaires dans les modèles d'apprentissage

Contents

1.1 Introduction

Les méthodes à noyaux et l'apprentissage profond sont deux familles de méthodes d'apprentissage qui ont essentiellement été étudiées séparément. Les mé-

thodes d'apprentissage profond utilisent les réseaux neuronaux pour apprendre des caractéristiques pertinentes des données et nécessitent un grand volume de données pour fonctionner. Les méthodes à noyaux quant à elles sont des outils puissants pour l'apprentissage de relations non linéaires dans les données et sont bien adaptées aux problèmes liés à des jeux de données de tailles limités. Leur puissance et leur popularité viennent de leur capacité à étendre les méthodes linéaires à des problèmes non linéaires avec des garanties théoriques mais aussi de leur aptitude à travailler avec des données non-vectorielles (graphes, par exemple). Toutefois la complexité calculatoire des méthodes à noyaux les rend difficile à utiliser avec de grandes bases de données.

Un élément essentiel du fonctionnement de ces méthodes est leur capacité à construire une représentation non-linéaire des données d'apprentissage afin de construire des fonctions de décision riches. Dans le cas des méthodes à noyaux, cette caractéristique repose sur le choix d'une fonction noyau adaptée ; alors que pour les réseaux neuronaux, c'est l'empilement minutieux de couches de calcul spécialisées qui permet au modèle d'apprendre une représentation non-linéaire à partir des données.

Cependant, le coût calculatoire de ces deux familles de modèles peut devenir rédhibitoire en pratique. Les méthodes à noyaux peuvent difficilement passer à l'échelle des quantités de données actuelles du fait de leur complexité quadratique par rapport à la taille du jeu de données. Les réseaux de neurones, eux, apprennent une quantité de paramètres si grande qu'il devient impossible de les déployer sur de petits appareils qui n'embarquent pas un grand espace de stockage.

Dans cette section introductive sur les méthodes d'apprentissage non-linéaires, nous commençons par revoir des concepts élémentaires relatifs à l'apprentissage avec les méthodes à noyaux, nous ferons apparaître leurs problèmes de complexité calculatoire et décrirons en détails des approches connues pour les résoudre. Ensuite, nous aurons une approche similaire avec les réseaux de neurones dont nous introduirons le fonctionnement essentiel avant de présenter certaines grandes familles de méthodes pour en réduire la complexité calculatoire.

1.2 Méthodes à noyaux

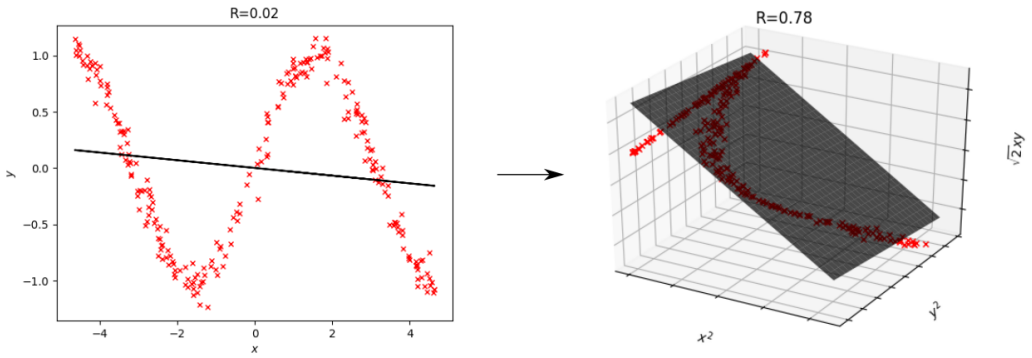


FIGURE 1.1 : Projection en 3D de la représentation non-linéaire implicite d'un noyau polynomial d'ordre 2. Une régression linéaire sur la représentation initiale des données ne donne pas de bons résultats. La régression linéaire dans l'espace de représentation induit par le noyau donne un bien meilleur coefficient de corrélation R.

Il existe plusieurs voies pour introduire les noyaux et la façon dont ils peuvent être utilisés en apprentissage. Nous choisissons de suivre une approche similaire à celle utilisée par CORNUÉJOLS et MICLET, 2011 qui introduisent les noyaux pour la résolution d'un problème de régression en passant par un espace de représentation des données.

Étant donné un ensemble d'apprentissage $\mathbf{X} := [\mathbf{x}_i]_{i=1}^N \in \mathcal{X} \subseteq \mathbb{R}^{N \times D}$ et le vecteur des étiquettes associées $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^N$, le problème de régression linéaire consiste à trouver une fonction de décision $\mathbf{h}$ qui pour toute donnée d'entrée $\mathbf{x}$ est capable de prédire son étiquette grâce à l'équation :

$$\mathbf{h}(\mathbf{x}) = \sum_{i=0}^{D} x_i w_i, \tag{1.1}$$

où les w_i sont les composantes d'un vecteur de poids $\mathbf{w}$ et $x_0 := 1$ pour tout $\mathbf{x}$. Résoudre le problème de régression revient à trouver le vecteur de poids $\mathbf{w}$ tel qu'un certain critère d'erreur empirique (classiquement le critère Moyenne des Erreurs au Carré (MSE) : $\frac{1}{N}\sum_i^N(\mathbf{h}(\mathbf{x}_i) - y_i)^2$) sur les données d'apprentissage soit minimum, c'est-à-dire lorsque la dérivée de l'erreur est nulle. À partir de là, on obtient la valeur optimale de $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\boldsymbol{\alpha}$ qui peut donc être vue comme une combinaison linéaire des données d'apprentissage avec $\boldsymbol{\alpha} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-2}\mathbf{X}^T\mathbf{y}$. Et donc la fonction de décision $\mathbf{h}$ peut être ré-écrite de façon

à ne dépendre que de produits scalaires entre les données d'apprentissage et la donnée d'entrée $\mathbf{x}$:

$$\mathbf{h}(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle. \tag{1.2}$$

Si le problème de régression que l'on doit résoudre nécessite une fonction de décision non-linéaire, alors une solution consiste à utiliser au préalable une fonction de représentation non-linéaire $\boldsymbol{\phi}$ sur les données afin de les placer dans un espace où on peut utiliser des outils de régression linéaire classique (Voir Figure 1.1 par exemple). La fonction de décision $\mathbf{h}$ devient donc :

$$\mathbf{h}(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}) \rangle. \tag{1.3}$$

L'enjeu est alors de trouver la bonne fonction de représentation $\boldsymbol{\phi}$ pour la tâche d'apprentissage et c'est ici que les fonctions noyaux entrent en scène. Dans le reste de cette section, nous commençons par donner une définition formelle des noyaux et des espaces de représentation qui leur sont associés. Ensuite nous donnerons quelques exemples classiques de fonctions noyaux. Enfin, nous exposerons les problèmes de complexité calculatoire inhérents aux méthodes à noyaux et nous détaillerons des solutions d'approximation de noyaux pour les résoudre.

1.2.1 Fonctions noyau et Espace de Hilbert à noyau reproduisant

Un noyau est une fonction qui permet de calculer la valeur d'un produit scalaire entre deux observations dans un nouvel espace de représentation $\mathcal{H}$ appelé Espace de Hilbert à Noyaux Reproduisant (RKHS). En quelques sortes, un noyau est donc une mesure de similarité entre deux observations :

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{y}) \rangle_{\mathcal{H}}, \tag{1.4}$$

où la fonction $\boldsymbol{\phi}$ peut être une fonction de représentation non-linéaire et l'espace de représentation $\mathcal{H}$ peut être de dimension infinie. Plus précisément, les fonctions noyaux et les RKHS sont définis comme suit :

Définition 1.1 (fonction noyau)**.** *Une fonction noyau, ou noyau, est une fonction* $\mathbf{k} : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ *telle que :*

- $\mathbf{k}$ *est symétrique, c'est à dire que* $\mathbf{k}(\mathbf{x}, \mathbf{y}) = \mathbf{k}(\mathbf{y}, \mathbf{x})$ *;*
- $\mathbf{k}$ *est positive semi-définie, c'est à dire que* $\sum_{i,j} u_i u_j \mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) \geq 0$.

Définition 1.2 (espace de Hilbert à noyau reproduisant (RKHS))**.** *Un espace fonctionnel de Hilbert* $\mathcal{H} := \{\mathbf{f} : \mathcal{X} \mapsto \mathbb{R}\}$ *est dit «à noyau reproduisant» s'il existe une fonction noyau* $\mathbf{k}$ *telle que :*

- $\forall x \in \mathcal{X},\ \mathbf{k}(\mathbf{x}, \cdot) \in \mathcal{H}$ *;*
- $\forall x \in \mathcal{X}$ *et* $\forall \mathbf{f} \in \mathcal{H},\ \langle \mathbf{f}, \mathbf{k}(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} = \mathbf{f}(\mathbf{x})$ *(propriété reproduisante).*

En particulier, on remarque que $\mathbf{k}(\mathbf{x}, \mathbf{y}) = \langle \mathbf{k}(\mathbf{x}, \cdot), \mathbf{k}(\mathbf{y}, \cdot)\rangle$ ce qui nous ramène à la définition de noyau de l'Équation (1.4) et nous donne $\forall \mathbf{x} \in \mathcal{X},\ \boldsymbol{\phi}(\mathbf{x}) = \mathbf{k}(\mathbf{x}, \cdot)$. On note que dans la littérature des noyaux, une fonction noyau n'est pas nécessairement Positive Semi-Définie (PSD) ou symétrique mais dans le reste de cette thèse, lorsque nous parlons de noyaux, nous faisons toujours référence aux noyaux PSD et symétriques comme définis dans la Définition 1.1. On voit que les notions de noyau et de RKHS sont intimement liées dans ces définitions. En fait, le théorème de Moore-Aronszajn (Aronszajn, 1950) affirme que pour toute fonction noyau PSD et symmétrique, il existe un RKHS associé et vice-versa. Nous ne nous aventurons pas plus dans les technicités relatives aux fonctions noyau car elles dépassent le cadre de cette thèse et les définitions déjà données sont suffisantes pour comprendre le reste de ce document ; davantage de détails peuvent être trouvés dans le livre de référence de Smola et Schölkopf, 1998.

Une fonction noyau peut donc se substituer au produit scalaire dans l'Équation (1.3) afin d'apprendre une fonction de décision linéaire après re-description des données dans le RKHS associé. Plus généralement, grâce à l'équivalence de l'Équation (1.4), les noyaux peuvent simplement être utilisé pour introduire une non-linéarité dans n'importe quelle méthode qui ferait normalement intervenir le produit scalaire (Smola et Schölkopf, 1998) tout en conservant ses éventuelles garanties théoriques. Cette équivalence entre fonction noyau et produit scalaire est communément appelée «l'astuce du noyau» (*the kernel trick*). Elle a rendu les méthodes à noyaux célèbres car en pratique, elle permet d'obtenir une représentation des données en très grande dimension —potentiellement infinie !— sans avoir à calculer explicitement cette représentation : tous les calculs sont faits dans l'espace de description initial. Nous nous contentons de définir les fonctions noyaux ici et ne décrivons pas les algorithmes d'apprentissage qui utilisent les noyaux et renvoyons plutôt le lecteur vers le livre de référence de Shawe-Taylor et Cristianini, 2004.

Pour finir, nous faisons remarquer que les noyaux ne nécessitent pas de travailler sur des données vectorielles. Ceci ne sera pas plus développé dans cette thèse mais on peut définir des noyaux pour mesurer des similarités entre graphes par exemple (Vishwanathan, Schraudolph, Kondor et al., 2010 ; Shervashidze, Schweitzer, Van Leeuwen et al., 2011). Nous verrons dans la section suivante quelques exemples de fonctions noyaux utiles.

1.2.2 Construire des fonctions noyau

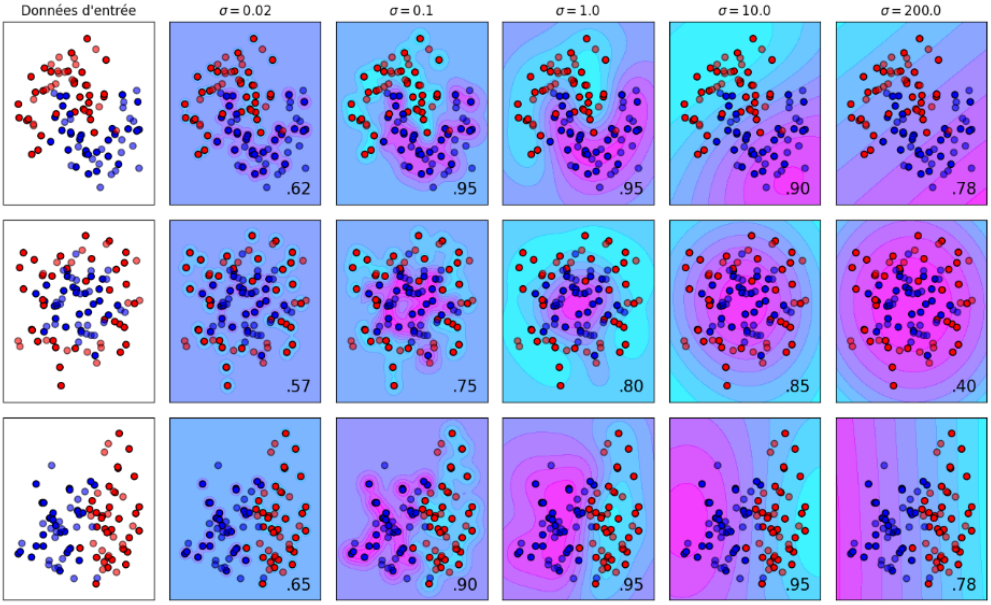


FIGURE 1.2 : Effet du paramètre σ sur l'apprentissage d'un modèle de classification utilisant un noyau Gaussien sur trois jeux de données synthétiques. Le numéro en bas à droite représente le pourcentage de précision de classification sur les données test. Les données test sont représentées avec des points de couleur atténuée. On voit qu'une valeur de σ plus petite conduit à un sur-apprentissage des données d'entraînement alors qu'une valeur de σ trop élevée conduit à un sous-apprentissage. Source : *Support Vector Machine Hyperparameter Tuning - A Visual Guide*

Puisque la fonction noyau définit l'espace de représentation des données, il est nécessaire de choisir minutieusement quelle fonction noyau utiliser pour une tâche d'apprentissage donnée. Ce choix peut revenir à un expert de la tâche qui saurait comment la similarité entre deux observations devrait être calculée ; sinon, des noyaux classiques déjà élaborés peuvent être ré-utilisés ou combinés entre eux ; ou encore le noyau peut même être appris pour correspondre à la tâche. Dans cette Section, nous commençons par donner quelques exemples de fonctions noyaux habituellement utilisées puis nous énonçons quelques règles utiles pour la combinaison de noyaux et enfin nous montrons rapidement la forme de quelques noyaux qui peuvent être appris. Commençons avec quelques exemples de noyaux classiques (une liste plus riche peut être trouvée sur le site de COUZA, 2020) :

Noyau linéaire. Le noyau le plus simple de tous correspond au produit scalaire direct entre les vecteurs d'entrée :

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle. \tag{1.5}$$

Dans ce cas, il n'y a pas de non-linéarité et l'utilisation d'un tel noyau permet de retrouver les méthodes linéaires classiques.

Noyau polynomial. A partir du noyau linéaire, on peut simplement construire le noyau polynomial d'ordre P qui correspond essentiellement à une somme de produits de noyaux linéaires. Intuitivement, l'espace de représentation de ce noyau utilise des combinaisons des caractéristiques initiales des données pour créer de nouvelles dimensions. Ce noyau fait intervenir une constante α qui sert de variable d'ajustement pour accorder plus ou moins d'importance aux termes d'ordre supérieur dans le polynome. La Figure 1.1 montre l'effet de la fonction de représentation implicite d'un noyau polynomial d'ordre 2.

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + \alpha)^P. \tag{1.6}$$

Noyau Gaussien. Le noyau Gaussien est sûrement le noyau le plus populaire en apprentissage automatique du fait des performances souvent très bonnes qu'il obtient en pratique. Il s'agit d'un noyau Fonction de base radiale (RBF), c'est-à-dire qu'il est invariant par translation des données d'entrée ; il dépend seulement de la distance entre ses entrées.

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = \exp\Big(-\frac{||\mathbf{x} - \mathbf{y}||^2}{\sigma^2}\Big). \tag{1.7}$$

Il est paramétré par une valeur σ appelée «largeur de bande» qui doit correspondre à une valeur de dispersion des données. Le paramètre σ peut être interprété comme un paramètre de sensibilité à la distance : si σ est petit alors le noyau pourra détecter des variations fines de distance et vice-versa s'il est grand. L'effet du paramètre σ sur la qualité de l'apprentissage est illustré en Figure 1.2. On peut classiquement utiliser une méthode de recherche exhaustive pour trouver le paramètre σ idéal pour une tâche d'apprentissage mais une heuristique commune pour décider de la valeur σ du noyau est d'utiliser la moyenne des écarts entre les données (ROJO-ÁLVAREZ, MARTÍNEZ-RAMÓN, MARÍ et al., 2018). Pour un ensemble de données $\mathbf{X}$ de taille N, l'heuristique donne :

$$\sigma = \frac{1}{N} \sum_{i,j=1}^{N} ||\mathbf{x}_i - \mathbf{x}_j||^2 \tag{1.8}$$

Comme évoqué dans la Section 1.2.1, certains noyaux ne sont pas définis positifs mais seulement conditionnellement définis positifs, c'est à dire que

$\sum_{i,j} u_i, u_j \mathbf{k}(\mathbf{x}, \mathbf{y}) \geq 0$ *si* $\sum_i u_i = 0$, ou même pas définis positifs. Ces noyaux sont appelés noyaux de Krein (ALPAY, 1991 ; ONG, MARY, CANU et al., 2004 ; OGLIC et GÄRTNER, 2018). Ils sont moins contraints que les noyaux PSD et symétriques et généralisent la notion de noyau. Même si la théorie des RKHS ne tient pas avec ces noyaux, ils peuvent être utilisés comme des noyaux PSD et donner de bonnes performances en pratique. Nous en donnons aussi quelques exemples :

Noyau χ^2. Le noyau χ^2, basé sur la distance χ^2 est capable de mesurer une distance entre deux histogrammes. Il est particulièrement apprécié en vision par ordinateur (LI, SAMORODNITSK et HOPCROFT, 2013 ; ZHANG, MARSZAŁEK, LAZEBNIK et al., 2007) :

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = -\sum_{i=1}^{D} \left(\frac{(x_i - y_i)^2}{x_i + y_i} \right) \tag{1.9}$$

Noyau tangente hyperbolique. Le noyau tangeante hyperbolique, aussi connu sous le nom de noyau sigmoïde, est connecté au domaine des réseaux de neurones car il permet de construire des modèles à noyau équivalents à des réseaux de neurones à deux couches avec une fonction d'activation sigmoïde (LIN et LIN, 2003). L'équation du noyau est

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = tanh(\alpha \mathbf{x}^T \mathbf{y} + \beta) \tag{1.10}$$

où α et β correspondent à la pente et l'ordonnée à l'origine de la sigmoide.

A présent que quelques noyaux de base ont été définis, nous voyons comment les composer entre eux pour construire de nouveaux noyaux. La Définition 1.1 donne deux conditions suffisantes pour qu'une fonction soit dite «noyau». Il est donc possible de composer des noyaux entre eux pour en construire de nouveaux, pour peu que cette composition respecte les conditions de symétrie et positivité. Le livre de BISHOP, 2006, Chap. 6 contient une liste plus complète de règles de compositions de noyaux mais nous en donnons quelques unes ici qui nous seront utiles plus loin dans la thèse. Pour deux observations $\mathbf{x}, \mathbf{y}$:

Lemme 1.1. *La somme de deux noyaux,* $\mathbf{k}_1(\mathbf{x}, \mathbf{y}) + \mathbf{k}_2(\mathbf{x}, \mathbf{y})$ *est un noyau.*

Lemme 1.2. *Soit* $\mathbf{x}_a, \mathbf{x}_b$ *deux vues différentes de la même entité* $\mathbf{x}$*. La somme de deux noyaux travaillant sur des vues différentes,* $\mathbf{k}_1(\mathbf{x}_a, \mathbf{y}_a) + \mathbf{k}_2(\mathbf{x}_b, \mathbf{y}_b)$ *est un noyau.*

Lemme 1.3. *La multiplication de deux noyaux* $\mathbf{k}_1(\mathbf{x}, \mathbf{y})\mathbf{k}_2(\mathbf{x}, \mathbf{y})$ *est un noyau.*

1.2.2.1 Apprendre des fonctions noyau

Les méthodes à noyaux fonctionnent bien lorsque le noyau utilisé est bien adapté à la tâche d'apprentissage. Une approche consiste à laisser l'utilisateur choisir la fonction noyau adaptée ou bien la construire sur mesure, comme nous l'avons vu en Section 1.2.2 mais il est aussi possible de traiter le choix de la fonction noyau comme une tâche d'apprentissage.

ABBASNEJAD, RAMACHANDRAM et MANDAVA, 2012 ont écrit une revue détaillée des différentes approches pour l'apprentissage de noyaux. Nous ne survolons ici que certaines techniques utiles pour comprendre certaines contributions de cette thèse.

Certains noyaux sont fonction d'une mesure de distance entre les observations. Le noyau Gaussien, par exemple, est calculé en fonction de la distance euclidienne standard. Une approche de l'apprentissage de noyau consiste à apprendre une métrique $\mathbf{d}$ sur laquelle le noyau s'appuie pour calculer la similarité (WEINBERGER et SAUL, 2009). Pour une distance de Mahalanobis, par exemple, on a :

$$\mathbf{d}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{M} (\mathbf{x} - \mathbf{y})}. \tag{1.11}$$

où la matrice $\mathbf{M}$ définit la métrique à apprendre et donc le noyau.

Dans le cas où plusieurs noyaux peuvent être utilisés, il peut être intéressant d'en apprendre une combinaison. Cette méthode s'appelle Apprentissage de Noyaux Multiple (MKL). Typiquement, la combinaison se fait sous la forme d'une somme

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{K} \beta_i \mathbf{k}^i(\mathbf{x}, \mathbf{y}), \tag{1.12}$$

où l'apprentissage des coefficients $\beta \geq 0$ se fait souvent conjointement à la tâche d'apprentissage (GONEN et ALPAYDIN, 2011).

1.2.3 Méthodes d'approximation de noyaux

Bien que les méthodes à noyaux soient particulièrement adaptées au cas de figure où peu de données sont disponibles, elles deviennent irréalisables lorsque le nombre de données disponible grandit. En effet, leur fonctionnement exige le stockage d'une matrice de Gram, ou matrice noyau, $\mathbf{K}$ telle que : à partir d'un jeu de données $\mathbf{X} := [\mathbf{x}_i]_{i=1}^N$, $\forall i, j \in [\![N]\!]$ $\mathbf{K}_{i,j} = \mathbf{k}(\mathbf{x}_i, \mathbf{x}_j)$. Cette matrice a une complexité en espace de l'ordre de $\mathcal{O}(N^2)$.

Étonnamment, une solution privilégiée pour contourner ce problème est de se passer de l'astuce du noyau pour, à la place, approximer la fonction noyau en déterminant une fonction de représentation explicite. Formellement, l'idée est de déterminer une fonction de représentation $\tilde{\phi}$ explicite telle que :

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{y}) \rangle \approx \langle \tilde{\boldsymbol{\phi}}(\mathbf{x}), \tilde{\boldsymbol{\phi}}(\mathbf{y}) \rangle. \tag{1.13}$$

Pour se faire, deux approches sont particulièrement utilisées : d'abord la méthode des caractéristiques aléatoires (RAHIMI et RECHT, 2008) (*random features*) qui se base sur la projection des caractéristiques d'entrée dans un espace de caractéristiques à faible dimension où les produits scalaires des points dans cet espace se rapprochent de la fonction du noyau ; ensuite la méthode de Nyström (WILLIAMS et SEEGER, 2001) qui est habituellement plutôt utilisée pour approximer la matrice noyau mais qui peut être adaptée pour obtenir une fonction de représentation explicite permettant d'approximer la fonction noyau elle-même.

Dans le reste de cette section, nous allons voir en détail le fonctionnement de ces deux méthodes ainsi que des stratégies existantes pour les accélérer.

1.2.3.1 Caractéristiques aléatoires

Les méthodes d'approximation de noyaux à base de caractéristiques aléatoires utilisent des matrices aléatoires dont les valeurs sont tirées suivant une loi de probabilité dépendante de la fonction noyau à approximer. Ces méthodes sont extrêmement simple à mettre en œuvre en pratique mais elles ont le défaut de ne fonctionner que pour certains types de noyaux, notamment (mais pas uniquement) les noyaux de types RBF, c'est à dire des noyaux qui ne dépendent que de la distance entre leurs données d'entrées : $\mathbf{k}(\mathbf{x}, \mathbf{y}) = \mathbf{k}(\mathbf{x} - \mathbf{y})$. Pour ces noyaux, la méthode repose sur le théorème de Bochner (RUDIN, 1962) :

Théorème 1.1 (Bochner). *Une fonction noyau $\mathbf{k}(\mathbf{x}, \mathbf{y}) = \mathbf{k}(\mathbf{x} - \mathbf{y})$ sur $\mathbb{R}^D$ est positive semi définie si et seulement si $\mathbf{k}(\boldsymbol{\delta})$ est la transformée de Fourrier d'une mesure non-négative.*

Étant donné p une densité de probabilité sur $\mathbb{R}^D$, le théorème de Bochner permet d'y associer un noyau RBF $\mathbf{k}$:

$$\mathbf{k}(\mathbf{x} - \mathbf{y}) = \int_{\mathbb{R}^D} p(\boldsymbol{\omega}) \exp(j\boldsymbol{\omega}^T(\mathbf{x} - \mathbf{y}))d\boldsymbol{\omega} = \mathbb{E}_\omega \left[\boldsymbol{\phi}_\omega(\mathbf{x})^* \boldsymbol{\phi}_\omega(\mathbf{y})\right] \tag{1.14}$$

où j correspond à l'unité imaginaire et $\boldsymbol{\phi}_\omega(\mathbf{x}) = \exp(j\boldsymbol{\omega}^T\mathbf{x})$. Toutefois, nous cherchons une fonction de représentation à valeurs réelles. Comme la distribution de probabilités de $\boldsymbol{\omega}$ est à valeurs réelles et $\mathbf{k}(\mathbf{x} - \mathbf{y})$ est aussi réel, alors on peut écrire :

$$\mathbf{k}(\mathbf{x} - \mathbf{y}) = \int_{\mathbb{R}^D} p(\boldsymbol{\omega}) \cos(\boldsymbol{\omega}^T(\mathbf{x} - \mathbf{y}))d\boldsymbol{\omega}, \tag{1.15}$$

où la fonction $\cos$ est appliquée composante par composante. Enfin, la solution proposée par RAHIMI et RECHT, 2008 pour ϕ_ω est

$$\phi_\omega(\mathbf{x}) = [\cos(\boldsymbol{\omega}\mathbf{x})\sin(\boldsymbol{\omega}\mathbf{x})]^T \tag{1.16}$$

et donc la fonction de représentation qui utilise des caractéristiques aléatoires pour approximer le noyau $\mathbf{k}$ grâce au produit scalaire dans un espace en D' dimensions est :

$$\phi_{rf}(\mathbf{x}) = \frac{1}{D'}[\cos(\mathbf{x}\mathbf{W})\sin(\mathbf{x}\mathbf{W})]^T \tag{1.17}$$

où les éléments de la matrice $\mathbf{W} \in \mathbb{R}^{D\times D'}$ sont échantillonnés suivant la loi de probabilité adaptée au noyau à approximer. Dans le cas du noyau Gaussien par exemple, la loi de probabilités à utiliser est la loi Normale $\mathcal{N}(0,\sigma^2)$.

Pour obtenir la valeur approximée du noyau $\mathbf{k}$ pour un couple d'exemples, il suffit de calculer :

$$\mathbf{k}(\mathbf{x},\mathbf{y}) \approx \langle \phi_{rf}(\mathbf{x}), \phi_{rf}(\mathbf{y})\rangle. \tag{1.18}$$

On note que l'apparition du sin dans l'Équation (1.17) prend son importance dans ce produit scalaire grâce à la règle de trigonométrie $cos(\boldsymbol{\omega}\mathbf{x} - \boldsymbol{\omega}\mathbf{y}) = \cos(\boldsymbol{\omega}\mathbf{x})\cos(\boldsymbol{\omega}\mathbf{y}) + \sin(\boldsymbol{\omega}\mathbf{x})\sin(\boldsymbol{\omega}\mathbf{y})$ qui permet de retrouver le $\cos(\boldsymbol{\omega}(\mathbf{x}-\mathbf{y}))$ de l'Équation (1.15).

Nous venons de montrer comment construire les caractéristiques aléatoires pour approximer des noyaux RBF mais il est aussi possible de construire une approximation pour certains autres types de noyaux, par exemple les noyaux polynomiaux comme l'ont montré KAR et KARNICK, 2012.

Quoique simple à mettre en œuvre, la fonction de représentation de l'Équation (1.17) nécessite le stockage de la matrice aléatoire $\mathbf{W}$ qui coûte $\mathcal{O}(DD')$ et le calcul du produit matrice-vecteur $\mathbf{W}$ qui a aussi un coût $\mathcal{O}(DD')$. Lorsque la dimension d'entrée des données et/ou la dimension de représentation est grande, ces coûts calculatoires peuvent devenir prohibitifs.

Accélération du calcul des caractéristiques aléatoires : *Fastfood* Une solution pour réduire le coût en temps et en espace des caractéristiques aléatoires consiste à utiliser l'approximation de *Fastfood* proposée par LE, SARLÓS et SMOLA, 2013. Cette méthode repose sur l'expression de la matrice $\mathbf{W}$ en un produit de matrices peu chères en calcul et en stockage mais qui offre des garanties d'approximation de certains noyaux tels que le noyau Gaussien :

$$\mathbf{W} = \frac{1}{\sigma\sqrt{D}}\mathbf{SHG\Pi HB} \tag{1.19}$$

où :

- $\mathbf{S}$ est une matrice diagonale qui contient des facteurs d'échelles tirés aléatoirement ;
- $\mathbf{H}$ est la matrice de Hadamard (voir Section 3.2.1) ;
- $\mathbf{G}$ est une matrice diagonale d'éléments aléatoires tirés suivant une loi Normale $\mathcal{N}(0, 1)$;
- $\mathbf{\Pi}$ est une matrice de permutation aléatoire ;
- $\mathbf{B}$ est une matrice diagonale de $+1$ et de -1.

et donc la transformation de `Fastfood` $\boldsymbol{\phi}_{ff}$ d'un vecteur d'entrée $\mathbf{x}$ est donnée par :

$$\boldsymbol{\phi}_{ff}(\mathbf{x}) = \left[cos\left(\frac{1}{\sigma\sqrt{D}}\mathbf{SHG\Pi HBx}\right), sin\left(\frac{1}{\sigma\sqrt{D}}\mathbf{SHG\Pi HBx}\right)\right]. \tag{1.20}$$

Au final, stocker les matrices parcimonieuses qui forment le produit de l'Équation (1.19) plutôt que $\mathbf{W}$ permet de réduire la complexité en stockage de la matrice de poids $\mathbf{W}$ à $\mathcal{O}(D)$ grâce à la définition récursive de la matrice de Hadamard. De plus, la complexité en calcul du produit $\mathbf{Wx}$ est réduite à $\mathcal{O}(DlogD)$ grâce aux algorithmes de transformation rapide de Hadamard. Pour obtenir une dimension de représentation D' plus grande que deux fois la dimension d'entrée D, il suffit d'empiler empilant plusieurs $\mathbf{W}$ différents, définis suivant l'Équation 1.20.

1.2.3.2 Approximation de Nyström

L'approximation de Nyström est une approximation de rang faible de la matrice noyau qui est calculée en échantillonnant un sous-ensemble des exemples du jeu de données d'entraînement.

Considérons un sous-ensemble $\mathbf{L} := [\mathbf{l}_i]_{i=1}^{L}$ de l'ensemble des données $\mathbf{X}$. En supposant que ce sous-ensemble de points de références inclut les premiers échantillons de $\mathbf{X}$, ou en réorganisant les échantillons d'entraînement tels que le sous-ensemble $\mathbf{L}$ apparaisse en premier, $\mathbf{K}$ peut être ré-écrit tel que :

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{21}^T \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix}, \tag{1.21}$$

où $\mathbf{K}_{11}$ correspond à la matrice noyau calculée sur le sous ensemble de données $\mathbf{L}$. Quand les éléments de $\mathbf{K}_{11}$ et $\mathbf{K}_{21}$ sont connus, on peut approximer la partie manquante $\mathbf{K}_{22}$ grâce à l'approximation de Nyström WILLIAMS et SEEGER, 2001 :

$$\mathbf{K} \approx \tilde{\mathbf{K}} = \begin{bmatrix} \mathbf{K}_{11} \\ \mathbf{K}_{21} \end{bmatrix} \mathbf{K}_{11}^{-1} \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{21}^T \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{21}^T \\ \mathbf{K}_{21} & \mathbf{K}_{21}\mathbf{K}_{11}^{-1}\mathbf{K}_{21}^T \end{bmatrix}. \tag{1.22}$$

Or, nous savons par l'Équation (1.13) que $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T \approx \tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^T$ où $\mathbf{\Phi}$ est la matrice des représentations par ϕ des exemples du jeu de données, c'est-à-dire que pour tout i, la $i^{ème}$ ligne $\mathbf{\Phi}$, $\boldsymbol{\phi}_i$ est la représentation de $\mathbf{x}_i$, $\boldsymbol{\phi}(\mathbf{x}_i)$. Une solution possible de $\tilde{\mathbf{\Phi}}$ est donc :

$$\tilde{\mathbf{\Phi}} = \begin{bmatrix} \mathbf{K}_{11} \\ \mathbf{K}_{21} \end{bmatrix} \mathbf{K}_{11}^{-1/2}, \tag{1.23}$$

d'où découle l'expression de la fonction de représentation $\boldsymbol{\phi}_{nys}$ sur un seul exemple $\mathbf{x}$:

$$\boldsymbol{\phi}_{nys}(\mathbf{x}) = \mathbf{k}_{\mathbf{x},\mathbf{L}}\mathbf{K}_{11}^{-1/2}, \tag{1.24}$$

où $\mathbf{k}_{\mathbf{x},\mathbf{L}} := [\mathbf{k}(\mathbf{x}, \mathbf{l}_1), \ldots, \mathbf{k}(\mathbf{x}, \mathbf{l}_L)]$ pour tout $\mathbf{l}_i \in \mathbf{L}$

Méthodes d'échantillonage La méthode de Nyström est dépendante du sous-ensemble de données pour fournir une approximation de bonne qualité et un grand nombre de recherches ont été menées sur les méthodes de sélection des points de référence pour en améliorer l'approximation. Par exemple, ZHANG, TSANG et KWOK, 2008 proposent d'utiliser la méthode des K-moyennes (voir Section 3.3.2) ; une autre possibilité consiste à attribuer une score d'influence aux exemples avant de réaliser l'échantillonnage avec des probabilités dépendantes de ces scores (GITTENS et MAHONEY, 2013). Toutefois, même s'il a été montré empiriquement et théoriquement qu'une méthode d'échantillonnage uniforme fonctionne moins bien dans certains cas que des méthodes plus perfectionnées (GITTENS et MAHONEY, 2013 ; GITTENS, 2011), cette technique reste une option de choix pour son rapport qualité-prix, du fait de son extrême simplicité et de ses bonnes performances en général (KUMAR, MOHRI et TALWALKAR, 2012).

Accélération de la méthode de Nyström : Nyström efficace SI, HSIEH et DHILLON, 2016 ont proposé un algorithme qui tire partie des transformées rapides connues telles que la transformée de Hadamard ou la transformée de Haar (voir Section 3.2.1) pour apprendre la matrice des points de référence $\mathbf{L}$ avec une certaine contrainte de structure afin que son utilisation soit rapide. Il en résulte une approximation de Nyström efficace et plus rapide à utiliser. Après avoir remarqué que le principal coût calculatoire de l'approximation de Nyström provient du calcul de la fonction noyau entre les observations à traiter et les points de référence $\mathbf{L}$, SI, HSIEH et DHILLON, 2016 proposent d'accélérer cette étape. En particulier, ils se concentrent sur une famille de fonctions noyau qui a la forme $\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{f}(\mathbf{x}_i)\mathbf{f}(\mathbf{x}_j)\mathbf{g}(\mathbf{x}_i^T\mathbf{x}_j)$, où $\mathbf{f} : \mathbb{R}^D \mapsto \mathbb{R}$ et $g : \mathbb{R} \mapsto \mathbb{R}$. Cette

famille de fonctions contient certains noyaux largement utilisés tels que le noyau gaussien et le noyau polynomial.

Étant donné les points de référence $\mathbf{L}$ et une observation $\mathbf{x}$, le temps de calcul du noyau entre $\mathbf{x}$ et chaque ligne de $\mathbf{L}$ (nécessaire pour l'approximation de Nyström) est dominé par le calcul du produit $\mathbf{Lx}$. Ils proposent donc d'écrire la matrice $\mathbf{L}$ comme la concaténation de $S = K/D$ produits de matrices tels que :

$$\mathbf{L} = \left[\mathbf{V}_1\mathbf{H}^T, \cdots, \mathbf{V}_S\mathbf{H}^T\right]^T, \tag{1.25}$$

où $\mathbf{H}$ est une matrice $D \times D$ associée à une transformation rapide telle que la matrice de Haar ou Hadamard, et les $\mathbf{V}_i$s sont des matrices diagonales $D \times D$. Les coefficients des matrices $\mathbf{V}_i$ sont à choisir avec une méthode de sélection de points de référence standard (échantillonnage uniforme, par exemple) ou à apprendre au moyen d'un algorithme que les auteurs proposent, similaire à l'algorithme des K-moyennes.

1.2.4 Conclusion sur les méthodes à noyaux

Les méthodes à noyaux sont puissantes et justifiées théoriquement ce qui en font des solutions de choix pour de nombreux problèmes d'apprentissage. Les noyaux permettent d'incorporer des non-linéarités dans des modèles d'apprentissage linéaire grâce à l'astuce du noyau. Ils permettent aussi le traitement de données au format particulier, telles que les graphes, pour peu qu'une mesure de similarité adéquate existe.

Un inconvénient des méthodes du noyau est la nécessité de choisir la fonction noyau adaptée au données à traiter. Ce choix est crucial pour la performance du modèle et, même si quelques méthodes existent pour apprendre le noyau, il est souvent nécessaire de faire appel à la connaissance d'un expert pour obtenir les meilleures performances possibles. Un autre problème de ces méthodes est qu'elles ont une complexité calculatoire élevée en fonction de la taille des jeux de données d'apprentissage. C'est pourquoi des méthodes d'approximation de noyaux telles que la méthode de Nyström ont vu le jour pour en réduire la complexité calculatoire.

1.3 Réseaux de neurones

Les réseaux de neurones artificiels —ou réseau neuronaux— sont des fonctions mathématiques composées d'une interconnexion de fonctions plus simples appelées «neurones artificiels». Ces neurones artificiels sont inspirés du fonctionnement des neurones biologiques capables d'émettre un signal de sortie en fonction d'une somme de signaux d'entrée.

Dans cette section nous commençons par définir les concepts de neurones artificiels et couches neuronales puis nous décrirons deux types de couches couramment utilisées en apprentissage automatique qui sont les couches entièrement connectées et les couches convolutives. Ensuite, nous verrons grâce à des exemples d'architectures neuronales comment ces couches elles même peuvent être combinées entre elles. Nous ferons ainsi apparaitre le problème de surparamétrisation des réseaux de neurones ce qui nous amènera à lister quelques approches de compression proposées dans la littérature.

1.3.1 Neurones artificiels et couches neuronales

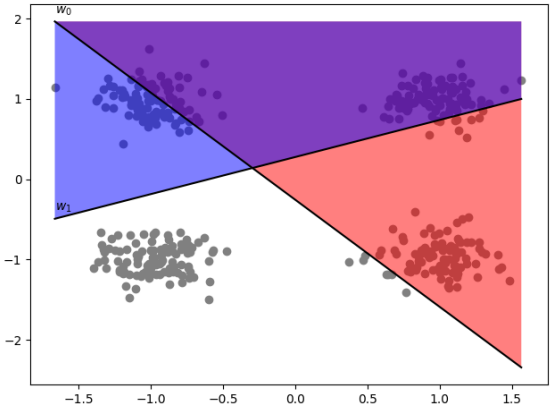


FIGURE 1.3 : Représentation en deux dimensions de deux neurones d'une couche cachée. Chaque neurone sépare l'espace en deux. Combiner les neurones dans une couche permet de quadriller l'espace d'entrée.

Soit une observation $\mathbf{x} \in \mathbb{R}^D$, un neurone est une fonction mathématique $\phi_{\mathbf{w}} : \mathbb{R}^D \mapsto \mathbb{R}$ paramétrée par un vecteur de poids $\mathbf{w}$ et qui calcule une combinaison linéaire des éléments de $\mathbf{x}$ suivie par une fonction non-linéaire $\mathbf{a} : \mathbb{R} \mapsto \mathbb{R}$:

$$\phi_{\mathbf{w}}(\mathbf{x}) = \mathbf{a}\left(\sum_{i=1}^{D}(x_i w_i) + w_0\right). \tag{1.26}$$

Pour simplifier, nous négligeons le biais w_0 dans la suite. Par analogie avec le neurone biologique qui est capable de relayer un message nerveux si la somme des signaux excitateurs et inhibiteurs entrants dépasse un certain seuil d'excitabilité (WIDMAIER, RAFF, STRANG et al., 2008), on peut voir le neurone formel de l'Équation (1.26) comme une fonction seuil qui s'active si la combinaison linéaire de ses entrées est suffisamment grande. Une interprétation géométrique du neurone artificiel consiste à le voir comme une fonction indicatrice de la position d'un point par rapport à l'hyper-plan défini par $\mathbf{w}$ (voir Figure 1.3). On note toutefois que, contrairement à une fonction seuil ou une fonction indicatrice,

la fonction d'activation doit être dérivable pour permettre à l'apprentissage de se faire par descente de gradient (voir note sur les fonctions d'activation Section 1.3.3).

Des neurones peuvent travailler en parallèle dans une couche $\phi : \mathbb{R}^D \mapsto \mathbb{R}^{D'}$ pour donner une représentation non-linéaire en D' dimensions d'une donnée d'entrée initialement en dimension D :

$$\phi(\mathbf{x}) = \begin{bmatrix} \phi_{\mathbf{w}_1}(\mathbf{x}) \\ \vdots \\ \phi_{\mathbf{w}_{D'}}(\mathbf{x}) \end{bmatrix} = \mathbf{a}\left(\mathbf{W}^T\mathbf{x}\right) \tag{1.27}$$

où les $\mathbf{w}_i$ peuvent être empilés pour former une matrice $\mathbf{W} \in \mathbb{R}^{D \times D'}$. Nous faisons un abus de notation assez courant ici, avec la fonction $\mathbf{a}$ qui est en fait appliquée composante par composante. En quelques sortes, une couche neuronale «quadrille» l'espace d'entrée des données où chaque ligne de ce quadrillage est définie par un neurone (voir Figure 1.3). En jouant sur les propriétés de $\mathbf{W}$, nous pouvons définir différents types de couches. Nous en montrons maintenant deux exemples.

Couches denses Dans le cas où la matrice $\mathbf{W}$ est dense, c'est-à-dire qu'elle ne contient pas ou peu de zéros, nous dirons simplement que la couche ϕ est «dense» ou bien «entièrement connectée». Ce type de couche calcule une représentation en fonction des combinaisons linéaires globales des données d'entrée ou de la couche précédente.

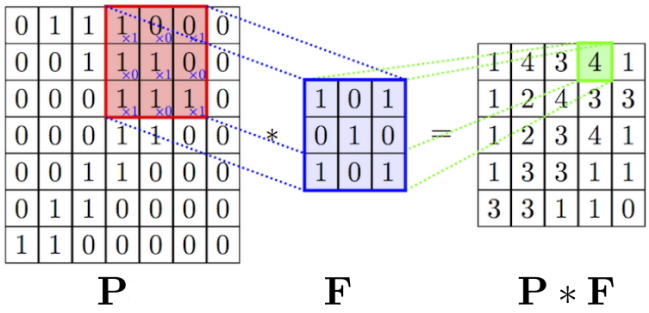


FIGURE 1.4 : Représentation de la convolution en deux dimensions. Le filtre $\mathbf{F}$ est appliqué à toutes les positions de l'image $\mathbf{P}$ en calculant le produit scalaire entre le filtre et les valeurs de l'image à la position où il est appliqué. Dans cet exemple, il n'y a qu'un canal en entrée et un seul filtre. Source : MOUTARDE, 2020.

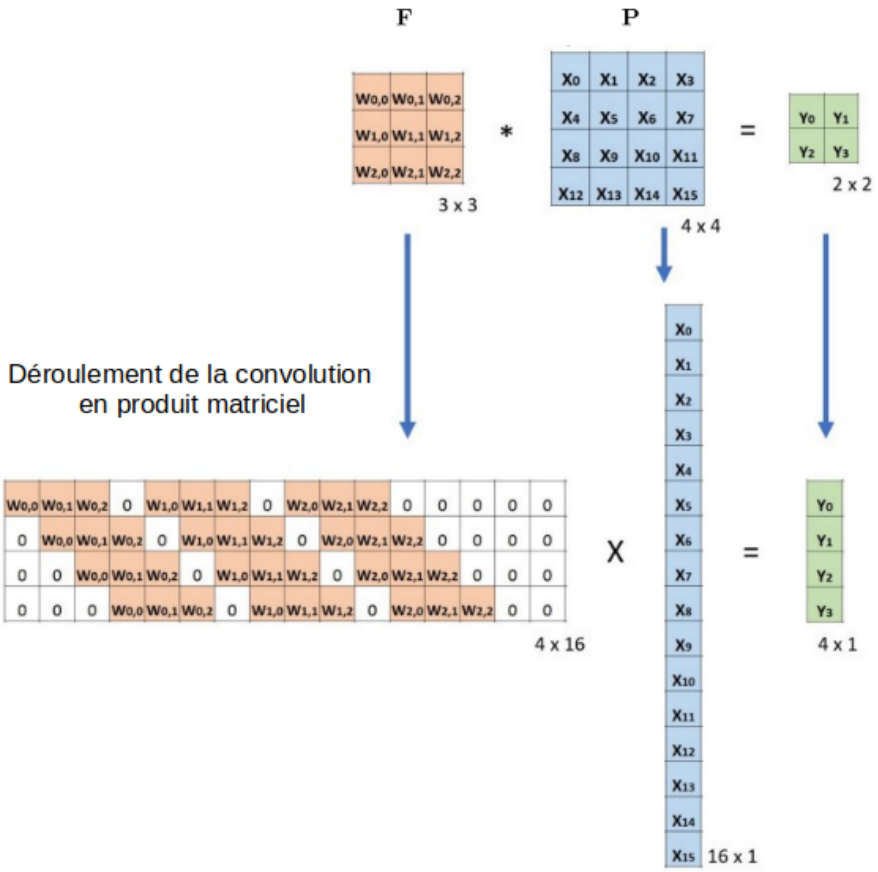


FIGURE 1.5 : Représentation de la convolution sous forme d'un produit matriciel creux. Dans cet exemple, il n'y a qu'un canal en entrée et un seul filtre. Source : BAI, 2019

Couches convolutives Un autre type de couche sont les couches dites «convolutives» ou «couches de convolution». Ces couches, comme leur nom l'indique, réalisent des opérations de convolution, c'est-à-dire qu'elles appliquent un filtre sur les données d'entrée de façon à y détecter des motifs locaux aux alentours de toutes les positions (voir Figure 1.4). Dans le cas en deux dimensions, où une image d'entrée $\mathbf{P} \in \mathbb{R}^{H \times W \times C}$ est convoluée par un noyau de convolution $\mathbf{F} \in \mathbb{R}^{S \times S \times C \times F}$, la convolution donne :

$$\phi_{conv}(\mathbf{P})_{i,j,f} = \sum_{w=1}^{S} \sum_{h=1}^{S} \sum_{c=1}^{C} k_{w,h,c,f} p_{i+w-1,j+h-1,c}, \tag{1.28}$$

où $\phi_{conv}(\mathbf{P})_{i,j,f}$ est la valeur retournée par le $f^{ème}$ filtre en coordonnées (i, j), $p_{i,j,c}$ est la valeur en coordonnée (i, j) du $c^{ème}$ canal de l'image d'entrée $\mathbf{P}$ et les $k_{w,h,c,f}$ sont les poids de convolution du noyau $\mathbf{F}$. Ce type de couche est particu-

lièrement utile pour le traitement de données structurées [1] telles que les images. Une propriété connue de l'opération de convolution est que celle-ci peut se réécrire sous la forme d'un produit entre une matrice de Toeplitz (matrice dont les coefficients sur une diagonale descendant de gauche à droite sont les mêmes) et le vecteur à convoluer (voir Figure 1.5). De même, on peut construire une couche de convolution avec plusieurs filtres qui aurait la forme de l'Équation (1.27) dont la matrice de poids $\mathbf{W} \in \mathbb{R}^{D \times FD}$ ($D := HWC$) serait construite à partir des valeurs du noyau de convolution $\mathbf{F}$ et essentiellement constituée de zéros (SALEHI, 2020). Toutefois, cette représentation de la convolution n'est pas très pratique à manipuler et nous préférons plutôt une autre expression qui fait intervenir une opération de remodelage (en anglais, *reshape*) des données d'entrée. Commençons par définir deux opérateur de remodelage $\mathbf{r}_S$ et $\mathbf{t}$:

- L'opérateur $\mathbf{r}_S$ est paramétré par une taille de patch S et prend comme entrée un tenseur de dimensions $(H \times W \times C)$:

$$\mathbf{r}_S : \mathbb{R}^{H \times W \times C} \mapsto \mathbb{R}^{HW \times CS^2}. \tag{1.29}$$

  Cette opération de remodelage crée la matrice de tous les patchs vectorisés de hauteur et de largeur S. Nous supposons sans perte de généralité que les patchs sont échantillonnés à toutes les positions (en anglais, nous dirions que le *stride* est égal à 1) et que le tenseur d'entrée est augmenté de $\lfloor \frac{S}{2} \rfloor$ zéros verticalement et horizontalement, ceci permettra de faire en sorte que le tenseur obtenu en sortie de convolution soit de même hauteur H et de même largeur W que le tenseur reçu en entrée ;

- L'opérateur $\mathbf{t}$ prend en entrée un tenseur de dimensions $(HW \times F)$ et reconstruit un tenseur de dimension $(H \times W \times F)$:

$$\mathbf{t} : \mathbb{R}^{HW \times F} \mapsto \mathbb{R}^{H \times W \times F}. \tag{1.30}$$

La Figure 1.6 donne une représentation schématique de ces opérations de remodelage. Maintenant que $\mathbf{r}_S$ et $\mathbf{t}$ sont correctement définis nous pouvons écrire l'opération de convolution en deux dimensions ϕ_{conv} telle que, pour toute image d'entrée $\mathbf{P} \in \mathbb{R}^{H \times W \times C}$:

$$\phi_{conv}(\mathbf{P}) = \mathbf{a}\left(\mathbf{t}\left(\mathbf{r}_S(\mathbf{P})\ \mathbf{W}\right)\right) \tag{1.31}$$

[1] Nous utilisons le terme de «données structurées» pour faire référence aux données dont l'organisation des éléments, la structure, est porteuse d'information (pour une image, la position des pixels les uns par rapport aux autres), en opposition aux données dont les dimensions peuvent être permutées (les données qui peuvent être organisées dans un tableau).

où $\mathbf{W} \in \mathbb{R}^{CS^2 \times F}$ est la matrice des poids (matrices de filtres) construite en remodelant le noyau de convolution $\mathbf{F}$ pour faire correspondre les dimensions . La couche de convolution utilise la matrice de filtres $\mathbf{W}$ afin de calculer les transformations linéaires locales des patchs du tenseur d'entrée définies par l'opérateur de remodelage $\mathbf{r}_S$ (Équation (1.29)). D'un certaine façon, cette définition de la couche convolutive peut correspondre à la définition de couche donnée en Équation (1.27) si l'on considère que chaque neurone n'utilise que des patchs du tenseur d'entrée.

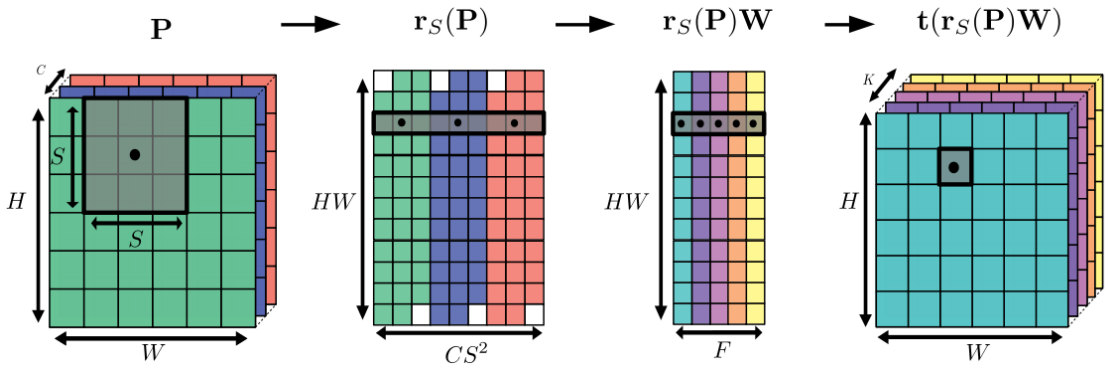


FIGURE 1.6 : Représentation schématique des opérations de remodelage nécessaires pour implémenter la convolution définie en Équation (1.31). La zone grise représente une zone d'application d'un filtre autour d'une coordonnée représentée par le point.

Bien entendu, il existe une variété d'autres couches pour la construction de réseaux de neurones comme par exemple la couche de *Max-Pooling* qui permet de sous-échantillonner les pixels d'une image au fil de sa représentation. Nous ne pouvons pas citer toutes les couches ici et renvoyons plutôt le lecteur vers la documentation de Keras TEAM, 2020b pour voir une liste plus complète des couches couramment utilisées. Notons toutefois qu'il serait vain d'essayer de lister tous les types de couches existants car de nouveaux sont proposés dans des papiers très régulièrement. Les couches des réseaux de neurones doivent être vues comme des modules indépendants au rôle bien défini qui peuvent être combinés entre eux et empilés les uns sur les autres pour obtenir une architecture neuronale spécialisée sur un type de tâche. Lorsqu'un réseau de neurones contient plusieurs couches, on dit que les couches intermédiaires avant la couche finale sont des couches cachées et on qualifie le réseau de neurone de «profond».

Dans la Section suivante, nous voyons quelques exemples d'architectures largement utilisés pour la reconnaissance d'image par apprentissage profond.

1.3.2 Exemples d'architectures de réseau de neurones convolutifs

Les réseaux de neurones dits «convolutifs» font intervenir des couches convolutives. Même si, théoriquement, un réseau de neurone à une seule couche avec un nombre suffisamment grand de neurones est capable d'être un estimateur universel, il a été montré empiriquement que les réseaux plus profonds atteignent de meilleures performance en pratique(CYBENKO, 1989 ; HORNIK, 1991 ; LESHNO, LIN, PINKUS et al., 1993).

De ce fait, un objectif qui a été poursuivi au fil des ans a été de construire des architectures toujours plus profondes, avec plus de couches, pour obtenir des performances sans cesse meilleures. Nous allons voir trois exemples d'architecture et leur spécificités dans l'ordre chronologique de leurs apparitions.

Lenet. L'architecture Lenet a été proposée par LECUN, BOTTOU, BENGIO et al., 1998 pour un problème de reconnaissance de caractères et est habituellement utilisée comme architecture de référence pour la reconnaissance de chiffres manuscrits. Cette architecture, visible en Figure 1.7, n'est constituée que de 5 couches et est souvent considérée comme une architecture «jouet».

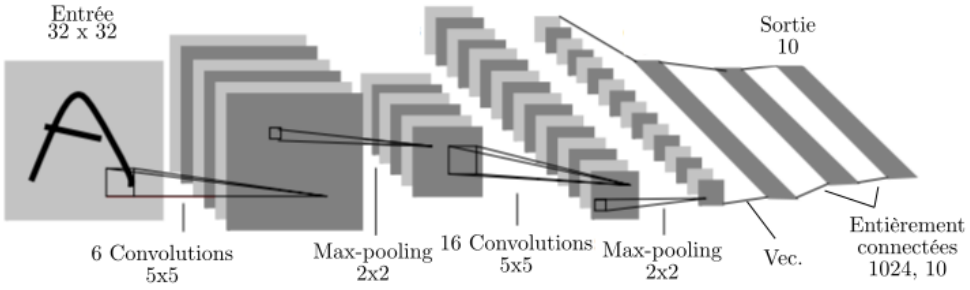


FIGURE 1.7 : Représentation schématique de l'architecture Lenet-5. Source : LECUN, BOTTOU, BENGIO et al., 1998

VGG. Les architectures VGG de SIMONYAN et ZISSERMAN, 2015 sont pionnières parmi les architectures à réellement tirer parti de la profondeur dans les réseaux de neurones grâce à l'utilisation de petits filtres de convolution de taille $S = 3$. La Figure 1.8 représente la version de l'architecture proposée avec 19 couches ; on note toutefois que des implémentations plus récentes (LI, 2020) intègrent des couches de normalisations par *batch* (*Batchnorm*) pour en faciliter l'apprentissage (voir Section 1.3.3).

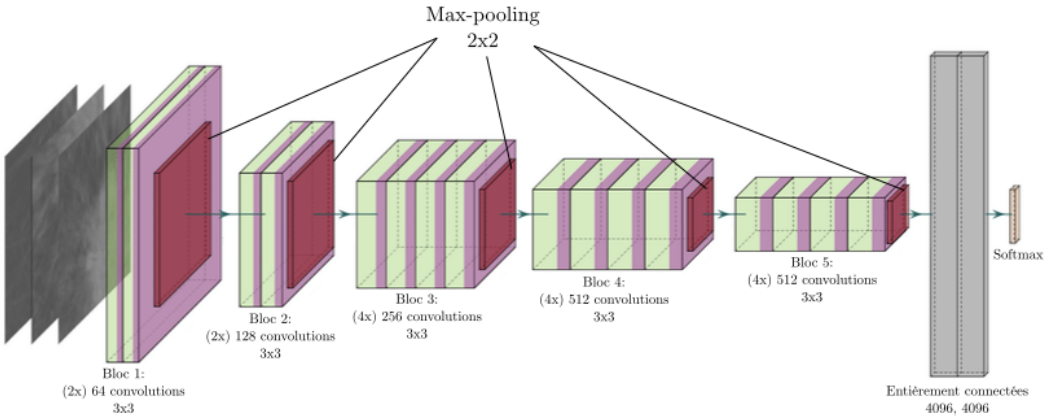


FIGURE 1.8 : Représentation schématique de l'architecture VGG19. Source : HASAN et ALEEF, 2019

Resnet. Les architectures Resnet de HE, ZHANG, REN et al., 2016 sont peut-être les architectures de reconnaissance d'images les plus populaires au moment de la rédaction de cette thèse. Un exemple de diz-huit couches est visible en Figure 1.9 mais d'autres déclinaisons, avec plus de couches, sont habituellement utilisées. Elle intègre des couches dites «résiduelles» qui sont composées d'une somme entre la sortie d'une couche et la sortie d'une couche de niveau inférieur mais de même dimension (voir Figure 1.10). Ceci permet d'une part d'encourager les couches à encoder des caractéristiques ré-utilisables à plusieurs niveaux de profondeur du réseau et d'autre part de faciliter le processus d'optimisation des couches en évitant le problème de disparition du gradient (HOCHREITER, BENGIO, FRASCONI et al., 2001). Des implémentations de ces réseaux peuvent aussi intégrer des couches «d'étranglement» (en anglais, *bottleneck*) qui permettent de limiter la complexité calculatoire des couches de convolution en passant par des représentations en plus petite dimension (SZEGEDY, LIU, JIA et al., 2014).

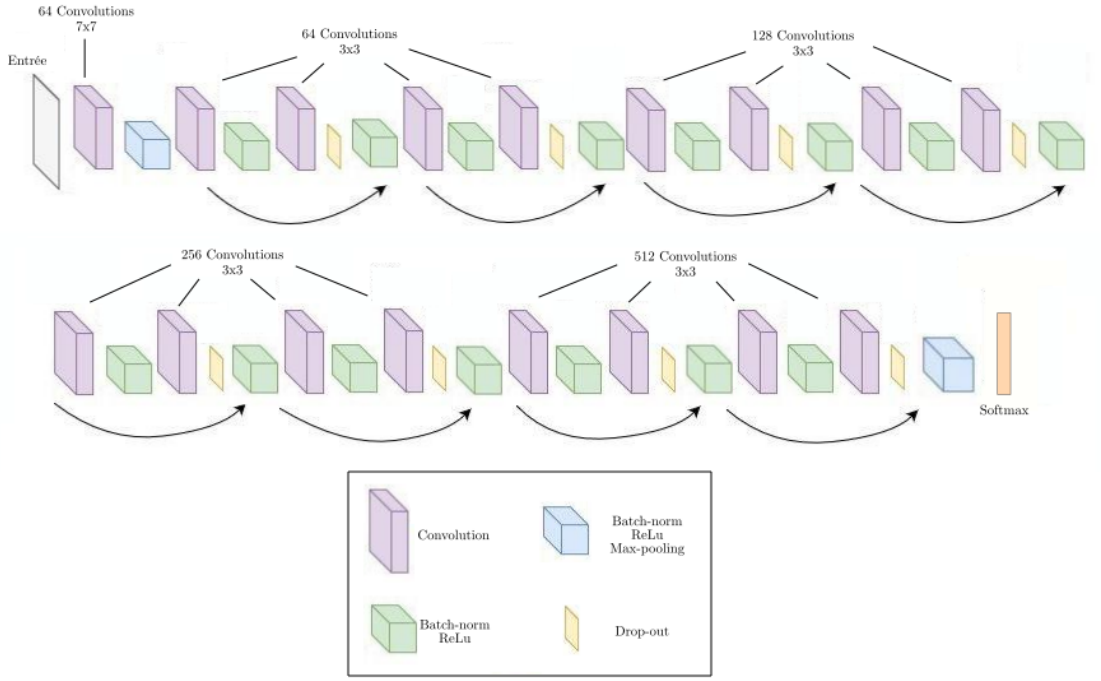


FIGURE 1.9 : Représentation schématique de l'architecture Resnet18. Source : AL RABBANI ALIF, AHMED et HASAN, 2017

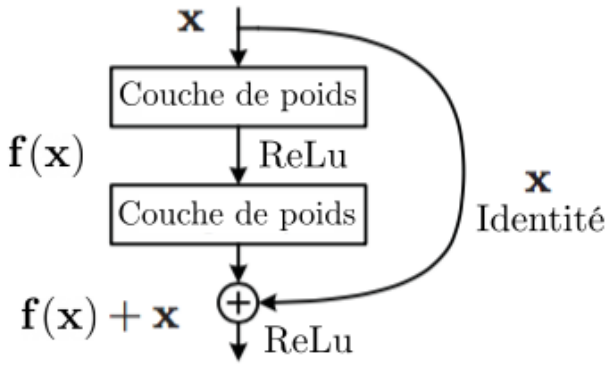


FIGURE 1.10 : Représentation schématique d'un bloc résiduel des réseaux Resnet. Source : HE, ZHANG, REN et al., 2016

1.3.3 Notes sur l'apprentissage de réseaux de neurones

La définition des couches et neurones dans un réseau neuronal a fait apparaître la notion de poids (Équation (1.31), Équation (1.27)). Ce sont ces poids qui définissent la capacité d'une architecture donnée à effectivement remplir une tâche

et donc une question légitime est de savoir comment ces poids sont-ils déterminés ? comment sont-ils appris ?.

Dans cette section, nous faisons un tour rapide du fonctionnement de l'apprentissage des réseaux de neurones et nous citerons quelques avancées majeures sur cette question.

Apprentissage par descente de gradient. Les poids du réseau de neurone sont appris suivant la technique générale de la descente du gradient de l'erreur. Grâce aux règles de dérivation des fonctions composées (*chainrule*), le gradient de l'erreur peut être rétro-propagé (*back-propagated*) (RUMELHART, HINTON et WILLIAMS, 1986) depuis les couches supérieures vers les couches inférieures du réseau. Spécifiquement, nous utilisons habituellement des algorithmes de descente de gradient stochastique tels que l'algorithme Adam (KINGMA et BA, 2014), qui est connu pour apporter de bonnes performances en général. D'autres algorithmes d'optimisation par descente de gradient existent bien sur ; une revue non exhaustive de ceux-ci est proposées par RUDER, 2017. La fonction d'erreur utilisée va dépendre de la tâche à réaliser. Typiquement, pour une tâche de régression, c'est plutôt la MSE qui est utilisée et pour une tâche de classification multi-classe, ce sera plutôt un critère d'entropie croisée $\mathbf{h}(\mathbf{y}, \hat{\mathbf{y}})$ qui compare de distribution $\mathbf{y}$ et $\hat{\mathbf{y}}$:

$$\mathbf{h}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{1.32}$$

où $\mathbf{y}$ est en fait au vecteur binaire qui encode la vraie classe d'un exemple et $\hat{\mathbf{y}}$ est le vecteur des probabilités de classes renvoyé par le réseau de neurone pour un exemple donné ($\sum_i \hat{y}_i = 1$).

Initialisation. La fonction encodée par un réseau de neurones n'étant pas convexe, la qualité de l'apprentissage est grandement conditionnée par une bonne initialisation du réseau de neurones. Ainsi, des stratégies d'initialisation adaptées telles que Xavier (GLOROT et BENGIO, 2010) ou He (HE, ZHANG, REN et al., 2016) ont été proposées. La stratégie proposée par GLOROT et BENGIO, 2010 vise à faire en sorte que la variance des activations après la partie linéaire d'une couche soit égale à 1 en initialisant les poids des couches aléatoirement suivant une loi normale centrée et de variance :

$$Var[\mathbf{W}] = \frac{2}{fan_{in} + fan_{out}} \tag{1.33}$$

où fan_{in} et fan_{out} correspondent respectivement au nombre de dimension en entrée et en sortie de la couche. La stratégie d'initialisation proposée par HE, ZHANG, REN et al., 2016 prend en compte la fonction d'activation qui suit la partie linéaire de toutes les couches. En particulier, elle s'intéresse à la fonction

d'activation $ReLU$ (voir après). En remarquant que cette fonction a pour effet de diviser par deux la variance d'éléments centrés, les auteurs proposent donc de multiplier par deux la valeur de la variance proposée par HE, ZHANG, REN et al., 2016 ce qui donne :

$$Var[\mathbf{W}] = \frac{2}{fan_{in}} \tag{1.34}$$

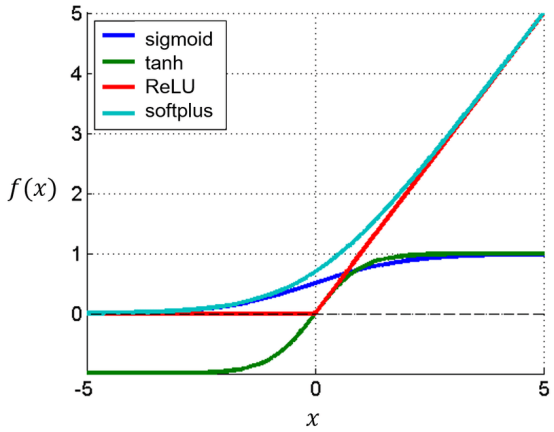


FIGURE 1.11 : Représentation graphiques de quelques fonctions d'activation connues. Source : MUSIOL, 2016.

Fonction d'activation. La fonction d'activation couramment utilisée dans les réseaux de neurones est la fonction $ReLU$ (GLOROT, BORDES et BENGIO, 2011) qui accélère l'apprentissage des réseaux neuronaux et souvent aussi leur performance. Cette amélioration vient du caractère «non-saturant» de la fonction $ReLU$ comparé à certaines de ses homologues telles que la fonction sigmoïde ou tangente hyperbolique (voir Figure 1.11). On dit que la fonction $ReLU$ est «non-saturante» car elle n'a pas de valeur maximale et est donc capable d'encoder plusieurs niveaux d'activation (activations plus ou moins fortes). En général, les fonctions d'activation doivent être dérivables afin de permettre à l'apprentissage de se faire par descente de gradient. On note toutefois que la fonction d'activation $ReLU$ n'est pas dérivable en zéro. Ceci ne pose pas de problème en pratique car il est extrêmement improbable qu'une valeur soit exactement égale à zéro au fil de l'apprentissage.

Normalisation par Batch. Avec l'augmentation de la profondeur du réseau, un phénomène de disparition (*vanishing*) ou explosion (*exploding*)du gradient est apparu du fait de l'effet de nombreuses multiplications successives entre poids et activations. Pour résoudre ce problème, une couche de normalisation par *batch* (*batchnorm*) a été proposée par IOFFE et SZEGEDY, 2015. Cette couche

a pour effet de normaliser les activations en cours de traitement de façon a éviter que certaines activations soient relativement trop grandes ou trop basses par rapport aux autres. De nombreuses alternatives à la couche *batchnorm* ont été proposées pour répondre à certains de ces problèmes. Sans les citer tous, nous pouvons pointer par exemple l'article de BA, KIROS et HINTON, 2016 qui adaptent la couche aux réseaux dits "récurrents" ou bien celui de WU et HE, 2018 qui s'intéressent à l'entraînement de réseaux avec des tailles de *batch* réduites.

Régularisation, *Dropout* et augmentation des données. Pour combattre le problème de sur-apprentissage des données d'entrainement, plusieurs stratégies ont été mises en place. D'abord, il est très courant d'ajouter un terme de régularisation à l'erreur. Habituellement, nous utiliserons plutôt une régularisation $L2$ au poids du réseau afin de les garder les plus petit possible. Ensuite, une couche appelée *dropout* a été proposée par HINTON, SRIVASTAVA, KRIZHEVSKY et al., 2012 et consiste a supprimer des activations aléatoirement à chaque itération du processus d'optimisation. Encore une fois, nous ne les citerons pas toutes mais d'autres approches similaire ont vu le jour, telles que notamment la méthode `Dropconnect` proposée par WAN, ZEILER, ZHANG et al., 2013 où les poids des couches sont mis à zéros aléatoirement plutôt que les activations Cette couche a pour effet d'éviter que certains neurones ne se spécialisent trop et encourage une certaine redondance entre les caractéristiques apprises par différents neurones. Enfin, il est courant d'utiliser l'augmentation des données d'apprentissage afin de rendre le réseau plus robustes à de faibles variations. Des stratégies courantes d'augmentation des données sont : la translation, la rotation ou la réflexion d'images.

1.3.4 Compression de réseaux neuronaux pré-entrainés

Architecture	**Performance**	**# Paramètres**
VGG16	71.3%	138,357,544
VGG19	71.3%	143,667,240
Resnet50	74.9%	25,636,712
Resnet101	76.4%	44,707,176
Resnet152 Type	76.6%	60,419,944

TABLE 1.1 : Exemples d'architectures neuronales, leur taille et leur performance en classification sur le jeu de données Imagenet. Source : TEAM, 2020a

Comme le montrent les exemples du Tableau 1.1, la plupart des modèles de réseaux de neurones à la pointe de la technologie contiennent tant de paramètres qu'ils deviennent coûteux à stocker et à faire fonctionner, et particulièrement peu

pratiques à déployer sur des appareils aux ressources limitées (mémoire, capacités de calcul) ou sur des appareils qui ne peuvent pas intégrer de GPU. Ceci s'explique par la volonté de construire des modèles avec un nombre de couche croissant. En effet, les couches denses contiennent une matrice de poids de dimensions $(D \times D')$ qui a une complexité en espace de $\mathcal{O}(DD')$. Les couches de convolution, elles, ont une complexité en espace de $\mathcal{O}(L^2CF)$ car elles contiennent un tenseur de dimensions $(L \times L \times C \times F)$ lorsqu'il y a C canaux d'entrée et F filtres à calculer. Lorsque les valeurs de C et F sont basses, comme dans les architectures les plus anciennes (voir Section 1.3.2), ce sont plutôt les couches denses qui contiennent l'essentiel des paramètres du modèle. Dans les architectures les plus modernes, cependant, les couches denses tendent à disparaître mais les valeurs C et F sont plus grandes et alors les couches de convolution contiennent aussi une grande quantité de paramètres.

Contrairement à l'idée communément admise en apprentissage automatique selon laquelle les modèles les plus complexes sont facilement sujets au sur-apprentissage, la sur-paramétrisation des réseaux neuronaux semble plutôt avoir un effet bénéfique sur leur performance lorsque beaucoup de données sont disponibles. Des travaux récents tels que ceux de NEYSHABUR, LI, BHOJANAPALLI et al., 2018 ont tenté d'élucider ce paradoxe en proposant une borne de généralisation des réseaux neuronaux qui dépend de la différence entre les poids du réseau à l'initialisation et après apprentissage. Intuitivement, leur borne suggère que les gros réseaux neuronaux fonctionnent bien car ils ont plus de chances de contenir, dès l'initialisation, une combinaison de paramètres adaptée à la tâche d'apprentissage. Cette idée est corroborée par FRANKLE et CARBIN, 2019 qui proposent une stratégie pour découvrir cette combinaison de paramètres "minimale", parmi tous les paramètres du réseau.

Toutefois, le paramétrage excessif des réseaux neuronaux est au cœur d'un axe de recherche populaire appelé «compression des réseaux neuronaux» qui vise à construire des modèles avec moins de paramètres tout en préservant au mieux leur qualité de prédiction. Ainsi, une stratégie courante consiste à d'abord entraîner un énorme réseau neuronal pour en tirer de bonnes performances, puis de le compresser en construisant un réseau plus petit mais similaire.

Cette section présente brièvement les principes et les limites des techniques courantes de compression de réseaux neuronaux pré-entraînés.

1.3.4.1 Décompositions en rang faible

Les méthodes de factorisation de rang faible reposent sur l'idée que les caractéristiques apprises à chaque couche d'un réseau peuvent être corrélées et donc que les poids des couches peuvent être factorisés.

La plupart des méthodes de décomposition de couches ont été étudiées pour comprimer soit des couches convolutives, soit des couches entièrement connectées. Par exemple, SAINATH, KINGSBURY, SINDHWANI et al., 2013 se sont concen-

trés sur la Décomposition en Valeurs Singulières (SVD) des couches entièrement connectées. Étant donné la matrice de poids $\mathbf{W} \in \mathbb{R}^{D \times D'}$ d'une couche dense, cette technique utilise la SVD pour en obtenir une décomposition de rang K :

$$\mathbf{W} \approx \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{1.35}$$

avec $\mathbf{U} \in \mathbb{R}^{D \times K}$, $\mathbf{V} \in \mathbb{R}^{D' \times K}$ et $\mathbf{S} \in \mathbb{R}^{K \times K}$ une matrice diagonale contenant les valeurs singulières. Pour compresser la couche, il faut la diviser en deux couches plus étroites contenant respectivement les matrices de poids $\mathbf{U}' = \mathbf{U}\mathbf{S}^{-1/2}$ et $\mathbf{V}' = \mathbf{S}^{-1/2}\mathbf{V}^T$. Notons qu'aucune non-linéarité ne sépare les deux couches nouvellement formées. En choisissant un K bien inférieur à D et D', cette technique permet de réduire la complexité en espace de la couche entièrement connectée de $\mathcal{O}(DD')$ à $\mathcal{O}(DK + D'K)$.

Similairement, LEBEDEV, GANIN, RAKHUBA et al., 2015 et KIM, PARK, YOO et al., 2016 ont respectivement étudié les décompositions CP (CANDECOMP / PARAFAC : *Canonical Decomposition / Parallel Factors Decomposition*) et Tucker pour la compression de noyaux de convolution, qui sont en fait des tenseurs d'ordre quatre. Ces décompositions peuvent être chacune vue comme une généralisation de la SVD aux tenseurs d'ordres élevés. Nous donnons maintenant un bref aperçu du fonctionnement de la décomposition de Tucker. Pour se faire, nous devons d'abord introduire quelques nouveaux concepts d'algèbre linéaire : le m-rang et le produit m-mode. Soit $\mathcal{T} \in \mathbb{R}^{D_1 \times \ldots \times D_M}$, un tenseur d'ordre M. La m-matricisation de $\mathcal{T}$ est $\mathbf{T}(m) \in \mathbb{R}^{D_m \times (D_1 \ldots D_{m-1} D_{m+1} \ldots D_M)}$; elle correspond au déroulement du tenseur $\mathcal{T}$ suivant son $m^{ième}$ mode. Ainsi, le m-rang de $\mathcal{T}$ correspond au rang de la m-matricisation de $\mathcal{T}$. Le produit m-mode entre le tenseur $\mathcal{T} \in \mathbb{R}^{D_1 \times \cdots \times D_M}$ et la matrice $\mathbf{U} \in \mathbb{R}^{K \times D_m}$ donne un tenseur de dimensions $(D_1, \cdots, D_{m-1}, K, D_{m+1}, \cdots, D_M)$. Ce produit s'écrit $\mathcal{T} \times_m \mathbf{U}$ et chacun de ses éléments sont définis tels que :

$$(\mathcal{T} \times_m \boldsymbol{U})_{d_1, \cdots, d_{m-1}, k, d_{m+1} \cdots, d_M} = \sum_{i_m=1}^{D_m} \mathcal{T}_{d_1, \cdots, d_{m-1}, i_m, d_{m+1}, \cdots, d_M} u_{k, i_m}. \tag{1.36}$$

Le rang de Tucker d'un tenseur d'ordre M est écrit $(R_1, \cdots, R_M)$ où pour tout $m \in 1, \ldots, M$, R_m est le m-rang du tenseur. Étant donné son rang de Tucker, un tenseur $\mathcal{T}$ peut-être décomposé tel que :

$$\mathcal{T} = \mathcal{C} \times_1 \mathbf{U}_1 \cdots \times_M \mathbf{U}_M \tag{1.37}$$

où $\mathcal{C} \in \mathbb{R}^{R_1 \times \cdots \times R_M}$ est appelé le noyau (*core*) de la décomposition et les matrices $\mathbf{U}_m \in \mathbb{R}^{R_m \times D_m}$ sont des matrices orthogonales. En prenant des valeurs de R_m suffisamment petites, cette méthode permet de réduire la complexité en stockage d'une couche de convolution de $\mathcal{O}(\prod_{m=1}^{M} D_m)$ à $\mathcal{O}(\prod_{m=1}^{M} R_m + \sum_{m=1}^{M} R_m D_m)$.

En outre, la décomposition *Tensor-train* (TT) a été explorée pour compresser

à la fois les couches denses et les couches de convolution. Dans un format TT, tous les éléments d'un tenseur d'ordre M sont exprimés par un produit de M matrices dont les dimensions sont déterminées par les rangs-TT $(R_0, R_1, \ldots, R_M)$ où $R_0 = R_M = 1$:

$$\mathcal{T}_{d_1,\ldots,d_M} = \boldsymbol{G}_1[d_1] \ldots \boldsymbol{G}_M[d_M]. \tag{1.38}$$

où pour tout $m \in 1 \ldots M$, $\boldsymbol{G}_m[d_m] \in \mathbb{R}^{R_{m-1} \times R_m}$. Techniquement, pour chaque dimension m du tenseur initial, les matrices $\boldsymbol{G}_m$ correspondantes à la dimension m du tenseur initial peuvent être empilées pour former un tenseur d'ordre 3 appelé "noyau" (*core*) et dénoté $\mathcal{G}_m$. En utilisant le format TT, on peut réduire la complexité en stockage d'un tenseur de $\mathcal{O}(\prod_{m=1}^{M} D_m)$ à $\mathcal{O}(\sum_{m=1}^{M} R_{m-1} R_m D_m)$.

NOVIKOV, PODOPRIKHIN, OSOKIN et al., 2015 s'appuient sur cette représentation des tenseurs pour factoriser les couches entièrement connectées. Pour se faire, ils procèdent en deux étapes : d'abord ils remodèlent les matrices de poids $\mathbf{W} \in \mathbb{R}^{D \times D'}$ en un tenseur de dimensions $(D_1 D'_1, \ldots, D_M D'_M)$ où $\prod_{m=1}^{M} D_m = D$ et $\prod_{m=1}^{M} D'_m = D'$; ensuite ils y appliquent le format TT. Ici, M est un hyper-paramètre du modèle habituellement égal à trois ou quatre. On note que pour appliquer la couche en format TT à un vecteur d'entrée, ce dernier doit aussi être remodelé en un tenseur d'ordre M. En choisissant des valeurs de rang-TT suffisamment petites, cette technique leur permet finalement d'obtenir un taux de compression colossal sur de très grande architectures neuronales. Les auteurs mettent l'accent sur l'idée que leur technique pourrait permettre de rendre possible l'encodage de réseaux de neurones très larges (beaucoup de neurones par couche) ; toutefois, la démonstration de l'intérêt pratique de tels modèles reste à faire.

GARIPOV, PODOPRIKHIN, NOVIKOV et al., 2016 ont adapté cette idée aux couches convolutives. Cependant, dans ce dernier cas de figure, la méthode TT induit également une inflation de la taille du *batch* en mémoire à tel point que l'implémentation qu'ils proposent (GARIPOV, 2020) devient inutilisable pour les larges réseaux convolutifs récents. Ce défaut est aussi visible dans l'équation du calcul de convolution que GARIPOV, PODOPRIKHIN, NOVIKOV et al., 2016 fournissent à la fin de la Section 4 de leur papier où on peut voir que toutes les valeurs du tenseur d'entrée de la convolution doivent d'abord être multipliées par une matrice de dimension $1 \times R_1$, provoquant ainsi une inflation de R_1 fois la taille du batch en mémoire. Nous ne détaillons pas plus ces techniques ici car bien qu'elles aient été utilisées en tant que référence dans nos expériences, elles sont d'un intérêt marginal pour la compréhension de cette thèse.

1.3.4.2 Utilisation de Transformées rapides

Comme avec les méthodes à noyaux (voir Section 1.2.3), les transformées rapides (voir Section 3) peuvent être utilisées dans les réseaux de neurones afin de réduire leur complexité. Par exemple, HIGHLANDER et RODRIGUEZ, 2016 pro-

posent d'utiliser la Transformée de Fourier Rapide (FFT) pour projeter les images d'entrée dans le domaine des fréquences afin d'en calculer une convolution rapide. YANG, MOCZULSKI, DENIL et al., 2015 proposent une architecture appelée `Deep Fried Convnets` dans laquelle ils remplacent directement les couches denses d'un réseau de neurone par l'approximation de noyaux `Fastfood` (voir Section 1.2.3.1) qui se base sur la transformée rapide de Hadamard afin de réduire le nombre de paramètres à stocker dans le réseau. Comme la matrice de Hadamard n'a pas besoin d'être stockée, cette technique permet de réduire la complexité en stockage d'une dense de $\mathcal{O}(DD')$ à $\mathcal{O}(D')$ où D est la dimension en entrée de la couche et D' la dimension en sortie. Ils proposent par ailleurs une version adaptative de `Fastfood` où les poids des matrices diagonales sont directement appris par descente de gradient. Comme pour l'approximation de `Fastfood` standard des noyaux, la dimension de sortie peut être adaptée en empilant des approximations de `Fastfood` les unes par dessus les autres.

LI, YANG, MARTIN et al., 2015 proposent aussi un exemple minimaliste où ils proposent d'apprendre un réseau de neurone avec une seule couche cachée qui est exprimée comme un produit de matrices «papillon», ce qui lui donne une structure similaire à des transformées rapides connues (voir Section 3.2). Toutefois, cette approche n'a pas été éprouvée sur des architectures neuronales conséquentes. Avec cette technique, le nombre de paramètres à stocker dans une couche dense devient de l'ordre de $\mathcal{O}(D \log D)$.

1.3.4.3 Utilisations de la parcimonie

Une autre approche de la compression des modèles est l'élagage des connections neuronales. Elle bifurque en deux branches : l'élagage «structuré» et l'élagage «non-structuré». Il convient de noter que l'élagage non structuré pourrait toujours être appliqué en plus de l'élagage structuré d'un réseau ; les deux procédures ne s'excluent pas mutuellement.

Élagage structuré. Les techniques d'élagage structuré visent essentiellement à réduire les dimensions des couches cachées du réseau —en supprimant complètement des neurones— tout en conservant un format matriciel dense dans chaque couche afin que le matériel et les bibliothèques habituels puissent toujours être utilisés pour des opérations rapides d'algèbre linéaire. Selon un récent article cependant LIU, SUN, ZHOU et al., 2018, ces techniques s'apparentent plus à de la «Recherche d'architecture neuronale» (*Neural Architecture Search (NAS)*) qu'à un véritable élagage car les expériences montrent que les architectures élaguées semblent fonctionner aussi bien lorsque les poids sont réinitialisés de manière aléatoire, c'est à dire que l'architecture trouvée par la méthode est plus importante que les poids eux mêmes.

Élagage non-structuré. Les techniques d'élagage non-structuré visent à supprimer les connexions pondérées dans le réseau, c'est-à-dire mettre à zéro des poids, ce qui permet d'effectuer des opérations matricielles creuses. Ceci peut être avantageux lorsque la parallélisation n'est pas possible ou avec du matériel moderne permettant l'utilisation d'algorithmes efficaces pour la multiplication de matrices creuses (YAVITS, MORAD et GINOSAR, 2017). Contrairement à l'élagage structuré, d'autres travaux récents (FRANKLE et CARBIN, 2019) ont montré empiriquement que les poids trouvés par une telle stratégie d'élagage sont effectivement importants car la réinitialisation des poids sur le support de parcimonie a un effet néfaste sur les performances. La suppression des connexions dans les techniques d'élagage non-structuré se fait par la remise à zéro de certains éléments dans le tenseur de poids d'une couche. Les approches pionnières parmi les techniques d'élagage non-structuré sont *Optimal Brain Dammage* et son amélioration *Optimal Brain Surgeon* proposées respectivement par (LECUN, DENKER et SOLLA, 1990) et HASSIBI et STORK, 1993 qui se basent sur la dérivée seconde de l'erreur par rapport aux poids pour déterminer quels sont les poids de plus faible importance. une approche plus simple et plus directe consiste à supprimer simplement les poids les plus faibles jusqu'à ce qu'un rapport donné de parcimonie soit atteint (ZHU et GUPTA, 2017). D'autres approches ont été proposées, comme celle basée sur le *dropout* variationnel (MOLCHANOV, ASHUKHA et VETROV, 2017) ou cette autre proposée par LOUIZOS, WELLING et KINGMA, 2018 qui tente une sorte de régularisation L_0 en optimisant les paramètres d'une distribution de Bernoulli.

1.3.4.4 Quantification des poids

La quantification dans les réseaux de neurones est une technique visant à représenter chaque poids, activation ou valeur de gradient par un élément issu d'un «dictionnaire» (*codebook*). Les éléments du dictionnaire sont typiquement codés sur un (COURBARIAUX, BENGIO et DAVID, 2016 ; RASTEGARI, ORDONEZ, REDMON et al., 2016) ou quelques bits (ZHOU, YAO, GUO et al., 2017), permettant ainsi de grandement réduire l'espace de stockage nécessaire pour les poids du réseau. Les technique de quantification peuvent par ailleurs permettre d'accélérer la vitesse d'exécution du réseau en utilisant par exemple de l'algèbre booléen simple en lieu et place des multiplications habituelles.

Une autre approche de la quantification est la quantification de vecteurs. L'idée ici est d'utiliser un algorithme de regroupement tel que l'algorithme des K-moyennes (voir Section 3.3.2) sur les colonnes des matrices de poids du réseau (GONG, LIU, YANG et al., 2014) et ensuite de remplacer les colonnes de poids par le centroïde auquel elles se rattachent. Une variante de cette méthode est la technique du produit de quantification (JÉGOU, DOUZE et SCHMID, 2011) où ce sont des morceaux de colonnes des matrices de poids qui sont regroupés et donc les colonnes de poids dans le réseau sont remplacés par une concaténation

de centroïdes (STOCK, JOULIN, GRIBONVAL et al., 2020). Les approches par quantification de vecteurs nécessitent habituellement d'utiliser un réseau pré-entraîné mais une étape de raffinage (*fine tuning)* des centroïdes peut être faite dans un second temps en utilisant la rétro-propagation du gradient comme proposé initialement par HAN, MAO et DALLY, 2016. Plus de détails sur la quantification de poids dans un réseau peuvent être trouvés dans la revue de littérature de GUO, 2018.

1.3.4.5 Distillation de connaissances

L'idée de base de la distillation de connaissances est de construire un modèle «étudiant» (*student*) de petite taille à partir d'un modèle «enseignant» (*teacher*) de taille conséquente. Cette idée initialement proposée par BUCILUǍ, CARUANA et NICULESCU-MIZIL, 2006 a été ensuite appliquée aux réseaux de neurones par HINTON, VINYALS et DEAN, 2015. L'idée de HINTON, VINYALS et DEAN, 2015 est d'utiliser le vecteurs de prédictions renvoyé par le modèle enseignant pour superviser le modèle étudiant, en plus des vecteurs des étiquettes réelles, car le vecteur de prédictions contient en quelques sortes l'information de la relation entre les différentes étiquettes. Par exemple, pour la reconnaissance de chiffre manuscrits : l'étiquette «3» est aussi éloignée de l'étiquette «8» que de l'étiquette «1». Or, visuellement, le chiffre «3» ressemble plus au chiffre «8» qu'au chiffre «1». Il est probable que le réseau enseignant ait appris cette similarité et que celle-ci apparaisse dans le vecteur de prédictions. Superviser le réseau étudiant avec ce vecteur de prédictions pourra lui permettre d'intégrer plus simplement les relations qui lie des classes différentes entre elles. Cette idée a ensuite été utilisée dans beaucoup de modèles de compression pour faciliter l'étape de raffinage du réseau comprimé. Un exemple notable d'utilisation est celui de ROMERO, BALLAS, KAHOU et al., 2015 où les auteurs appliquent la distillation sur des représentations intermédiaires d'un réseau profond. Plus de détails sur la distillation de réseaux de neurones peuvent être trouvés dans la revue de littérature écrite par GOU, YU, MAYBANK et al., 2020.

1.3.5 Conclusion sur les réseaux de neurones

Les réseaux neuronaux sont des modèles d'apprentissage complexes, composés d'une interconnexion de modules élémentaires, qui sont capables d'obtenir d'excellentes performance sur des tâches de traitement de données structurées, telles que les images notamment. Les résultats théoriques et expérimentaux semblent indiquer que plus les réseaux sont larges et profonds, plus ils obtiennent de bonnes performances. Pendant un temps, une direction de recherche privilégiée a donc été vers le développement de nouvelles stratégies pour permettre l'apprentissage de réseaux toujours plus gros, plus puissants. Toutefois, ces réseaux deviennent si chers à exécuter qu'il est crucial aujourd'hui de les alléger pour pou-

voir effectivement les déployer en production. C'est pourquoi un nouvel axe de recherche très populaire consiste à explorer les différentes approches pour compresser des réseaux de neurones pré-entraînés : d'abord un réseau sur-paramétré et entraîné pour obtenir de bonnes performances ; puis des algorithmes de compression sont appliqués sur le réseau afin d'en réduire le nombre de paramètres tout en préservant sa performance.

1.4 Conclusion

Dans ce chapitre, nous avons introduit les concepts essentiels des méthodes à noyaux et des réseaux neuronaux qui seront utilisés au fil de ce document de thèse. Nous nous sommes notamment intéressés aux notions de représentations non-linéaires et de compression pour ces méthodes.

Dans cet état de l'art ainsi que dans la littérature, les méthodes à noyaux et réseaux de neurones ont essentiellement été étudiés séparément. Toutefois, des travaux récents ont visé à mélanger ces deux familles de méthodes. Ceux-ci se divisent essentiellement en trois types d'approches.

Certains travaux se focalisent sur l'idée d'expliquer le fonctionnement de l'apprentissage des réseaux de neurones artificiels en utilisant les méthodes à noyaux. En particulier, les *Neural Tangent Kernel* de JACOT, GABRIEL et HONGLER, 2018 permettent de décrire l'apprentissage des réseaux neuronaux par descente de gradient. Ces noyaux apportent des éclaircissements sur la capacité des réseaux sur-paramétrés à bien généraliser (ALLEN-ZHU, LI et SONG, 2019 ; ARORA, DU, HU et al., 2019).

Quelques travaux ont exploré la conception de noyaux profonds qui permettraient de travailler avec une hiérarchie (un empilement) de représentations de façon analogue à l'apprentissage profond CHO et SAUL, 2009 ; MONTAVON, BRAUN et MÜLLER, 2011 ; JOSE, GOYAL, AGGRWAL et al., 2013 ; HEINEMANN, LIVNI, EBAN et al., 2016 ; STEINWART, THOMANN et SCHMID, 2016 ; WILSON, HU, SALAKHUTDINOV et al., 2016.

D'autres études ont porté sur les différentes façons de "brancher" les noyaux dans les réseaux profonds en utilisant notamment des méthodes d'approximation de noyaux (MAIRAL, KONIUSZ, HARCHAOUI et al., 2014 ; YANG, MOCZULSKI, DENIL et al., 2015 ; HAZAN et JAAKKOLA, 2015 ; MAIRAL, 2016 ; ZHANG, LI, XIE et al., 2017). Parmi eux, on remarque notamment les travaux de MAIRAL, KONIUSZ, HARCHAOUI et al., 2014 ; MAIRAL, 2016 ; CHEN, JACOB et MAIRAL, 2019a qui proposent un nouveau type d'architecture appelée les *Convolutional Kernel Networks* qui utilisent les fonctions noyaux pour imiter le calcul de couches convolutives. Cette méthode s'appuie en pratique sur la méthode de Nyström pour rendre le calcul de chaque couche réalisable. Elle a ensuite été étendue au traitement de donnée séquentielles (CHEN, JACOB et MAIRAL, 2019b) et de données en forme de graphe (CHEN, JACOB et MAIRAL, 2020).

Le chapitre suivant présente une contribution de cette thèse qui se place dans cette dernière ligne de recherche avec la proposition d'une nouvelle couche pour les réseaux de neurones appelée "couche Nyström".

2 Approximation de Nyström adaptative pour l'apprentissage de réseaux neuronaux

Contents

2.1 Introduction

Les machines à noyaux et l'apprentissage profond ont principalement été étudiés séparément. Pourtant, tous deux ont des forces et des faiblesses et apparaissent comme des familles de méthodes complémentaires. Les méthodes d'apprentissage profond peuvent apprendre à partir de zéro des caractéristiques pertinentes des données mais nécessitent en fait un grand volume de données pour exploiter pleinement leur potentiel et peuvent ne pas donner de bons résultats avec des ensembles de données d'entraînement limités. En outre, les réseaux profonds sont complexes et difficiles à concevoir et nécessitent beaucoup de ressources informatiques et de mémoire, tant pour l'entraînement que pour l'inférence. Les méthodes à noyaux quant-à elles sont des outils puissants pour l'apprentissage des relations non linéaires dans les données et sont bien adaptées aux problèmes liés à des ensembles de données d'entraînement de petite taille. Leur puissance vient de leur capacité à étendre les méthodes linéaires aux méthodes non linéaires

avec des garanties théoriques. Cependant, elles ne s'adaptent pas bien au cas où les ensembles de données d'entraînement sont grands et elles n'apprennent pas une représentation à partir des données. Elles nécessitent généralement le choix préalable d'un noyau pertinent parmi les plus connus, voire la définition d'un noyau approprié pour une tâche particulière.

Il existe un certain nombre d'études à l'interface des domaines de l'apprentissage profond et des méthodes à noyau qui ont examiné comment certains concepts peuvent être transférés d'un domaine à l'autre. Parmi elles, certaines se sont penchées sur la question de "brancher" les noyaux dans les réseaux profonds, notamment en utilisant des méthodes d'approximation de noyaux car celles-ci peuvent passer à l'échelle de grands volumes de données MAIRAL, KONIUSZ, HARCHAOUI et al., 2014 ; YANG, MOCZULSKI, DENIL et al., 2015 ; HAZAN et JAAKKOLA, 2015 ; MAIRAL, 2016 ; ZHANG, LI, XIE et al., 2017. Les `Deepfried Convnets` que nous avons vu en Section 1.3.4.2, par exemple, visent à remplacer les dernières couches denses des réseaux neuronaux par une approximation de `Fastfood`. L'approximation de Nyström a aussi été étudiée en conjonction avec les réseaux de neurones. MAIRAL, 2016 utilise la méthode de Nyström pour des réseaux de noyaux convolutifs en projetant les caractéristiques des données dans un espace de faible dimension. CROCE, ROSSINI et BASILI, 2018 l'utilisent pour transformer les données d'entrée avant de les transmettre à un réseau de neurones afin d'obtenir une meilleure interprétabilité. Enfin, SONG, THIAGARAJAN, SATTIGERI et al., 2018 ont utilisé un ensemble d'approximations de Nyström et a ensuite appliqué un réseau de neurones pour optimiser une machine à noyaux.

Une caractéristique intéressante de la méthode de Nyström est que son nombre de paramètres à dépend surtout du nombre d'observations de référence et pas tant de la dimension des données. Cette observation nous a amené à étudier l'utilisation de la méthode de Nyström en remplacement des couches standard entièrement connectées dans les réseaux profonds.

Résumé des contributions Nous proposons d'utiliser l'approximation de Nyström pour remplacer les couches entièrement connectées des réseaux de neurones profonds. Ceci permet d'apprendre une métrique sur l'espace des caractéristiques induites par les filtres de convolution qui précèdent les couches entièrement connectées dans les architectures classiques que nous considérons (Lenet, VGG). L'un des principaux avantages de notre méthode est sa flexibilité qui permet d'utiliser n'importe quelle fonction du noyau ou combinaison de fonctions noyau.

Nos expériences sur quatre ensembles de données (`MNIST`, `SVHN`, `CIFAR10` et `CIFAR100`) mettent en évidence trois caractéristiques importantes de notre méthode. Premièrement, notre approche innovante se compare bien aux réseaux neuronaux standard tout en nécessitant un nombre réduit de paramètres à apprendre. Ensuite, ce nombre réduit de paramètres dans notre modèle permet de traiter des ensembles d'entraînement de taille limitée, comme nous le mon-

trons en réalisant des expériences avec des dizaines, voire moins, d'échantillons d'entraînement par classe. Enfin, la méthode peut exploiter plusieurs noyaux, ce qui permet de mettre en œuvre une version «profonde» de l'apprentissage multi-noyaux (MKL) (voir Section 1.2.2.1) ou de prendre en compte la richesse des informations contenues dans les multiples cartes de caractéristiques des réseaux de convolution.

2.2 Couche Nyström adaptative

Nous proposons de remplacer les couches denses d'un réseau neuronal par l'approximation de Nyström d'une fonction noyau. Cela permet de tirer parti de la faible complexité de la méthode Nytsröm pour réduire considérablement le nombre de paramètres à apprendre des couches entièrement connectées dans les réseaux de neurones. Nous introduisons les couches Nyström dans le contexte des réseaux de neurones convolutifs classiques où les couches denses succèdent simplement aux couches de convolution. Toutefois, elles peuvent également être appliqués à d'autres architectures de réseaux neuronaux profonds. Par ailleurs, elles peuvent fonctionner avec n'importe quelle fonction noyau ou combinaison de noyaux, et donc ne nécessitent pas obligatoirement de représentation vectorielle en entrée. Tout d'abord, nous commençons par revoir le concept d'approximation de Nyström du point de vue des *Empirical Kernel Map* (EKM) ; ensuite nous décrivons le principe de fonctionnement de la couche Nyström adaptative et comment elle peut être utilisée dans le cadre de l'apprentissage de noyau (voir Section 1.2.2).

2.2.1 L'approximation de Nyström du point de vue des *Empirical Kernel Map* (EKM)

Les EKM sont des fonctions de représentation explicite en N-dimensions qui sont obtenues en appliquant la fonction noyau sur l'ensemble des données d'entraînement. Elles sont définies comme suit :

$$\begin{aligned} \boldsymbol{\phi}_{emp} : \quad & \mathbb{R}^D \to \mathbb{R}^N \\ & \mathbf{x} \mapsto \mathbf{k}_{\mathbf{x},\mathbf{X}}. \end{aligned}$$

où $\mathbf{k}_{\mathbf{x},\mathbf{X}} := \left[\mathbf{k}(\mathbf{x}, \mathbf{x}_1), \ldots, \mathbf{k}(\mathbf{x}, \mathbf{x}_N)\right]$ pour tout $\mathbf{x}_i \in \mathbf{X}$ et $X = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$.

Une caractéristique intéressante des EKM (SCHOLKOPF, MIKA, BURGES et al., 1999) est que si nous considérons la mesure dans $\mathbb{R}^N$ $\langle \cdot, \cdot \rangle_{\mathbf{W}} = \langle \cdot, \mathbf{W} \cdot \rangle$ avec $\mathbf{W}$ une matrice PSD, alors nous pouvons retrouver la matrice noyau $\mathbf{K}$ grâce à $\boldsymbol{\phi}_{emp}$

en choisissant $\mathbf{W} = \mathbf{K}^{-1}$; formellement, on obtient :

$$\left[\langle \boldsymbol{\phi}_{emp}(\mathbf{x}_i), \boldsymbol{\phi}_{emp}(\mathbf{x}_j)\rangle_{\mathbf{K}}^{-1}\right]_{i,j=1}^{N} = \mathbf{K}\mathbf{K}^{-1}\mathbf{K} = \mathbf{K}. \tag{2.1}$$

Dans ce cas, nous pouvons construire la fonction de représentation explicite $\boldsymbol{\phi}'_{emp} : \mathbf{x} \mapsto \mathbf{K}^{-1/2}\boldsymbol{\phi}_{emp}(\mathbf{x})$ qui permet de reconstruire exactement la matrice noyau $\mathbf{K}$ obtenue sur l'ensemble d'apprentissage. Toutefois, cette fonction offre une représentation en N dimensions qui n'est pas intéressante en pratique lorsque le nombre d'exemples d'entraînement est grand.

Du point de vue des EKM, la fonction de représentation donnée par l'approximation de Nyström (Équation (1.24)) est répétée ici :

$$\boldsymbol{\phi}_{nys}(\mathbf{x}) = \mathbf{k}_{\mathbf{x},\mathbf{L}}\mathbf{K}_{11}^{-1/2}. \tag{2.2}$$

Elle peut-être considérée comme une EKM sur un sous-ensemble des données d'apprentissage et $\mathbf{K}_{11}^{-1/2}$ définirait une mesure dans l'espace des caractéristiques empiriques, c'est-à-dire l'espace de sortie de l'EKM. Suivant cette perspective nous pensons qu'il pourrait être utile d'apprendre la mesure dans l'espace des caractéristiques empiriques plutôt que de supposer qu'elle est définie par $\mathbf{K}_{11}^{-1/2}$. Dans un sens, cela devrait permettre d'apprendre un noyau via l'apprentissage de sa représentation de Nyström. Dans ce qui suit, nous appelons «Nyström adaptatif» la méthode qui consiste à apprendre la matrice $\mathbf{W}$ en utilisant la rétropropagation du gradient dans le réseau neuronal.

2.2.2 Principe de la méthode

Comme l'illustre la Figure 2.1, le modèle que nous proposons est basé sur l'utilisation de l'approximation de Nyström pour intégrer une fonction noyau quelconque après les couches convolutives d'un réseau de neurones profond.

En partant d'un réseau neuronal profond habituel, nous remplaçons les couches cachées supérieures –les couches entièrement connectées– par la fonction de représentation $\boldsymbol{\phi}_{nys}$ (Équation (2.2)). Pour calculer cette représentation de Nyström d'un échantillon $\mathbf{x}$, il faut considérer un sous-échantillon $\mathbf{L}$ de L observations de référence.

On note qu'avant d'atteindre la couche Nyström, les observations sont d'abord traitées par les couches de convolution $conv$. Pour permettre de calculer la valeur du noyau, les échantillons de $\mathbf{L}$ doivent être représentés dans le même espace et doivent donc être traités par les couches convolutives également. Une fois les représentations convolutives calculées, la fonction noyau peut être calculée entre la représentation de l'exemple à traiter $conv(\mathbf{x})$ et chaque instance de $conv(\mathbf{L})$; ceci permet d'obtenir ce que nous appelons le vecteur noyau $\mathbf{k}_{\mathbf{x},\mathbf{L}}$ qui est ensuite transformé linéairement par une matrice $\mathbf{W}$ avant la couche de classification linéaire (voir Figure 2.1). Cependant, un problème se pose avec $\mathbf{W}$ qui est définie

par $\mathbf{K}_{11}^{-1/2}$ dans l'approximation de Nyström car celle-ci nécessite le calcul de la SVD de $\mathbf{K}_{11}$ à chaque itération lors de l'apprentissage. En effet, après chaque itération, les poids des couches de convolution changent. Donc les représentations des observations de référence ainsi que leur matrice noyau $\mathbf{K}_{11}$ associée changent aussi. Dans le cas où la taille du sous-échantillon $\mathbf{L}$, L, est relativement grande, la complexité calculatoire de la SVD ($\mathcal{O}(L^3)$) rendrait l'apprentissage intenable. Cette question peut être au moins partiellement réglée par la couche Nyström adaptative où, au lieu de fixer les poids $\mathbf{W} = \mathbf{K}_{11}^{-1/2}$ comme dans Équation (2.2), on apprend les poids en tant que paramètres du réseau via la descente de gradient.

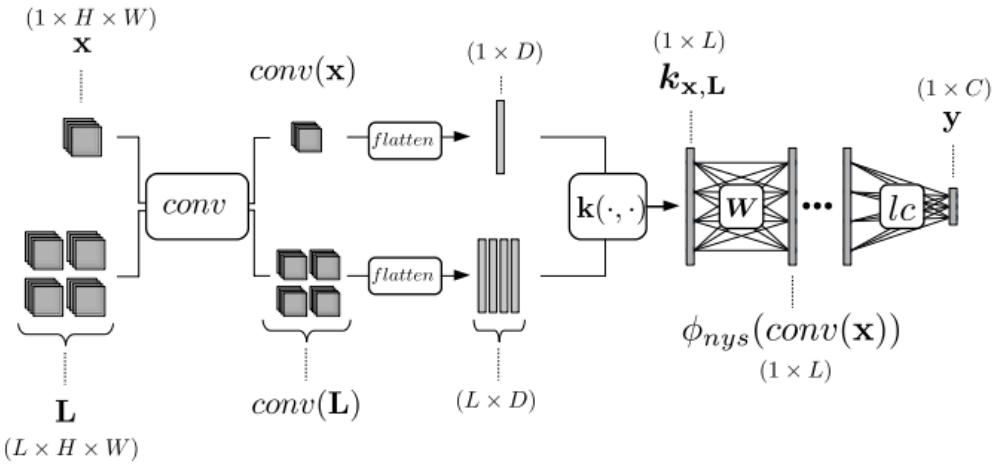


FIGURE 2.1 : L'architecture proposée est constituée d'une partie convolutive, *conv*, d'une couche Nyström et d'une couche de classification. La couche de Nyström calcule la fonction noyau entre la sortie de la convolution de l'exemple à classifier ainsi que des exemples du sous-ensemble d'apprentissage L. Ensuite, une transformation linéaire $\mathbf{W}$ est appliquée au vecteur noyau $\mathbf{k}_{\mathbf{x},\mathbf{L}}$ ainsi obtenu. La couche $flatten$ correspond à une couche de vectorisation.

L'utilisation de la couche Nyström adaptative a deux avantages essentiels par rapport à l'utilisation d'une couche dense standard ou même par rapport aux `Deep Fried Convnets` de YANG, MOCZULSKI, DENIL et al., 2015 qui consistent également à remplacer les couches denses supérieures d'un réseau neuronal convolutif en utilisant une approximation de noyau, mais en utilisant l'approximation de `Fastfood`. Nous décrivons maintenant ces avantages.

D'abord, l'utilisation de Nyström adaptatif permet de réduire la quantité de poids à apprendre : $\mathcal{O}(L^2)$ pour la couche Nyström contre $\mathcal{O}(D^2)$ pour une couche dense carrée ou même $\mathcal{O}(D)$ pour une couche `Fastfood`, où D est la dimension des observations en entrée de l'une ou l'autre de ces couches. En pratique, le nombre d'observations de références L est bien inférieur à la di-

mension des données D, ce qui amène un nombre de paramètres à apprendre dans la couche bien inférieur aux techniques concurrentes (nous le verrons empiriquement en Section 2.3.2). Par ailleurs, le nombre de poids à apprendre dans la couche suivante –habituellement une couche de classification– passe de $\mathcal{O}(DC)$ dans le cas d'une couche entièrement connectée ou d'une couche `Fastfood` contre $\mathcal{O}(LC)$ pour la couche de Nyström adaptative, avec C le nombre de classes (ou le nombre de neurones dans la couche suivante). Dans cette analyse de complexité, nous avons parlé de poids *à apprendre* et cette tournure n'est pas anodine. En effet, nous remarquons que l'utilisation de la couche Nyström nécessite de conserver en mémoire l'ensemble des observations de référence $\mathbf{L}$, ce qui représente un coût en stockage de l'ordre de $\mathcal{O}(LD)$. Toutefois, même si son intérêt pratique est limité en terme de compression de réseau de neurones, nous pensons que la couche Nyström peut se révéler être avantageuse dans le contexte de l'apprentissage avec peu de données où une couche dense standard fonctionnerait assez mal. Cette intuition est illustrée expérimentalement en Section 2.3.3.

Ensuite, contrairement aux `Deep Fried Convnets`, la couche Nyström est générique : elle permet d'utiliser n'importe quels noyaux ou même de combiner ceux-ci. La couche Nyström adaptative permet de réaliser l'apprentissage de noyaux multiples, ou les noyaux $\mathbf{k}^1, \dots, \mathbf{k}^K$ correspondent à K couches de Nyström qui peuvent être calculées en parallèle puis fusionnées pour encoder les informations fournies par les différentes représentations des noyaux. L'apprentissage des matrices de poids $\mathbf{W}_1, \dots, \mathbf{W}_K$ dans ce cas est, d'une certaine manière, lié au MKL (voir Section 1.2.2.1). Il est parfois difficile de savoir à l'avance quel noyau sera le plus performant pour une tâche particulière, comme le montre les résultats expérimentaux de la Figure 2.2 où on voit qu'en fonction de la tâche, le noyau le plus efficace peut varier. Une stratégie de fusion possible pour les différentes représentations fournies par les différents noyaux consiste simplement à les concaténer, par exemple on peut faire :

$$\boldsymbol{\phi}_{nys_{mkl}} = \begin{bmatrix} \mathbf{W}_1 \boldsymbol{k}^1_{\mathbf{x},\mathbf{L}} \\ \vdots \\ \mathbf{W}_K \boldsymbol{k}^K_{\mathbf{x},\mathbf{L}} \end{bmatrix} \tag{2.3}$$

où pour tout i, $\mathbf{k}^i_{\mathbf{x},\mathbf{L}}$ est le vecteur noyau obtenu à l'aide de la $i^{ème}$ fonction noyau $\mathbf{k}_i$. Utiliser une concaténation de couches Nyström correspond d'une certaine façon à réaliser un apprentissage de noyaux multiples où le noyau à apprendre est en fait une somme de noyaux (voir Section 1.2.2).

Dans la section suivante, nous illustrons l'efficacité de notre méthode sur certains ensembles de données de classification d'images classiques, puis nous essayons d'explorer ses extensions possibles pour l'apprentissage avec peu de données et l'apprentissage à noyaux multiples, en exploitant certains avantages bien connus des méthodes à noyaux.

2.3 Résultats expérimentaux

Nous présentons une série de résultats expérimentaux qui permettent d'explorer le potentiel des réseaux profonds contenant des couches Nyström adaptatives sur plusieurs tâches de classification d'images. Nous parlons «de réseau Nyström adaptatif» pour faire référence à ce type de réseau. Nous considérons d'abord un cadre plutôt standard et comparons notre approche avec des modèles habituels en faisant varier le nombre total de paramètres *à apprendre*. Nous explorons en particulier le comportement des couches Nyström adaptatives avec divers noyaux et soulignons la taille très limitée du sous-échantillon $\mathbf{L}$ nécessaire pour obtenir une bonne performance en terme de précision. Ensuite, nous étudions l'utilisation des couches de Nyström dans le cadre d'une tâche d'apprentissage avec un petit jeu de données. En effet, le nombre réduire de paramètres à apprendre dans les couches Nyström adaptatif nous amène à penser que notre approche peut permettre d'apprendre avec très peu d'exemples d'entraînement. Enfin, nous étudions une architecture avec des noyaux multiples et illustrons son intérêt pour sélectionner l'hyper-paramètre d'un noyau RBF. Nous démontrons également l'avantage d'une approche Nyström multiple, combinant des noyaux calculés à partir des cartes de convolution individuelles. Avant de décrire tous ces résultats, nous détaillons les paramètres expérimentaux nos expériences.

2.3.1 Paramètres expérimentaux

Nom	Dimensions	# classes	Taille d'entraînement	Taille de validation	Taille de test
`MNIST`	$(28 \times 28 \times 1)$	10	40 000	10 000	10 000
`SVHN`	$(32 \times 32 \times 3)$	10	63 257	10 000	26 032
`CIFAR10`	$(32 \times 32 \times 3)$	10	50 000	10 000	10 000
`CIFAR100`	$(32 \times 32 \times 3)$	100	50 000	10 000	10 000

TABLE 2.1 : Charactéristiques des jeux de données

Jeux de données. Nous avons mené des expériences sur quatre ensembles de données de classification d'images bien connus : `MNIST` (DENG, 2012), `SVHN` (NETZER, WANG, COATES et al., 2011), `CIFAR10` et `CIFAR100` (KRIZHEVSKY, 2009), les détails de ces ensembles de données sont fournis dans le Tableau 2.1.

Architectures neuronales. Nous avons implémenté les couches convolutives en utilisant deux architectures standard sur ces jeux de données : `Lenet` (LECUN, BOTTOU, BENGIO et al., 1998) pour `MNIST` et `VGG19` (SIMONYAN et ZISSERMAN, 2015) pour `SVHN`, `CIFAR10` et `CIFAR100`. Nous avons légèrement modifié la taille des filtres dans le réseau `Lenet` afin de garantir que la dimension des données

après les blocs de convolution soit une puissance de 2 (pratique pour l'architecture de convolution `Deep Fried Convnets` à laquelle nous nous comparons). Ces expériences visent à mettre en évidence le potentiel de l'apprentissage d'une couche Nyström adaptative dans une architecture profonde. Les expériences font intervenir des couches de convolution pré-entraînés et fixées pour l'apprentissage des couches supérieures.

Nous comparons trois architectures dans toutes les expériences menées. Les trois architectures sont constituées de couches de convolutions pré-entraînées identiques mais diffèrent par leurs couches supérieures : (1) Les architectures dites "denses" qui utilisent des couches cachées denses, c'est-à-dire que ce sont des architectures de convolution classiques ; (2) les architectures "`Deep Fried Convnets` adaptative" qui mettent en œuvre l'approximation `Fastfood` (voir Section 1.3.4.2) ; (3) les architectures "Nyström" qui correspondent à la couche Nyström adaptative.

Pour les architectures denses, nous avons considéré une couche cachée avec fonction d'activation *relu* (voir Section 1.3.3), et avons fait varier la dimension de sortie parmi {2, 4, 8, 16, 32, 64, 128, 1024} afin de bien mettre en évidence l'évolution de la qualité du modèle en fonction du nombre de paramètres. L'exploitation de plusieurs couches cachées n'a pas augmenté de manière significative les performances dans nos expériences et donc nous n'en conservons qu'une seule afin de faciliter la lisibilité des résultats. Pour l'approximation de `Fastfood` dans `Deep Fried Convnets`, nous considérons que ϕ_{ff} (Équation 1.20) donne une représentation de même taille que sa dimension d'entrée, sauf dans les expériences de la Section 2.3.4 qui donne une dimension de représentation jusqu'à 5 fois plus grande (dans ce cas il s'agit d'un empilement de 5 représentations de `Fastfood` en parallèle). En ce qui concerne notre approche et ϕ_{nys}, nous avons fait varier la taille du sous-ensemble $\mathbf{L}$ parmi {2, 4, 8, 16, 32, 64, 128} et nous avons comparé les noyaux linéaires, RBF et *Chi2* (voir Section 1.2.2). Enfin, nous en avons exploré les variantes adaptatives et non adaptatives de la couche Nyström ; dans le cas non adaptatif, c'est la matrice $\mathbf{K}_{11}^{-1/2}$ obtenue avec les couches de convolutions pré-entraînées qui est figée.

Hyper-paramètres. Les modèles ont été entrainés sur 50 itérations pour optimiser un critère d'entropie croisée (Équation (1.32)) avec l'optimiseur `Adam` et un pas de gradient fixé à $1e^{-4}$. Par défaut, la largeur de bande du noyau RBF est fixée à la distance moyenne entre les représentations convolutives de paires d'échantillons d'entraînement (Équation (1.8)).

Détails d'implémentation. Toutes les expériences ont été réalisées avec Keras (CHOLLET, 2015) et Tensorflow (ABADI, AGARWAL, BARHAM et al., 2015). L'ensemble des paramètres utilisés et du code pour générer les résultats et les figures sont disponibles publiquement sur deux dépôts du laboratoire (GIFFON,

2020a ; GIFFON, 2020b). Un exemple de fonctionnement minimum de la couche Nyström est aussi disponible sur un dépôt séparé (GIFFON, 2020c).

Les expériences ci-dessous explorent le potentiel de notre architecture et n'ont pas la prétention d'essayer de battre les résultats actuels de l'état de l'art sur les ensembles de données considérés. Par conséquent, nous n'avons pas eu recours à des astuces telles que l'augmentation des données ni à un réglage précautionneux des hyper-paramètres. Enfin, bien que certains travaux aient été réalisés sur des techniques d'échantillonnage plus avancés pour l'approximation de Nyström, nous avons adopté un échantillonnage uniforme stratifié des exemples de l'ensemble de d'entraînement labellisé. Comme l'ont montré KUMAR, MOHRI et TALWALKAR, 2012, cette technique constitue une bonne stratégie d'échantillonnage en termes de rapport coût/performance.

2.3.2 Exploration du potentiel de la méthode

Nous comparons maintenant les réseaux Nyström adaptatifs profonds à deux architectures similaires : les `Deep Fried Convnets` et les réseaux convolutionnels classiques (inspirés de `VGG19` ou `Lenet` selon l'ensemble de données). Nous faisons varier le nombre de paramètres de chaque architecture afin de mettre en évidence la précision de la classification par rapport à la quantité de poids à apprendre. Ces observations nous encouragent dans la Section 2.3.3 à explorer la capacité des réseaux Nyström adaptatif à apprendre avec peu de données d'apprentissage.

La Figure 2.2 montre la précision des réseaux par rapport au nombre de paramètres estimés dans la dernière couche de représentation et la couche de classification linéaire. Notons que nous ignorons les paramètres des couches de convolutions pour faciliter la lisibilité des résultats puisque ces couches sont identiques dans les différents modèles comparés. Nous avons répété chaque expérience 10 fois et avons tracé les moyennes et écarts-types. Les résultats obtenus avec la couche Nyström, d'une complexité croissante (nombre de paramètres), correspondent à l'utilisation d'un sous-échantillon de taille croissante, passant de 2 (point le plus à gauche) à 128 (point le plus à droite). On peut voir qu'il n'est pas nécessaire de constituer un grand sous-échantillon dans ces expériences. Cela peut s'expliquer par le fait que la partie convolutive du réseau a appris à produire des représentations assez robustes et stables des images d'entrée ; nous illustrons ceci en Section 2.3.5 (Figure 2.4).

Le réseau Nyström adaptatif profond est capable d'atteindre la même performance de précision tout en apprenant beaucoup moins de paramètres que des couches classiques entièrement connectées. De plus, nous observons également des variations plus faibles entre les réplicats (illustré par de petits écarts types), ce qui souligne la robustesse et la stabilité de notre modèle. La flexibilité dans le choix de la fonction du noyau est un avantage de notre méthode, comme illustré, puisque le meilleur noyau dépend apparemment de l'ensemble de don-

nées : noyau linéaire sur `MNIST`, Chi2 sur `SVHN` et `CIFAR100`, RBF sur `CIFAR10`. Nous remarquons également l'avantage des variantes adaptatives de la couche de Nyström, ce qui suggère que notre modèle est capable d'apprendre une représentation explicite du noyau adapté à la tâche, plus expressive que celle que l'on pourrait obtenir à partir d'une approximation de Nyström classique et avec encore moins de paramètres à apprendre que l'approximation de `Fastfood`.

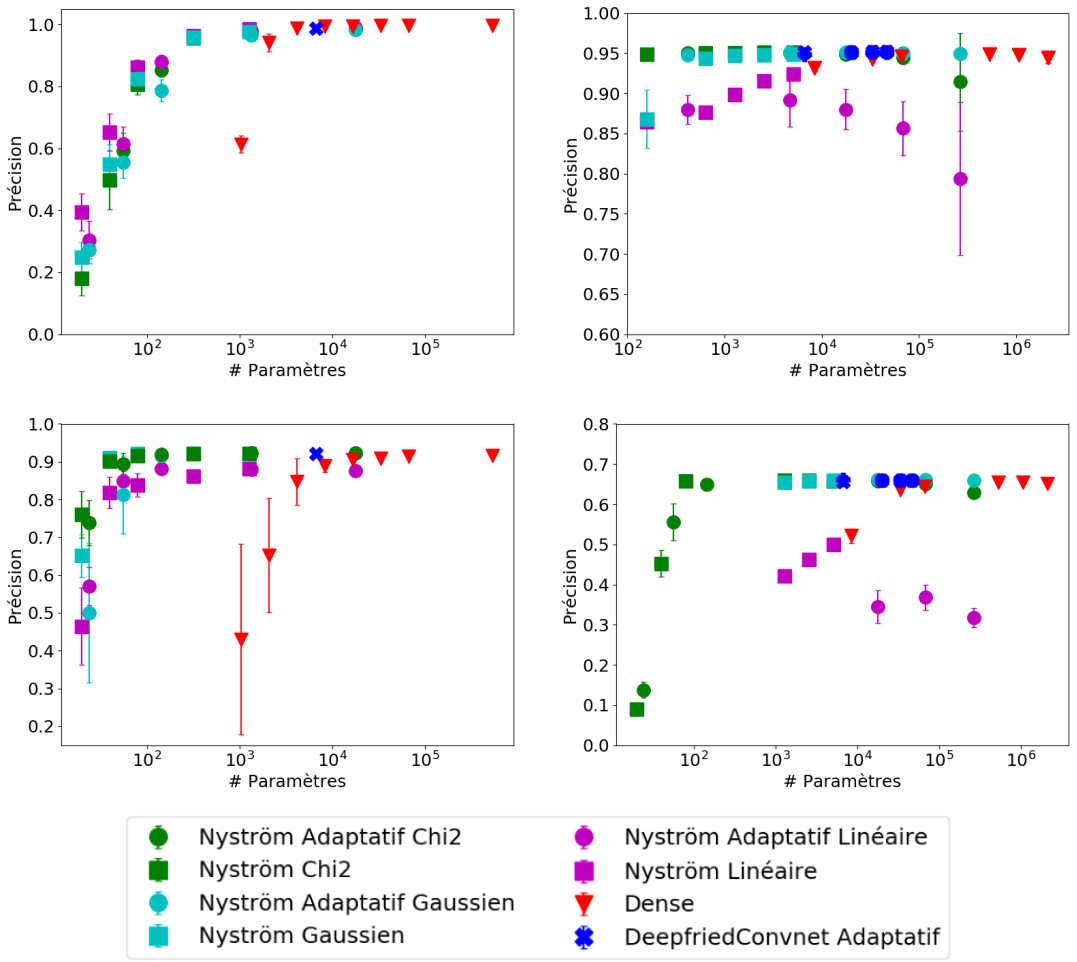


FIGURE 2.2 : Précision des modèles en fonction du nombre de paramètres à apprendre (paramètres du module de convolution non-inclus). Les jeux de données considérés sont MNIST (en haut à gauche), SVHN (en haut à droite), CIFAR10 (en bas à gauche) et CIFAR100 (en bas à droite). Dans le cas de l'architecture Nyström, les noyaux *Chi2*, linéaire et RBF (gaussien) sont évalués.

	MNIST		SVHN		CIFAR10		CIFAR100	
	5	20	5	20	5	20	5	20
Dense	**49.7** (4)	94.4 (0.5)	65.6 (11.6)	81.7 (3.9)	39.1 (3.3)	87.1 (3.7)	19.2 (2.2)	35.7 (2.7)
`Deepfried` Adaptatif	12.4 (3.3)	12.4 (1.4)	16.7 (5)	21.0 (6.4)	28.3 (9.2)	41.2 (3.6)	3.9 (1.2)	6.4 (0.8)
Nyström Adaptatif L	48.1 (5.5)	95.0 (0.5)	22.4 (6.9)	29.6 (13.5)	12.0 (5.6)	27.8 (7.6)	1.2 (0.6)	1.9 (0.8)
Nyström Adaptatif R	41.2 (7.7)	**95.5** (0.3)	42.1 (29.6)	53.5 (33.6)	**70.8** (4.4)	**92.2** (0.1)	**24.7** (2.6)	**62.1** (1.2)
Nyström Adaptatif C	26.4 (7.7)	92.3 (1.8)	**89.6** (3.1)	**93.3** (1.3)	67.1 (4.7)	**92.2** (1)	20.2 (2.2)	55.4 (1.9)

TABLE 2.2 : Précision en classification des architectures Dense, *Deep Fried Convnets* (*Deepfried*) et Nyström adaptatif avec les noyaux linéaire (L), RBF Gaussien (R) et *Chi2* (C). Les jeux de données considérés contiennent 5 ou 20 exemples par classe. Les résultats sont moyennés sur 30 réplicats et les écarts types sont donnés entre parenthèses.

2.3.3 Apprentissage avec peu de données

Nous examinons ici la capacité de notre modèle à fonctionner avec peu d'échantillons d'entraînement, une dizaine d'échantillons par classe, ce qui est un avantage attendu de la méthode étant donné le peu de paramètres à apprendre et la disposition notoire des méthodes à noyaux à être capable de tirer profit de petits ensembles d'entraînement.

A partir de couches de convolution pré-entraînées sur l'ensemble complet d'entraînement, nous avons étudié les performances de la couche Nyström adaptative pour ré-apprendre les couches supérieures du modèle en fonction d'un petit sous-ensemble d'apprentissage. Une des perspectives de ce travail est d'exploiter une telle stratégie dans le cadre de l'adaptation de domaine où le modèle convolutif serait pré-entraîné sur des données issues d'un domaine différent de celui des classes finales à reconnaître.

Le Tableau 2.2 présente une comparaison des architectures de réseau sur les quatre ensembles de données. Nous considérons les cas où la couche Nyström adaptative utilise les noyaux linéaires, RBF ou *Chi2* et nous la comparons avec les `Deep Fried Convnets` et une architecture dense standard pour des ensembles d'entraînement de 5 et 20 échantillons par classe. Nous ne considérons ici que les variantes adaptatives, car elles ont donné de meilleurs résultats que leurs homologues non adaptatives. Les résultats sont moyennés sur 30 réplicats.

On peut d'abord constater que les réseaux Nyström adaptatifs profonds dépassent les modèles témoins dans tous les contextes, à l'exception du cas `MNIST` avec 5 exemples d'apprentissage par classe. Le noyau linéaire fonctionne bien sur `MNIST` mais est nettement moins performant que les autres modèles sur des ensembles de données plus élaborés. À l'opposé, la couche Nyström adaptative avec le noyau RBF gaussien ou le noyau *Chi2* surpasse largement l'architecture `Deep Fried Convnet` pour tous les ensembles de données et a des performances supérieures à celles des architectures denses sur l'ensemble de données `CIFAR100`, le plus dur. Curieusement, les `Deep Fried Convnets` sont particulièrement mauvais dans ce contexte. Enfin, nous constatons qu'aucune représentation de Nyström basée sur un seul noyau ne domine dans tous les cas, ce qui montre l'intérêt

Modèle	Précision (std)	Architecture
Dense	68.0 (0.7)	1 couche cachée de 1024 neuronnes
`Deepfried` Adaptatif	67.6 (0.5)	5 `Fastfood`
Nyström Adaptatif	**69.1** (0.2)	256 échantillons + 512 Noyaux linéaires
Nyström Adaptatif	67.6 (0.2)	16 échantillons + 512 Noyaux Chi2

TABLE 2.3 : Évaluation de l'architecture Nyström multiple sur `CIFAR100`. Les résultats sont moyennés sur 10 réplicats et les écarts types sont montrés entre parenthèses. La colonne de droite décrit l'architecture sélectionnée dans chaque cas.

potentiel de combiner plusieurs noyaux comme le montreront les expériences suivantes.

2.3.4 Apprentissage de noyaux multiples (MKL)

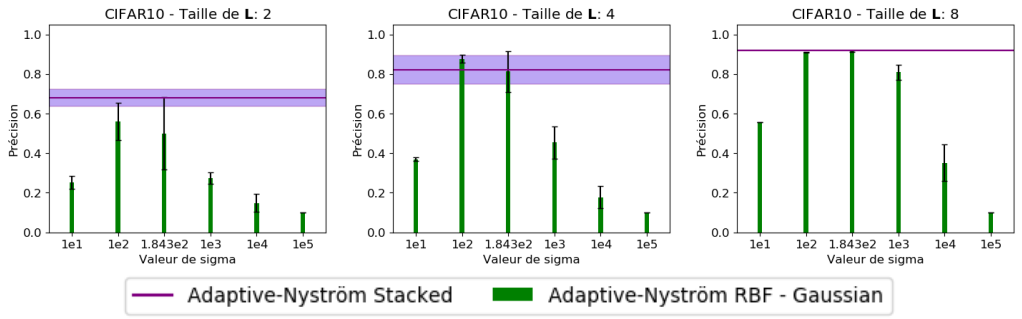


FIGURE 2.3 : Évaluation d'une couche Nyström adaptative qui combine plusieurs noyaux RBF avec des valeurs différentes de σ sur le jeu de données `CIFAR10`. La performance du noyau multiple est illustrée par la ligne horizontale alors que les barres verticales représentent la performance obtenue avec un seul noyau RBF et la valeur de sigma correspondante. Les figures de gauche à droite correspondent aux résultats obtenus avec des sous-ensembles de taille respective $\{2, 4, 8\}$. Les résultats sont moyennés sur 10 réplicats et les écarts types sont représentés par les barres d'erreur ainsi que par la zone plus claire autour de la ligne horizontale correspondant à MKL

Nous présentons ici les résultats validant les stratégies à noyaux multiples (MKL), que nous avons décrites dans la Section 2.2 et l'Équation (2.3). Nous avons mené deux expériences différentes :

Tout d'abord, nous avons envisagé une combinaison de noyaux RBF avec différentes largeurs de bande σ et pour différentes tailles de sous-échantillons. Dans

ce cas, chaque fonction noyau $\{\mathbf{k}^i\}_{i=1}^K$ est une fonction RBF avec une valeur de largeur de bande σ différente (Équation (1.7)). Cette stratégie à noyaux multiples permet de gérer automatiquement le réglage de l'hyper-paramètre σ qui nécessite généralement une procédure intensive de validation croisée. La Figure 2.3 montre la précision de notre méthode sur l'ensemble de données `CIFAR10` en fonction de la valeur de σ. Sur cette figure, la performance de l'architecture Nyström à noyaux multiples est indiquée par une ligne horizontale. Les graphiques présentent les résultats pour différentes taille de sous-échantillons, avec une moyenne sur 10 réplicats. Comme on peut le voir, l'utilisation de notre réseau à noyaux multiples permet d'optimiser la combinaison de noyaux de manière à partir des données sans nécessiter de choix préalable sur la largeur de bande du noyau RBF.

Ensuite, nous avons étudié une variante de l'architecture de Nyström que nous appelons Nyström multiple. Nous considérons ici en parallèle plusieurs approximations de Nyström où les noyaux sont dédiés à traiter chacun la sortie d'une seule carte de caractéristiques des couches de convolution. En d'autre termes, chaque carte de convolution est traitée comme une vue séparée par une fonction noyau qui lui est dédiée. Cette méthode à noyaux multiples diffère de la méthode précédente car cette fois-ci, chaque noyau travaille sur des données d'entrées différentes : soit $conv(\mathbf{x})_i$ la $i^{ème}$ carte de convolution de $\mathbf{x}$: dans l'architecture Nyström multiple, nous utilisons $\mathbf{k}^i(conv(\mathbf{x}), conv(\mathbf{y})) = \mathbf{k}(conv(\mathbf{x})_i, conv(\mathbf{y})_i)$. Le Tableau 2.3 présente les résultats sur le jeu de données `CIFAR100`. Nous montrons les meilleures performances obtenues pour chaque méthode en effectuant une recherche sur une grille d'hyper-paramètres pour chaque modèle, pour des plages de nombre de paramètres similaires. Pour le modèle dense, nous avons considéré une ou deux couches cachées de 16, 64, 128, 1024, 2048 ou 4096 neurones. Pour `Deep Fried Convnets` nous avons fait varier la dimension de sortie entre 1, 3, 5 ou 7 fois la dimension d'entrée. Pour l'architecture Nyström, nous avons utilisé des sous-échantillons de taille 16, 64, 128, 256 ou 512. Nous observons que les deux types de couches Nyström considérées surpassent les architectures de référence, ce qui illustre l'intérêt de combiner les approximations de Nyström.

2.3.5 Analyse des Représentations apprises en deux dimensions

La Figure 2.4 présente les représentations bidimensionnelles obtenues par ϕ_{nys} sur des échantillons de test de CIFAR10. Ces représentations on été obtenues grâce à un sous-échantillon de taille égale à 2 (alors que le nombre de classes est de 10) et deux noyaux différents. On peut voir ici que les 10 classes sont déjà très bien séparées dans cet espace de représentation à faible dimension, illustrant qu'un sous-ensemble de très petite taille est déjà suffisamment expressif pour discriminer la plupart des exemples de classes différentes.

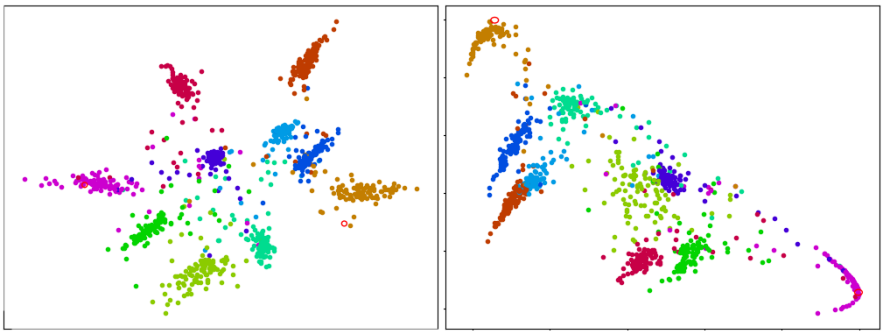


FIGURE 2.4 : Représentation en deux dimensions données par ϕ_{nys} sur 1000 exemples de test sélectionnés aléatoirement dans le jeu de données CIFAR10. Ces représentations ont été obtenues grâce à un ensemble d'observations de référence de taille 2 et en utilisant un noyau linéaire (à gauche) ou *Chi2* (à droite). Chaque couleur représente une classe différente.

2.4 Conclusion

Nous avons introduit des couches Nyström adaptatives qui peuvent être utilisées en remplacement des couches denses dans les réseaux neuronaux et forment ainsi une nouvelle architecture hybride qui mélange les réseaux profonds et les méthodes à noyaux. Cette architecture est basée sur l'approximation de Nyström qui permet de considérer n'importe quel type de fonction noyau. Nous avons illustré ceci sur des expériences faisant intervenir différents jeux de données d'images où différentes fonctions noyaux fonctionnaient plus ou moins bien. Une perspective possible de ce travail est d'étendre cette technique au traitement de données qui ne seraient pas forcément vectorielles, telles que les graphes comme dans le papier récent de CHEN, JACOB et MAIRAL, 2020. En outre, la méthode permet de traiter facilement plusieurs noyaux et de les combiner pour obtenir une représentation plus riche qui peut amener à de meilleures performances.

La méthode que nous proposons atteint les performances de précision des couches standard entièrement connectées tout en réduisant considérablement le nombre de paramètres. Nous remarquons toutefois que, même si le nombre de paramètres à apprendre dans le réseau est réduit, cette technique nécessite de calculer et stocker la représentation convoluée de l'ensemble des points de références. Ceci est décevant du point de vue de la quantité d'espace mémoire nécessaire au fonctionnement du réseau. Cependant, la réduction du nombre de paramètres à apprendre dans le réseau permet toutefois de fonctionner avec peu de données, comme nous le montrons dans la partie expérimentale. Cette caractéristique de notre approche offre plusieurs perspectives d'utilisation de la couche Nyström adaptative : l'apprentissage semi-supervisé où les données non-étiquetées seraient utilisées comme observations de référence dans $\mathbf{L}$; ou l'adap-

tation de domaine : les couches de convolution pourrait être apprise dans un domaine source bénéficiant de nombreuses données d'apprentissage alors que les couches supérieures dont la couche Nyström seraient apprises dans le domaine où moins de données sont disponibles.

3 Produits de matrices parcimonieuses pour la compression de modèles d'apprentissage

Contents

3.1 Introduction

Les transformations linéaires et les produits matriciels sont omniprésents en apprentissage automatique. Dans les réseaux de neurones, par exemple, la plupart des couches peuvent s'exprimer sous la forme d'un produit matriciel (voir Équation (1.27)). Idem pour les approximations de noyaux telles que les caractéristiques aléatoires (Équation (1.17)) qui nécessitent une étape de projection aléatoire ou l'approximation de Nyström avec certaines fonctions noyaux (voir Section 1.2.3.2) qui impliquent un produit avec la matrice des points de référence. Lorsque la dimension est élevée, les matrices à stocker pour réaliser ces transformations peuvent devenir colossales et, souvent, elles dominent la complexité des algorithmes comme c'est le cas dans nos travaux présentés en Chapitre 2.

Ce constat a alimenté l'intérêt de la communauté d'apprentissage automatique vers l'utilisation d'algorithmes de transformation rapides pour alléger les méthodes d'apprentissage. Contrairement aux matrices carrées habituelles qui nécessitent $\mathcal{O}(D^2)$ opérations pour être appliquées à un vecteur, certains opérateurs linéaires comme la transformée de Fourrier ou la transformée de Hadamard, dits «Transformées rapides», sont associés à des algorithmes capables de traiter leurs entrées en seulement $\mathcal{O}(Dlog(D))$ opérations. Ces accélérations viennent du caractère récursif de la définition de ces opérateurs qui permettent d'utiliser des algorithmes rapides de type «diviser pour régner» plutôt que l'algorithme standard de multiplication matricielle.

On remarque notamment que la transformée rapide de Hadamard a été étudiée pour réduire la complexité de méthodes à noyau. C'est le cas de la variante efficace de Nyström proposée par Si, Hsieh et Dhillon, 2016 (voir Sections 1.2.3.1 et 1.2.3.2) ou de la méthode `Fastfood` de Le, Sarlós et Smola, 2013. D'ailleurs, la méthode `Fastfood` a même voyagé jusqu'à l'apprentissage profond où elle a été utilisée en remplacement des couches denses d'un réseau

de neurones afin d'en réduire le nombre de paramètres (voir Section 1.3.4.2 ou le papier de YANG, MOCZULSKI, DENIL et al., 2015).

Toutefois, ces méthodes emploient des transformées rapides fixées à l'avance dont nous pensons que la structure pourrait avoir un effet délétère sur la performance des méthodes auxquelles elles sont appliquées. On peut donc se demander s'il ne serait pas possible d'apprendre la transformée rapide adaptée à chaque tâche afin de préserver au maximum la qualité de l'apprentissage ? Les travaux de MAGOAROU et GRIBONVAL, 2016 et DAO, GU, EICHHORN et al., 2019 suggèrent que oui. Ils s'appuient sur le constat que les algorithmes de transformation rapide peuvent être exactement décrits sous la forme d'un produit de matrices parcimonieuses pour proposer des algorithmes afin d'apprendre ou redécouvrir des transformées rapides exprimées de cette façon.

Nous commençons ce chapitre par une section d'état de l'art sur les méthodes capables de réaliser l'apprentissage de transformées rapides exprimées sous la forme de produits de matrices parcimonieuses. Ceci posera le cadre de travail dans lequel nous nous plaçons pour proposer, dans les sections suivantes, d'intégrer l'apprentissage de transformées rapides à des problèmes d'apprentissage classiques que sont le problème de regroupement en K moyennes et l'apprentissage de réseaux neuronaux convolutifs.

3.2 Apprentissage de Transformées rapides

La popularité de certains opérateurs linéaires de $\mathbb{R}^D$ dans $\mathbb{R}^D$ (avec $D < \infty$) comme les transformations de Fourier ou d'Hadamard provient de leurs propriétés mathématiques d'une part, mais aussi de leur capacité à être appliqués à une entrée $\mathbf{x} \in \mathbb{R}^D$ avec efficacité, grâce aux algorithmes de transformation rapide correspondants ; typiquement la version rapide de ces transformations se fait en $\mathcal{O}(D \log D)$ opérations au lieu de $\mathcal{O}(D^2)$.

Dans cette section introductive, nous résumons des travaux récents pour expliquer que les algorithmes de transformées rapides connus peuvent être écrits sous la forme de produits de matrices creuses et qu'en conséquence, elles peuvent être apprises pour mieux coller aux données.

3.2.1 Transformées rapides structurées comme des produits de matrices creuses

Une caractéristique intéressante des algorithmes de transformées rapides connus est qu'ils peuvent être écrits sous la forme du produit

$$\prod_{q=1}^{Q} \mathbf{S}_q \tag{3.1}$$

de $Q = \mathcal{O}(\log D)$ matrices creuses $\mathbf{S}_q$ avec $\|\mathbf{S}_q\|_0 = \mathcal{O}(D)$ coefficients non-nuls par facteur (MAGOAROU et GRIBONVAL, 2016; LI, YANG, MARTIN et al., 2015; VAN LOAN, 1992) . Pour tout vecteur $\mathbf{x} \in \mathbb{R}^D$, la transformation $\prod_{q=1}^{Q} \mathbf{S}_q \mathbf{x} = \mathbf{S}_1 (\mathbf{S}_2 (\cdots (\mathbf{S}_Q \mathbf{x})))$ peut donc être calculée avec $\mathcal{O}(\log D)$ produits entre matrice creuse et vecteur. Le coût de chaque produit étant de $\mathcal{O}(D)$, la complexité calculatoire totale est de $\mathcal{O}(D \log D)$. Nous étudions maintenant deux exemples présentés dans l'article de LI, YANG, MARTIN et al., 2015 qui montrent comment des transformées rapides peuvent être exprimées en produits de facteurs creux. D'autres exemples sont disponibles en annexe de l'article de LI, YANG, MARTIN et al., 2015.

Transformée de Fourier Rapide (FFT). La FFT est l'algorithme accéléré pour réaliser la Transformée de Fourier Discrète (DFT) et est peut-être l'exemple le plus populaire de transformée rapide du fait de sa grande utilité en traitement du signal. Soit $\mathbf{F}_D \in \mathbb{C}^{D \times D}$ la matrice permettant de calculer la DFT d'un vecteur de taille D. Celle-ci peut être écrite récursivement :

$$\mathbf{F}_D = \begin{bmatrix} \mathbf{I}_{D/2} & \mathbf{\Omega}_{D/2} \\ \mathbf{I}_{D/2} & -\mathbf{\Omega}_{D/2} \end{bmatrix} \begin{bmatrix} \mathbf{F}_{D/2} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{D/2} \end{bmatrix} \mathbf{P}_D, \tag{3.2}$$

où $\mathbf{I}_{D/2}$ est la matrice identité de taille $D/2$, $\mathbf{P}_D$ est une matrice de permutation de taille D qui regroupe les indices pairs entre eux et les indices impairs entre eux et enfin $\mathbf{\Omega}_{D/2}$ est une matrice diagonale de taille $D/2$ contenant les $D/2$ racines-$D/2^{\text{ème}}$ de l'unité, c'est à dire $1, \boldsymbol{\omega}^{-1}, \cdots, \boldsymbol{\omega}^{-(D/2-1)}$ où $\boldsymbol{\omega} := e^{2i\pi/D}$. En déroulant la récursion, on obtient l'expression de $\mathbf{F}_D$ en produit de facteurs creux :

$$\begin{aligned}
\mathbf{F}_D &= \mathbf{B}_D \begin{bmatrix} \mathbf{F}_{D/2} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{D/2} \end{bmatrix} \mathbf{P}_D \\
&= \mathbf{B}_D \begin{bmatrix} \mathbf{B}_{D/2} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{D/2} \end{bmatrix} \begin{bmatrix} \mathbf{F}_{D/4} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{D/4} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{F}_{D/4} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{F}_{D/4} \end{bmatrix} \begin{bmatrix} \mathbf{P}_{D/2} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_{D/2} \end{bmatrix} \mathbf{P}_D \\
&\vdots \\
&= \mathbf{B}_D \cdots \begin{bmatrix} \mathbf{B}_2 & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{B}_2 \end{bmatrix} \begin{bmatrix} \mathbf{P}_2 & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{P}_2 \end{bmatrix} \cdots \mathbf{P}_D,
\end{aligned} \tag{3.3}$$

où les blocs $\mathbf{B}_D, \cdots, \mathbf{B}_2$ sont appelés facteurs papillon (*butterfly*) et, pour tout d, ils sont de la forme :

$$\mathbf{B}_d = \begin{bmatrix} \mathbf{D}_1 & \mathbf{D}_2 \\ \mathbf{D}_3 & \mathbf{D}_4 \end{bmatrix}, \tag{3.4}$$

où les $\mathbf{D}_i$ sont des matrices diagonales. Ces facteurs papillon contiennent exactement deux valeurs non nulles par ligne et par colonne soit un total de $\mathcal{O}(D)$ coefficients chacun. Chaque matrice contenant des blocs papillon est appelée une matrice papillon ; des exemples de matrices papillon sont représentés en Figure 3.1. Ainsi, comme chaque étape de récursion divise par deux la taille des facteurs papillon, nous obtenons un total de $log(D)$ matrices papillon. Les matrices de permutation à droite peuvent être fusionnées en une seule permutation $\mathbf{P}$. La matrice de la DFT peut donc finalement être écrite sous la forme de l'Équation 3.1 où pour tout q, $\mathbf{S}_q$ est la $q^{ème}$ matrice papillon en partant de la gauche, excepté pour $q = log(D)$ où la matrice papillon est multipliée à droite par $\mathbf{P}$ ce qui ne change pas son nombre de coefficients non nuls. Finalement, la matrice de la DFT a bien été exprimée en un produit de matrice creuse dont l'application à un vecteur coûte $\mathcal{O}(DlogD)$ à la place de $\mathcal{O}(D^2)$, comme pour l'algorithme de la FFT.

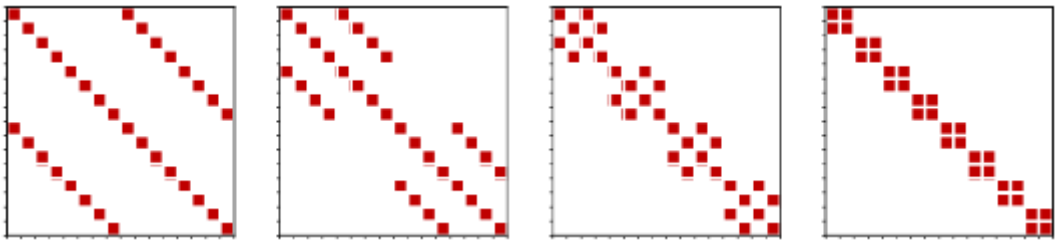


FIGURE 3.1 : Exemples de facteurs papillon. Les cases colorées correspondent aux valeurs non nulles de la matrice. On voit que dans chaque cas, un motif se répète sur les blocs de la diagonale : de gauche à droite, 1 fois, 2 fois, 4 fois et 8 fois. Source : LI, YANG, MARTIN et al., 2015

Transformée de Hadamard rapide. La matrice de Hadamard de taille D, $\mathbf{H}_D$, est définie récursivement par :

$$\mathbf{H}_1 = 1 \quad \text{et} \quad \mathbf{H}_D = \begin{bmatrix} \mathbf{H}_{D/2} & \mathbf{H}_{D/2} \\ \mathbf{H}_{D/2} & -\mathbf{H}_{D/2} \end{bmatrix} \tag{3.5}$$

donc on peut écrire :

$$\mathbf{H}_D = \begin{bmatrix} \mathbf{I}_{D/2} & \mathbf{I}_{D/2} \\ \mathbf{I}_{D/2} & -\mathbf{I}_{D/2} \end{bmatrix} \begin{bmatrix} \mathbf{H}_{D/2} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_{D/2} \end{bmatrix}. \tag{3.6}$$

Similairement à la FFT, en déroulant la récursion, on obtient :

$$\mathbf{H}_D = \mathbf{B}_D \cdots \begin{bmatrix} \mathbf{B}_2 & \cdots & \mathbf{0} \\ \vdots & \ldots & \vdots \\ \mathbf{0} & \cdots & \mathbf{B}_2 \end{bmatrix}, \tag{3.7}$$

où cette fois ci, les $logD$ facteurs papillon sont de la forme :

$$\mathbf{B}_{D/2^k} = \begin{bmatrix} \mathbf{I}_{D/2^{k+1}} & \mathbf{I}_{D/2^{k+1}} \\ \mathbf{I}_{D/2^{k+1}} & -\mathbf{I}_{D/2^{k+1}} \end{bmatrix}. \tag{3.8}$$

La matrice de Hadamard peut donc aussi être écrite sous la forme d'un produit de matrices creuses dont l'application à un vecteur se fait en temps $\mathcal{O}(DlogD)$ (illustré en Figure 3.2).

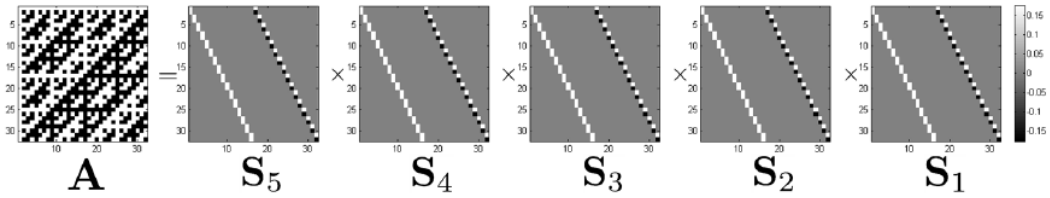


FIGURE 3.2 : Factorisation exacte de la matrice de Hadamard en un produit de facteurs parcimonieux. Source : MAGOAROU et GRIBONVAL, 2016

3.2.2 Apprentissage de produits de matrices creuses

Même si nous avons vu que les transformées rapides peuvent être utilisées dans certains algorithmes, il est raisonnable de penser que la structure imposée par les algorithmes de transformation rapide connus peut ne pas être adaptée à toutes les tâches d'apprentissage. On peut donc se demander s'il est possible d'apprendre des transformations rapides dépendantes des données. Cette question a été abordée par DAO, GU, EICHHORN et al., 2019, lorsqu'ils proposent de s'appuyer sur la structure hierarchique des transformations rapides pour proposer un algorithme qui apprend des factorisations dites "papillon" (LI, YANG, MARTIN et al., 2015) en fixant un ensemble de motifs de parcimonie possible (voir l'exemple de la FFT dans la section précédente). MAGOAROU et GRIBONVAL, 2016 ont une procédure moins contrainte permettant d'approcher n'importe quelle matrice par un produit de matrices creuses. Dans la suite, nous détaillons ces deux modèles.

Modèle papillon. Le modèle d'apprentissage de transformées rapides proposé par LI, YANG, MARTIN et al., 2015 s'appuie sur le fait que les transformées rapides connues peuvent s'écrire sous la forme d'un produit de matrices papillon suivies par une permutation, comme illustré dans les exemples de la Section 3.2.1 :

$$\mathbf{T} = \mathbf{B}_D \cdots \begin{bmatrix} \mathbf{B}_2 & \cdots & \mathbf{0} \\ \vdots & \ldots & \vdots \\ \mathbf{0} & \cdots & \mathbf{B}_2 \end{bmatrix} \mathbf{P}. \tag{3.9}$$

Dans ce modèle les paramètres à apprendre pour définir la transformée rapide sont donc les éléments des diagonales des facteurs papillon ainsi que la permutation finale. Pour contourner le problème combinatoire relatif à l'exploration de l'espace des permutations, les auteurs proposent de définir $\mathbf{P}$ comme :

$$\mathbf{P} = \prod_{s=a,b,c} (p_s \mathbf{P}^{(s)} + (1 - p_s)\mathbf{I}), \tag{3.10}$$

où les poids p_s sont à apprendre et les matrices de permutation $\mathbf{P}^{(s)}$ sont fixées à l'avance à partir de connaissances expertes sur les décompositions de transformations rapides connues. L'apprentissage des transformées rapides se fait par descente de gradient tel que, pour une transformée connue $\mathbf{T} \in \mathbb{R}^{D \times D}$, on résout le problème d'optimisation suivant :

$$\underset{\{\mathbf{B}_D, \cdots, \mathbf{B}_2\}, \{p_a, p_b, p_c\}}{\arg\min} \left\| \mathbf{T} - \left(\mathbf{B}_D \cdots \begin{bmatrix} \mathbf{B}_2 & \cdots & \mathbf{0} \\ \vdots & \ldots & \vdots \\ \mathbf{0} & \cdots & \mathbf{B}_2 \end{bmatrix} \mathbf{P} \right) \right\|_F + \sum_{d=0}^{logD} \delta(\mathbf{B}_{D/2^d}), \tag{3.11}$$

où $\delta(\mathbf{B}_{D/2^d}) = 0$ si $\mathbf{B}_{D/2^d}$ a bien une forme de facteur papillon telle que définie en Équation (3.4), sinon c'est égal à $\inf$. En d'autres terme δ encode la contrainte de parcimonie des blocs papillon. En pratique, la contrainte de structure en forme de matrice papillon est satisfaite par construction ; nous écrivons le terme de régularisation avec la fonction δ dans l'Équation (3.11) par soucis d'homogénéité.

Modèle `Palm4MSA` Le modèle papillon est grandement inspiré des décomposition des matrices des transformées rapides connues et laisse très peu de liberté au support de parcimonie des matrices creuses de la factorisation. Les travaux de MAGOAROU et GRIBONVAL, 2016 proposent un algorithme appelé `Palm4MSA` (*Palm for Multilayer Sparse Approximation*) qui permet l'apprentissage de transformées rapides avec très peu de contraintes imposées sur la structure des facteurs. Ils proposent un algorithme pour résoudre le problème d'optimisation suivant :

$$\min_{\lambda, \{\mathbf{S}_q\}_{q \in [\![Q]\!]}} \left\| \mathbf{T} - \lambda \prod_{q=1}^{Q} \mathbf{S}_q \right\|_F^2 + \sum_{q=1}^{D} \delta_{\mathcal{E}_q}(\mathbf{S}_q), \tag{3.12}$$

où λ est un paramètre d'échelle et pour chaque $q \in 1, \cdots, Q$, $\delta_{\mathcal{E}_q}(\mathbf{S}_q) = 0$ si

Algorithme 1 Algorithme `Palm4MSA`

Entrée: La matrice à factoriser $\mathbf{U} \in \mathbb{R}^{K \times D}$, le nombre de facteurs Q, les contraintes pour chaque facteur $\mathcal{E}_q$, $q \in [\![Q]\!]$ et le critère d'arrêt (ici, le nombre d'itérations I).

$\lambda \leftarrow ||S_1||_F$ $\quad \mathcal{O}(B)$

$S_1 \leftarrow \frac{1}{\lambda} S_1$ $\quad \mathcal{O}(B)$

pour $i \in 1 \dots I$ **faire**

  pour $q = Q$ down to 1 **faire**

    $\mathbf{L}_q \leftarrow \prod_{l=1}^{q-1} \mathbf{S}_l^{(i)}$

    $\mathbf{R}_q \leftarrow \prod_{l=q+1}^{Q} \mathbf{S}_l^{(i+1)}$

    Choisir $c > \lambda^2 ||\mathbf{R}_q||_2^2 ||\mathbf{L}_q||_2^2$ $\quad \mathcal{O}(A \log B + B)$

    $\mathbf{D} \leftarrow \mathbf{S}_q^i - \frac{1}{c} \lambda \mathbf{L}_q^T \left(\lambda \mathbf{L}_q \mathbf{S}_q^i \mathbf{R}_q - \mathbf{U} \right) \mathbf{R}_q^T$ $\quad \mathcal{O}(AB \log B)$

    $\mathbf{S}_q^{(i+1)} \leftarrow P_{\mathcal{E}_q}(\mathbf{D})$ $\quad \mathcal{O}(A^2 \log A)$ or $\mathcal{O}(AB \log B)$

  fin pour

  $\hat{\mathbf{U}} := \prod_{j=1}^{Q} \mathbf{S}_q^{(i+1)}$ $\quad \mathcal{O}(A^2 \log B + AB)$

  $\lambda \leftarrow \frac{Trace(\mathbf{U}^T \hat{\mathbf{U}})}{Trace(\hat{\mathbf{U}}^T \hat{\mathbf{U}})}$ $\quad \mathcal{O}(AB)$

fin pour

$S_1 \leftarrow \lambda S_1$ $\quad \mathcal{O}(B)$

Sortie: $\{\mathbf{S}_q : \mathbf{S}_q \in \mathcal{E}_q\}_{q \in [\![Q]\!]}$ tel que $\prod_{q \in [\![Q]\!]} \mathbf{S}_q \approx \mathbf{U}$

$\mathbf{S}_q \in \mathcal{E}_q$ et $\delta_{\mathcal{E}_q}(\mathbf{S}_q) = +\infty$ sinon. $\mathcal{E}_q$ est l'ensemble des éléments qui satisfont des contraintes d'échelle et de parcimonie. Leur algorithme (Algorithme 1) se base sur l'algorithme `Palm` (BOLTE, SABACH et TEBOULLE, 2014) pour apprendre conjointement les coefficients des matrices creuses ainsi que le support de parcimonie en utilisant une descente de gradient projeté et en alternant l'optimisation des différentes matrices $\mathbf{S}_q$ (la méthode de projection utilisée est détaillée en Annexe 3.A). Bien que ce problème soit non convexe et que le calcul d'un optimum global ne puisse pas être garanti, l'algorithme Palm4msa est capable de trouver des minima locaux avec des garanties de convergence. MAGOAROU et GRIBONVAL, 2016 vont plus loin que la simple procédure `Palm4MSA` et proposent un algorithme hierarchique appelé `Palm4MSA`-hierarchique où les facteurs parcimonieux $\mathbf{S}_q$ sont appris un par un et les contraintes de parcimonie sont endurcies au fil de l'apprentissage.

Le modèle `Palm4MSA`, quoique plus simple conceptuellement que le modèle papillon, explore un espace de solution beaucoup plus large, du fait de la liberté sur le support de parcimonie des matrices de la factorisation. En fait, il est assez simple de voir que l'espace des solutions atteignables par le modèle papillon fait aussi partie de l'espace des solutions atteignable par le modèle `Palm4MSA` : les $\mathcal{E}_q$ contiennent les matrices papillon. Il n'est pas clair cependant que le modèle `Palm4MSA` soit meilleur pour apprendre une transformée rapide que le modèle papillon et ceci nécessiterait *a minima* des experimentations empiriques. Toute-

fois, la liberté offerte par le modèle de MAGOAROU et GRIBONVAL, 2016 en a fait un candidat privilégié dans nos études décrites dans la suite de ce chapitre et qui visent à intégrer l'apprentissage de transformées rapides à des modèles d'apprentissage automatique existants.

3.3 Quick-means : Accélérer l'utilisation des K-moyennes via l'apprentissage de transformées rapides

3.3.1 Introduction

L'algorithme des K-moyennes – ou algorithme de Lloyd – est l'un des algorithmes de regroupement (*clustering*) les plus populaires (HARTIGAN et WONG, 1979 ; JAIN, 2010) et est largement utilisé dans de nombreuses applications, telles que l'indexation, la compression de données, la recherche du plus proche voisin et la détection de communautés dans les graphes (MUJA et LOWE, 2014 ; VAN LAARHOVEN et MARCHIORI, 2016). Combiné à l'approximation de Nyström, l'algorithme des K-moyennes s'avère également essentiel pour augmenter la vitesse et la précision de l'apprentissage avec les méthodes à noyaux (SI, HSIEH et DHILLON, 2016) ou les réseaux RBF (QUE et BELKIN, 2016). Dans toutes ces applications, les K centroïdes D-dimensionnels renvoyés par l'algorithme, $\mathbf{u}_1, \ldots, \mathbf{u}_K$, sont empilés les uns sur les autres pour former une matrice $\mathbf{U} \in \mathbb{R}^{K \times D}$ qui est employée comme un opérateur linéaire, que nous appelons l'opérateur *centroïde*. Dans le cas de l'algorithme des K-moyennes, les centroïdes sont en fait les moyennes de chaque groupe d'observation. Désormais, nous utiliserons les termes *d'apprentissage* ou *d'entraînement* pour désigner l'exécution de l'algorithme de regroupement, c'est-à-dire le processus qui permet d'extraire les centroïdes à partir des données d'entraînement, et nous utiliserons le terme *inférence* pour désigner l'application de l'opérateur centroïde à une ou plusieurs observations inconnues.

D'une part, la phase d'entraînement des K-moyennes est soumise à une complexité de $\mathcal{O}(NDK)$ par itération lorsque le nombre de points à regrouper est N. D'autre part, la phase d'inférence nécessite l'application de l'opérateur centroïde $\mathbf{U}$ au vecteur à traiter, ce qui implique une complexité $\mathcal{O}(KD)$. Nous verrons en Section 3.3.3 que de nombreux travaux ont visé à accélérer la phase d'apprentissage des K-moyennes mais aucun, à notre connaissance, ne propose d'accélération de la phase d'inférence. Pourtant, avec l'augmentation des flux de données que nous connaissons, c'est certainement cette partie de l'utilisation des K-moyennes qui est la plus gourmande car le nombre d'applications de la matrice des centroïdes aux nouvelles données d'entrées grandit infiniment avec le temps.

Résumé des contributions Après avoir présenté le fonctionnement de l'algorithme des K-moyennes standard dans la Section 3.3.2, nous introduisons `QK-means`, un algorithme qui étend les K-moyennes par l'apprentissage de la matrice des centroides sous la forme d'un produit de matrices parcimonieuses (Section 3.3.4.1). Nous montrons que chaque étape de mise à jour de notre algorithme réduit l'objectif global, ce qui établit la convergence de `QK-means` (Section 3.3.4.2). Nous fournissons ensuite une analyse de la complexité de `QK-means`, où nous montrons que la structure de transformée rapide de l'opérateur centroïde appris permet de présenter une complexité compétitive par rapport à celle de l'opérateur centroïde appris par la méthode des K-moyennes standard (Section 3.3.4.3). Enfin, nous réalisons une évaluation empirique de `QK-means` qui démontre l'efficacité de la méthode sur une variété de jeux de données dans le contexte de différentes tâches d'apprentissage qui s'appuient habituellement sur la méthode des K-moyennes (Section 3.3.5).

3.3.2 Le problème des K-moyennes

Le problème des K-moyennes vise à partitionner un ensemble $\mathbf{X} = [\mathbf{x}_i]_{i=1}^N$ de N vecteurs $\mathbf{x} \in \mathbb{R}^D$ en K *clusters* –des groupes de vecteurs– tels que la distance de chaque vecteur $\mathbf{x}$ au centroide de son *cluster* soit minimale, le centroïde optimal $\mathbf{U}_k$ du *cluster* k étant le vecteur moyen des points attribués à ce *cluster*. Le problème d'optimisation des K-moyennes est :

$$\arg\min_{\mathbf{U},\mathbf{t}} \sum_{k=1}^{K} \sum_{n:t_n=k} \|\mathbf{x}_n - \mathbf{u}_k\|^2, \tag{3.13}$$

où $\mathbf{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_K\}$ est l'ensemble des centroïdes et $\mathbf{t} \in [\![K]\!]^N$ est le vecteur qui assigne $\mathbf{x}_n$ au cluster k si $t_n = k$.

L'algorithme de Lloyd. La procédure la plus populaire pour résoudre approximativement le problème des K-moyennes est l'algorithme de Lloyd, tant et si bien qu'il est souvent directement appelé l'algorithme des K-moyennes, comme ici. C'est un algorithme itératif qui alterne entre : I) une étape d'affectation qui décide du *cluster* actuel auquel chaque point $\mathbf{x}_n$ appartient en fonction de sa distance aux *clusters* et II) une étape de ré-estimation qui ajuste les centroïdes des *clusters* en fonction des nouvelles affectations. Après une initialisation de l'ensemble des K centroïdes $\mathbf{U}^{(0)}$, l'algorithme procède comme suit. À l'itération τ, les affectations sont mises à jour telles que $\forall n \in [\![N]\!]$:

$$
\begin{aligned}
t_n^{(\tau)} &\leftarrow \arg\min_{k\in[\![K]\!]} \left\|\mathbf{x}_n - \mathbf{u}_k^{(\tau-1)}\right\|_2^2 \\
&= \arg\min_{k\in[\![K]\!]} \left\|\mathbf{u}_k^{(\tau-1)}\right\|_2^2 - 2\left\langle \mathbf{u}_k^{(\tau-1)}, \mathbf{x}_n \right\rangle \\
&= \arg\min_{k\in[\![K]\!]} \left\|\mathbf{u}_k^{(\tau-1)}\right\|_2^2 - 2\left[\mathbf{U}^{(\tau-1)}\mathbf{x}_n\right]_k ,
\end{aligned}
\tag{3.14}
$$

et les centroides de chaque *cluster* sont mis à jour tels que :

$$
\mathbf{u}_k^{(\tau)} \leftarrow \hat{\mathbf{x}}_k(\mathbf{t}^{(\tau)}) := \frac{1}{n_k^{(\tau)}} \sum_{n:t_n^{(\tau)}=k} \mathbf{x}_n, \quad \forall k \in [\![K]\!] , \tag{3.15}
$$

où $n_k^{(\tau)}$ est le nombre de points dans le cluster k au temps τ et $\hat{\mathbf{x}}_k(\mathbf{t})$ est le vecteur moyen des éléments du cluster k selon l'affectation $\mathbf{t}$.

Complexité de l'algorithme de Lloyd. Le coût de l'étape d'affectation (3.14) est de $\mathcal{O}(NDK)$ alors que celui de la mise à jour des centroïdes (3.15) est de $\mathcal{O}(ND)$. Par conséquent, le goulot d'étranglement de la complexité calculatoire d'entraînement $\mathcal{O}(NDK)$ provient de l'étape d'affectation. On note par ailleurs que la phase d'inférence des K-moyennes consiste à affecter un vecteur à l'un des clusters pré-calculés. De ce fait, l'étape d'inférence pour un vecteur fait intervenir aussi l'Équation (3.14) et a un coût de $\mathcal{O}(KD)$ opérations.

Notre principale contribution repose sur l'idée que l'assignation d'un vecteur à un cluster ((3.14)), et ainsi la phase d'inférence, peut être calculée plus efficacement si la matrice $\mathbf{U}$ des centroïdes est exprimée par une décomposition en facteurs parcimonieux, qui peut être apprise par l'une des procédures décrites en Section 3.2.2. Cette amélioration pourrait en théorie aussi accélérer la phase d'entraînement des K-moyennes mais ça n'a pas été observé en pratique.

3.3.3 Accélération du calcul des K-moyennes

Certaines techniques ont été proposées pour alléger la charge de calcul de l'algorithme de Lloyd's. Une approche populaire consiste à projeter les observations d'entrée dans un espace de dimension inférieure où l'on obtient un regroupement bon marché qui est ensuite utilisé dans l'espace de représentation initial pour y obtenir les centroïdes (Sculley, 2010 ; Boutsidis, Zouzias, Mahoney et al., 2014 ; Shen, Liu, Tsang et al., 2017 ; Liu, Shen et Tsang, 2017). Une autre idée est d'utiliser l'inégalité triangulaire afin de supprimer les mesures de distances redondantes tout en préservant la solution exacte de l'algorithme de LLoyd standard (Hamerly, 2010 ; Elkan, 2003). Enfin, un croquis (*sketch*) de l'ensemble complet des données d'entraînement peut être utilisé pour obtenir les

Algorithme 2 Algorithme `QK-means` avec annotations de complexité. $A{:=}\min(K,D)$ et $B{:=}\max(K,D)$

Entrée: $\mathbf{X} \in \mathbb{R}^{N\times D}$, K, initialisation des matrices creuses $\left\{\mathbf{S}_q^{(0)} : \mathbf{S}_q^{(0)} \in \mathcal{E}_q\right\}_{q\in[\![Q]\!]}$

1: Définir $\mathbf{V}^{(0)} : \mathbf{x} \mapsto \prod_{q\in[\![Q]\!]} \mathbf{S}_q^{(0)}\mathbf{x}$
2: **pour** $\tau = 1, 2, \ldots$ jusqu'à convergence **faire**
3: $\mathbf{t}^{(\tau)}{:=}\arg\min_{\mathbf{t}\in[\![K]\!]^N} \sum_{n\in[\![N]\!]} \left\|\mathbf{x}_n - \mathbf{v}_{t_n}^{(\tau-1)}\right\|^2$ $\quad \mathcal{O}\left(N\left(A\log B\ + B\right) + AB\right)$
4: $\forall k \in [\![K]\!]\,, \mathbf{u}_k^{(\tau)}{:=}\frac{1}{n_k}\sum_{n:t_n^{(\tau)}=k} \mathbf{x}_n$ with $n_k^{(\tau)}{:=}|\{n : t_n^{(\tau)} = k\}|$ $\quad \mathcal{O}(ND)$
5: $\mathbf{A}^{(\tau)}{:=}\mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}} \times \mathbf{U}^{(\tau)}$ $\quad \mathcal{O}(KD)$
6: $\mathcal{E}_0{:=}\left\{\mathbf{D}_{\sqrt{\mathbf{n}}}\right\}$
7: $\left\{\mathbf{S}_q^{(\tau)}\right\}_{q=0}^{Q} := \arg\min_{\{\mathbf{S}_q\}_{q=0}^{Q}} \left\|\mathbf{A}^{(\tau)} - \prod_{q=0}^{Q}\mathbf{S}_q\right\|_F^2 + \sum_{q=0}^{Q}\delta_{\mathcal{E}_q}(\mathbf{S}_q)$ $\quad \mathcal{O}(AB\left(\log^2 B\right))$
8: Définir $\mathbf{V}^{(\tau)} : \mathbf{x} \mapsto \prod_{q\in[\![Q]\!]} \mathbf{S}_q^{(\tau)}\mathbf{x}$ $\quad \mathcal{O}(1)$
9: **fin pour**

Sortie: vecteur d'assignation $\mathbf{t}$ et matrices creuses $\{\mathbf{S}_q : \mathbf{S}_q \in \mathcal{E}_q\}_{q\in[\![Q]\!]}$

K-moyennes d'un ensemble de données de taille immense à faible coût (KERIVEN, TREMBLAY, TRAONMILIN et al., 2017). Dans tous les cas, ces techniques se concentrent sur l'accélération de la *phase d'entraînement* des K-moyennes et non sur la *phase d'inférence* car l'opérateur centroïde résultant reste d'une complexité inchangée. La possibilité de réduire le coût de cette dernière phase a fait l'objet d'une attention assez réduite voire inexistante mais présente pourtant un intérêt majeur pour les applications du monde réel où le nombre d'observations en production est généralement illimité.

3.3.4 `Quick-means`

Nous présentons maintenant notre principale contribution, `QuicK-means` abrégé en `QK-means`. Nous commençons par décrire l'algorithme et le problème qu'il résout, puis nous démontrons sa convergence et fournissons une analyse de sa complexité calculatoire.

3.3.4.1 Encodage des K-moyennes en produits de matrices creuses grâce à l'algorithme `Quick-means`

`QK-means` est une variante des K-moyennes dans laquelle la matrice des centroïdes $\mathbf{U}$ est approchée par un produit $\prod_q^Q \mathbf{S}_q$ de matrices creuses $\mathbf{S}_q$.

Similairement au problème d'optimisation des K-moyennes (3.13) le problème d'optimisation de `QK-means` s'écrit :

$$\underset{\{\mathbf{S}_q\}_{q=1}^{Q},\mathbf{t}}{\arg\min}\; g\left(\{\mathbf{S}_q\}_{q=1}^{Q},\mathbf{t}\right), \tag{3.16}$$

où

$$\mathbf{g}\left(\{\mathbf{S}_q\}_{q=1}^{Q},\mathbf{t}\right) := \sum_{k}^{K}\sum_{n:t_n=k}\|\mathbf{x}_n-\mathbf{v}_k\|^2 + \sum_{q\in[\![Q]\!]}\delta_{\mathcal{E}_q}(\mathbf{S}_q) \tag{3.17}$$

et

$$\mathbf{V} = \prod_{q\in[\![Q]\!]}\mathbf{S}_q. \tag{3.18}$$

C'est une version contrainte du problème des K-moyennes (3.13) dans laquelle les centroïdes $\mathbf{v}_k$ sont contraints de former une matrice $\mathbf{V}$ avec une structure d'opérateur rapide, les fonctions indicatrices $\delta_{\mathcal{E}_q}$ imposant la parcimonie des matrices $\mathbf{S}_q$ (voir Section 3.2.2). Des détails sur les choix de modélisation tels que la dimension des facteurs $\mathbf{S}_q$ ou la procédure de projection sur le support de parcimonie sont donnés dans la Section 3.3.5.1.

Le problème (3.16) peut être résolu en utilisant l'Algorithme 2 qui procède de manière similaire à l'algorithme de Lloyd en alternant une étape d'assignation (ligne 3) et une étape de mise à jour des centroïdes (lignes 4–8). L'étape d'assignation peut être calculée efficacement grâce à la structure d'opérateur rapide de $\mathbf{V}$. En pratique, la mise à jour des centroides repose sur la procédure `Palm4MSA` pour résoudre le problème d'optimisation (ligne 7), détaillée en Section 3.2.2. Cette procédure construit l'opérateur rapide $\mathbf{V}$ approchant la véritable matrice des centroïde $\mathbf{U}$ dont les lignes sont pondérées par le nombre d'exemples n_k attribués à chaque *cluster* k. Notons, que même si ces travaux reposent sur l'algorithme `Palm4MSA`, nous pourrions en fait utiliser n'importe quel algorithme pour résoudre le problème d'approximation de la ligne 7.

On pourrait se demander pourquoi ne pas employer une stratégie plus simple qui consisterait à appliquer la procédure `Palm4MSA` directement sur l'opérateur centroïde appris par l'algorithme standard des K-moyennes ? cela produirait également des centroïdes avec une structure d'opérateur rapide. Cependant, en procédant ainsi, on ne trouverait qu'une approximation de la solution des K-moyennes exprimée comme un produit de matrices creuses et ne résoudrait pas nécessairement le problème de l'Équation (3.16). Dans la Section 3.3.4.2, nous offrons des garanties théoriques que la procédure `QK-means` converge effectivement vers un point stationnaire de la fonction objectif Équation (3.17). Ce postulat théorique est illustré expérimentalement Section 3.3.5.3 où nous montrons que la valeur objective obtenue par `QK-means` est meilleure que celle obtenue par `Palm4MSA` appliqué sur la solution des K-moyennes.

3.3.4.2 Convergence de l'algorithme

Comme pour l'algorithme des K-moyennes, `QK-means` converge localement.

Proposition 1. *Les versions successives de* $\left\{\mathbf{S}^{(\tau)}\right\}_{q\in[\![Q]\!]}$ *et* $\mathbf{t}^{(\tau)}$ *dans l'Algorithme 2 donnent,* $\forall \tau \geq 1$

$$\mathbf{g}\left(\left\{\mathbf{S}_q^{(\tau)}\right\}_{q=1}^{Q}, \mathbf{t}^{(\tau)}\right) \leq \mathbf{g}\left(\left\{\mathbf{S}_q^{(\tau-1)}\right\}_{q=1}^{Q}, \mathbf{t}^{(\tau-1)}\right),$$

ce qui implique que la procédure `QK-means` *fait converger la valeur de l'objectif* $\mathbf{g}$ *vers une valeur limite.*

Démonstration. Nous montrons que chaque étape de l'algorithme n'augmente pas la valeur de la fonction objective.

Étape d'assignation (Ligne 3). Pour un $\mathbf{V}^{(\tau-1)}$ fixé, le problème d'optimisation de la ligne 3 peut être traité séparément pour chaque exemple indexé par $n \in [\![N]\!]$. Le nouveau vecteur indicateur $\mathbf{t}^{(\tau)}$ est donc défini par :

$$t_n^{(\tau)} = \underset{k\in[\![K]\!]}{\arg\min} ||\mathbf{x}_n - \mathbf{v}_k^{(\tau-1)}||_2^2. \tag{3.19}$$

Cette étape minimise le premier terme de l'Équation (3.17) par rapport à $\mathbf{t}$ alors que le deuxième terme est constant, ce qui fait que nous avons

$$\mathbf{g}\left(\left\{\mathbf{S}_q^{(\tau-1)}\right\}_{q=1}^{Q}, \mathbf{t}^{(\tau)}\right) \leq \mathbf{g}\left(\left\{\mathbf{S}_q^{(\tau-1)}\right\}_{q=1}^{Q}, \mathbf{t}^{(\tau-1)}\right). \tag{3.20}$$

Étape de mise à jour des centroïdes (Lignes 4–8). Nous considérons maintenant un vecteur indicateur $\mathbf{t}^{(\tau)}$ fixe. Nous notons d'abord que pour tout vecteur $\mathbf{u}_k^{(\tau)}$ –le vrai centroïde du *cluster* k– et tout vecteur $\mathbf{v}_k^{(\tau)}$, ce qui suit est valide :

$$
\begin{aligned}
\sum_{n:t_n^{(\tau)}=k} ||\mathbf{x}_n - \mathbf{v}_k^{(\tau)}||^2 &= \sum_{n:t_n=k} ||\mathbf{x}_n - \mathbf{u}_k^{(\tau)} + \mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)}||^2 \\
&= \sum_{n:t_n^{(\tau)}=k} \left(||\mathbf{x}_n - \mathbf{u}_k^{(\tau)}||^2 + ||\mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)}||^2 - 2\left\langle \mathbf{x}_n - \mathbf{u}_k^{(\tau)}, \mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)} \right\rangle \right) \\
&= \sum_{n:t_n^{(\tau)}=k} ||\mathbf{x}_n - \mathbf{u}_k^{(\tau)}||^2 + n_k ||\mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)}||^2 - 2\left\langle \underbrace{\sum_{n:t_n^{(\tau)}=k} \left(\mathbf{x}_n - \mathbf{u}_k^{(\tau)}\right)}_{=0}, \mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)} \right\rangle \\
&= \sum_{n:t_n^{(\tau)}=k} ||\mathbf{x}_n - \mathbf{u}_k^{(\tau)}||^2 + ||\sqrt{n_k}\left(\mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)}\right)||^2.
\end{aligned}
\tag{3.21}
$$

Pour $\mathbf{t}^{(\tau)}$ fixé, les nouvelles matrices creuses $\left\{\mathbf{S}_q^{(\tau)}\right\}_{q=1}^{Q}$ qui définissent les centroides sont obtenues par la résolution du problème $\arg\min_{\mathbf{S}_1,\ldots,\mathbf{S}_Q} g(\mathbf{S}_1,\ldots,\mathbf{S}_Q,\mathbf{t}^{(\tau)})$ qui peut se réécrire, grâce à (3.21) :

$$
\begin{aligned}
\left\{\mathbf{S}_q^{(\tau)}\right\}_{q=1}^{Q} = \underset{\mathbf{S}_1,\ldots,\mathbf{S}_Q}{\arg\min} \left(||\mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}}(\mathbf{U}^{(\tau)} - \mathbf{V})||_F^2 + \sum_{k\in[\![K]\!]} c_k^{(\tau)} + \sum_{q\in[\![Q]\!]} \delta_q(\mathbf{S}_q) \right) \\
\text{s. t. } \mathbf{V} = \prod_{q\in[\![Q]\!]} \mathbf{S}_q
\end{aligned}
\tag{3.22}
$$

où :

- $\mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}}$ est la matrice diagonale de $\sqrt{\mathbf{n}^{(\tau)}}$ et $\forall k \in [\![K]\!]$, le $k^{\text{ème}}$ élément de $\mathbf{n}^{(\tau)}$ étant $n_k^{(\tau)} := \left|\left\{n : t_n^{(\tau)} = k\right\}\right|$, c'est-à-dire le nombre d'observations dans le cluster k à l'étape τ. D'où $\mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}}(\mathbf{U}^{(\tau)} - \mathbf{V})$ est la matrice avec $\sqrt{n_k^{(\tau)}}\left(\mathbf{u}_k^{(\tau)} - \mathbf{v}_k\right)$ sur sa $k^{\text{ème}}$ ligne ;
- $\mathbf{U}^{(\tau)} \in \mathbb{R}^{K\times d}$ se réfère à la matrice centroïde non contrainte obtenue à partir des données $\mathbf{X}$ et du vecteur indicateur $\mathbf{t}^{(\tau)}$: $\mathbf{u}_k := \frac{1}{n_k}\sum_{n:t_n=k} \mathbf{x}_n$ (voir Line 4) ;
- $c_k^{(\tau)} := \sum_{n:t_n^{(\tau)}=k} ||\mathbf{x}_n - \mathbf{u}_k^{(\tau)}||$ est constant par rapport à $\mathbf{S}_1,\ldots,\mathbf{S}_Q$.

Nous introduisons maintenant $\mathbf{A}^{(\tau)} := \mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}}\mathbf{U}^{(\tau)}$ qui est la matrice des centroïdes non contrainte mais pondérée par la taille de chaque *cluster* (voir Ligne 5), donc l'Équation (3.22) peut être simplifiée en :

$$
\left\{\mathbf{S}_q^{(\tau)}\right\}_{q=1}^{Q} = \underset{\mathbf{S}_1,\ldots,\mathbf{S}_Q}{\arg\min} ||\mathbf{A}^{(\tau)} - \prod_{q\in\{[\![Q]\!]\cup\{0\}\}} \mathbf{S}_q||_F^2 + \sum_{q\in\{[\![Q]\!]\cup\{0\}\}} \delta_q(\mathbf{S}_q), \tag{3.23}
$$

où le facteur supplémentaire $\mathbf{S}_0$ est contraint d'être $\mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}}$ en définissant $\mathcal{E}_0$ par le singleton $\{\mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}}\}$ à la ligne 6.

En démarrant le processus de minimisation aux estimations $\left\{\mathbf{S}_q^{(\tau-1)}\right\}_{q\in[\![Q]\!]}$ précédentes , on obtient

$$g(\mathbf{S}_1^{(\tau)},\ldots,\mathbf{S}_Q^{(\tau)},\mathbf{t}^{(\tau)}) \leq g(\mathbf{S}_1^{(\tau-1)},\ldots,\mathbf{S}_Q^{(\tau-1)},\mathbf{t}^{(\tau)}), \tag{3.24}$$

et en ré-assemblant l'Équation (3.24) avec l'Équation (3.20), nous avons enfin, pour toute itération τ :

$$g\left(\mathbf{S}_1^{(\tau)},\ldots,\mathbf{S}_Q^{(\tau)},\mathbf{t}^{(\tau)}\right) \leq g\left(\mathbf{S}_1^{(\tau-1)},\ldots,\mathbf{S}_Q^{(\tau-1)},\mathbf{t}^{(\tau)}\right) \leq g\left(\mathbf{S}_1^{(\tau-1)},\ldots,\mathbf{S}_Q^{(\tau-1)},\mathbf{t}^{(\tau-1)}\right)$$

ce qui est une condition suffisante pour affirmer que l'Algorithme 2 converge, c'est-à-dire que la séquence de valeurs objectives est non croissante et limitée par une limite inférieure (zéro, pour des raisons évidentes), donc convergente. Notez cependant que le développement précédent prouve que la valeur de la fonction objectif converge vers une valeur limite mais ne garantit pas que les centroïdes eux-mêmes convergent vers une position fixe. En fait, tout comme avec l'algorithme des K-moyennes, les centroïdes peuvent osciller pour toujours alors que la valeur objective reste inchangée.

□

3.3.4.3 Analyse de la complexité

Pour cette analyse de la complexité, nous fixons d'abord $A := \min(K, D)$ et $B := \max(K, D)$, et nous supposons que le nombre de facteurs est $Q = \mathcal{O}(\log B)$. L'analyse est proposée sous les hypothèses suivantes : le produit entre deux matrices denses de dimensions $N_1 \times N_2$ et $N_2 \times N_3$ peut être calculé en $\mathcal{O}(N_1 N_2 N_3)$ opérations ; le produit entre une matrice creuse avec $\mathcal{O}(S)$ entrées non nulles et un vecteur dense coûte $\mathcal{O}(S)$ opérations ; le produit entre deux matrices creuses de dimensions $N_1 \times N_2$ et $N_2 \times N_3$, ayant toutes deux $\mathcal{O}(S)$ valeurs non nulles coûte $\mathcal{O}(S \min(N_1, N_3))$ et le nombre d'entrées non nulles dans la matrice résultante est $\mathcal{O}(S^2)$.

Complexité de l'algorithme des K-moyennes. La complexité de l'algorithme est dominée par son étape d'assignation des *clusters*, nécessitant $\mathcal{O}(NKD) = \mathcal{O}(NAB)$ opérations (voir Équation (3.14)).

Complexité de l'algorithme `QK-means` en phase d'apprentissage. La complexité de chaque itération de `QK-means` est de $\mathcal{O}(N(A \log B + B) + AB \log^2 B)$.

Les complexités des principales étapes sont données dans Algorithme 2 et détaillées ici.

L'étape d'assignation (Ligne 3 et Équation (3.14)) bénéficie du calcul rapide de $\mathbf{VX}$ en $\mathcal{O}(N\,(A \log B\ + B))$ alors que le calcul des normes des centroïdes des *clusters* est en $\mathcal{O}(AB)$.

Le calcul de la mise à jour des centroïdes de chaque *cluster* à la Ligne 4 est le même que dans l'algorithme des K-moyennes et nécessite $\mathcal{O}(ND)$ opérations.

La mise à jour de la transformée rapide, aux Lignes 5 à 8 induit un surcoût calculatoire par rapport à l'algorithme des K-moyennes. Sa complexité en temps dépend de la procédure employée pour résoudre le problème d'optimisation de l'Équation (3.23) ; mais dans le cas où la procédure `Palm4MSA` est utilisée, la complexité de cette étape est dominée par la mise à jour des facteurs creux à la Ligne 7, en $\mathcal{O}(AB \log^2 B)$ opérations. Notez que, si A et B ont le même ordre de grandeur, le coût global de `QK-means` est dominé par le coût de l'étape d'affectation dès que le nombre d'exemples N est supérieur à $B \log^2 B$. Asymptotiquement, la *phase d'entraînement* de l'algorithme `QK-means` est plus efficace en termes de calcul que l'algorithme standard des K-moyennes. En pratique, ce gain ne pourrait être observé que lorsque les N, K et D sont très importants à cause de constantes multiplicatrices qui peuvent être assez conséquentes (nombre d'itérations de `Palm4MSA` notamment). Nous nous concentrons donc sur la *phase d'inférence* dans les expériences.

Complexité de l'algorithme `QK-means` en phase d'inférence L'inférence de `QK-means` consiste à appliquer l'opérateur rapide résultant de `QK-means` à une observation. Cela se fait en multipliant l'observation testée par la factorisation parcimonieuse des centroïdes, ce qui coûte $\mathcal{O}(A \log B)$ opérations au lieu des $\mathcal{O}(KD)$ attendues avec une matrice dense des centroides obtenus par l'algorithme des K-moyennes standard. Notez que cette complexité temporelle est directement liée à la complexité spatiale de l'opérateur `QK-means` qui est également de $\mathcal{O}(A \log B)$.

3.3.5 Résultats expérimentaux

Nous avons réalisé une série d'expériences pour démontrer l'efficacité de notre approche. Nous évaluons (I) le gain en espace –le nombre de valeurs non nulles dans la factorisation parcimonieuse – et le coût en calcul –le nombre d'Opérations en Virgule Flottante (FLOP)– lors de l'utilisation de cette factorisation parcimonieuse en inférence ; et (II) la qualité des centroïdes obtenus. Dans l'ensemble, ces expériences montrent que l'algorithme `QK-means` construit des centroïdes de qualité similaire à ceux obtenus avec l'algorithme des K-moyennes standard, mais avec un coût de calcul beaucoup plus faible. Notons que nous nous intéressons surtout au nombre de valeurs non nulles dans la matrice des centroïdes

et qu'en principe cette quantité ne peut pas être directement retranscrite en espace de stockage. Pour ceci, il faudrait aussi prendre en compte le stockage des indices des position de ces valeurs ainsi que la taille en bit de leur codage. Toutefois, pour simplifier l'analyse dans cette section, nous faisons l'amalgame entre complexité en stockage et nombre de valeurs non nulle à stocker.

Dans ce qui suit, nous commençons par donner des détails sur la mise en œuvre de notre méthode dans la Section 3.3.5.1 ; dans la Section 3.3.5.2, nous étudions l'influence de divers hyper-paramètres sur la fonction objectif ; dans la Section 3.3.5.3, la qualité des centroïdes estimés est illustrée ; dans la Section 3.3.5.4, nous illustrons les économies en espace et en calcul associées aux centroïdes de `QK-means` par rapport à ceux des K-moyennes standard et enfin, dans les Sections 3.3.5.5 et 3.3.5.6, nous montrons que cette version accélérée des centroïdes maintient des performances comparables dans leur utilisation ultérieure pour la recherche rapide du plus proche voisin ou l'approximation rapide de Nyström.

3.3.5.1 Paramètres expérimentaux

Données	Dimension D	# classes	Taille d'entraînement N	Taille de test N'
MNIST (Deng, 2012)	784	10	60 000	10 000
Fashion-MNIST (Xiao, Rasul et Vollgraf, 2017)	784	10	60 000	10 000
Caltech256 (Griffin, Holub et Perona, 2007)	2 352	256	19 952	9 828
Breast-cancer (Dua et Graff, 2017)	30	2	469	100
Coverage Type (Dua et Graff, 2017)	54	7	576 012	5 000
Coil20 (Nene, Nayar et Murase, 1996)	1024	10	964	476
Kddcup99 (Dua et Graff, 2017)	116	23	4 893 431	5 000

Table 3.1 : Main features of the datasets. Discrete, unordered attributes for dataset Kddcup99 have been encoded as one-hot attributes. For the Caltech and Coil dataset, the images have been resized to 32×32 images.

Méthodes concurrentes. Nous comparons essentiellement notre méthode à la version standard des K-moyennes car aucune autre méthode connue ne vise à accélérer la *phase d'inférence* des K-moyennes, comme expliqué dans la Section 3.3.3, et les résultats obtenus dans la littérature (Sculley, 2010 ; Boutsidis, Zouzias, Mahoney et al., 2014 ; Shen, Liu, Tsang et al., 2017 ; Liu, Shen et Tsang, 2017 ; Keriven, Tremblay, Traonmilin et al., 2017) montrent empiriquement que les K-moyennes standard génèrent les meilleurs résultats la plupart du temps, en terme de qualité des *clusters*.

Nous montrons également des résultats utilisant la méthode `K-means++` proposée par Arthur et Vassilvitskii, 2006 pour montrer que notre méthode bénéficie également de cette technique d'initialisation bien connue utilisée pour une convergence plus rapide et une meilleure qualité des *clusters*.

Details d'implémentation. Le code a été écrit en Python, y compris l'algorithme `Palm4MSA`.

Les opérateurs rapides $\mathbf{V}$ basés sur des matrices creuses $\mathbf{S}_q$ sont implémentés avec les objets `csr_matrix` du paquet `scipy.linalg`. Bien que des implémentations plus efficaces puissent être bénéfiques pour un déploiement plus important, notre implémentation (disponible en ligne (GIFFON et EMIYA, 2020)) est suffisante comme preuve de concept pour évaluer la performance de l'approche proposée grâce au décompte du nombre de FLOP dans la Section 3.3.5.4.

Datasets. Nos expériences sont menées sur sept ensembles de données de référence pour montrer la bonne qualité des centroïdes obtenus et la réduction relative du nombre de FLOP offerte par notre méthode lorsque le nombre de clusters et la dimensionnalité des données augmentent. Voir Table 3.1 pour plus de détails sur les jeux de données.

Configuration des algorithmes. Soit $B := \max(K, D)$ et $A := \min(K, D)$. `QK-means` est utilisé avec $Q := \log_2(B)$ facteurs creux $\mathbf{S}_q$ et tous les facteurs sont de dimension $A \times A$ sauf le plus à gauche, $\mathbf{S}_1$ de dimension $K \times A$, et le plus à droite, $\mathbf{S}_Q$ de dimension $A \times D$. La contrainte de parcimonie de chaque facteur $\mathbf{S}_q$ est régie par un paramètre global appelé *niveau de parcimonie*, qui indique le nombre souhaité de coefficients non nuls dans chaque ligne et dans chaque colonne de $\mathbf{S}_q$; l'impact de ce paramètre est examiné dans la Section 3.3.5.3. Étant donné que cette projection peut être coûteuse en terme de calcul, elle est assouplie dans la mise en œuvre de `Palm4MSA` et garantit uniquement que le nombre de coefficients non nuls dans chaque ligne et chaque colonne est au moins le niveau de parcimonie, comme dans MAGOAROU et GRIBONVAL, 2016. Le véritable nombre de coefficients non nuls dans les facteurs creux est mesuré à la fin du processus d'optimisation et est indiqué dans les résultats. Des détails sur la méthode de projection sont fournis en Annexe 3.A. Le critère d'arrêt pour les algorithmes des K-moyennes et `QK-means` consiste en un nombre maximum d'itérations fixé à 50, ce qui est un nombre suffisant pour atteindre la convergence du critère objectif dans tous les ensembles de données. Le critère d'arrêt de l'algorithme `Palm4MSA` consiste en un seuil de tolérance fixé à 10^{-6} sur la variation relative de la fonction objective et un nombre maximum d'itérations ; nous étudions également l'influence de ce nombre maximum d'itérations sur la fonction objective dans la Section 3.3.5.3. Chaque expérience a été reproduite 5 fois en utilisant des valeurs de graine différentes pour les générateurs aléatoires. Les techniques concurrentes partagent les mêmes valeurs de graine, et donc la même initialisation des centroïdes puisqu'ils sont échantillonnés à partir du jeu de données d'entraînement. Les facteurs creux utilisés pour l'initialisation de l'algorithme `QK-means` ont été obtenus en utilisant une fois la procédure `Palm4MSA`-hierarchique sur un sous-échantillon des données d'entrée ; la procé-

dure d'échantillonnage pour l'initialisation est examinée dans la Section 3.3.5.3. Palm4MSA-hierarchique est une heuristique pour Palm4MSA proposée par MAGOAROU et GRIBONVAL, 2016. Cette heuristique est plus coûteuse que Palm4MSA mais il a été démontré empiriquement qu'elle fournit de meilleurs résultats d'approximation. Elle est utilisée à la place de Palm4MSA pour l'initialisation mais pas dans QK-means car nous n'avons pas observé de réelle amélioration des performances dans ce cas. Pour la toute première initialisation de Palm4MSA, nous utilisons la même initialisation que celle proposée par MAGOAROU et GRIBONVAL, 2016, à savoir : tous les facteurs sont initialisés à l'identité, sauf le premier à être mis à jour, qui est entièrement initialisé à zéro.

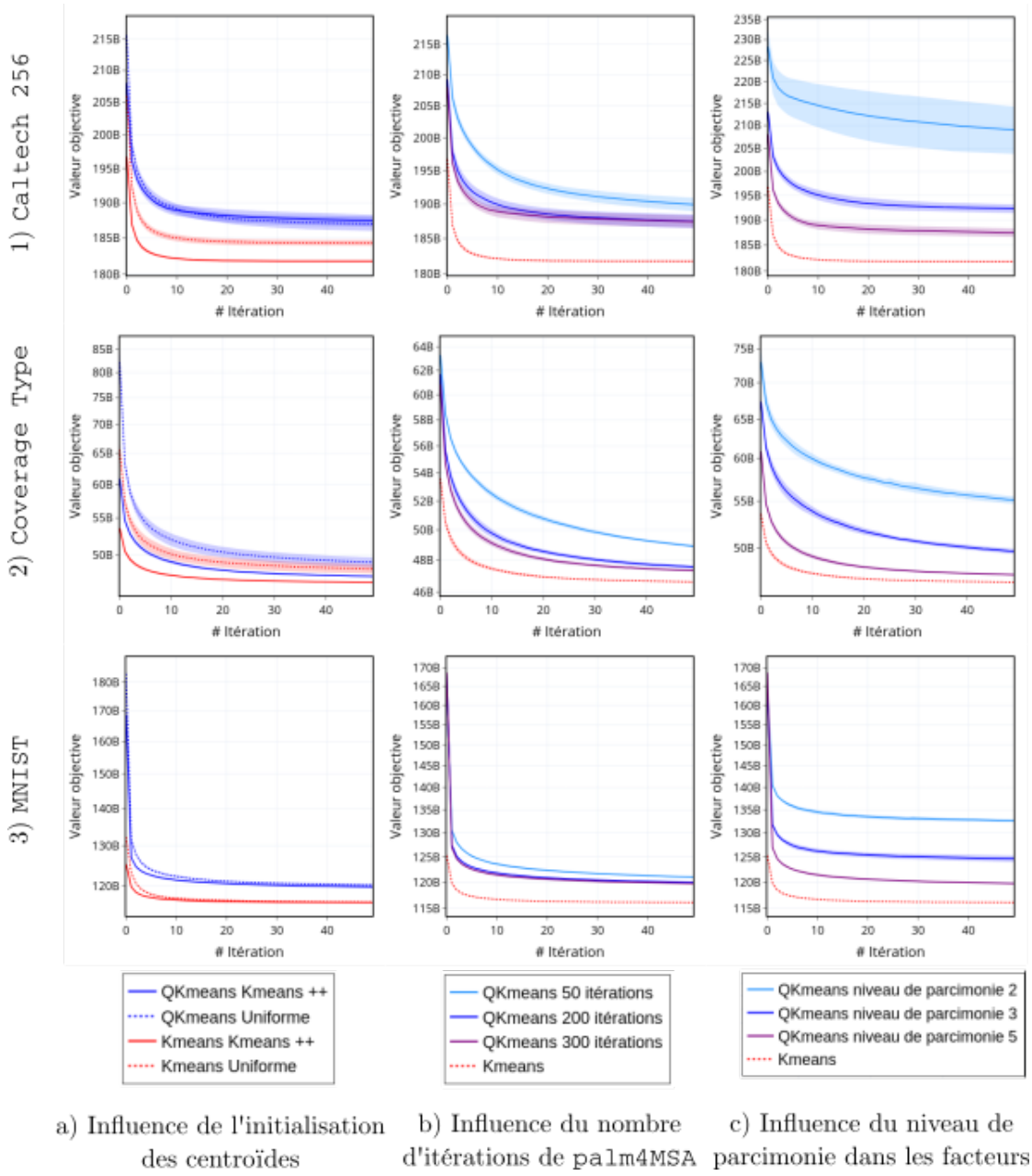
3.3.5.2 Influence des hyper-paramètres

Dans cette section, nous analysons l'influence de la procédure d'initialisation des centroïdes dans l'algorithme QK-means, du nombre d'itérations dans l'algorithme Palm4MSA et du niveau de parcimonie dans chaque matrice de la factorisation parcimonieuse.

Initialisation des centroides. Tout comme pour l'algorithme des K-moyennes, la solution fournie par QK-means n'est garantie de converger que vers un point stationnaire de la fonction objectif. Cela signifie que l'algorithme de QK-means est également sensible à une bonne initialisation. En pratique, nous montrons dans la colonne la plus à gauche de la Figure 3.3 que l'initialisation Kmeans++ (ARTHUR et VASSILVITSKII, 2006) offre le même avantage pour l'optimisation dans l'algorithme de QK-means que dans l'algorithme des K-moyennes, à savoir : une convergence plus rapide et une valeur plus basse de la fonction objectif. Nous utilisons donc la méthode d'initialisation kmeans++ pour le reste des expériences.

Nombre d'itérations dans Palm4MSA. L'algorithme Palm4MSA est un algorithme itératif qui s'accompagne de garanties de convergence, à condition qu'on lui laisse suffisamment d'itérations pour atteindre son minimum local. À cette fin, on peut choisir une valeur seuil pour la différence de fonction objectif entre deux itérations successives, mais il faut également choisir un nombre maximum d'itérations pour arrêter l'optimisation lorsqu'elle devient trop longue. Dans la colonne centrale de la Figure 3.3, nous montrons l'influence du nombre maximum d'itérations de Palm4MSA. Nous voyons qu'au-delà d'un nombre d'itérations donné, nous n'obtenons plus vraiment d'amélioration de la valeur de la fonction objectif et du rythme de convergence. Notons également que, parfois, la fonction objectif de QK-means peut subir des «sursauts» isolés que nous expliquons par la nature non monotone de l'objectif de Palm4MSA. Ceci peut conduire à un résultat plus mauvais que l'initialisation lorsque celle-ci est déjà proche du minimum local ; cela n'est pas vraiment visible dans les figures en raison de l'effet

de lissage généré par la moyenne entre les réplicats des expériences. Ce comportement a été observé le plus souvent lorsque le niveau de parcimonie et/ou le nombre d'itérations dans `Palm4MSA` étaient faibles. Nous utilisons 300 itérations pour `Palm4MSA` dans le reste des expériences, ce qui a empiriquement montré une réduction du nombre de ces «sursauts».



a) Influence de l'initialisation des centroïdes

b) Influence du nombre d'itérations de `palm4MSA`

c) Influence du niveau de parcimonie dans les facteurs

FIGURE 3.3 : Moyenne et écart-type (zone ombrée) de la valeur de la fonction objective en fonction du numéro d'itération dans diverses configurations de `QK-means`. Seuls trois jeux de données sont considérés ici, mais les mêmes tendances générales peuvent être observées sur tous les jeux de données.

Niveau de parcimonie. Nous appelons "niveau de parcimonie" le nombre approximatif de valeurs non nulles dans chaque ligne et chaque colonne des matrices creuses constituant l'opérateur des centroïde finaux. Nous montrerons plus

loin que le choix d'un niveau de parcimonie inférieur produit un taux de compression plus élevé des centroïdes (Figure 3.5) mais la contre-partie est une valeur plus mauvaise du critère objectif et un rythme de convergence plus lent. En effet, la colonne de droite de la Figure 3.3 montre que sur les trois ensembles de données considérés, plus le budget de parcimonie est petit, plus la valeur finale de la fonction objective est élevée.

3.3.5.3 Qualité des centroides.

Données	K-moyennes	`QK-means`	K-moyennes + `Palm4MSA`
Coverage Type	**4.663E+10**	<u>4.734E+10</u>	5.068E+10
Coil20	**1.332E+04**	<u>1.661E+04</u>	1.804E+04
Caltech256	**1.818E+11**	<u>1.875E+11</u>	1.953E+11
Fashion Mnist	**8.454E+10**	<u>8.946E+10</u>	9.224E+10
Breast Cancer	**5.579E+05**	<u>5.645E+05</u>	5.861E+05
Kddcup99	**6.107E+03**	<u>1.367E+04</u>	3.552E+04
Mnist	**1.161E+11**	<u>1.198E+11</u>	1.209E+11

TABLE 3.2 : Tableau des valeurs d'erreurs finales obtenues par l'algorithme des K-moyennes, `QK-means` ou `Palm4MSA` appliqué sur la solution des K-moyennes. Les résultats ont été obtenus avec un niveau de parcimonie de 5. Les meilleurs résultats sont en gras, les seconds sont soulignés.

(a) Centroïdes obtenus avec les K-moyennes.

(b) Centroïdes obtenus avec `QK-means`.

FIGURE 3.4 : Représentation visuelle des centroïdes obtenus avec les K-moyennes (à gauche) et `QK-means` (à droite) sur le jeu de données MNIST pour $K = 30$ *clusters*. Les images ont été obtenues avec un niveau de parcimonie égal à 5.

Une question importante est de savoir si le modèle que nous proposons est capable à s'adapter à des données arbitraires. Afin d'évaluer la qualité des *clusters* de `QK-means`, nous commençons par comparer le critère d'erreur obtenu avec `QK-means` par rapport a l'algorithme des K-moyennes standard ou encore la stratégie visant à appliquer `Palm4MSA` sur la solution obtenue par l'algorithme des

	Algorithme	MNIST D=784 K=64	F.-MNIST D=784 K=64	B.-cancer D=30 K=64	Covtype D=54 K=256	Coil20 D=1024 K=64	Kddcup99 D=116 K=256	Caltech D=2352 K=256
# FLOP	K-moyennes	100 352	100 352	3 840	27 648	131 072	59 392	1 572 864
	`QK-means`	**9 581**	**9 181**	**1 809**	**5 254**	**11 170**	**8 117**	**48 174**
# non-zero values	K-moyennes	50 176	50 176	1 920	13 824	65 536	29 696	786 432
	`QK-means`	**4 790**	**4 590**	**904**	**2 627**	**5 585**	**4 058**	**24 087**
Compression rate	$\frac{K\text{-moyennes}}{\texttt{QK-means}}$	×10.5	×10.9	×2.1	×5.3	×11.7	×7.3	×32.7
1-NN Accuracy	K-moyennes	0.9507	0.8353	**0.916**	0.9669	0.9794	**0.9992**	**0.1065**
	`QK-means`	0.9514	0.8353	0.914	**0.9673**	0.9782	**0.9993**	0.1007
	Ball-tree	**0.9690**	**0.8497**	0.9280	*N/A*	**0.9895**	*N/A*	*N/A*
Nyström approximation error	K-moyennes	**0.0322**	**0.0191**	0.0001	0.0001	**0.0218**	**0.0003**	**0.0138**
	`QK-means`	0.0427	0.0299	$\mathbf{1E-05}$	$\mathbf{3E-05}$	0.0343	0.0008	0.0259
	Uniform	0.0673	0.0443	0.005	0.0002	0.0541	0.0051	0.0194
	Un. F-Nys.	0.1702	0.2919	0.0449	0.0582	0.2501	0.1509	0.2191
	K. F-Nys.	0.1576	0.2623	0.0598	0.0381	0.2371	0.1275	0.2382
Nyström + SVM Accuracy	K-moyennes	**0.9235**	**0.8185**	0.914	0.6830	0.9702	**0.9989**	**0.1694**
	`QK-means`	0.9200	0.8119	0.914	**0.6848**	**0.9798**	0.9982	0.1588
	Uniform	0.905	0.8142	**0.9340**	0.6818	0.9546	0.9972	0.1575
	Un. F-Nys.	0.7937	0.7341	0.932	0.5936	0.7399	0.9944	0.0954
	K. F-Nys.	0.7337	0.6872	0.93	0.6061	0.6756	0.9948	0.0751
AMI	K-moyennes	**0.5523**	**0.4883**	**0.1790**	**0.1040**	0.7093	0.5800	**0.0989**
	`QK-means`	0.5337	0.4769	0.1772	0.1010	**0.7124**	**0.6637**	0.0841

TABLE 3.3 : Résultats des expériences numériques : FLOP et nombre de paramètres ; erreur moyenne de la transformation de Nyström pour un ensemble d'échantillons de taille 5000 ; précision en classification test de l'algorithme du plus proche voisin (1-NN) et de la Machine à Vecteurs Support (SVM) sur des données transformées par l'approximation de Nyström. «Un. F-Nys» et «K. F-nys» représentent l'agorithme de Nyström rapide (SI, HSIEH et DHILLON, 2016) avec des méthodes d'échantillonnage uniforme et K-moyennes respectivement. Les résultats de `QK-means` sont obtenus avec des facteurs parcimonieux avec un niveau de rareté de 3. Les meilleurs résultats sont en gras tandis que les seconds sont soulignés (si nécessaire). Les verions «Brute» et «kd-tree» du plus proche voisin sont omises car ils obtiennent toujours de moins bons résultats que «Ball-tree». Seuls les résultats avec le plus grand nombre de *clusters* sur chaque jeu de données sont affichés.

K-moyennes (dénoté K-moyennes + `Palm4MSA`). Dans le Tableau 3.2, nous remarquons que dans tous les ensembles de données considérés, la valeur objective obtenue avec `QK-means` est toujours meilleure que celle obtenue avec K-moyennes + `Palm4MSA`. Notez que, pour une comparaison équitable, nous avons utilisé la version hiérarchique de `Palm4MSA` dans le régime K-moyennes + `Palm4MSA`, tout comme dans la procédure d'initialisation de `QK-means`. Ces expériences illus-

trent l'affirmation, dans la Section 3.3.4.1, qu'appliquer `Palm4MSA` sur la solution fournie par les K-moyennes revient simplement à approximer la solution des K-moyennes par un produit de facteurs creux alors que `QK-means` minimise en fait directement l'objectif de l'Équation (3.17).

Une mesure habituelle de la qualité de *clusters* lorsque les étiquettes des données sont disponibles est l'information mutuelle partagée entre les étiquettes réelles des données et les *clusters*. Les deux dernières lignes du tableau 3.3 montrent l'Information Mutuelle Ajustée (AMI) (Vinh, Epps et Bailey, 2010) entre les étiquettes réelles des ensembles de données et les clusters donnés par l'algorithme des K-moyennes et `QK-means`. Du point de vue de cette mesure les *clusters* donnés par `QK-means` sont de qualité comparable à celle des K-moyennes, parfois même meilleure.

La qualité de l'approximation des centroïdes peut également être évaluée visuellement, d'une manière plus subjective et interprétable, dans la Figure 3.4 où les centroïdes obtenus sur le jeu de données `MNIST` sont affichés sous forme d'images. Bien que certaines dégradations puissent être observées sur certaines instances, on peut noter que chaque image obtenue avec `QK-means` représente clairement un seul élément visuel sans interférence notable avec d'autres éléments.

3.3.5.4 Performance en compression

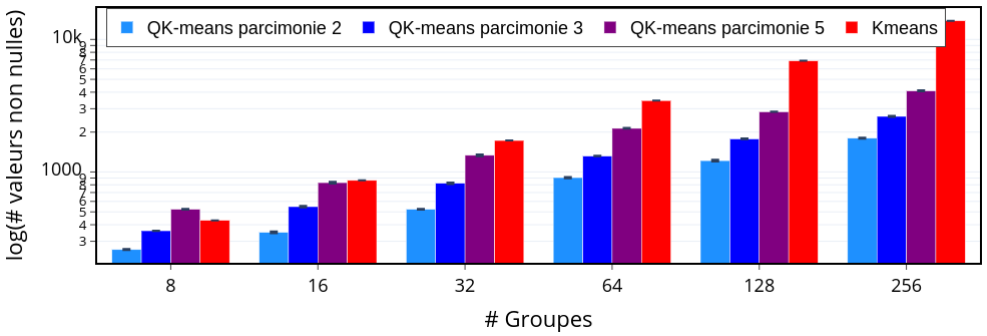


Figure 3.5 : Nombre de FLOP en fonction du nombre de *cluster* et du niveau de parcimonie sur le jeu de données *Coverage type*. On remarque que l'allure de cette Figure serait exactement identique si nous avions choisi de représenter le nombre de valeur non nulles plutôt que le nombre de FLOP étant donné que la dépendance entre les deux mesures est linéaire.

Nous montrons que l'opérateur des centroïdes obtenu par `QK-means` a un coût en espace et en calcul inférieur à celui des K-moyennes.

Comme les temps d'exécution dépendent grandement de l'implémentation et ne sont pas reproductibles, nous indiquons dans le tableau 3.3 le nombre

de FLOP requis pour que les opérateurs des centroïdes de `QK-means` et des K-moyennes soient appliqués à un vecteur donné, ainsi que le nombre correspondant de valeurs non nulles dans ces opérateurs.

Nous montrons également le taux de compression qui est donné par le ratio K-moyennes : `QK-means`. On peut noter que le taux de compression est le même pour le nombre de FLOP et le nombre de valeurs non nulles car le nombre de FLOP est proportionnel au nombre de valeurs non nulles (deux fois plus élevé). Ces résultats illustrent clairement l'avantage en complexité de la méthode `QK-means` : plus l'échelle est grande, plus le taux de compression est élevé, comme avec l'ensemble de données Caltech256 où le taux de compression atteint $\times 32,7$. On peut remarquer que, pour la méthode `QK-means`, le nombre de valeurs non nulles et de FLOP peut varier pour deux ensembles de données avec les mêmes D et K comme avec MNIST et Fashion-MNIST. Ce comportement est causé par notre procédure de projection qui ne garantit pas un nombre fixe de valeurs non nulles dans chaque facteur projeté (voir Annexe 3.A).

La Figure 3.5 montre comment le nombre de *clusters* a un impact relativement faible sur le nombre de FLOP induites par l'opérateur factorisé par rapport à celui de la matrice dense standard générée par l'algorithme des K-moyennes. Comme prévu, nous voyons également que l'augmentation du niveau de parcimonie augmente également le nombre de FLOP. Cette figure montre les résultats sur le jeu de données *Coverage Type* mais les mêmes schémas de compression peuvent être observés pour tous les jeux de données.

3.3.5.5 Recherche des plus proches voisins

Nous montrons à présent que l'opérateur des centroïdes de `QK-means` est de qualité comparable à l'opérateur obtenu par l'algorithme des K-moyennes pour la recherche du plus proche voisin (1-NN).

La recherche du plus proche voisin est une tâche fondamentale qui souffre de limitations calculatoires lorsque le jeu de données est de taille importante. En effet, c'est une méthode d'apprentissage qui nécessite en inférence de calculer la distance de l'observation à traiter avec tous les exemples d'apprentissages, ce qui est rédhibitoire lorsque la taille de l'ensemble d'entraînement est très grande. Des stratégies rapides ont été proposées, par exemple en utilisant des «arbres KD» (*kd-tree*) (BENTLEY, 1975) ou des «arbres boule» (OMOHUNDRO, 1989) (*ball-tree*) qui sont des structures de données utilisées pour partitionner un espace en grande dimension afin d'y organiser des points et faciliter leur recherche. On peut également utiliser une stratégie de partitionnement basée sur des *clusters* de l'ensemble des données d'entraînement pour effectuer une recherche approximative du plus proche voisin : l'exemple à classifier est d'abord comparé aux K centroïdes calculés au préalable, ensuite la recherche du plus proche voisin est effectuée parmi un nombre inférieur d'observations, au sein de la partition de données correspondante. Nous comparons cette stratégie uti-

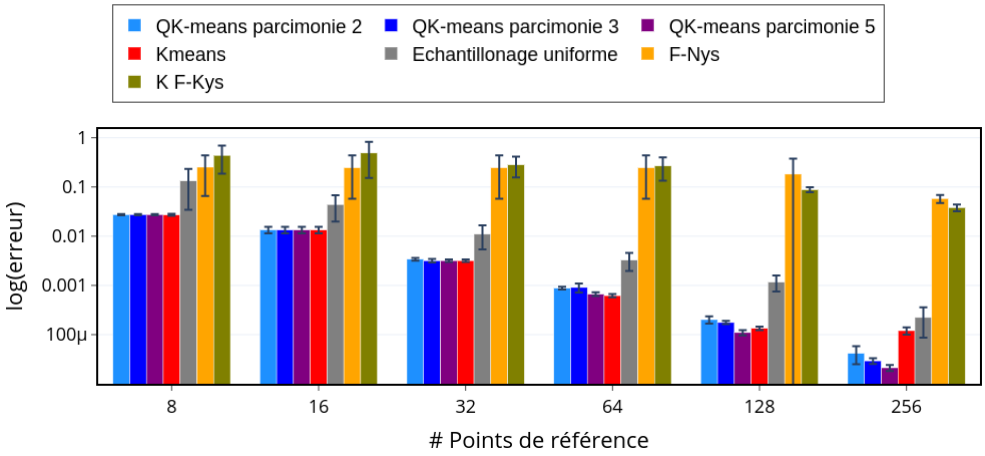
lisant les K-moyennes ou `QK-means` à l'implémentation `scikit-learn` (PEDREGOSA, VAROQUAUX, GRAMFORT et al., 2011) de la recherche du plus proche voisin (recherche par force brute, «*kd-tree*», «*ball-tree*»). Les résultats en précision de classification sont affichés dans le Tableau 3.3. Les méthodes par force brute et «*kd-tree*» sont regroupées sous le nom de «*Ball-tree*» qui a été aussi bon en terme de qualité mais plus rapide que les deux autres. Les résultats de la recherche par *Ball-tree* ne sont pas disponibles pour certains ensembles de données (N/A) parce qu'ils étaient plus longs que 10 fois la version de recherche basée sur les K-moyennes. Nous notons que cela se produit sur les plus grands ensembles de données, ce qui souligne l'utilité de l'utilisation de l'opérateur centroïde pour réaliser la recherche sur ces jeux de données. Nous constatons que la performance de prédiction pour la classification n'est pas trop affectée lorsque l'on utilise la factorisation obtenue par `QK-means` au lieu de la matrice centroïde obtenue par l'algorithme des K-moyennes pour le partitionnement. Lorsque les résultats sont disponibles, nous constatons également qu'en utilisant l'algorithme 1-NN partitionné par les K-moyennes, les performances ne sont pas trop diminuées par rapport aux algorithmes 1-NN classiques.

3.3.5.6 Approximation de Nyström efficiente

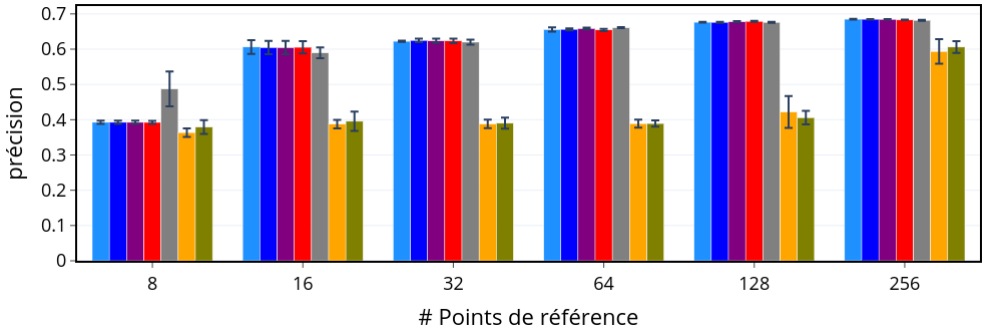
Les centroïdes donnés par `QK-means` sont de bons points de repère pour une approximation de Nyström précise et efficace. Dans cette section, nous montrons comment nous pouvons tirer parti de l'opérateur rapide obtenu en sortie de notre algorithme `QK-means` afin d'alléger le calcul dans l'approximation de Nyström. Pour se faire, nous nous appuyons sur les travaux de SI, HSIEH et DHILLON, 2016 visant à intégrer les transformées rapides au calcul de l'approximation de Nyström (décrits dans la Section 1.2.3.2 de cette thèse).

Nous proposons d'utiliser notre algorithme `QK-means` afin d'apprendre directement la matrice des points de référence $\mathbf{L}$ dans l'approximation de Nyström, de sorte que la multiplication matrice-vecteur $\mathbf{Lx}$ soit peu coûteuse à calculer sans que la structure de $\mathbf{L}$ ne soit contrainte par une matrice de transformation prédéfinie, ce qui peut se traduire par de meilleures performances en pratique. Nous utilisons le noyau RBF ($k(\mathbf{x}, \mathbf{y}) = \exp(-\gamma||\mathbf{x} - \mathbf{y}||^2)$) parce qu'il s'agit de la fonction noyau la plus populaire et SI, HSIEH et DHILLON, 2016 montrent que cette fonction noyau est appropriée pour la technique de Nyström efficace.

Le Tableau 3.3 résume les résultats obtenus dans le cadre de l'approximation de Nyström. Nous voyons que notre opérateur rapide conserve les informations pertinentes pour que l'approximation de Nyström soit aussi précise qu'avec des points de références issus des K-moyennes. Pour cette évaluation, nous considérons l'erreur de reconstruction de la matrice $\tilde{\mathbf{K}}$ de l'approximation de Nyström avec différentes méthodes d'échantillonnage (`QK-means`, K-moyennes, échantillonage uniforme) par rapport à la matrice noyau réelle $\mathbf{K}$. Cette erreur est calculée à partir de la norme de Froebenius de la différence entre les matrices :



(a) Erreur de reconstruction



(b) Précision du SVM

FIGURE 3.6 : `Coverage Type` : Résultats de la méthode de Nyström en terme d'erreur de reconstruction de la matrice noyau (Fig. 3.6a) et de la précision du SVM avec les vecteurs d'entrée transformés en utilisant l'approximation de Nyström (Fig. 3.6b).

$erreur = \frac{||\mathbf{K}-\tilde{\mathbf{K}}||_F}{||\mathbf{K}||_F}$. On peut souligner que la méthode d'échantillonnage uniforme des points de références est connues pour donner une approximation de Nyström de moins bonne qualité que la méthode des K-moyennes (KUMAR, MOHRI et TALWALKAR, 2012) et ce comportement est toujours observable avec `QK-means`. Nous utilisons également l'approximation de Nyström basée sur `QK-means` comme entrée pour un SVM linéaire, qui obtient d'aussi bonnes performances que celui basé sur l'approche des K-moyennes. Nous montrons également dans le Tableau 3.3 et sur la Figure 3.6 que, pour un nombre fixe de points de repère, notre technique surpasse nettement la technique du Nyström efficace (SI, HSIEH et DHILLON, 2016) en termes d'erreur d'approximation et de précision des prédictions. Pour cette technique, nous avons utilisé la transformée de Hadamard pour $\mathbf{H}$ dans l'Équation (1.25) et les diagonales $\mathbf{V}_i$ (Équation (1.25)) ont été obtenues à partir d'un échantillonnage uniforme ou un échantillonnage par les K-moyennes,

respectivement appelés «Un. F-Nys» et «K. F-Nys». Ces résultats illustrent notre affirmation selon laquelle il est avantageux en pratique d'apprendre une transformation rapide qui s'adapte aux données au lieu d'utiliser un algorithme de transformation rapide fixe avec un fort biais structurel comme la transformée de Hadamard. La Figure 3.6 montre enfin que le niveau de parcimonie semble avoir un impact assez limité sur la qualité de l'approximation de Nyström et la précision en classification, alors qu'une augmentation du nombre de clusters entraîne de meilleurs résultats, comme attendu.

3.3.6 Ouverture

Nous avons proposé un nouvel algorithme de regroupement appelé `QK-means`. Notre algorithme étend l'algorithme standard des K-moyennes et permet d'obtenir une matrice de centroïde exprimée sous la forme d'un produit de matrices parcimonieuses. Exprimer la matrice des centroïdes de cette façon permet d'accélérer ce que nous appelons la phase «d'inférence» des K-moyennes, à savoir son utilisation dans des applications subséquentes, telles que l'approximation de fonctions noyaux ou la recherche des plus proches voisins dans un ensemble d'apprentissage. On note toutefois que cette accélération a été mesurée en terme de FLOP et non en terme de temps d'exécution effectif car notre implémentation des produits de matrices parcimonieuses ne se révélaient pas suffisamment efficaces pour faire apparaitre un bénéfice de temps avec les dimensions de données, les bibliothèques de calcul et les architectures matérielles utilisées.

En théorie, notre algorithme devrait aussi accélérer la phase d'entraînement des K-moyennes lorsque la taille du jeu de données devient suffisamment grande. Cependant, nous n'avons pas pu illustrer ce comportement en pratique. Une perspective possible de ces travaux est donc de vérifier cette affirmation en comptant le nombre d'opérations au cours de l'apprentissage. De plus, notre méthode n'est pas fondamentalement incompatible avec les heuristiques d'accélération de la phase d'entraînement des K-moyennes basées sur l'inégalité triangulaire ELKAN, 2003 pour éviter les calculs de distances redondants à chaque itération. Enfin, la méthode `Palm4MSA` est *une* méthode pour résoudre la problème d'approximation d'une matrice en un produit de matrices parcimonieuses. Cette méthode est relativement lente et donc nous imaginons qu'en utiliser une autre (à définir), plus rapide, pourrait aussi accélérer la phase d'entraînement des K-moyennes.

3.4 Produits de matrices creuses pour la compression de réseaux de neurones

3.4.1 Introduction

Les réseaux de neurones ont démontré leur suprématie sur de nombreux problèmes complexes relatifs, notamment, au traitement de données structurées telles que les images ou le langage naturel. Leur succès est dû (I) à leur capacité à apprendre à partir d'énormes bases de données avec des millions d'échantillons, (II) à leur sur-paramétrage qui joue un rôle dans leur capacité de généralisation, et (III) à leur capacité à tirer profit de la puissance de parallélisation des cartes graphiques (GPU) modernes pour l'apprentissage.

Cette combinaison rend la plupart des modèles de pointe si énormes qu'ils sont coûteux à stocker et à faire fonctionner, et particulièrement peu pratiques à déployer sur des appareils aux ressources limitées (mémoire, capacités de calcul) ou sur des appareils qui ne peuvent pas intégrer de carte graphique (GPU). Le paramétrage excessif des réseaux neuronaux est au cœur d'un axe de recherche populaire appelé «compression des réseaux neuronaux» qui vise à construire des modèles avec moins de paramètres tout en préservant leur qualité.

Nous intéressons ici à la compression des couches denses et des couches convolutives des réseaux neuronaux via la factorisation des matrices de poids de ces couches en produits de matrices parcimonieuses (ou creuses). La motivation derrière cette idée est double : d'une part, un produit de matrices creuses est plus expressif et peut transmettre plus d'informations qu'une seule matrice creuse ; d'autre part, le produit de matrices creuses peut produire une matrice de rang complet par opposition aux méthodes de factorisation de rang inférieur existantes. L'utilisation d'une factorisation de matrices parcimonieuses pour stocker les poids des couches dans un réseau neuronal apparaît donc comme un bon compromis entre les méthodes d'induction de parcimonie et les méthodes de factorisation à faible rang pour la compression de réseaux neuronaux (voir Section 1.3.4).

D'une certaine façon, les `Deep Fried Convnets` de YANG, MOCZULSKI, DENIL et al., 2015 abordés en Section 1.3.4.2 implémentent déjà cette idée de remplacer les couches par des factorisations parcimonieuses. En effet, l'approximation de `Fastfood` consiste en un produit de matrices diagonales (donc creuses), d'une matrice de permutation (creuse aussi) et de la matrice de Hadamard qui peut elle-même être décomposée en un produit de $\log D$ matrices parcimonieuses ou D est la dimension d'entrée de la transformation(voir Section 3.2.1). Toutefois, les `Deep Fried Convnets` ne proposent que de remplacer les couches denses des réseaux convolutifs par cette factorisation, or les architectures récentes contiennent aussi une grande quantité de leurs poids dans les couches de convolution. De plus, l'utilisation de l'approximation de Hadamard impose selon nous une contrainte structurelle forte sur les matrices de poids et ne se-

rait peut-être pas adaptée pour toutes les couches d'une architecture profonde (les concepteurs des `Deep Fried Convnets` ne proposent d'ailleurs de remplacer qu'une seule couche du réseau).

Résumé des contributions. Dans cette section, nous introduisons un cadre général pour la compression de réseaux neuronaux utilisant la factorisation des couches en produits de matrices parcimonieuses. Nous explorons ensuite l'utilisation de l'algorithme `Palm4MSA` sur des couches de réseaux neuronaux pré-entraînés pour les exprimer en tant que produit de facteurs parcimonieux. Ces matrices creuses sont ensuite raffinées par descente de gradient pour coller au mieux à la tâche finale de prédiction. Nous évaluons l'effet de différents hyper-paramètres sur notre modèle et nous montrons notamment que les couches peuvent être factorisées en deux ou trois matrices parcimonieuses (et pas forcément $\log D$) pour obtenir des taux de compression maximaux tout en préservant la qualité des résultats. Enfin, nous présentons les résultats d'une expérience comparative entre notre technique de compression et plusieurs méthodes de l'état de l'art.

3.4.2 Approximation des matrices de poids en produits de matrices creuses

La Figure 3.7 donne une représentation schématique de l'effet de notre procédure de compression sur une architecture simple qui contiendrait seulement une couche de convolution et une couche dense. Nous ne nous limitons pas au cas où $Q := \log D$ mais voyons le paramètre Q comme un hyper-paramètre du modèle. Étant donné un vecteur d'entrée $\mathbf{x}$, l'utilisation d'une telle stratégie de compression dans une couche entièrement connectée donne en sortie $\mathbf{z}$:

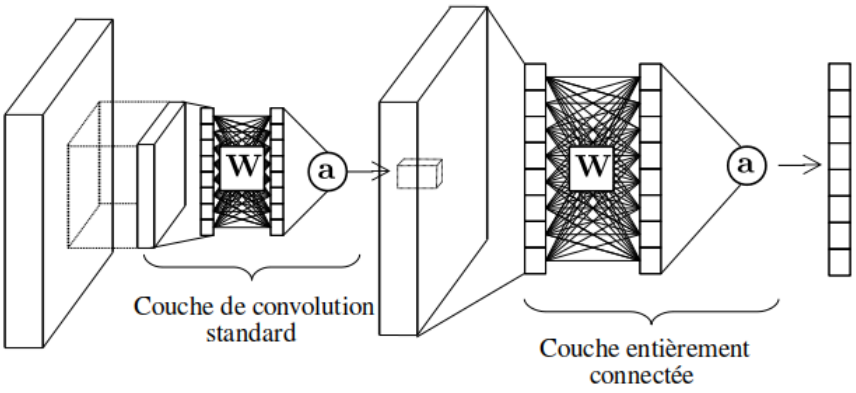
$$\mathbf{z} = \mathbf{a}\left(\prod_{i=1}^{Q} \mathbf{S}_i \mathbf{x}\right) \tag{3.25}$$

où $||S_i||_0 = \mathcal{O}(\max(D, d))$ de sorte que la complexité en temps et en espace de cette couche est de $\mathcal{O}(Q \cdot \max(D, d))$ au lieu de $\mathcal{O}(D^2)$.

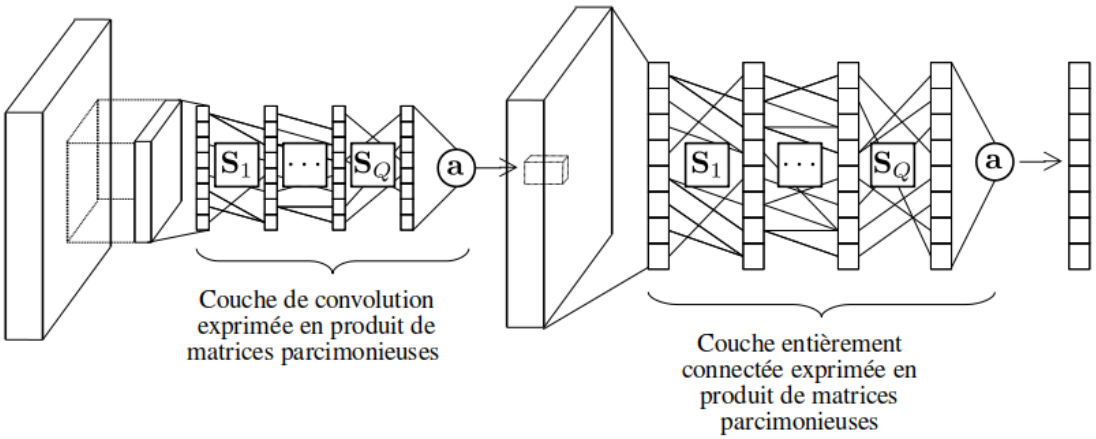
De même, dans les couches de convolution, on obtient la sortie $\mathbf{Z}$ à partir d'un tenseur d'entrée $\mathbf{X}$:

$$\mathbf{Z} = \mathbf{a}\left(\mathbf{t}\left(\mathbf{r}_S\left(\mathbf{X}\right) \prod_{i=1}^{Q} \mathbf{S}_i\right)\right) \tag{3.26}$$

où $||S_i|||_0 = \mathcal{O}(\max(S^2C, K))$ de sorte que la complexité en temps de la couche est réduite de $\mathcal{O}(HWCS^2K)$ à $\mathcal{O}(HWQ \cdot \max(CS^2, K))$ et la complexité en espace est réduite de $\mathcal{O}(CS^2K)$ à $\mathcal{O}(Q \cdot \max CS^2, K)$. Les opérateurs $\mathbf{t}$ et $\mathbf{r}_S$ sont



(a) Architecture standard



(b) Architecture compressée

FIGURE 3.7 : Illustration du passage d'un modèle standard au modèle compressée où les couches de convolutions et dense sont remplacées par des factorisations parcimonieuses.

les opérateurs de remodelage définis pour l'opération de convolution (voir Section 1.28).

L'expression des poids des couches à l'aide de tels produits de matrices parcimonieuses permet d'apprendre des couches qui implémentent une combinaison plus dense (c'est à dire contenant moins de zéros) des caractéristiques d'entrée qu'une seule matrice parcimonieuse avec le même nombre total de valeurs non nulles comme le montre la figure 3.8. Nous remarquons par ailleurs que le rang d'un produit de matrices étant le rang de sa composante de rang le plus faible, ce type de factorisation permet des décompositions de rang élevé.

3.4.3 Procédure de compression

Maintenant que notre cadre général de compression de réseaux neuronaux a été introduit, nous proposons une procédure simple pour apprendre les couches compressées d'un réseau neuronal pré-entraîné.

Après avoir entraîné un gros réseau neuronal à une tâche donnée jusqu'à ce que la performance souhaitée soit atteinte, nous voulons approximer chaque matrice de poids $\mathbf{W}$ par un produit de facteurs parcimonieux $\prod_{i=1}^{Q} \mathbf{S}_i$. Nous devons donc résoudre le problème d'optimisation suivant :

$$\min_{\{\mathbf{S}_i\}_{i=1}^{Q}} \left\| \mathbf{W} - \prod_{i=1}^{Q} \mathbf{S}_i \right\|_F^2 + \sum_{i=1}^{Q} \delta_{\mathcal{E}_i}(\mathbf{S}_i), \tag{3.27}$$

où pour chaque $i \in [\![Q]\!]$, $\delta_{\mathcal{E}_i}(\mathbf{S}_i) = 0$ si $\mathbf{S}_i \in \mathcal{E}_i$ et $\delta_{\mathcal{E}_i}(\mathbf{S}_i) = +\infty$ sinon. $\mathcal{E}_i$ est un ensemble de matrices contraintes qui impose notamment une structure de parcimonie à ses éléments. MAGOAROU et GRIBONVAL, 2016 ont proposé des stratégies algorithmiques pour apprendre une telle factorisation (voir Section 3.2.2).

Une fois que le problème de l'Équation (3.27) est résolu pour toutes les couches, nous remplaçons les matrices de poids dans le réseau par leur homologue produit de matrices parcimonieuses, comme décrit dans l'Équation (3.25) pour les couches entièrement connectées et dans l'équation (3.26) pour les couches convolutives. Ensuite, nous raffinons les valeurs non nulles de ces factorisations en utilisant l'objectif initial de la tâche mais le support de parcimonie reste fixé ; c'est-à-dire que les connections entre neurones sont figées pour le reste de l'entraînement.

3.4.4 Résultats expérimentaux préliminaires

Nous avons réalisé une série d'expériences pour démontrer l'intérêt d'utiliser notre procédure de décomposition des couches d'un réseau neuronal en produit de matrices parcimonieuses. Nous montrons que la technique proposée obtient un meilleur rapport de compression que certaines autres méthodes existantes avec une dégradation minime de la performance. Toutefois, la méthode de compression concurrente basée sur la magnitude des poids pour l'élagage des connections s'avère être à la fois plus simple à mettre en oeuvre et plus efficace. A l'instar de la section expérimentale de `QK-means` (Section 3.3.5), nous faisons ici l'amalgame entre le nombre de valeur non nulle et l'espace de stockage nécessaire pour les modèles considérés. Nous pensons que ce raccourci est raisonnable dans le sens où toutes les techniques considérées sont logées à la même enseigne en ce qui concerne le codage des nombres flottants et l'excédent de mémoire nécessaire pour stocker des matrices creuses est normalement marginal.

Nous reconnaissons que l'intérêt pratique de la décomposition des couches

en produits de facteurs parcimonieux reste à prouver mais les résultats préliminaires obtenus sont prometteurs et ouvrent des pistes d'amélioration que nous discutons en Section 3.4.5 et qui pourraient permettre de surpasser les méthodes concurrentes.

Pour décrire nos résultats, nous commençons en Section 3.4.4.1 par exposer les différents paramètres expérimentaux utilisés dans nos expériences afin d'en assurer la reproductibilité. En Section 3.4.4.2, nous fournissons une étude comparative de notre méthode avec d'autres méthodes de l'état de l'art que nous avons implémentées. Enfin, nous analysons l'effet de différents hyper-paramètres sur la performance de notre méthode en Section 3.4.4.3.

3.4.4.1 Paramètres expérimentaux

Nom	Input shape	# classes	Taille d'entraînement	Taille de validation	Taille de test	Architecture
MNIST	$(28 \times 28 \times 1)$	10	40 000	10 000	10 000	Lenet
SVHN	$(32 \times 32 \times 3)$	10	63 257	10 000	26 032	VGG19
CIFAR10	$(32 \times 32 \times 3)$	10	50 000	10 000	10 000	VGG19
CIFAR100	$(32 \times 32 \times 3)$	100	50 000	10 000	10 000	VGG19, Resnet50, Resnet20

TABLE 3.4 : Charactéristiques des jeux de données

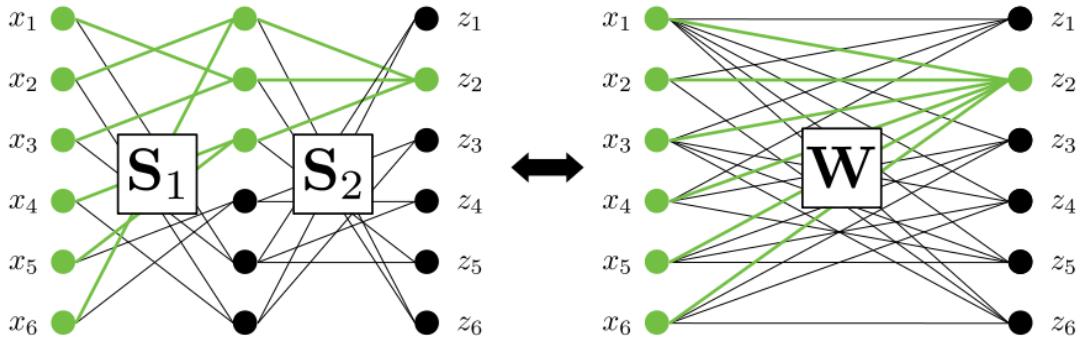


FIGURE 3.8 : Exemple jouet où un produit de 2 facteurs parcimonieux (à gauche) avec un nombre donné de coefficient non nuls (25) peut définir une matrice avec plus de coefficient non nuls (26) (à droite). Dans cette figure, nous voyons que le neurone de sortie z_2 est une combinaison linéaire dense de tous les neurones d'entrée $x_1 \cdots x_6$ mais la matrice de poids $\mathbf{W}$ a plus de connexions que le nombre total de connexions dans les matrices parcimonieuses $\mathbf{S}_1$ et $\mathbf{S}_2$.

Méthodes concurrentes. Par soucis d'équité dans la comparaison, nous avons choisi des méthodes concurrentes travaillant sur des réseaux pré-entrainés, à l'instar de la procédure que nous proposons. Nous nous comparons à plusieurs

méthodes de la famille des décompositions de rang faible que sont la décomposition de Tucker (KIM, PARK, YOO et al., 2016), la décomposition Tensortrain (NOVIKOV, PODOPRIKHIN, OSOKIN et al., 2015 ; GARIPOV, PODOPRIKHIN, NOVIKOV et al., 2016) et la décomposition en valeurs singulières (SAINATH, KINGSBURY, SINDHWANI et al., 2013) (ces méthodes sont évoquées en Section 1.3.4). Par ailleurs, nous souhaitions aussi nous comparer à une méthode visant à élaguer simplement des connections dans les couches du réseau de neurones à compresser. Nous pensons que notre méthode pourrait être plus intéressante que ces stratégies car un produit de matrices parcimonieuses devrait couvrir un plus large espace de matrice qu'une seule matrice creuse, comme l'illustre l'exemple jouet de la Figure 3.8. Pour illustrer cela, nous choisissons la méthode basée sur la magnitude des poids (ZHU et GUPTA, 2017) que GALE, ELSEN et HOOKER, 2019 ont montré être en général aussi efficace que des techniques plus élaborées telles que celle proposée par LOUIZOS, WELLING et KINGMA, 2018 basée sur une régularisation L_0 ou MOLCHANOV, ASHUKHA et VETROV, 2017 qui utilise le *dropout* variationel (*Variational dropout*). En nous comparant à ces méthodes, nous espérions montrer l'intérêt, en terme d'expressivité, des décompositions en matrices creuses comparé aux matrices creuses simples ou aux décomposition en rang faible.

Une autre méthode concurrente que nous avions envisagée était la méthode des `Deepfried convnets` étendue aux couches convolutives. Comme évoqué en Section 3.4.1, la méthode `FastFood` s'apparente en fait à une décomposition en produit de matrices creuses : la matrice de Hadamard est exactement égale à un produit de matrices creuses et les autres matrices de la décomposition `FastFood` sont soit des matrices diagonales (donc creuses), soit une matrice de permutation (creuse aussi). Il nous semblait donc pertinent de nous comparer à une version des réseaux de convolution `Deepfried` étendue aux couches de convolution de façon analogue à l'Équation (3.26). Toutefois, cette approche n'a pas été concluante et, excepté pour le jeu de données `MNIST`, le réseau ainsi construit s'est révélé incapable d'apprendre. Ces résultats ne sont donc pas présentés.

Enfin, nous évaluons l'intérêt d'utiliser l'algorithme `Palm4MSA` pour découvrir une décomposition en matrice parcimonieuse des couches en comparant cette méthode avec des décompositions aléatoires. Plus précisément, nous évaluons (I) la performance d'un modèle dont les couches seraient décomposées par `Palm4MSA` pour découvrir un support de parcimonie mais dont les poids seraient ré-initialisés ; (II) la performance d'un modèle dont les poids et le support de parcimonie seraient décidés aléatoirement à l'initialisation.

Détails d'implémentation. Le code a été écrit en Python, y compris l'algorithme `Palm4MSA`. Le code de l'algorithme `Palm4MSA` et le code des expériences pour la compression de réseaux neuronaux sont disponibles sur des dépôts en

ligne (GIFFON et EMIYA, 2020 ; GIFFON, 2020d) [1] [2]. Les réseaux de neurones ont été implémentés avec `Keras` (CHOLLET, 2015) et `Tensorflow` (ABADI, AGARWAL, BARHAM et al., 2015).

Faute d'implémentation efficace des matrices creuses en Tensorflow, nous avons dû détourner les implémentations du produit matriciel dense et de la convolution offertes par Keras afin que l'apprentissage des réseaux soit le plus efficace possible.

Implémentation des matrices creuses. Dans notre implémentation, une matrice creuse est en fait définie par le produit de Hadamard $\odot$ (le produit point à point) entre une matrice dense qui encode les poids à apprendre et une matrice d'éléments binaires constants qui encode le support de parcimonie. Ainsi, l'implémentation d'une matrice parcimonieuse $\mathbf{S} \in \mathbb{R}^{D \times D},\ \|\mathbf{S}\|_0 = \mathcal{O}(D)$ est :

$$\mathbf{S} = \mathbf{W} \odot \mathbf{M}, \tag{3.28}$$

où $\mathbf{W} \in \mathbb{R}^{D \times D}$ est une matrice de poids dense et $\mathbf{M} \in \{0,1\}^{D \times D},\ \|\mathbf{M}\|_0 = \mathcal{O}(D)$ est une matrice binaire constante. Avec cette implémentation, il est clair que les valeurs des poids du gradient de $\mathbf{W}$ en dehors du support de parcimonie défini par $\mathbf{M}$ sont toujours égaux à zéro et donc les poids correspondants de $\mathbf{W}$ ne sont jamais mis à jour. En d'autres termes, le support de parcimonie de $\mathbf{S}$ est bien fixé par $\mathbf{M}$. Cette implémentation est utilisée comme preuve de concept de notre méthode et n'a aucun intérêt pratique dans un contexte concret de compression de réseau de neurone. En effet, elle nécessite le stockage dense de toutes les valeurs de $\mathbf{W}$ et $\mathbf{M}$ et donc la mémorisation de $2D^2$ valeurs pour simuler la parcimonie d'une matrice de taille D^2 qui contient $\mathcal{O}(D)$ valeurs non-nulles. Ironiquement, cette implémentation théoriquement sous-optimale profite en fait des algorithmes parallèle du produit matriciel et s'avère être plus rapide sur GPU qu'une implémentation qui utiliserait la classe `SparseTensor` de Tensorflow [3].

Implémentation de la convolution. Pour calculer les convolutions dans le réseau, nous reconstruisons le noyau de convolution à partir du produit des matrices creuses, puis nous utilisons directement ce noyau de convolution en

[1] Jusqu'à la publication de ces travaux, le code pour la compression de réseaux neuronaux par factorisation parcimonieuse est accessible seulement pour les personnes ayant demandé un accès.

[2] Une implémentation de l'algorithme est aussi disponible dans le paquet `Faust` distribué par l'Inria (*FAμST* 2020). Nous n'avons pas pu conduire nos expériences avec cette implémentation du fait de la présence de bugs à ce moment.

[3] Il est en fait possible d'utiliser la classe `SparseTensor` en conjonction avec la classe `Variable` pour implémenter ces couches parcimonieuses et avoir exactement un nombre de paramètre réduit mais cette implémentation était plus lente et nous avons préféré ne pas l'utiliser pour les expériences.

paramètre de la fonction `conv2d` de Tensorflow. Une fois encore, cette implémentation sert de preuve de concept pour observer le comportement de notre méthode et n'a pas vocation à présenter un intérêt pratique mais plutôt d'être la plus rapide possible à mettre en oeuvre, compte tenu des librairies existantes. En effet, reconstruire le noyau de convolution à partir des matrices creuses nullifie totalement l'éventuel gain en espace de la décomposition ; en revanche, l'utilisation de la machinerie de Tensorflow pour calculer la convolution apporte un gain de temps au moment de l'apprentissage.

Implémentation de la décomposition Tensortrain. La décomposition Tensortrain a été réalisée en appliquant la fonction de décomposition `matrix_product_state` fournie par la bibliothèque Tensorly sur les tenseurs obtenus sur les réseaux pré-entrainés. Dans tous les cas les tenseurs étaient décomposés en 4 et la valeur de rang maximale possible des décompositions est spécifiée par la valeur R dans les expériences. Après décomposition des couches pré-apprises, les poids sont raffinés.

Implémentation de la décomposition de Tucker et de la décompostion SVD. Les décompositions de Tucker et décomposition en valeurs singulières sont respectivement dédiées à la compression des couches de convolution et des couches denses. Nous nous comparons donc à un modèle hybride où les deux techniques sont combinées. Le rang de la décomposition de Tucker est détecté automatiquement par la méthode VBMF (*Variational Bayes Matrix Factorization*), comme expliqué par KIM, PARK, YOO et al., 2016. Le rang des couches denses est choisi tel qu'un certain pourcentage (précisé dans les expériences) des valeurs singulières obtenues par SVD soient obtenues. Après décomposition de Tucker et SVD des couches pré-apprises, les poids sont raffinés.

Implémentation de l'élagage des connections en fonction de la magnitude. Nous avons utilisé la fonction `prune_low_magnitude` de la librairie `tensorflow_model_optimization` fourni par Tensorflow. Avec cette méthode, l'élagage et le raffinement des poids se font de façon jointe en supprimant peu à peu les connections pendant l'apprentissage jusqu'à obtenir le pourcentage d'élagage désiré.

(Ré)-Initialisation des poids d'une décomposition en matrices creuses. Lorsque les poids d'une factorisation en matrices creuses ne sont pas fournis par `Palm4MSA`, il faut que l'initialisation des poids soit adaptée au nombre réduit de connections dans la couche. Pour ce faire, nous avons combiné les initialisations `Xavier` (GLOROT et BENGIO, 2010) et `He` (HE, ZHANG, REN et al., 2015) détaillées en Section 1.3.3. Le premier facteur creux, c'est-à-dire le premier a être appliqué aux vecteurs d'entrées, est initialisé en utilisant la méthode `He` car les vecteurs

en entrée sont passés par la fonction d'activation $ReLU$ de la couche précédente précédente et donc ne sont pas centrés en 0. Les facteurs creux suivant, eux, sont initialisés avec la méthode `Xavier` car il n'y a pas de non-linéarité entre chaque facteur. Pour l'initialisation d'un facteur creux donné, la valeur fan_{in} est choisie égale au nombre de valeurs non-nulles dans chaque colonne du facteur plutôt que la dimension totale du vecteur d'entrée. Cette procédure s'est avérée nécessaire au fonctionnement du modèle lorsque les poids d'une décomposition en facteurs creux étaient ré-initialisés.

Jeux de données et architectures neuronales. Nos expériences sont menées sur quatre jeux de données de classification d'image standard de difficulté variable : `MNIST` (DENG, 2012), `SVHN` (NETZER, WANG, COATES et al., 2011), `CIFAR10`, `CIFAR100` (KRIZHEVSKY, 2009). Les réseaux de neurones à compresser sont des réseaux classiquement utilisés avec ces jeux de donées c'est-à-dire `Lenet`, `VGG19`, `Resnet50` et `Resnet20`; ces architectures sont détaillées en Section 1.3.2. Les détails concernant les caractéristiques de ces jeux de données et les architectures neuronales associées sont visibles dans le Tableau 3.4

Hyper-paramètres. Le pré-apprentissage des réseaux neuronaux, la décomposition des couches ainsi que le raffinement des réseaux nécessitent des paramètres soigneusement choisis que nous détaillons maintenant.

Pré-apprentissage des réseaux de neurones. Les architectures `Lenet`, `VGG19` et `Resnet` ont été pré-entraînées avec respectivement 100, 300 et 200 itérations. Nous utilisons des paramètres classiques pour l'apprentissage de ce type de réseaux pour ce type de tâches (LI, 2020). Pour le réseau `Lenet`, l'algorithme de descente de gradient `RMSProp` (HINTON, SRIVASTAVA et SWERSKY, 2012) a été utilisé avec un pas d'optimisation de 10^{-4}. Pour les réseaux `VGG19` et `Resnet`, nous avons utilisé l'algorithme classique de Descente de Gradient Stochastique (SGD) avec un paramètre d'élan de *Nesterov* (*Nesterov momentum*) (SUTSKEVER, MARTENS et DAHL, 2013) de 0.9 et un pas d'optimisation de 0.1. Dans tous les cas, nous avons utilisé une stratégie classique d'augmentation des données d'apprentissage par translation, rotation et retournement des images.

Décomposition par Palm4MSA. Les critères d'arrêt employés pour l'algorithme `Palm4MSA` étaient 300 itérations maximum ou une modification relative entre deux itérations inférieure à 10^{-6}. La méthode de projection est celle utilisée par et détaillée en Annexe 3.A. Avec K le niveau de parcimonie désiré, cette méthode garantit que chaque facteur creux contient au moins K valeurs non-nulles par ligne et par colonne et au maximum $2K$ en moyenne.

Apprentissage des réseaux de neurones. Nous avons réalisé le raffinement des réseaux après compression avec le même nombre d'itérations que le réseau initial. Pour le réseau `Lenet` comprimé, nous avons ré-utilisé l'optimiseur `RMSProp`. En revanche, les réseaux `VGG19` et `Resnet` ont été raffinés avec l'optimiseur `Adam` (KINGMA et BA, 2014). Pour chaque méthode de compression (et chaque configuration), la meilleure valeur de pas d'optimisation a été choisie en utilisant l'erreur de classification sur un échantillon de validation après un apprentissage de 10 itérations avec des valeurs de pas égales à $\{10^{-3}, 10^{-4}, 10^{-5}\}$.

Matériel. Les expériences ont été menées, au total, en moins de 9000h sur le super-calculateur Jean Zay du ministère de l'Enseignement supérieur, de la Recherche et de l'Innovation, installé à l'IDRIS, centre national de calcul du CNRS. Les noeuds de calcul sont dotés de processeurs Intel Cascade Lake 6248 et 6226 ainsi que de GPU Nvidia Tesla V100 SXM2. Le budget calcul total estimé est de moins de 5000€. Cette information était communiquée au moment de l'enregistrement du projet sur le site de l'IDRIS.

3.4.4.2 Étude comparative de méthodes de compression de réseaux de neurones

Nous réalisons à présent une étude comparative de notre méthode vis-à-vis d'autres méthodes de compression existantes que sont la décomposition `Tensortrain`, la méthode hybride des décompositions de Tucker et SVD et l'élagage des connections en fonction de la magnitude de leur poids.

Décomposition Tensortrain. Les quatre premiers graphiques de la Figure 3.9 montrent que la décomposition `Tensortrain` est extrêmement efficace pour réduire le nombre de poids dans le modèle et notre procédure de décomposition des couches peine à atteindre les mêmes performances. Toutefois, comme décrit en Section 1.3.4.1, la décomposition `Tensortrain` nécessite un grossissement de la taille de l'entrée par un facteur égal au rang de la décomposition. Le Tableau 3.5 montre la taille maximale que peut prendre une image traitée par un réseau comprimé avec la méthode `Tensortrain` comparé à n'importe quelle autre méthode de compression que nous considérons. Nous voyons donc qu'en pratique, la méthode de compression `Tensortrain` ne peut pas être utilisée avec des réseaux neuronaux qui contiennent un trop grand nombre de filtres de convolution tels que les réseaux Resnet, ce qui justifie l'absence de point pour ces réseaux dans la Figure 3.9.

Décomposition de Tucker avec SVD. Cet hybride de compression est capable de réduire raisonnablement la taille d'un réseau de neurones pré-entrainé

	MNIST Lenet	SVHN Vgg19	Cifar10 Vgg19	Cifar100 Vgg19	Cifar100 Resnet20	Cifar100 Resnet50
Autres	1024	65536	65536	65536	65536	262144
`Tensortrain` R=6 K=4	2304	393216	393216	393216	393216	1572864
`Tensortrain` R=10 K=4	3840	655360	655360	655360	655360	2621440
`Tensortrain` R=14 K=4	5376	917504	917504	917504	917504	3670016

TABLE 3.5 : Tableau illustrant le grossissement des représentations des exemples induit par l'architecture `Tensortrain`. Le nombre maximal de valeurs non-nulles dans la représentation d'un exemple au cours de l'exécution de l'architecture est affiché.

sans impacter la performance mais elle ne peut pas atteindre les taux de compression atteints par les autres méthodes.

Élagage des connections en fonction de la magnitude des poids. Excepté pour le jeu de données `MNIST`, cette méthode a fonctionné extrêmement bien et arrive a préserver une bonne qualité en précision en supprimant jusqu'à 98% des connections du modèle. En général, notre méthode atteint des taux de compression équivalents voire légèrement moins bons pour des qualités de prédiction équivalentes.

3.4.4.3 Analyse de la méthode.

Dans cette Section, nous nous attardons sur le comportement de la méthode. Nous commençons par justifier l'utilisation de `Palm4MSA` pour la décomposition des couches en produits de facteurs parcimonieux puis nous verrons l'effet de différents hyper-paramètres sur la performance du modèle et sur le taux de compression.

Erreur d'approximation. La méthode `Palm4MSA` est capable d'approcher une matrice dense par un produit de matrices creuses. Une question légitime est de savoir si l'approximation trouvée est proche de la matrice initiale. Malheureusement, il n'existe à notre connaissance pas de borne théorique sur la capacité d'approximation de la méthode. La Figure 3.10 montre, dans le cas de réseaux `VGG19` et `Resnet50` pré-appris sur `Cifar100`, à quel point la décomposition de chaque couche donne des poids proches de la matrice initiale. Pour une matrice de poids $\mathbf{W}$ et sa version approchée par un produit de Q facteurs parcimonieux $\tilde{\mathbf{W}} := \prod_{q=1}^{Q} \mathbf{S}_q$, l'erreur relative est calculée comme suit :

$$erreur = \frac{\|\mathbf{W} - \tilde{\mathbf{W}}\|_F^2}{\|\mathbf{W}\|_F^2}. \tag{3.29}$$

La Figure 3.10 montre déjà, comme attendu, que plus le niveau de parcimonie est élevé (plus K, le nombre minimal de valeurs non nulles par ligne et par

colonne, est grand) plus l'approximation est bonne. Dans certains cas (lorsque les matrices à approximer sont plutôt petites), l'erreur est même nulle. Toutefois, nous observons que pour certaines couches, l'erreur est extrêmement élevée, et donc nous pouvons nous demander dans quelle mesure l'utilisation de l'algorithme `Palm4MSA` est utile. Pour contrôler cette utilité, nous avons construit deux autres types de modèles qui implémentent la décomposition de couches en produit de matrices creuses :

- «Sparse Facto. aléatoire» : Nous construisons complètement aléatoirement des factorisations creuses aléatoires. Le support de parcimonie est choisi en faisant la projection d'une matrice Gaussienne centrée réduite. Les poids sont initialisés en utilisant la procédure décrite en Section 3.4.4.1 ;
- «Sparse Facto. re-init.» : Nous utilisons le support de parcimonie obtenu par `Palm4MSA` mais nous ré-initialisons les poids en utilisant la procédure décrite en Section 3.4.4.1.

Le Tableau 3.6 montre que dans tous les cas, c'est le réseau comprimé avec la méthode `Palm4MSA` qui obtient la meilleure performance en classification après raffinement du réseau. Ceci justifie l'utilisation de l'algorithme d'approximation. On remarque par ailleurs que la version «Sparse Facto. re-init» obtient globalement[4] de meilleurs résultats que «Sparse Facto. aléatoire» ce qui suggère que non seulement les poids trouvés par `Palm4MSA` sont importants mais aussi le support de parcimonie.

Effet du niveau de parcimonie et du nombre de facteur. Nous appelons niveau de parcimonie le nombre approximatif de valeurs non nulles dans chaque ligne et chaque colonne des matrices creuses constituant les décompositions parcimonieuses. Sur la Figure 3.9, nous représentons la performance de notre modèle avec différents niveaux de parcimonie K et différents nombres de facteurs Q dans la décomposition. Nous voyons sur la figure que le nombre de facteur semble avoir un effet assez limité sur la qualité de la performance et que le niveau de parcimonie est un facteur plus déterminant de la qualité de l'approximation. Notons ici que nous ne nous sommes pas servi de la version Hierarchique de `Palm4MSA` (MAGOAROU et GRIBONVAL, 2016).

3.4.5 Ouverture

Nous avons décrit une approche unique pour comprimer les couches denses et convolutives d'un réseau neuronal en exprimant les matrices de poids par des

[4]Sur le modèle `VGG19` + `SVHN`, la version «Sparse Facto. re-init» fonctionne étonnamment mal. Nous ne sommes pas en mesure d'expliquer cette chute de performance autrement que par un bug possible mais qui reste à ce jour introuvable.

	MNIST Lenet	SVHN Vgg19	Cifar10 Vgg19	Cifar100 Vgg19	Cifar100 Resnet20	Cifar100 Resnet50
Base	**0.99**	**0.96**	**0.93**	**0.67**	**0.73**	**0.76**
Sparse Facto. Q=2 K=2	**0.99**	0.92	0.84	0.46	0.56	0.67
Sparse Facto. re-init. Q=2 K=2	**0.99**	0.82	0.81	0.42	0.53	0.57
Sparse Facto. aléatoire Q=2 K=2	0.98	0.91	0.81	0.44	0.48	0.41
Sparse Facto. Q=2 K=14	**0.99**	0.95	0.92	0.64	0.69	0.72
Sparse Facto. re-init. Q=2 K=14	**0.99**	0.44	0.86	0.57	0.63	0.63
Sparse Facto. aléatoire Q=2 K=14	**0.99**	0.92	0.85	0.58	0.62	0.62
Sparse Facto. Q=3 K=2	**0.99**	0.94	0.85	0.42	0.57	0.67
Sparse Facto. re-init. Q=3 K=2	0.98	0.91	0.80	0.32	0.48	0.51
Sparse Facto. aléatoire Q=3 K=2	0.98	0.90	0.79	0.39	0.29	0.47
Sparse Facto. Q=3 K=14	**0.99**	0.95	0.92	0.62	0.70	0.72
Sparse Facto. re-init. Q=3 K=14	**0.99**	0.89	0.84	0.31	0.60	0.58
Sparse Facto. aléatoire Q=3 K=14	**0.99**	0.93	0.84	0.51	0.60	0.59

TABLE 3.6 : Tableau de résultats présentant la performance de modèles de réseaux de neurones comprimés par 3 techniques de décomposition des couches en produit de matrices creuses. «Sparse Facto.» fait référence à la procédure centrale du papier où `Palm4MSA` est utilisé sur les couches pré-entrainées ; «Sparse Facto. re-init» consiste à l'utilisation de `Palm4MSA` pour définir le support de parcimonie mais les poids sont ré-initialisés ; «Sparse Facto. aléatoire» correspond à la version du modèle où les poids ainsi que le support de parcimonie des matrices sont initialisés aléatoirement.

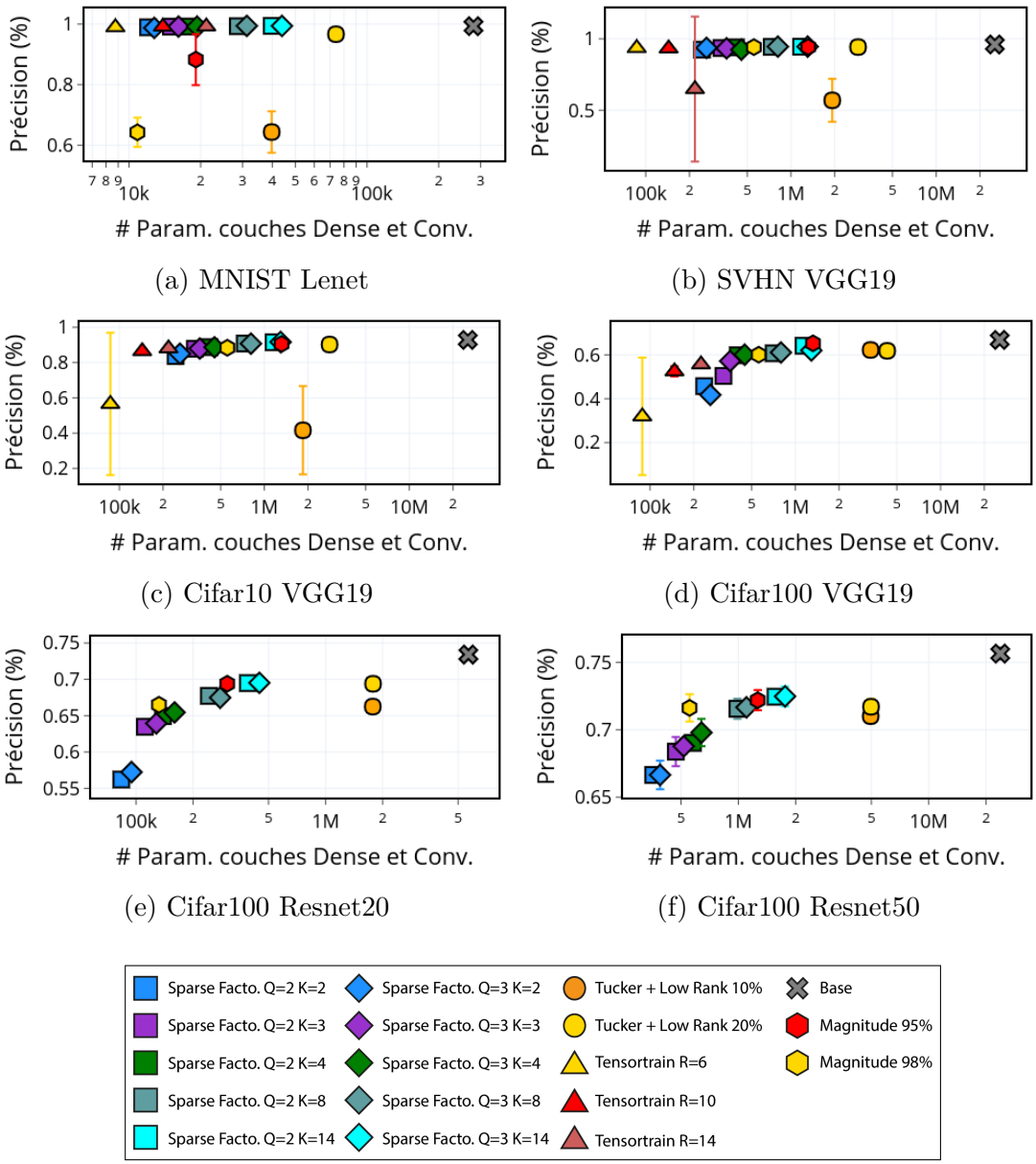
produits de facteurs parcimonieux. L'idée sous-jacente qui motive notre méthode est que, pour un budget de parcimonie total fixé, un produit de matrices parcimonieuses devrait être plus expressif qu'une matrice parcimonieuse seule. Notre approche se base en pratique sur l'algorithme `Palm4MSA` pour approximer les matrices de poids d'un réseau pré-entrainé, avant de réaliser un raffinement des couches en utilisant le critère d'erreur de classification et un algorithme de descente de gradient standard. Cette approche très directe nous a permis d'obtenir des résultats assez satisfaisants qui se comparent bien à certaines méthodes standard de l'état de l'art. Toutefois, nous n'avons pas pu illustrer un bénéfice pratique à l'utilisation de notre modèle comparé à une méthode d'élagage rudimentaire qui se base sur la magnitude des poids des couches pour supprimer certaines connections (ZHU et GUPTA, 2017). Nous envisageons cependant d'améliorer notre méthode en utilisant plutôt les activations de chaque couche pour réaliser les décompositions : de façon similaire à ce qui est fait par STOCK, JOULIN, GRIBONVAL et al., 2020, nous pourrions décomposer une matrice de poids $\mathbf{W}$ en résolvant le problème

$$\underset{\mathbf{S}_1,\dots,\mathbf{S}_Q}{\arg\min} \|\mathbf{X}\mathbf{W} - \mathbf{X}\prod_{q=1}^{Q}\mathbf{S}_q\|_F^2 \tag{3.30}$$

où $\mathbf{X}$ correspond à une matrice d'observations d'entrées pour $\mathbf{W}$. L'idée ici serait de faire en sorte que l'approximation de chaque couche produise des activations similaires à celles de la couche correspondante du modèle de base. Cette

approche évoque la stratégie utilisée habituellement en distillation (HINTON, VINYALS et DEAN, 2015 ; ROMERO, BALLAS, KAHOU et al., 2015).

Un résultat expérimental intéressant de notre méthode est que, même si les résultats d'approximation couche par couche sont parfois extrêmement mauvais, les poids contenus dans les décompositions découvertes par `Palm4MSA` ont en fait un intérêt important pour préserver la performance du réseau : ré-initialiser les poids de la décomposition conduit à une importante dégradation des performances. Ce constat corrobore l'hypothèse du ticket de loto (*lottery ticket hypothesis*) proposée par FRANKLE et CARBIN, 2019 qui suggère que les réseaux neuronaux sur-paramétrés contiennent en fait un «sous-réseau» avec peu de paramètres qui serait capable d'accomplir la tâche d'apprentissage. Sous cette hypothèse, notre méthode pourrait correspondre à une décomposition en matrices parcimonieuses des couches de poids de ce sous-réseau.



(a) MNIST Lenet

(b) SVHN VGG19

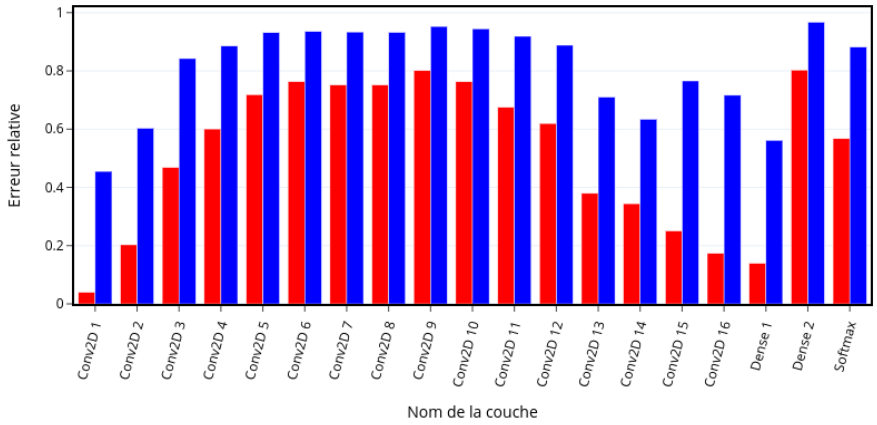
(c) Cifar10 VGG19

(d) Cifar100 VGG19

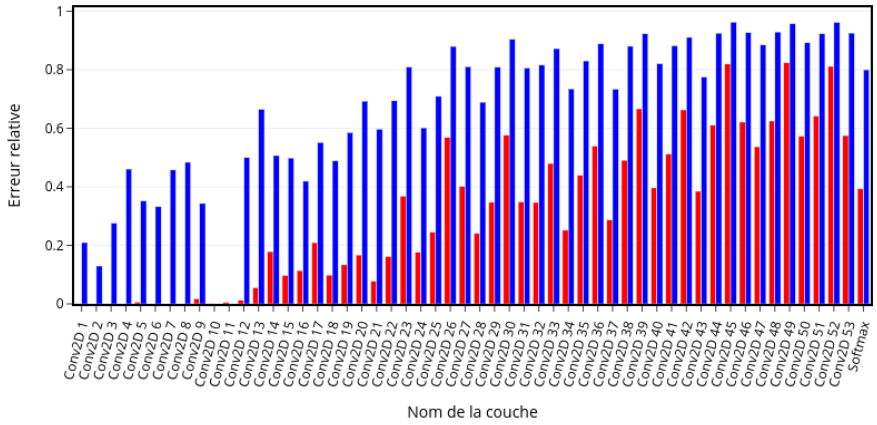
(e) Cifar100 Resnet20

(f) Cifar100 Resnet50

FIGURE 3.9 : Performance des méthodes en fonction du nombre de paramètres cumulé des couches Dense et couches de Convolution sur plusieurs jeux de données et plusieurs modèles.



(a) Cifar100 Vgg19



(b) Cifar100 Resnet50

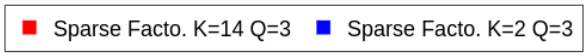


FIGURE 3.10 : Erreur relative d'approximation des couches par un produit de matrices creuses en utilisant `Palm4MSA`. Seuls les modèles `Resnet50` et `VGG19` sont représentés mais des résultats similaires sont observables avec les autres modèles.

3.5 Conclusion

Nous avons examiné l'intégration de produits de matrices parcimonieuses à la place de matrices denses dans des algorithmes commun d'apprentissage automatique. En particulier, nous proposons une variante de l'algorithme des K-moyennes, appelée `QK-means`, qui, à chaque itération, fournit une matrice de centroïdes approchée par un produit de matrices parcimonieuses. Cette décomposition peut ensuite être utilisée à bas coût dans d'autres algorithmes d'apprentissage qui auraient habituellement utilisé une matrice de centroïdes standard. Similairement, nous proposons d'utiliser les matrices de poids d'un réseaux neuronal de grande taille pour construire un nouveau réseau, plus léger, dont les couches sont constituées de produits de matrices parcimonieuses. D'une certaine façon, ces deux approches s'apparentent à de la «distillation» (BUCILUǍ, CARUANA et NICULESCU-MIZIL, 2006 ; HINTON, VINYALS et DEAN, 2015) où on utilise un modèle lourd mais performant pour construire un modèle plus léger de performance équivalente.

Dans les deux cas que nous avons étudiés, il est apparu assez clairement que les infrastructures logicielles et matérielles couramment utilisées ne permettaient pas pour l'instant de tirer partie de ce type de modèles pour accélerer effectivement des modèles d'apprentissage. Toutefois, ces approches en sont encore à leur balbutiemments et les gains théoriques déjà obtenus peuvent servir de motivation vers le développement de nouveaux matériels et algorithmes qui prendraient en compte ce type de structures. On peut aussi noter que l'algorithme `Palm4MSA` sur lequel nos travaux se sont basés pourrait lui même être amélioré à des fins d'accélération ou bien de qualité d'approximation. Pour l'instant, très peu de résultats existent pour caractériser les conditions de bon fonctionnement de `Palm4MSA` ou évaluer la qualité de la décomposition d'une matrice quelconque. Des recherches dans cette direction renforceraient certainement l'intérêt des méthodes que nous proposons. Dans ce chapitre, nous nous sommes essentiellement intéressés à la mesure du nombre de paramètres pour évaluer la capacité de compression de nos modèles. Or, la taille effective d'un modèle d'apprentissage en mémoire dépend non seulement du nombre de paramètres à stocker mais aussi de la taille, en bit, de chacun de ces paramètres. En ce sens, il pourrait être envisageable de combiner les modèles que nous proposons à des techniques de quantification de poids, qui visent précisément à réduire la taille en bit de chaque paramètre à stocker.

Annexes

3.A Fonction de projection pour `Palm4MSA`

La fonction utilisée pour projeter les facteurs sur l'ensemble des solutions possibles (Ligne 9 de l'Algorithme 1) est la même que celle utilisée dans le code source de MAGOAROU et GRIBONVAL, 2016. Cette fonction de projection relâche la contrainte d'avoir exactement :

$$\forall j \in Q,\ ||\mathbf{S}_j||_0 \leq \epsilon_j \tag{3.31}$$

pour devenir la contrainte plus faible d'avoir :

$$\begin{array}{l} \forall j \in Q,\ \epsilon_j \leq\ ||\mathbf{S}_j||_0 \leq 2\epsilon_j \\ \forall i,\ ||\mathbf{S}_j[i]||_0 = \lceil \epsilon_j / A \rceil \text{ et } ||\mathbf{S}_j^T[i]||_0 = \lceil \epsilon_j / A \rceil \end{array}, \tag{3.32}$$

avec $\mathbf{S}_j[i]$ la $i^{ème}$ ligne de $\mathbf{S}_j$ et $\mathbf{S}_j \in \mathbb{R}^{A \times A}$. En d'autres termes, elle garantit que chaque ligne et chaque colonne ait *au moins* $\lceil \epsilon_j / A \rceil$ valeurs non nulles : (I) chaque ligne de A doit avoir $\lceil \epsilon_j / A \rceil$ valeurs non nulles alors la matrice a au moins ϵ_j valeurs au total et (II) elle ne peut pas avoir plus de $2\epsilon_j$ valeurs non nulles même en considérant que les ligne et les colonnes de la matrice ne partagent pas les mêmes valeurs non nulles. Nous voyons en pratique qu'il y a en fait collision entre les valeurs non nulles des lignes et des colonnes ce qui permet de faire en sorte que la contrainte (3.31) soit à peu près respectée. La procédure est résumée dans l'Algorithme 3 où la fonction $get\_max\_by\_line(\mathbf{S}, c)$ renvoie $\mathbf{S}$ avec seulement ses C plus grandes valeurs dans chaque ligne et les autres valeurs à zéro et $\neg mask(\mathbf{S})$ renvoie le négatif du masque de $\mathbf{S}$ *e. g.* la matrice $\mathbf{M}$ avec $\mathbf{M}_{i,j} = 1$ où $\mathbf{S}_{i,j} = 0$ et $\mathbf{M}_{i,j} = 0$ sinon.

Algorithme 3 Fonction de projection pour `Palm4MSA`

Entrée: Une matrice $\mathbf{S} \in \mathbb{R}^{A \times A}$ à projeter telle que sa norme $L0$ est $\epsilon \leq ||\mathbf{S}||_0 \leq 2\epsilon$

1: $C \leftarrow \lceil \epsilon / A \rceil$
2: $\mathbf{S}' \leftarrow get\_max\_by\_line(\mathbf{S}, C)$
3: $\mathbf{S}'' \leftarrow get\_max\_by\_line(\mathbf{S}^T, C)^T$
4: $P_\epsilon(\mathbf{S}) \leftarrow \mathbf{S}' + \mathbf{S}'' \odot \neg mask(\mathbf{S}'')$
5: $P_\epsilon(\mathbf{S}) \leftarrow \mathbf{S}' + \mathbf{S}'' \odot \mathbf{M}$

Sortie: $P_\epsilon(\mathbf{S})$ la projection de $\mathbf{S}$ sur l'ensemble des solutions possibles

4 Élagage de forêts aléatoires avec pondération des arbres par l'algorithme OMP

Contents

4.1 Introduction

Les forêts aléatoires sont des méthodes d'apprentissage ensembliste pour la classification ou la régression supervisée qui consiste à construire un grand nombre d'arbres aléatoires dont les prédictions sont combinées grâce à un vote majoritaire ou une moyenne. Dans l'article de référence sur les forêts aléatoires de Breiman, 2001, il est montré qu'une augmentation du nombre d'arbres ne provoque pas de sur-apprentissage mais donne au contraire une borne asymptotique de l'erreur de généralisation. En conséquence, les bonnes performances des forêts aléatoires sont souvent obtenues grâce à des modèles contenant un grand nombre d'arbres et donc lourds et peu interprétables. Une approche commune pour surmonter ces problèmes consiste à réduire la taille des forêts aléatoires au moyen de techniques d'élagage (Kulkarni et Sinha, 2012) qui désigne le processus de construction d'une petite forêt aléatoire de taille K à partir d'une Forêt initiale de taille $L \gg K$. Similairement à ce qui est vu au Chapitre 3, l'élagage d'une forêt aléatoire s'apparente à de la distillation depuis une forêt initiale de taille conséquente vers une forêt compressée contenant moins d'arbres mais conservant une bonne performance.

Comme nous le voyons en Section 4.2, la décision prise par une forêt aléatoire étant donnée une observation fait intervenir une combinaison linéaire, habituellement avec des coefficients uniformes, des prédictions réalisées par chacun des arbres de la forêt. Dans cette perspective, l'élagage d'une forêt aléatoire correspond à remplacer des coefficients de cette combinaison par des zéros, autrement dit, à rendre la combinaison linéaire plus parcimonieuse.

Résumé des contributions Nous formulons le problème de l'élagage des forêts aléatoires sous la forme d'un problème d'optimisation avec contrainte L_0. Pour approcher la solution de ce problème, nous proposons d'utiliser l'algorithme *Orthogonal Matching Pursuit* (OMP), ce qui a aussi pour effet de donner des poids *d'importance* aux arbres de la forêt qui sont sélectionnés. Nous évaluons la qualité de la méthode ainsi que l'effet des poids non-uniformes attribués aux arbres de la forêt sur un large ensemble de jeux de données. Cette évaluation nous amène à proposer une version non-négative de la méthode qui permet d'obtenir des performances équivalentes ou meilleures que toutes les autres techniques d'élagage auxquelles nous nous comparons.

4.2 Arbres de décisions et forêts aléatoires

Les arbres de décisions pour la classification ou la régression (BREIMAN, FRIEDMAN, STONE et al., 1984) sont des modèles d'apprentissage qui associent une donnée d'entrée à une variable cible, ils sont représentés par une fonction $\mathbf{t} : \mathcal{X} \mapsto \mathcal{Y}$. Un arbre de décision représente un diagramme de flux de données ou chaque noeud de l'arbre représente un test sur la donnée d'entrée et chaque branche un résultat de ce test. En partant de la racine, le processus de décision passe de noeud en noeud en fonction des résultats des tests successifs jusqu'à arriver à l'une des feuilles de l'arbre, qui représentent les valeurs cibles. L'apprentissage d'un arbre de décision se fait itérativement en construisant les tests de chaque noeud de l'arbre de façon à ce qu'à chaque étape un certain critère soit optimisé. Nous n'entrons pas plus dans les détails de l'apprentissage d'arbres de décision et renvoyons le lecteur vers des livres tels que notamment celui de BREIMAN, FRIEDMAN, STONE et al., 1984 ou encore le livre généraliste sur l'apprentissage artificiel de CORNUÉJOLS et MICLET, 2011.

Toutefois, le caractère glouton de l'apprentissage des arbres de décision les rends particulièrement instables et sujets au sur-apprentissage. Les forêts aléatoires ont par la suite été élaborées par BREIMAN, 2001 pour contre-balancer ce problème. Les forêts aléatoires se basent sur les techniques de *bagging* (*bootstrap aggregating*) et d'échantillonnage de sous-espace aléatoire (*feature bagging*) pour construire une collection d'arbres de décisions. Le *bagging* consiste à utiliser un échantillon aléatoire différent du jeu de données total pour l'apprentissage de chaque arbre de décision. De façon analogue, le *feature bagging* utilise un sous

ensemble aléatoire des caractéristiques des observations pour l'entraînement des arbres. Les arbres entraînés de cette façon sont donc tous légèrement différents les uns des autres. Individuellement, ces arbres sont des prédicteurs assez faibles mais la forêt aléatoire qui utilise la moyenne de leurs prédictions est capable de mieux généraliser qu'un arbre seul. La Figure 4.1 illustre la capacité d'une forêt à généraliser, en comparaison avec un arbre de décision simple ; on voit sur cette figure que la frontière de décision de la forêt aléatoire est plus "propre" que celle d'un arbre et que sa précision en classification est meilleure.

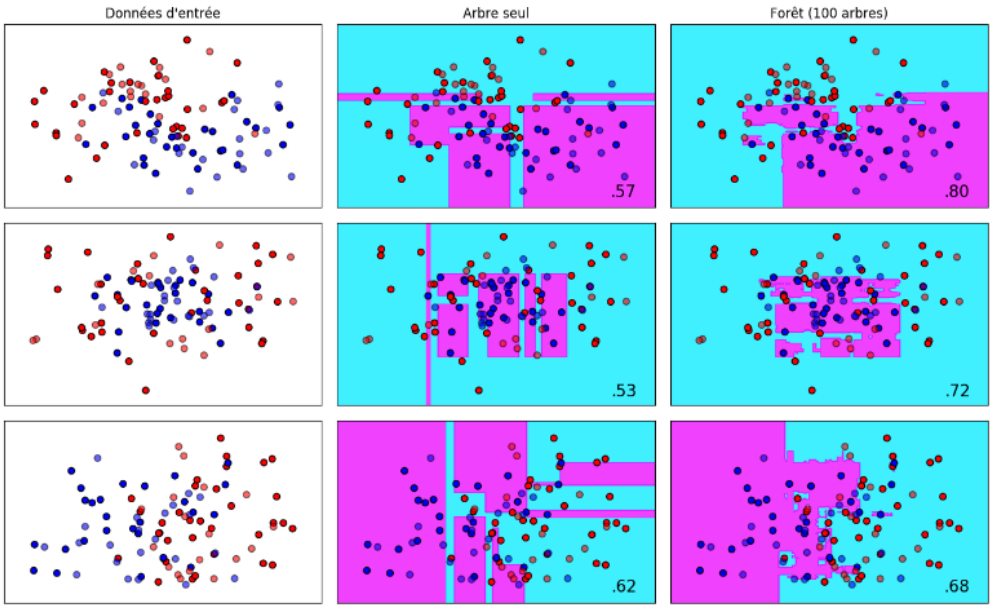


FIGURE 4.1 : Comparaison de la performance en classification d'un arbre de décision seul avec une forêt aléatoire de cent arbres sur des jeux de données synthétiques. Le numéro en bas à droite représente le pourcentage de précision de classification sur les données test. Les données test sont représentées avec des points de couleur atténuée. On voit que la frontière de décision obtenue par un arbre de décision seul est moins précise que celle obtenue avec une forêt aléatoire. Source : *Support Vector Machine Hyperparameter Tuning - A Visual Guide*

L'algorithme des forêts aléatoires construit une collection de L arbres de décision indépendants $\{\mathbf{t}_i : \mathcal{X} \mapsto \mathcal{Y}\}_{i=1}^{L} \subset \mathcal{T}$ pour réaliser la tâche de prédire la valeur correcte $\mathbf{y} \in \mathcal{Y}$ à partir d'une observation $\mathbf{x} \in \mathcal{X}$ donnée. On note $\mathbf{t}(\mathbf{x})$ le vecteur de tous les votes de l'arbre tel que $\mathbf{t}(\mathbf{x}) := [\mathbf{t}_1(\mathbf{x}) \dots \mathbf{t}_L(\mathbf{x})]^T$. L'estimateur de la forêt aléatoire $\mathbf{f} : \mathcal{T} \times \mathcal{X} \to \mathcal{Y}$ combine une fonction d'activation $\mathbf{a}$ et une combinaison linéaire des votes exprimés par les arbres de décision sous-jacents : $\mathbf{f}\left(\{\mathbf{t}_i\}_{i=1}^{L}, \cdot\right) = \mathbf{a}\left(\mathbf{w}^T\mathbf{t}(\cdot)\right)$. La fonction $\mathbf{a}$ est par défaut la fonction d'identité pour la régression ($\mathcal{Y} := \mathbb{R}$) et la fonction $sign$ pour la classification binaire

($\mathcal{Y} := \{-1, +1\}$). Les poids sont fixés tels que pour tout i, $\mathbf{w}_i := \frac{1}{L}$ car les importances de tous les arbres sont généralement considérées comme égales, même si des valeurs arbitraires pourraient être utilisées.

4.3 Élagage de forêts aléatoires

Dans les applications du monde réel, lorsque les données sont en grande dimension, les forêts aléatoires peuvent nécessiter un grand nombre d'arbres de décision pour atteindre de bonnes performances. De ce fait, même si chaque arbre individuellement est relativement peu cher à exécuter, l'ensemble d'arbres peut rapidement devenir lourd à mettre en œuvre. Ce constat a conduit la communauté scientifique à chercher des moyens de réduire le nombre d'arbres d'une forêt sans en impacter trop la performance.

BREIMAN, 2001 a montré qu'un grand nombre d'arbres dans une forêt aléatoire est *suffisant* pour obtenir une borne sur l'erreur de généralisation du modèle. Pourtant, il n'est pas sûr qu'avoir un grand nombre d'arbres soit *nécessaire* pour obtenir de bonnes performances. Au contraire, les arbres étant construits à partir d'échantillons d'apprentissage sélectionnés aléatoirement, on peut assez intuitivement se dire qu'il doit exister une certaine redondance entre les arbres d'une forêt et donc qu'il est peut-être possible d'en supprimer certains sans pour autant dégrader la performance.

Nous pouvons définir la fonction de décision de la forêt aléatoire élaguée telle que :

$$\mathbf{f}_K\left(\{\mathbf{t}_i\}_{i=1}^{L}, \cdot\right) := \sigma\left(\mathbf{w}^T \mathbf{t}(\cdot)\right), s.t.\ \|\mathbf{w}\|_0 = K, \tag{4.1}$$

avec K la taille de la forêt élaguée. Ainsi, le problème de l'élagage de la forêt peut se résumer à trouver les indices des K valeurs optimales non nulles dans le vecteur de poids $\mathbf{w}$.

KULKARNI et SINHA, 2012 proposent de classifier les méthodes d'élagage des forêts dans deux catégories différentes appelées «élagage statique» et «élagage dynamique» :

Élagage dynamique. Les techniques d'élagage dynamique construisent la forêt élaguée directement : elles ne nécessitent pas la construction préalable d'un grand ensemble de L arbres mais créent plutôt la forêt élaguée de taille K à partir de zéro en ajoutant à chaque étape un nouvel arbre s'il satisfait un certain critère qui prend en compte l'état actuel de la forêt (KULKARNI et SINHA, 2012 ; BERNARD, ADAM et HEUTTE, 2012).

Élagage statique. Les approches d'élagage statique sont basées sur le paradigme «Surproduction puis choix» (*Overproduce and choose*). Les techniques de

cette famille commencent avec un grand ensemble d'arbres L et construisent ensuite la forêt de taille $K \ll L$ en utilisant un processus itératif qui optimise un critère jusqu'à ce que la taille souhaitée soit atteinte. Les techniques sont classées comme «en avant» (*forward*), si la forêt finale est construite à partir d'un ensemble vide en y ajoutant itérativement de nouveaux arbres, ou «en arrière» (*backward*) si la forêt élaguée est obtenue en enlevant les arbres les uns après les autres de l'ensemble initial.

Bien que les techniques d'élagage dynamique atteignent l'objectif final de produire une forêt de taille limitée, elles ne procèdent pas à une sélection d'arbres à proprement parler et donc nous nous concentrons sur la seconde famille de méthodes d'élagage dans la suite de cet état de l'art.

Les approches d'élagage statique sont des algorithmes itératifs qui utilisent un critère d'évaluation sur les arbres à chaque étape afin de sélectionner les arbres qui doivent être ajoutés ou retirés de la forêt. La différence essentielle entre les techniques réside dans le choix du critère susmentionné. Ces critères s'appuient essentiellement sur deux caractéristiques que les arbres individuels de la forêt aléatoire sont censés intégrer : une performance *prédictive* élevée ainsi qu'une décorrélation mutuelle significative, également connue sous le nom de *diversité* dans les méthodes d'apprentissage d'ensemble (Brown et Kuncheva, 2010). Yang, Lu, Luo et al., 2012 et Zhang et Wang, 2009 proposent de supprimer les arbres de la forêt qui ont le plus faible impact sur la marge – *i.e.* la confiance de la forêt dans ses classifications correctes – et la précision de la forêt, respectivement. C'est-à-dire qu'ils se concentrent essentiellement sur la performance de prédiction des arbres de la forêt. A l'inverse, Caruana, Niculescu-Mizil, Crew et al., 2004 proposent de faire une sélection positive des arbres avec remplacement, de façon à améliorer la précision de prédiction de la forêt à chaque pas de temps. Enfin, Hu, Yu, Xie et al., 2007 effectuent également une sélection basée sur la précision tant que l'ajout de nouveau arbres ne dégrade pas la précision sur un jeu de données de validation. Ces méthodes reposent sur l'intuition selon laquelle élaguer sur la base de la précision devrait donner de bonnes performances mais, comme indiqué ci-dessus, on sait que la diversité entre prédicteurs est également une caractéristique essentielle des modèles d'ensemble (Ali et Pazzani, 1995). Suivant ce cheminement de pensée, Zhang et Wang, 2009 proposent aussi de supprimer les arbres présentant une corrélation moyenne maximale avec les autres arbres de la forêt, ce qui augmenterait leur diversité globale. Cependant, on peut penser à combiner les deux critères ensemble – précision et diversité – afin d'obtenir le meilleur des deux mondes. C'est l'approche adoptée par Fawagreh, Gaber et Elyan, 2016, où la diversité entre les classificateurs est obtenue au moyen de la technique des K-moyennes (évoquée en Section 3.3.2) et Rokach, 2009, où une mesure d'information mutuelle est utilisée pour combiner la précision des arbres avec l'accord mutuel entre les arbres pour la sélection.

4.4 Procédure de sélection des arbres basée sur OMP

Algorithme 4 Algorithme OMP (PATI, REZAIIFAR et KRISHNAPRASAD, 1993)

Entrée : $K, \mathbf{Y}, \mathbf{T} := [\mathbf{t}_1(\mathbf{X}), \ldots, \mathbf{t}_L(\mathbf{X})]$

Sortie : $\mathbf{w}$, une solution approximée du problème (4.3)

1: $(\mathbf{r}_0, \mathbf{w}_0, \lambda) = (\mathbf{Y}, 0, \emptyset)$
2: **pour** $\tau = 0, .., K$ **faire**
3: $i^* \in \underset{i \in \{1,\ldots,L\}}{\operatorname{argmax}} \; |\langle \mathbf{t}_i(\mathbf{X}), \mathbf{r}_\tau \rangle|$
4: $\lambda_{\tau+1} = \lambda_\tau \cup \{(\mathbf{t}_{i^*}(\mathbf{X}))\}$
5: $\mathbf{w}_{\tau+1} \in \underset{\substack{\mathbf{w} \in \mathbb{R}^L \text{ s.t.} \\ \mathbf{Tw} \in \operatorname{span}(\lambda_{\tau+1})}}{\operatorname{argmin}} ||\mathbf{Y} - \mathbf{Tw}||_2^2$
6: $\mathbf{r}_{\tau+1} = \mathbf{Y} - \mathbf{Tw}_{\tau+1}$
7: **fin pour**
8: **retourne** $\mathbf{w}_{\tau+1}$

Soit $\mathbf{X} \in \mathbb{R}^{N \times D}$ la matrice des observations empilées et $\mathbf{Y} \in \mathbb{R}^N$ le vecteur des étiquettes correspondantes. Soit $\mathbf{t}_i(\mathbf{X}) := [\mathbf{t}_i(\mathbf{x}_1) \ldots \mathbf{t}_i(\mathbf{x}_N)]$ le vecteur de prédiction de l'arbre $\mathbf{t}_i$ sur l'ensemble des données $\mathbf{X}$. Ainsi, la matrice des prédictions d'un ensemble d'arbres $\{\mathbf{t}_i\}_{i=1}^L$ pour cet ensemble de données peut être définie telle que :

$$\mathbf{T} := [\mathbf{t}_1(\mathbf{X}), \ldots, \mathbf{t}_L(\mathbf{X})] \,. \tag{4.2}$$

Comme expliqué précédemment, une forêt aléatoire produit sa décision sur la base d'une combinaison linéaire de votes ; ainsi, les étiquettes prédites associées à $\mathbf{X}$ sont $\hat{\mathbf{Y}} = \mathbf{a}\,(\mathbf{Tw})$.

Dans ces travaux, nous proposons de réaliser l'élagage des forêts aléatoires sur la base de la performance des arbres sur le jeu de données d'entrainement. Nous utilisons le critère d'erreur quadratique moyenne : $\|\hat{\mathbf{Y}} - \mathbf{Y}\|^2$. Sans perdre de généralité, nous abandonnons la fonction σ de l'Équation (4.1) pour simplifier le problème d'élagage :

$$\underset{\mathbf{w} \in \mathbb{R}^L}{\arg\min} \; \|\mathbf{Tw} - \mathbf{Y}\|^2 \,, \text{s.t. } \|\mathbf{w}\|_0 \leq K \tag{4.3}$$

où la contrainte L_0 permet d'atteindre l'objectif d'élagage, c'est-à-dire que les K éléments non-nuls de $\mathbf{w}$ indiquent les arbres à garder dans la forêt élaguée. Comme le problème de l'Équation (4.3) est connu pour être NP-difficile (DAVIS, MALLAT et AVELLANEDA, 1997), nous proposons d'utiliser l'algorithme OMP de PATI, REZAIIFAR et KRISHNAPRASAD, 1993 qui est l'un des algorithmes les plus

adaptés à cette tâche.

Le problème de l'Équation (4.3) a attiré beaucoup d'attention et est au centre de nombreux problèmes de recherche sur le traitement du signal. En effet, quelques années seulement après que DAVIS, MALLAT et AVELLANEDA, 1997 aient introduit ce problème, de nombreuses techniques différentes ont été proposées pour lui donner une solution approximative. Parmi elles, deux algorithmes gourmands (*greedy*) se sont distingués : l'algorithme de *Matching Pursuit* (MP) déjà proposé par DAVIS, MALLAT et AVELLANEDA, 1997 et son homologue orthogonal, OMP PATI, REZAIIFAR et KRISHNAPRASAD, 1993. Ces deux algorithmes sont devenus célèbres pour leur forme très simple et intuitive mais aussi pour leurs fortes garanties théoriques (TROPP, 2004 ; GRIBONVAL et VANDERGHEYNST, 2006). Dans ce travail, nous concentrons notre attention sur l'algorithme OMP qui s'est avéré meilleur que MP dans de nombreux cas et dont la propriété d'orthogonalité apporte une forme d'interprétabilité à la méthode. Dans ce qui suit, nous utilisons la terminologie du traitement du signal et appelons $\mathbf{T}$ un «dictionnaire» et ses colonnes des «atomes». Nous notons que les atomes doivent être normalisés pour que l'algorithme OMP fonctionne correctement ; ceci pourra être fait en divisant chaque colonne de $\mathbf{T}$ par sa norme. Le vecteur des normes des prédictions des arbres est noté $\mathbf{n}$, il sera à nouveau utilisé lors de l'étape d'inférence de la forêt élaguée.

L'algorithme OMP est un algorithme itératif qui sélectionne un nouvel atome à chaque pas de temps pour raffiner sa solution. Pour un pas de temps donné $\tau \geq 0$, nous indiquons par $\mathbf{w}_\tau$ la solution actuelle et par $\mathbf{r}_\tau$ le résidu :

$$\mathbf{r}_\tau = \mathbf{Y} - \mathbf{T}\mathbf{w}_\tau. \tag{4.4}$$

OMP sélectionne l'atome qui forme le plus grand produit scalaire avec le résidu $\mathbf{r}_\tau$ et projette ensuite le vecteur cible $\mathbf{y}$ sur l'ensemble des atomes sélectionnés jusqu'alors, d'où l'intérêt de normaliser les atomes afin que seul l'angle des vecteurs ne rentre en considération pour le choix. Ces étapes sont répétées jusqu'à ce que le nombre d'atomes souhaité soit obtenu. La procédure globale est résumée dans l'Algorithme 4.

Grossièrement, l'algorithme OMP augmente d'un élément l'ensemble des indices non nuls de $\mathbf{w}$ à chaque pas de temps et recalcule ensuite les meilleurs coefficients de la combinaison linéaire. Nous remarquons ici que le résidu est toujours orthogonal à tous les atomes sélectionnés et donc que le prochain atome choisi devrait être un atome quelque peu différent des précédents. Dans le contexte de l'élagage des forêts, cette dernière remarque fournit une justification intuitive supplémentaire à pourquoi l'OMP est bénéfique pour cette tâche : non seulement l'algorithme se concentre sur la recherche d'une combinaison linéaire d'arbres qui se rapproche bien des véritables étiquettes, mais aussi il évite que deux arbres identiques soient sélectionnés, encourageant ainsi la diversité entre les arbres.

Une fois que tous les atomes – les arbres – ont été sélectionnés, l'étape d'in-

férence se déroule en faisant la combinaison linéaire des prédictions des arbres de la forêt suivant les coefficients obtenus dans $\mathbf{w}$. Attention toutefois, les coefficients ayant été obtenus sur les prédictions normalisés, il faudra, pour chaque observation future, diviser à nouveau les prédictions des arbres par le même vecteur des normes $\mathbf{n}$ obtenu lors de la phase d'entraînement.

La procédure complète d'entraînement de l'algorithme `OMPForest` est résumée dans l'Algorithme 5. Cette procédure est assez simple et nous ne proposons pas de quelconque garanties théoriques pour l'accompagner. L'essentiel de ce travail repose sur l'évaluation empirique de la procédure et sur une heuristique que nous proposons pour en améliorer les performances : imposer une contrainte de non-négativité sur les coefficients de $\mathbf{w}$. Cette analyse est conduite dans la section suivante.

Algorithme 5 Algorithme `OMPForest`

Entrée : $K, \mathbf{X}, \mathbf{Y}$

1: $\{\mathbf{t}_i\}_{i=1}^{L} \leftarrow$ `ForetAleatoire` $(\mathbf{X}, \mathbf{Y})$
2: $\mathbf{T} \leftarrow [\mathbf{t}_1(\mathbf{X}), \ldots, \mathbf{t}_L(\mathbf{X})]$
3: $\mathbf{n} \leftarrow$ `NormeColonnes` $(\mathbf{T})$
4: $\mathbf{T} \leftarrow \mathbf{T}/\mathbf{n}$
5: $\mathbf{w} \leftarrow$ `OMP` $(K, \mathbf{Y}, \mathbf{T})$
6: **retourne** $\mathbf{w}$, $\{\mathbf{t}_i\}_{i=1}^{L}$, $\mathbf{n}$

4.5 Résultats expérimentaux

Dans cette section, nous commençons par décrire le cadre expérimental, puis nous décrivons nos résultats sur de nombreux ensembles de données en accordant une attention particulière à l'impact des pondérations des arbres sur les performances. Nous montrons que notre méthode, et en particulier sa capacité à apprendre des poids pour les arbres, est très performante dans certains cas, mais qu'elle souffre parfois d'un sur-apprentissage préjudiciable. Nous remarquons une association entre la négativité des poids et les mauvaises performances, ce qui nous conduit à proposer une variante de notre algorithme qui introduit une contrainte non-négative sur les poids des arbres de la forêt. Cette heuristique permet à notre technique non seulement d'obtenir des performances encore meilleures que la version standard d'`OMP`, mais elle fournit également un critère d'arrêt pour le choix de la taille de la forêt élaguée.

4.5.1 Paramètres expérimentaux

Jeux de données. Des expériences ont été menées sur 11 jeux de données pour la régression ou la classification binaire. Un résumé des caractéristiques

Jeu de données	Acronyme	Taille	Dimension	Tâche (mesure du score)	Taille de forêt initiale
California housing (Pace et Barry, 1997)	C.H.	20 640	8	Régression (MSE)	1000
Kin8nm (Corke, 1996)	Kin.	8 192	8	Régression (MSE)	1000
Diamonds (Agrawal, 2017)	Diam.	53 940	9	Régression (MSE)	429
Diabetes (Efron, Hastie, Johnstone et al., 2004)	Diab.	442	10	Régression (MSE)	108
Boston house prices (Belsley, Kuh et Welsch, 1980)	Bos.	506	13	Régression (MSE)	100
Gamma (Bock, Chilingarian, Gaug et al., 2004)	Gam.	19 020	11	Classification binaire (% de précision)	100
Breast cancer (Mangasarian, Street et Wolberg, 1995)	B.C.	569	30	Classification binaire (% de précision)	1000
Steel Plates (Buscema, 1998)	St. P.	1941	33	Classification binaire (% de précision)	1000
King-Rook vs. King-Pawn (Shapiro, 1987)	KR-KP	3 196	36	Classification binaire (% de précision)	1000
Spambase (Cranor et LaMacchia, 1998)	Sp. B.	4 601	57	Classification binaire (% de précision)	1000
LFW pairs (Huang, Mattar, Berg et al., 2008)	LFW	13 233	5 828	Classification binaire (% de précision)	1000

Table 4.1 : Caractéristiques des jeux de données. Nous avons utilisé un partitionnement de 60-20-20% pour les jeux de données d'entraînement, validation et test.

des jeux de données utilisés est disponible dans le Tableau 4.1 avec la taille de la forêt de base pour chaque ensemble de données.

Méthodes concurrentes. Nous comparons notre technique à de multiples méthodes d'élagage de forêts aléatoires. En suivant la classification présentée dans la Section 4.3, nous utilisons la méthode dite «`Ensemble`» de Caruana, Niculescu-Mizil, Crew et al., 2004 comme un élagage «en avant», basé sur la précision ; «`Zhang Prédictions`» de Zhang et Wang, 2009 comme un élagage «en arrière», basé sur la précision ; «`Zhang Similarité`» de Zhang et Wang, 2009 comme un élagage «en arrière», basé sur la diversité ; et enfin «`Kmeans`» de Fawagreh, Gaber et Elyan, 2016 qui est de type «en avant» et une approche basée à la fois sur la précision et la diversité. Nous comparons également nos résultats à une technique de sélection «`Random`» qui correspond à un sélection aléatoire des arbres dans la forêt, ce qui équivaut à construire directement une forêt de la taille souhaitée. Enfin, pour les méthodes à base de `OMP`, nous comparons la performance des forêts élaguées avec ou sans les poids associés aux arbres.

Détails expérimentaux. Tous les algorithmes ont été implémentés en Python, en utilisant la bibliothèque «`scikit-learn`» (Buitinck, Louppe, Blondel et al., 2013) pour la forêt aléatoire et l'algorithme `OMP`. Le code des expériences et de génération des figures est disponible publiquement dans les dépots du laboratoire (Lamothe, Bouscarat, Giffon et al., 2020). Nous avons utilisé un partitionnement de 60-20-20% pour les données d'apprentissage, validation et test. Dans un premier temps, nous avons utilisé le jeu d'apprentissage et le jeu de validation séparemment pour faire la recherche d'hyper-paramètres (profondeur de la forêt, nombre d'arbre optimal) de la forêt aléatoire de base. Afin d'accélérer le processus de recherche des hyper-paramètres, nous avons utilisé la biliothèque «`skopt`» d'optimisation bayésienne. Les données d'apprentissage et de validation ont ensuite été rassemblées pour ré-apprendre entièrement la forêt avec les bons

Tâches de Régression :

	Diamonds		Diabetes		Kin8nm		California Housing		Boston	
Ensemble	3.032E+05	86	3.431E+03	32	1.892E-02	200	2.187E-01	267	1.267E+01	30
Kmeans	3.024E+05	143	3.281E+03	36	*2.024E-02*	*33*	*2.449E-01*	*33*	*1.278E+01*	*13*
NN-OMP w/o weights	**3.024E+05**	**86**	3.317E+03	36	1.921E-02	133	2.239E-01	100	**1.214E+01**	**33**
NN-OMP	3.033E+05	86	3.549E+03	36	1.809E-02	133	**2.180E-01**	**133**	1.253E+01	33
OMP w/o weights	3.025E+05	143	3.324E+03	36	1.931E-02	67	*2.267E-01*	*33*	1.247E+01	27
OMP	*3.087E+05*	*29*	*3.607E+03*	*25*	**1.776E-02**	**333**	2.197E-01	133	*1.293E+01*	*13*
Random	3.025E+05	114	3.303E+03	32	2.002E-02	333	2.390E-01	333	1.253E+01	33
Zhang Predictions	3.047E+05	143	3.282E+03	36	2.089E-02	333	2.536E-01	333	1.430E+01	33
Zhang Similarities	3.032E+05	143	**3.241E+03**	**32**	2.017E-02	333	2.452E-01	333	1.283E+01	33

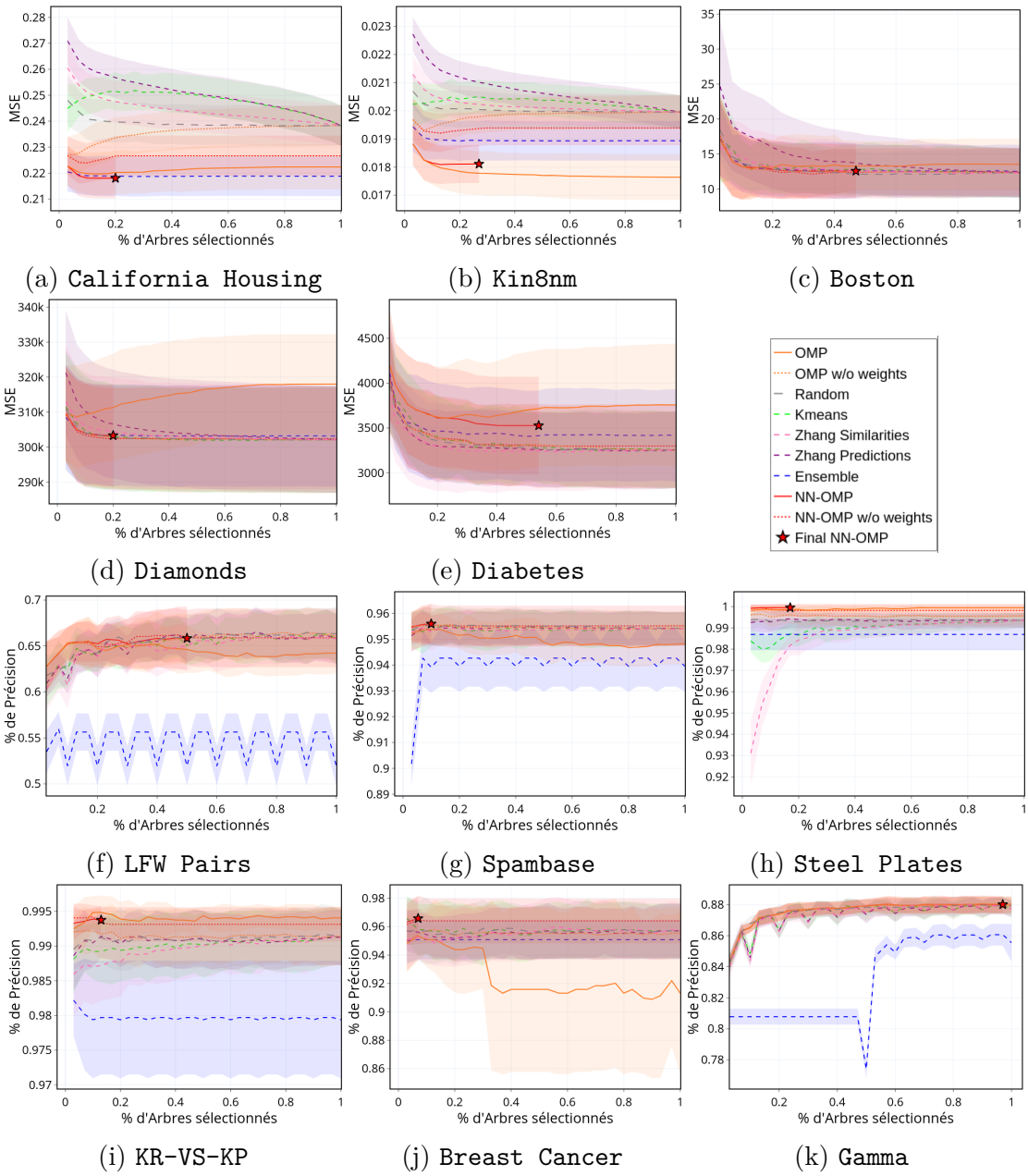
Tâches de Classification :

	Spambase		Steel Plates		KR-VS-KP		Breast Cancer		LFW Pairs		Gamma	
Ensemble	94.27%	133	98.69%	233	*98.22%*	*33*	95.09%	100	*56.00%*	*67*	*80.78%*	*3*
Kmeans	95.52%	167	99.05%	267	99.00%	333	***96.58%***	***33***	65.25%	333	87.68%	33
NN-OMP w/o weights	*95.57%*	*100*	*99.95%*	*67*	99.42%	100	96.49%	67	**66.02%**	**333**	87.75%	33
NN-OMP	*95.59%*	*100*	**99.95%**	**100**	99.39%	100	**96.58%**	**67**	65.73%	233	87.75%	33
OMP w/o weights	95.56%	167	*99.64%*	*67*	99.22%	100	95.79%	133	65.32%	133	87.75%	33
OMP	95.39%	133	99.90%	333	**99.48%**	**100**	95.35%	67	65.55%	167	87.75%	33
Random	**95.59%**	**167**	*99.41%*	*67*	99.14%	267	95.88%	300	65.98%	267	**87.76%**	**33**
Zhang Predictions	95.45%	333	99.43%	167	99.14%	133	*95.70%*	*33*	65.43%	333	87.72%	33
Zhang Similarities	95.46%	167	98.92%	300	98.94%	333	95.61%	333	65.27%	333	87.68%	33

TABLE 4.2 : Tableaux représentant la meilleure performance atteinte par chaque modèle ainsi que le nombre d'arbres requis pour cette performance. La taille maximale considérée de la forêt élaguée est de 10% de la taille de la forêt aléatoire initiale. Le meilleur résultat sur chaque tâche est **en gras**, le deuxième meilleur est souligné. Le plus petit nombre d'arbres est écrit en *italique*. La co-localisation de l'italique et du gras ou du souligné signifie que la technique permet d'obtenir d'excellentes performances avec le plus petit nombre d'arbres.

hyper-paramètres. Le même jeu de données (apprentissage plus validation) a été utilisé pour faire la sélection d'arbres. Cela semble contre-intuitif de ne pas utiliser les données de validation séparément pour la sélection des arbres, mais nos expériences ont montré que, pour toutes les techniques, les meilleurs résultats ont été obtenus lorsque toutes les données disponibles étaient utilisées pour la sélection. Dans toutes les expériences menées, la moyenne des résultats a été calculée sur 10 réplicats.

4.5.2 Évaluation de la méthode d'élagage basée sur OMP



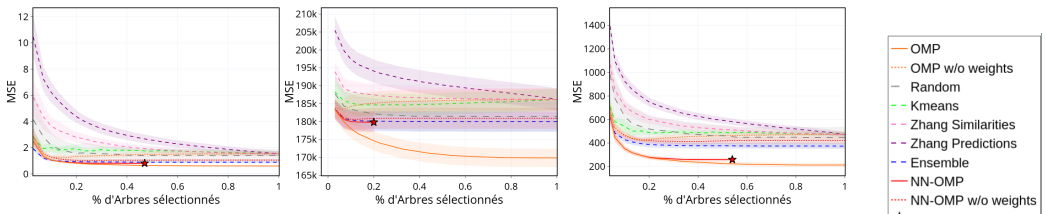
(a) California Housing (b) Kin8nm (c) Boston

(d) Diamonds (e) Diabetes

(f) LFW Pairs (g) Spambase (h) Steel Plates

(i) KR-VS-KP (j) Breast Cancer (k) Gamma

FIGURE 4.2 : Résultats sur les données de **test** des différentes méthodes en fonction du nombre d'arbres sélectionnés. Nous montrons un graphique pour chaque jeu de données testé. La moyenne des résultats est calculée sur 10 réplicats et la zone de couleur claire autour des lignes indique l'écart type. Voir le Tableau 4.1 pour la taille de la forêt de base.

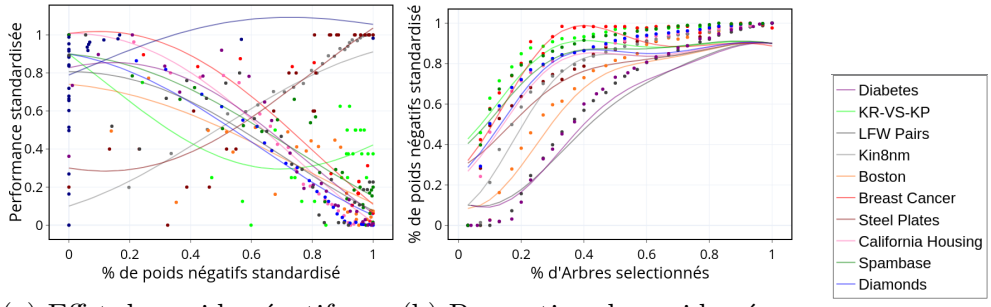


(a) `Boston` (b) `Diamonds` (c) `Diabetes`

FIGURE 4.3 : Résultats sur les données de **validation** des différentes méthodes en fonction du nombre d'arbres sélectionnés. Nous présentons un graphique pour chaque jeu de données qui a donné de mauvais résultats dans la Figure 4.2. La moyenne des résultats est calculée sur 10 réplicats et la zone de couleur claire autour des lignes indique l'écart type.

Les résultats donnés dans la Figure 4.2 comparent les performances des méthodes d'élagage basées sur `OMP` à celles d'autres techniques. Nous constatons tout d'abord que sur certains jeux de données, à savoir `California Housing`, `Kin8nm`, `Steel Plates` et `KR-VS-VP`, la méthode `OMP` surpasse clairement la plupart des autres méthodes, atteignant à leur valeur maximale des performances encore meilleures que la forêt entière. Sur ces ensembles de données, nous voyons aussi clairement l'avantage de l'apprentissage des poids pour les arbres car la version pondérée de `OMP` a une meilleure performance que son homologue non pondérée. La méthode `Ensemble` est en concurrence avec `OMP` et donne d'excellents résultats en matière de régression, comme par exemple sur `California housing`. Nous remarquons que la méthode `Ensemble` est autorisée à prendre plusieurs fois le même arbre dans la forêt, ce qui équivaut à donner un poids plus important à cet arbre ; cela souligne à nouveau l'intérêt d'utiliser des poids pour les arbres de la forêt élaguée. Cependant, cette méthode `Ensemble` est extrêmement peu performante pour les tâches de classification binaire par rapport aux autres méthodes. Les méthodes dites `Zhang` et `K-moyennes`, bien que stables en général, ont du mal à être ne serait-ce qu'aussi performantes que la sélection aléatoire des arbres dans la forêt et sont donc moins performantes que la technique d'élagage basée sur `OMP`.

Sur les autres jeux de données, la version non pondérée de notre technique fonctionne toujours *au moins* aussi bien que les autres techniques mais dans certains cas, à savoir sur les jeux de données `Diamonds`, `Boston`, `Diabetes` et `Breast-cancer`, les poids ont un effet néfaste incontestable sur les performances. Nous remarquons tout d'abord que, dans ces cas, aucune technique ne semble pouvoir donner un résultat significativement meilleur qu'une simple sélection aléatoire d'arbres. Deuxièmement, ces trois ensembles de données sont de très petite taille et nous pensons que cela les rend vulnérables au sur-apprentissage des poids, comme l'illustre la Figure 4.3 qui montre que la forêt pondérée se



(a) Effet des poids négatifs sur la performance

(b) Proportion des poids négatifs

FIGURE 4.4 : Relation entre la taille des forêts, la proportion de poids négatifs et les performances. Dans la Figure 4.4a, seules les forêts élaguées de taille supérieure à 10% de la taille de la forêt de base sont prises en compte. L'inverse de la mesure MSE est prise comme mesure de performance pour les jeux de données de régression, de sorte qu'une valeur plus élevée signifie une meilleure performance. Les valeurs de performance et les pourcentages de poids négatifs sont normalisés en utilisant l'échelle min-max ($f(x) = (x - min)/(max - min)$) de sorte que toutes les valeurs soient comprises entre 0 et 1, rendant les différents jeux de données comparables sur une seule échelle de mesure. Les résultats ne sont pas représentés pour l'ensemble de données `Gamma` car il ne présentait pas de poids négatifs. Les courbes sont utilisées pour faciliter la lecture des figures et ont été obtenues en entraînant pour chaque ensemble de données un SVM avec un noyau rbf et une valeur gamma égale à 1 sur les observations brutes (les points).

comporte très bien sur les données de validation, utilisées pour la sélection des arbres.

4.5.3 Effet d'une contrainte de non-négativité sur la pondération des arbres

Même si la technique d'élagage basée sur `OMP` a souvent d'excellentes performances lorsqu'il s'agit de sélectionner un petit nombre d'arbres, elle donne également de mauvais résultats lorsque le nombre d'arbres sélectionnés augmente. Cela est assez problématique car aucune indication particulière n'est donnée pendant la phase de sélection des arbres qui nous permettrait de concevoir un critère d'arrêt précoce pour fixer la taille maximale de la forêt élaguée. En effet, la Figure 4.3 montre que la mesure de performance sur les données de validation ne semble pas atteindre un plateau particulier qui pourrait servir d'indicateur de fin d'apprentissage.

Encouragés par la bonne performance globale de la version non pondérée de la sélection par OMP, nous étudions les valeurs de pondération fournies par OMP (plutôt que les arbres eux-mêmes) afin d'expliquer le mauvais comportement de notre technique. Nous constatons, comme le montre la Figure 4.4b que la *proportion* des arbres ayant un poids négatif augmente avec le nombre d'arbres sélectionnés. Ce comportement a certainement un effet néfaste sur les performances de la forêt élaguée et nous fournissons une explication intuitive pour la classification binaire où les étiquettes sont $\{-1, +1\}$; le cheminement de pensée est cependant facilement transférable à la régression.

Il est raisonnable de penser que n'importe lequel des arbres pré-entrainés de la forêt aléatoire de base devrait avoir une performance *au moins* meilleure que celle du hasard, même marginalement, ce qui signifie que la précision moyenne de ces arbres devrait être supérieure à 0.5 dans le cas de la classification binaire. Dans ce cas, combiner linéairement des arbres avec des poids négatifs équivaudrait à inverser le vote de ces arbres, à en faire délibérément de *mauvais* classificateur et à les utiliser pour le vote final de la forêt élaguée. Ce comportement contre-intuitif s'explique principalement par le fait que OMP choisit des poids négatifs afin de correspondre parfaitement aux étiquettes de l'ensemble d'entraînement. Cette analyse est corroborée par la Figure 4.4a où l'on voit clairement que la performance a tendance à diminuer lorsque la proportion de poids négatifs augmente. Cependant, nous devons noter que cette tendance n'est clairement pas observée sur les jeux de données Kin8nm et LFW. Notez que seules les forêts élaguées dont la taille est supérieure à 10% de la taille de la forêt de base sont représentées ici. Nous avons choisi de ne rapporter que ces points car les observations survenant avant ce seuil ont naturellement de mauvaises performances puisqu'elles correspondent à des forêts si petites qu'elles n'auraient de toute façon pas de bonnes performances (Voir Figure 4.2).

Cette analyse nous amène à la conclusion suivante : l'utilisation d'une version non négative de OMP (Bruckstein, Elad et Zibulevsky, 2008) devrait permettre d'éviter le sur-apprentissage. La version non négative d'OMP (NN-OMP) est similaire à celle décrite dans l'Algorithme 4 sauf à la Ligne 3 où seuls les atomes positivement corrélés peuvent dorénavant être sélectionnés et à la Ligne 5 où une régression non négative par les moindres carrés est effectuée pour obtenir uniquement des coefficients non négatifs. Nous remarquons que le fait de ne sélectionner que des atomes positivement corrélés rendra éventuellement NN-OMP incapable d'en sélectionner de nouveaux et empêchera donc le sur-apprentissage en limitant la taille maximale de la forêt élaguée. Cette contrainte fournit ainsi un critère d'arrêt précoce pour la sélection des arbres de la forêt qui permet de limiter la taille de la forêt élaguée. Les résultats obtenus avec cet algorithme sont illustrés en Figure 4.2 où nous voyons clairement l'avantage de notre méthode en terme de performance. Dans cette figure, l'étoile représente le point où la méthode NN-omp ne pouvait plus sélectionner de nouvel arbre sans rompre la contrainte de non-négativité et donc où la taille de la forêt élaguée est maximale.

Nous voyons que ce point arrive parfois très tôt ce qui illustre la capacité de notre modèle à contruire des forêts efficaces avec très peu d'arbres. Une description générale des résultats est disponible dans le Tableau 4.2 où sont indiquées les meilleures performances obtenues par chaque méthode, ainsi que le nombre d'arbres correspondant. Nous voyons dans ce tableau que les techniques se basant sur OMP obtiennent quasiment toujours la meilleure et/ou seconde meilleure performance, en particulier sur les jeux de données de classification binaire.

4.6 Conclusion

Nous avons présenté une méthode originale pour réduire la taille des forêts aléatoires. Notre méthode utilise un algorithme glouton, à savoir l'Orthogonal Matching Pursuit, sur les prédictions de l'arbre de chaque forêt pour sélectionner un sous-ensemble d'arbres dont la combinaison linéaire des prédictions se rapproche le plus du véritable vecteur des prédictions. Notre méthode ne se contente pas de sélectionner un sous-ensemble d'arbres, mais associe également un coefficient à chaque arbre sélectionné. Ces poids s'avèrent bénéfiques dans certains cas mais peuvent également conduire à un sur-entraînement dans certains autres scénarios, comme le montre nos expériences. Ces résultats expérimentaux nous ont conduit à proposer une variante non-négative de la méthode qui s'avère plus efficace encore et semble apporter de meilleures performances que toutes les autres techniques d'élagage de forêts aléatoires connues.

Les résultats empiriques obtenus sont très encourageants : la technique est simple mais permet pourtant d'obtenir d'excellentes performances avec de très forts taux de compression. Toutefois, même si nous proposons quelques intuitions, nous avons failli pour l'instant à apporter une réelle justification théorique au bon fonctionnement de la méthode.

Conclusion

Dans cette thèse, nous avons présenté de nouvelles approches pour la compression de modèles d'apprentissage automatique en utilisant des méthodes à noyaux ou des méthodes d'approximation parcimonieuses. Nous les résumons maintenant et proposons quelques perspectives pour étendre leur portée.

Dans le Chapitre 2, nous proposons de remplacer les couches denses d'un réseau neuronal par une couche que nous appelons «Nyström adaptatif». Cette couche utilise un sous-ensemble d'apprentissage comme points de références pour calculer, grâce à une fonction noyau à définir, une représentation non-linéaire des données au sortir des couches de convolutions. Nous avons montré par un série d'expériences que la couche Nyström adaptative contenait peu de paramètres, ce qui la rendait propice au traitement de jeux de données contenant peu d'observations étiquetées.

Une perspective possible de ce travail consisterait à appliquer cette couche au problème de l'adaptation de domaine semi-supervisé. Ce problème consiste en l'adaptation d'un modèle apprit sur un domaine d'entrée source à un domaine cible. Dans le cas semi-supervisé, les données annotées issues du domaine source sont disponibles en abondance alors que seulement quelques unes sont disponibles dans le domaine cible. Afin que la fonction noyau de la couche Nyström soit capable de reconnaître des similarités entre données issues de domaines différents, il serait certainement nécessaire de construire une représentation invariante entre les domaines. Pour répondre à ce problème, nous pourrions utiliser la méthode d'adaptation de domaine antagoniste (*adversarial*) proposée par GANIN, USTINOVA, AJAKAN et al., 2016. Dans ces travaux, les auteurs construisent un réseau de neurones capable de fournir une représentation des données remplissant deux fonctions : (I) elle permet de discriminer les différentes classes du domaine source et (II) elle est invariante en fonction du domaine des données considérées. Un tel réseau de neurones possède un module d'extraction de caractéristiques servant de base à deux modules de prédiction, l'un chargé de reconnaître les classes des données d'entrée et l'autre chargé d'identifier le domaine duquel proviennent les données traitées. Pour obtenir un module d'extraction de caractéristiques à la fois (I) discriminant et (II) invariant, ses poids sont optimisés de façon à minimiser l'erreur du module de prédiction de classe et, grâce à la technique de renversement du gradient (*gradient reversal*), maximiser l'erreur du module de prédiction de domaine, d'où la notion d'apprentissage «antagoniste». En utilisant la couche de Nyström adaptative, nous pourrions profiter de sa capacité à apprendre avec peu de données annotées pour ajouter un troisième

module de prédiction dédié à la classification des données du domaine cible, si les classes à prédire dans ce cas sont différentes de celles du domaine source.

Une autre perspective possible de ce travail consisterait à appliquer la couche Nyström à d'autres types de réseaux, traitant d'autres types de données, comme les réseaux de convolution sur graphe où nous pourrions profiter pleinement de la capacité des noyaux à traiter des données non vectorielles. Nous pensons notamment aux réseaux de convolution sur graphe qui construisent une représentation convolutive de chaque noeud du graphe avant de réaliser une opération de *readout* qui permet d'obtenir une représentation vectorielle globale du graphe. Habituellement, ce *readout* consiste en une simple somme de toutes les représentations des nœuds (WU, PAN, CHEN et al., 2020), qui détruit définitivement l'information du voisinage entre les nœuds. L'utilisation à cet endroit de la couche Nyström adaptative équipée d'un noyau sur graphe (VISHWANATHAN, SCHRAUDOLPH, KONDOR et al., 2010 ; SHERVASHIDZE, SCHWEITZER, VAN LEEUWEN et al., 2011) pourrait permettre d'obtenir une représentation vectorielle du graphe moins destructrice, conduisant peut-être à de meilleures performances dans la tâche de prédiction finale.

Un défaut important de la couche Nyström adaptative, cependant, est le fait qu'elle nécessite de garder en mémoire les observations de référence, ce qui contre balance la réduction du nombre de paramètres à stocker dans la couche elle-même.

Historiquement, ce dernier constat nous avait conduit à nous intéresser à la méthode de Nyström efficace de SI, HSIEH et DHILLON, 2016 afin de réduire l'empreinte mémoire des observations de référence à stocker. Ceci nous amène à la première contribution du Chapitre 3 où nous proposons un variant de la méthode des K-moyennes comme algorithme de regroupement, appelé `QK-means`. La méthode que nous proposons fournit une matrice de centroïdes exprimée sous la forme d'un produit de matrices parcimonieuses dont le nombre total de valeurs non-nulles est grandement inférieur au nombre de valeurs non nulles à stocker dans la matrice dense correspondante. Nous avons démontré dans une série d'expériences que l'intérêt de la méthode que nous proposons pouvait facilement se transférer à de nombreux cas d'application de l'algorithme des K-moyennes. Nous pensons que nous sommes loin d'avoir fait le tour de toutes les applications possibles de cette méthode et donc des travaux futurs pourraient simplement consister à y trouver d'autres cas d'utilisation. En outre, notre méthode repose sur la résolution d'un problème d'approximation d'une matrice par un produit de facteurs parcimonieux. Afin de résoudre ce problème, nous avons choisi d'utiliser l'algorithme `Palm4MSA` pour des raisons essentiellement historiques mais nous pensons qu'il y a un large espace pour travailler à trouver des alternatives à cet algorithme qui seraient soit plus rapides, soit meilleures en terme de qualité d'approximation. L'une des pistes de réflexion que nous avons envisagé repose notamment sur la reformulation du produit de facteurs parcimonieux de façon à obtenir un meilleur contrôle sur l'opération de projection dans `Palm4MSA`.

Dans la deuxième partie du Chapitre 3, nous comprimons un réseau neuronal en en approximant les couches par des produits de matrices parcimonieuses. Cette idée, simple à mettre en œuvre, nous a permit d'obtenir d'excellents taux de compression sur plusieurs architectures. Toutefois, dans son état actuel, nous avons failli à démontrer un intérêt supérieur à cette méthode comparé à certaines autres méthodes de l'état de l'art. Nous planifions d'essayer d'améliorer les performances de notre procédure, notamment en incorporant les activations des couches du réseau à la fonction objectif de `Palm4MSA` afin d'assurer que les activations du réseau comprimé imite au maximum les activations du réseau initial, de façon analogue à ce qui a été fait dans les travaux de ROMERO, BALLAS, KAHOU et al., 2015. Dans l'optique d'intégrer les factorisations parcimonieuses aux réseaux neuronaux, la méthode `QK-means` pourrait être utilisée dans le cadre de la compression des réseaux neuronaux par quantification. Par exemple, les travaux de STOCK, JOULIN, GRIBONVAL et al., 2020 pourraient être revisités en utilisant `QK-means` à la place des K-moyennes lors de l'étape de regroupement pour la construction du *codebook* de chaque couche et ainsi réduire encore la complexité de leur modèle. Dans l'idée de combiner les méthodes sur lesquelles nous avons travaillé, nous pouvons aussi nous demander si la méthode `QK-means` ne pourrait pas être utilisée pour exprimer sous une forme comprimée les points de référence de la couche Nyström adaptative. L'approche qui nous semble la plus simple dans ce cas serait de pré-entraîner un réseau, calculer une représentation intermédiaire de tous les exemples d'un jeu de données, puis utiliser `QK-means` pour obtenir un ensemble de points de référence exprimés sous la forme d'un produit de matrices parcimonieuses, utilisé pour alléger la couche de Nyström adaptative.

Enfin, dans le Chapitre 4 qui est essentiellement expérimental, nous utilisons une autre méthode d'approximation parcimonieuse pour comprimer cette fois-ci des forêts aléatoires. Nous appliquons l'algorithme `OMP` pour pondérer et sélectionner les arbres d'une forêt aléatoire de façon à préserver au mieux la précision sur le jeu de données d'entraînement. L'algorithme `OMP` a été largement étudié en traitement du signal et de nombreux résultats théoriques accompagnent cet algorithme. Dans des travaux futurs, mettre en relation ces résultats avec le fonctionnement des forêts aléatoires pourrait permettre d'offrir une justification théorique aux excellents résultats empiriques obtenus dans nos travaux. Tout reste à faire dans cette voie et nous ne pouvons que spéculer pour l'instant mais une piste qui nous semble intéressante serait de mettre en lien l'étape de projection orthogonale dans `OMP` avec l'intérêt d'obtenir une forêt la plus diverse possible.

Au travers de ces différentes contributions, nous pensons avoir illustré l'intérêt d'utiliser des méthodes d'approximation parcimonieuses et les méthodes d'approximation de noyaux pour réduire la complexité de modèles d'apprentissage sans en impacter trop les performances.

Nous terminons ce document en tentant de prendre un peu de recul par rap-

port à ce qui a été fait dans cette thèse. L'apprentissage automatique connaît en ce moment un essor phénoménal, surtout grâce à l'explosion de la puissance de calcul disponible pour faire fonctionner des algorithmes d'apprentissage gourmands. Au fil de cette thèse, nous nous sommes efforcés de garder à l'esprit l'idée de construire des modèles avec une complexité réduite et nous pensons avoir fait au moins quelques avancées dans cette direction. Toutefois, il convient de se poser la question des raisons profondes qui motivent cette idée. Pourquoi souhaitons-nous travailler sur cette problématique ? en particulier : sommes nous motivés par des aspirations écologistes ? si oui, sommes nous sûrs de rendre service à ces aspirations en produisant des algorithmes moins chers à exécuter ? ne serait-ce pas, au contraire, une invitation à exécuter plus ? les centaines de milliers d'heures de calcul sur GPU utilisées dans le cadre de cette thèse me laissent dubitatif sur cette question. Étaient-t-elles nécessaires ou bien est-ce du gâchis ? peut-être que la méthode expérimentale n'était pas adaptée ? difficile à dire mais il me semble que ces questions devraient toujours être gardées à l'esprit.

Bibliographie

[1] Martín Abadi, Ashish Agarwal, Paul Barham et al. « TensorFlow : Large-Scale Machine Learning on Heterogeneous Systems ». In : Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. 2015. URL : https://www.tensorflow.org/ (cf. p. 47, 90).

[2] M. Ehsan Abbasnejad, Dhanesh Ramachandram et Rajeswari Mandava. « A Survey of the State of the Art in Learning the Kernels ». In : *Knowledge and Information Systems* 31.2 (mai 2012), p. 193-221. ISSN : 0219-1377, 0219-3116. DOI : 10.1007/s10115-011-0404-6. URL : http://link.springer.com/10.1007/s10115-011-0404-6 (visité le 31/07/2020) (cf. p. 15).

[3] Shivam Agrawal. *Diamonds Dataset*. 2017. URL : https://www.kaggle.com/shivam2503/diamonds (cf. p. 110).

[4] Mujadded Al Rabbani Alif, Sabbir Ahmed et Muhammad Abul Hasan. « Isolated Bangla Handwritten Character Recognition with Convolutional Neural Network ». In : 2017 20th International Conference of Computer and Information Technology (ICCIT). 2017, p. 1-6. DOI : 10.1109/ICCITECHN.2017.8281823 (cf. p. 28).

[5] Kamal M Ali et Michael John Pazzani. *On the Link between Error Correlation and Error Reduction in Decision Tree Ensembles*. 1995 (cf. p. 106).

[6] Zeyuan Allen-Zhu, Yuanzhi Li et Zhao Song. « A Convergence Theory for Deep Learning via Over-Parameterization ». In : *International Conference on Machine Learning*. PMLR. 2019, p. 242-252 (cf. p. 38).

[7] Daniel Alpay. « Some Remarks on Reproducing Kernel Krein Spaces ». In : *The Rocky Mountain Journal of Mathematics* (1991), p. 1189-1205 (cf. p. 14).

[8] Nachman Aronszajn. « Theory of Reproducing Kernels ». In : *Transactions of the American mathematical society* 68.3 (1950), p. 337-404 (cf. p. 11).

[9] Sanjeev Arora, Simon S Du, Wei Hu et al. « On Exact Computation with an Infinitely Wide Neural Net ». In : *Advances in Neural Information Processing Systems*. 2019, p. 8141-8150 (cf. p. 38).

[10] David Arthur et Sergei Vassilvitskii. « K-Means++ : The Advantages of Careful Seeding ». In : Symposium on Discrete Algorithms (SODA). Stanford, 7 juin 2006. URL : http://ilpubs.stanford.edu:8090/778/ (visité le 19/06/2020) (cf. p. 72, 74).

[11] Jimmy Lei Ba, Jamie Ryan Kiros et Geoffrey E Hinton. *Layer Normalization*. 2016. URL : https://www.cs.utoronto.ca/~hinton/absps/LayerNormalization.pdf (cf. p. 31).

[12] Kunlun Bai. *A Comprehensive Introduction to Different Types of Convolutions in Deep Learning*. Medium. 2019. URL : https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215 (visité le 15/07/2020) (cf. p. 23).

[13] David A Belsley, Edwin Kuh et Roy E Welsch. *Regression Diagnostics : Identifying Influential Data and Sources of Collinearity*. T. 571. John Wiley & Sons, 1980 (cf. p. 110).

[14] Jon Louis Bentley. « Multidimensional Binary Search Trees Used for Associative Searching ». In : *Communications of the ACM* 18.9 (1975), p. 509-517. ISSN : 0001-0782. DOI : 10.1145/361002.361007. URL : https://doi.org/10.1145/361002.361007 (visité le 22/07/2020) (cf. p. 80).

[15] Simon Bernard, Sébastien Adam et Laurent Heutte. « Dynamic Random Forests ». In : *Pattern Recognition Letters* 33.12 (2012), p. 1580-1586. ISSN : 0167-8655. DOI : 10.1016/j.patrec.2012.04.003. URL : http://www.sciencedirect.com/science/article/pii/S0167865512001274 (visité le 29/06/2020) (cf. p. 105).

[16] Christopher M Bishop. *Pattern Recognition and Machine Learning*. springer, 2006 (cf. p. 14).

[17] RK Bock, A Chilingarian, M Gaug et al. « Methods for Multidimensional Event Classification : A Case Study Using Images from a Cherenkov Gamma-Ray Telescope ». In : *Nuclear Instruments and Methods in Physics Research Section A : Accelerators, Spectrometers, Detectors and Associated Equipment* 516.2-3 (2004), p. 511-528 (cf. p. 110).

[18] Jérôme Bolte, Shoham Sabach et Marc Teboulle. « Proximal Alternating Linearized Minimization for Nonconvex and Nonsmooth Problems ». In : *Mathematical Programming* 146.1-2 (2014), p. 459-494. ISSN : 0025-5610, 1436-4646. DOI : 10.1007/s10107-013-0701-9. URL : http://link.springer.com/10.1007/s10107-013-0701-9 (visité le 21/07/2020) (cf. p. 62).

[19] Christos Boutsidis, Anastasios Zouzias, Michael W Mahoney et al. « Randomized Dimensionality Reduction for k -Means Clustering ». In : *IEEE Transactions on Information Theory* 61.2 (2014), p. 1045-1062 (cf. p. 65, 72).

[20] Leo Breiman. « Random Forests ». In : *Machine learning* 45.1 (2001), p. 5-32 (cf. p. 102, 103, 105).

[21] Leo Breiman, Jerome Friedman, Charles J Stone et al. *Classification and Regression Trees*. CRC press, 1984 (cf. p. 103).

[22] Gavin Brown et Ludmila I. Kuncheva. « “Good” and “Bad” Diversity in Majority Vote Ensembles ». In : *Multiple Classifier Systems*. Lecture Notes in Computer Science. Springer, 2010, p. 124-133. isbn : 978-3-642-12127-2. doi : 10.1007/978-3-642-12127-2_13 (cf. p. 106).

[23] Alfred M Bruckstein, Michael Elad et Michael Zibulevsky. « Sparse Non-Negative Solution of a Linear System of Equations Is Unique ». In : *2008 3rd International Symposium on Communications, Control and Signal Processing*. IEEE. 2008, p. 762-767 (cf. p. 115).

[24] Cristian Buciluǎ, Rich Caruana et Alexandru Niculescu-Mizil. « Model Compression ». In : *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2006, p. 535-541 (cf. p. 37, 100).

[25] Lars Buitinck, Gilles Louppe, Mathieu Blondel et al. « API Design for Machine Learning Software : Experiences from the Scikit-Learn Project ». In : *ECML PKDD Workshop : Languages for Data Mining and Machine Learning*. 2013, p. 108-122 (cf. p. 110).

[26] Massimo Buscema. « Metanet* : The Theory of Independent Judges ». In : *Substance use & misuse* 33.2 (1998), p. 439-461 (cf. p. 110).

[27] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew et al. « Ensemble Selection from Libraries of Models ». In : Twenty-First International Conference on Machine Learning - ICML ’04. Banff, Alberta, Canada : ACM Press, 2004, p. 18. doi : 10.1145/1015330.1015432. url : http://portal.acm.org/citation.cfm?doid=1015330.1015432 (visité le 29/06/2020) (cf. p. 106, 110).

[28] Dexiong Chen, Laurent Jacob et Julien Mairal. « Biological Sequence Modeling with Convolutional Kernel Networks ». In : *Bioinformatics* 35.18 (2019), p. 3294-3302 (cf. p. 38).

[29] Dexiong Chen, Laurent Jacob et Julien Mairal. « Recurrent Kernel Networks ». In : *Advances in Neural Information Processing Systems 32*. Sous la dir. de H. Wallach, H. Larochelle, A. Beygelzimer et al. Curran Associates, Inc., 2019, p. 13431-13442. url : http://papers.nips.cc/paper/9499-recurrent-kernel-networks.pdf (cf. p. 38).

[30] Dexiong Chen, Laurent Jacob et Julien Mairal. « Convolutional Kernel Networks for Graph-Structured Data ». In : Proceedings of the International Conference on Machine Learning. 2020. arXiv : 2003.05189 [cs, stat]. url : http://arxiv.org/abs/2003.05189 (visité le 17/07/2020) (cf. p. 38, 53).

[31] Youngmin Cho et Lawrence K Saul. « Kernel Methods for Deep Learning ». In : *Advances in Neural Information Processing Systems*. 2009, p. 342-350 (cf. p. 38).

[32] Francois Chollet. *Keras : The Python Deep Learning API*. 2015. URL : https://keras.io/ (visité le 17/06/2020) (cf. p. 47, 90).

[33] Peter I Corke. « A Robotics Toolbox for MATLAB ». In : *IEEE Robotics & Automation Magazine* 3.1 (1996), p. 24-32 (cf. p. 110).

[34] Antoine Cornuéjols et Laurent Miclet. *Apprentissage Artificiel : Concepts et Algorithmes*. Editions Eyrolles, 2011 (cf. p. 9, 103).

[35] Matthieu Courbariaux, Yoshua Bengio et Jean-Pierre David. *BinaryConnect : Training Deep Neural Networks with Binary Weights during Propagations*. 18 avr. 2016. arXiv : 1511.00363 [cs]. URL : http://arxiv.org/abs/1511.00363 (visité le 30/07/2020) (cf. p. 36).

[36] César Couza. *Kernel Functions for Machine Learning Applications*. 2020. URL : http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/ (visité le 13/07/2020) (cf. p. 12).

[37] Lorrie Faith Cranor et Brian A LaMacchia. « Spam ! » In : *Communications of the ACM* 41.8 (1998), p. 74-83 (cf. p. 110).

[38] Danilo Croce, Daniele Rossini et Roberto Basili. « Explaining Non-Linear Classifier Decisions within Kernel-Based Deep Architectures ». In : *Proceedings of the 2018 EMNLP Workshop BlackboxNLP : Analyzing and Interpreting Neural Networks for NLP*. 2018, p. 16-24 (cf. p. 41).

[39] George Cybenko. « Approximation by Superpositions of a Sigmoidal Function ». In : *Mathematics of control, signals and systems* 2.4 (1989), p. 303-314 (cf. p. 26).

[40] Tri Dao, Albert Gu, Matthew Eichhorn et al. « Learning Fast Algorithms for Linear Transforms Using Butterfly Factorizations ». In : Proceedings of Machine Learning Research. T. 97. PMLR, 2019, p. 1517-1527. arXiv : 1903.05895 [cs, stat]. URL : http://arxiv.org/abs/1903.05895 (visité le 01/07/2020) (cf. p. 57, 60).

[41] Geoff Davis, Stephane Mallat et Marco Avellaneda. « Adaptive Greedy Approximations ». In : *Constructive approximation* 13.1 (1997), p. 57-98 (cf. p. 107, 108).

[42] Li Deng. « The MNIST Database of Handwritten Digit Images for Machine Learning Research ». In : *IEEE Signal Processing Magazine* 29.6 (2012), p. 141-142. ISSN : 1558-0792. DOI : 10.1109/MSP.2012.2211477 (cf. p. 46, 72, 92).

[43] Dheeru Dua et Casey Graff. *UCI Machine Learning Repository*. 2017. URL : http://archive.ics.uci.edu/ml (cf. p. 72).

[44] Bradley Efron, Trevor Hastie, Iain Johnstone et al. « Least Angle Regression ». In : *The Annals of statistics* 32.2 (2004), p. 407-499 (cf. p. 110).

[45] Charles Elkan. « Using the Triangle Inequality to Accelerate K-Means ». In : *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, p. 147-153 (cf. p. 65, 83).

[46] Khaled Fawagreh, Mohamed Medhat Gaber et Eyad Elyan. « An Outlier Ranking Tree Selection Approach to Extreme Pruning of Random Forests ». In : *International Conference on Engineering Applications of Neural Networks*. Springer. 2016, p. 267-282 (cf. p. 106, 110).

[47] *FAμST*. 2020. URL : https://faust.inria.fr/fr/ (visité le 23/07/2020) (cf. p. 90).

[48] Jonathan Frankle et Michael Carbin. « The Lottery Ticket Hypothesis : Finding Sparse, Trainable Neural Networks ». In : International Conference on Learning Representations (ICLR). 2019. arXiv : 1803.03635 [cs]. URL : http://arxiv.org/abs/1803.03635 (visité le 30/04/2020) (cf. p. 32, 36, 97).

[49] Trevor Gale, Erich Elsen et Sara Hooker. « The State of Sparsity in Deep Neural Networks ». In : Joint Workshop on On-Device Machine Learning & Compact Deep Neural Network Representations (ODML-CDNNR). 2019. arXiv : 1902.09574 [cs, stat]. URL : http://arxiv.org/abs/1902.09574 (visité le 30/04/2020) (cf. p. 89).

[50] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan et al. « Domain-Adversarial Training of Neural Networks ». In : *The Journal of Machine Learning Research* 17.1 (2016), p. 2096-2030 (cf. p. 117).

[51] Timur Garipov. *Github Tensorizing Neural Networks*. 2020. URL : https://github.com/timgaripov/TensorNet-TF (visité le 15/07/2020) (cf. p. 34).

[52] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov et al. *Ultimate Tensorization : Compressing Convolutional and FC Layers Alike*. 2016. arXiv : 1611.03214 [cs]. URL : http://arxiv.org/abs/1611.03214 (visité le 06/05/2020) (cf. p. 34, 89).

[53] Luc Giffon. *Gitlab Adaptive Nystrom*. GitLab. 2020. URL : https://gitlab.lis-lab.fr/luc.giffon/deepFriedConvnet (visité le 02/09/2020) (cf. p. 47).

[54] Luc Giffon. *Gitlab Adaptive Nystrom*. 2020. URL : https://gitlab.lis-lab.fr/luc.giffon/deepstrom_network (visité le 02/09/2020) (cf. p. 48).

[55] Luc Giffon. *Gitlab Couche Nyström*. GitLab. 2020. URL : https://gitlab.lis-lab.fr/luc.giffon/nystrom-layer (visité le 02/09/2020) (cf. p. 48).

[56] Luc Giffon. *Gitlab Palmnet : Compression de Réseaux de Neuronnes Par Factorisations Parcimonieuses*. GitLab. 2020. URL : https://gitlab.lis-lab.fr/luc.giffon/palmnet (visité le 02/09/2020) (cf. p. 90).

[57] Luc Giffon et Valentin Emiya. *Github QK-Means*. 2020. URL : https://github.com/lucgiffon/qkmeans (visité le 22/07/2020) (cf. p. 73, 90).

[58] Alex Gittens. *The Spectral Norm Error of the Naive Nystrom Extension*. 2011. arXiv : 1110.5305 [cs, math]. URL : http://arxiv.org/abs/1110.5305 (visité le 25/06/2020) (cf. p. 19).

[59] Alex Gittens et Michael W. Mahoney. « Revisiting the Nystrom Method for Improved Large-Scale Machine Learning ». In : *Proceedings of the 30th International Conference on Machine Learning (PMLR)*. Proceedings of the 30th International Conference on Machine Learning (PMLR). 2013. arXiv : 1303.1849. URL : http://arxiv.org/abs/1303.1849 (visité le 25/06/2020) (cf. p. 19).

[60] Xavier Glorot et Yoshua Bengio. « Understanding the Difficulty of Training Deep Feedforward Neural Networks ». In : *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, p. 249-256 (cf. p. 29, 91).

[61] Xavier Glorot, Antoine Bordes et Yoshua Bengio. « Deep Sparse Rectifier Neural Networks ». In : *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, p. 315-323 (cf. p. 30).

[62] Mehmet Gonen et Ethem Alpaydin. « Multiple Kernel Learning Algorithms ». In : *The Journal of Machine Learning Research* (2011), p. 58 (cf. p. 15).

[63] Yunchao Gong, Liu Liu, Ming Yang et al. *Compressing Deep Convolutional Networks Using Vector Quantization*. 2014. arXiv : 1412.6115 [cs]. URL : http://arxiv.org/abs/1412.6115 (visité le 30/07/2020) (cf. p. 36).

[64] Jianping Gou, Baosheng Yu, Stephen John Maybank et al. *Knowledge Distillation : A Survey*. Version 1. 2020. arXiv : 2006.05525 [cs, stat]. URL : http://arxiv.org/abs/2006.05525 (visité le 30/07/2020) (cf. p. 37).

[65] R. Gribonval et P. Vandergheynst. « On the Exponential Convergence of Matching Pursuits in Quasi-Incoherent Dictionaries ». In : *IEEE Transactions on Information Theory* 52.1 (2006), p. 255-261. ISSN : 1557-9654. DOI : 10.1109/TIT.2005.860474 (cf. p. 108).

[66] Gregory Griffin, Alex Holub et Pietro Perona. *Caltech-256 Object Category Dataset*. 2007. URL : https://authors.library.caltech.edu/7694/ (cf. p. 72).

[67] Yunhui Guo. *A Survey on Methods and Theories of Quantized Neural Networks*. 2018. arXiv : 1808.04752 [cs, stat]. URL : http://arxiv.org/abs/1808.04752 (visité le 29/07/2020) (cf. p. 37).

[68] Greg Hamerly. « Making K-Means Even Faster ». In : *Proceedings of the 2010 SIAM International Conference on Data Mining*. SIAM. 2010, p. 130-140 (cf. p. 65).

[69] Song Han, Huizi Mao et William J. Dally. « Deep Compression : Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding ». In : International Conference on Learning Representations (ICLR). 2016. arXiv : 1510.00149 [cs]. URL : http://arxiv.org/abs/1510.00149 (visité le 30/07/2020) (cf. p. 37).

[70] John A Hartigan et Manchek A Wong. « Algorithm AS 136 : A k-Means Clustering Algorithm ». In : *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979), p. 100-108 (cf. p. 63).

[71] Md Hasan, Tajwar Abrar Aleef et al. *Automatic Mass Detection in Breast Using Deep Convolutional Neural Network and Svm Classifier*. 2019. arXiv : 1907.04424. URL : https://arxiv.org/abs/1907.04424 (cf. p. 27).

[72] Babak Hassibi et David G. Stork. « Second Order Derivatives for Network Pruning : Optimal Brain Surgeon ». In : *Advances in Neural Information Processing Systems 5*. Morgan-Kaufmann, 1993, p. 164-171. URL : http://papers.nips.cc/paper/647-second-order-derivatives-for-network-pruning-optimal-brain-surgeon.pdf (visité le 28/06/2020) (cf. p. 36).

[73] Tamir Hazan et Tommi Jaakkola. *Steps toward Deep Kernel Methods from Infinite Neural Networks*. 2015. arXiv : 1508.05133. URL : https://arxiv.org/abs/1508.05133 (cf. p. 38, 41).

[74] Kaiming He, Xiangyu Zhang, Shaoqing Ren et al. « Delving Deep into Rectifiers : Surpassing Human-Level Performance on ImageNet Classification ». In : IEEE International Conference on Computer Vision. 2015. arXiv : 1502.01852 [cs]. URL : http://arxiv.org/abs/1502.01852 (visité le 27/06/2020) (cf. p. 91).

[75] Kaiming He, Xiangyu Zhang, Shaoqing Ren et al. « Deep Residual Learning for Image Recognition ». In : 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA : IEEE, 2016, p. 770-778. ISBN : 978-1-4673-8851-1. DOI : 10.1109/CVPR.2016.90. URL : http://ieeexplore.ieee.org/document/7780459/ (visité le 26/06/2020) (cf. p. 27-30).

[76] Uri Heinemann, Roi Livni, Elad Eban et al. « Improper Deep Kernels ». In : *Artificial Intelligence and Statistics*. 2016, p. 1159-1167 (cf. p. 38).

[77] Tyler Highlander et Andres Rodriguez. « Very Efficient Training of Convolutional Neural Networks Using Fast Fourier Transform and Overlap-and-Add ». In : Proceedings of the British Machine Vision Conference. 2016 (cf. p. 34).

[78] Geoffrey Hinton, Nitish Srivastava et Kevin Swersky. « Lecture on Neural Networks for Machine Learning ». 2012. URL : http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (visité le 23/07/2020) (cf. p. 92).

[79] Geoffrey Hinton, Oriol Vinyals et Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv : 1503.02531 [cs, stat]. URL : http://arxiv.org/abs/1503.02531 (visité le 23/07/2020) (cf. p. 37, 97, 100).

[80] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky et al. *Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors*. 2012. arXiv : 1207.0580. URL : https://arxiv.org/abs/1207.0580 (cf. p. 31).

[81] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi et al. *Gradient Flow in Recurrent Nets : The Difficulty of Learning Long-Term Dependencies*. 2001 (cf. p. 27).

[82] Kurt Hornik. « Approximation Capabilities of Multilayer Feedforward Networks ». In : *Neural networks* 4.2 (1991), p. 251-257 (cf. p. 26).

[83] Qinghua Hu, Daren Yu, Zongxia Xie et al. « EROS : Ensemble Rough Subspaces ». In : *Pattern Recognition* 40.12 (1er déc. 2007), p. 3728-3739. ISSN : 0031-3203. DOI : 10.1016/j.patcog.2007.04.022. URL : http://www.sciencedirect.com/science/article/pii/S0031320307002099 (visité le 29/06/2020) (cf. p. 106).

[84] Gary B Huang, Marwan Mattar, Tamara Berg et al. *Labeled Faces in the Wild : A Database Forstudying Face Recognition in Unconstrained Environments*. 2008 (cf. p. 110).

[85] Sergey Ioffe et Christian Szegedy. « Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift ». In : Proceedings of the 32nd International Conference on Machine Learning (ICML). 2015. arXiv : 1502.03167 (cf. p. 30).

[86] Arthur Jacot, Franck Gabriel et Clément Hongler. « Neural Tangent Kernel : Convergence and Generalization in Neural Networks ». In : *Advances in Neural Information Processing Systems*. 2018, p. 8571-8580 (cf. p. 38).

[87] Anil K Jain. « Data Clustering : 50 Years beyond K-Means ». In : *Pattern recognition letters* 31.8 (2010), p. 651-666 (cf. p. 63).

[88] Cijo Jose, Prasoon Goyal, Parv Aggrwal et al. « Local Deep Kernel Learning for Efficient Non-Linear Svm Prediction ». In : *International Conference on Machine Learning*. 2013, p. 486-494 (cf. p. 38).

[89] H Jégou, M Douze et C Schmid. « Product Quantization for Nearest Neighbor Search ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (2011), p. 117-128. ISSN : 0162-8828. DOI : 10.1109/TPAMI.2010.57. URL : http://ieeexplore.ieee.org/document/5432202/ (visité le 30/07/2020) (cf. p. 36).

[90] Purushottam Kar et Harish Karnick. « Random Feature Maps for Dot Product Kernels ». In : *Artificial Intelligence and Statistics*. 2012, p. 583-591 (cf. p. 17).

[91] Nicolas Keriven, Nicolas Tremblay, Yann Traonmilin et al. « Compressive K-Means ». In : *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, p. 6369-6373 (cf. p. 66, 72).

[92] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo et al. « Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications ». In : International Conference on Learning Representations (ICLR). 2016. arXiv : 1511.06530 [cs]. URL : http://arxiv.org/abs/1511.06530 (visité le 01/06/2020) (cf. p. 33, 89, 91).

[93] Diederik P Kingma et Jimmy Ba. « Adam : A Method for Stochastic Optimization ». In : International Conference on Learning Representations (ICLR). 2014. arXiv : 1412.6980 (cf. p. 29, 93).

[94] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009 (cf. p. 46, 92).

[95] Vrushali Y Kulkarni et Pradeep K Sinha. « Pruning of Random Forest Classifiers : A Survey and Future Directions ». In : *2012 International Conference on Data Science & Engineering (ICDSE)*. IEEE. 2012, p. 64-68 (cf. p. 102, 105).

[96] Sanjiv Kumar, Mehryar Mohri et Ameet Talwalkar. « Sampling Methods for the Nyström Method ». In : *Journal of Machine Learning Research* 13.34 (2012), p. 981-1006. ISSN : 1533-7928. URL : http://jmlr.org/papers/v13/kumar12a.html (visité le 17/06/2020) (cf. p. 19, 48, 82).

[97] Charly Lamothe, Léo Bouscarat, Luc Giffon et al. *Gitlab OMPForest*. GitLab. 2020. URL : https://gitlab.lis-lab.fr/luc.giffon/bolsonaro (visité le 02/09/2020) (cf. p. 110).

[98] Quoc Le, Tamás Sarlós et Alex Smola. « Fastfood — Approximating Kernel Expansions in Loglinear Time ». In : Proceedings of the 30 Th International Conference on Machine Learning. 2013, p. 9 (cf. p. 17, 56).

[99] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba et al. « Speeding-up Convolutional Neural Networks Using Fine-Tuned CP-Decomposition ». In : International Conference on Learning Representations (ICLR). 2015. arXiv : 1412.6553. URL : http://arxiv.org/abs/1412.6553 (visité le 27/06/2020) (cf. p. 33).

[100] Yann LeCun, Léon Bottou, Yoshua Bengio et al. « Gradient-Based Learning Applied to Document Recognition ». In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324 (cf. p. 26, 46).

[101] Yann LeCun, John S. Denker et Sara A. Solla. « Optimal Brain Damage ». In : *Advances in Neural Information Processing Systems 2*. Morgan-Kaufmann, 1990, p. 598-605. URL : http://papers.nips.cc/paper/250-optimal-brain-damage.pdf (visité le 28/06/2020) (cf. p. 36).

[102] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus et al. « Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function ». In : *Neural networks* 6.6 (1993), p. 861-867 (cf. p. 26).

[103] Ping Li, Gennady Samorodnitsk et John Hopcroft. « Sign Cauchy Projections and Chi-Square Kernel ». In : *Advances in Neural Information Processing Systems 26*. Advances in Neural Information Processing Systems 26. Curran Associates, Inc., 2013, p. 2571-2579. URL : http://papers.nips.cc/paper/5075-sign-cauchy-projections-and-chi-square-kernel.pdf (visité le 24/06/2020) (cf. p. 14).

[104] Wei Li. *Github Cifar10 Cnn Implementations*. 2020. URL : https://github.com/BIGBALLON/cifar-10-cnn (visité le 15/07/2020) (cf. p. 26, 92).

[105] Yingzhou Li, Haizhao Yang, Eileen R Martin et al. « Butterfly Factorization ». In : *Multiscale Modeling & Simulation* 13.2 (2015), p. 714-732 (cf. p. 35, 58-60).

[106] Hsuan-Tien Lin et Chih-Jen Lin. « A Study on Sigmoid Kernels for SVM and the Training of Non-PSD Kernels by SMO-Type Methods ». In : *Neural computation* (2003), p. 32 (cf. p. 14).

[107] Weiwei Liu, Xiaobo Shen et Ivor Tsang. « Sparse Embedded k -Means Clustering ». In : *Advances in Neural Information Processing Systems*. 2017, p. 3319-3327 (cf. p. 65, 72).

[108] Zhuang Liu, Mingjie Sun, Tinghui Zhou et al. « Rethinking the Value of Network Pruning ». In : International Conference on Learning Representations (ICLR). 2018. arXiv : 1810.05270 [cs, stat]. URL : http://arxiv.org/abs/1810.05270 (visité le 09/05/2020) (cf. p. 35).

[109] Christos Louizos, Max Welling et Diederik P. Kingma. « Learning Sparse Neural Networks through L_0 Regularization ». In : International Conference on Learning Representations. 22 juin 2018. arXiv : 1712.01312 [cs, stat]. URL : http://arxiv.org/abs/1712.01312 (visité le 30/04/2020) (cf. p. 36, 89).

[110] Luc Le Magoarou et Rémi Gribonval. « Flexible Multi-Layer Sparse Approximations of Matrices and Applications ». In : *IEEE Journal of Selected Topics in Signal Processing* 10.4 (2016), p. 688-700. ISSN : 1932-4553, 1941-0484. DOI : 10.1109/JSTSP.2016.2543461. arXiv : 1506.07300. URL : http://arxiv.org/abs/1506.07300 (visité le 18/05/2020) (cf. p. 57, 58, 60-63, 73, 74, 87, 95, 101).

[111] Julien Mairal. « End-to-End Kernel Learning with Supervised Convolutional Kernel Networks ». In : Advances in Neural Information Processing Systems (NIPS). 2016, p. 1399-1407 (cf. p. 38, 41).

[112] Julien Mairal, Piotr Koniusz, Zaid Harchaoui et al. « Convolutional Kernel Networks ». In : *Advances in Neural Information Processing Systems*. 2014, p. 2627-2635 (cf. p. 38, 41).

[113] Olvi L Mangasarian, W Nick Street et William H Wolberg. « Breast Cancer Diagnosis and Prognosis via Linear Programming ». In : *Operations Research* 43.4 (1995), p. 570-577 (cf. p. 110).

[114] Dmitry Molchanov, Arsenii Ashukha et Dmitry Vetrov. « Variational Dropout Sparsifies Deep Neural Networks ». In : International Conference on Machine Learning. 2017. arXiv : 1701.05369. URL : http://arxiv.org/abs/1701.05369 (visité le 28/06/2020) (cf. p. 36, 89).

[115] Grégoire Montavon, Mikio L Braun et Klaus-Robert Müller. « Kernel Analysis of Deep Networks. » In : *Journal of Machine Learning Research* 12.9 (2011) (cf. p. 38).

[116] Fabien Moutarde. *Convolutional Networks Notebook*. 2020. URL : http://perso.mines-paristech.fr/fabien.moutarde/ES_MachineLearning/TP_convNets/convnet-notebook.html (visité le 15/07/2020) (cf. p. 22).

[117] Marius Muja et David G Lowe. « Scalable Nearest Neighbor Algorithms for High Dimensional Data ». In : *IEEE transactions on pattern analysis and machine intelligence* 36.11 (2014), p. 2227-2240 (cf. p. 63).

[118] Martin Musiol. *Speeding up Deep Learning Computational Aspects of Machine Learning*. 2016 (cf. p. 30).

[119] Sameer A. Nene, Shree K. Nayar et Hiroshi Murase. *Columbia Object Image Library (COIL-20)*. 1996 (cf. p. 72).

[120] Yuval Netzer, Tao Wang, Adam Coates et al. *Reading Digits in Natural Images with Unsupervised Feature Learning*. 2011, p. 9 (cf. p. 46, 92).

[121] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli et al. « The Role of Over-Parametrization in Generalization of Neural Networks ». In : *International Conference on Learning Representations*. 2018 (cf. p. 32).

[122] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin et al. « Tensorizing Neural Networks ». In : Advances in Neural Information Processing Systems. 2015. arXiv : 1509.06569 [cs]. URL : http://arxiv.org/abs/1509.06569 (visité le 27/06/2020) (cf. p. 34, 89).

[123] Dino Oglic et Thomas Gärtner. « Learning in Reproducing Kernel Krein Spaces ». In : International Conference on Machine Learning (ICML). 2018 (cf. p. 14).

[124] Stephen M. Omohundro. *Five Balltree Construction Algorithms*. 1989 (cf. p. 80).

[125] Cheng Soon Ong, Xavier Mary, Stéphane Canu et al. « Learning with Non-Positive Kernels ». In : *Proceedings of the Twenty-First International Conference on Machine Learning*. 2004, p. 81 (cf. p. 14).

[126] R Kelley Pace et Ronald Barry. « Sparse Spatial Autoregressions ». In : *Statistics & Probability Letters* 33.3 (1997), p. 291-297 (cf. p. 110).

[127] Yagyensh Chandra Pati, Ramin Rezaiifar et Perinkulam Sambamurthy Krishnaprasad. « Orthogonal Matching Pursuit : Recursive Function Approximation with Applications to Wavelet Decomposition ». In : *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*. IEEE. 1993, p. 40-44 (cf. p. 107, 108).

[128] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort et al. « Scikit-Learn : Machine Learning in Python ». In : *Journal of machine learning research* 12 (Oct 2011), p. 2825-2830 (cf. p. 81).

[129] Qichao Que et Mikhail Belkin. « Back to the Future : Radial Basis Function Networks Revisited. » In : *AISTATS*. 2016, p. 1375-1383 (cf. p. 63).

[130] Ali Rahimi et Benjamin Recht. « Random Features for Large-Scale Kernel Machines ». In : *Advances in Neural Information Processing Systems 20*. Curran Associates, Inc., 2008, p. 1177-1184. URL : http://papers.nips.cc/paper/3182-random-features-for-large-scale-kernel-machines.pdf (visité le 15/06/2020) (cf. p. 16, 17).

[131] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon et al. « XNOR-Net : ImageNet Classification Using Binary Convolutional Neural Networks ». In : European Conference on Computer Vision (ECCV). 2016. arXiv : 1603.05279 [cs]. URL : http://arxiv.org/abs/1603.05279 (visité le 30/07/2020) (cf. p. 36).

[132] José Luis Rojo-Álvarez, Manel Martínez-Ramón, Jordi Muñoz Marí et al. *Digital Signal Processing with Kernel Methods*. Wiley Online Library, 2018 (cf. p. 13).

[133] Lior Rokach. « Collective-Agreement-Based Pruning of Ensembles ». In : *Computational Statistics & Data Analysis* 53.4 (2009), p. 1015-1026 (cf. p. 106).

[134] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou et al. « FitNets : Hints for Thin Deep Nets ». In : International Conference on Learning Representations (ICLR). 2015. arXiv : 1412.6550 [cs]. URL : http://arxiv.org/abs/1412.6550 (visité le 30/07/2020) (cf. p. 37, 97, 119).

[135] Sebastian Ruder. *An Overview of Gradient Descent Optimization Algorithms*. 2017. arXiv : 1609.04747. URL : http://arxiv.org/abs/1609.04747 (visité le 28/08/2020) (cf. p. 29).

[136] Walter Rudin. *Fourier Analysis on Groups*. T. 121967. Wiley Online Library, 1962 (cf. p. 16).

[137] David E Rumelhart, Geoffrey E Hinton et Ronald J Williams. « Learning Representations by Back-Propagating Errors ». In : *nature* 323.6088 (1986), p. 533-536 (cf. p. 29).

[138] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani et al. « Low-Rank Matrix Factorization for Deep Neural Network Training with High-Dimensional Output Targets ». In : IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Vancouver, BC, Canada : IEEE, 2013, p. 6655-6659. isbn : 978-1-4799-0356-6. doi : 10.1109/ICASSP.2013.6638949. url : http://ieeexplore.ieee.org/document/6638949/ (visité le 27/06/2020) (cf. p. 32, 89).

[139] Ali Salehi. *Convolution as Multiplication*. 2020. url : https://github.com/alisaaalehi/convolution_as_multiplication (visité le 15/07/2020) (cf. p. 24).

[140] B. Scholkopf, S. Mika, C.J.C. Burges et al. « Input Space versus Feature Space in Kernel-Based Methods ». In : *IEEE Transactions on Neural Networks* 10.5 (1999), p. 1000-1017. issn : 10459227. doi : 10.1109/72.788641. url : http://ieeexplore.ieee.org/document/788641/ (visité le 16/06/2020) (cf. p. 42).

[141] David Sculley. « Web-Scale k-Means Clustering ». In : *Proceedings of the 19th International Conference on World Wide Web*. ACM. 2010, p. 1177-1178 (cf. p. 65, 72).

[142] Alen D Shapiro. *Structured Induction in Expert Systems*. Addison-Wesley Longman Publishing Co., Inc., 1987 (cf. p. 110).

[143] John Shawe-Taylor, Nello Cristianini et al. *Kernel Methods for Pattern Analysis*. Cambridge university press, 2004 (cf. p. 11).

[144] Xiaobo Shen, Weiwei Liu, Ivor Tsang et al. « Compressed K-Means for Large-Scale Clustering ». In : *Thirty-First AAAI Conference on Artificial Intelligence*. 2017 (cf. p. 65, 72).

[145] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen et al. « Weisfeiler-Lehman Graph Kernels ». In : *Journal of Machine Learning Research* 12.9 (2011) (cf. p. 11, 118).

[146] Si Si, Cho-Jui Hsieh et Inderjit Dhillon. « Computationally Efficient Nyström Approximation Using Fast Transforms ». In : *International Conference on Machine Learning*. 2016, p. 2655-2663 (cf. p. 19, 56, 63, 78, 81, 82, 118).

[147] Karen Simonyan et Andrew Zisserman. « Very Deep Convolutional Networks for Large-Scale Image Recognition ». In : International Conference on Learning Representations (ICLR). 2015. arXiv : 1409.1556 [cs]. url : http://arxiv.org/abs/1409.1556 (visité le 26/06/2020) (cf. p. 26, 46).

[148] Alex J Smola et Bernhard Schölkopf. *Learning with Kernels*. T. 4. Citeseer, 1998 (cf. p. 11).

[149] Huan Song, Jayaraman J Thiagarajan, Prasanna Sattigeri et al. « Optimizing Kernel Machines Using Deep Learning ». In : *IEEE Transactions on Neural Networks and Learning Systems* (2018) (cf. p. 41).

[150] Ingo Steinwart, Philipp Thomann et Nico Schmid. *Learning with Hierarchical Gaussian Kernels*. 2016. arXiv : 1612.00824. URL : http://arxiv.org/abs/1612.00824 (cf. p. 38).

[151] Pierre Stock, Armand Joulin, Rémi Gribonval et al. « And the Bit Goes down : Revisiting the Quantization of Neural Networks ». In : *International Conference on Learning Representations*. 2020. URL : https://openreview.net/forum?id=rJehVyrKwH (cf. p. 37, 96, 119).

[152] Ilya Sutskever, James Martens et George Dahl. « On the Importance of Initialization and Momentum in Deep Learning ». In : International Conference on Machine Learning. 2013, p. 9 (cf. p. 92).

[153] Christian Szegedy, Wei Liu, Yangqing Jia et al. « Going Deeper with Convolutions ». In : Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2014. arXiv : 1409.4842 [cs]. URL : http://arxiv.org/abs/1409.4842 (visité le 26/06/2020) (cf. p. 27).

[154] Keras Team. *Keras Documentation : Keras Applications*. 2020. URL : https://keras.io/api/applications/ (visité le 16/07/2020) (cf. p. 31).

[155] Keras Team. *Keras Documentation : Keras Layers API*. 2020. URL : https://keras.io/api/layers/ (visité le 08/08/2020) (cf. p. 25).

[156] J.A. Tropp. « Greed Is Good : Algorithmic Results for Sparse Approximation ». In : *IEEE Transactions on Information Theory* 50.10 (2004), p. 2231-2242. ISSN : 0018-9448. DOI : 10.1109/TIT.2004.834793. URL : http://ieeexplore.ieee.org/document/1337101/ (visité le 29/06/2020) (cf. p. 108).

[157] Twan Van Laarhoven et Elena Marchiori. « Local Network Community Detection with Continuous Optimization of Conductance and Weighted Kernel K-Means ». In : *The Journal of Machine Learning Research* 17.1 (2016), p. 5148-5175 (cf. p. 63).

[158] Charles Van Loan. *Computational Frameworks for the Fast Fourier Transform*. SIAM, 1992 (cf. p. 58).

[159] Vladimir N Vapnik. « An Overview of Statistical Learning Theory ». In : *IEEE transactions on neural networks* 10.5 (1999), p. 988-999 (cf. p. 3).

[160] Kevin Vecmanis. *Support Vector Machine Hyperparameter Tuning - A Visual Guide*. Kevin Vecmanis. URL : https://kevinvecmanis.io/machine%20learning/hyperparameter%20tuning/dataviz/python/svm/2019/05/12/Support-Vector-Machines-Visual-Guide.html (visité le 10/08/2020) (cf. p. 12, 104).

[161] Nguyen Xuan Vinh, Julien Epps et James Bailey. « Information Theoretic Measures for Clusterings Comparison : Variants, Properties, Normalization and Correction for Chance ». In : *The Journal of Machine Learning Research* 11 (2010), p. 2837-2854 (cf. p. 79).

[162] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor et al. « Graph Kernels ». In : *The Journal of Machine Learning Research* 11 (2010), p. 1201-1242 (cf. p. 11, 118).

[163] Li Wan, Matthew Zeiler, Sixin Zhang et al. « Regularization of Neural Networks Using Dropconnect ». In : *International Conference on Machine Learning*. 2013, p. 1058-1066 (cf. p. 31).

[164] Kilian Q Weinberger et Lawrence K Saul. « Distance Metric Learning for Large Margin Nearest Neighbor Classification. » In : *Journal of Machine Learning Research* 10.2 (2009) (cf. p. 15).

[165] Eric P Widmaier, Hershel Raff, Kevin T Strang et al. *Vander's Human Physiology : The Mechanisms of Body Function*. Boston : McGraw-Hill Higher Education, 2008 (cf. p. 21).

[166] Christopher K. I. Williams et Matthias Seeger. « Using the Nyström Method to Speed Up Kernel Machines ». In : *Advances in Neural Information Processing Systems 13*. MIT Press, 2001, p. 682-688. URL : http://papers.nips.cc/paper/1866-using-the-nystrom-method-to-speed-up-kernel-machines.pdf (visité le 15/06/2020) (cf. p. 16, 18).

[167] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov et al. « Deep Kernel Learning ». In : *Artificial Intelligence and Statistics*. 2016, p. 370-378 (cf. p. 38).

[168] Yuxin Wu et Kaiming He. « Group Normalization ». In : *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, p. 3-19 (cf. p. 31).

[169] Zonghan Wu, Shirui Pan, Fengwen Chen et al. « A Comprehensive Survey on Graph Neural Networks ». In : *IEEE Transactions on Neural Networks and Learning Systems* (2020) (cf. p. 118).

[170] Han Xiao, Kashif Rasul et Roland Vollgraf. *Fashion-MNIST : A Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. arXiv : 1708.07747 [cs, stat]. URL : http://arxiv.org/abs/1708.07747 (visité le 19/06/2020) (cf. p. 72).

[171] Fan YANG, Wei-hang LU, Lin-kai LUO et al. « Margin Optimization Based Pruning for Random Forest ». In : *Neurocomputing* 94 (2012), p. 54-63. ISSN : 0925-2312. DOI : 10.1016/j.neucom.2012.04.007. URL : http://www.sciencedirect.com/science/article/pii/S0925231212003396 (visité le 29/06/2020) (cf. p. 106).

[172] Zichao YANG, Marcin MOCZULSKI, Misha DENIL et al. « Deep Fried Convnets ». In : IEEE International Conference on Computer Vision (ICCV). 2015. arXiv : 1412.7149 [cs, stat]. URL : http://arxiv.org/abs/1412.7149 (visité le 12/05/2020) (cf. p. 35, 38, 41, 44, 57, 84).

[173] L. YAVITS, A. MORAD et R. GINOSAR. « Sparse Matrix Multiplication On An Associative Processor ». In : *IEEE Transactions on Parallel and Distributed Systems* (2017). arXiv : 1705.07282. URL : http://arxiv.org/abs/1705.07282 (visité le 01/06/2020) (cf. p. 36).

[174] Heping ZHANG et Minghui WANG. « Search for the Smallest Random Forest ». In : *Statistics and its interface* 2.3 (2009), p. 381. ISSN : 1938-7989. pmid : 20165560. URL : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2822360/ (visité le 29/06/2020) (cf. p. 106, 110).

[175] Jianguo ZHANG, Marcin MARSZAŁEK, Svetlana LAZEBNIK et al. « Local Features and Kernels for Classification of Texture and Object Categories : A Comprehensive Study ». In : *International journal of computer vision* 73.2 (2007), p. 213-238 (cf. p. 14).

[176] Kai ZHANG, Ivor W. TSANG et James T. KWOK. « Improved Nyström Low-Rank Approximation and Error Analysis ». In : Proceedings of the 25th International Conference on Machine Learning (ICML). Helsinki, Finland : ACM Press, 2008, p. 1232-1239. ISBN : 978-1-60558-205-4. DOI : 10.1145/1390156.1390311. URL : http://portal.acm.org/citation.cfm?doid=1390156.1390311 (visité le 25/06/2020) (cf. p. 19).

[177] Shuai ZHANG, Jianxin LI, Pengtao XIE et al. *Stacked Kernel Network*. 2017. arXiv : 1711.09219. URL : https://arxiv.org/abs/1711.09219 (cf. p. 38, 41).

[178] Aojun ZHOU, Anbang YAO, Yiwen GUO et al. « Incremental Network Quantization : Towards Lossless CNNs with Low-Precision Weights ». In : Internation Conference on Learning Representations (ICLR). 2017. arXiv : 1702.03044. URL : http://arxiv.org/abs/1702.03044 (visité le 30/07/2020) (cf. p. 36).

[179] Michael ZHU et Suyog GUPTA. « To Prune, or Not to Prune : Exploring the Efficacy of Pruning for Model Compression ». In : International Conference on Learning Representations (ICLR). 2017. arXiv : 1710.01878. URL : http://arxiv.org/abs/1710.01878 (visité le 30/04/2020) (cf. p. 36, 89, 96).