UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE

APPLIQUÉES

PRÉSENTÉ PAR

YOUSSEF BARKAOUI

CLASSIFICATION PROFONDE ET RÉSEAUX DE NEURONES : APPLICATION EN SCIENCE DES DONNÉES

MARS 2022

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Liste des acronymes et d'abréviations

ADAM	Adaptive Moment Estimation
ANN	Artificial Neural Networks
ASPC	Clustering avec apprentissage en rythme libre
ASPCDA	Clustering avec apprentissage en rythme libre avec l'augumentation des données
CAE	Clustering Autoencoder
CNN	Reseau de neurones convolutionnel
Datatest	La base MNIST de test à 10000 enregistrements
Datatrain	La base MNIST d'entrainement à 60000 enregistrements
DCAE	Deep Clustering Autoencoder
DCEC	Deep Clustering Embedded Clustering
DEC	Deep Embedded Clustering
DNN	Deep Neural Networks ou réseaux profonds
FA	La fonction d'activation
FD	La fonction de décision de la SVM
MLP	Multi Layer Perceptron
RBF	Le noyau gaussien
RELU	Unité Linéaire Rectifiée
SGD	Decente de gradient stochastique
SPL	Self paced Learning ou apprentissage en rythme libre
SVM	Support Vector Machine

Résumé

Les réseaux de neurones artificiels ont montré des résultats prometteurs sur les problèmes de classification et d'autres tâches complexes. En particulier, la méthode neuronale basée sur la classification profonde a pu contourner plusieurs limitations des méthodes de classification classiques, principalement dans l'analyse des bases de données complexes. Néanmoins, plusieurs méthodes neuronales ne sont pas assez robustes pour analyser les classes qui se chevauchent et donc qui ne sont pas linéairement séparables.

Ce mémoire propose d'utiliser la classification profonde pour répondre à la problématique des classes non linéairement séparables. En particulier, on a amélioré les résultats obtenus, aussi bien au niveau de la classification que de la prédiction, sur la base de données MNIST relative aux chiffres manuscrits. Pour ce faire, nous avons utilisé la classification profonde basée sur les autoencodeurs. Ces derniers ont été optimisés par l'ajout de l'apprentissage adapté et la modification de leur architecture et de certains hyperparamètres.

Malgré de nombreuses années de recherche sur les réseaux de neurones profonds, ils sont souvent considérés comme une boîte noire, avec peu d'informations sur la dynamique des couches cachées. La raison en est que ces réseaux sont souvent de grande dimension, qu'ils contiennent de nombreuses couches et que leur architecture est généralement conçue de manière ad hoc.

Abstract

Artificial neural networks have shown promising results on classification problems and other complex tasks. In particular, the neural method based on deep classification has been able to overcome several limitations of classical classification methods mainly in the analysis of complex databases. Nevertheless, many neural methods are not robust enough to analyze overlapping classes that are not linearly separable.

This thesis proposes to use deep classification to address the problem of non-linearly separable classes. In particular, we have improved the results obtained, both in terms of classification and prediction, on the MNIST database of handwritten digits. To do so, we used deep classification based on autoencoders. The latter have been optimized by adding adaptive learning and modifying their architecture and some hyperparameters.

Despite many years of research on deep neural networks, they are often considered as a black box, with little information on the dynamics of the hidden layers. This is because neural networks are often very dimensional, contain many layers, and their architecture is usually designed in an ad hoc manner.

Avant-propos

Au terme de la rédaction de ce mémoire, je souhaiterais adresser mes vifs remerciements à mon directeur de recherche M. Usef Faghihi et à ma codirectrice Mme Nadia Ghazzali, pour leur appui pédagogique tout au long de la maitrise et dans la lecture et la formulation des corrections de fond et de forme. Je les remercie également du soutien financier qu'ils m'ont accordé. Mes remerciements vont également aux professeurs Alfred Michel Grundland et Ismaïl Biskri pour avoir accepté d'évaluer mon mémoire de maîtrise. Je tiens à exprimer ma reconnaissance au personnel administratif et au corps professoral du département de mathématiques et d'informatique de l'Université du Québec à Trois-Rivières.

Finalement, je remercie ma famille, pour leur soutien moral et leurs encouragements, mes amis et tous ceux et celles qui ont contribué de près ou de loin à la rédaction de ce mémoire.

Introduction

Le présent mémoire s'intéresse principalement aux réseaux de neurones profonds qui chapeautent les méthodes de classifications classiques au niveau de la couche latente. On s'intéresse également à les rendre robustes incorporant l'apprentissage adapté en rythme libre. Ce mémoire se décompose de cinq parties principales. Le premier chapitre présente le contexte de développement des réseaux de neurones ainsi que les aspects mathématiques qui prouvent rigoureusement leur existence, efficacité et Interprétabilité, puis les deux chapitres suivants sur les techniques et algorithmes de bases utilisées ainsi que l'état de l'art des réseaux de neurones. Nous discuterons ensuite le cadre général de l'intégration des algorithmes classiques de classification dans les réseaux de neurones, plus précisément dans les autoencodeurs, ainsi qu'on améliorera les résultats présentés dans [25].
Le dernier chapitre entamera l'incorporation de l'apprentissage en rythme libre et l'augmentation des données. La dernière partie présentera la conclusion du travail et propose d'autres perspectives afin d'améliorer les méthodes traitées dans des futurs travaux.

1 Intérêt et mise en contexte

Cette partie s'intéresse au contexte général d'apparition et de développement des réseaux de neurones. On introduira également des perspectives mathématiques pour bien comprendre leur fonctionnement, ainsi que les fondements mathématiques qui prouvent l'existence et l'efficacité de ces réseaux.
Une dernière section sera consacrée à l'explication du paradigme d'apprentissage des réseaux et leur raisonnement interne.

1.1 Réseaux de neurones artificiels

De nombreux travaux pionniers et majeurs dans le domaine de l'apprentissage automatique ont été produits et publiés à la fin des années 1980 [60]. Les premiers résultats remarquables ont été tirés suite à l'application d'algorithmes d'apprentissage profond à des chiffres manuscrits. Ce qui fait que les domaines de reconnaissance des manuscrits à être les premiers à bénéficier de ces résultats. LeCun [62] et Cores & Vapnik [21] sont les pionniers qui ont introduit des méthodes innovantes en apprentissage automatique tout en profitant du développement de la puissance de calcul des machines. Leurs travaux dans le domaine ont déclenché une cascade d'axes de recherche, par la suite, visant à améliorer, réévaluer et étendre le champ d'application des méthodes présentées, y compris pour les chiffres manuscrits.

Le réseau neuronal artificiel (ANN) est la pièce d'un système informatique conçu pour simuler la façon dont le cerveau humain analyse et traite l'information. L'unité de base du calcul dans un réseau neuronal est le *neuron*, souvent appelé nœud ou unité. Il reçoit des données en provenance d'autres nœuds ou d'une source externe et calcule une sortie. La figure 1[a] montre un neurone biologique et un neurone artificiel. Un type courant

et simple de réseau neuronal est le réseau neuronal à action directe. Il contient plusieurs neurones, disposés en couches. Les nœuds des couches adjacentes sont reliés entre eux par des connexions ou des arêtes. Toutes ces connexions sont associées à des poids. Un exemple est illustré à la figure 1[b].

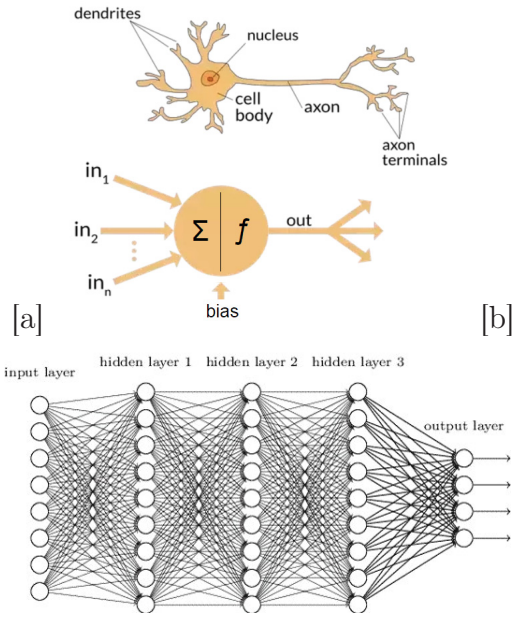

Figure 1 – Visualisation (a) des neurones biologiques et artificiels et (b) un exemple d'un réseau neuronal

Au cours de la dernière décennie, les approches ANN ont permis de réaliser des progrès remarquables et d'améliorer l'état de l'art dans de nombreux domaines pratiques [34, 115, 42, 41] et ont impliqué des avancées et des développements dans de nombreux domaines de recherche, notamment la classification, la reconnaissance des objets [24, 100], la vision par ordinateur [110, 44], la traduction des langues [26], les voitures autonomes [10, 86], les soins de santé [28, 73], etc.

Il existe une multitude d'architectures de réseaux de neurones, bien que la structure fondamentale de ces modèles soit la même, chaque architecture présente des mécanismes de raisonnement internes différents qui méritent

d'être explorés et sont souvent utilisés dans une application spécifique. Les modèles de réseaux de neurones les plus connus sont les suivants : Le perceptron multicouche (MLP) [29] qui est le modèle le plus basique et qui n'a pas connu de succès pendant longtemps, car il reste généralement bloqué dans un optimum local lors de la formation, le réseau neuronal convolutif (CNN) [2] est surtout utilisé pour la reconnaissance faciale - la recherche et l'édition d'images, le réseau neuronal récurrent (RNN) [72] est appliqué à la traduction automatique, à la reconnaissance vocale et à la prédiction de séries temporelles. L'autocodeur (AE) [78] est un célèbre modèle de réduction de la dimensionnalité et les réseaux adversariaux génératifs (GAN) [32] sont utilisés pour générer de nouvelles instances synthétiques de données qui peuvent passer pour des données réelles[31].

1.2 Aspects mathématiques

1.2.1 Un problème d'approximation

Une perspective mathématique simple pour comprendre les réseaux de neurones est de les considérer comme un problème d'optimisation ou d'approximation [85].

Étant donné un ensemble de données $S = \{(\boldsymbol{x}_i, y_i = f^*(\boldsymbol{x}_i), i \in \{1, 2, ..., n\}\}$, le but est d'approximer f^* aussi précisément que possible. Si f^* prend des valeurs continues, on appelle cela un problème de régression. Si f^* prend des valeurs discrètes, cela s'appelle un problème de classification.

Pour simplifier, nous négligerons ce que l'on appelle le « bruit de mesure », car il ne change pas grand-chose à l'image globale que nous allons décrire, même s'il est important pour un certain nombre de questions spécifiques.

Nous supposerons que $\boldsymbol{x}_i \in X = [0, 1]^d$, et nous désignons par P la distribution de $\{\boldsymbol{x}_i\}$.

Nous supposons également pour simplifier que $\sup_{x \in X} |f^*(\boldsymbol{x})| \leq 1$.

Il s'agit manifestement d'un problème d'approximation de fonction. En tant que tel, il peut être considéré soit comme un problème d'analyse numérique, soit comme un problème de statistique.

La procédure standard d'apprentissage supervisé est la suivante :

1. Choisir un espace d'hypothèses, l'ensemble des fonctions d'essai, qui sera noté $\mathcal{H}_m$. En analyse numérique classique, on utilise souvent des polynômes ou des polynômes par morceaux. En apprentissage automatique moderne, il est beaucoup plus courant d'utiliser des modèles de réseaux de neurones. L'indice m caractérise la taille du modèle. Il peut s'agir du nombre de paramètres ou de neurones, et sera précisé ultérieurement pour chaque modèle.

2. Choisissez une fonction de perte. Notre objectif principal est d'ajuster les données. Par conséquent, le choix le plus populaire est le « risque empirique »[85] :

$$\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_i \left(f\left(\boldsymbol{x}_i\right) - y_i\right)^2 = \frac{1}{n}\sum_i \left(f\left(\boldsymbol{x}_i\right) - f^*\left(\boldsymbol{x}_i\right)\right)^2$$

Parfois, on ajoute des termes de régularisation.

3. Choisir un algorithme d'optimisation et les hyperparamètres. Les choix les plus populaires sont la descente de gradient (GD), la descente de gradient stochastique (SGD) et les optimiseurs avancés tels que Adam[52].

L'objectif global est de minimiser le « risque de population », également connu sous le nom d'« erreur de généralisation » [85] :

$$\mathcal{R}(f) = \mathbb{E}_{\boldsymbol{x}}\left(f(\boldsymbol{x}) - f^*(\boldsymbol{x})\right)^2$$

1.2.2 Fondements théoriques

La réponse à la question "Pourquoi les réseaux neuronaux fonctionnent-ils si bien dans la pratique ?" est certainement basée sur le fait que les réseaux neuronaux peuvent bien approximer une grande famille de fonctions réelles qui dépendent de variables d'entrée. L'objectif de ce chapitre est de fournir les preuves mathématiques de ce comportement pour différentes variantes de cibles. Le fait que les réseaux neuronaux se comportent comme des approximateurs universels se traduit par le fait que leur sortie est une approximation précise à tout degré de précision souhaité de fonctions. Le processus d'obtention d'une approximation précise est interprété comme un processus d'apprentissage d'une fonction cible donnée. Le chapitre traite l'existence du processus d'apprentissage, mais il ne fournit pas de construction explicite du réseau. L'idée suivie ici est qu'il est logique de savoir d'abord qu'une solution existe avant de commencer à la chercher.

Dans la théorie mathématique des réseaux de neurones artificiels, les théorèmes d'approximation universels sont des résultats qui établissent la densité d'une classe de fonctions générées algorithmiquement dans un espace de fonctions d'intérêt donné. Typiquement, ces résultats concernent les capacités d'approximation de l'architecture feedforward sur l'espace des fonctions continues entre deux espaces euclidiens et l'approximation se fait par rapport à la topologie de convergence compacte.

Pour ceci on introduira les théorèmes fondamentaux de l'existence des réseaux neuronaux, à savoir « les théorèmes d'approximation universels », en fournissant des démonstrations dans le cas simple des fonctions continues. Le lecteur pourrait découvrir plus en profondeur les démonstrations des autres théorèmes qu'on énoncera dans le septième chapitre de la deuxième partie du livre « Deep Learning Architectures, A Mathematical Approach »[13].

On commencera par annoncer le **lemme 1** qui prouve l'existence des approximations des fonctions lipschitziennes dans le cas d'une seule dimension, puis on étendra ce résultat dans le **lemme 2** pour les dimensions supérieures.

Lemme 1. Supposons que $g : \mathbb{R} \to \mathbb{R}$ est ρ-Lipschitzienne i.e.
$\forall (x, y) \in \mathbb{R}^2,\ |f(x) - f(y)| \leq \rho\,|x - y|$. Pour toute erreur $\epsilon > 0$, il existe un réseau à 2 couches f avec $\left\lceil \frac{\rho}{\epsilon} \right\rceil$ nœuds(ou $\lceil x \rceil$ représente la partie entière de x) $z \mapsto \mathbf{1}[z \geq 0]$ tels que $\sup_{x \in [0,1]} |f(x) - g(x)| \leq \epsilon$

Démonstration. On pose: $m = \left\lceil \frac{\rho}{\epsilon} \right\rceil$, et $b_i = i\epsilon/\rho$ pour $i \in \{0, \ldots, m-1\}$, et

$$a_0 = g(0), \quad a_i = g\left(b_i\right) - g\left(b_{i-1}\right)$$

et soit $f(x) = \sum_{i=0}^{m-1} a_i \mathbf{1}\left[x_i \geq b_i\right]$. Alors pour tout $x \in [0, 1]$, et k le plus grand indice tel que $b_k \leq x$, alors f est constant sur $[b_k, x]$, et rappelons que $\sum_{i=1}^{k} \left(g\left(b_i\right) - g\left(b_{i-1}\right)\right)$ une somme télescopique qui vaut $\left|g\left(b_k\right) - g\left(b_0\right)\right|$:

$$\begin{aligned}
|g(x) - f(x)| &\leq |g(x) - g\left(b_k\right)| + \left|g\left(b_k\right) - f\left(b_k\right)\right| + \underbrace{\left|f\left(b_k\right) - f(x)\right|}_{0} \\
&\leq \rho\,|x - b_k| + \left|g\left(b_k\right) - \sum_{i=0}^{k} a_i\right| \\
&\leq \rho(\epsilon/\rho) + \left|g\left(b_k\right) - g\left(b_0\right) - \sum_{i=1}^{k} \left(g\left(b_i\right) - g\left(b_{i-1}\right)\right)\right| \\
&= \epsilon
\end{aligned}$$

□

Lemme 2. Soit une fonction continue $g : \mathbb{R}^d \to \mathbb{R}$ et $\epsilon > 0$, il existe $\delta > 0$ tel que $\|x - x'\|_\infty \leq \delta$ implique $|g(x) - g\left(x'\right)| \leq \epsilon$ et soit l'ensemble $U \subset \mathbb{R}^d$, avec une partition $\mathcal{P}$ de U en rectangles (produits d'intervalles) $\mathcal{P} = (R_1, \ldots, R_N)$ avec une longueur inférieure à δ. Alors, il existe des

scalaires $(\alpha_1, \ldots, \alpha_N)$ tel que

$$\sup_{x \in U} |g(x) - h(x)| \leq \epsilon, \quad \text{avec} \quad h = \sum_{i=1}^{N} \alpha_i \mathbf{1}_{R_i}$$

Démonstration. Soit une partition $\mathcal{P} = (R_1, \ldots, R_N)$, et pour tout R_i, on choisit $x_i \in R_i$ et un ensemble $\alpha_i := g(x_i)$. On a bien la longueur R_i inférieure à δ

$$\begin{aligned}
\sup_{x \in U} |g(x) - h(x)| &= \sup_{i \in \{1,\ldots,N\}} \sup_{x \in R_i} |g(x) - h(x)| \\
&\leq \sup_{i \in \{1,\ldots,N\}} \sup_{x \in R_i} \left(|g(x) - g(x_i)| + |g(x_i) - h(x)|\right) \\
&\leq \sup_{i \in \{1,\ldots,N\}} \sup_{x \in R_i} \left(\epsilon + |g(x_i) - \alpha_i|\right) = \epsilon
\end{aligned}$$

□

Théorème 1. Soit une fonction continue $g : \mathbb{R}^d \to \mathbb{R}$ et $\epsilon > 0$, il existe $\delta > 0$ tel que $\|x - x'\|_\infty \leq \delta$ implique $|g(x) - g(x')| \leq \epsilon$ comme défini dans le **lemme 2**, Alors, il existe un réseau à 3 couches ReLU f avec $\int_{[0,1]^d} |f(x) - g(x)| \mathrm{d}x \leq 2\epsilon$

Le théorème dans le cas multivarié est intéressant vu qu'il assure l'existence des réseaux peu profonds. Toutefois, il se concentre sur les fonctions lipschitziennes et ne considère pas la largeur du réseau. Par la suite, on étendra ces théorèmes pour les fonctions mesurables et intégrables.

Démonstration. Pour procéder, on considère de même une partition de U, en utilisant le lemme 2, on est sûr de l'existence de $h = \sum_{i=1}^{N} \alpha_i \mathbf{1}_{R_i}$ qui satisfait $\| g - h \|_1 \leq \epsilon$. Intuitivement, on sait que f est de forme : $f(x) := \sum_{i=1}^{N} \alpha_i g_i(x)$ avec les g_i des réseaux neuronaux à 1 couche cachée. le but à démontrer est :$\| f - g \|_1 \leq 2\epsilon$ s'obtient avec l'inégalité triangulaire.

□

Après avoir démontré l'existence d'un réseau de neurone approximant toute fonction lipschitzienne et intégrable d'une profondeur fini, mais ne précisant pas la largeur.

Le théorème d'approximation universelle stipule que toute fonction continue $f : [0,1]^n \to [0,1]$ peut être approximée arbitrairement bien par un réseau neuronal comportant au moins une couche cachée avec un nombre fini de poids, ce que nous allons illustrer dans les prochaines sous-sections. Mathématiquement, la forme classique du théorème d'approximation universelle pour une largeur arbitraire et une profondeur bornée est la suivante.

Théorème 2. Universal Approximation Theorem:

Soit une fonction continue $\sigma : \mathbb{R} \to \mathbb{R}$ (fonction d'activation) et des entiers positifs d, D.

La fonction σ n'est pas un polynôme si et seulement si, pour toute fonction continue $f : \mathbb{R}^d \to \mathbb{R}^D$ (fonction cible), tout sous-ensemble compact K de $\mathbb{R}^d$, et tout $\epsilon > 0$ il existe une fonction continue $f_\epsilon : \mathbb{R}^d \to \mathbb{R}^D$ (la sortie de la couche) avec représentation

$$f_\epsilon = W_2 \circ \sigma \circ W_1,$$

où W_2, W_1 sont des fonctions affines composables et $\circ$ désigne la composition , telle que la borne supérieure d'approximation.

$$\sup_{x \in K} \|f(x) - f_\epsilon(x)\| < \varepsilon$$

tient pour tout ϵ arbitrairement petit (la distance de f à f_ϵ peut-être infiniment petite).

Le théorème stipule que le résultat de la première couche f_ϵ peut approcher toute fonction bien conformée f. Une telle fonction bien conduite peut

également être approximée par un réseau de plus grande profondeur en utilisant la même construction pour la première couche et en approximant la fonction d'identité avec les couches ultérieures. Donc ce théorème démontre la capacité d'un réseau neuronal peu profond (de 2 couches cachées au maximum) à approximer n'importe quelle fonction continue. Cependant, le théorème ne tient pas en considération la largeur, car un tel réseau neuronal pourrait avoir un très grand nombre d'unités cachées
Le théorème de représentation de Kolmogorov-Arnold (ou théorème de superposition) stipule que toute fonction continue multivariable peut être représentée comme une superposition de fonctions continues d'une variable[46].

Théorème 3. Kolmogorov-Arnold:
Toute fonction continue $f : [0,1]^n \to \mathbb{R}$ peut s'écrire comme suit

$$f(x) = f(x_1, \ldots, .., x_n) = \sum_{q=1}^{Z_m} \phi_q \left(\sum_{q=1}^{m} \Psi_q (x_q) \right)$$

avec ϕ_q, Ψ_q sont des fonctions continues d'une variable.

Cela implique, entre autres, que si nous pouvons choisir la non-linéarité de chaque unité, nous pouvons représenter n'importe quelle fonction continue exactement avec un NN à 1 couche cachée.

Dans les sections précédentes, nous nous sommes concentrés sur le cadre des réseaux de neurones limités en profondeur. Ensuite, nous verrons des résultats intéressants pour les réseaux neuronaux limités en largeur.

Théorème 4. Théorème d'approximation universel pour les réseaux activés par ReLU limités en largeur [46] Pour toute fonction de Lebesgue-intégrable $f : \mathbb{R}^n \to \mathbb{R}$ et tout $\epsilon > 0$, il existe un réseau ReLU entièrement connecté A avec une largeur $d_m \leq n + 4$, tel que la fonction F_A représentée par ce réseau satisfait:

$$\int_{\mathbb{R}^n} |f(x) - F_A(x)|\, dx < \epsilon$$

Théorème 5. Universal Approximation Theorem (contrainte en largeur arbitraire).

Soit $\mathcal{X}$ un sous-ensemble compact de $\mathbb{R}^d$. Soit $\sigma : \mathbb{R} \to \mathbb{R}$ tout non-affine fonction continue qui est différentiable en au moins un point, avec une dérivée non nulle en ce point.

Soit $\mathcal{N}^{\sigma}_{d,D:d+D+2}$ désigne l'espace des réseaux neuronaux à action directe avec d neurones d'entrée, D neurones de sortie, et un nombre arbitraire de couches cachées comportant chacune $d + D + 2$ neurones, tels que chaque neurone caché a une fonction d'activation σ et chaque neurone de sortie a la fonction identité comme fonction d'activation, avec la couche d'entrée ϕ, et la couche de sortie ρ. Alors, étant donné toute $\varepsilon > 0$ et toute $f \in C(\mathcal{X}, \mathbb{R}^D)$, il existe $\hat{f} \in \mathcal{N}^{\sigma}_{d,D:d+D+2}$ telle que

$$\sup_{x \in \mathcal{X}} \left\| \hat{f}(x) - f(x) \right\| < \varepsilon.$$

i.e, $\mathcal{N}$ est dense in $C(\mathcal{X}; \mathbb{R}^D)$ par rapport à la topologie uniforme.

Les résultats prouvés dans ce chapitre montrent qu'un réseau neuronal à une couche cachée et une activation linéaire dans la couche de sortie peut apprendre des fonctions continues, des fonctions intégrables, ainsi que des fonctions mesurables. La qualité des réseaux neuronaux pour être des approximateurs universels potentiels n'est pas le choix spécifique de la fonction d'activation, mais plutôt le choix de la fonction d'activation, mais plutôt l'architecture du réseau feedforward elle-même. Le prix à payer est que le nombre de neurones dans la couche cachée n'a pas de limite a priori. Cependant, il existe un résultat indiquant que pour chaque chiffre supplémentaire de gain de précision de l'approximation de la cible, le nombre de neurones cachés doit être multiplié par 100. Les résultats de l'approximation fonctionnent également si la fonction d'activation est

une ReLU ou sigmoid [46]. Cependant, la preuve n'est pas une modification simple des preuves fournies ci-dessus, mais plutôt des méthodes bien spécifiques au choix de la fonction d'activation. Le compromis entre la profondeur, la largeur et l'erreur d'approximation est toujours un sujet de recherche actif. Par exemple, en 2017, Lu et al. [68] ont prouvé un théorème d'approximation universel pour les réseaux de neurones profonds limités en largeur avec ReLU qui peuvent approximer toute fonction intégrable de Lebesgue sur un espace d'entrée à n dimensions. Peu de temps après, Hanin [40] a amélioré ce résultat en utilisant des réseaux ReLU pour approximer toute fonction convexe continue de variables d'entrée à n dimensions.

1.2.3 Un système dynamique

Les réseaux neuronaux peuvent également être considérés comme des systèmes dynamiques. La manière la plus directe de relier pratiquement n'importe quel réseau neuronal aux systèmes dynamiques consiste à considérer les poids apprenables comme l'état d'un système discret, provenant d'un espace de paramètres euclidien. La carte est définie par la mise à jour des poids pour chaque étape de la descente de gradient. Cela ouvre la voie à une compréhension plus approfondie du fonctionnement de l'opération de rétropropagation, qui à son tour éclaire la compréhension des réseaux neuronaux en général.

Nous pouvons également considérer les réseaux de neurones comme un graphe orienté dont les nœuds sont les neurones et les arêtes les poids synaptiques. Chaque nœud est caractérisé par une équation d'évolution où l'état du neurone dépend de ses voisins. En suivant les notations de [14] nous pouvons caractériser chaque neurone i par son état, X_i, qui appartient à un ensemble compact $\mathcal{I} \subset \mathbb{R}^M$, où M est le nombre de variables carac-

térisant l'état d'un neurone. D'après [14], on peut modéliser l'évolution de N neurones, donnée par un système dynamique déterministe,

$$\frac{d\mathbf{X}}{dt} = \mathbf{F}_\gamma(\mathbf{X}, t), \text{temps continue}, \tag{1}$$

où,

$$\mathbf{X}(t+1) = \mathbf{F}_\gamma[\mathbf{X}(t), t], \quad \text{temps discret} . \tag{2}$$

La variable $\mathbf{X} = \{X_i\}_{i=1}^N$ représente l'état dynamique d'un réseau de N neurones au temps t. Typiquement, $\mathbf{X} \in \mathcal{M} = \mathcal{I}^N$ où $\mathcal{M}$ est l'espace d'état de Eq.(2), et la carte $F_\gamma : \mathcal{M} \to \mathcal{M}$ dépend d'un ensemble de paramètres $\gamma \in \mathbb{R}^p$. De manière générale, en regardant l'Eq.(5), on peut dire qu'elle obéit à la forme d'un système dynamique discret où le temps correspond à la profondeur du réseau.

Ceci s'accorde particulièrement bien avec les RNN, où les activations cachées suivent une formule de récursion dans chaque couche du réseau. Dans [101], ces réseaux étaient considérés comme continus dans le temps, où les auteurs considéraient un RNN respectant les équations suivantes :

$$\dot{x}_i = \frac{dx}{dt} = -x_i + \sum_k^N J_{ik} r_k + \sum_k^I B_{ik} u_k \tag{3}$$

$$N r_i = h(x_i) \tag{4}$$

où x est un état à N dimensions appelé les activations et $r = h(x)$ sont les taux de tir, définis comme l'application par élément de la fonction non linéaire h à x. Dans cette optique, les auteurs de [101] ont pu révéler des aspects cruciaux de la manière dont les RNN mettent en œuvre leurs calculs en étudiant les points fixes de ce système dynamique.

Cependant, l'examen des réseaux neuronaux à action directe, comme dans la figure 1, révèle un défi dans le cadre de [14]. Chaque couche ℓ est de

largeur N_ℓ est définie par l'équation

$$x^\ell = \phi(h^\ell),\ h^\ell = W^\ell x^{\ell-1} + b^\ell, \tag{5}$$

avec $W^\ell \mathbb{R}^{N_{\ell-1} \times N_\ell}$ une matrice de poids, b^ℓ le vecteur de biais, h^ℓ la pré-activation, x^ℓ la post-activation, et enfin $\phi : \mathbb{R} \to \mathbb{R}$ la fonction d'activation.

1.3 Explication et interprétation des réseaux de neurones profonds

Bien que les réseaux de neurones artificiels figurent parmi les outils de prédiction les plus puissants aujourd'hui, leur raisonnement interne est très complexe et encore peu compris. Jusqu'à présent, il n'existe pas dans la littérature d'outil complet basé sur une théorie solide qui explique ou interprète leur structure interne ou leur paradigme d'apprentissage. Les réseaux d'apprentissage profond sont souvent qualifiés de "boîtes noires" [1, 94]. La raison en est que ces modèles sont généralement extrêmement grands et compliqués, contenant un très grand nombre de paramètres à optimiser. Leur manque d'interprétabilité est également considéré comme leur malédiction. Le mystère de la dynamique interne des ANN crée un défi pour la communauté de l'IA [66] car ces modèles sont utilisés pour des prises de décision à fort enjeu et pénètrent des domaines critiques tels que les soins de santé, le système de justice pénale, les voitures à conduite autonome et les marchés financiers. L'incapacité d'interpréter et de comprendre les ANNs semble problématique [90, 66].

Récemment, de plus en plus de scientifiques se sont penchés sur le problème de l'interprétabilité des réseaux neuronaux et ce domaine de recherche connaît une croissance rapide. La littérature académique a offert une myr-

iade de techniques qui tentent d'expliquer le phénomène d'apprentissage des ANNs, mais ces techniques sont généralement erronées et trompeuses, et elles fournissent peu d'indications pour "opening the black box problem". En d'autres termes, ces méthodes explicatives ne peuvent en général pas corriger les idées fausses intégrées dans les modèles ANN pendant la formation.

Selon [74] l'interprétabilité est définie comme la mise en correspondance d'un concept abstrait dans un domaine qui a du sens pour les humains, tandis que l'explication est définie comme la collection de caractéristiques du domaine interprétable, qui ont contribué pour un exemple donné à produire une décision. Dans le même article, les auteurs présentent diverses techniques issues de la littérature pour interpréter et expliquer les modèles DNN, ces techniques sont soit basées sur des méthodes statistiques et probabilistes, soit sur l'analyse et les expansions de Taylor, soit sur la compréhension théorique des couches internes des DNN.

Un résultat intéressant a été proposé dans [88], où le fonctionnement des réseaux ReLU profonds est étudié, et une relation entre les sorties du réseau et ses poids est établie. Le cadre [14] discuté ci-dessus, a motivé des applications spécifiques, par exemple [82], qui étudie la nature de la propagation du signal dans les réseaux de neurones profonds. Les auteurs montrent que même les réseaux neuronaux profonds aléatoires peuvent construire des représentations internes cachées dont la courbure extrinsèque globale croît exponentiellement avec la profondeur mais pas avec la largeur et peuvent apprendre des fonctions d'une classe générale de non-linéarités. Dans [92], les auteurs ont étudié la dynamique de l'apprentissage (par exemple, les poids, les activations pendant la formation) en trouvant des solutions exactes de ces dynamiques non linéaires en termes de statistiques d'entrée-sortie. Cela pourrait aider, par exemple, à comprendre comment la forma-

tion s'accélère ou se ralentit en fonction de la profondeur du réseau.

Un autre travail intéressant dans [1] propose de surveiller les caractéristiques à chaque couche d'un modèle DNN en branchant et en entraînant un classificateur linéaire appelé "*probe*". Ce réseau de sonde peut mesurer l'aptitude du modèle à la classification, il peut également être utilisé comme un outil d'explication/interprétation de la dynamique interne. Dans l'article [94], les auteurs proposent d'analyser les réseaux neuronaux profonds dans le *Plan d'Information*, qui est le plan de l'information mutuelle entre les couches internes du réseau, cette analyse peut également être utile pour comprendre le paradigme de formation, le comportement de décentrement stochastique du gradient pendant la formation et la convergence des couches pendant l'optimisation de la perte.

Une autre direction de recherche sur la problématique de l'ouverture de la boîte noire dans l'apprentissage profond est de trouver des représentations significatives dans leurs couches internes. Dans l'article [47], les auteurs abordent la question de l'apprentissage de la relation sémantique entre les objets dans les modèles DNN, ils ont prouvé que pour certains types de modèles (CNN), les couches internes intègrent des représentations de catégories d'objets qui sont organisées de manière hiérarchique. Un résultat similaire appliqué à différents ensembles de données a été trouvé dans les articles [80, 91]. Ces résultats nous indiquent que si les modèles d'apprentissage profond apprennent à réaliser des tâches complexes aussi parfaitement que les humains, ils sont également capables d'apprendre sur leur propre relation sémantique au sein des objets d'entrée.

Des travaux récents s'attaquent à la problématique de la saisie des représentations internes des réseaux neuronaux profonds en comparant les représentations entre les couches et entre différents modèles formés sur une tâche identique ou similaire. Dans [84] une nouvelle technique appelée "Singu-

lar Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability " (SVCCA) a été proposée, cette technique est basée sur la décomposition en valeurs singulières et la corrélation canonique. Cet outil permet de comparer les représentations internes et peut également suggérer de nouveaux régimes de formation pour éviter l'overfitting et réduire l'énorme coût de calcul. Une question importante lors de la comparaison des représentations est la suivante : "Des réseaux neuronaux différents apprennent-ils les mêmes représentations ?", c'est-à-dire que si nous formons différents réseaux neuronaux (par exemple, une architecture ou une initialisation différente) sur la même tâche, convergeront-ils vers la même représentation ? Bien que la technique SVCCA dans [84] soit simple, l'article montre qu'il est possible d'aligner l'espace d'activation neuronale de différents réseaux neuronaux dans certains cas. La même idée a été discutée dans [53] où les auteurs proposent l'utilisation de l'alignement du noyau centré (CKA) [22, 20] comme mesure de similarité entre représentations. Les auteurs ont montré empiriquement que pour les modèles CNN les plus courants formés sur des ensembles de données d'images, différents modèles apprennent des représentations similaires.

Avant de procéder à formaliser les théories d'apprentissage profond, on présentera les approches d'interprétabilité de ces systèmes qui semblent être les plus acceptées dans la communauté scientifique au cours des cinq dernières années.

1.3.1 Méthodes d'attribution de caractéristiques additives

La meilleure explication d'un modèle simple est le modèle lui-même ; il se représente parfaitement et est facile à comprendre. Pour les modèles complexes, tels que les méthodes d'ensemble ou les réseaux profonds, nous ne pouvons pas utiliser le modèle original comme sa propre meilleure ex-

plication, car il n'est pas facile à comprendre. Au lieu de cela, nous devons utiliser un modèle d'explication plus simple, que nous définissons comme toute approximation interprétable du modèle original. Nous montrons ci-dessous que six méthodes d'explication actuelles de la littérature utilisent toutes le même modèle d'explication. Cette unité jusqu'alors non appréciée à des implications intéressantes, que nous décrivons dans les sections suivantes.

Soit f le modèle de prédiction original à expliquer et g le modèle d'explication. Ici, nous nous concentrons sur les méthodes locales conçues pour expliquer une prédiction $f(x)$ basée sur une seule entrée x, comme proposé dans LIME [87]. Les modèles d'explication utilisent souvent des entrées simplifiées x' qui correspondent aux entrées originales par le biais d'une fonction de correspondance $x = h_x(x')$. Les méthodes locales essaient de garantir que $g(z') \approx f(h_x(z'))$ chaque fois que $z' \approx x'$. (Notez que $h_x(x') = x$ même si x' peut contenir moins d'informations que x car h_x est spécifique à l'entrée actuelle x).

Définition 1.1. Les méthodes d'attribution de caractéristiques additives ont un modèle d'explication qui est une fonction linéaire de variables binaires :

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i$$

où $z' \in \{0,1\}^M, M$ est le nombre de caractéristiques d'entrée simplifiées, et $\phi_i \in \mathbb{R}$.

Les méthodes dont les modèles d'explication correspondent à la définition 1 attribuent un effet ϕ_i à chaque caractéristique, et la somme des effets de toutes les attributions de caractéristiques se rapproche de la sortie $f(x)$ du modèle original. De nombreuses méthodes actuelles correspondent à la définition 1, dont plusieurs sont présentées ci-dessous.

La méthode LIME interprète les prédictions des modèles individuels en se basant sur une approximation locale du modèle autour d'une prédiction donnée. Le modèle d'explication linéaire local que LIME utilise adhère exactement à l'équation 1 et est donc une méthode d'attribution de caractéristiques additives. LIME désigne les entrées simplifiées x' comme des "entrées interprétables", et la correspondance $x = h_x(x')$ convertit un vecteur binaire d'entrées interprétables dans l'espace d'entrée original. Différents types de mappings h_x sont utilisés pour différents espaces d'entrée. Pour les caractéristiques de texte de type sac de mots, h_x convertit un vecteur de 1 ou 0 (présent ou non) en nombre de mots d'origine si l'entrée simplifiée est égale à un, ou à zéro si l'entrée simplifiée est égale à zéro. Pour les images, h_x traite l'image comme un ensemble de super-pixels ; il associe ensuite 1 au fait de laisser le super-pixel à sa valeur d'origine et 0 au fait de remplacer le super-pixel par une moyenne des pixels voisins (ceci est censé représenter l'absence). Pour trouver ϕ, LIME minimise la fonction objective suivante :

$$\xi = \arg\min_{g \in \mathcal{G}} L(f, g, \pi_{x'}) + \Omega(g)$$

La fidélité du modèle d'explication $g(z')$ au modèle original $f(h_x(z'))$ est assurée par la perte L sur un ensemble d'échantillons dans l'espace d'entrée simplifié pondéré par le noyau local $\pi_{x'}$. $\cdot\ \Omega$ pénalise la complexité de g. Puisque dans LIME g suit l'équation 1 et que L est une perte au carré, l'équation 2 peut être résolue en utilisant la régression linéaire pénalisée.
DeepLIFT [93] a été proposé en 2017 comme méthode d'explication de prédiction récursive pour l'apprentissage profond. Elle attribue à chaque entrée x_i une valeur $C_{\Delta x_i . \Delta_y}$ qui représente l'effet de cette entrée définie à une valeur de référence par rapport à sa valeur d'origine. Cela signifie que pour DeepLIFT, le mappage $x = h_x(x')$ convertit les valeurs binaires

en entrées originales, où 1 indique qu'une entrée prend sa valeur originale, et 0 indique qu'elle prend la valeur de référence. La valeur de référence, bien que choisie par l'utilisateur, représente une valeur de fond typique non informative pour la caractéristique du réseau. DeepLIFT utilise une propriété de "sommation à delta" qui stipule :

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta o} = \Delta o$$

où $o = f(x)$ est la sortie du modèle, $\Delta o = f(x) - f(r), \Delta x_i = x_i - r_i$, et r est l'entrée de référence. Si on laisse $\phi_i = C_{\Delta x_i \Delta o}$ et $\phi_0 = f(r)$, alors le modèle d'explication de DeepLIFT correspond à l'équation ci-dessous et donc une autre méthode d'attribution de caractéristiques additives.
La méthode de propagation de la pertinence par couche interprète les prédictions des réseaux profonds. Cette méthode est équivalente à DeepLIFT avec les activations de référence de tous les neurones fixées à zéro. Ainsi, $x = h_x(x')$ convertit des valeurs binaires dans l'espace d'entrée original, où 1 signifie qu'une entrée prend sa valeur originale, et 0 signifie qu'une entrée prend la valeur 0. Le modèle d'explication de la propagation de la pertinence par couche, comme celui de DeepLIFT, correspond à l'équation des caractéristiques additives.
Soit j et k les neurones de deux couches consécutives du réseau neuronal. La propagation des scores de pertinence $(R_k)_k$ d'une couche donnée sur les neurones de la couche inférieure s'effectue en appliquant la règle :

$$R_j = \sum_k \frac{z_{jk}}{\sum_j z_{jk}} R_k$$

La quantité z_{jk} modélise la mesure dans laquelle le neurone j a contribué à rendre le neurone k pertinent. Le dénominateur sert à faire respecter la propriété de conservation. La procédure de propagation se termine lorsque les caractéristiques d'entrée ont été atteintes. Si l'on utilise la règle ci-

dessus pour tous les neurones du réseau, il est facile de vérifier la propriété de conservation par couche $\sum_j R_j = \sum_k R_k$, et par extension la propriété de conservation globale $\sum_i R_i = f(\boldsymbol{x})$.

1.3.2 SHAP (SHapley Additive exPlanation)

Théorème 6. Un seul modèle d'explication possible g suit la définition des modèles additives localement précis et consistent:

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!\,(M - |z'| - 1)!}{M!} \left[f_x\left(z'\right) - f_x\left(z' \backslash i\right)\right]$$

où $|z'|$ est le nombre d'entrées non nulles dans z', $z' \backslash i$ désigne $z'_i = 0$ et $z' \subseteq x'$ représente tous les vecteurs z' dont les entrées non nulles sont un sous-ensemble des entrées non nulles dans x' [69].

Lundberg et al. [69] proposent les valeurs SHAP comme une mesure unifiée de l'importance des caractéristiques. Il s'agit des valeurs de Shapley d'une fonction d'espérance conditionnelle du modèle original ; elles sont donc la solution de l'équation du théorème SHAP, où $f_x\left(z'\right) = f\left(h_x\left(z'\right)\right) = E\left[f(z) \mid z_S\right]$, et S est l'ensemble des indices non nuls dans z' (figure 1). Cette définition des valeurs SHAP comporte implicitement un mappage d'entrée simplifié, $h_x\left(z'\right) = z_S$, où z_S a des valeurs manquantes pour les caractéristiques ne faisant pas partie de l'ensemble S. Comme la plupart des modèles ne peuvent pas gérer des modèles arbitraires de valeurs d'entrée manquantes, nous approximons $f\left(z_S\right)$ par $E\left[f(z) \mid z_S\right]$. Cette définition des valeurs SHAP est conçue pour s'aligner étroitement sur la régression de Shapley, l'échantillonnage de Shapley et les attributions quantitatives de caractéristiques d'influence d'entrée, tout en permettant également des connexions avec LIME, DeepLIFT et la propagation de pertinence en couches.

Le calcul exact des valeurs SHAP est difficile. Cependant, en combi-

nant les idées des méthodes actuelles d'attribution additive des caractéristiques, nous pouvons les approximer. Les auteurs décrivent deux méthodes d'approximation indépendantes du modèle, l'une déjà connue (valeurs d'échantillonnage de Shapley) et l'autre nouvelle (Kernel SHAP). Ils décrivent également quatre méthodes d'approximation spécifiques au type de modèle, dont deux sont nouvelles (Max SHAP, Deep SHAP). Lors de l'utilisation de ces méthodes, l'indépendance des caractéristiques et la linéarité du modèle sont deux hypothèses facultatives qui simplifient le calcul des valeurs attendues (notez que $\bar{S}$ est l'ensemble des caractéristiques qui ne sont pas dans S) :

$$\begin{aligned} f\left(h_x\left(z'\right)\right) &= E\left[f(z) \mid z_S\right] \\ &= E_{z\bar{S}|z_S}[f(z)] \\ &\approx E_{z_{barS}[f(z)]} \\ &\approx f\left(\left[z_S, E\left[z_{\bar{S}}\right]\right]\right). \end{aligned}$$

Les valeurs Shap attribuent à chaque caractéristique la modification de la prédiction attendue du modèle lors du conditionnement sur cette caractéristique. Comme décrit dans la figure 2, elles expliquent comment passer de la valeur de base $E[f(z)]$ qui serait prédite si nous ne connaissions aucune caractéristique à la sortie actuelle f(x). Ce diagramme montre un seul ordonnancement. Cependant, lorsque le modèle n'est pas linéaire ou que les caractéristiques d'entrée ne sont pas indépendantes, l'ordre dans le diagramme est différent. Les valeurs SHAP résultent de la moyenne des valeurs $\phi(x)$ sur tous les ordres possibles.

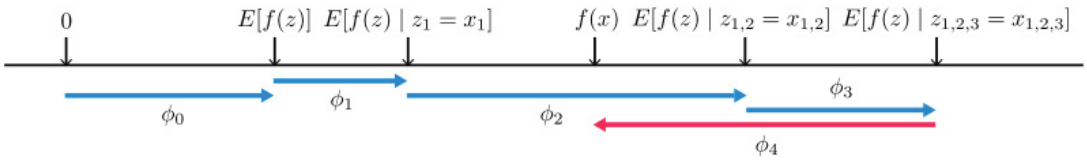

Figure 2 – Schéma explicatif de l'attribution des valeurs SHAP à chaque caractéristique [69]

2 Optimisation

2.1 Fonction de perte

Dans la classification, l'entropie croisée et le logarithme négatif de la vraisemblance (NLL) sont souvent utilisés comme perte[61]. On peut également utiliser la divergence de Kullback-Leibler (divergence KL)[62] comme fonction de perte. Formellement, l'entropie croisée est définie comme suit :

$$\epsilon(y_i, \hat{y}_i) = \begin{cases} -log(\hat{y}_i) & \text{if} \quad y_i = 1 \\ -log(1 - \hat{y}_i) & \text{if} \quad y_i = 0. \end{cases}$$

Nous pouvons traiter la perte NLL comme une extension de la perte d'entropie croisée de la classification binaire à la classification multi-classes. Si nous supposons que y est un vecteur à un coup qui indique à quelle classe appartient un exemple, $y_i = \mathbf{1}_{i \in C_j}$ où i est l'étiquette. la perte NLL est donc:

$$\epsilon(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

avec N le nombre total de classes.

La divergence KL $D_{\mathrm{KL}}(P \parallel Q)$ est une mesure de l'information perdue

lorsque Q est utilisé pour approximer P. Formellement, il est décrit comme suit :

$$\epsilon(y, \hat{y}) = D_{\mathrm{KL}}(P \parallel Q) \tag{6}$$

$$= \sum_i \log\left(\frac{Q(y \parallel x)}{P(y \parallel x)}\right) P(y \parallel x) \tag{7}$$

$$= \sum_i \log\left(\frac{y_i}{\hat{y}_i)}\right) y_i \tag{8}$$

Le gradient de la divergence KL :

$$\frac{\partial \epsilon_{KL}(y, \hat{y})}{\partial \hat{y}_i} = \frac{\partial}{\partial \hat{y}_i} y_i(log(y_i)log(\hat{y}_i)) \tag{9}$$

$$= \frac{\partial}{\partial \hat{y}_i}(y_i log(\hat{y}_i) \tag{10}$$

$$= \frac{\partial \epsilon_{NLL}(y, \hat{y})}{\partial \hat{y}_i} \tag{11}$$

d'où on conclut que minimiser la perte de vraisemblance logarithmique négative est exactement la même chose que la minimisation de la divergence KL.

2.2 Algorithmes du gradient

La descente de gradient est un algorithme d'optimisation du premier ordre utilisé pour trouver un minimum local d'une fonction objective.
L'algorithme de descente de gradient et sa version stochastique sont don-

nées par :

FindALocalMinima $\{x \in X : x = \arg\min_{y \in U^\circ,\ U \subset X} \epsilon(y)\}$

inputs : fonction de perte ϵ, taux d'apprentissage η, Nexemples (X,y), model $F(\theta, x)$

output: θ optimal minimizant le ϵ

while *non convergé* **do**

$\hat{y} \leftarrow F(\theta, x)$;

$\theta \leftarrow \theta - \eta . \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \epsilon(y, \hat{y})}{\partial \theta}$;

end

return θ;

Algorithme 1: Descente de gradient

SGD $\{x \in X : x = \arg\min_{y \in U^\circ,\ U \subset X} \epsilon(y)\}$

inputs : fonction de perte ϵ, taux d'apprentissage η, exemples (X,y), model $F(\theta, x)$

output: θ optimal minimizant le ϵ

while *non convergé* **do**

$melangerX, y$;

foreach $x_i, y_i \in X, y$ **do**

$\hat{y}_i \leftarrow F(\theta, x_i)$;

$\theta \leftarrow \theta - \eta . \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \epsilon(y_i, \hat{y}_i)}{\partial \theta}$;

end

end

return θ

Algorithme 2: Descente de gradient stochastique

Il existe également une forme populaire de descente de gradient appelée descente de gradient « mini-batch » qui divise les données d'apprentissage en plusieurs petits lots de taille 64, 128 ou plus, puis exécute la descente de gradient stochastique sur chaque mini-batch (en calculant la somme des gradients dans chaque mini-batch) pour optimiser le modèle. Le choix de la taille correcte des mini-lots est également un facteur de réussite de

l'optimisation par descente de gradient.
Le problème avec tels hyperparamètres est qu'ils doivent être choisis manuellement pour chaque session d'apprentissage donnée et qui peuvent varier considérablement en fonction du problème à traiter ou du modèle utilisé. Pour lutter contre ce problème, il existe de nombreux types d'algorithmes de descente de gradient adaptatifs tels que Adagrad, Adadelta, RMSprop et Adam [52].
Une des premiers majeurs contributions qui ont amélioré la descente du gradient fut l'ajout du momentum. En effet, le momentum ou le moment permet de continuer dans la même direction qu'à l'itération précédente, ce qui permet d'accélérer l'optimisation. En particulier, cette méthode est très efficace lorsque que la direction du gradient reste relativement la même.

L'idée est d'utiliser une moyenne mobile du gradient du paramètre au lieu de simplement utiliser le gradient réel actuel :

$$v_t = \beta.v_{t-1} - \eta.\nabla F(\theta_{t-1})$$
$$\theta_t = \theta_{t-1} + v_t$$

où θ désigne les paramètres du modèle, v est le momentum du gradient, est un facteur de momentum, et η est le taux d'apprentissage pour le t^{eme}

cycle d'apprentissage.

Adam g_t^2 *indique le carré par éléments carré ou le produit Hadamrd* $(g_t \circ g_t)_{ij} = (g_t \odot g_t)_{ij} = (g_t)_{ij}(g_t)_{ij}$. *Les bons paramètres par défaut pour les problèmes d'apprentissage automatique testés par les auteurs sont* α= *0,001,* β_1 = *0.9,* β_2 = *0.999 et* $\epsilon = 10^{-8}$.

inputs : α *: Stepsize*

$\beta_1, \beta_2 \in [0, 1)$*: Taux de décroissance exponentielle pour les estimations du moment.*

$f(\theta)$ *: Fonction objectif stochastique avec des paramètres* θ

θ_0 *: Vecteur initial des paramètres*

output: θ *minimize F*

$m_0 \leftarrow 0$ *(Initialisation du vecteur de moment 1);*

$v_0 \leftarrow 0$ *(Initialiser le vecteur du 2ème moment);*

$t \leftarrow 0$*(Initialiser le pas de temps);*

while θ_t*non convergé* **do**

$t \leftarrow t + 1$;

$g_t \leftarrow \nabla_\theta f_t(_{t-1})$ *(Obtenir les gradients par rapport à l'objectif stochastique au pas de temps t)*

$m_t \leftarrow \beta_1.m_{t-1} + (1 - \beta_1) - g_t$ *(Mise à jour de l'estimation du premier moment biaisé)*

$v_t \leftarrow \beta_2 - v_{t-1} + (1 - \beta_2) - g_t^2$ *(Mise à jour de l'estimation biaisée du second moment brut)*

$\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$*(Calculer la première estimation du moment corrigée du biais)*

$\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ *(Calcul de la deuxième estimation brute du moment corrigée du biais)*

$\theta_t \leftarrow \theta_{t-1} - \alpha.\hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ *(Mise à jour des paramètres)*

end

return θ_t*;*

Algorithme 3: Algorithme Adam [52]

2.3 Barème des taux d'apprentissage

En général, si nous utilisons un grand taux d'apprentissage η, nous pouvons faire converger l'apprentissage plus rapidement, mais ce choix conduira à une divergence. Les petites valeurs de η sont plus susceptibles de d'être piégées dans des minima locaux. Pour s'assurer que la formation(training) converge, nous devons réduire le taux d'apprentissage pendant la formation.

Il existe trois méthodes couramment utilisées pour programmer la décrémentation du taux d'apprentissage [99] : constant, factorisé, et décroissance exponentielle. Le taux d'apprentissage est programmé pour changer après ζ étapes. La planification constante réduit le taux d'apprentissage selon une fonction de pas définie manuellement avec un ζ arbitraire. Le programme exponentiel calcule le taux d'apprentissage comme suit :

$$\eta_t = \eta_0.\gamma^{t/\epsilon} \tag{12}$$

avec γ le taux de décroissance.

Le programme factorisé est similaire au programme exponentiel,mais dans un format de fonction pas à pas :

$$\eta_t = \eta_0.\gamma^{\lfloor t/\epsilon \rfloor} \tag{13}$$

En utilisant les programmes constants ou pondérés, nous pouvons facilement juger de l'effet de décroissance du taux d'apprentissage en visualisant la courbe d'apprentissage. En général, on utilise $\gamma = 0.1$ pour réduire le taux d'apprentissage 10 fois à chaque étape.

2.4 Décroissance des poids

La décroissance des poids est également couramment utilisée dans la communauté des réseaux neuronaux pour effectuer une régularisation L2 pendant la formation. pour effectuer une régularisation L2 pendant la formation. La régularisation est essentielle pour éviter le overfitting et améliorer la généralisation du modèle. Formellement, le régularisateur L2 pour une fonction $F(\theta, x)$ est donné par la formule suivante :

$$\Omega(\theta) = \| \theta \|^2$$

$$\hat{\epsilon}(F(\theta, x), y) = \epsilon(F(\theta, x), y) + \frac{1}{2}\lambda\Omega(\theta)$$

La décroissance des poids est la méthode de régularisation par défaut utilisée pour la formation des réseaux de neurones. réseaux neuronaux. Habituellement, nous fixons λ à 0,0004. En cas de normalisation supplémentaire, telle que Dropout et la normalisation par lots , un λ plus petit accélérera la formation.

2.5 Batch Normalisation

Comme son nom l'indique, cette technique vise à normaliser chaque exemple du lot selon les paramètres de distribution de ce dernier. C'est-à-dire, enlever la moyenne et diviser par l'écart-type. Plus formellement :

$$x_i = \frac{x_i \mathbb{E}[X]}{\sqrt{Var[X]+\epsilon}}$$

Cette technique a plusieurs avantages. Notamment, elle accélère l'entraînement, et tend aussi à entraîner des réseaux plus performants.

2.6 Dropout

Le Dropout est une méthode de régularisation proposée pour la première fois dans Srivastava et al. [98] qui réduit considérablement le sur-ajustement des réseaux neuronaux et améliore les performances de généralisation. Le Dropout est rapidement devenu un composant par défaut des méthodes de formation de réseaux neuronaux. Formellement, la formation avec Dropout pour la couche l s'exprime comme suit par :

$$r_j^{(l)} = Bernoulli(p)$$

$$\hat{h}^{(l)} = r_j^{(l)} * h^{(l)}$$

où $h^{(l)}$ est la sortie originale de la couche l, et $\hat{h}^{(l)}$ est la nouvelle sortie de la couche l avec Dropout et $*$ définit dans ce cas comme étant le produit matriciel d'Hadamard tel que $(A * B)_{i,j} = (A)_{i,j} \times (B)_{i,j}$. Pour obtenir un résultat déterministe au moment du test, nous utilisons l'espérance comme sortie:

$$\mathbb{E}[\hat{h}_j^{(l)}] = p * 0 + (1-p) * h_j^{(l)} = (1-p) * h_j^{(l)}$$

L'espérance est une moyenne approximative d'un nombre exponentiel de modèles abandonnés de modèles abandonnés. Cela explique également pourquoi l'utilisation de Dropout améliorera la performance de la généralisation, puisqu'elle peut être interprétée comme une moyenne sur un grand ensemble.

2.7 Rétropropagation

La rétropropagation (Back-propagation) est la méthode fondamentale utilisée pour la formation des modèles de réseaux neuronaux. En utilisant la rétropropagation, on peut facilement calculer le gradient des paramètres de la couche supérieure à la couche inférieure. La capacité à calculer efficacement le gradient permet d'utiliser la descente de gradient pour optimiser les paramètres dans l'ensemble du réseau. Dans le cas d'une perception multicouche, si la structure du réseau est un graphe acyclique dirigé, nous pouvons simplement utiliser la règle de la chaîne pour calculer efficacement le gradient des couches supérieures aux couches inférieures, comme le montre l'algorithme ci-dessous.

Supposons que nous disposions d'un ensemble d'apprentissage fixe $\left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$ de m exemples d'apprentissage. Nous pouvons entraîner notre réseau de neurones en utilisant la descente de gradient par lots. En détail, pour un seul exemple d'apprentissage (x, y), nous définissons la fonction de coût par rapport à cet exemple unique comme suit:

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2$$

donc la fonction de perte general:

$$\begin{aligned} J(W, b) &= \left[\frac{1}{m} \sum_{i=1}^{m} J\left(W, b; x^{(i)}, y^{(i)}\right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \\ &= \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}\left(x^{(i)}\right) - y^{(i)} \right\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \end{aligned}$$

BackProp

inputs : Un réseau de l couches, la fonction d'activation de chaque couche l et de sortie $h_n = \sigma(W_n^T h_{n-1} + b_n)$

output: Calculer le gradient g de la couche de sortie

$g \leftarrow \frac{\partial \epsilon(y,\hat{y}}{\partial \hat{y}}$

for $i \leftarrow l$ **do**

Calculer le gradient pour la couche actuelle

$\frac{\partial \varepsilon(y,\hat{y})}{\partial W_l} = \frac{\partial \varepsilon(y,\hat{y})}{\partial hl} \frac{\partial h_l}{\partial W_l} = g \frac{\partial h_l}{\partial W_l}$;

$\frac{\partial \varepsilon(y,\hat{y})}{\partial b_l} = \frac{\partial \varepsilon(y,y)}{\partial h_l} \frac{\partial h_l}{\partial b_l} = g \frac{\partial h_l}{\partial b_l}$;

Calculer le gradient

$\frac{\partial \varepsilon(y,\hat{y})}{\partial W_l}$ and $\frac{\partial \varepsilon(y,\hat{y})}{\partial b_l}$;

Propager le gradient dans la couche inferieure ;

$g \leftarrow \frac{\partial \varepsilon(y,\hat{y})}{\partial h_l} \frac{\partial h_l}{\partial h_{l-1}} = g \frac{\partial h_l}{\partial h_{l-1}}$

end

Algorithme 4: Back-propagation

2.8 Perceptron Multi-couches

La régression logistique et le softmax sont des algorithmes de classification linéaire qui échoueront lorsque les données ne sont pas linéairement séparables. Pour résoudre de tels problèmes, on peut utiliser des caractéristiques élaborées manuellement, des noyaux, le boosting ou des perceptions multicouches. En général, un modèle de perception multicouche empile plusieurs transformations non linéaires. Chaque couche se présente sous la forme de l'équation:

$$h_n = \sigma(W_n^T h_{n-1} + b_n)$$

où h_n est la sortie de la n^{me} couche, $\sigma(x)$ est une fonction d'activation non linéaire, et W_n, b_n sont les paramètres de cette couche. Pour h_0, h_{n-1} est la donnée d'entrée X. Au sommet d'un perceptron multicouche se trouve une couche de transformation logistique ou une couche de transformation softmax pour transformer la sortie dans la plage $[0, 1]$. Le gradient de la fonction de perte peut ensuite être propagé vers le bas du modèle pour mettre à jour les paramètres.

Si nous fixons toutes les transformations à la fonction sigmoïde $\sigma(x) = \frac{1}{1+e^{-x}}$, nous pouvons traiter le perceptron multicouche comme plusieurs couches de modèles de régression logistique.

2.9 Réseau de neurones à convolution

Les réseaux de neurones convolutifs (CNN)[62] sont des estimateurs de fonctions non linéaires, qui fonctionnent selon le principe de l'apprentissage d'une fonction f qui fait correspondre les entrées x à des sorties y définies. Alors qu'un perceptron multicouche entièrement connecté est un approximateur de fonction universel qui peut approximer toute fonction continue sur des sous-ensembles topologiques compacts de $\mathbb{R}^n$ l'utilisation d'une architecture peu profonde pose trois problèmes :

Premièrement, pour les images haute résolution du monde réel, l'utilisation d'une couche entièrement connectée fera exploser le nombre de paramètres dans le modèle. Par exemple, pour une image couleur avec 224x224 pixels et 3 canaux de couleur, si nous voulons la mapper 256 dimensions en utilisant un réseau entièrement connecté à une seule couche, la matrice de poids aura 38 millions de paramètres. Il faudrait trop de mémoire et de puissance de mémoire et de puissance de calcul pour effectuer une multiplication matricielle aussi importante.

Deuxièmement, un réseau entièrement connecté est enclin à l'ajustement excessif[61]. Bien qu'il soit capable d'approximer n'importe quelle fonction, il est également capable d'apprendre une fonction qui s'adapte exactement à n'importe quel bruit dans les données d'apprentissage. qui s'adapte exactement à n'importe quel bruit dans les données d'apprentissage, ce qui conduira à une mauvaise adaptation des données de test.

Troisièmement, un réseau entièrement connecté n'est qu'un approximateur de fonction générale qui n'exploite aucune connaissance du domaine. Il

serait plus efficace si nous concevoir un réseau qui intègre une certaine connaissance du domaine. Un réseau convolutif est capable de résoudre ces trois problèmes.

Yann LeCun [61] et d'autres chercheurs ont initialement proposé des réseaux convolutifs dans le cadre d'une structure de réseau célèbre, LeNet-5 [62] contenant 7 couches, qui a été utilisée pour résoudre avec succès le problème de la classification numérique manuscrite. Formellement, pour une image bidimensionnelle I et d'un noyau de convolution bidimensionnel $K^{m,n}$, la sortie S est :

$$\begin{aligned} s[i,j] = (I * K)[i,j] = \textstyle\sum_m \sum_n I[m,n]K[i-m, j-n] \\ = \textstyle\sum_m \sum_n I[i-m, j-n]K[m,n] \end{aligned}$$

En général dans le cas continue, la convolution * se fait comme suit :

$$s(t) = (f * g)(t) = \textstyle\int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau.$$

De plus, un paramètre stride peut être utilisé pour spécifier la distance de déplacement du noyau entre les mesures successives de l'image d'entrée. est déplacé entre les mesures successives de l'image d'entrée.

2.9.1 Couche de convolution

L'essence de l'idée des CNN est la couche convolutive. Les couches convolutives sont constituées d'un groupe de noyaux (kernels ou filtres) dont les poids (ou paramètres) peuvent être appris. L'opération de convolution est un produit scalaire sur des vecteurs, qui est présenté dans l'Équation ci-dessus. Lorsque la convolution est étendue aux images couleur, chaque canal couleur est convolué séparément par le noyau correspondant. ce qui signifie qu'une image couleur à trois canaux nécessite trois noyaux distincts ou un noyau tridimensionnel.

Un exemple d'une opération de convolution bidimensionnelle est illustré dans la figure 3. Les dimensions de la carte de caractéristiques (sortie de la convolution) dans l'exemple sont de 3x2 parce que la convolution est additionnée sur tous les canaux de taille 2x2, c'est-à-dire que le filtre 2x2 sélectionné sélectionné a 24 valeurs et le produit scalaire est pris par ces valeurs et la zone correspondante zone correspondante de taille 3x2 de l'image. Comme l'image est plus grande dans les dimensions x et y, le filtre est glissé sur l'image, de sorte que l'image entière est convoluée séquentiellement.

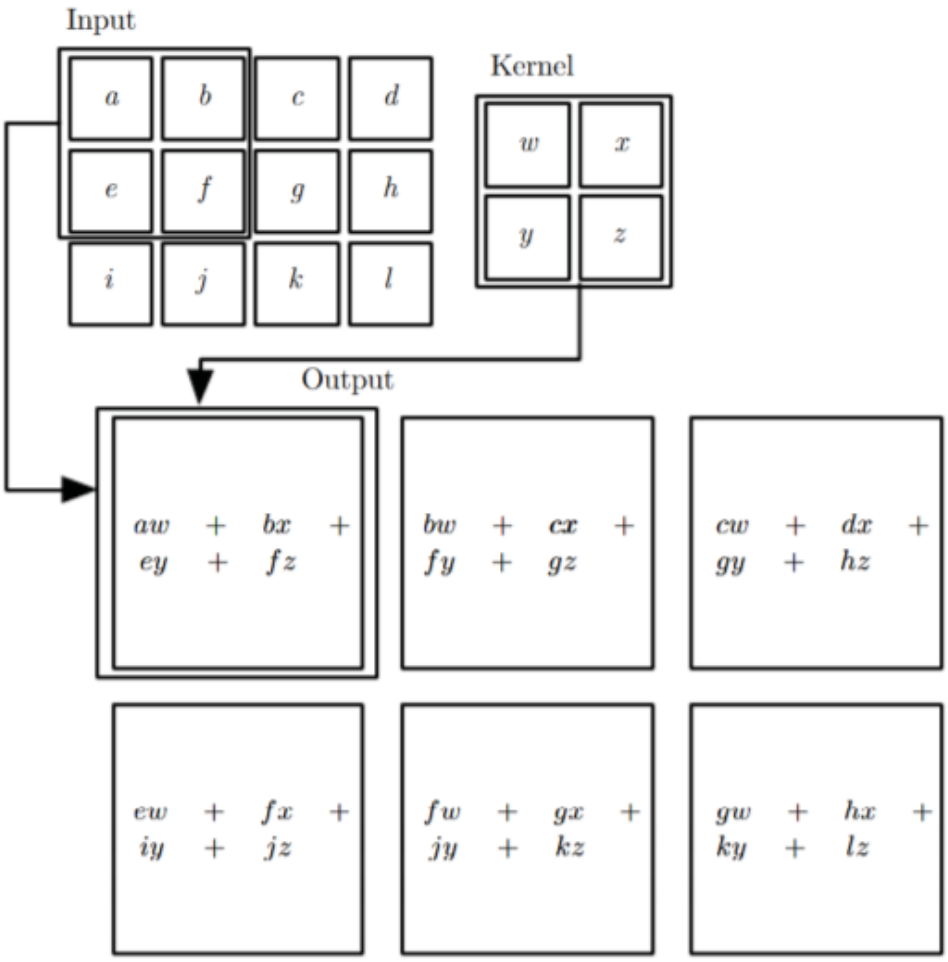

Figure 3 – Représentation de l'opération de la convolution [62]

La taille du noyau peut varier en fonction de l'implémentation, mais elle est généralement inférieure que la taille de l'entrée. Lorsque la taille du noyau est plus petite que l'entrée, on dit que la convolution a des interactions ou des poids épars. Cette propriété est importante pour les CNN car les images peuvent avoir des millions de pixels, mais même de petits morceaux de l'image peuvent être utilisés pour extraire des caractéristiques importantes de l'image, comme les bords. Grâce à cette propriété, les filtres peuvent extraire des caractéristiques significatives de l'image, avec moins de paramètres qu'en ayant à d'enregistrer les paramètres pour chaque pixel.

2.9.2 Couche d'activation

Le rôle principal de la couche d'activation est d'ajouter de la non-linéarité au réseau. La non-linéarité est nécessaire pour approximer des problèmes non linéaires. Il existe différentes fonctions d'activation non linéaires qui peuvent être utilisées comme la sigmoïde, la tanh, la ReLU et leurs variantes. variantes. ReLu est probablement la fonction d'activation la plus couramment utilisée dans les CNN. ReLU fait correspondre les valeurs de sortie de la convolution à des valeurs allant de zéro à plus élevé via la fonction de transfert comme indiqué dans l'équation.

$$\sigma(x) = max(0, x)$$

De cette façon, les valeurs négatives sont supprimées de la sortie de la convolution et les valeurs positives ont une correspondance linéaire. valeurs positives ont une correspondance linéaire, ce qui rend la ReLU non linéaire avec la transformation.

2.9.3 Couche Pooling

Les couches Pooling agissent comme des résumés statistiques de l'entrée en calculant une valeur unique à partir des valeurs voisines de l'entrée. Les avantages du Pooling sont de rendre l'entrée spatialement invariante aux petites translations, ce qui permet une meilleure généralisation des caractéristiques. et de réduire l'échantillonnage de l'entrée, ce qui rend les calculs suivants plus efficaces. Par exemple, dans la figure 4 on a le cas du max-pooling, c'est-a-dire la valeur maximale de la zone d'entrée sélectionnée est transmise à la couche suivante. Si la position de cette valeur maximale change un peu, mais dans la zone de l'opération de mise en commun, la même valeur maximale est sélectionnée.

Les paramètres de la couche de mise en commun sont l'opération de mise en commun, la taille de l'opération de mise en commun et le pas. l'opération de mise en commun et le pas. Deux opérations de mise en commun courantes sont le max-pooling, qui sélectionne la valeur maximale dans la zone de pooling , et le pooling moyenne, qui moyenne des valeurs de la zone de pooling et transmet cette valeur. Le max-pooling est illustré ci-dessous.

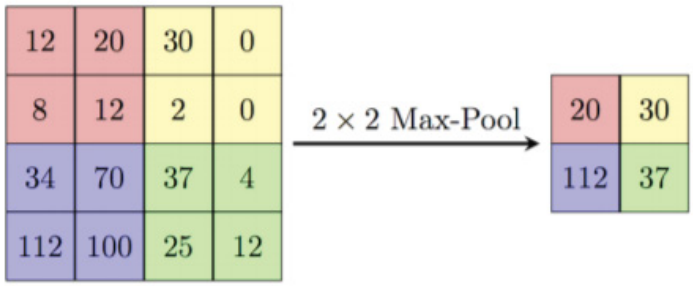

Figure 4 – Max-pooling avec une zone de pooling de 2 × 2 pixels et un stride de 2

2.9.4 Réseaux de convolution remarquables

- **AlexNet**[54] : est le nom d'un réseau de neurones convolutifs qui a eu un impact important sur le domaine de l'apprentissage automatique, en particulier dans l'application de l'apprentissage profond à la vision artificielle. Il a remporté le concours ImageNet 2012 par une large marge . Le réseau avait une architecture très similaire à celle de LeNet, mais était plus profond, avec plus de filtres par couche, et avec des couches convolutives empilées. Il était composé de 11×11, 5×5, 3×3, convolutions, max pooling, dropout, augmentation des données, activations ReLU, SGD avec momentum. Il a attaché des activations ReLU après chaque couche convolutionnelle .

- **ResNet**[41] : Cette architecture a remporté le concours ImageNet en 2015. L'écart de performance montre une véritable rupture architecturale, par rapport aux autres réseaux. Des expériences ont montré que les CNN de type AlexNet augmentent de performance proportionnellement au nombre de couches. Cependant, on a constaté qu'au-delà d'une certaine profondeur, les performances se dégradent. L'explication la plus convaincante est que lorsque le réseau devient trop profond, le gradient ne se propage efficacement vers les couches inférieures, ce qui diminue les performances. Pour surmonter ceci, les auteurs de Resnet ont inventé ce qu'ils ont appelé une connexion par saut. Les auteurs ont trouvé aussi qu'il est possible d'optimiser la performance en considérant la sortie comme résidu.

- **VGG16**[95]: est un modèle de réseau neuronal convolutif qui a atteint une précision de 92,7 % dans le test top-5 d'ImageNet. C'est l'un des célèbres modèles soumis à l'ILSVRC-2014. Il améliore AlexNet en remplaçant les grands filtres à noyau (11 et 5 dans la première et la deuxième couche convolutive, respectivement) par de multiples

filtres à noyau 3×3, l'un après l'autre. L'entrée de la couche cov1 est une image RVB de taille fixe 224 x 224. L'image passe par une pile de couches convolutionnelles, où les filtres ont été utilisés avec un champ réceptif très petit : 3×3 (qui est la plus petite taille pour capturer la notion de gauche/droite, haut/bas, centre). Dans l'une des configurations, il utilise également des filtres de convolution 1×1, qui peuvent être vus comme une transformation linéaire des canaux d'entrée (suivie d'une non-linéarité). Le pas de convolution est fixé à 1 pixel ; le remplissage spatial de la couche de convolution d'entrée est tel que la résolution spatiale est préservée après convolution, c'est-à-dire que le remplissage est de 1 pixel pour les couches de convolution 3×3. La mise en commun spatiale est effectuée par cinq couches de max-pooling, qui suivent certaines des couches de convolution (toutes les couches de convolution ne sont pas suivies de max-pooling). Le max-pooling est effectué sur une fenêtre de 2×2 pixels, avec un pas de 2.

- **Xception**[17] : est une architecture CNN dont la nouveauté est l'adaptation de couches de convolution séparables en profondeur modifiées. Xception a été développé chez Google, et il est basé sur les architectures Inception [33] publiées précédemment, également développées par Google. Dans l'architecture Xception, les modules d'Inception ont été modifiés en convolutions séparables en profondeur modifiées, d'où le nom de "Extreme Inception". Xception a atteint une précision Top-1 de 0,790 sur ImageNet en 2016. En 2020, les réseaux avec un nombre de paramètres similaire atteignent une précision de 0,826 précision Top-1 [43]. Xception a été choisi pour être l'architecture CNN de l'étude, en raison de ses bonnes performances sur le terrain. étude, en raison de ses bonnes performances sur le jeu de données ImageNet, et de la disponibilité des modèles

pré-entraînés avec l'ImageNet. De plus, les implémentations facilement disponibles de l'architecture ont permis des exécutions simples à travers les différentes couches de l'architecture. L'architecture Xception, où les couches de convolution séparable sont les couches de convolution séparable en profondeur modifiées. L'architecture Xception comprend également les couches résiduelles et la normalisation par lots. L'architecture Xception est divisée en 14 blocs différents qui sont séparés par les couches résiduelles. par les couches résiduelles. Les couches de l'implémentation suivent également la séparation en 14 blocs dans leur dénomination. dans leur dénomination, en regroupant les couches par le bloc correspondant auquel elles appartiennent.

3 État de l'art

3.1 EfficientNet

Pour adapter un réseau neuronal à une tâche plus robuste, la façon la plus naturelle de procéder est d'ajouter des couches. Cependant, nous avons également le contrôle sur la largeur des couches et la résolution de notre entrée. En pratique, cependant, il n'est pas évident de mettre à l'échelle ces trois composants ; l'article d'EfficientNet[102] présente une méthode de principe Compound Scaling du modèle qui permet d'obtenir un meilleur compromis précision-FLOP.

Un réseau neuronal convolutif peut être considéré comme un empilement ou une composition de différentes couches convolutives. De plus, ces couches peuvent être divisées en différentes étapes.Comme a été précédemment décrit ResNet a cinq étapes, et toutes les couches de chaque étape ont le même type de convolution. Par conséquent, un CNN peut être représenté mathématiquement comme suit :

$$\mathcal{N} = \bigodot_{i=1\ldots s} \mathcal{F}_i^{L_i} \left(X_{\langle H_i, W_i, C_i \rangle} \right)$$

avec $\bigodot_{i=1\ldots s}$ désigne la composition ou l'empilement des couches, $\mathcal{F}_i^{L_i}$ représente la couche F_i répétée L_i fois à l'étape i, (H_i, W_i, C_i) représente la dimension de l'entrée à la couche X.

L'augmentation de la profondeur, en empilant davantage de couches convolutionnelles, permet au réseau d'apprendre des caractéristiques plus complexes. Cependant, les réseaux plus profonds ont tendance à souffrir de gradients évanescents(Vanishing gradient) et deviennent difficiles à former. Bien que de nouvelles techniques telles que la normalisation des lots et les connexions sautées(ResNet) soient efficaces pour résoudre ce problème, les études empiriques suggèrent que les gains de précision réels obtenus en augmentant uniquement la profondeur du réseau se saturent rapidement. Par exemple, ResNet-1000 fournit la même précision que ResNet-100 malgré toutes les couches supplémentaires.

L'augmentation de la largeur des réseaux permet aux couches d'apprendre des caractéristiques plus fines. Ce concept a été largement utilisé dans de nombreux travaux tels que Wide ResNet et Mobile Net. Cependant, comme dans le cas de l'augmentation de la profondeur, l'augmentation de la largeur empêche le réseau d'apprendre des caractéristiques complexes, ce qui entraîne une diminution des gains de précision.

Une résolution d'entrée plus élevée fournit un plus grand nombre de détails sur l'image et améliore donc la capacité du modèle à raisonner sur des objets plus petits et à extraire des motifs plus fins. Mais comme les autres dimensions de mise à l'échelle, elle ne permet qu'un gain de précision limité. La figure 5 donne (a) un exemple de réseau de base ; (b)-(d) sont des mises à l'échelle conventionnelles qui n'augmentent qu'une seule dimension du réseau : la largeur, la profondeur ou la résolution. largeur, la profondeur ou la résolution. (e) est la méthode Compound composée que l'article

propose et qui met uniformément à l'échelle les trois dimensions avec un rapport fixe.

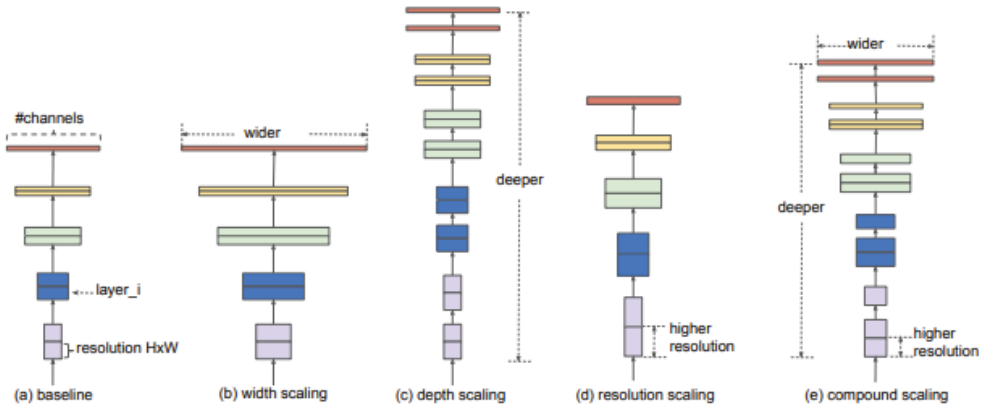

Figure 5 – Différentes méthodes de scaling en comparaison avec Compound Scaling [102]

Comme on peut le déduire ci-dessus, L_i contrôle la profondeur du réseau, C_i est responsable de la largeur du réseau tandis que H_i et W_i affectent la résolution d'entrée. Trouver un ensemble de bons coefficients pour mettre à l'échelle ces dimensions pour chaque couche est impossible, car l'espace de recherche est énorme. Ainsi, afin de restreindre l'espace de recherche, les auteurs établissent un ensemble de règles de base.

Toutes les couches étapes du modèle mis utiliseront les mêmes opérations de convolution que le réseau de base.

Toutes les couches doivent être mises à l'échelle de manière uniforme avec un ratio constant.

Donc ceci vient à optimiser le problème avec des contraintes à respecter :

$$\max_{d,w,r} \text{Accuracy}(\mathcal{N}(d,w,r))$$

i.e.

$$\mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\left(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\right)$$

Dans cet article, les auteurs proposent le Compound Scaling, qui utilise un coefficient composé ϕ pour mettre à l'échelle de manière uniforme la largeur, la profondeur et la résolution du réseau de manière raisonnée : Profondeur : $d = \alpha^\phi$ Largeur : $w = \beta^\phi$ Résolution : $r = \gamma^\phi$ s.t. $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

où α, β, γ sont des constantes qui peuvent être déterminées par une recherche sur une petite grille. Intuitivement, ϕ est un coefficient spécifié par l'utilisateur qui contrôle combien de ressources supplémentaires sont disponibles pour la mise à l'échelle du modèle, tandis que α, β, γ spécifient comment affecter ces ressources supplémentaires à la largeur, la profondeur et la résolution du réseau respectivement.

3.2 ViT

Les modèles Transformers ont bel et bien révolutionné le traitement du langage naturel. Lorsqu'ils ont été introduits pour la première fois, ils ont battu de nombreux records en matière de traitement du langage naturel et ont repoussé l'état de l'art de l'époque. Aujourd'hui, ils sont devenus un standard de facto pour les tâches modernes du NLP et ils apportent des gains de performance spectaculaires par rapport à la génération précédente de modèles comme les LSTM et les GRU.

L'article de loin le plus important qui a transformé le paysage de la NLP est l'article "Attention is all you need"[108]. L'architecture du transformateur a été introduite dans cet article.

Ainsi, l'inspiration principale derrière le transformateur était de se débarrasser de cette récurrence tout en capturant presque toutes les dépendances,

pour être précis les dépendances globales, car la fenêtre de référence des transformateurs est une gamme complète. Ceci a été réalisé en utilisant une variante du mécanisme d'attention appelé auto-attention (à têtes multiples) qui est très important pour leur succès. Un autre avantage des modèles Tranformer est qu'ils sont hautement parallélisables.

Une fonction d'attention peut être décrite comme la mise en correspondance d'une requête et d'un ensemble de paires clé-valeur avec une sortie, où la requête, les clés, les valeurs et la sortie sont toutes des vecteurs. La sortie est calculée comme une somme pondérée des valeurs, où le poids attribué à chaque valeur est calculé par une fonction de compatibilité de la requête avec la clé correspondante. Le nom attention provient de la fonction Scaled Dot-Product Attention. L'entrée consiste en des requêtes et des clés de dimension d_k et de valeurs de dimension d_v . Nous calculons les produits scalaires de la requête avec toutes les clés, divisons chacun par $\sqrt{d_k}$ et appliquons une fonction softmax pour obtenir les poids sur les valeurs.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

avec Q, K et V sont respectivement les matrices requêtes, clés et valeurs.

Les principales contributions de cet article de Google [27] ne concernent pas une nouvelle architecture, mais plutôt l'application de cette architecture au domaine de la vision par ordinateur. C'est la méthode d'entraînement et le jeu de données utilisé pour pré-entraîner le réseau qui ont permis à ViT d'obtenir d'excellents résultats par rapport a l'état d'art sur ImageNet.

Pour traiter les images 2D, nous remodelons l'image $\mathrm{x} \in \mathbb{R}^{H \times W \times C}$ en une séquence de patchs aplatis de 2D $\mathbf{x}_p \in \mathbb{R}^{N \times \left(P^2 \cdot C\right)}$, où (H, W) est la résolution de l'image originale, C est le nombre de canaux, (P, P) est la résolution de chaque patch d'image, et $N = HW/P^2$ est le nombre de patchs résul-

tant, qui sert également de longueur de séquence d'entrée effective pour le Transformateur. Le transformateur utilise une taille constante de vecteur latent D à travers toutes ses couches, nous aplatissons donc les patchs et les cartographions en dimensions D avec une projection linéaire entraînable comme précise ci dessous Nous désignons la sortie de cette projection comme étant l'intégration des patchs.

$$\mathbf{z}_0 = \left[\mathbf{x}_{\text{class}} ; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots ; \mathbf{x}_p^N\mathbf{E}\right] + \mathbf{E}_{\text{pos}} , \mathbf{E} \in \mathbb{R}^{\left(P^2 \cdot C\right) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$$

Ensuite nous ajoutons un encastrement apprenable à la séquence de patchs encastrés ($\mathbf{z}_0^0 = \mathbf{x}_{\text{class}}$), dont l'état à la sortie de l'encodeur Transformer ($\mathbf{z}_L^0$) sert de représentation d'image y.

$$\mathbf{y} = \text{LN}\left(\mathbf{z}_L^0\right)$$

Pendant le pré-entraînement et le réglage fin, une tête de classification est attachée à $\mathbf{z}_L^0$. La tête de classification est implémentée par un MLP avec une couche cachée lors du pré-entraînement et par une seule couche linéaire lors du réglage fin.

Les incorporations de position sont ajoutées aux incorporations de patchs pour conserver l'information de position. Nous utilisons des incorporations de position standard apprenables 1D, car nous n'avons pas observé de gains de performance significatifs en utilisant des incorporations de position plus avancées tenant compte de la 2D . La séquence résultante de vecteurs d'incorporation sert d'entrée à l'encodeur.

L'encodeur Transformer[108] est constitué de couches alternées de blocs d'auto-attention à têtes multiples et de blocs MLP.

$$\mathbf{z}'_\ell = \text{MSA}\left(\text{LN}\left(\mathbf{z}_{\ell-1}\right)\right) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \ldots L$$

$$\mathbf{z}_\ell = \text{MLP}\left(\text{LN}\left(\mathbf{z}'\ell\right)\right) + \mathbf{z}'\ell, \qquad \ell = 1 \ldots L$$

Le Layernorm (LN) est appliqué avant chaque bloc, et les connexions résiduelles après chaque bloc.

N.B.: Le Perceptron contient deux couches avec des GELU d'activation.

$\text{GELU}(x) := x\mathbb{P}(X \leq x) = x\Phi(x) = 0.5x\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$

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

Cependant, récemment, des chercheurs de Google Brain ont affirmé que Gated Multi-Layer Perceptron (gMLP), un modèle d'apprentissage profond qui ne contient que des perceptrons multicouches de base avec moins de paramètres, surpasse la performance des modèles Transformer dans les tâches de traitement du langage naturel (NLP). gMLP aussi atteint presque la même précision que Transformer dans les tâches de vision par ordinateur (CV)[67].

3.3 Autoencodeur

Un autoencodeur est un type de réseau de neurones artificiel utilisé pour apprendre les représentations ou les distributions des données entrantes d'une manière non supervisée. Typiquement utilisé pour la réduction de dimensionnalité, en apprenant au réseau à ignorer le bruit de l'entrée (débruitage). Parallèlement au côté réduction, un côté reconstruction est appris, où l'autoencodeur tente de générer à partir du codage réduit une représentation aussi proche que possible de son entrée d'origine.

La forme la plus simple d'un autoencodeur est un réseau de neurones non récurrent à propagation avant, similaire aux perceptrons monocouche, qui participent à des perceptrons multicouches (MLP) dont le but de re-

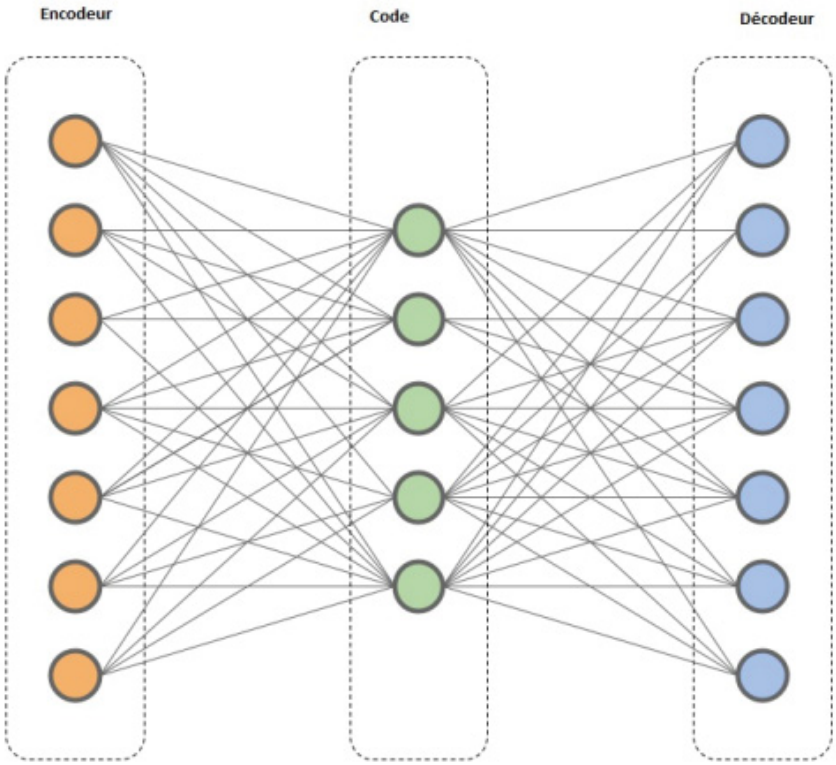

Figure 6 – Schéma explicatif des différentes composantes des AE [25]

construire ses entrées (minimiser la différence entre l'entrée et la sortie $|x - \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})\ |$). Par conséquent, les autoencodeurs sont des modèles d'apprentissage non supervisés (ne nécessitent pas d'entrées étiquetées pour permettre l'apprentissage).

Un autoencodeur se compose de deux parties comme l'indique la figure 6, l'encodeur qui représente la partie d'entrée à gauche de la figure et le décodeur comme sortie du réseau, qui peuvent être définis comme des transitions ϕ et ψ [78] , telles que:

On note $\mathcal{X}$ l'espace des entrées, $\mathcal{X}'$ l'espace des sorties et $\mathcal{L}$ l'espace latent. l'encodeur $\phi : \mathcal{X} \to \mathcal{L}$

le décodeur $\theta : \mathcal{L} \to \mathcal{X}'$

$$\phi, \theta = \arg\min_{\phi,\psi} \|X - (\theta \circ \phi)X\|^2$$

Les autoencodeurs sont formés pour minimiser les erreurs de reconstruction (ou la fonction de perte $\mathbf{L}$) :

$$\mathbf{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$
$$\mathcal{J}_{AE}(\theta, \psi) = \sum_{i=1}^{n} \mathbf{L}\left(x_i, g_\psi\left(f_\theta\left(x_i\right)\right)\right) = \sum_{i=1}^{n} \mathbf{L}\left(x_i, g_\psi\left(z_i\right)\right)$$
$$= \sum_{i=1}^{n} \mathbf{L}\left(x_i, \widehat{x_i}\right)$$

avec σ est une fonction d'activation élément par élément (comme la fonction logistique Sigmoid, ou la RELU, unité linéaire rectifiée). $\mathbf{W}, \mathbf{W}'$ des matrices de poids et $\mathbf{b}, \mathbf{b}'$ des vecteurs de biais et $\mathbf{x}, \mathbf{x}'$ les données de l'entrée et de sortie successivement. Les poids et les biais sont généralement initialisés au hasard, puis mis à jour de manière itérative pendant la formation via la rétropropagation. Ensuite, l'étape de décodeur de l'autoencodeur met en correspondance $\mathbf{h}$ à la reconstruction $\mathbf{x}'$ de la même forme que $\mathbf{x}$.

3.3.1 Autoencodeur variationnel

La différence principale entre les autoencodeurs variationnels et les autres types d' autoencodeurs consiste à utiliser une approche variationnelle pour la représentation latente dans l'apprentissage, en d'autres termes, on suppose que les données d'entrée ont une sorte de distribution de probabilité $p(\mathbf{x}|\mathbf{z})$ et comme l'indique la figure 7, on tente ensuite de trouver les paramètres (tels que μ, σ) de la distribution par l'approximation de l'encodeur $q_\phi(\mathbf{z}|\mathbf{x})$ (inférée au niveau du codeur) à la distribution a posteriori $p_\theta(\mathbf{x}|\mathbf{z})$ (générée au niveau du décodeur), ce qui engendre que le réseau

optimise la fonction suivante :
$\mathcal{L}(\phi, \theta, \mathbf{x}) = D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big(\log p_\theta(\mathbf{x}|\mathbf{z})\big)$ avec D_{KL} la divergence Kullback–Leibler ($D_{\mathrm{KL}}(P \parallel Q) := \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$).
La figure 7 illustre les composantes de l'autoencodeur ainsi que la distribution de probabilité inférée dans chacune de ces composantes.

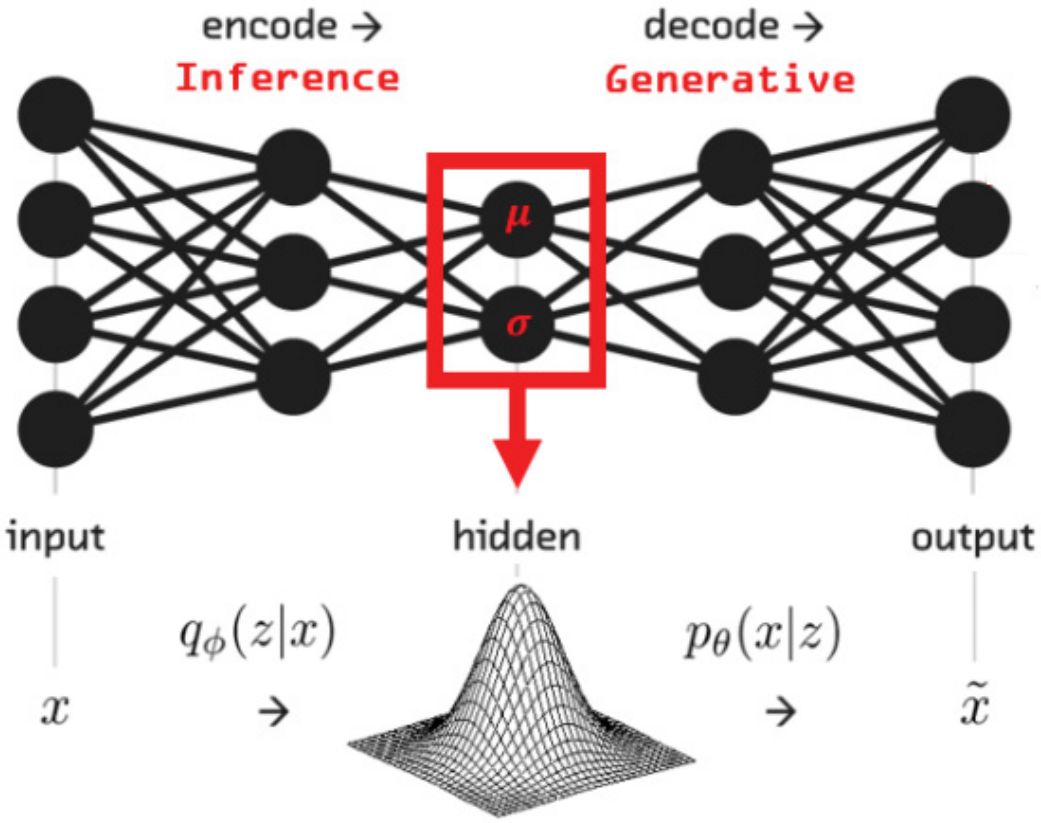

Figure 7 – schéma explicatif des autoencodeurs variationnels [78]

3.3.2 Autoencodeur contractif

L'autoencodeur contractif [78] est un type de réseau dans lequel on pénalise la norme de Frobenius $\|A\|_F = \mathrm{tr}\,(A^*A)$ avec $A \in \mathcal{M}_{m,n}$, A^* sa matrice adjointe de la matrice Jacobienne des activations de l'encodeur par rapport à l'entrée en ajoutant un terme supplémentaire dans la fonction objectif afin

d'apprendre des représentations utiles :

$$\mathcal{J}_{AE}(\theta, \psi) = \sum_{i=1}^{n} \mathcal{L}\left(x_i, \widehat{x}_i\right) + \lambda \sum_{i=1}^{m} \left\| \frac{\partial \hat{x}_i}{\partial x} \right\|_F^2,$$

3.3.3 Autoencodeur épars

Les autoencodeurs épars utilisent une pénalisation pour minimiser le nombre de neurones dans la couche latente [78]. Plus précisément, il s'agit de construire une fonction objectif en pénalisant les activations dans une couche compte tenu de l'observation en question. Donc le réseau de neurones effectue l'encodage et le décodage en se fondant sur l'activation d'un certain nombre de neurones. Ceci consiste à ajouter un terme supplémentaire dans la fonction objectif pendant l'entraînement afin de pénaliser la norme L_1 de Lebesgue

$$\mathcal{J}_{AE}(\theta, \psi) = \sum_{i=1}^{n} \mathcal{L}\left(x_i, \widehat{x}_i\right) + \lambda \parallel \hat{x} \parallel_1$$

où $\parallel \hat{x} \parallel_1 := \sum_{i=1}^{n} |\hat{x}_i|$, la norme L_1.

Dans la section suivante, nous utiliserons une familles des autoencodeurs pour le clustering profond ou deep clustering.

4 Deep Clustering

Le but de cette partie est d'améliorer les résultats de l'intégration des méthodes de classification dans la couche latente des autoencodeurs[25].
Fondamentalement, il y a deux façons d'utiliser les caractéristiques profondes pour le clustering. La première consiste au regroupement des caractéristiques cachées qui sont extraites d'un réseau profond bien entraîné [3]. Cependant, ces approches ne peuvent pas exploiter pleinement la puissance du réseau neuronal profond pour le clustering. L'autre consiste à intégrer une méthode de clustering existante dans les modèles DL,qui est une approche de bout en bout. Par exemple, intègrer l'algorithme K-means dans les autoencodeurs profonds et effectuer l'affectation des clusters dans les couches intermédiaires [37].

4.1 Principes

Aljalbout et al.[3] ont introduit une approche en boucle fermée dans laquelle l'entraînement de l'autoencodeur et le Kmeans sont effectués dans une boucle itérative. Leur réseau profond apprend d'abord les représentations des entrées pour obtenir leurs caractéristiques, ensuite ces représentations apprises sont transmises comme entrée de Kmeans. Leur objectif est d'utiliser les représentations des données pour améliorer le regroupement, et d'exploiter les résultats du regroupement pour fournir des indices sur les étiquettes lors de l'apprentissage des représentations.
Cependant, cette méthode n'est pas adaptée aux grands ensembles de données vu le coût de calcul et de mémoire qui nécessitent l'exécution itérative

d'un algorithme de clustering pendant l'apprentissage et la préformation du réseau DEC[37].

Dans [105], les auteurs proposent un cadre général pour intégrer les méthodes traditionnelles de clustering dans les modèles d'apprentissage profond et développent un algorithme pour optimiser les objectifs non convexes et non linéaires sous-jacents en utilisons la méthode ADMM (Alternating Direction of Multiplier Method). Un algorithme simple, mais puissant qui convient bien à l'optimisation convexe distribuée, et en particulier aux problèmes rencontrés en statistique appliquée et en apprentissage automatique. Il prend la forme d'une procédure de décomposition-coordination, dans laquelle les solutions de petits sous-problèmes locaux sont coordonnées pour trouver une solution à un grand problème problème global. L'ADMM peut être considérée comme une tentative de combiner les avantages de la décomposition duale et des méthodes lagrangiennes augmentées pour l'optimisation sous contraintes, deux approches antérieures [105].

Définition 4.1. Une fonction $f : \mathbb{R}^n \Rightarrow \mathbb{R}$ est dite convexe si son image est un ensemble convex et $x, y \in$ l'image de f, $\lambda \in [0, 1]$, on a

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

Les auteurs K.Tian et al. ont considéré dans l'article [105] le problème général d'optimisation suivant:

$$\min_{x \in \mathbb{R}^n} h(x) + o(Dx)$$

où D une matrice à m lignes et n colonnes, h et o deux fonctions convexes respectivement de $\mathbb{R}^n$ et $\mathbb{R}^m$. Considérons une variable $z = Dx \in \mathbb{R}^m$, on

a :

$$\min h(x) + o(z)$$

ADMM utilise le lagrangien augmenté pour faire la décomposition suivante en y intégrant un paramètre $\rho > 0$:

$$\begin{cases} x^{k+1} \in \arg\min_{x \in \mathbb{R}^n} \left\{ h(x) + o\left(z^k\right) + \left\langle \lambda^k, Dx - z^k \right\rangle + \frac{e}{2} \left\| Dx - z^k \right\|^2 \right\} \\ z^{k+1} \in \arg\min_{z \in \mathbb{R}^m} \left\{ h\left(x^{k+1}\right) + o(z) + \left\langle \lambda^k, Dx^{k+1} - z \right\rangle + \frac{e}{2} \left\| Dx^{k+1} - z \right\|^2 \right\} \end{cases}$$

Finalement, ils ont obtenu l'équation suivante, indépendante des fonctions h et o :

$$\lambda^{k+1} = \lambda^k + \rho \left(Dx^{k+1} - z^{k+1} \right)$$

La classification profonde se fait au niveau de la couche latente et peut être formulée comme suit :

$$\min : \|X - \hat{X}\|_F^2 + \lambda \times \mathcal{G}_w(Y)$$

avec X, les données brutes à classer et $\hat{X}$ la reconstruction de X; $\mathcal{G}_w(Y)$, une fonction de classification spécifique dépendant de la méthode classique à intégrer ; λ, une constante permettant une transaction entre le réseau de neurones et la classification. Son lagrangien augmenté est donnée par :

$$\mathcal{L}_\rho(\theta, Y, U, w) = \|X - \hat{X}\|_F^2 + \lambda \times \mathcal{G}_w(Y) + \frac{\rho}{2} \|Y - \phi_{\theta 1}(X) + U\|_F^2$$

avec U la réciproque de λ dans la solution donnée par la méthode ADMM $\theta = \{\theta_1, \theta_2\}$ l'ensemble des paramètres de l'autoencodeur profond, avec θ_1 les paramètres de l'encodeur et θ_2 ceux du décodeur.
Guo et al. [37] ont amélioré la perfermonce du Deep Embeded Clustering en ajoutant des propriétés qui préservent les structures locales des points.

DeepCluster $\{\mathbf{w} \in \mathcal{W} : \mathbf{w} = \text{argmin}_{\mathbf{w}}\, \mathcal{G}_{\mathbf{w}}(\mathbf{Y})\}$
inputs : Donnees d'entrees X, taux d'apprentissage η, hyperparametres ρ, λ
output: Regroupement des donnees
Initialiser les paramètres de la DAE θ, les paramètres du modèle de clustering w ;
while *non convergé* **do**
mettre a jour $\theta : \theta = \theta - \eta d\theta$
mettre a jour Y
$\mathbf{Y} = \text{argmin}_{\mathbf{Y}}\, \lambda * \mathcal{G}_{\mathbf{w}}(\mathbf{Y}) + \frac{\rho}{2} \left\| \mathbf{Y} - f_{\boldsymbol{\theta}_1}(\mathbf{X}) + \mathbf{U} \right\|_F^2$
mettre a jour $\mathbf{U} : \mathbf{U} = \mathbf{U} + \mathbf{Y} - f_{\boldsymbol{\theta}_1}(\mathbf{X})$
mettre a jour $\mathbf{w} : \mathbf{w} = \text{argmin}_{\mathbf{w}}\, \mathcal{G}_{\mathbf{w}}(\mathbf{Y})$
end
return θ, w

Algorithme 5: l'algorithme DeepCluster

Pour assurer l'efficacité de la perte de clustering, l'autoencodeur de débruitage empilé utilisé dans la pré-formation n'est plus approprié. En effet, le clustering doit être effectué sur des caractéristiques de données propres, au lieu des données bruitées utilisées dans l'autoencodeur de débruitage. Nous supprimons donc directement le bruit. Le codeur automatique de débruitage empilé devient alors en un codeur automatique sous-complet, c'est-à-dire qui n'a pas de terme de régularisation explicite. La perte de reconstruction est mesurée par l'erreur quadratique moyenne (EQM) :

$$L_r = \sum_{i=1}^{n} \left\| x_i - g_{W'}\left(z_i\right) \right\|_2^2$$

où $z_i = f_W\left(x_i\right)$ et f_W et $g_{W'}$ sont des mappages de codeur et de décodeur respectivement. Les autoencodeurs peuvent préserver la structure locale de la distribution génératrice de données. Sous cette condition, manipuler légèrement l'espace incorporé en utilisant la perte de clustering ne causera pas d'erreur. Il est donc préférable que le coefficient de distorsion γ soit inférieur à 1.

ImpDeepCluster

inputs : Donnees d'entrees X, nombre de clusters K,Interval de mis a jour T, seuil d'arret δ, maximum d'iterations $MaxIter$

output: les poids de l'autoencodeur W, W', les centres des clusters μ, et les etiquettes s

Initialiser les paramètres μ, W, W'

for $iter \in \{0, 1, ..., MaxIter\}$ **do**

  if $iter\%T == 0$ **then**

    Calculer les points incorporés $\{z_i = f_W(x_i)\}_i^n = 1$

    $s_{old} \leftarrow s$

    $s_i = argmax(q_{ij})$

  end

  if $sum(s_{old} \neq s)/n \leq \delta$ **then**

    $Arret$

  end

end

return μ, W, W'

Algorithme 6: l'algorithme Improved DeepCluster

Guo et al. [38] ont également proposé une méthode de clustering avec Deep Clustering AutoEncoder (DCAE) . Pour se faire, tout d'abord, ils ont préparé un réseau DCAE avec une couche de regroupement utilisant la divergence KL pour appliquer une distribution cible, puis ils ont formé un modèle DCAE pour apprendre la représentation des caractéristiques et ont ensuite formé un modèle DCAE pour apprendre la représentation des caractéristiques des images. Grâce à un processus itératif, ils ont introduit les caractéristiques apprises dans un algorithme Kmeans pour regrouper les points de données dans leurs centroïdes identiques. L'apprentissage de l'espace des caractéristiques et le processus de regroupement sont deux procédures distinctes, et leur procédure de regroupement ne contribue pas à la fonction de perte globale. De plus, la divergence KL n'est peut-être pas la méthode la plus efficace pour le clustering, car elle n'est pas symétrique et, par conséquent, la distance entre le point de données A et le point de données B n'est pas la même que celle entre B et A .

Dans cette étude, la fonction objectif (perte) des méthodes de cluster-

ing par apprentissage profond est principalement composée de la perte du réseau profond et de la perte du clustering. Tout d'abord, L'autoencodeur est entraîné avec la perte de reconstruction standard à erreur quadratique moyenne. Par la suite, les auteurs utilisent l'algorithme de perte de k-Means pour obtenir des données uniformément réparties autour des centres de clusters. finalement, les auteurs utilisent la distribution t de Student comme noyau pour mesurer la similarité entre les points et les centroïdes.

4.2 Architecture DCEC

L'architecture DCEC se compose principalement en deux parties à savoir l'autoencodeur et la couche latente du clustering, d'où la fonction de perte qui se compose aussi en deux : celle d'apprentissage et celle du clustering.

En outre, la structure du DCEC est composée d'un CAE et d'une couche de regroupement connectée à la couche intégrée du CAE. La couche de regroupement fait correspondre chaque point incorporé z_i de l'image d'entrée x_i à une étiquette souple. Ensuite, la perte de regroupement L_c est définie comme la divergence de Kullback-Leibler (divergence KL) entre la distribution des étiquettes souples et la distribution cible prédéfinie. Le CAE est utilisé pour apprendre les caractéristiques intégrées et la perte de regroupement exploite les caractéristiques intégrées pour qu'elles soient enclines à former des clusters. L'objectif de DCEC est le suivant

$$L = L_r + \gamma L_c$$

où L_r et L_c sont respectivement la perte de reconstruction et la perte de regroupement, et $\gamma > 0$ est un coefficient qui contrôle le degré de distorsion de l'espace incorporé. Nous passons brièvement en revue leurs définitions

pour que la structure de DEC soit complète.

La couche de clustering maintient les centres de regroupement $\{\mu_j\}_1^K$ comme poids entraînables et met en correspondance chaque point intégré z_i en étiquette souple q_i par la distribution t de Student :

$$q_{ij} = \frac{\left(1 + \|z_i - \mu_j\|^2\right)^{-1}}{\sum_j \left(1 + \|z_i - \mu_j\|^2\right)^{-1}}$$

où q_{ij} est la j ème entrée de q_i, représentant la probabilité d'appartenance de z_i au cluster j. La perte de clustering est définie comme suit

$$L_c = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

où P est la distribution cible, définie comme suit

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_j \left(q_{ij}^2 / \sum_i q_{ij}\right)}$$

4.3 Optimisation

Nous optimisons $L = L_r + \gamma L_c$ en utilisant le décentrage par gradient stochastique (SGD) et la rétropropagation en mini-batch. Plus précisément, il y a trois types de paramètres à optimiser ou à mettre à jour : les poids de l'autoencodeur, les centres des clusters et la distribution cible P. Si on fixe la distribution cible P, les gradients de L_c par rapport au point intégré z_i et au centre de grappe μ_j peuvent être calculés comme suit [37]:

$$\frac{\partial L_c}{\partial z_i} = 2 \sum_{j=1}^{K} (1 + \|z_i - \mu_j\|^2)^{-1} (p_{ij} - q_{ij})(z_i - \mu_j)$$

$$\frac{\partial L_c}{\partial \mu_j} = 2\sum_{i=1}^{n}(1 + \|z_i - \mu_j\|^2)^{-1}(q_{ij} - p_{ij})(z_i - \mu_j)$$

Ensuite, étant donné un mini-lot avec m échantillons et un taux d'apprentissage λ, μ_j est mis à jour par[37]

$$\mu_j = \mu_j - \frac{\lambda}{m}\sum_{i=1}^{m}\frac{\partial L_c}{\partial \mu_j}$$

Les poids du décodeur sont mis à jour par [37]

$$W' = W' - \frac{\lambda}{m}\sum_{i=1}^{m}\frac{\partial L_r}{\partial W'}$$

4.4 Mesures d'évaluation

Pour justifier nos méthodes, deux approches d'évaluation sont utilisées pour calculer la qualité du cluster : La précision (ACC) et l'information mutuelle normalisée (NMI), qui distinguent les résultats de clustering générés par notre méthode DeepCluster et les étiquettes réelles.

4.4.1 Précision (ACC)

La précision du clustering est une mesure largement utilisée pour évaluer les résultats du clustering. Elle est calculée à l'aide des résultats de clustering obtenus et des étiquettes réelles en utilisant la forme suivante [89] :

$$\text{Accuracy } = \frac{\sum_{i=1}^{n}\delta\left(y_i, \text{map}\left(c_i\right)\right)}{n}$$

où N est le nombre d'échantillons, y_i désigne les vraies étiquettes, c_i les clusters obtenus, $\delta(y, c)$ est une fonction qui vaut 1 si $y - c$ et 0 sinon, et

map (c_i) est la fonction de permutation qui fait correspondre les étiquettes de cluster obtenues aux étiquettes de vérité fondamentale correspondantes.

4.4.2 Information mutuelle normalisée (NMI)

La NMI est une autre métrique utilisée pour mesurer la qualité du clustering. Elle est définie entre deux variables aléatoires comme [89] :

$$\mathrm{NMI}(X;Y) = \frac{I(X;Y)}{\sqrt{H(X)H(Y)}}$$

où X désigne les étiquettes réelles, Y est le cluster obtenu, $I(X;Y)$ est l'information mutuelle entre X et Y, et $H(X)$ et $H(Y)$ désignent l'entropie utilisée, qui normalise la valeur de l'information mutuelle dans une plage de $[0, 1]$.

4.5 Expérimentations et résultats

En se basant sur l'architecture DCEC [38] et Keskar et al. [50], on se limitera dans nos expériences sur une taille de batch principalement comprise entre 8 et 512.

Selon [50] la méthode de descente de gradient stochastique et ses variantes fonctionnent dans un régime de petits lots dans lequel une fraction des données d'apprentissage, généralement 32 à 512 images, est échantillonnée pour calculer une approximation du gradient. Nous avons observé dans la pratique que l'utilisation d'un lot plus important entraîne une dégradation significative de la qualité du modèle, mesurée par sa capacité à généraliser. Le manque de capacité de généralisation est dû au fait que les méthodes à grands lots ont tendance à converger vers des minima nets de la fonction d'apprentissage. Ces minima sont caractérisés par de grandes valeurs propres positives et ont tendance à moins bien se généraliser. En revanche, les

	ACC	**NMI**
Guo et. al [38]	88,97 %	$88,49\%$
JND [25]	$90,25\%$	$89,47\%$
Y.B. Proposée	$97,48\%$	$93,73\%$

Table 2 – Comparaison des résultats obtenus sur l'architecture DCEC [38]

méthodes à petits lots convergent vers des minima plats caractérisés par de petites valeurs propres positives. On a observé que l'ensemble des fonctions de perte des réseaux neuronaux profonds est tel que les méthodes qui traitent des grands échantillons sont presque invariablement attirées vers des régions présentant des minima aigus et qui, contrairement aux méthodes à petits lots, sont incapables d'échapper aux bassins de ces minima.
Les résultats que nous avons obtenus ont été principalement inspirés de Smith et al. [96] "Don't Decay the Learning Rate, Increase the Batch Size". Les auteurs de [96] suggèrent que nous pouvons souvent obtenir les avantages de la décroissance du taux d'apprentissage en augmentant plutôt la taille du lot pendant la formation.
Ils soutiennent cette affirmation avec des expériences sur les bases de données CIFAR-10 et ImageNet[96], et avec une gamme d'optimiseurs, y compris la descente de gradient stochastique et Adam en augmentant le taux d'apprentissage et le paramètre de momentum m, tout en mettant à l'échelle $B \propto /(1-m)$.

On note que nous avons réalisé plusieurs expériences avec différents hyperparamètres, qui ont abouti aux résultats décrits dans le tableau 2 en augmentant la taille du lot à B=512 avec un taux d'apprentissage $\eta = 10^{-3}$ en fixant l'intervalle de mis à jour à 40 images.

Pour visualiser les résultats du clustering, on utilisera l'algorithme t-SNE(t-distributed stochastic neighbor embedding).Il a été développé par Geoffrey Hinton et Laurens van der Maaten [107], et comprend deux étapes principales. Premièrement, t-SNE construit une distribution de probabilité

sur des paires d'objets de grande dimension de telle sorte que des objets similaires se voient attribuer une probabilité plus élevée tandis que des points différents se voient attribuer une probabilité plus faible $p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}$. Deuxièmement, t-SNE définit une distribution de probabilité similaire sur les points de la carte de faible dimension $q_{ij} = \frac{(1+\|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l\neq k}(1+\|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$ et minimise la divergence Kullback – Leibler (divergence KL) précédemment expliquée entre les deux distributions, en ce qui concerne les emplacements des points dans la carte $\mathrm{KL}(P \parallel Q) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$ par un algorithme quelconque de la descente gradient. Alors que $p_{i|j}$ tend à converger facilement, ceci rend les similarités entre les points énormes à cause de l'utilisation de la distance euclidienne.

On note que t-SNE est fondamentalement de complexité en temps $\mathcal{O}(N^2)$en raison de la nécessité de calculer des distances par paires ce qui vient à itérer sur l'ensemble des points par deux boucles emboîtées. Le prochain paragraphe introduira une méthode linéaire efficace et plus rapide que la t-SNE. On introduit ci-dessous l'algorithme de la méthode .

tSNE

inputs : Donnees d'entrees Y, nobmre de points N
output: Nuage de points des clusters
a) Calculer les probabilités
$p_{i|j}$**indice de similarités des points**
b) Calculer la matrice de probabilités jointes
$p_i j = p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$
c) Calculer la matrice q_{ij} **pour les points** y_i
d) utiliser la descente pour mettre a jour les points y_i
et minimiser la fonction de perte de $\sum_i^N \sum_j^N p_{ij} \log \frac{p_{ij}}{q_{ij}}$

Algorithme 7: l'algorithme t-SNE

La visualisation des classes en utilisant t-SNE sur un sous-ensemble aléatoire de MNIST 1000 échantillons est présentée la figure 8. On remarque que la forme de chaque cluster est presque maintenue. On conclut que l'architecture DCEC peut préserver la structure intrinsèque des données.

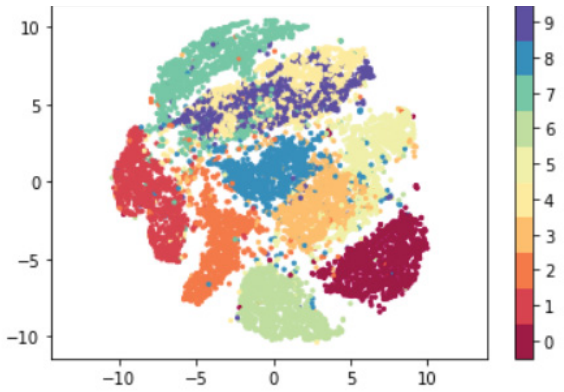

Figure 8 – Visualisation des résultats du clustering sur un sous-ensemble de MNIST pendant la 50^{eme} époque

5 Adaptive Self-Paced Deep Clustering with Data Augmentation

La plupart des algorithmes de clustering profond existants règlent les paramètres du DNN en utilisant une fonction de perte définie par les centres et les affectations des centres de clusters, qui sont généralement obtenus sur la base des sorties du DNN dans la dernière itération. Cependant, nous observons que ces méthodes ne sont pas assez robustes car elles ne considèrent pas explicitement l'effet des exemples marginaux,(i.e les éléments se trouvant sur les frontières de deux ou plusieurs clusters) sur le réseau.

Comme l'objectif du DNN est d'apprendre des caractéristiques qui sont plus appropriées pour le clustering, les exemples près des limites du cluster peuvent ne pas fournir une orientation convaincante. Ceci est en contraste avec l'apprentissage supervisé où toutes les étiquettes cibles sont données au préalable et où donc tous les exemples peuvent fournir des signaux de supervision fiables. En fait, les exemples marginaux dans l'apprentissage supervisé jouent un rôle plus important dans la recherche des limites de classe. D'un autre côté, ces algorithmes de clustering ne tiennent pas compte de la technique d'augmentation des données qui a été largement employée dans

les modèles d'apprentissage profond supervisés pour améliorer la généralisation.
Guo et al.[39] essaye donc d'exploiter ces deux ingrédients afin d'améliorer la performance du clustering et le rendre plus robuste à ce type d'instance.

Pour résoudre ce problème, Guo et al.[39] propose un algorithme de regroupement profond en deux étapes en incorporant l'augmentation des données et l'apprentissage à rythme libre [56].
Spécifiquement, dans la première étape l'algorithme apprend des caractéristiques robustes en formant un autoencodeur avec des exemples qui sont augmentés. Ensuite, dans la deuxième étape, on stimule les caractéristiques apprises à être orientées vers les clusters en affinant alternativement l'encodeur avec les exemples augmentés et en mettant à jour les affectations de clusters des exemples propres.
Pendant le réglage fin des hyperparamètres de l'encodeur, la cible de chaque exemple augmenté dans la fonction de perte est le centre de la classe à laquelle l'exemple propre est affecté.
Pour stabiliser l'apprentissage du réseau, on sélectionne les exemples les plus fiables à chaque itération en utilisant l'apprentissage adaptatif self-paced.

5.1 Augmentation des données

On entend par l'augmentation des données une technique utilisée pour accroître la quantité de données en ajoutant des copies légèrement modifiées de données existantes ou des données synthétiques nouvellement créées à partir de données existantes.
Elle agit comme un régularisateur et permet de réduire le sur-ajustement lors de l'apprentissage d'un modèle d'apprentissage automatique. Elle permet aussi d'améliorer la précision de prédiction des modèles en ajoutant

davantage de données d'entraînement dans les modèles et augmenter la capacité de généralisation des modèles.
Les techniques les plus simples de l'augmentation des données sont :

- remplissage (ajout de pixels de remplissage supplémentaires autour de la limite de notre image d'entrée, augmentant ainsi la taille effective de l'image)
- rotation aléatoire
- remise à l'échelle
- retournement vertical et horizontal
- translation (l'image est déplacée dans les directions X et Y)
- recadrage
- zooming
- assombrissement et éclaircissement/modification des couleurs
- mise à l'échelle des gris
- modification du contraste
- ajout de bruit
- effacement aléatoire

Il existe des modèles plus avancés d'augmentation des données qui ne sont pas utilisées par [39] sont :

- Adverserial training : Il génère des exemples contradictoires qui perturbent un modèle d'apprentissage automatique et les injecte dans l'ensemble de données pour l'entraînement.

- Les réseaux adversaires génératifs (GAN) : Les algorithmes GAN peuvent apprendre des modèles à partir d'ensembles de données d'entrée et créer automatiquement de nouveaux exemples qui ressemblent aux données d'entraînement.

- Transfert de style neuronal : Les modèles neuronaux de transfert de style peuvent mélanger l'image du contenu et l'image du style et séparer le style du contenu.

Perez et al. [79] suggèrent que l'augmentation des données lors de l'apprentissage aide à généraliser le classificateur en montrant empiriquement que l'augmentation aide à prévenir le sur-ajustement.

5.2 Apprentissage Self-paced classique

Les modèles de variables latentes sont un outil puissant pour aborder plusieurs tâches d'apprentissage automatique, tel que l'inférence des paramètres des distributions des entrées. Cependant, les algorithmes d'apprentissage des paramètres de ces modèles sont enclins à rester bloqués dans un mauvais optimum local lorsque les fonctions sont non convexes.
Pour remédier à ce problème, Kumar et al.[56] se basent sur l'intuition que, plutôt que de considérer tous les échantillons simultanément, l'algorithme devrait recevoir les données d'apprentissage dans un ordre significatif qui facilite l'apprentissage. Cette idée simule la procédure d'apprentissage humain : du facile au difficile. Étant donné quelques exemples d'une nouvelle tâche, nous avons tendance à choisir d'abord les exemples les plus faciles pour apprendre des connaissances de base. Après avoir amélioré notre connaissance de la tâche, nous pouvons collecter des exemples plus difficiles pour acquérir plus de connaissances progressivement. Ainsi, l'ordre des échantillons est déterminé par leur degré de « facilité ». Le principal

problème est que, souvent, nous ne disposons pas d'une mesure facilement calculable de la « facilité » des échantillons.

Dans ce contexte, Kumar et al. [56] définit la « **facilité** » des exemples de deux façons, un échantillon est facile si:

- nous sommes confiants quant à la valeur d'une variable cachée.
- s'il est facile de prédire sa véritable sortie.

Ces deux définitions sont en quelque sorte liées : si nous sommes plus certains de la variable cachée, nous pouvons être plus sûrs de la prédiction. Elles sont différentes dans la mesure où la certitude n'implique pas n'implique pas l'exactitude, et les variables cachées peuvent ne pas être directement pertinentes à ce qui rend la sortie d'un échantillon facile à prédire.
L'article [56] se concentre donc sur la deuxième définition : les échantillons faciles sont ceux dont la sortie peut être prédite facilement (sa probabilité est élevée, ou elle se situe loin de la marge du cluster).

Formellement, étant donné des exemples d'apprentissage $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ et un modèle d'apprentissage f avec des paramètres $\mathbf{w}$, le problème traditionnel d'apprentissage automatique est le suivant

$$\min_{\mathbf{w}} \sum_{i=1}^{n} L\left(f_{\mathbf{w}}\left(x_i\right), y_i\right)$$

Pour faire ceci, on introduit une mesure de « facilité » qui pénalise la fonction de perte une fois qu'elle dépasse un certain seuil pour être sûr de choisir les exemples simples en premier, Donc le problème d'optimisation

de l'auto-apprentissage est le suivant

$$\min_{\mathbf{w},\mathbf{v}} \sum_{i=1}^{n} v_i L\left(f_{\mathbf{w}}\left(x_i\right), y_i\right) + g\left(\lambda, v_i\right)$$
$$\text{avec} \quad v_i \in [0,1]$$

où $\mathbf{v} = [v_1, v_2, \ldots, v_n]^\top$ sont des poids d'exemples et $g(\lambda, v_i)$ est appelé terme de régularisation autoproportionnelle.

Considérons l'apprentissage automatique simple à pondération dure où $g(\lambda, v_i) = -\lambda v_i$ et $v_i 0, 1\}$, c'est-à-dire,

$$\min_{\mathbf{w},\mathbf{v}} \sum_{i=1}^{n} v_i L\left(f_{\mathbf{w}}\left(x_i\right), y_i\right) - \lambda v_i$$
$$\text{avec} \quad v_i \in \{0,1\}$$

Étant donné les poids des exemples $\mathbf{v}$, la minimisation sur $\mathbf{w}$ est un problème de minimisation des pertes pondérées. Et lorsque le paramètre de modèle $\mathbf{w}$ est fixé, le v_i optimal est déterminé par la forme fermée

$$v_i = \begin{cases} 1, & \text{si } L_i < \lambda \\ 0, & \text{sinon} \end{cases}$$

5.3 Apprentissage self-paced adaptatif

On considère le problème de clustering avec les paramètres suivants: le nombre de classes K est donné comme une connaissance a priori et le centre de la classe j est représenté par $\mathbf{m}_j \in \mathbb{R}^d$. Définissons la matrice de regroupement comme $\mathbf{M} = [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_K] \in \mathbb{R}^{d \times K}$. Soit $s_i \in \{0,1\}^K$ désigne l'indice de cluster attribué à l'exemple z_i. Alors $y_i = \mathbf{M}s_i$ indique le centre du cluster auquel appartient l'exemple z_i.

SPL $(\mathbf{w}_{t+1}, \mathbf{v}_{t+1}) =$
$\underset{\mathbf{w}\in\mathbb{R}^d, \mathbf{v}\in\{0,1\}^n}{\operatorname{argmin}} \left(r(\mathbf{w}) + \sum_{i=1}^{n} v_i f\left(\mathbf{x}_i, \mathbf{y}_i; \mathbf{w}\right) - \frac{1}{K}\sum_{i=1}^{n} v_i\right)$
inputs: $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}, w_0, K_0.\epsilon$
$t \leftarrow 0, K \leftarrow K_0$
while $v_i \neq 1 \forall i = 1, .., N$ **do**
mettre a jour $h_i^* = \operatorname{argmax}_{h_i\in\mathcal{H}} \mathbf{w}_t^\top Phi\left(\mathbf{x}_i, \mathbf{y}_i, \mathbf{h}_i\right)$
mettre a jour $\mathbf{w}_{t+1}$ en utilisant la recherche alternative convexe pour minimizer l'objectif $\frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n} v_i\xi_i - \frac{1}{K}\sum_{i=1}^{n} v_i$
$t \leftarrow t+1, K \leftarrow \frac{K}{\mu}$
end

Algorithme 8: Algorithme d'apprentissage self-paced

Notre objectif est de trouver un bon réseau neuronal (extracteur de caractéristiques) $f_{\mathbf{w}}(\cdot)$ qui permette aux points transformés $\{z_i\}_{i=1}^n$ d'être plus adaptés au clustering. À cette fin, nous proposons un modèle de regroupement profond en deux étapes pour former $f_{\mathbf{w}}(\cdot)$. Comme précédemment précisé, l'objectif est d'optimiser :

$$\min_{\mathbf{v},\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} v_i \parallel f_{\mathbf{w}}\left(x_i\right) - y_i \parallel_2^2 - \lambda v_i, \quad \text{avec} \quad v_i \in \{0,1\}$$

où $\mathbf{v} = [v_1, v_2, \ldots, v_n]^\top$ sont les poids des exemples d'apprentissage, et λ est le paramètre d'âge qui contrôle le nombre d'exemples sélectionnés[39].

En appliquant le principe SPL, nous empêcherons l'apprentissage automatique de sélectionner les exemples marginaux i.e. proches des limites au début de la formation du réseau, ce qui est nocif à la formation des clusters même à la fin [39].

Dans ce contexte, Guo et al. [39] propose le SPL adaptatif **ASPC** qui a comme objectif d'optimiser

$$\min_{\mathbf{v},\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} v_i \left\| f_{\mathbf{w}}\left(x_i\right) - \mathbf{M}s_i\right\|_2^2 - \lambda v_i$$

$$\text{avec } v_i \in \{0,1\}$$

et

$$\min_{s} \frac{1}{n} \sum_{i=1}^{n} \|f_{\mathbf{w}}(x_i) - \mathbf{M}s_i\|_2^2$$

$$\text{avec } s_i \in \{0,1\}^K, \mathbf{1}^\top s_i = 1,$$

en considérant λ, comme étant un paramètre adapté au coût de la formation en fonction des itérations du modèle et non pas un hyperparamètre indépendant et statique définit par :

$$\lambda = \mu\left(L^t\right) + \frac{t}{T}\sigma\left(L^t\right)$$

où L^t représente toutes les pertes dans la t^{me} itération, T est le nombre d'itérations maximales, $\mu(\cdot)$ et $\sigma(\cdot)$ sont la moyenne et l'écart type des pertes.

Rendu à ce point-là où le problème d'optimisation est fixé, on intègre directement les exemples résultants de l'augmentation des données dans le réseau. Cette intuition est appuyée par le fait que ces exemples appartiennent au même espace topologique des exemples originaux.
Si on note l'ensemble des transformations qui génèrent les nouveaux exemples par la notation suivante:

$$\mathcal{T}: \begin{array}{ccc} \mathbb{R}^d & \rightarrow & \mathbb{R}^d \\ x_i & \mapsto & \hat{x}_i = \mathcal{T}(x_i) \end{array}$$

alors on a bien que la fonction cible à minimiser devient :

$$L_r = \frac{1}{n} \sum_{i=1}^{n} \|\hat{x}_i - h_{\mathbf{u}}\left(f_{\mathbf{w}}\left(\hat{x}_i\right)\right)\|_2^2$$

On rappelle que : $f_{\mathbf{w}}$ et $h_{\mathbf{u}}$ sont les fonctions de compression et de reconstruction respectivement.

ASPCDA $\min_{\mathbf{v},\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} v_i \left\|f_{\mathbf{w}}\left(x_i\right)-\mathbf{M}s_i\right\|_2^2-\lambda v_i$

inputs : *Ensemble de données X, Tranformation d'auguementation $\mathcal{T}$,Nombre de clusters K, seuil d'arret δ, Maximum d'iterations T*

output: *Affectation des classes s*

Initialiser

$\mathbf{w}=\underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}}\frac{1}{n}\sum_{i=1}^{n}\left\|\hat{x}_i-h_{\mathbf{u}}\left(f_{\mathbf{w}}\left(\hat{x}_i\right)\right)\right\|_2^2$ *avec* $\hat{x}_i=\mathcal{T}(x_i)$

Initialiser M, s en utilisant $k-means$ sur la sortie de l'encodeur

for $\forall t=0,1,..,T$ **do**

Mettre a jour

$$s_{ij}\leftarrow s_{ij}=\begin{cases}1, & si\ j=\arg\min_k \left\|f_{\mathbf{w}}\left(x_i\right)-\mathbf{m}_k\right\|_2^2\\ 0, & sinon\end{cases}$$

Mettre a jour $v_i \leftarrow v_i=\begin{cases}1, & si\ \left\|f_{\mathbf{w}}\left(\hat{x}_i\right)-\mathbf{M}s_i\right\|_2^2<\lambda\\ 0, & sinon\end{cases}$

Mettre a jour $\mathbf{W}\leftarrow_{\mathbf{w}}\frac{1}{n}\sum_{i=1}^{n} v_i\left\|f_{\mathbf{w}}\left(\hat{x}_i\right)-\mathbf{M}s_i\right\|_2^2$

if $1-\frac{1}{n}\sum_{i,j} s_{ij}^t s_{ij}^{t-1}<\delta$ **then**

Arret

end

end

Algorithme 9: Algorithme d'apprentissage self-paced adaptatif avec l'augumentation des données

5.4 Expérimentations et résultats

Cette partie se base principalement sur l'algorithme ASPCDA proposé par [39].

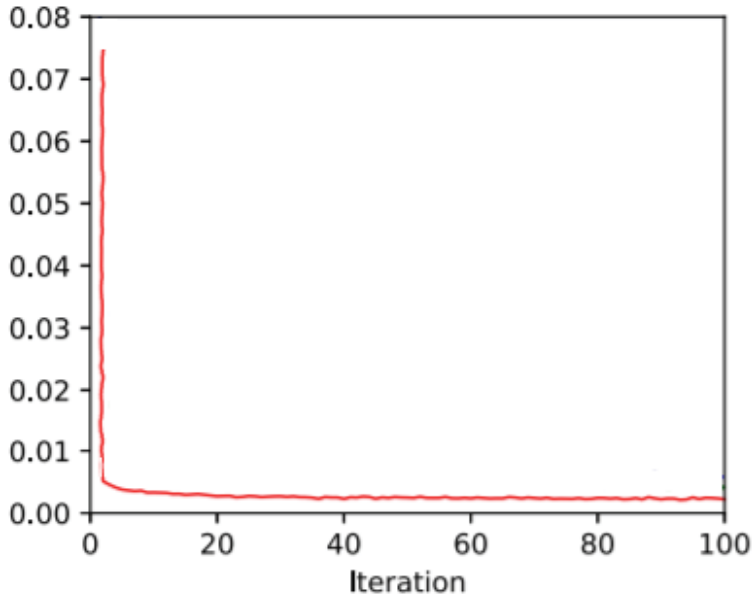

Figure 9 – Graphique de perte de l'entraînement en fonction des itérations

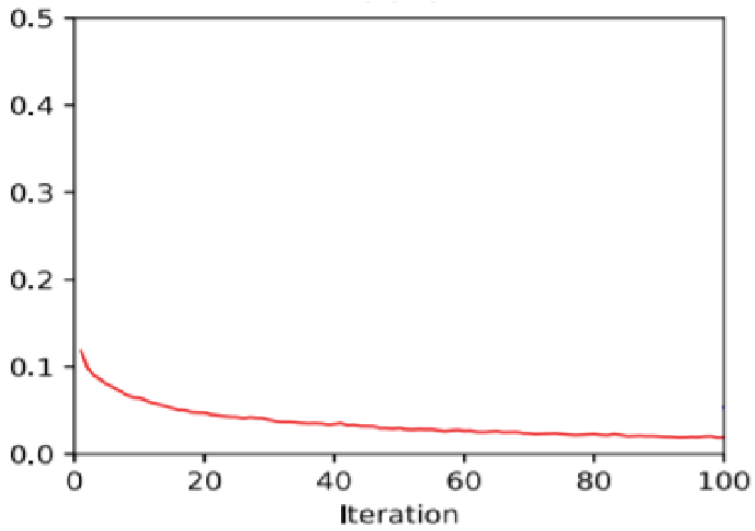

Figure 10 – Graphique de perte du cluserting en fonction des itérations

Dans la phase de pré-entraînement, pour améliorer les résultats du benchmark de Guo et al.[39], nous avons utilisé un autoencodeur composé de huit couches entièrement connectées et activées par ReLU. L'autoencodeur est entraîné pendant 500 époques en utilisant l'optimiseur SGD avec un taux d'apprentissage $\eta = 1{,}0$.

Pendant le paramétrage, l'encodeur possède quatre couches avec le nombre maximal d'itérations fixé à T=100. Dans chaque itération, nous entraînons l'encodeur pendant une époque en utilisant l'optimiseur Adam avec un taux d'apprentissage initial de η =0,0001 et une taille du lot comprise principalement entre 32 à 512 avec un seuil du critère d'arrêt fixé à δ= 0.001.

Les résultats de l'optimisation des fonctions de perte de l'entraînement fig.9 et de clustering fig.10 diminuent rapidement vers 0 lorsque l'itération de formation augmente. En d'autres termes, cette méthode converge dans la pratique.

Les résultats obtenus (disponible sur le présent répertoire `https://github.com/youssefbar/memoire`) au paramétrage précédemment mentionné de l'architecture sont quasi identiques à celles du benchmark de Guo et al.[39] avec une précision de 0.98 $\pm$0.009.

6 Conclusion et Perspectives

L'objectif de ce mémoire est d'établir une étude comparative des méthodes de classification et de contourner les limitations de ces dernières. Plus précisément, on a montré l'efficacité des réseaux de neurones, justement les autoencodeurs dans les bases de données complexes et volumineuses.
Cependant, malgré la puissance computationnelle énorme des autoencodeurs variationnelles classiques et leur capacité d'inférence, ces méthodes ne sont pas assez robustes pour traiter les classes qui se chevauchent, même en ajoutant une méthode classique de classification (généralement K-means ou le modèle de mélange gaussien) au niveau de la couche latente. Notre contribution dans le cadre de ce mémoire se situe à trois niveaux afin de remédier à certaines limitations. Le premier est certainement le plus important. Il concerne la sélection d'exemple par l'intégration de l'apprentissage adapté dans les autoencodeurs. Le deuxième niveau se concentre sur l'arrimage entre le processus d'apprentissage non supervisé des caractéristiques à l'apprentissage supervisé par l'augmentation des données. Le but est de généraliser le modèle d'apprentissage. Le troisième niveau concerne l'optimisation des hyperparamètres eu égard aux données d'application, nommément la base de données MNIST. En effet, les résultats obtenus ont montré que cette méthode d'apprentissage adapté surpassent les résultats des méthodes de classification existantes. Ce mémoire vise à mettre en exergue l'apport et l'impact d'intégrer l'apprentissage adapté pour l'analyse des données MNIST.
En guise de perspectives, il serait intéressant d'essayer d'autres critères de mesure de « facilité » des échantillons pour pénaliser l'algorithme d'apprentissage adapté, ainsi que d'essayer d'incorporer les techniques d'augmentation pour les données qui ne sont pas sous forme d'images. Il serait aussi avantageux de mener une étude comparative prenant en compte les trans-

formateurs en vision[27] et les réseaux de neurones à graphe hiérarchique de classification[11].

References

[1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[3] Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, Maximilian Strobel, and Daniel Cremers. Clustering with deep learning: Taxonomy and new methods. *arXiv preprint arXiv:1801.07648*, 2018.

[4] David Alvarez-Melis, Stefanie Jegelka, and Tommi S Jaakkola. Towards optimal transport with global invariances. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1870–1879. PMLR, 2019.

[5] Matthew Amodio and Smita Krishnaswamy. Magan: Aligning biological manifolds. In *International Conference on Machine Learning*, pages 215–223. PMLR, 2018.

[6] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[7] Fayeem Aziz, Aaron SW Wong, and Stephan Chalup. Semi-supervised manifold alignment using parallel deep autoencoders. *Algorithms*, 12(9):186, 2019.

[8] Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Scalable nearest neighbor search for optimal transport. In *International Conference on Machine Learning*, pages 497–506. PMLR, 2020.

[9] Ari S Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

[10] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[11] Thomas Bonald, Bertrand Charpentier, Alexis Galland, and Alexandre Hollocou. Hierarchical graph clustering using node pair sampling, 2018.

[12] Daniel B Burkhardt, Jay S Stanley, Ana Luisa Perdigoto, Scott A Gigante, Kevan C Herold, Guy Wolf, Antonio J Giraldez, David van Dijk, and Smita Krishnaswamy. Quantifying the effect of experimental perturbations in single-cell rna-sequencing data using graph signal processing. *BioRxiv*, page 532846, 2019.

[13] Ovidiu Calin. *Deep learning architectures*. Springer, 2020.

[14] Bruno Cessac. A view of neural networks as dynamical systems. *International Journal of Bifurcation and Chaos*, 20(06):1585–1629, 2010.

[15] Liqun Chen, Zhe Gan, Yu Cheng, Linjie Li, Lawrence Carin, and Jingjing Liu. Graph optimal transport for cross-domain alignment. In *International Conference on Machine Learning*, pages 1542–1553. PMLR, 2020.

[16] Zhi Chen, Yijie Bei, and Cynthia Rudin. Concept whitening for interpretable image recognition. *Nature Machine Intelligence*, 2(12):772–782, 2020.

[17] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[18] Taco Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Convolutional networks for spherical signals. *arXiv preprint arXiv:1709.04893*, 2017.

[19] Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

[20] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13(1):795–828, 2012.

[21] Corinna Cortes and Vladimir Vapnik. Support vector machine. *Machine learning*, 20(3):273–297, 1995.

[22] Nello Cristianini, Jaz Kandola, Andre Elisseeff, and John Shawe-Taylor. On kernel target alignment. In *Innovations in machine learning*, pages 205–256. Springer, 2006.

[23] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26:2292–2300, 2013.

[24] Philippe Pérez de San Roman, Jenny Benois-Pineau, Jean-Philippe Domenger, Florent Paclet, Daniel Cataert, and Aymar De Rugy. Saliency driven object recognition in egocentric videos with deep cnn: toward application in assistance to neuroprostheses. *Computer Vision and Image Understanding*, 164:82–91, 2017.

[25] Jean Noel Diouf. Classification, aprrentissage pronfond et reseaux de neurones. In *Mémoire de maîtrise, Université du Québec à Trois-Rivières*, 2020.

[26] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1723–1732, 2015.

[27] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Min-

derer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[28] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[29] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

[30] Aude Genevay, Lénaic Chizat, Francis Bach, Marco Cuturi, and Gabriel Peyré. Sample complexity of sinkhorn divergences. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1574–1583. PMLR, 2019.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[32] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[33] John C Gower. Generalized procrustes analysis. *Psychometrika*, 40(1):33–51, 1975.

[34] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[35] Alexander Grigor'yan, Jiaxin Hu, and Ka-Sing Lau. Heat kernels on metric measure spaces. In *Geometry and Analysis of Fractals*, volume 88 of *PROMS*, pages 147–207, 2014.

[36] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[37] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *Ijcai*, pages 1753–1759, 2017.

[38] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep clustering with convolutional autoencoders. In *International conference on neural information processing*, pages 373–382. Springer, 2017.

[39] Xifeng Guo, Xinwang Liu, En Zhu, Xinzhong Zhu, Miaomiao Li, Xin Xu, and Jianping Yin. Adaptive self-paced deep clustering with data augmentation. *IEEE Transactions on Knowledge and Data Engineering*, 32(9):1680–1693, 2019.

[40] Boris Hanin. Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics*, 7(10):992, 2019.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[42] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[43] Martin Hofmann. Support vector machines-kernels and the kernel trick. *Notes*, 26(3):1–16, 2006.

[44] Lu Hongtao and Zhang Qinchuan. Applications of deep convolutional neural network in computer vision. *Journal of Data Acquisition and Processing*, 31(1):1–17, 2016.

[45] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. *arXiv preprint arXiv:1806.10758*, 2018.

[46] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[47] Taicheng Huang, Zonglei Zhen, and Jia Liu. Semantic relatedness emerges in deep convolutional neural networks designed for object recognition. *Frontiers in computational neuroscience*, 15:16, 2021.

[48] Jay S. Stanley III, Scott Gigante, Guy Wolf, and Smita Krishnaswamy. *Harmonic Alignment*, pages 316–324.

[49] Leonid V Kantorovich. On the translocation of masses. *Journal of mathematical sciences*, 133(4):1381–1382, 2006.

[50] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[51] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *International Conference on Machine Learning*, pages 1857–1865. PMLR, 2017.

[52] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[53] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[55] Manik Kuchroo, Jessie Huang, Patrick Wong, Jean-Christophe Grenier, Dennis Shung, Alexander Tong, Carolina Lucas, Jon Klein, Daniel Burkhardt, Scott Gigante, et al. Multiscale phate exploration of sars-cov-2 data reveals multimodal signatures of disease. 2020.

[56] M Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23:1189–1197, 2010.

[57] John Lafferty, Guy Lebanon, and Tommi Jaakkola. Diffusion kernels on statistical manifolds. *Journal of Machine Learning Research*, 6(1), 2005.

[58] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[59] Tam Le, Viet Huynh, Nhat Ho, Dinh Phung, and Makoto Yamada. On scalable variant of wasserstein barycenter. *ArXiv Preprint*, 2019:2, 1910.

[60] Yann LeCun. 1.1 deep learning hardware: past, present, and future. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 12–19. IEEE, 2019.

[61] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[62] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, John Denker, Harris Drucker, Isabelle Guyon, UA Muller, Eduard Sackinger,

et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia, 1995.

[63] Roy R Lederman and Ronen Talmon. Common manifold learning using alternating-diffusion. *submitted, Tech. Report YALEUIDCSITR1497*, 2014.

[64] William Leeb and Ronald Coifman. Hölder–lipschitz norms and their duals on spaces with semigroups, with applications to earth mover's distance. *Journal of Fourier Analysis and Applications*, 22(4):910–953, 2016.

[65] Ofir Lindenbaum, Arie Yeredor, Moshe Salhov, and Amir Averbuch. Multi-view diffusion maps. *Information Fusion*, 55:127–149, 2020.

[66] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[67] Hanxiao Liu, Zihang Dai, David R. So, and Quoc V. Le. Pay attention to mlps, 2021.

[68] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6232–6240, 2017.

[69] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems*, pages 4768–4777, 2017.

[70] Yunqian Ma and Yun Fu. *Manifold learning theory and applications*. CRC press, 2011.

[71] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[72] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[73] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 19(6):1236–1246, 2018.

[74] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[75] Kevin R Moon, Jay S Stanley III, Daniel Burkhardt, David van Dijk, Guy Wolf, and Smita Krishnaswamy. Manifold learning-based methods for analyzing single-cell rna-sequencing data. *Current Opinion in Systems Biology*, 7:36–46, 2018.

[76] Kevin R Moon, David van Dijk, Zheng Wang, Scott Gigante, Daniel B Burkhardt, William S Chen, Kristina Yim, Antonia van den Elzen, Matthew J Hirn, Ronald R Coifman, et al. Visualizing structure and transitions in high-dimensional biological data. *Nature biotechnology*, 37(12):1482–1492, 2019.

[77] Debarghya Mukherjee, Aritra Guha, Justin M Solomon, Yuekai Sun, and Mikhail Yurochkin. Outlier-robust optimal transport. In *International Conference on Machine Learning*, pages 7850–7860. PMLR, 2021.

[78] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[79] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[80] Joshua C Peterson, Paul Soulos, Aida Nematzadeh, and Thomas L Griffiths. Learning hierarchical visual representations in deep neural networks using hierarchical linguistic labels. *arXiv preprint arXiv:1805.07647*, 2018.

[81] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[82] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pages 3360–3368, 2016.

[83] Svetlozar T Rachev. *Probability metrics and the stability of stochastic models*, volume 269. Wiley, 1991.

[84] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability, 2017.

[85] Joan Bruna Raman Arora, Sanjeev Arova. Theory of deep learning, 2021.

[86] Qing Rao and Jelena Frtunikj. Deep learning for self-driving cars: Chances and challenges. In *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, pages 35–38, 2018.

[87] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[88] David Rolnick and Konrad Kording. Reverse-engineering deep relu networks. In *International Conference on Machine Learning*, pages 8178–8187. PMLR, 2020.

[89] Simone Romano, James Bailey, Nguyen Xuan Vinh, and Karin Verspoor. Standardized mutual information for clustering comparisons: One step further in adjustment for chance. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page II–1143–II–1151. JMLR.org, 2014.

[90] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[91] A Saxe, J McClelland, and S Ganguli. Learning hierarchical category structure in deep neural networks. 2013.

[92] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[93] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, pages 3145–3153. PMLR, 2017.

[94] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[95] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[96] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[97] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (TOG)*, 34(4):1–11, 2015.

[98] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[99] Ruoyu Sun. Optimization for deep learning: theory and algorithms. *arXiv preprint arXiv:1912.08957*, 2019.

[100] Xin Sun, Junyu Shi, Lipeng Liu, Junyu Dong, Claudia Plant, Xinhua Wang, and Huiyu Zhou. Transferring deep knowledge for object recognition in low-quality underwater videos. *Neurocomputing*, 275:897–908, 2018.

[101] David Sussillo and Omri Barak. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–649, 2013.

[102] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[103] Matus Telgarsky. Deep learning theory lecture notes. `https://mjt.cs.illinois.edu/dlt/`, 2021. Version: 2021-09-15 v0.0-c15f57bc (alpha).

[104] Bruce Thompson. *Canonical correlation analysis: Uses and interpretation*. Number 47. Sage, 1984.

[105] Kai Tian, Shuigeng Zhou, and Jihong Guan. Deepcluster: A general clustering framework based on deep learning. In *ECML/PKDD*, 2017.

[106] Devis Tuia and Gustau Camps-Valls. Kernel manifold alignment for domain adaptation. *PloS one*, 11(2):e0148655, 2016.

[107] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[108] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[109] Sagar Verma. A survey on machine learning applied to dynamic physical systems. *arXiv preprint arXiv:2009.09719*, 2020.

[110] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[111] Chang Wang, Peter Krafft, Sridhar Mahadevan, Y Ma, and Y Fu. Manifold alignment. In *Manifold Learning: Theory and Applications*, pages 95–120. CRC Press Boca Raton, FL, USA, 2011.

[112] Chang Wang and Sridhar Mahadevan. Manifold alignment using procrustes analysis. In *Proceedings of the 25th international conference on Machine learning*, pages 1120–1127, 2008.

[113] Chang Wang and Sridhar Mahadevan. Manifold alignment without correspondence. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[114] Chang Wang and Sridhar Mahadevan. Heterogeneous domain adaptation using manifold alignment. In *IJCAI proceedings-international joint conference on artificial intelligence*, volume 22, page 1541, 2011.

[115] Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

[116] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.