Université du Québec
Institut National de la Recherche Scientifique
Centre Énergie Matériaux Télécommunications

Interprétation des réseaux de neurones et analyse des opérations en utilisant les formes multilinéaires

Par

Tayssir Doghri

Mémoire présenté pour l'obtention du grade de
Maître es Sciences, M.Sc.
en télécommunications

Jury d'évaluation

Examinateur externe	Professeur Robert Sabourin ETS
Examinateur interne	Professeur Jean-Charles Grégoire INRS-EMT
Directeur de recherche	Professeur Leszek Szczecinski INRS-EMT
Codirecteur de recherche	Professeur Jacob Benesty INRS-EMT
Codirecteur de recherche	Professeur Amar Mitiche INRS-EMT

Remerciements

Que connaissons-nous du monde? Je suis trop jeune pour répondre à cette question. Mais, commençons par: que connaissons-nous de l'apprentissage automatique? Au cours de mes études de maîtrise, j'ai appris beaucoup de choses que nous ne pouvons trouver dans les livres; du moins jusqu'à présent. Pour cela, je tiens à remercier toutes les personnes qui m'ont aidée ou soutenue dans cette quête de savoir et dans la réalisation de ce projet.

Je dois beaucoup à mon directeur de recherche le professeur Leszek Szczecinski. Sa disponibilité, son assistance, ses précieux conseils, ses encouragements, ses remarques, ses analyses, ses interrogations, ses suggestions et sa méthodologie d'aborder les problèmes m'ont beaucoup aidée pour mener à bien mes études de maîtrise et m'ont éclairée pendant mon cheminement.

Je remercie également mes co-directeurs de recherche le professeur Jacob Benesty et le professeur Amar Mitiche, d'avoir consacré du temps pour nos discussions, de leurs conseils et de leurs suggestions tout au long de ce projet.

Je tiens à remercier les membres de jury d'évaluation d'avoir accepté d'évaluer ce travail. Je suis reconnaissante de leurs suggestions et leurs corrections.

À Madame Hélène Sabourin, Mademoiselle Tatiana Brahmi et tous les employés de l'INRS qui ont veillé à ce que nos études se déroulent dans de bonnes conditions.

À mes nouveaux amis que j'ai rencontrés à l'INRS, je dis merci.

À ma famille, le pilier de ma vie, à ma mère, à mon frère Ali, à mes soeurs: Meimouna et Soumaya qui m'ont soutenue et encouragée à donner toujours les meilleurs résultats. Aucun mot ne peut exprimer assez ma gratitude envers eux.

À mon défunt père, ses conseils résonnent encore en moi.

Résumé

Ce mémoire aborde deux problèmes fondamentaux en apprentissage automatique: la perte des relations qui peuvent exister entre les données, par exemple celles qui décrivent la structure spatiale d'une image, et le flot d'opérations qui ne sont généralement pas interprétables dans les réseaux de neurones artificiels. Ce qui, par conséquent, peut compliquer et rendre difficile leur développement.

Nous allons aborder le premier problème dans le cadre du traitement d'images monochromatiques. Dans ce cas, l'approche la plus courante consiste à utiliser une représentation vectorielle de l'image ce qui ignore la relation spatiale entre les pixels. Pour y pallier, nous allons remplacer l'opérateur linéaire conventionnellement utilisé par un opérateur bilinéaire qui agit directement sur l'image en préservant sa nature bi-dimensionnelle. En effet, nous montrons qu'ainsi nous pouvons exploiter plus efficacement les relations spatiales ce qui se traduit par un nombre sensiblement plus faible de paramètres nécessaires pour obtenir les mêmes performances que celles obtenues en utilisant les modèles linéaires courants.

Nous abordons le second problème, celui causé par un flot d'opérations sans interprétation explicite, en explorant l'idée d'expansion de bases par un produit tensoriel. Nous proposons deux approches. La première, inspirée par notre travail sur les opérateurs bilinéaires, repose sur les formes multilinéaires qui nous permettent de manipuler un espace multi-dimensionnel engendré par le produit de fonctions de bases. La deuxième approche suit la même piste tout en résolvant les difficultés d'optimisation que nous pouvons rencontrer avec la première. Nous pouvons interpréter les opérations définissant les couches d'un réseau de neurones en termes de produits de fonctions de bases et de la somme de ces produits, où l'opération dans une couche dépend du type de fonctions d'activations utilisées. Cette interprétation est possible si on considère que le produit est réalisé par une somme dans le domaine logarithmique. Ainsi, elle nous permet de mieux cerner le rôle des paramètres dans les structures des réseaux utilisés actuellement. Plus particulièrement, nous pouvons justifier l'utilisation des contraintes de positivité sur certains paramètres du réseau. Les exemples expérimentaux indiquent que l'approche proposée d'entraînement des réseaux offre une possibilité d'améliorer les résultats de reconnaissance.

Nous évaluons nos modèles sur des données synthétiques et/ou réelles pour résoudre des problèmes de classification.

Mots-clés Classification, Formes bilinéaires, Formes multilinéaires, Tensor product basis expansion

Abstract

In this work, we tackle two problems often encountered in machine learning: the loss of structure of the input data which is generally transformed to a vector and the lack of interpretability of the operations especially in artificial neural networks.

To address the first problem, we focus on the case of monochromatic image pattern recognition. In general, the image is vectorized which leads to the loss of spatial relationship between the pixels. Hence, we propose to replace the conventional linear operation by its bilinear counterpart which takes the image bi-dimensional structure into account. In fact, we show that we can exploit better the spatial relationship which translates into significantly smaller number of parameters required to yield the same performance as linear models.

Then, to tackle the problem related to the interpretability of neural networks, we exploit the idea of tensor product basis expansion. We propose two approaches. In the first one, inspired by our work using bilinear operations, we explore tensor product basis expansion in a multilinear setting. This allows us to work in a high dimensional space created by the product of different basis functions. However, due to a possibility of encountering a vanishing gradient problem, we define a second approach by following the same lead of tensor product basis expansion. We can interpret the operations defining the layers of the neural network as products of basis functions and the sum of those products. The operations of each layer depend on the chosen activation function. This interpretation is possible if we consider that the product is done using a sum in the logarithmic domain which allows us to pin down the role of the parameters in the currently used neural networks. Namely, we can justify the positivity constraints imposed to certain parameters of the network. The numerical results show that the proposed approach can increase the performance of the neural network in term of accuracy.

To test our proposed models, we use synthetic and/or real data and we show the numerical results obtained to solve a classification task.

Keywords Classification, Bilinear forms, Multilinear forms, Tensor product basis expansion

Table des matières

Liste des figures

Liste des algorithmes

Liste des abréviations

BET-NN	Basis Expansion via Tensor product NN
BLR	Bilinear logistic regression
BSR	Bilinear softmax regression
CNN	Convolutional neural network
LLR	Linear logistic regression
LR	Logistic regression
LSR	Linear softmax regression
MLP	Multilayer perceptron
MARS	Multivariate adaptive regression splines
ReLU	Rectified linear unit
SR	Softmax regression
SVD	Singular value decomposition
TMLP	Tensor-based multilayer perceptron
TPB	Tensor-product-basis
TPBE	Tensor-product-basis expansion

Chapitre 1

Introduction

1.1 Contexte du projet

L'apprentissage automatique est un sous-domaine de l'intelligence artificielle visant à définir des opérateurs permettant de réaliser des tâches telles que la classification des données multidimensionnelles et dont les exemples les plus connus sont la reconnaissance d'images ou de la parole. Ces opérateurs sont utilisées, par la suite, afin de prendre de décisions sans aucune intervention humaine. Grâce à l'avancée des capacités de calcul et de stockage, l'apprentissage automatique est devenu un sujet d'actualité et une technologie prometteuse en pleine expansion. Il est maintenant omniprésent avec plusieurs applications comme le traitement automatique des langues, la reconnaissance de formes, les systèmes de recommandations, etc. Nous portons un intérêt particulier à son application pour résoudre des problèmes de classification.

Les problèmes de classification font partie de l'apprentissage automatique supervisé où nous avons un ensemble de données $\{\mathbf{x}_t\}_{t=1}^{T}$ avec leurs étiquettes correspondantes $\{c_t\}_{t=1}^{T}$. L'objectif est d'assigner chaque entrée $\mathbf{x}$ à une des K classes ou catégories C_k avec $k = 1, \ldots, K$. Il existe trois approches pour trouver la solution optimale [5]:

— **Méthode 1:** Déterminer d'abord la densité de probabilité conditionnelle $\Pr(\mathbf{x}|C_k)$ pour chaque classe. Ensuite, déterminer la probabilité à priori de la classe $\Pr(C_k)$. Ainsi, nous

pouvons trouver les probabilités à posteriori $\Pr(C_k|\mathbf{x})$ en utilisant le théorème de Bayes:

$$\Pr(C_k|\mathbf{x}) = \frac{\Pr(\mathbf{x}|C_k)\Pr(C_k)}{\Pr(\mathbf{x})} = \frac{\Pr(\mathbf{x}|C_k)\Pr(C_k)}{\sum_{k=1}^{K}\Pr(\mathbf{x}|C_k)\Pr(C_k)} \tag{1.1}$$

Cette approche est dite générative vue le fait que nous devons déterminer explicitement les probabilités $p(\mathbf{x}|C_k)$ ce qui définit la façon de générer les données.

— **Méthode 2:** Elle est connue sous le nom de modèle discriminatif. Notre objectif consiste à trouver directement, le plus souvent d'une manière approximative, les probabilités à posteriori $\Pr(C_k|\mathbf{x})$ à partir des données.

— **Méthode 3:** Nous essayons de trouver une fonction $f(\mathbf{x})$ appelée fonction discriminante qui associe directement chaque entrée $\mathbf{x}$ à l'étiquette d'une classe de telle sorte que:

$$y = f(\mathbf{x}) \tag{1.2}$$

où y représente l'étiquette de la classe de $\mathbf{x}$. Contrairement à la deuxième approche, les résultats de cette méthode ne peuvent pas être interprétés en tant que des probabilités.

Dans les méthodes 2 et 3, nous assignons à $\mathbf{x}$ la classe qui possède la probabilité à posteriori la plus élevée.

Nous nous intéressons à la deuxième approche où nous modélisons $\Pr(C_k|\mathbf{x})$ directement à partir des données.

1.2 Problématique et objectifs

Plusieurs modèles, ayant une dizaine de paramètres [10, 26] et allant jusqu'à des millions [18, 22, 28], ont été proposés afin de résoudre des problèmes de classification comme les réseaux de neurones artificiels par exemple. Par conséquent, une analyse des opérations des modèles s'avère importante afin de mieux comprendre et représenter le problème à résoudre.

Nous commençons par un modèle simple appelé Logistic regression (LR) qui constitue le fondement de plusieurs modèles beaucoup plus complexes. Nous avons remarqué que le point de départ avant d'appliquer LR consiste à représenter les données sous forme d'un vecteur. Malgré que cette opération nous permet de simplifier les opérations à suivre, elle engendre une perte de structure

des données. Prenons l'exemple d'une image monochromatique, comme celle de la figure 1.1, elle possède de nature une structure matricielle dont chaque élément indique l'intensité du pixel dans une position précise dans l'image. Ainsi, en transformant cette image en un vecteur, nous perdons l'information sur le voisinage des pixels. Alors, nous nous sommes fixés l'objectif de développer un

Figure 1.1 – Exemple d'image monochromatique prise de la base de données MNIST.

modèle qui s'adapte aux données en entrée et qui ne requiert pas une représentation vectorielle.

Cependant, même si nous résolvons le problème de perte de structure des données, LR reste un modèle linéaire et ne permet de résoudre que certains nombre de problèmes. Alors, nous passons à un modèle très connu – mais peu interprétable – qui est le réseau de neurones artificiel. A ce sujet, notre objectif consiste à analyser les opérations d'un réseau de neurones conventionnel et proposer un modèle construit selon une logique bien définie visant l'obtention des opérations interprétables. Une telle interprétation nous guide dans le choix des hyperparamètres et des contraintes.

1.3 Contributions

En abordant les problèmes énoncés dans la Section 1.2, nous avons abouti à plusieurs solutions qui constituent les contributions principales de ce mémoire et qui se résument comme suit :

1. Application d'opérateur bilinéaire pour la classification d'images monochromatiques:

   Il nous permet de nous passer de la transformation vectorielle des données qui nous fait perdre leurs structures matricielles. Ainsi, nous explorons les relations spatiales entre les pixels et nous réduisons par conséquent le nombre de paramètres utilisés dans le modèle.

2. Application d'opérateur multilinéaire pour la formation d'expansion de base:

   Nous enrichissons l'ensemble de fonctions de bases en créant un espace multi-dimensionnel engendré par le produit de fonctions d'activation. Nous manipulons ce nouvel espace grâce aux formes multilinéaires qui nous permettent d'opérer implicitement sur des objets tensoriels.

3. Interprétation des réseaux neurones en tant qu'une expansion tensoriel des fonctions de bases:

Nous pouvons interpréter les opérations définissant les couches d'un réseau de neurones en termes de produits (transformés en des opérations de sommation dans le domaine logarithmique) de fonctions de bases et de la somme de ces produits.

1.4 Structure du document

Ce mémoire est organisé comme suit:

— Le chapitre 2 exploite les formes bilinéaires afin de résoudre des problèmes de classification ayant des données qui ont une structure matricielle.

— Le chapitre 3 aborde le problème d'interprétation des réseaux de neurones conventionnels, à savoir MLP. Il explique les détails de développement du modèle TMLP basé sur les formes multilinéaires.

— Le chapitre 4 présente le modèle BET-NN comme une version plus général du réseau de neurones TMLP.

Dans les trois chapitres, nous évaluons nos modèles sur des données synthétiques et/ou réelles. Enfin, nous concluons ce mémoire par quelques remarques et perspectives de ce travail.

Chapitre 2

Modèles bilinéaires pour la classification d'images

> Il y deux façons de gérer les difficultés : les modifier ou s'y adapter.
>
> Phyllis Bottome

La majorité des modèles proposés pour résoudre des problèmes de classification utilisent des opérations linéaires appliqués sur les données en entrée [5]. Pour ce faire, nous devons placer leurs éléments dans un vecteur suivant un certain ordre, e.g., ligne par ligne. Cependant, les données, lorsqu'elles sont acquises, possèdent naturellement une structure multidimensionnelle. Prenons l'exemple des images monochromatiques qui ont naturellement une structure matricielle et où il existe souvent une certaine relation de similarité entre les pixels voisins.

Ainsi, la transformation vectorielle, ignorant la structure de l'image, rend la tâche de classification potentiellement moins efficace. Pour aborder ce problème, nous proposons des modèles qui prennent en considération la structure des données telle qu'elle est obtenus lors de leur acquisition. Nous nous focalisons sur la régression logistique (LR) étant donné qu'elle constitue un modèle de base dans l'apprentissage automatique. Nous remplaçons l'opération linéaire par une opération bilinéaire qui exploite la relation spatiale entre les pixels définissant ainsi une régression logistique bilinéaire (BLR). Nous montrons que nous pouvons obtenir des résultats similaires avec beaucoup moins de paramètres. En effet, pour une image $\mathbf{X} \in \mathbb{R}^{M \times N}$, l'opération linéaire conventionnelle opère sur

une version vectorisée de cette image $\mathbf{x} \in \mathbb{R}^{MN}$, ce qui requiert MN paramètres alors qu'une forme bilinéaire requiert seulement $M + N$ paramètres; une différence qui peut être très grande pour des valeurs élevées de M et N. Si nous utilisons une somme L de formes bilinéaires, le nombre de paramètres augmente linéairement avec L. Ceci nous montre l'importance de la manipulation des données avec leur structure initiale.

L'idée de l'utilisation des formes bilinéaires afin de remplacer la combinaison linéaire des données n'est pas entièrement nouvelle. Par exemple, elle a été adoptée dans le contexte des signaux acoustiques afin de déterminer la réponse du canal acoustique [4, 12]. Dans le domaine de l'apprentissage automatique, [25] propose d'utiliser les traitements bilinéaires avec les machines à vecteurs de support (connu en anglais par support vector machines) et [19] explore les formes bilinéaires pour la classification des signaux médicaux en utilisant LR. Aussi, elles sont utilisées pour les systèmes de recommandation [9] et l'analyse discriminante [14].

Cependant, [19] se limite à BLR de rang-1 et [9, 14] utilisent une forme bilinéaire de rang-1. Dans notre travail, nous proposons BLR de rang supérieur. Ceci est possible grâce à nos algorithmes qui se basent sur l'optimisation alternée des paramètres contrairement à la méthode d'optimization proposée dans [19] dont la convergence n'est pas garantie.

Dans ce qui suit, nous présentons le concept des modèles bilinéaires appliqués sur LR ainsi que leur interprétation.

2.1 Régression logistique conventionnelle

Avant de présenter notre modèle BLR, il est important de discuter la version conventionnelle de la régression logistique que nous nommons régression logistique linéaire (LLR).

LLR nous permet de résoudre des problèmes de classification binaire. Étant donné un ensemble de caractéristiques contenu dans un vecteur $\mathbf{x}_t$, indexé par t, notre objectif est d'estimer la probabilité à posteriori de la classe $C_t \in \{0, 1\}$ à laquelle appartient $\mathbf{x}_t$. En d'autres termes, nous cherchons $y_t = \Pr(C_t = 1|\mathbf{x}_t)$ [1].

1. Dans la classification binaire, nous avons $\Pr(C_t = 0|\mathbf{x}_t) = 1 - y_t$.

LLR est un modèle qui nous permet d'approximer y_t en utilisant une fonction non linéaire appliquée sur une transformation linéaire de $\mathbf{x}_t \in \mathbb{R}^D$ [5]:

$$y_t = \sigma(z_t), \tag{2.1}$$

$$z_t = \mathbf{w}^{\mathrm{T}}\mathbf{x}_t, \tag{2.2}$$

où $\mathbf{w} = [w_1, \cdots, w_D]^{\mathrm{T}} \in \mathbb{R}^D$ contient les paramètres du modèle et $\sigma(\cdot)$ dénote la fonction logistique définie par:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}. \tag{2.3}$$

En utilisant un ensemble d'apprentissage constitué de paires $\{(\mathbf{x}_t, c_t)\}_{t=1}^{T}$ dont $c_t \in \{0, 1\}$ désigne la classe de $\mathbf{x}_t$ et T représente le nombre de données d'apprentissage, nous trouvons les meilleures valeurs des paramètres encapsulés dans le vecteur $\mathbf{w}$ en solutionnant le problème d'optimisation suivant:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w}), \tag{2.4}$$

$$J(\mathbf{w}) = V(\mathbf{w}) + \alpha R(\mathbf{w}), \tag{2.5}$$

où:

$$V(\mathbf{w}) = -\frac{1}{T}\sum_{t=1}^{T}\Big[c_t \log y_t + (1 - c_t)\log(1 - y_t)\Big], \tag{2.6}$$

est l'entropie croisée qui nous permet de mesurer à quel point notre estimation de la probabilité à posteriori y_t est proche de la réalité. Le paramètre α est un facteur de régularisation et $R(\mathbf{w})$ est la fonction de régularisation qui est généralement définie par [5, Ch. 3.1.4]:

$$R(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2 = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}. \tag{2.7}$$

Étant donné que la fonction objective $J(\mathbf{w})$ est convexe, la solution de (2.4) est unique et nous pouvons l'obtenir en utilisant suivant le gradient calculé comme suit:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{T}\sum_{t=1}^{T}(y_t - c_t)\mathbf{x}_t + \alpha\mathbf{w}. \tag{2.8}$$

Ainsi, le fait de considérer les caractéristiques sous forme vectorielle, représentée par $\mathbf{x}_t$, permet de simplifier les calculs. Cependant, toute relation qui peut exister entre les éléments de $\mathbf{x}_t$ est délibérément ignorée. Particulièrement, si nos données d'entrée sont représentées naturellement sous forme matricielle $\mathbf{X}_t \in \mathbb{R}^{M \times N}$, comme le cas des images monochromatiques, la relation spatiale entre les pixels dans $\mathbf{X}_t$ est perdue lors de la transformation vectorielle de $\mathbf{X}_t$ en $\mathbf{x}_t$.

Cette transformation ne change pas seulement la structure des données mais elle rend l'interprétation des résultats moins évidente. En effet, il est plus convenable d'écrire la relation (2.2) sous forme matricielle :

$$z_t = \langle \mathbf{W}, \mathbf{X}_t \rangle, \tag{2.9}$$

où $\langle \cdot, \cdot \rangle$ dénote le produit scalaire. $\mathbf{W}$ est le format matriciel de $\mathbf{w}$ dans (2.2) et il peut être considéré comme un filtre spatial appliqué sur l'image représentée par $\mathbf{X}_t$.

2.2 Régression logistique bilinéaire

Malgré que la formulation de z_t dans (2.9) fournit une interprétation des poids $\mathbf{W}$ dans le domaine des images, ce n'est qu'une réorganisation de ses éléments et la relation entre les pixels n'est pas prise en compte.

Afin de pallier ce problème, nous proposons d'utiliser une formulation bilinéaire définie comme suit:

$$z_t = \sum_{l=1}^{L} \mathbf{a}_l^{\mathrm{T}} \mathbf{X}_t \mathbf{b}_l. \tag{2.10}$$

Les vecteurs $\mathbf{a}_l$ et $\mathbf{b}_l$ sont regroupés dans les matrices:

$$\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_L] \tag{2.11}$$

$$\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_L], \tag{2.12}$$

et nous pouvons les voir comme des filtres appliqués respectivement sur les lignes et les colonnes de l'image $\mathbf{X}$.

Avec ce nouveau modèle proposé, nous remarquons que:

— Pour $L = \min\{M, N\}$, BLR définie avec (2.10) est équivalent à la régression logistique linéaire (2.1) pour un choix spécifique des paramètres $\mathbf{A}$ et $\mathbf{B}$. Nous démontrons cela dans la section 2.2.1.

— Pour $L < \min\{M, N\}$, BLR introduit des dépendances entre les pixels d'une même ligne et d'une même colonne. Nous le démontrons dans la section 2.2.2.

2.2.1 Équivalence entre LLR et BLR

Afin de démontrer l'équivalence entre (2.9) et (2.10), nous écrivons (2.9) comme:

$$z = \langle \mathbf{W}, \mathbf{X} \rangle = \mathrm{Tr}(\mathbf{W}^{\mathrm{T}}\mathbf{X}), \tag{2.13}$$

où $\mathrm{Tr}(\cdot)$ dénote la trace d'une matrice et nous avons omis temporairement l'indexation avec t.

La matrice $\mathbf{W}$ peut être décomposée en utilisant SVD comme suit:

$$\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}} = \sum_{l=1}^{L} s_l \mathbf{u}_l \mathbf{v}_l^{\mathrm{T}}, \tag{2.14}$$

où $\mathbf{U} = [\mathbf{u}_1, \cdots, \mathbf{u}_M] \in \mathbb{R}^{M \times M}$ et $\mathbf{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_N] \in \mathbb{R}^{N \times N}$ sont des matrices orthogonales et $\mathbf{S} = \mathrm{diag}(\mathbf{S}_1, \mathbf{0})$, $\mathbf{S}_1 = \mathrm{diag}(s_1, \cdots, s_L)$.

Ainsi, nous obtenons

$$z = \mathrm{Tr}\Big[\big(\sum_{l=1}^{L} s_l \mathbf{u}_l \mathbf{v}_l^{\mathrm{T}}\big)^{\mathrm{T}} \mathbf{X}\Big] = \sum_{l=1}^{L} s_l \mathbf{u}_l^{\mathrm{T}} \mathbf{X} \mathbf{v}_l. \tag{2.15}$$

En définissant les vecteurs $\mathbf{a}_l = \sqrt{s_l}\mathbf{u}_l$ et $\mathbf{b}_l = \sqrt{s_l}\mathbf{v}_l$, nous obtenons (2.10); nous pouvons aussi écrire (2.14) comme:

$$\mathbf{W} = \sum_{l=1}^{L} \mathbf{a}_l \mathbf{b}_l^{\mathrm{T}}. \tag{2.16}$$

2.2.2 Dépendance conditionnelle induite

Une interprétation intéressante du modèle bilinéaire que nous proposons peut être obtenue si nous considérons le modèle génératif implicite derrière la régression logistique linéaire.

Notamment, si nous supposons que la distribution des données $\mathbf{x}$ conditionnée sur la classe est obtenue par:

$$p(\mathbf{x}|C=1) \propto \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x})g(\mathbf{x}), \tag{2.17}$$

où $g(\cdot)$ est une fonction arbitraire indépendante de la classe C. Ainsi, nous obtenons (2.1) d'une manière similaire à [5, Ch. 4.2.1]. Cette relation signifie aussi que, sachant la classe C, nous connaissons les paramètres w_i et par conséquent, les caractéristiques dans $\mathbf{x}$ sont indépendantes. Ceci est équivalent au fait d'assumer que (2.1) implémente la règle dite « naïve » de Bayes [16, Ch. 6.6.3].

Nous pouvons représenter graphiquement ces dépendances probabilistes comme dans la figure 2.1 à l'aide d'un réseau bayésien où les flèches orientées modélisent les dépendances [3, Ch. 3.3]. D'où, le fait de connaître C bloque tout chemin liant les poids $w_{i,j}$ qui deviennent ainsi indépendants (conditionnellement).

D'autre part, dans le cas de BLR avec $L = 1$ nous pouvons réécrire (2.16) comme $\mathbf{W} = \mathbf{a}\mathbf{b}^{\mathrm{T}}$, c.-à-d, chaque terme de la matrice $\mathbf{W}$ s'écrit comme $w_{i,j} = a_i b_j$ où a_i et b_i sont les éléments des vecteurs $\mathbf{a}$ et $\mathbf{b}$. Cette relation est représentée dans la figure 2.1b. Nous pouvons constater que le fait de connaître C ne bloque pas les chemins entre les poids $w_{i,j}$ et ils restent connectés grâce aux éléments a_i et b_j. Par exemple, il existe un chemin entre $w_{1,1}$ et $w_{1,N}$ via la variable a_1 et qui n'inclut pas C. Nous devons dire que ceci signifie simplement que les éléments $w_{i,j}$ ne sont pas indépendants du point de vue structurel. Cependant, leur indépendance peut être obtenue selon un choix approprié des vecteurs $\mathbf{a}$ et $\mathbf{b}$.

Nous insistons à dire aussi que nous n'avons pas besoin du modèle génératif (2.17) afin de résoudre le problème de classification. Nous l'utilisons plutôt dans la figure 2.1 afin de clarifier la différence entre LLR et BLR. La plus importante réflexion derrière cela est: tandis que les dépendances entre les caractéristiques (dans notre cas, les pixels) sont souvent imposées par la transformation non linéaire de $\mathbf{x}$ (comme dans [16, Ch. 4.1]), avec notre modèle elles sont imposées par la structure même de l'opérateur bilinéaire.

2.2.3 Apprentissage du modèle BLR

Notre objectif est d'apprendre les paramètres des vecteurs $\mathbf{a}_l$ et $\mathbf{b}_l$, regroupés dans les matrices $\mathbf{A}$ et $\mathbf{B}$, directement à partir de l'ensemble d'apprentissage $\{(\mathbf{X}_t, c_t)\}_{t=1}^{T}$. Comme pour la régression

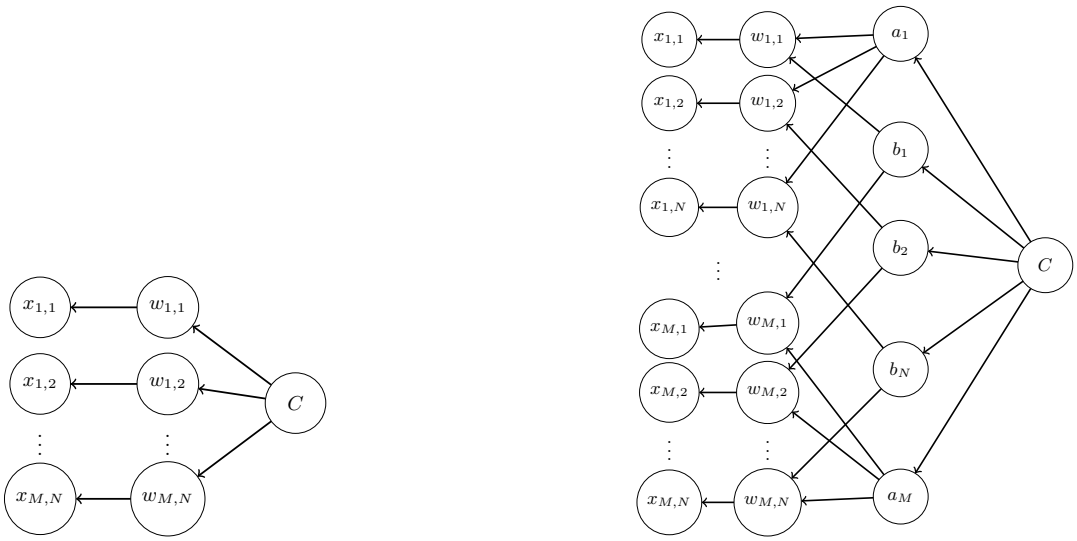


Figure 2.1 – Modèle génératif implicite derrière a) régression logistique conventionnelle et b) régression logistique bilinéaire pour $L = 1$.

logistique conventionnelle, il faut résoudre le problème d'optimisation suivant:

$$[\hat{\mathbf{A}}, \hat{\mathbf{B}}] = \arg\min_{\mathbf{A},\mathbf{B}} J(\mathbf{A}, \mathbf{B}), \tag{2.18}$$

$$J(\mathbf{A}, \mathbf{B}) = V(\mathbf{A}, \mathbf{B}) + \alpha R(\mathbf{A}, \mathbf{B}), \tag{2.19}$$

où $V(\mathbf{A}, \mathbf{B})$ représente la formulation bilinéaire de l'entropie croisée définie dans (2.6). La fonction de régularisation $R(\mathbf{A}, \mathbf{B})$ joue le même rôle que $R(\mathbf{w})$ dans (2.5). Le choix de cette fonction n'est pas aussi trivial. Mais, pour des fins d'apprentissage du modèle, nous supposons qu'elle prend une forme similaire à (2.7), c.-à-d,

$$R(\mathbf{A}, \mathbf{B}) = \frac{1}{2} \sum_{l=1}^{L} \Big(\|\mathbf{a}_l\|_2^2 + \|\mathbf{b}_l\|_2^2 \Big). \tag{2.20}$$

Nous remarquons que i) la fonction (2.5) est convexe par rapport à $\mathbf{w}$ vu la relation linéaire qui se trouve entre z et $\mathbf{w}$ (définie dans (2.2)) et ii) z dans (2.10) n'est pas linéaire en $\mathbf{a}_l$ et $\mathbf{b}_l$. Par conséquent, la convexité de $J(\mathbf{A}, \mathbf{B})$ par rapport à $\mathbf{a}_1, \ldots, \mathbf{a}_L, \mathbf{b}_1, \ldots, \mathbf{b}_L$ n'est pas garantie et l'utilisation des méthodes du gradient directement n'est pas conseillée.

Néanmoins, si nous fixons tout, sauf un, des termes dans $\mathbf{A}$ et $\mathbf{B}$, le problème d'optimisation se transforme en un problème similaire à celui que nous avons résolu dans le cas de LLR dans la section 2.1.

Nous pouvons le constater si nous calculons le gradient de la fonction objective $J(\mathbf{A}, \mathbf{B})$ par rapport à $\mathbf{a}_l$ et $\mathbf{b}_l$:

$$\nabla_{\mathbf{a}_l} J(\mathbf{A}, \mathbf{B}) = \frac{1}{T} \sum_{t=1}^{T} (y_t - c_t) \mathbf{X}_t \mathbf{b}_l + \alpha \mathbf{a}_l, \tag{2.21}$$

$$\nabla_{\mathbf{b}_l} J(\mathbf{A}, \mathbf{B}) = \frac{1}{T} \sum_{t=1}^{T} (y_t - c_t) \mathbf{a}_l^{\mathrm{T}} \mathbf{X}_t + \alpha \mathbf{b}_l, \tag{2.22}$$

ce qui nous donne des équations similaires à (2.8).

Ainsi, la fonction $J(\mathbf{A}, \mathbf{B})$ est convexe par rapport à $\mathbf{a}_l$ ou $\mathbf{b}_l$ si nous fixons tous les autres vecteurs. Ce qui nous incite à utiliser la procédure d'optimisation alternée où nous optimisons par rapport aux vecteurs $\mathbf{a}_l$ et $\mathbf{b}_l$ un à la fois comme décrit dans Algorithme 1. Puisque chaque problème que nous résolvons est convexe, nous pouvons utiliser les méthodes de gradient, c.-à-d, nous optimisons par rapport à $\mathbf{a}_l$ (lignes 11-16 dans Algorithme 1) et après par rapport à $\mathbf{b}_l$ (lignes 17-22 dans Algorithme 1).

Cette optimisation se fait sur plusieurs itérations dénotées par $i = 1, \cdots, i_{\max}$. L'initialisation des poids (lignes 2-7 dans Algorithme 1) est importante afin d'accélérer la convergence.

D'une part, pour chaque l, nous cherchons les poids $\mathbf{a}_l$ et $\mathbf{b}_l$ connaissant $\mathbf{a}_k, \mathbf{b}_k, k < l$ tout en supposant la non contribution des poids $\mathbf{a}_k, \mathbf{b}_k, k > l$. En d'autres termes, nous considérons chaque rang l comme un niveau d'approximation additionnel. Ce qui explique le fait d'initialiser $\mathbf{a}_l$ à zéro. Toutefois, il est possible d'initialiser tous les vecteurs $\mathbf{a}_k$ et $\mathbf{b}_k$ d'une façon aléatoire mais ça va ralentir la convergence puisque, durant l'apprentissage de $\mathbf{a}_l$ et $\mathbf{b}_l$, nous sommes affectés par les valeurs aléatoires assignées aux $\mathbf{a}_k, \mathbf{b}_k, k > l$.

D'autre part, pour un rang l spécifique, puisque $\mathbf{a}_l$ et $\mathbf{b}_l$ affecte z_t à travers la multiplication, nous ne pouvons pas les mettre à zéro car nous obtenons un gradient nul (voir (2.21) et (2.22)); ce qui explique l'initialisation aléatoire de b_l (ligne 3 dans Algorithme 1). Aussi, le fait d'initialiser les vecteurs $\mathbf{b}_l$, $l = 1, \ldots, L$ (lignes 4-7 dans Algorithme 1) d'une façon qu'ils soient orthogonaux va accélérer la convergence. Cette approche est inspirée par la décomposition SVD (2.14). Ceci dit,

Algorithme 1 : Apprentissage du modèle BLR

Initialisation:
$\mathbf{a}_l = \mathbf{0}, l = 1, \ldots, L$
$\mathbf{b}_l =$ distribution uniforme dans [-1,1]
pour $l = 2, \cdots, L$ **faire**
  $\mathbf{S}_{l-1} = \left[\frac{\mathbf{b}_1}{\|\mathbf{b}_1\|}, \cdots, \frac{\mathbf{b}_{l-1}}{\|\mathbf{b}_{l-1}\|}\right]$
  $\mathbf{b}_l \leftarrow \mathbf{b}_l - \mathbf{S}_{l-1}(\mathbf{S}_{l-1}^{\mathrm{T}}\mathbf{b}_l)$
fin
Optimisation:
pour $i = 1, \cdots, i_{\max}$ **faire**
  pour $l = 1, \cdots, L$ **faire**
    tant que $\mathbf{a}_l$ n'a pas convergé **faire**
      $\mathbf{g}_l = \nabla_{\mathbf{a}_l} J(\mathbf{A}, \mathbf{B})$
      $\mathbf{D}_l = [\mathbf{0}, \cdots, \mathbf{g}_l, \cdots, \mathbf{0}]$
      $\hat{\eta} \approx \arg\min_\eta J(\mathbf{A} - \eta\mathbf{D}_l, \mathbf{B})$
      $\mathbf{a}_l \leftarrow \mathbf{a}_l - \hat{\eta}\mathbf{g}_l$
    fin
    tant que $\mathbf{b}_l$ n'a pas convergé **faire**
      $\mathbf{g}_l = \nabla_{\mathbf{b}_l} J(\mathbf{A}, \mathbf{B})$
      $\mathbf{D}_l = [\mathbf{0}, \cdots, \mathbf{g}_l, \cdots, \mathbf{0}]$
      $\hat{\eta} \approx \arg\min_\eta J(\mathbf{A}, \mathbf{B} - \eta\mathbf{D}_l)$
      $\mathbf{b}_l \leftarrow \mathbf{b}_l - \hat{\eta}\mathbf{g}_l$
    fin
  fin
fin

nous n'imposons pas une telle contrainte à notre solution. En effet, le fait de le faire détériore un peu la performance.

Un dernier commentaire à faire concerne la non-unicité de la solution vu la structure même de (2.16). N'importe quelle solution qui prend la forme $(\beta\mathbf{a}_l, \frac{1}{\beta}\mathbf{b}_l)$ donnera exactement les mêmes résultats et ce, parce que le produit dans (2.16) élimine n'importe quel $\beta \neq 0$.

2.2.4 Stratégies de régularisation

Le choix de la fonction de régularisation est souvent incité par la simplicité du problème d'optimisation, ainsi, le choix de (2.20) est justifié par la simplicité du calcul du gradient.

En prenant en considération le fait que l'approche bilinéaire est équivalente à l'approche linéaire, voir section 2.2.1, il peut être intéressant d'utiliser une fonction de régularisation $R(\mathbf{A}, \mathbf{B})$ qui est

équivalente à celle définie dans LLR (2.7), c.-à-d, en utilisant (2.16), nous définissons:

$$R(\mathbf{A},\mathbf{B}) = \frac{1}{2}\|\sum_{l=1}^{L}\mathbf{a}_l\mathbf{b}_l^{\mathrm{T}}\|_{\mathrm{F}}^2 \tag{2.23}$$

via la norme de Frobenius.

Cependant, une telle définition engendre des complexités dans le calcul du gradient $\nabla_{\mathbf{a}_l}R(\mathbf{A},\mathbf{B})$ et $\nabla_{\mathbf{b}_l}R(\mathbf{A},\mathbf{B})$. Par conséquent, nous optons pour (2.20) ou pour:

$$R(\mathbf{A},\mathbf{B}) = \frac{1}{2}\sum_{l=1}^{L}\|\mathbf{a}_l\|_2^2\,\|\mathbf{b}_l\|_2^2\,, \tag{2.24}$$

qui est, en fait, équivalente à (2.20) pour $L = 1$.

Aussi, pour $L = 1$, la régularisation (2.20) donnera les mêmes résultats que la régularisation (2.24). Nous pouvons voir, sans perte de généralité, que les solutions basées sur (2.24) peuvent être forcées à satisfaire $\|\mathbf{a}_1\|_2 = \|\mathbf{b}_1\|_2$ (comme indiqué avant la section 2.2.4, le produit des termes est le plus important et il peut être fixé lors de la normalisation). La même chose peut être dite pour la régularisation (2.20) qui est minimisée pour $\|\mathbf{a}_1\|_2 = \|\mathbf{b}_1\|_2$. En d'autres termes, seule la norme $\|\mathbf{a}_1\|_2 = \|\mathbf{b}_1\|_2$ affecte les solutions pour (2.20) et (2.24).

Cependant, une telle équivalence ne peut pas être garantie si nous augmentons le rang L.

2.3 Généralisation à des problèmes de classification multiple

La généralisation de la régression logistique linéaire qui peut résoudre des problèmes de classification multiple se fait à l'aide de la régression softmax (SR): étant donné K classes, nous cherchons la probabilité à posteriori pour chaque classe $C_t \in \{1, 2, \ldots, K\}$, c.-à-d, $y_{t,k} = \Pr(C_t = k|\mathbf{X}_t), k = 1, \ldots, K$. Ceci est réalisé grâce au modèle suivant:

$$y_{t,k} = \frac{\exp\left(z_{t,k}\right)}{\sum_{j=1}^{K}\exp\left(z_{t,j}\right)}, \tag{2.25}$$

où:

$$z_{t,k} = \langle\mathbf{W}_k, \mathbf{X}_t\rangle, \tag{2.26}$$

et chaque $\mathbf{W}_k, k = 1, \ldots, K$ représente les poids correspondant à la classe k. Pour $K = 2$, l'indexation de la sortie du modèle avec k peut être éliminée tel que discuté lors de la classification binaire avec LLR.

En utilisant les mêmes arguments qu'avant, nous remplaçons le produit scalaire (2.26) par sa formulation bilinéaire:

$$z_{t,k} = \sum_{l=1}^{L} \mathbf{a}_{l,k}^{\mathrm{T}} \mathbf{X}_t \mathbf{b}_{l,k}, \tag{2.27}$$

ce qui nous donne la régression softmax bilinéaire (BSR).

Ensuite, étant donné un ensemble d'apprentissage $\{(\mathbf{X}_t, \mathbf{c}_t)\}_{t=1}^{T}$ où $\mathbf{c}_t \in \mathbb{R}^K$ est le vecteur qui encode la classe tel que $c_{t,k} = 1$ si $C_t = k$, nous cherchons d'apprendre les poids $\mathbf{A}_l = [\mathbf{a}_{l,1}, \cdots, \mathbf{a}_{l,K}] \in \mathbb{R}^{M \times K}$ et $\mathbf{B}_l = [\mathbf{b}_{l,1}, \cdots, \mathbf{b}_{l,K}] \in \mathbb{R}^{N \times K}$ pour $l = 1, \cdots, L$.

Pour ce faire, nous minimisons cette fonction objective:

$$J(\mathbf{A}, \mathbf{B}) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{k=1}^{K} c_{t,k} \ln y_{t,k} + \alpha R(\mathbf{A}, \mathbf{B}), \tag{2.28}$$

où $\mathbf{A} = [\mathbf{A}_1, \ldots, \mathbf{A}_L]$ et $\mathbf{B} = [\mathbf{B}_1, \ldots, \mathbf{B}_L]$ regroupent tous les poids.

Le gradient de la fonction objective $J(\mathbf{A}, \mathbf{B})$ par rapport à $\mathbf{a}_{l,k}$ et $\mathbf{b}_{l,k}$ est donné par:

$$\nabla_{\mathbf{a}_{l,k}} J(\mathbf{A}, \mathbf{B}) = \frac{1}{T} \sum_{t=1}^{T} (y_{t,k} - c_{t,k}) \mathbf{X}_t \mathbf{b}_{l,k} + \alpha \nabla_{\mathbf{a}_{l,k}} R(\mathbf{A}, \mathbf{B}), \tag{2.29}$$

$$\nabla_{\mathbf{b}_{l,k}} J(\mathbf{A}, \mathbf{B}) = \frac{1}{T} \sum_{t=1}^{T} (y_{t,k} - c_{t,k}) \mathbf{a}_{l,k}^{\mathrm{T}} \mathbf{X}_t + \alpha \nabla_{\mathbf{b}_{l,k}} R(\mathbf{A}, \mathbf{B}). \tag{2.30}$$

Nous optimisons $J(\mathbf{A}, \mathbf{B})$ en utilisant la méthode du gradient descendant comme décrit dans Algorithme 2 qui généralise l'apprentissage de BLR défini dans Algorithme 1 pour l'apprentissage de BSR.

Algorithme 2 : Apprentissage du mdoèle BSR

Initialization:
$\mathbf{A} = \mathbf{0}$
$\mathbf{B}$ = drawn from a uniform distribution over [-1,1]
pour $k = 1, \cdots, K$ **faire**
  pour $l = 2, \cdots, L$ **faire**
    $\mathbf{S}_{l-1,k} = \left[\frac{\mathbf{b}_{1,k}}{\|\mathbf{b}_{1,k}\|}, \cdots, \frac{\mathbf{b}_{l-1,k}}{\|\mathbf{b}_{l-1,k}\|}\right]$
    $\mathbf{b}_{l,k} \leftarrow \mathbf{b}_{l,k} - \mathbf{S}_{l-1,k}(\mathbf{S}_{l-1,k}^{\mathrm{T}}\mathbf{b}_{l,k})$
  fin
fin
Optimization:
pour $i = 1, \cdots, i_{\max}$ **faire**
  pour $l = 1, \cdots, L$ **faire**
    tant que $\mathbf{A}_l$ not converged **faire**
      $\mathbf{G}_l = [\nabla_{\mathbf{a}_{l,1}} J(\mathbf{A}, \mathbf{B}), \cdots, \nabla_{\mathbf{a}_{l,K}} J(\mathbf{A}, \mathbf{B})]$
      $\mathbf{D}_l = [\mathbf{0}, \cdots, \mathbf{G}_l, \cdots, \mathbf{0}]$
      $\hat{\eta} \approx \arg\min_\eta J(\mathbf{A} - \eta \mathbf{D}_l, \mathbf{B})$
      $\mathbf{A}_l \leftarrow \mathbf{A}_l - \hat{\eta}\mathbf{G}$
    fin
    tant que $\mathbf{B}_l$ not converged **faire**
      $\mathbf{G}_l = [\nabla_{\mathbf{b}_{l,1}} J(\mathbf{A}, \mathbf{B}), \cdots, \nabla_{\mathbf{b}_{l,K}} J(\mathbf{A}, \mathbf{B})]$
      $\mathbf{D}_l = [\mathbf{0}, \cdots, \mathbf{G}_l, \cdots, \mathbf{0}]$
      $\hat{\eta} \approx \arg\min_\eta J(\mathbf{A}, \mathbf{B} - \eta \mathbf{D}_l)$
      $\mathbf{B}_l \leftarrow \mathbf{B}_l - \hat{\eta}\mathbf{G}_l$
    fin
  fin
fin

2.4 Résultats expérimentaux de la classification des images MNIST

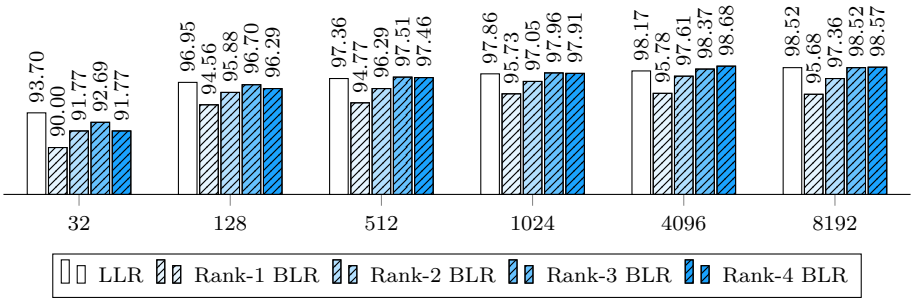
Expérience 1. *Afin de tester l'approche proposée avec BLR, nous considérons la base de données MNIST qui consiste des images monochromatiques de taille $M \times N$ de chiffres écrits à la main allant de 0 à 9 [21] avec $M = N = 28$ (nous montrons quelques images dans la figure 2.2). Ainsi, LLR requiert $MN = 784$ poids dans $\mathbf{w}$ alors que BLR requiert $L(M + N) = 56L$ poids afin de représenter $\mathbf{A}$ et $\mathbf{B}$.*

Nous avons utilisé un ensemble d'apprentissage avec différents nombres d'éléments $T \in \{32, 128, 512, 1024, 4096, 8192\}$. Le taux de reconnaissance est donné en se basant sur un ensemble de test de taille $T_{\text{test}} = 2000$. Nous adoptons la régularisation (2.24) et le paramètre de régularisation α est choisi en utilisant la méthode de validation croisée sur un ensemble de validation de taille $T_{\text{val}} = 2000$.

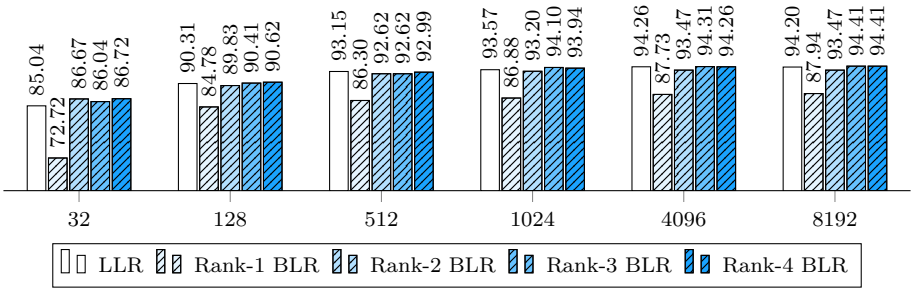
Les résultats de classification binaire des chiffres 8 *et* 9 *sont donnés dans la figure 2.3a) et ceux des chiffres* 5 *et* 8 *sont donnés dans la figure 2.3b).*

Nous constatons que i) pour $L = 1$*, LLR présente des meilleurs résultats que BLR; ceci est du au fait que les deux fonctions de régularisation sont équivalentes pour* $L = 1$ *et le nombre de paramètres est beaucoup plus petit dans BLR, et ii) cette différence dans les résultats est presque compensée en utilisant* $L = 2$*; ainsi, avec seulement* 112 *paramètres requis dans* $\mathbf{A}$ *et* $\mathbf{B}$ *nous obtenons presque la même performance que la régression logistique conventionnelle qui requiert approximativement sept fois plus de paramètres. Ce qui nous indique que le fait d'ignorer la structure de l'image mène au sur-paramétrage de la solution.*

Figure 2.2 – Vingt exemples des images de MNIST choisis aléatoirement de l'ensemble d'apprentissage.



(a) Classification binaire des chiffres 8 et 9



(b) Classification binaire des chiffres 5 et 8

Figure 2.3 – Taux de reconnaissance (%) obtenu sur la base de données MNIST en utilisant différentes valeurs du paramètre T **pour deux problèmes de classification différents.**

Expérience 2. *Nous avons appliqué le modèle BSR sur la même base de données MNIST que dans Expérience 1 mais en utilisant les* $K = 10$ *classes. Les ensembles de validation et de test contiennent chacuns* 10000 *images.*

Le taux de reconnaissance est montré dans la figure 2.4 et nos conclusions sont en concordance avec celles dans Expérience 1. La différence majeure c'est que nous devons augmenter le rang de l'opération bilinéaire jusqu'à $L = 3$ *afin d'aboutir à des résultats comparables avec ceux obtenus avec la régression softmax linéaire (LSR). Par conséquent, au lieu de* $784K$ *paramètres requit dans LSR, BSR n'a besoin que de* $224K$ *paramètres.*

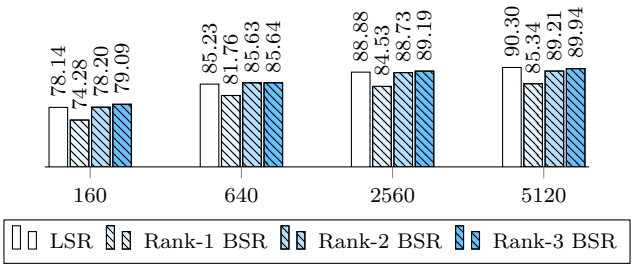


Figure 2.4 – Classification multiple: Taux de reconnaissance obtenu sur la base de données MNIST en utilisant différentes valeurs du paramètre T**.**

Chapitre 3

Réseaux de neurones basés sur les formes multilinéaires

> Si vous changez la façon dont vous regardez les choses, les choses que vous regardez changeront.
>
> Wayne Dyer

Les modèles discutés dans le chapitre précédent, LLR et LSR, restent des modèles linéaires (les frontières de décision sont données par $\mathbf{w}^{\mathrm{T}}\mathbf{x} = \text{constante}$) et ne permettent de résoudre que certains types de problèmes. Ainsi, nous avons besoin d'un modèle qui nous permet d'obtenir une meilleure approximation de y_t pour des problèmes dont les données ne sont pas linéairement séparables. Parmi les solutions proposées dans la littérature, il y a un modèle – très connu et prometteur – qui est le réseau de neurones artificiel.

3.1 Réseau de neurones MLP conventionnel

Pour un problème de classification à K classes, le perceptron multicouche (connu en anglais sous le nom: Multilayer perceptron (MLP)) approxime la probabilité à posteriori de chaque classes $C \in \{1, 2, \ldots, K\}$, c.-à-d, $y_k = \Pr(C = k|\mathbf{x})$ avec $k = 1, \ldots, K$, en utilisant une série de transformations non linéaires appliquées sur l'entrée $\mathbf{x}$ [5, Ch. 5.1]. Chaque fois que nous appliquons une fonction

non linéaire, nous ajoutons une « couche » au réseau. Il peut donc être représenté graphiquement comme dans la figure 3.1 où chaque cercle représente une variable et chaque flèche connectant deux cercles indique l'opération appliquée sur la variable liée à l'origine de la flèche. La profondeur du réseau, c.-à-d, le nombre de couches excepté la couche d'entrée, est donnée par le paramètre D.

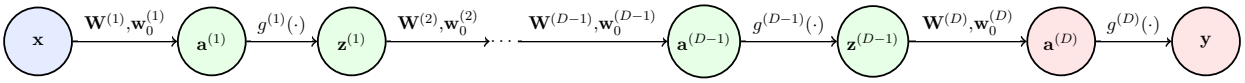


Figure 3.1 – Graphe définissant les relations entre les variables dans MLP ayant D couches.

En supposant que toutes les variables sont des vecteurs ($\mathbf{x} \in \mathbb{R}^{N^{(0)}}$, $\mathbf{a}^{(d)} \in \mathbb{R}^{N^{(d)}}$, $\mathbf{z}^{(d)} \in \mathbb{R}^{N^{(d)}}$ et $\mathbf{y} \in \mathbb{R}^{N^{(D)}}$ avec $N^{(D)} = K$), la série d'opérations définissant le réseau est la suivante:

$$\mathbf{y} = g^{(D)}\Big(\mathbf{a}^{(D)}\Big), \tag{3.1}$$

$$\mathbf{a}^{(d)} = \mathbf{W}^{(d)}\mathbf{z}^{(d-1)} + \mathbf{w}_0^{(d)}, \quad \text{pour} \quad d = 1, \ldots, D, \tag{3.2}$$

$$\mathbf{z}^{(d)} = g^{(d)}(\mathbf{a}^{(d)}), \quad \text{pour} \quad d = 1, \ldots, D-1, \tag{3.3}$$

$$\mathbf{z}^{(0)} = \mathbf{x}. \tag{3.4}$$

Les vecteurs $\mathbf{a}^{(d)}$, appelés *activations*, et $\mathbf{z}^{(d)}$, appelés les *unités cachées*, permettent de définir la couche d du réseau. Les lignes des matrices de poids $\mathbf{W}^{(d)} = \begin{bmatrix} \mathbf{w}_1^{(d)} & \mathbf{w}_2^{(d)} & \ldots & \mathbf{w}_{N^{(d)}}^{(d)} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{N^{(d)} \times N^{(d-1)}}$ agissent comme des filtres linéaires appliqués sur les sorties de la couche précédente, c.-à-d, $a_n^{(d)} = \langle \mathbf{w}_n^{(d)}, \mathbf{z}^{(d-1)} \rangle + \mathbf{w}_0^{(d)}$ avec $n = 1, \ldots, N^{(d)}$. Les vecteurs $\mathbf{w}_0^{(d)} \in \mathbb{R}^{N^{(d)}}$ contiennent des termes constants appelées des biais. Les fonctions $g^{(d)}(\cdot)$, pour $d = 1, \ldots, D-1$, sont appelées des *fonctions d'activation* et elles sont appliquées sur chaque élément du vecteur $\mathbf{a}^{(d)}$. Les fonctions d'activation les plus populaires sont la fonction logistique donnée par (2.3) et la fonction ReLU définie comme:

$$\mathrm{ReLU}(a) = \max(0, a). \tag{3.5}$$

Étant donné que la dérivée de cette fonction n'est pas définie en zéro, il existe une version dérivable de ReLU, appelé softplus donnée par:

$$\mathrm{SP}(a) = \ln\left[1 + \exp\left(a\right)\right]. \tag{3.6}$$

Pour la couche de sortie, nous utilisons typiquement la fonction softmax comme dernière transformation appliquée à $\mathbf{a}^{(D)}$ de sorte que:

$$y_k = g^{(D)}\Big(a_k^{(D)}\Big) = \frac{\exp\Big(a_k^{(D)}\Big)}{\sum_{j=1}^{K}\exp\Big(a_j^{(D)}\Big)}, \tag{3.7}$$

ceci est important afin de produire un vecteur de sortie $\mathbf{y}$ dont les éléments se somment à 1 et qui peuvent être interprétés comme des probabilités à posteriori.

Malgré que la motivation derrière la définition et l'utilisation des fonctions non linéaires $g^{(d)}(\cdot)$ est la ressemblance avec notre système nerveux [16, Sec.11.3] [24], elle ne nous aide pas d'avoir une interprétation des résultats obtenus. Cependant, du point de vue algébrique et à partir de la série d'opérations (3.1)-(3.4), nous pouvons interpréter $z_n^{(d)}$ comme des fonctions de base dans l'espace de $\mathbf{z}^{(d-1)}$ étant donné que $z_n^{(d)} = g^{(d)}\Big(\langle \mathbf{w}_n^{(d)}, \mathbf{z}^{(d-1)}\rangle\Big)$. Ainsi, tout élément du vecteur d'activation $a_m^{(d+1)}$ est obtenu via une expansion de bases linéaire dans l'espace de $\mathbf{z}^{(d-1)}$ [13, Sec. 6.2.1],[16, Ch. 5.1]:

$$\begin{aligned} a_m^{(d+1)} &= \langle \mathbf{w}_m^{(d+1)}, \mathbf{z}^{(d)}\rangle, && (3.8)\\ &= \sum_{n=1}^{N^{(d)}} w_{m,n}^{(d+1)}\; g^{(d)}\Big(a_n^{(d)}\Big), && (3.9)\\ &= \sum_{n=1}^{N^{(d)}} w_{m,n}^{(d+1)}\; g^{(d)}\Big(\langle \mathbf{w}_n^{(d)}, \mathbf{z}^{(d-1)}\rangle\Big). && (3.10) \end{aligned}$$

Nous notons ici, qu'en principe, la base des fonctions satisfait deux conditions : i) elle doit contenir des fonctions linéairement indépendantes, et ii) elle doit permettre d'exprimer n'importe quelle fonction de l'espace par une combinaison linéaires de ses éléments. En général, ces deux conditions ne sont pas satisfaites. Par conséquent, l'utilisation du terme « base » est un abus de langage que nous perpétuons sciemment considérant son ancrage solide dans la littérature à ce sujet [13, Sec. 6.2.1] [16, Sec. 5.1].

Alors, dans ce contexte, nous nous posons la question suivante: pourquoi associons-nous une seule fonction de base à chaque activation $a_n^{(d)}$? Par exemple, dans le cas de fonctions d'activation logistique, pourquoi pas utiliser aussi son complémentaire $\bar{\sigma}(a) = 1-\sigma(a) = \sigma(-a)$? Certains travaux de recherche se sont intéressés à répondre à cette question en utilisant des paires de fonctions complémentaires comme [15] avec son modèle Multivariate adaptive regression splines (MARS). Dans ce

qui suit, nous proposons une autre alternative basée sur les formes multilinéaires. Quelques notions élémentaires relatives aux tenseurs nécessaires pour la compréhension de la suite du chapitre se trouvent dans l'annexe A.

3.2 Fonctions d'activation basées sur les formes multilinéaires

Supposons qu'en effet nous considérons les complémentaires des fonctions logistiques ou dans le cas général, supposons que nous transformons chaque $z_n^{(d)}$ dans (3.3) en un vecteur $\tilde{\mathbf{z}}_n^{(d)} \in \mathbb{R}^I$ tel que:

$$\tilde{\mathbf{z}}_n^{(d)} = \begin{bmatrix} \tilde{z}_{n,i}^{(d)} \\ \vdots \\ \tilde{z}_{n,I}^{(d)} \end{bmatrix} = \begin{bmatrix} g_1^{(d)}\Big(a_n^{(d)}\Big) \\ \vdots \\ g_I^{(d)}\Big(a_n^{(d)}\Big) \end{bmatrix}, \tag{3.11}$$

où $g_i^{(d)} : \mathbb{R} \to \mathbb{R}$ est la i$^{\text{ème}}$ transformation de l'activation $a_n^{(d)}$, $i = 1, \ldots, I$.

Ensuite, nous regroupons ces différents vecteurs $\tilde{\mathbf{z}}_n^{(d)}$ dans un tenseur $\tilde{\boldsymbol{\mathcal{Z}}}^{(d)} \in \mathbb{R}^{I^{N^{(d)}}}$ de rang-1 comme suit:

$$\tilde{\boldsymbol{\mathcal{Z}}}^{(d)} = \tilde{\mathbf{z}}_1^{(d)} \circ \ldots \circ \tilde{\mathbf{z}}_{N^{(d)}}^{(d)}, \tag{3.12}$$

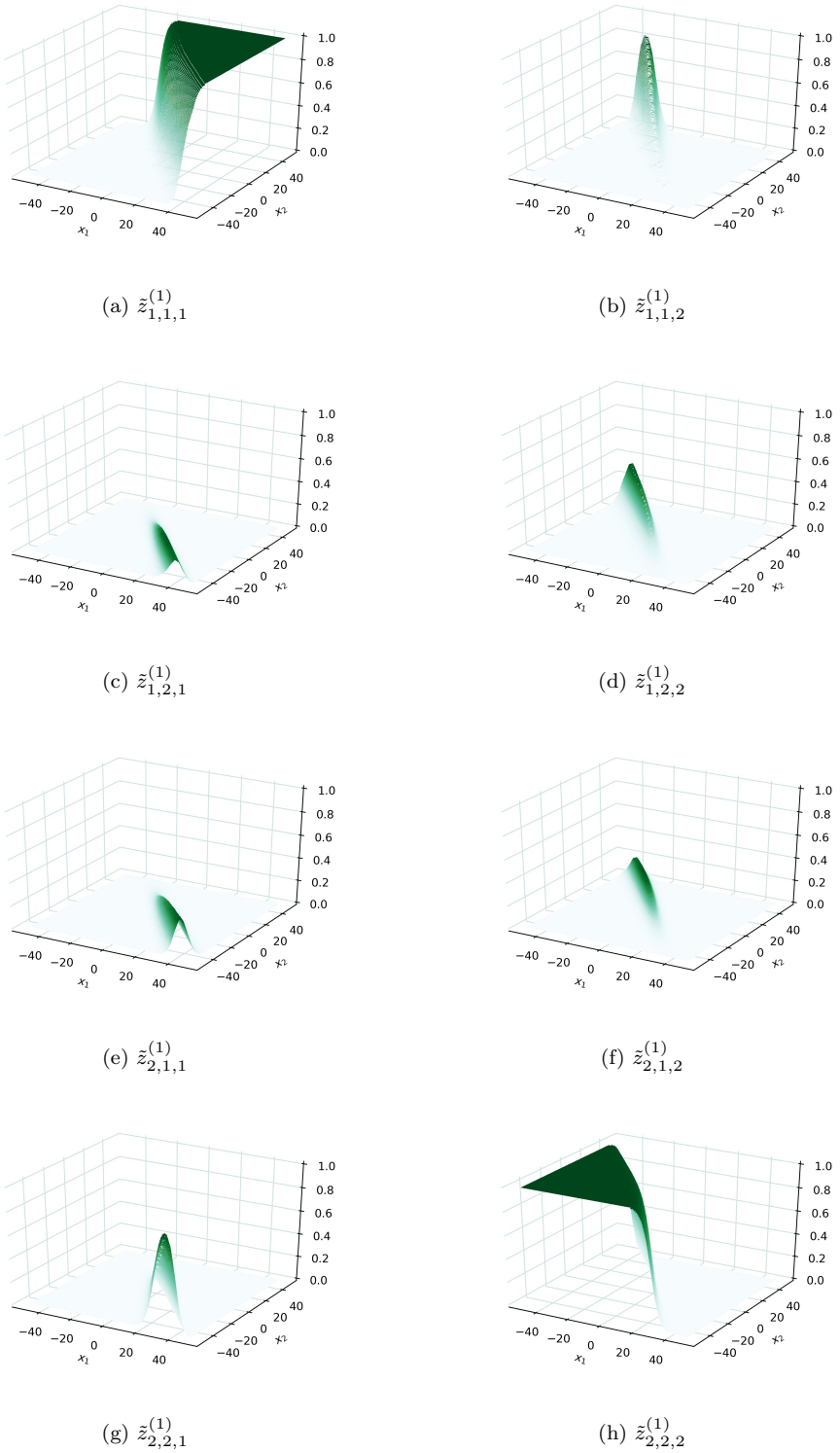
où $\circ$ dénote le produit vectoriel extérieur, c.-à-d, chaque élément de $\tilde{\boldsymbol{\mathcal{Z}}}^{(d)}$ est donné par [20]:

$$\tilde{z}_{i_1,\ldots,i_{N^{(d)}}}^{(d)} = \tilde{z}_{1,i_1}^{(d)}\ \tilde{z}_{2,i_2}^{(d)} \ldots \tilde{z}_{N^{(d)},i_{N^{(d)}}}^{(d)} \quad \text{pour chaque} \quad 1 \leq i_n \leq I. \tag{3.13}$$

En fait, nous pouvons considérer chaque élément de $\tilde{\boldsymbol{\mathcal{Z}}}^{(d)}$ comme une nouvelle fonction de base qui se crée. Pour illustrer, nous supposons que $\mathbf{x} \in \mathbb{R}^2$, $N^{(1)} = 3$ et nous fixons les poids $\mathbf{W}^{(1)} \in \mathbb{R}^{3\times 2}$ permettant de calculer les activations $\mathbf{a}^{(1)}$, c.-à-d, $\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{x}$. Aussi, supposons que $I = 2$ et $\tilde{\mathbf{z}}_n^{(1)}$ se calcule ainsi:

$$\tilde{\mathbf{z}}_n^{(1)} = \begin{bmatrix} \sigma\Big(a_n^{(1)}\Big) \\ \sigma\Big(-a_n^{(1)}\Big) \end{bmatrix} = \begin{bmatrix} \sigma\Big(\mathbf{w}_n^{(1)\mathrm{T}}\mathbf{x}\Big) \\ \sigma\Big(-\mathbf{w}_n^{(1)\mathrm{T}}\mathbf{x}\Big) \end{bmatrix}, \tag{3.14}$$

Comme nous pouvons le voir dans la figure 3.2, avec chaque élément de $\tilde{\boldsymbol{\mathcal{Z}}}^{(1)}$ obtenu via le produit (3.13), nous couvrons de nouvelles régions potentiellement plus utiles pour la classification.



(a) $\tilde{z}^{(1)}_{1,1,1}$ (b) $\tilde{z}^{(1)}_{1,1,2}$

(c) $\tilde{z}^{(1)}_{1,2,1}$ (d) $\tilde{z}^{(1)}_{1,2,2}$

(e) $\tilde{z}^{(1)}_{2,1,1}$ (f) $\tilde{z}^{(1)}_{2,1,2}$

(g) $\tilde{z}^{(1)}_{2,2,1}$ (h) $\tilde{z}^{(1)}_{2,2,2}$

Figure 3.2 – Les éléments de $\tilde{\mathcal{Z}}^{(1)}$ en fonction de x où nous avons fixé les paramètres $\mathbf{W}^{(1)}$ aléatoirement avec $I = 2$ et $N^{(1)} = 3$.

Maintenant, étant donné ce tenseur $\tilde{\boldsymbol{\mathcal{Z}}}^{(d)}$, l'étape suivante consiste à combiner linéairement ses éléments afin de calculer les activations de la couche suivante $\mathbf{a}^{(d+1)}$ tel que:

$$a_m^{(d+1)} = \langle \tilde{\boldsymbol{\mathcal{W}}}_m^{(d+1)}, \tilde{\boldsymbol{\mathcal{Z}}}^{(d)} \rangle, \quad \text{pour} \quad m = 1, \ldots, N^{(d+1)}, \tag{3.15}$$

où $\langle ., . \rangle$ dénote le produit intérieur définie dans (A.8) et $\tilde{\boldsymbol{\mathcal{W}}}_m^{(d+1)} \in \mathbb{R}^{L \times I^{N^{(d)}}}$ est définie comme:

$$\tilde{\boldsymbol{\mathcal{W}}}_m^{(d+1)} = \sum_{l=1}^{L} \tilde{\mathbf{w}}_{m,l,1}^{(d+1)} \circ \ldots \circ \tilde{\mathbf{w}}_{m,l,N^{(d)}}^{(d+1)}. \tag{3.16}$$

Cependant, pour des limitations numériques, nous ne pouvons pas construire explicitement $\tilde{\boldsymbol{\mathcal{Z}}}^{(d)}$ et $\tilde{\boldsymbol{\mathcal{W}}}_m^{(d+1)}$ pour des grandes valeurs de $N^{(d)}$. Néanmoins, nous pouvons toujours calculer (3.15) parce que nous pouvons la réécrire comme:

$$a_m^{(d+1)} = \sum_{l=1}^{L} \prod_{n=1}^{N^{(d)}} \langle \tilde{\mathbf{w}}_{m,l,n}^{(d+1)}, \tilde{\mathbf{z}}_n^{(d)} \rangle = \sum_{l=1}^{L} \prod_{n=1}^{N^{(d)}} \tilde{\mathbf{w}}_{m,l,n}^{(d+1)\mathrm{T}} \, \tilde{\mathbf{z}}_n^{(d)}. \tag{3.17}$$

Démonstration. Nous voulons prouver l'équivalence entre (3.15) et (3.17). Pour simplifier, nous ôtons temporairement les indices de couches et m de $\boldsymbol{\mathcal{Z}}^{(d)}$, $\tilde{\boldsymbol{\mathcal{W}}}_m^{(d+1)}$ et $a_m^{(d+1)}$.

$$a = \langle \tilde{\boldsymbol{\mathcal{W}}}, \tilde{\boldsymbol{\mathcal{Z}}} \rangle \tag{3.18}$$

$$= \sum_{l=1}^{L} \sum_{i_1=1}^{I} \cdots \sum_{i_N=1}^{I} \tilde{z}_{i_1 \ldots i_N} \; \tilde{w}_{l,i_1 \ldots i_N} \tag{3.19}$$

$$= \sum_{l=1}^{L} \sum_{i_1=1}^{I} \cdots \sum_{i_N=1}^{I} \tilde{z}_{1,i_1} \; \cdots \; \tilde{z}_{N,i_N} \; \tilde{w}_{l,1,i_1} \; \cdots \; \tilde{w}_{l,N i_N} \tag{3.20}$$

$$= \sum_{l=1}^{L} \sum_{i_1=1}^{I} \cdots \sum_{i_N=1}^{I} (\tilde{z}_{1,i_1} \tilde{w}_{l,1,i_1}) \ldots (\tilde{z}_{N^{(d)},i_N} \tilde{w}_{l,N,i_N}) \tag{3.21}$$

$$= \sum_{l=1}^{L} \sum_{i_1=1}^{I} (\tilde{z}_{1,i_1} \tilde{w}_{l,1,i_1}) \ldots \sum_{i_N=1}^{I} (\tilde{z}_{N,i_N} \tilde{w}_{l,N,i_N}) \tag{3.22}$$

$$= \sum_{l=1}^{L} \prod_{n=1}^{N} \langle \tilde{\mathbf{w}}_{l,n}, \tilde{\mathbf{z}}_n \rangle. \tag{3.23}$$

□

Une propriété intéressante de la formulation (3.17) du produit intérieur (3.15) est qu'elle nous permet d'explorer le produit des fonctions de bases $g_i^{(d)}(\cdot)$ sans pour autant avoir à créer les tenseurs $\tilde{\boldsymbol{\mathcal{Z}}}^{(d)}$ et $\tilde{\boldsymbol{\mathcal{W}}}_m^{(d+1)}$ qui sont d'ordre très grand.

3.2.1 Définition du réseau TMLP

Pour résumer, la série d'opérations définissant notre réseau ayant D couches, que nous dénotons par TMLP (Tensor-based multilayer perceptron) vue la manipulation des objets tensoriels, est:

$$\mathbf{y} = g^{(D)}\Big(\mathbf{a}^{(D)}\Big), \tag{3.24}$$

$$a_m^{(d)} = \langle \tilde{\boldsymbol{\mathcal{W}}}_m^{(d)}, \tilde{\boldsymbol{\mathcal{Z}}}^{(d-1)} \rangle, \quad \text{pour} \quad m = 1, \ldots, N^{(d)} \text{ et } d = 2, \ldots, D, \tag{3.25}$$

$$\tilde{\boldsymbol{\mathcal{Z}}}^{(d)} = \tilde{g}^{(d)}(\mathbf{a}^{(d)}), \quad \text{pour} \quad d = 1, \ldots, D-1, \tag{3.26}$$

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{z}^{(0)}, \tag{3.27}$$

$$\mathbf{z}^{(0)} = \mathbf{x}, \tag{3.28}$$

avec $\tilde{g}^{(d)}(\cdot)$ la fonction qui nous permet de créer le tenseur $\tilde{\boldsymbol{\mathcal{Z}}}^{(d)}$ définie comme:

$$\begin{aligned} \tilde{g}^{(d)} \colon \mathbb{R}^{N^{(d)}} &\to \mathbb{R}^{I^{N^{(d)}}} \\ \mathbf{a}^{(d)} &\mapsto \begin{bmatrix} g_1^{(d)}\Big(a_1^{(d)}\Big) \\ \vdots \\ g_I^{(d)}\Big(a_1^{(d)}\Big) \end{bmatrix} \circ \ldots \circ \begin{bmatrix} g_1^{(d)}\Big(a_{N^{(d)}}^{(d)}\Big) \\ \vdots \\ g_I^{(d)}\Big(a_{N^{(d)}}^{(d)}\Big) \end{bmatrix}. \end{aligned} \tag{3.29}$$

3.2.2 Comparaison entre TMLP et MLP

Le réseau TMLP peut être représenté graphiquement comme dans la figure 3.3 dont chaque tenseur de poids $\boldsymbol{\mathcal{W}}^{(d)} \in \mathbb{R}^{N^{(d)} \times L \times I^{N^{(d-1)}}}$ est défini comme:

$$\boldsymbol{\mathcal{W}}_{m,:,\ldots,:}^{(d)} \equiv \tilde{\boldsymbol{\mathcal{W}}}_m^{(d)} \quad \text{pour} \quad m = 1, \ldots, N^{(d)}. \tag{3.30}$$

En fait, son architecture est très similaire à celle de MLP illustrée dans la figure 3.1. La couche de sortie est donnée encore par la fonction d'activation $g^{(D)}(\cdot)$ dont la définition correspond à (3.7).

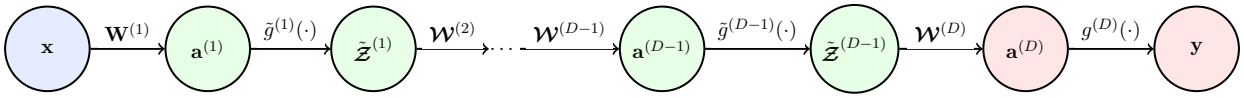


Figure 3.3 – Graphe définissant les relations entre les variables dans le réseau TMLP ayant D couches.

Il diffère d'un réseau MLP conventionnel par ces points:

1. Les fonctions d'activation des couches cachées sont devenues maintenant multidimensionnelles données par $\tilde{g}^{(d)}(\cdot)$. Comme indiqué précédemment, cette fonction $\tilde{g}^{(d)}(\cdot)$ nous permet d'extraire plus d'information à partir des activations et ainsi d'accéder à des régions de l'espace qui étaient en sorte ignorées ou tout simplement inaccessibles auparavant. Nous pouvons voir cette opération comme une augmentation de dimensionnalité; avec chaque application de fonction $\tilde{g}^{(d)}(\cdot)$ nous passons d'un vecteur de $N^{(d)}$ éléments à un tenseur d'ordre $N^{(d)}$.
2. Les unités cachées sont devenues maintenant sous forme tensorielle donnée par $\tilde{\boldsymbol{\mathcal{Z}}}^{(d)}$. Ensuite, dans la couche suivante nous allons procéder à une réduction de dimentionnalité réalisée par le produit intérieur (3.15) avec les poids $\tilde{\boldsymbol{\mathcal{W}}}^{(d)}$. De plus, concrètement, nous calculons ce produit en suivant (3.17) ce qui nous permet de créer et manipuler virtuellement les objets $\tilde{\boldsymbol{\mathcal{Z}}}^{(d)}$ et $\tilde{\boldsymbol{\mathcal{W}}}^{(d)}$.

Ainsi, TMLP offre une généralisation du réseau MLP. Plus précisément, avec un choix spécifique des paramètres $\boldsymbol{\mathcal{W}}^{(d)}$ et cette définition de la fonction $\tilde{g}^{(d)}(\cdot)$:

$$\begin{aligned}\tilde{g}^{(d)}\colon \mathbb{R}^{N^{(d)}} &\to \mathbb{R}^{I^{N^{(d)}}} \\ \mathbf{a}^{(d)} &\mapsto \begin{bmatrix} g^{(d)}\left(a_1^{(d)}\right) \\ 1 \\ \vdots \\ 1 \end{bmatrix} \circ \begin{bmatrix} 1 \\ g^{(d)}\left(a_2^{(d)}\right) \\ \vdots \\ 1 \end{bmatrix} \circ \ldots \circ \begin{bmatrix} 1 \\ \vdots \\ 1 \\ g^{(d)}\left(a_{N^{(d)}}^{(d)}\right) \end{bmatrix},\end{aligned} \tag{3.31}$$

nous pouvons reconstruire MLP.

3.2.3 Apprentissage du modèle TMLP

Étant donné un ensemble d'apprentissage $\{(\mathbf{x}_t, \mathbf{c}_t)\}_{t=1}^{T}$ où $\mathbf{c}_t \in \mathbb{R}^K$ est le vecteur qui encode la classe tel que $c_{t,k} = 1$ if $C_t = k$, notre objective est de minimiser la fonction $J(\cdot)$ afin de trouver les

paramètres du modèle. Elle est définie comme:

$$J(\mathbf{W}^{(1)}, \boldsymbol{\mathcal{W}}^{(2)}, \ldots, \boldsymbol{\mathcal{W}}^{(D)}) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{k=1}^{K} c_{t,k} \ln y_{t,k} + \alpha R(\mathbf{W}^{(1)}, \boldsymbol{\mathcal{W}}^{(2)}, \ldots, \boldsymbol{\mathcal{W}}^{(D)}); \tag{3.32}$$

la fonction de régularisation $R(\cdot)$ peut suivre une définition similaire à (2.20) du modèle BLR, c.-à-d,

$$\begin{aligned} R(\mathbf{W}^{(1)}, \boldsymbol{\mathcal{W}}^{(2)}, \ldots, \boldsymbol{\mathcal{W}}^{(D)}) = \frac{1}{2}(\left\|\mathbf{w}_1^{(1)}\right\|^2 + \ldots + \left\|\mathbf{w}_{N^{(1)}}^{(1)}\right\|^2) \\ + \frac{1}{2}\sum_{d=2}^{D}\sum_{m=1}^{N^{(d)}}\sum_{l=1}^{L}(\left\|\tilde{\mathbf{w}}_{m,l,1}^{(d)}\right\|^2 + \ldots + \left\|\tilde{\mathbf{w}}_{m,l,N^{(d-1)}}^{(d)}\right\|^2). \end{aligned} \tag{3.33}$$

Vu que dans les réseaux de neurones nous avons une succession d'opérations non linéaires, le calcul du gradient de $J(\cdot)$ par rapport aux paramètres n'est pas si facile, surtout pour un réseau très profond. Une technique très efficace, connue sous le nom de rétro-propagation de l'erreur, nous permet de calculer le gradient de la fonction objective $J(\cdot)$ en se basant sur la règle de dérivation en chaîne [5, Sec.5.3]. Puisqu'ici nous manipulons des objets tensoriels, la méthode de rétro-propagation de l'erreur appliquée à TMLP n'est pas aussi évidente. D'où la nécessité de la détailler.

Sans perte de généralité, nous supposons que $D = 2$ afin d'illustrer l'application de la méthode de rétro-propagation de l'erreur pour le calcul du gradient de $J(\cdot)$ dans Algorithme 3.

3.3 Résultats expérimentaux

Nous générons des données synthétiques $\{(\mathbf{x}_t, \mathbf{c}_t)\}_{t=1}^{T}$ où $\mathbf{c}_t$ a un encodage 1 parmi K avec $K = 3$ et $\mathbf{x}_t \in \mathbb{R}^2$ est tiré à partir d'une mixture de trois distributions gaussiennes de sorte que:

$$\Pr(\mathbf{x}_t|C_1) = \pi_1\mathcal{N}\left(\begin{bmatrix}-10\\-3\end{bmatrix}, \Sigma_1\right) + \pi_2\mathcal{N}\left(\begin{bmatrix}5\\3\end{bmatrix}, \Sigma_2\right) + \pi_3\mathcal{N}\left(\begin{bmatrix}5\\-10\end{bmatrix}, \Sigma_3\right), \tag{3.38}$$

$$\Pr(\mathbf{x}_t|C_2) = \pi_1\mathcal{N}\left(\begin{bmatrix}0\\7\end{bmatrix}, \Sigma_1\right) + \pi_2\mathcal{N}\left(\begin{bmatrix}-4\\3\end{bmatrix}, \Sigma_2\right) + \pi_3\mathcal{N}\left(\begin{bmatrix}0\\-9\end{bmatrix}, \Sigma_3\right), \tag{3.39}$$

$$\Pr(\mathbf{x}_t|C_3) = \pi_1\mathcal{N}\left(\begin{bmatrix}-5\\-10\end{bmatrix}, \Sigma_1\right) + \pi_2\mathcal{N}\left(\begin{bmatrix}2\\-3\end{bmatrix}, \Sigma_2\right) + \pi_3\mathcal{N}\left(\begin{bmatrix}-1\\-1\end{bmatrix}, \Sigma_3\right), \tag{3.40}$$

Algorithme 3 : Rétro-propagation de l'erreur pour TMLP

Pour chaque vecteur d'entrée $\mathbf{x}_t$:

1. Propager $\mathbf{x}$ dans le réseau et calculer les vecteurs $\mathbf{z}^{(0)}$, $\mathbf{a}^{(1)}$, $\tilde{\mathbf{z}}_n^{(1)}$, $\mathbf{a}^{(2)}$ et $\mathbf{y}$ en suivant (3.24)-(3.28).
2. Calculer l'erreur au niveau des unités de la couche de sortie:

$$\delta_m^{(2)} \equiv \frac{\partial J}{\partial a_m^{(2)}} \tag{3.34}$$

3. Calculer l'erreur au niveau des unités de la couche cachée:

$$\delta_n^{(1)} \equiv \frac{\partial J}{\partial a_n^{(1)}} = \sum_{m=1}^{N^{(2)}} \frac{\partial J}{\partial a_m^{(2)}} \frac{\partial a_m^{(2)}}{\partial a_n^{(1)}} = \sum_{m=1}^{N^{(2)}} \delta_m^{(2)} \sum_{l=1}^{L} \left[\prod_{\substack{j=1 \\ j \neq n}}^{N^{(1)}} \tilde{\mathbf{w}}_{m,l,j}^{(2)\mathrm{T}} \, \tilde{\mathbf{z}}_j^{(1)} \right] \tilde{\mathbf{w}}_{m,l,n}^{(2)\mathrm{T}} \begin{bmatrix} \frac{\partial g_1^{(1)}}{\partial a_n^{(1)}} \\ \vdots \\ \frac{\partial g_I^{(1)}}{\partial a_n^{(1)}} \end{bmatrix} \tag{3.35}$$

4. Calculer les dérivées partielles:

$$\nabla_{\tilde{\mathbf{w}}_{m,l,n}^{(2)}} J = \frac{\partial J}{\partial a_m^{(2)}} \nabla_{\tilde{\mathbf{w}}_{m,l,n}^{(2)}} a_m^{(2)} = \delta_m^{(2)} \, \tilde{\mathbf{z}}_n^{(1)} \left[\prod_{\substack{j=1 \\ j \neq n}}^{N^{(1)}} \tilde{\mathbf{w}}_{m,l,j}^{(2)\mathrm{T}} \, \tilde{\mathbf{z}}_j^{(1)} \right] \tag{3.36}$$

$$\nabla_{\mathbf{w}_n^{(1)}} J = \nabla_{\mathbf{w}_n^{(1)}} a_n^{(1)} = \delta_n^{(1)} \mathbf{z}^{(0)} \tag{3.37}$$

avec $\pi_1 = 0.3$, $\pi_1 = 0.5$, $\pi_1 = 0.2$, $\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ et $\Sigma_3 = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$.

Les expériences ci-dessous ont été établies sur une base contenant 1000 échantillons de chaque classe. Étant donné que nous connaissons la distribution à priori des données, nous pouvons trouver les régions de décisions en utilisant le théorème de Bayes comme illustré dans la figure 3.4.

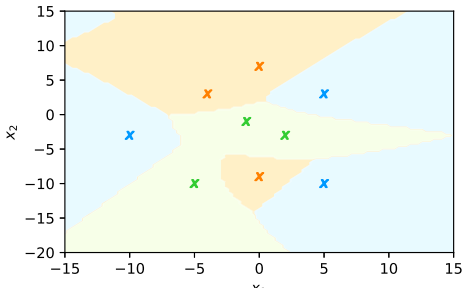
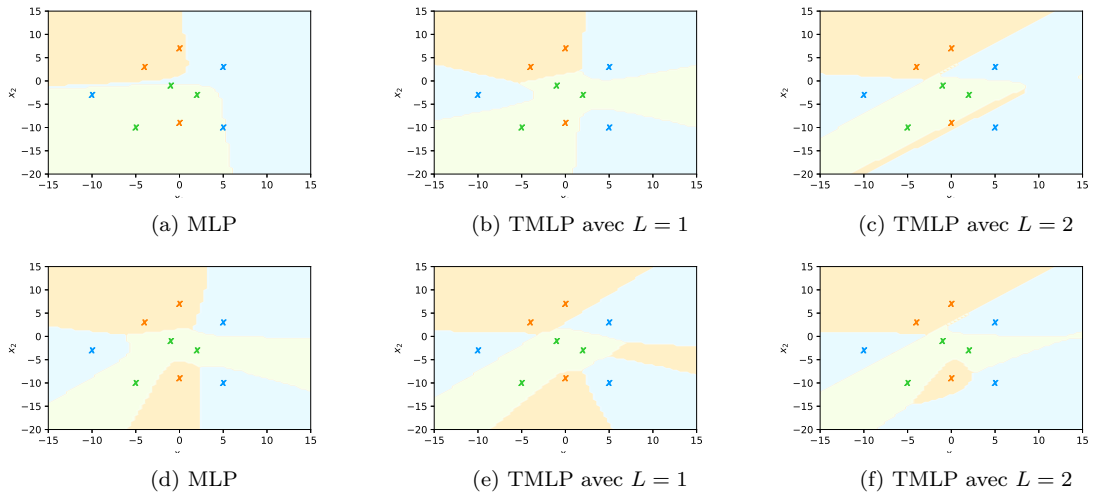


Figure 3.4 – Régions de décisions bayésiennes. Les croix correspondent aux moyennes des distributions gaussiennes à partir desquelles les données ont été tirées.

Nous testons le modèle TMLP sur cet exemple synthétique où la fonction $\tilde{g}^{(1)}$ suit:

$$\begin{aligned}\tilde{g}^{(d)}&: \mathbb{R}^{N^{(d)}} \to \mathbb{R}^{I^{N^{(d)}}} \\ &\mathbf{a} \mapsto \begin{bmatrix} \sigma\left(a_1^{(1)}\right) \\ \sigma\left(-a_1^{(1)}\right) \end{bmatrix} \circ \ldots \circ \begin{bmatrix} \sigma\left(a_{N^{(1)}}^{(1)}\right) \\ \sigma\left(-a_{N^{(1)}}^{(1)}\right) \end{bmatrix}.\end{aligned} \tag{3.41}$$

Comme nous pouvons le constater dans la figure 3.5, TMLP nous permet de trouver des régions de décisions plus intéressantes qu'un réseau MLP conventionnel ayant le même nombre de neurones. Ceci revient au fait que nous avons exploré le produit de fonctions de base données par la fonction $\tilde{g}^{(d)}(\cdot)$. Aussi, nous observons un comportement similaire au modèle BLR par rapport au rang des paramètres donné par L. Plus nous augmentons le rang L, plus la performance de TMLP s'améliore. Ceci ce traduit dans les régions de décisions qui s'approchent plus des régions bayésiennes données dans la figure 3.4.



(a) MLP (b) TMLP avec $L = 1$ (c) TMLP avec $L = 2$

(d) MLP (e) TMLP avec $L = 1$ (f) TMLP avec $L = 2$

Figure 3.5 – Régions de décisions obtenues avec MLP (avec des fonctions d'activation logistiques) et TMLP avec $D = 2$. Pour a), b) et c) $N^{(1)} = 3$ et pour d), e) et f) $N^{(1)} = 6$.

Les résultats de TMLP sur cet exemple synthétique sont prometteurs et nous encourage de tester ce modèle sur des données réelles. Cependant, pour des données plus complexes, comme les images, les dimensions du réseaux sont beaucoup plus grandes afin de pouvoir classifier correctement les entrées. Pour des grandes valeurs de $N^{(1)}$, nous avons remarqué un problème avec le gradient qui tend vers zéro; le produit dans (3.35) devient très faible et par conséquent, il n'aura pas une mise à jour du gradient. Alors, dans le chapitre suivant, nous continuons d'explorer l'idée d'enrichissement de fonctions d'activation avec le produit des fonctions de bases mais vu d'un autre angle.

Chapitre 4

Réseaux de neurones en tant qu'une expansion de bases

> Le design n'est pas seulement ce à quoi ça ressemble. Le design est comment ça fonctionne.
>
> Steve Jobs

Dans le chapitre précédent, nous avons exploré une perspective dans les réseaux reposant sur le produit de fonctions de bases. Cependant, afin de construire d'une manière méthodique un réseau, une meilleure analyse de ses opérations s'avère nécessaire. Plusieurs travaux antérieurs ont analysé les opérations des réseaux de neurones dans le contexte du problème d'approximation universelle qui évalue la possibilité d'approximer n'importe quelle fonction $y = f(x)$ en utilisant un réseau de neurones. En particulier, [11] conclut que toute fonction peut être arbitrairement bien approximée avec un réseau de neurones ayant $D = 2$ et des fonctions d'activation de type logistique (2.3). Tandis que [23] conclut qu'un réseau de largeur finie avec des fonctions ReLU (3.5) peut résoudre le même problème.

Dans ce chapitre, nous analysons en plus de détails les opérations définissant un réseau de neurones de point de vue d'expansion de bases.

4.1 Réseau de neurones comme une expansion de bases

Tel que discuté dans la section 3.1, nous pouvons considérer les activations $a_m^{(d+1)}$ comme le résultat d'une expansion de bases linéaire dans l'espace de $\mathbf{z}^{(d-1)}$ pour un réseau MLP conventionnel. Après avec TMLP, nous avons procédé à une expansion de bases via le produit des fonctions $g_i^{(d)}(\cdot)$. Ici, nous proposons d'adopter une manière différente afin de définir l'expansion de bases, celle reposant sur le produit tensoriel de $z_n^{(d)}(\cdot)$ pour $n = 1, \ldots, N^{(d)}$ connue sous le nom de Tensor-product-basis (TPB) [27].

Plus précisément et sans perte de généralité, nous supposons que les fonctions d'activation $g^{(d)}(\cdot)$ prennent la forme (2.3), c.-à-d, $g^{(d)}\Big(\mathbf{a}^{(d)}\Big) = \sigma\Big(\mathbf{a}^{(d)}\Big)$. Ensuite, au lieu de prendre directement les fonctions de bases $\sigma\Big(a_n^{(d)}\Big)$ dans la somme (3.9), nous procédons en deux étapes:

1. Nous créons de nouvelles fonctions de bases en utilisant le produit des anciennes, c.-à-d,

$$\phi_l\Big(\mathbf{a}^{(d)}\Big) = \prod_{n=1}^{N^{(d)}} \Big[\sigma\Big(a_n^{(d)}\Big)\Big]^{t_{l,n}^{(d+1)}}, \quad \text{pour} \quad l = 1, \ldots, L, \tag{4.1}$$

   où $t_{l,n}^{(d+1)} \in \{0, 1\}$ nous permet de déterminer quel terme $\sigma\Big(a_n^{(d)}\Big)$ doit entrer dans (4.1) $\Big($dans ce cas, nous posons $t_{l,n}^{(d+1)} = 1\Big)$ ou doit être éliminé $\Big(\ t_{l,n}^{(d+1)} = 0\ \Big)$.

2. Nous combinons linéairement les fonctions (4.1):

$$a_m^{(d+2)} = \sum_{l=1}^{L} w_{m,l}^{(d+2)}\ \phi_l\Big(\mathbf{a}^{(d)}\Big) + w_{0,m}^{(d+1)}, \quad \text{pour} \quad m = 1, \ldots, N^{(d+2)}. \tag{4.2}$$

Nous remarquons que: l'ensemble de fonctions de bases dans (4.1) a une cardinalité $L = 2^{N^{(d)}}$ (Le nombre de toutes les combinaisons possibles de $N^{(d)}$ variables binaires $t_{l,n}^{(d+1)}$) qui peut être très grand. Alors, pour des fins de simplicité, nous limitons le nombre de termes dans la somme (4.2); nous définissons cette valeur a priori, c.-à-d, $L = N^{(d+1)} \ll 2^{N^{(d)}}$. De plus, étant donné que nous travaillons avec des valeurs discrètes $t_{l,n}^{(d+1)} \in \{0, 1\}$, il n'est pas pratique de point de vue numérique d'optimiser les valeurs $t_{l,n}^{(d+1)}$ [1, 6]. Alors, nous relaxons cette formulation et nous permettons aux coefficients de prendre des valeurs positives, c.-à-d, $t_{l,n}^{(d+1)} \in \mathbb{R}_+$. Cette contrainte de positivité est généralement facile à implémenter et peut être considérée comme une utilisation des puissances positives de la fonction logistique ce ne doit pas changer dramatiquement la capacité

d'approximation des fonctions. Nous évitons les puissances négatives vu qu'elles correspondent à la division des fonctions logistiques qui peut mener à des résultats non bornés.

Malgré l'indexation différente selon les couches dans (4.2), elle généralise (3.9) dans le sens qu'il est possible de trouver $w_{m,l}^{(d+2)}$ pour lesquels (4.2) est équivalente à (3.9).

4.1.1 TPB comme un réseau de neurones

Une propriété intéressante des deux étapes ci-dessus est qu'elles peuvent être implémentées en utilisant deux fonctions d'activation connues. Supposons que $D = 3$ et $L = N^{(2)}$, alors:

1. Nous transformons (4.1) dans le domaine logarithmique:

$$c_l^{(2)} = \ln \phi_l(\mathbf{a}^{(1)}) = \sum_{n=1}^{N^{(1)}} t_{l,n}^{(2)} \ \ln \sigma\Big(a_n^{(1)}\Big), \quad \text{pour} \quad l = 1, \ldots, N^{(2)}. \tag{4.3}$$

En utilisant la fonction softminus, une version de (3.6), définie comme suit:

$$\rho(a) = -\mathrm{SP}(-a) = \ln \sigma(a), \tag{4.4}$$

nous réécrivons (4.3) comme:

$$c_l^{(2)} = \sum_{n=1}^{N^{(1)}} t_{l,n}^{(2)} \ \rho\Big(a_n^{(1)}\Big), \tag{4.5}$$

$$= \langle \mathbf{t}_l^{(2)}, \mathbf{z}^{(1)} \rangle, \tag{4.6}$$

avec $\mathbf{z}^{(1)} = \rho\Big(\mathbf{a}^{(1)}\Big)$. Maintenant, en définissant $\mathbf{T}^{(2)} = \begin{bmatrix} \mathbf{t}_1^{(2)} & \mathbf{t}_2^{(2)} & \ldots & \mathbf{t}_{N^{(2)}}^{(2)} \end{bmatrix}^{\mathrm{T}}$, nous obtenons:

$$\mathbf{c}^{(2)} = \mathbf{T}^{(2)} \mathbf{z}^{(1)}. \tag{4.7}$$

Malgré que $c_l^{(2)}$ est le logarithme des fonctions $\phi_l(\mathbf{a}^{(1)})$, il est aussi la sortie d'une combinaison linéaire définie dans (4.6); la seule différence avec un réseau de neurones défini dans (3.1)-(3.4) est que si nous voulons ajouter un biais dans chaque couche alors nous le fixons à zéro dans (4.6), c.-à-d, $c_l^{(2)} = \langle \mathbf{t}_l^{(2)}, \mathbf{z}^{(1)} \rangle + t_{0,l}^{(2)}$ avec $t_{0,l}^{(2)} = 0$. Ainsi, nous pouvons traiter $\mathbf{c}^{(2)}$ comme un signal d'activation de la deuxième couche du réseau que nous construisons avec des fonctions d'activation softminus dans la première couche.

2. Nous implémentons (4.2), c.-à-d, nous combinons les fonctions TPB représentées par $c_l^{(2)}$:

$$a_m^{(3)} = \sum_{l=1}^{N^{(2)}} w_{m,l}^{(3)}\ \phi_l\Big(\mathbf{a}^{(1)}\Big) + w_{0,m}^{(3)}, \tag{4.8}$$

$$= \sum_{l=1}^{N^{(2)}} w_{m,l}^{(3)}\ \exp\Big(c_l^{(2)}\Big) + w_{0,m}^{(3)}, \tag{4.9}$$

$$= \langle \mathbf{w}_m^{(3)}, \mathbf{v}^{(2)} \rangle + w_{0,m}^{(3)}, \tag{4.10}$$

où $\mathbf{v}^{(2)} = \xi\Big(\mathbf{c}^{(2)}\Big)$ est défini en utilisant une fonction d'activation exponentielle:

$$\xi(a) = \exp(a). \tag{4.11}$$

Nous aboutissons à une expansion via TPB implémentée avec un réseau de neurones que nous appelons BET-NN (Basis Expansion via Tensor product NN) et que nous représentons graphiquement dans la figure 4.1. La fonction d'activation de la couche de sortie $g^{(3)}(\cdot)$ suit la définition (3.7).

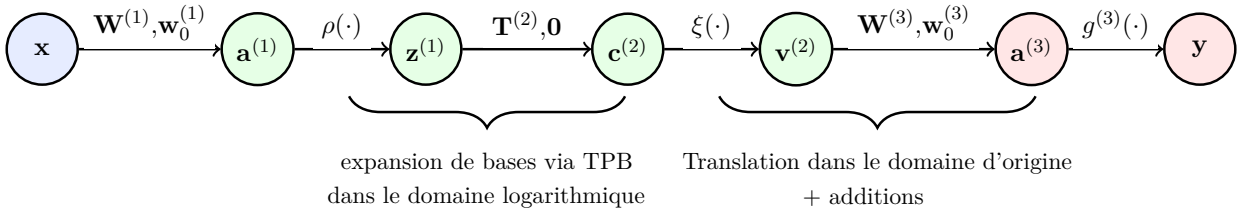


Figure 4.1 – Graphe définissant les relations entre les variables dans BET-NN ayant trois couches.

En général, étant donné que nous avons $\mathbf{a}^{(3)}$, nous pouvons continuer l'extension, c.-à-d, nous définissons $z_m^{(3)} = \rho\Big(a_m^{(3)}\Big)$ comme de nouvelles fonctions de bases et nous les utilisons afin de faire une expansion via TPB en ajoutant deux nouvelles couches au réseau comme suit:

$$\mathbf{z}^{(3)} = \rho\Big(\mathbf{a}^{(3)}\Big), \tag{4.12}$$

$$\mathbf{c}^{(4)} = \mathbf{T}^{(4)}\mathbf{z}^{(3)}, \tag{4.13}$$

$$\mathbf{v}^{(4)} = \xi\Big(\mathbf{c}^{(4)}\Big), \tag{4.14}$$

$$\mathbf{a}^{(5)} = \mathbf{W}^{(5)}\mathbf{z}^{(4)} + \mathbf{w}_0^{(5)}. \tag{4.15}$$

Ainsi, la profondeur du réseau BET-NN augmente par 2 à chaque fois que nous procédons à une expansion de bases via TPB.

4.1.2 Apprentissage du modèle BET-NN

L'apprentissage du réseau BET-NN peut se faire en utilisant la méthode de rétro-propagation de l'erreur [5, Ch. 5.3]. Les contraintes de positivité imposées à $\mathbf{T}^{(2)}$ peuvent être implémentées:

— en mettant à zéro toutes les valeurs qui deviennent négatives après la mise à jour des poids par la méthode du gradient.

— ou bien en utilisant une fonction de pénalité comme:

$$b(t) = \begin{cases} \mu t^2 & \text{if } t < 0 \\ 0 & \text{if } t \geq 0 \end{cases} \tag{4.16}$$

que nous ajoutons à la fonction objective lors de l'optimisation. Le paramètre μ, appelé paramètre de pénalité, définie à quel point nous pénalisons chaque $t^{(2)}$. Cette définition de fonction (4.16) ne nous assure pas que les éléments $t^{(2)}$ restent positifs mais au moins elle assure qu'ils ne tendent pas vers des grandes valeurs négatives.

4.1.3 Expansion de bases dans BET-NN et dans TMLP

Avec le réseau TMLP, nous pouvons dire aussi que nous sommes en train d'implémenter une expansion de bases via TPB grâce aux éléments du tenseur $\tilde{\boldsymbol{\mathcal{Z}}}^{(d)}$; chacun de ces éléments correspond à une multiplication de $N^{(D)}$ fonctions de bases données par $\tilde{z}^{(d)}_{n,i_n}$ (voir (3.13)). D'où l'utilité de comparer TMLP avec BET-NN:

1. En fait, pour $I = 2$ et $\tilde{g}^{(d)}(\cdot)$ définie comme:

$$\begin{aligned} \tilde{g}^{(d)} \colon \mathbb{R}^{N^{(d)}} &\to \mathbb{R}^{I^{N^{(d)}}} \\ \mathbf{a}^{(d)} &\mapsto \begin{bmatrix} \sigma\left(a_1^{(d)}\right) \\ \mathbf{1} \end{bmatrix} \circ \ldots \circ \begin{bmatrix} \sigma\left(a_{N^{(d)}}^{(d)}\right) \\ \mathbf{1} \end{bmatrix}. \end{aligned} \tag{4.17}$$

la définition (3.26) est équivalente à (4.1) si $t^{(d+1)}_{l,n} \in \{0, 1\}$ et $L = 2^{N^{(d)}}$ dans BET-NN. Par conséquent, les deux réseaux TMLP et BET-NN deviennent exactement pareils.

2. Si $I > 2$, nous pouvons toujours créer un réseau BET-NN équivalent à TMLP. Supposons que $\tilde{g}^{(d)}(\cdot)$ suit (3.29). Alors pareillement à (4.1), nous définissons pour BET-NN:

$$\phi_l\Big(\mathbf{a}^{(d)}\Big) = \prod_{n=1}^{N^{(d)}} \left\{ \Big[g_1^{(d)}\Big(a_n^{(d)}\Big)\Big]^{t_{1,l,n}^{(d+1)}} \dots \Big[g_I^{(d)}\Big(a_n^{(d)}\Big)\Big]^{t_{I,l,n}^{(d+1)}} \right\}, \quad \text{pour} \quad l = 1, \dots, L, \tag{4.18}$$

avec $t_{i,l,n}^{(d+1)} \in \{0, 1\}$, $i = 1, \dots, I$. Nous commençons par le passage au domaine logarithmique avec:

$$c_l^{(d+1)} = \ln \phi_l(\mathbf{a}^{(d)}) \tag{4.19}$$

$$= \sum_{n=1}^{N^{(d)}} \left\{ t_{1,l,n}^{(d+1)} \ \ln g_1^{(d)}\Big(a_n^{(d)}\Big) + \ldots + t_{I,l,n}^{(d+1)} \ \ln g_I^{(d)}\Big(a_n^{(d)}\Big) \right\}. \tag{4.20}$$

Ensuite, nous translatons au domaine d'origine et nous sommons les nouvelles fonctions de bases avec:

$$a_m^{(d+2)} = \sum_{l=1}^{L} w_{m,l}^{(3)} \ \phi_l\Big(\mathbf{a}^{(1)}\Big), \tag{4.21}$$

$$= \sum_{l=1}^{L} w_{m,l}^{(3)} \ \exp\Big(c_l^{(2)}\Big). \tag{4.22}$$

Cependant, malgré cette équivalence, TMLP est moins complexe que BET-NN vu que nous somme en train de manipuler implicitement un espace à $N^{(d)}$ dimensions alors qu'il peut être impossible de le faire pour BET-NN dont le nombre de coefficients augmente d'une façon exponentielle avec $N^{(d)}$.

3. Sinon, dans le cas général, BET-NN est plus expressive que TMLP si nous ne restreignons pas $t_{l_i,n}^{(d+1)}$ à être 0 ou 1 dans (4.18).

4.2 Comparaison entre BET-NN et les réseaux de neurones conventionnels

Dans cette section, nous examinons de plus près les opérations de BET-NN comparées à celles de deux réseaux de neurones conventionnels, à savoir MLP et CNN.

4.2.1 Comparaison entre BET-NN et MLP

Précédemment, nous avons montré que les opérations des couches de BET-NN peuvent être interprétées comme des expansions via TPB. Toutefois, BET-NN est très similaire aux autres réseaux de neurones:

1. La fonction softminus (4.4) est déjà utilisée dans les réseaux de neurones conventionnels mais dans la forme « softplus » (3.6). Cette dernière a été introduite afin de pallier le problème de dérivation avec la fonction ReLU. Cependant, la présence de la fonction softminus dans BET-NN est nécessaire afin d'implémenter le produit (4.1).

2. La fonction exponentielle $\xi(c) = \exp(c)$ n'est pas très utilisée dans les réseaux de neurones due aux problèmes numériques que nous pouvons rencontrer. Cependant, avec BET-NN, nous appliquons la fonction exponentielle sur un vecteur $\mathbf{c}^{(2)}$ dont les éléments sont toujours négatifs; ceci vient du fait que $\mathbf{z}^{(1)}$ est négatif et nous avons imposé des contraintes de positivité sur les paramètres $\mathbf{T}^{(2)}$ (voir (4.7)). Par conséquent, la stabilité numérique de la fonction $\xi(a)$ est garantie dans BET-NN (pas de problème de dépassement dans la mémoire). De plus, $\xi(a)$ a une forme similaire à la fonction softplus. En particulier, pour $c < 0$, nous avons $\exp(c) \approx \ln\left(1 + \exp(c)\right)$ et avec $c \to -\infty$,

$$\lim_{c \to -\infty} \frac{\exp(c)}{\ln\left(1 + \exp(c)\right)} = 1. \tag{4.23}$$

3. Dans BET-NN, nous imposons des contraintes de positivité sur $\mathbf{T}^{(2)}$. De telles contraintes ont été utilisées dans la littérature [2, 8, 17] et justifiées par le fonctionnement du système biologique [8] par exemple. Dans notre cas, les contraintes sur $\mathbf{T}^{(2)}$ sont justifiées par leurs interprétations comme étant des puissances des fonctions logistiques dans l'expansion de bases via TPB.

Maintenant, BET-NN diffère d'un réseau de neurones conventionnel par ces points:

1. Nous alternons les fonctions d'activation entre $\rho(a)$ (représentation logarithmique) et $\xi(a)$ (représentation exponentielle afin de revenir au domaine original). De plus, nous assignons une interprétation spécifique aux noeuds du réseau: les sorties de la couche-log ont l'interprétation des fonctions TPB dans le domaine logarithmique et les sortie de la couche-exp correspondent à la sommation des fonctions TPB tel que illustré dans la figure 4.1. Cette interprétation, malgré sa simplicité, nous donne une idée sur le fonctionnement du réseau. En particulier,

un réseau ayant trois couches est maintenant interprétable alors qu'en général toute tentative d'interprétation est abandonnée pour $D > 2$ [13, Sec. 6.2.2].

2. Les coefficients combinant les sorties de la couche-log, ayant softminus comme fonction d'activation, sont positives et aucun biais n'est requis. En fait, un biais dans (4.7) peut être inclus dans le biais de la couche suivante, c.-à-d,

$$a_m^{(3)} = \sum_{l=1}^{N^{(2)}} w_{m,l}^{(3)} \ \exp\left(c_l^{(2)} + t_{0,l}^{(2)}\right) + w_{0,m}^{(3)}, \tag{4.24}$$

$$= \sum_{l=1}^{N^{(2)}} \tilde{w}_{m,l}^{(3)} \ \exp\left(c_l^{(2)}\right) + w_{0,m}^{(3)}, \tag{4.25}$$

avec $\tilde{w}_{m,l}^{(3)} = w_{m,l}^{(3)} \exp(t_{0,l}^{(2)})$.

3. Les contraintes de positivité sont imposées seulement sur les paramètres de la couche-log.

La perspective donnée par l'expansion de bases via TPB nous permet de voir la fonction ReLU, donnée dans (3.5), sous un autre angle. Supposons que les activations avec des fonctions ReLU correspondent encore une fois à une transformation dans le domaine logarithmique des fonctions TPB. En effet, en suivant la même démarche que dans (4.3)-(4.6) et en utilisant les fonctions ReLU, nous calculons:

$$c_l^{(d+1)} = \sum_{n=1}^{N^{(d)}} t_{l,n}^{(d+1)} \ \mathrm{ReLU}\left(a_n^{(d)}\right), \tag{4.26}$$

$$= \ln \phi_l(\mathbf{a}^{(d)}), \tag{4.27}$$

avec cette nouvelle définition de fonctions de base $\phi_l(\mathbf{a}^{(d)})$:

$$\phi_l(\mathbf{a}^{(d)}) = \prod_{n=1}^{N^{(d)}} \left[\exp\left\{\mathrm{ReLU}\left(a_n^{(d)}\right)\right\}\right]^{t_{l,n}^{(d+1)}} \tag{4.28}$$

$$= \prod_{n=1}^{N^{(d)}} \left[\exp\left\{\max\left(0, a_n^{(d)}\right)\right\}\right]^{t_{l,n}^{(d+1)}}, \tag{4.29}$$

$$= \prod_{n=1}^{N^{(d)}} \left[f\left(a_n^{(d)}\right)\right]^{t_{l,n}^{(d+1)}}, \tag{4.30}$$

où la fonction d'activation $f(\cdot)$ est définie comme:

$$f(a) = \begin{cases} 1 & \text{if } a \leq 0 \\ \exp(a) & \text{if } a > 0 \end{cases} \tag{4.31}$$

Ainsi, nous pouvons considérer qu'en général et en suivant la même méthodologie de dérivation des équations (4.3)-(4.6), chaque fonction d'activation dans un réseau de neurones conventionnel correspond à une expansion de bases via TPB dans le domaine logarithmique.

4.2.2 Comparaison entre BET-NN et CNN

L'addition dans le domaine logarithmique peut être vue comme une opération de max-pooling utilisée dans les réseaux de neurones à convolution (CNN). En effet, supposons que nous limitons les poids des fonctions TPB dans (4.8) à être $\tilde{w}_{m,l}^{(2)} = w_{m,l}^{(3)} \in \{0, 1\}$. Ainsi, nous éliminons l'opération d'addition dans la deuxième couche et nous obtenons une nouvelle représentation dans le domaine logarithmique:

$$\tilde{c}_m^{(2)} = \ln \sum_{l=1}^{N^{(2)}} \tilde{w}_{m,l}^{(2)} \ \phi_l(\mathbf{a}^{(1)}), \tag{4.32}$$

$$= \ln \sum_{l=1}^{N^{(2)}} \tilde{w}_{m,l}^{(2)} \ \exp\left(c_l^{(2)}\right), \tag{4.33}$$

$$\approx \max_{l \in \{1,\ldots,N^{(2)}\}} c_l^{(2)}, \quad \text{avec} \quad \tilde{w}_{m,l}^{(2)} = 1. \tag{4.34}$$

C'est ce que nous appelons une opération de max-pooling utilisée dans les réseaux CNN; ici l'opération de maximisation se fait sur les éléments identifiés par $\tilde{w}_{m,l}^{(2)}$. Nous avons laissé le même indice de couche pour $\tilde{c}_m^{(2)}$ afin d'indiquer que l'opération de sommation est simplifié et qu'elle s'exécute encore dans le domaine logarithmique. Nous obtenons ainsi un réseau de neurones tel que illustré dans la figure 4.2.

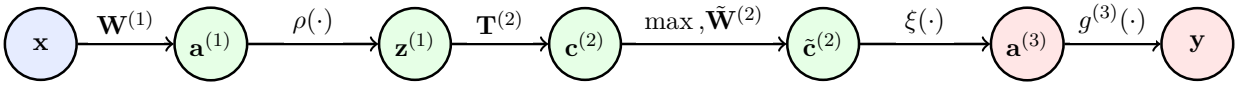


Figure 4.2 – Réseau de neurones BET-NN dont la transformation de $\mathbf{c}^{(2)}$ à $\tilde{\mathbf{c}}^{(2)}$ se fait via une opération de maximisation.

De plus, avec cette nouvelle perspective, nous pouvons simplifier davantage le réseau BET-NN. En effet, puisque $\tilde{w}_{m,l}^{(2)} \in \{0,1\}$, nous pouvons réécrire (4.34) comme:

$$\tilde{c}_m^{(2)} = \max_{l \in \{1,\dots,N^{(2)}\}} \tilde{w}_{m,l}^{(2)}\ c_l^{(2)}, \tag{4.35}$$

$$= \max_{l \in \{1,\dots,N^{(2)}\}} \tilde{w}_{m,l}^{(2)} \sum_{n=1}^{N^{(1)}} t_{l,n}^{(2)}\ z_n^{(1)}. \tag{4.36}$$

En supposant que $t_{l,n}^{(2)} \in \{0,1\}$, nous introduisons une nouvelle variable $\theta_{m,l,n}^{(2)} \in \{0,1\}$ et $\tilde{c}_m^{(2)}$ devient:

$$\tilde{c}_m^{(2)} = \max_{l \in \{1,\dots,N^{(2)}\}} \sum_{n=1}^{N^{(1)}} \theta_{m,l,n}^{(2)}\ z_n^{(1)}, \tag{4.37}$$

$$= \max_{l \in \{1,\dots,N^{(2)}\}} \theta_{m,l}^{(2)\mathrm{T}} \mathbf{z}^{(1)}. \tag{4.38}$$

Par conséquent, nous réduisons le nombre de paramètres à trouver dans le réseau. BET-NN simplifié est illustré dans la figure 4.3.

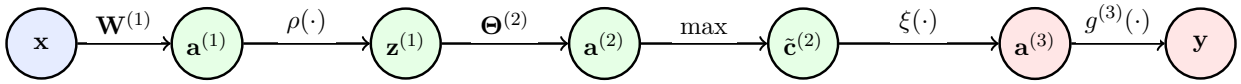


Figure 4.3 – Réseau de neurones BET-NN après l'opération de maximisation et simplification avec les paramètres donnés par $\mathbf{\Theta}^{(2)}$.

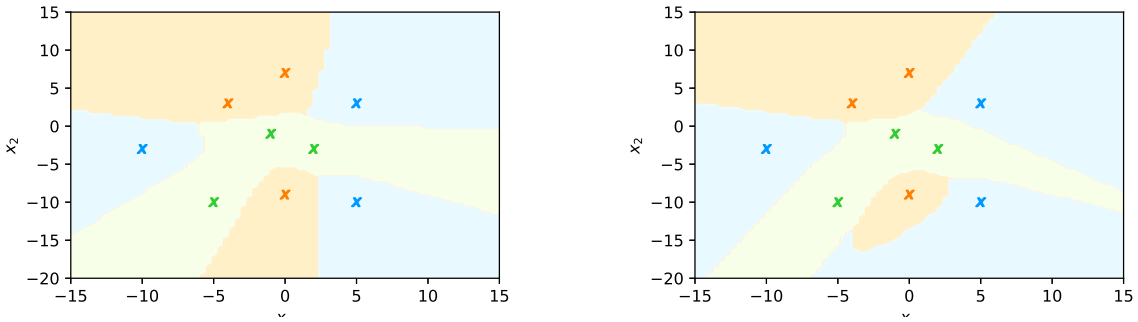
4.3 Résultats expérimentaux

Dans cette section, nous évaluons BET-NN ayant trois couches sur un exemple synthétique et sur des données réelles où nous imposons que $\mathbf{T}^{(2)} \in \mathbb{R}_+^{N^{(2)} \times N^{(1)}}$ en utilisant la fonction de pénalité définie dans (4.16).

4.3.1 Exemple synthétique

Nous testons BET-NN sur le même exemple synthétique que nous avons utilisé dans la section 3.3. Le paramètre de pénalité est fixé à $\mu = 10^{-4}$. Étant donné que le problème d'optimisation n'est pas convexe, nous exécutons 10 essais avec différentes initialisations des poids des différentes couches et

dans la figure 4.4 nous montrons les régions de décisions obtenues avec les modèles produisant les meilleurs résultats en terme du taux de reconnaissance.



(a) MLP avec des fonctions d'activation logistique

(b) BET-NN

Figure 4.4 – Régions de décisions obtenues avec MLP et BET-NN ayant tous les deux trois couches ($N^{(1)} = 6, N^{(2)} = 64$).

4.3.2 Classification d'images

Nous testons notre modèle pour résoudre un problème de classification à 10 classes sur les bases de données suivantes:

— **MNIST**: Nous l'avons utilisé aussi afin de tester les modèles BLR et BSR dans le chapitre 2.

— **Fashion-MNIST**: Cette base de données contient $70,000$ images de vêtements noir et blanc de taille 28×28 [29].

Figure 4.5 – Vingt exemples des images de Fashion-MNIST choisis aléatoirement de l'ensemble d'apprentissage.

Pour toutes les expériences, nous utilisons un ensemble d'apprentissage avec différents nombres d'éléments $T \in \{2560, 5120, 10240, 20480\}$ et un ensemble de test avec $T_{\text{test}} = 10000$. Le facteur de régularisation α est choisi selon la méthode de validation croisée sur un ensemble de validation de taille $T_{\text{val}} = 10000$. MLP-Logistique désigne le réseau MLP ayant la fonction logistique comme fonction d'activation et MLP-ReLU désigne le réseau MLP ayant la fonction ReLU comme fonction d'activation. Le paramètre de pénalité pour BET-NN est fixé à $\mu = 10^{-4}$. La médiane, 25^{e} centile,

75^{e} centile, les valeurs minimum et maximum du taux de reconnaissance sont affichés pour 20 initialisations aléatoires des poids des réseaux dans les figures 4.6 et 4.7 pour Fashion-MNIST et MNIST respectivement. Ces diagrammes nous montrent le taux de reconnaissance obtenu avec des réseaux ayant le même nombre de couches et le même nombre de neurones dans chacune des couches. D'autres modèles appliqués sur ces bases de données se trouvent par exemple dans [7] et [29].

Nous observons que BET-NN améliore la performance du réseau en terme du taux de reconnaissance par rapport à MLP-Logistique et MLP-ReLU, surtout pour un petit ensemble d'apprentissage ($T = 2560$). En suivant l'interprétation que nous avons attribué à BET-NN, l'amélioration peut provenir du fait que nous explorons le produit des fonctions logistiques dans BET-NN alors que dans MLP-Logistique nous considérons leur combinaison linéaire.

Cela dit, suite à la discussion de la Section 4.2.1, nous pouvons considérer que le réseau MLP conventionnel implémente aussi une expansion de bases via TPB dans le domaine logarithmique pour une définition spécifique de la fonction de base $f(\cdot)$ dépendante de la fonction d'activation utilisé (ReLU par exemple). Donc, les différences entre BET-NN et MLP proviennent du choix de la fonction $f(\cdot)$ et des contraintes de positivité que nous imposons dans BET-NN sur les paramètres de la couche-log afin de revenir au domaine d'origine.

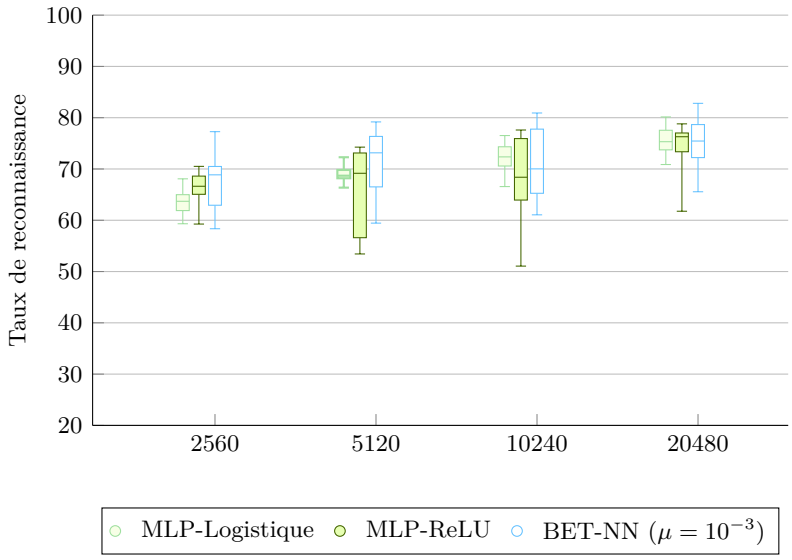
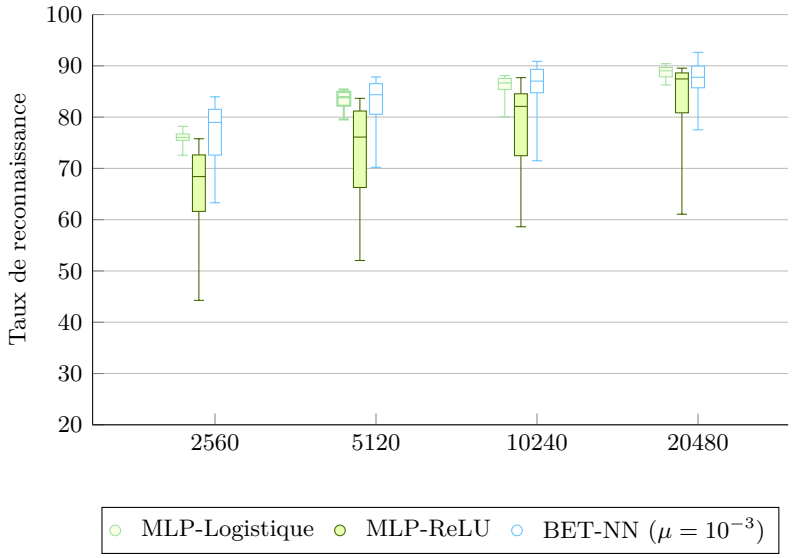
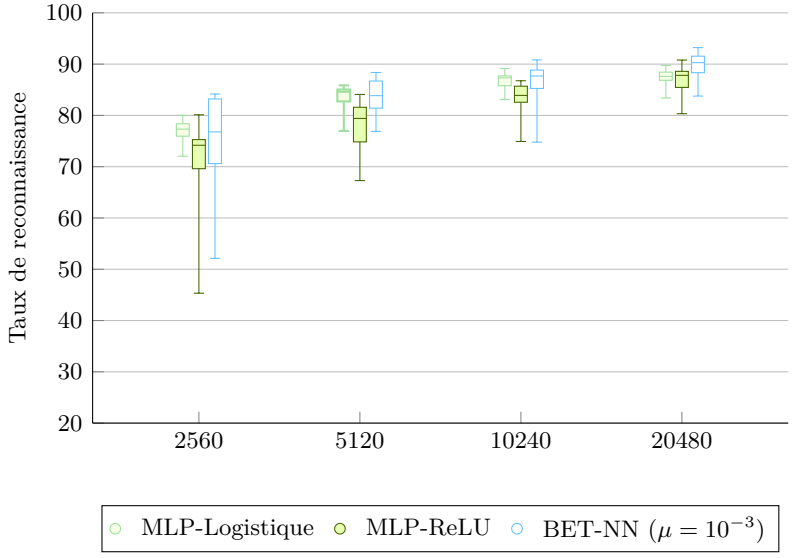


Figure 4.6 – Classification multiple: Taux de reconnaissance obtenu sur la base de données Fashion-MNIST en utilisant différentes valeurs du paramètre T. $N^{(1)} = 20$ et $N^{(2)} = 40$. La médiane, 25^{e} centile, 75^{e} centile, les valeurs minimum et maximum du taux de reconnaissance sont affichés pour 20 initialisations aléatoires des poids des réseaux.



(a) $N^{(1)} = 20, N^{(2)} = 20$



(b) $N^{(1)} = 20, N^{(2)} = 40$

Figure 4.7 – Classification multiple: Taux de reconnaissance obtenu sur la base de données MNIST en utilisant différentes valeurs du paramètre T et différents choix de $N^{(1)}$ et $N^{(2)}$. La médiane, 25^{e} centile, 75^{e} centile, les valeurs minimum et maximum du taux de reconnaissance sont affichés pour 20 initialisations aléatoires des poids des réseaux.

Conclusion générale

Dans ce mémoire, nous avons abordé deux problèmes liés à la représentation et la manipulation des données dans l'apprentissage automatique.

Nous avons commencé par explorer les formes bilinéaires permettant d'opérer sur des objets de type matriciel. Les tests de nos modèles ont donné des performances similaires à celles obtenues en utilisant une régression logistique conventionnelle mais avec un nombre de paramètres nettement plus faible. Ce qui nous indique que les modèles qui ignorent la structure spatiale des données souffrent d'un sur-paramétrage.

Par la suite, nous nous sommes intéressés aux réseaux de neurones; plus particulièrement à leur fonctionnement vu qu'ils nous permettent de résoudre un ensemble plus grand de problèmes (selon le théorème de l'approximation universelle). Nous nous sommes demandés si nous pouvons enrichir l'ensemble de fonctions de bases sans pour autant augmenter la complexité de l'algorithme. À l'issue de cette idée, nous avons proposé le modèle TMLP basé sur les formes multilinéaires des données qui nous permet de manipuler implicitement un espace dimensionnel très grand engendré par le produit des fonctions de base.

Ensuite, afin de palier les problèmes potentiels relatifs à l'optimisation lors de la manipulation des données multi-dimensionnelles, nous avons proposé une seconde approche inspirée de TMLP avec le modèle BET-NN. Ce dernier nous a permis d'obtenir une nouvelle perspective sur les réseaux de neurones. Nous avons pu les interpréter du point de vue d'expansion de bases par un produit tensoriel dans le domaine logarithmique. Nous avons comparé BET-NN avec MLP et CNN du point de vue fonctionnel des réseaux. Enfin, les résultats expérimentaux ont montré que la performance de notre modèle dépasse celle de MLP même avec l'imposition de contraintes de positivité sur les paramètres dans BET-NN.

Au cours de ce travail, nous avons constaté qu'il reste encore des pistes à explorer pour la continuité de notre projet. Pour le modèle BET-NN, nous avons formulé le problème d'optimisation avec une forme particulière de fonction de régularisation. De plus, nous avons formulé les contraintes de positivité en utilisant une fonction de pénalité. Ces choix arbitraires devraient être étudiés davantage afin de pouvoir les définir d'une manière plus formelle. En général, des problèmes persistent encore autant dans les réseaux de neurones conventionnels que dans nos modèles entre autres relativement au choix du nombre de couches cachées ainsi que du nombre de neurones dans chacune.

Annexe A

Notions élémentaires sur les tenseurs

Un tenseur est une généralisation des matrices qui peut représenter des données qui se trouvent dans un espace vectoriel multidimensionnel [20]. Les données, lorsqu'elles sont acquises, possèdent naturellement une structure multidimensionnelle comme les images couleurs.

A.1 Tenseur d'ordre-P

L'ordre d'un tenseur représente son nombre de dimensions. Un tenseur $\boldsymbol{\mathcal{T}}$ est d'ordre-P si $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_p}$ et dont $\boldsymbol{\mathcal{T}}_{i_1, i_2, \ldots, i_p} \in \mathbb{R}$ pour $i_1 \in [d_1], \ldots, i_p \in [d_P]$. Par exemple:

- Un scalaire est un tenseur d'ordre-0.
- Un vecteur est un tenseur d'ordre-1.
- Une matrice est un tenseur d'ordre-2.
- Un tenseur $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ d'ordre-3 possède trois indices et il est illustré dans la figure A.1. Une image couleur est un tenseur d'ordre-3: la dimension d_1 représente les intensités des couleurs rouge, bleu et vert, les dimensions d_2 et d_3 représentent les position des pixels.

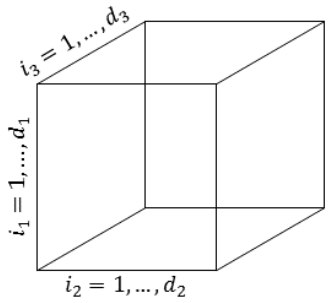


Figure A.1 – Tenseur d'ordre-3: $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

A.2 Transformation du tenseur

A.2.1 Transformation matricielle d'un tenseur

Étant donné un tenseur $\boldsymbol{\mathcal{T}}$, la transformation matricielle consiste à transformer ce tenseur en une large matrice. Pour un tenseur d'ordre-3, nous pouvons avoir trois modes de transformation matricielle:

$$\mathbf{T}_{[1]} = \boldsymbol{\mathcal{T}}_{:,1:d_2,1:d_3} = \begin{bmatrix} \boldsymbol{\mathcal{T}}_{::1} & \ldots & \boldsymbol{\mathcal{T}}_{::d_3} \end{bmatrix}, \mathbf{T}_{[1]} \in \mathbb{R}^{d_1 \times d_2 d_3}, \tag{A.1}$$

$$\mathbf{T}_{[2]} = \boldsymbol{\mathcal{T}}_{1:d_1,:,1:d_3}, \mathbf{T}_{[2]} \in \mathbb{R}^{d_2 \times d_1 d_3}, \tag{A.2}$$

$$\mathbf{T}_{[3]} = \boldsymbol{\mathcal{T}}_{1:d_1,1:d_2,:}, \mathbf{T}_{[3]} \in \mathbb{R}^{d_3 \times d_1 d_2}, \tag{A.3}$$

avec $\boldsymbol{\mathcal{T}}_{::i_3} \in \mathbb{R}^{d_1 \times d_2}$ pour $i_3 = 1, \ldots, d_3$.

A.2.2 Transformation vectorielle d'un tenseur

Cette opération consiste à transformer un tenseur en un long vecteur. Nous commençons par transformer le tenseur en une matrice selon le premier mode. Nous obtenons ainsi une matrice que nous transformons en un vecteur ensuite. Cette opération est notée $\text{vec}(\boldsymbol{\mathcal{T}}) = \text{vec}(\mathbf{T}_{[1]})$ et elle est définie par:

$$\text{vec}(\boldsymbol{\mathcal{T}}) = \begin{bmatrix} \text{vec}(\boldsymbol{\mathcal{T}}_{::1}) \\ \ldots \\ \text{vec}(\boldsymbol{\mathcal{T}}_{::d_3}) \end{bmatrix} \in \mathbb{R}^{d_1 d_2 d_3}. \tag{A.4}$$

A.3 Rang d'un tenseur

Un tenseur $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_P}$ est dit de rang-1 s'il peut s'écrire comme le produit extérieur de P vecteurs, c.-à-d,

$$\boldsymbol{\mathcal{T}} = \mathbf{a}_1 \circ \mathbf{a}_2 \circ \ldots \circ \mathbf{a}_P, \tag{A.5}$$

avec $\circ$ dénote le produit extérieur vectoriel ordinaire. Les éléments du tenseur sont donc donnés par:

$$\boldsymbol{\mathcal{T}}_{i_1 i_2 \ldots i_p} = a_{1 i_1} a_{2 i_2} \ \ldots \ a_{P i_P}, \tag{A.6}$$

pour chaque $i_p = 1, \ldots, d_p$.

En général, le rang d'un tenseur $\boldsymbol{\mathcal{T}}$, noté rank($\boldsymbol{\mathcal{T}}$), est défini comme le nombre minimum de tenseurs de rang-1 qui peuvent former $\boldsymbol{\mathcal{T}}$ comme étant leur somme. Par exemple, un tenseur $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ est dit de rang-R s'il s'écrit comme suit:

$$\boldsymbol{\mathcal{T}} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \tag{A.7}$$

pour $\mathbf{a}_1, \ldots, \mathbf{a}_R \in \mathbb{R}^{d_1}$, $\mathbf{b}_1, \ldots, \mathbf{b}_R \in \mathbb{R}^{d_2}$ et $\mathbf{c}_1, \ldots, \mathbf{c}_R \in \mathbb{R}^{d_3}$.

A.4 Opérations sur les tenseurs

Comme pour les matrices, plusieurs produits peuvent être appliqués sur les tenseurs. Ils existent ceux qui sont similaires à ceux appliquer sur les matrices. Mais, ils existent d'autres qui sont spécifiques aux tenseurs. Dans ce qui suit nous présentons quelques-uns.

A.4.1 Produit intérieur

Un produit intérieur entre deux tenseurs de même ordre $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_P}$, noté $\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle$, est défini comme:

$$\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle = \sum_{i_1=1}^{d_1} \sum_{i_2=1}^{d_2} \ldots \sum_{i_P=1}^{d_P} a_{i_1 i_2 \ \ldots \ i_P} b_{i_1 i_2 \ \ldots \ i_P}. \tag{A.8}$$

A.4.2 Produit en mode-p

Produit (matrice) en mode-p

Supposons que nous avons un tenseur $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_P}$ et une matrice $\mathbf{M} \in \mathbb{R}^{J \times d_p}$. Le produit en mode-$p$ entre $\boldsymbol{\mathcal{T}}$ et $\mathbf{M}$, dénoté $\boldsymbol{\mathcal{T}} \times_p \mathbf{M}$, génère un autre tenseur $\boldsymbol{\mathcal{U}} = \boldsymbol{\mathcal{T}} \times_p \mathbf{M}$ de dimension $d_1 \times \ldots \times d_{p-1} \times J \times d_{p+1} \times \ldots \times d_P$ et dont les éléments sont donnés par:

$$u_{i_1 \ldots i_{p-1} j i_{p+1} \ldots i_P} = \sum_{i_p=1}^{d_p} t_{i_1 i_2 \ldots i_P} m_{j i_p}. \tag{A.9}$$

Nous pouvons combiner plusieurs produit (matrice) en mode-p mais l'ordre n'affecte pas le résultat final. En effet, nous avons:

$$\boldsymbol{\mathcal{T}} \times_p \mathbf{M}_1 \times_q \mathbf{M}_2 = \boldsymbol{\mathcal{T}} \times_q \mathbf{M}_2 \times_p \mathbf{M}_1. \tag{A.10}$$

Produit (vecteur) en mode-p

Le produit en mode-p entre un tenseur $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_P}$ et un vecteur $\mathbf{v} \in \mathbb{R}^{d_p}$, dénoté par $\boldsymbol{\mathcal{T}} \,\bar{\times}_p\, \mathbf{M}$, donne un tenseur $\boldsymbol{\mathcal{U}}$ d'ordre $p-1$ avec $\boldsymbol{\mathcal{U}} = \boldsymbol{\mathcal{T}} \,\bar{\times}_p\, \mathbf{M} \in \mathbb{R}^{d_1 \times \ldots \times d_{p-1} \times d_{p+1} \times \ldots \times d_P}$ où:

$$u_{i_1 \ldots i_{p-1} i_{p+1} \ldots i_P} = \sum_{i_p=1}^{d_p} t_{i_1 i_2 \ldots i_P} v_{i_p}. \tag{A.11}$$

Contrairement au produit (matrice) mode-p, si nous combinons plusieurs produit (vecteur) en mode-p l'ordre des opérations influence le tenseur résultant. À titre d'exemple, nous avons:

$$\boldsymbol{\mathcal{T}} \,\bar{\times}_p\, \mathbf{v}_1 \,\bar{\times}_q\, \mathbf{v}_2 \neq \boldsymbol{\mathcal{T}} \,\bar{\times}_q\, \mathbf{v}_2 \,\bar{\times}_p\, \mathbf{v}_1. \tag{A.12}$$

Références

[1] E. Amaldi, E. Mayoraz, and D. de Werra. A review of combinatorial problems arising in feedforward neural network design. *Discrete Applied Mathematics*, 52(2):111 – 138, 1994.

[2] B. O. Ayinde and J. M. Zurada. Deep learning of constrained autoencoders for enhanced understanding of data. *IEEE Transactions on Neural Networks and Learning Systems*, 29(9):3969–3979, 2018.

[3] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[4] J. Benesty, C. Paleologu, and S. Ciochină. On the identification of bilinear forms with the Wiener filter. *IEEE Signal Processing Letters*, 24(5):653–657, May 2017.

[5] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[6] Avrim L. Blum and Ronald L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117 – 127, 1992.

[7] Adam Byerly, Tatiana Kalganova, and Ian Dear. A branching and merging convolutional network with homogeneous filter capsules, 2020.

[8] J. Chorowski and J. M. Zurada. Learning understandable neural networks with nonnegative weight constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 26(1):62–69, 2015.

[9] Wei Chu and Seung-Taek Park. Personalized recommendation on dynamic content using predictive bilinear models. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, page 691–700, New York, NY, USA, 2009. Association for Computing Machinery.

[10] J.S. Cramer. The origins of logistic regression. *Tinbergen Institute, Tinbergen Institute Discussion Papers*, 01 2002.

[11] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[12] Laura Dogariu, Silviu Ciochina, Jacob Benesty, and Constantin Paleologu. System identification based on tensor decompositions: A trilinear approach. *Symmetry*, 11:556, 04 2019.

[13] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, USA, 2000.

[14] Mads Dyrholm, Christoforos Christoforou, and Lucas C. Parra. Bilinear discriminant component analysis. *J. Mach. Learn. Res.*, 8:1097–1111, 2007.

[15] Jerome H. Friedman. Rejoinder: Multivariate adaptive regression splines. *Ann. Statist.*, 19(1):123–141, 03 1991.

[16] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statiscs, 2009.

[17] E. Hosseini-Asl, J. M. Zurada, and O. Nasraoui. Deep learning of part-based representation of data using sparse autoencoders with nonnegativity constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 27(12):2486–2498, 2016.

[18] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*, 2019.

[19] Hung Hung and Chen-Chien Wang. Matrix variate logistic regression model with application to EEG data. *Biostatistics (Oxford, England)*, 14, 07 2012.

[20] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51:455–500, 2009.

[21] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[22] S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[23] 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*, NIPS'17, page 6232–6240, Red Hook, NY, USA, 2017. Curran Associates Inc.

[24] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.

[25] Hamed Pirsiavash, Deva Ramanan, and Charless C. Fowlkes. Bilinear classifiers for visual recognition. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1482–1490. Curran Associates, Inc., 2009.

[26] Frank F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[27] Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46(1):159 – 216, 1990.

[28] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019.

[29] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 08 2017.